

MAY 2018

M.Sc. in Aircraft and Aerospace Engineering

AHMED EL İBRAHİM

**UNIVERSITY OF GAZİANTEP
GRADUATE SCHOOL OF
NATURAL & APPLIED SCIENCES**

EFFICIENT WING DESIGN FOR SAILPLANES



M. Sc. THESIS

IN

AIRCRAFT AND AEROSPACE ENGINEERING

BY

AHMED EL İBRAHİM

MAY 2018

Efficient Wing Design for Sailplanes

M.Sc. Thesis

In

Aircraft and Aerospace Engineering

University of Gaziantep

Supervisors

İbrahim GÖV

Sohayb ABDUL KARIM

by

AHMED EL İBRAHİM

May 2018

© 2018 [Ahmed EL İBRAHİM]

REPUBLIC OF TURKEY
UNIVERSITY OF GAZİANTEP
GRADUATE SCHOOL OF NATURAL & APPLIED SCIENCES
AIRCRAFT AND AEROSPACE ENGINEERING DEPARTMENT

Name of the thesis: Efficient Wing Design for Sailplanes

Name of the student: Ahmed EL İBRAHİM


Exam date: 08.08.2018

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Ahmet Necmeddin YAZICI

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.


Dr. Öğr. Üyesi M. Orkun ÖĞÜCÜ

Head of Department

This is to certify that we have read this thesis and that in our consensus opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Sohayb Abdul KARIM

Doç. Dr. İbrahim GÖV

Co-Supervisor




Supervisor




Examining Committee Members (**Title and Name-surname**)

Signature

Doç. Dr. İbrahim GÖV

Dr. Öğr. Üyesi Tahir DURHASAN

Dr. Öğr. Üyesi Khaled Al Cheikh HAMOUD

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Ahmed EL İBRAHİM

ABSTRACT

EFFICIENT WING DESIGN FOR SAILPLANES

EL İBRAHİM, Ahmed

M.Sc. in Aircraft and Aerospace Eng.

Supervisors:

Assoc. Prof. Dr. İbrahim GÖV

Asst. Prof. Dr. Sohayb ABDUL KARIM

May 2018

81 pages

The purpose of this research is to obtain optimized sailplane wing design, which contributes to performance in terms of range and endurance. This study presents aerostructural design optimization using Multidisciplinary Design Optimization approach (MDO). This optimization is carried out using OpenAeroStruct (OAS), an open-source low-fidelity aerostructural analysis and optimization tool written in Python and developed in NASA's OpenMDAO framework. The aerostructural model in OAS coupled a Vortex Lattice Method for aerodynamic with a 1-D Finite-Element Analysis for structure, to model the sailplane wing. An aerostructural optimization problem was formulated to minimize the Sink Speed, as the objective function, with varying twelve design variables, and subjected to two constraints. An initial simple wing design is used as baseline for the optimization. The findings of the aerostructural optimization indicated that the optimized wing exhibited improvements in both disciplines, resulting a better performance for the sailplane in terms of range and endurance. The optimized wing achieved increase of 37% in endurance and 27% in range.

Keywords: aero-structural optimization, multidisciplinary design optimization, sailplane wing design, open AeroStruct code, open MDAO code.

ÖZET

PLANÖRLER İÇİN VERİMLİ KANAT TASARIMI

EL İBRAHİM, Ahmed

Yüksek Lisans Tezi, Uçak ve Uzay Müh Bölümü

Tez Yöneticileri:

Doç.Dr. İbrahim GÖV

Dr.Öğr.Üyesi Sohayb ABDUL KARIM

Mayıs 2018

81 sayfa

Bu araştırmanın amacı menzil ve havada kalma süresi açısından performansa katkıda bulunan optimize edilmiş planör kanat tasarımı elde etmektir. Bu çalışma, Multidisipliner Tasarım Optimizasyonu yaklaşımını (MDO) kullanarak aero-yapısal tasarım optimizasyonu sunmaktadır. Bu optimizasyon Python'da yazılmış ve NASA'nın OpenMDAO çerçevesinde geliştirilen açık kaynaklı, düşük performanslı bir aero-yapısal (aerostructural) analiz ve optimizasyon aracı olan OpenAeroStruct (OAS) kullanarak gerçekleştirilir. Planör kanadını modellemek için, OAS'deki aero-yapısal model, aerodinamik için bir Vortex Lattice Metodu'nu, yapı için bir 1-B Sonlu Elemanlar Analizi ile birleştirdi. Alçalma Hızını hedef fonksiyonu olarak en aza indirmek için, değişken on iki tasarım değişkeniyle ve iki kısıtlayıcıya tabi bir aero-yapısal optimizasyon problemi formüle edilmiştir. Optimizasyon için başlangıç olarak basit bir kanat tasarımı kullanılmıştır. Aero-yapısal optimizasyon bulguları, optimize edilmiş kanadın her iki disiplinde de gelişme gösterdiğini işaret etmiş olup bunun da mesafe ve havada kalma süresi açısından planörün daha iyi bir performans sergilediğini gösterdi. Optimize edilmiş kanat, havada kalma süresinde %37, mesafede ise %27 artış göstermiştir.

Anahtar Kelimeler: aero-yapısal optimizasyon, multidisipliner tasarım optimizasyonu, yelkenli kanat tasarımı, open AeroStruct code, open MDAO code.



To All of Them.

ACKNOWLEDGEMENTS

The author wishes to express his deepest gratitude to his supervisor Assoc. Prof. Dr. İbrahim GÖV and Asst. Prof. Dr. Sohayb ABDUL KARIM for their guidance, advice, criticism, encouragements and insight throughout the research.



TABLE OF CONTENTS

| | Page |
|--|------|
| ABSTRACT | v |
| ÖZET | vi |
| ACKNOWLEDGEMENTS | viii |
| CHAPTER 1 | 1 |
| INTRODUCTION | 1 |
| 1.1 Thesis Outline..... | 1 |
| 1.2 Motivation..... | 1 |
| 1.3 Aircraft Design Background..... | 2 |
| 1.3.1 Wing Design..... | 2 |
| 1.3.2 Aircraft Design Phases..... | 2 |
| 1.3.3 Levels of Design Fidelity..... | 4 |
| CHAPTER 2 | 6 |
| LITERATURE REVIEW | 6 |
| 2.1 Introduction..... | 6 |
| 2.2 Evolution of Sailplane Wing Design..... | 6 |
| 2.3 Aero-structural Design Optimization in the Literature..... | 9 |
| CHAPTER 3 | 12 |
| SOARING AND SAILPLANE | 12 |
| 3.1 Introduction..... | 12 |
| 3.2 Soaring Flight..... | 12 |
| 3.1.1 Brief History..... | 12 |
| 3.1.2 Basics of Soaring and Gliding Flight..... | 14 |
| 3.1.3 Phases of Soaring Flight..... | 17 |
| 3.2 Sailplane Flight Performance..... | 19 |
| 3.3 Sailplanes Classification..... | 21 |
| CHAPTER 4 | 22 |
| MULTIDISCIPLINARY DESIGN OPTIMIZATION | 22 |
| 4.1 Introduction..... | 22 |
| 4.2 Multidisciplinary Design Optimization Approach..... | 22 |
| 4.2.1 MDO Definition and Application..... | 22 |
| 4.2.2 MDO Problem Formulation..... | 23 |
| 4.2.3 Optimization Problem Statement..... | 23 |
| 4.3 Optimization Algorithms..... | 24 |
| 4.3.1 Gradient-based Algorithms..... | 24 |
| 4.3.2 Gradient-free Algorithms..... | 25 |
| 4.4 Extensions to the Design Structure Matrix (XDSM)..... | 25 |
| 4.5 MDO Frameworks and Architecture..... | 26 |
| CHAPTER 5 | 30 |
| AEROSTRUCTURAL IDEALIZATION OF THE SAILPLANE WING | 30 |

| | | |
|--|--|-----------|
| 5.1 | Introduction | 30 |
| 5.2 | Aerostructural Optimization Framework | 30 |
| 5.2.1 | Open MDAO Framework | 30 |
| 5.2.2 | Open <i>AeroStruct</i> Framework | 31 |
| 5.3 | Aerostructural Idealization of the Wing | 32 |
| 5.3.1 | Aerodynamic Idealization | 32 |
| 5.3.2 | Structural Idealization | 35 |
| 5.3.3 | Aero-Structural Coupling | 36 |
| 5.3.4 | The Optimizer | 40 |
| 5.4 | Setting up the Aerostructural Optimization Problem | 40 |
| 5.5 | Case Study of a Sailplane Wing | 41 |
| CHAPTER 6 | | 43 |
| THE RESULTS | | 43 |
| 6.1 | Introduction | 43 |
| 6.2 | Aerodynamic Optimization Findings | 43 |
| 6.3 | Structural Optimization Findings | 44 |
| 6.4 | Aero-structural Optimization Findings | 45 |
| CHAPTER 7 | | 48 |
| DISCUSSION | | 48 |
| 7.1 | Aerodynamic Optimization | 48 |
| 7.2 | Structural Optimization | 49 |
| 7.3 | Aero-structural Optimization | 49 |
| CHAPTER 8 | | 51 |
| CONCLUSION | | 51 |
| 8.1 | Thesis Conclusion | 51 |
| 8.2 | Future Work | 51 |
| REFERENCES | | 53 |
| APPENDIX A: Modified Sections of OAS Code | | 58 |
| A1 - The Run-file Script for the Aerostructural Optimization | | 58 |
| A2 - Script of Modified <i>functionals.py</i> | | 60 |
| A3 - Script of Modified <i>run_classes.py</i> | | 67 |

LIST OF TABLES

| | Page |
|--|-------------|
| Table 3.1. Competeion sailplanes classification by FAI. | 21 |
| Table 4.1. List of symbols used in XDSM. | 26 |
| Table 5.1. Initial sailplane wing specifications. | 41 |
| Table 5.2. Aerodynamic and structural models specifications. | 42 |
| Table 6.1. Aerostructural optimization results | 46 |

LIST OF FIGURES

| | Page |
|---|-------------|
| Figure 1.1. Phases of aircraft design, adapted from. | 3 |
| Figure 1.2. Design knowledge and freedom throughout the design phases. | 4 |
| Figure 1.3. Levels of design fidelity. | 5 |
| Figure 2.1. Evolution of sailplane wing design. | 9 |
| Figure 3.1. Early gliding attempts. | 13 |
| Figure 3.2. Forces on a sailplane. | 14 |
| Figure 3.3. Natural atmospheric phenomena to gain lift. | 16 |
| Figure 3.4. Relationship of drag components with airspeed. | 17 |
| Figure 3.5. An example of cross-country soaring using thermals. | 18 |
| Figure 3.6. Typical speed polar of a sailplane. | 20 |
| Figure 4.1. Local versus global minima/maxima. | 24 |
| Figure 4.2. Example of XDSM of gradient-based optimization problem. | 26 |
| Figure 4.3. XDSM diagram of AAO Architecture. | 27 |
| Figure 4.4. XDSM diagram of IDF Architecture. | 27 |
| Figure 4.5. XDSM diagram of MDF architecture with a Gauss–Seidel. | 28 |
| Figure 4.6. XDSM diagram of CSSO Architecture. | 28 |
| Figure 4.7. XDSM diagram of CO Architecture. | 28 |
| Figure 4.8. XDSM diagram of BLISS-2000 Architecture. | 29 |
| Figure 5.1. Single horseshoe vortex on lifting surface. | 33 |
| Figure 5.2. Planar airfoil section. | 33 |
| Figure 5.3. Distributed horseshoe vortices over a wing. | 34 |
| Figure 5.4. 6-DOF spatial beam element. | 35 |
| Figure 5.5. Load transfer from a panel on the aerodynamic mesh. | 36 |
| Figure 5.6. XDSM diagram for default aerostructural optimization. | 39 |
| Figure 5.7. Initial sailplane wing. | 42 |
| Figure 6.1. Aerodynamic design variables during the optimization process. | 44 |
| Figure 6.2. Beam thickness for the five control points along the span. | 45 |

Figure 6.3. The objective function during the optimization process..... 46
Figure 6.4. Aerostructural optimization at four iteration points. 47
Figure 7.1. Bent-up tip of the optimized wing. 49



LIST OF SYMBOLS

| | |
|----------|---------------------------------|
| A | Cross-section area of the beam |
| A | Influence coefficients matrix |
| AR | Aspect ratio |
| CL | Lift coefficient |
| CD | Drag coefficient |
| C_{di} | Incuced drag coefficient |
| D | Drag force |
| E | Young modulus |
| e | Oswald factor |
| F | Aerodynamic resultant force |
| G | Shear modulus |
| I | The torsional moment of inertia |
| J | The polar moment of inertia |
| K | Global stiffness matrix |
| L | Lift force |
| S | Reference wing area |
| u | Displacement vector |
| V | Airspeed |
| V_s | Sink Speed |
| v_i | Induced velocity |
| W | Sailplane weight |
| ρ | Air density |
| γ | Glide angle |

CHAPTER 1

INTRODUCTION

1.1 Thesis Outline

This thesis is organized into eight chapters, **the first chapter** concerns with clarifying the motivation for this research and gives a brief background on the aircraft design. In **the second chapter**, a literature review of similar studies and researches on aerostructural design optimization has been presented and discussed. In addition to a summary of sailplane wing design evolution. **The third chapter** introduces the basics of soaring flight, starting from brief history and summaries the basics of soaring flight and its performance. After that, **the fourth chapter** presents the approach of multidisciplinary design optimization and discusses the MDO problem formulation and the different types of architectures used in the application of MDO. **The fifth chapter** discusses the components of the aerostructural idealization of the sailplane wing and highlights the theoretical background of the aerodynamic and structural methods used in this thesis. In addition, this chapter introduces the aerostructural framework OAS. At the end of this chapter, a case study for the sailplane wing is presented. In **the sixth chapter**, all the results of the aerostructural optimization are shown. Then, **the seventh chapter** discusses the findings of the aerostructural optimization and summaries the results. Finally, the **eighth chapter** concludes the research of this thesis and gives few directives for the future work.

1.2 Motivation

Sailplanes are symbolic of aerodynamic efficiency. Many factors contribute to this efficiency, yet the wing is the key factor here. That is why the wing design is the focus of attention for designers.

Wings with high aspect ratio usually distinguish sailplanes in order to achieve high aerodynamic efficiency. At the same time, high aspect ratio is challenging from structural point of view, which usually requires thicker and shorter wings. Therefore, such these inter-disciplinary considerations, call for special design and trade-off procedures to integrate the aerodynamic efficiency with robust structure in single design. For this

reason, the integration between these two disciplines; structure and aerodynamic should be considered and examined in the early preliminary design stage of the wing. An early integration analysis and optimization leads to smoother transition to the detailed design stage and, as a result, less time and resources.

The methodology of this work is to apply MDO approach, aero-structural optimization, along with low-fidelity analysis methods to a preliminary design of sailplane wing. This work aims to obtain an “efficient” wing design through achieving the optimum design of a “relatively good” initial wing design. For the optimized wing to be an “efficient”, it should evidently contribute to better performance characteristics and design features than the initial design at the same conditions. Another important objective of this thesis is to provide an aerostructural design framework for sailplane wing through adapting an existing framework for powered aircrafts wings.

1.3 Aircraft Design Background

1.3.1 Wing Design

The wing design plays the key role in the sailplane performance and it has a variety of design configurations that lead to a good design. Therefore, making a design for a certain configuration is not an easy assignment. Especially with all the interactions among the disciplines that control the wing design. For example, selecting a certain planform could have a positive effect on both aerodynamic and structural properties, yet it could be hard to manufacture. However, the wing design, as the aircraft, passes through three main phases sequentially. These phases are conceptual design, preliminary design and detailed design. Each process has differences in activities, tools, expertise, cost, time ... etc.

1.3.2 Aircraft Design Phases

1.3.2.1 Conceptual Design

During this phase, we need to expand our knowledge of the new aircraft, so we identify the mission requirements and a set of specifications for this aircraft. Determination of the shape of aircraft, weight, size and performance is carried out in this phase with a certain design freedom. Regarding the wing, the fundamental configurations are determined in this phase, such as planform, sweep, location ...etc. the product of this phase is usually a layout of the aircraft configurations.

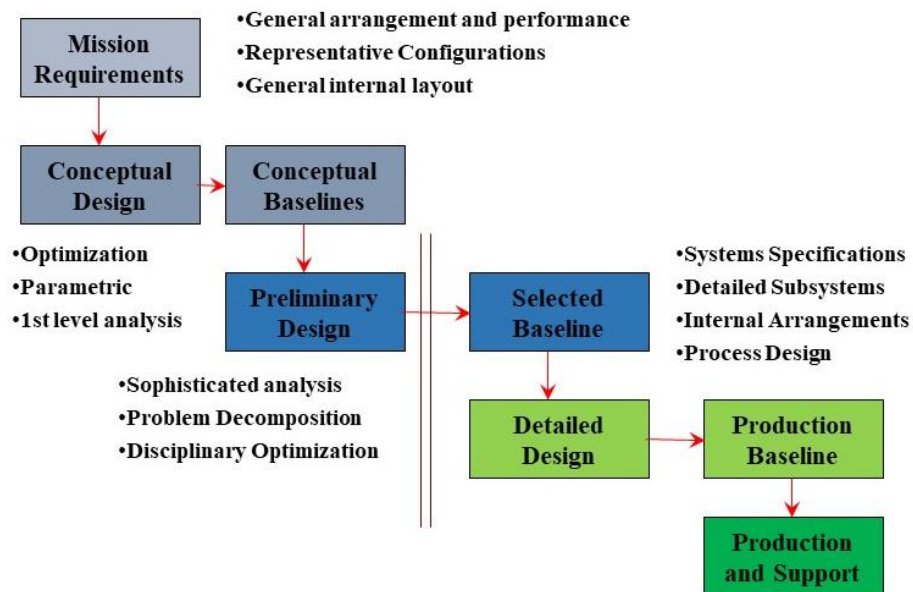


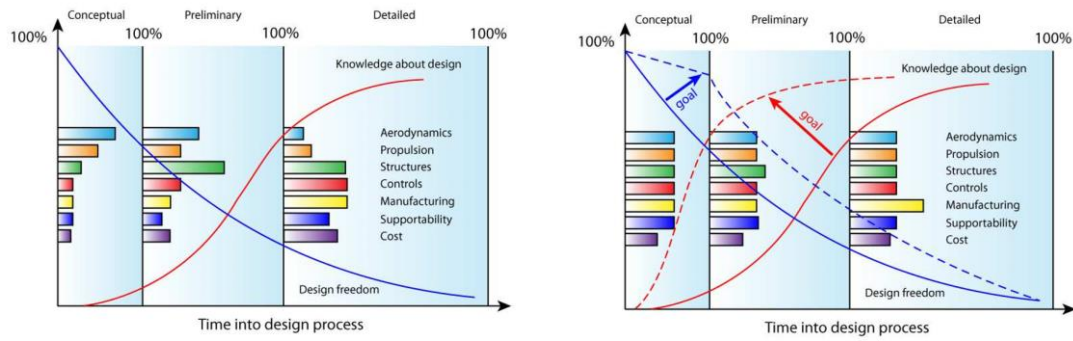
Figure 1.1. Phases of aircraft design, adapted from [1].

1.3.2.2 Preliminary Design

In this phase, our knowledge of the design is better and, at the same time, our flexibility to change the design is reducing. The preliminary phase uses the baseline design configurations resulted from the conceptual design phase. Then, more development and analysis could be used during this phase until sufficient quality of the design achieved. Therefore, computational analysis and optimization tools in along with wind tunnel testing will be carried out. At the end of this phase, the configurations will be frozen and precisely defined

1.3.2.3 Detailed Design

As the name of this phase indicates, the “details” are main concern here. The design now has been analyzed and finalized in all disciplines at the preliminary design phase. In this phase, the detailed design of each individual part (panels, spars, ribs, rivets...etc) can take place. At the end of this phase, the aircraft is all set to be fabricated.



a. Traditional aircraft design process b. Preferred aircraft design process

Figure 1.2. Design knowledge and freedom throughout the design phases [1], [2].

1.3.3 Levels of Design Fidelity

Design fidelity refers to the level of details and functionality built into a model or a problem. In aerostructural design optimization, two levels of fidelity are mainly used in the literature; low and high fidelity. However, a third level could be countered also, called mid-fidelity level. The multi-fidelity model basically a function of the CFD method and the geometry detail of the model, Fig. 1.3.

The low-fidelity analysis is usually used at the early design phases (conceptual and preliminary) and used simpler CFD solvers, such as Vortex Lattice Method (VLM). Where, it determines the best general characteristics for the design, regarding the requirements defined by the engineer [3]. At this level, our design still flexible and we still have a good amount of freedom, so certain changes and modifications are applicable given the low computational cost of this level of fidelity. At the end of design analysis in low fidelity level, a comprehensive idea of the design characteristics can be obtained. In wing design space, this level of analysis is often used for determine lift to drag ratio and structural weight.

On the other hand, high-fidelity analysis is used to obtain more detailed improvements that can refine design characteristics of the model. This level of analysis uses rigorous analysis methods, such as RANS solver. Thus, the computational resources required are considerably high, could last days [4]. In wing design space, this level of analysis could used to refine the twist over the span.

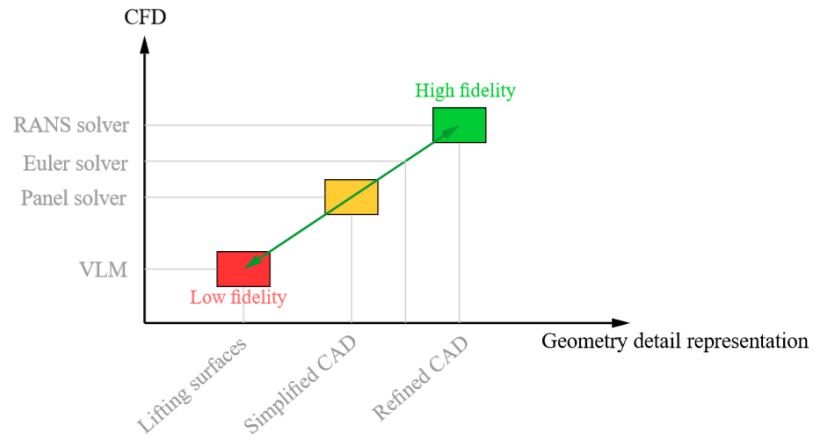


Figure 1.3. Levels of design fidelity [4].



CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The first section of this chapter explores the development of sailplane wing design since the early beginnings of soaring and highlights the key turning points in wing design. In addition, the second section explores the past and current work and researches in aerostructural optimization for sailplane wing.

2.2 Evolution of Sailplane Wing Design

The evolution of sailplane wing design accompanied with the passion to soaring more whether in time or distance. Many important stations were behind the development in sailplane wing design, such; discovering natural resources to gain lifts, developing laminar flow airfoils, advances in material science and computational and optimization technologies. This section provides a quick review on evolution of sailplane wig design.

One of the first and important station of the evolution of sailplane wing design is with Otto Lilienthal's works in the late 19th century. Lilienthal experimented various models of hang gliders with spans ranging from 6 to 9.5 m, areas of 8 to 13 m², and aspect ratios of approximately 4.0 [5]. Later, in early 1900s, Wright brothers built a glider (Fig. 2.1-b) based on small-scale wind-tunnel tests and trial-and-error. They used rectangular planform with span of 9.8 m and an aspect ratio of 6.8. The lift to drag ratio (glide ratio) has been estimated to be about five.

After World War I (WWI), and a results of Treaty of Versailles that put restrictions on powered flight, German engineers and designers focused on sailplanes and improving their performance. The Vampyr (Fig. 2.1-c) is a sample sailplane of that era, it was designed to meet the requirements of a ridge-soaring, including low sinking speed, good gliding angle, sufficient strength, and good manoeuvrability [6]. The Vampyr used a wing with span of 12.4-m and a maximum lift-to-drag ratio of 16. It exhibited a superior performance with more than 1-hour endurance at that time. Another good example of

sailplane design in that period is the Fafnir (Fig. 2.1-d). Alexander Lippisch designed it in 1930. Fafnir had 19-m span wing, which was completely cantilevered to save drag, requiring that the root airfoil has a very high thickness ratio. The wing planform tapered to very narrow tips in addition to gentle gull dihedral distribution. Günther Groenhoff managed to fly with Fafnir a distance of 220 km, made Fafnir the first sailplane to fly a cross-country distance more than 200 km [6]. The evolution of sailplane design and performance continued after the World War II (WWII), especially by holding annual sailplane competitions in *Rhön Mountains*. Designers started to research deeper on effects of wing geometry and laminar flow airfoils on the performance. The introduction of the Eppler airfoil sections to be found in this period in line with the advancements in fiberglass construction, which, in turn, made it possible to construct robust lightweight structures with sufficiently smooth lifting surfaces. The *Phoenix* sailplane (Fig. 2.1-e) was the first to take advantage of these advancements, where it was able to achieve extensive laminar flow through using fiberglass reinforced plastic construction, which insured a surface quality. In 1957, *Phoenix* was able to achieve a glide ratio of 40:1 [6]. Furthermore, in the 1950's sailplane classes were introduced to handle the wide variety of designs used in competitions. The classes were described earlier in Section 2.3.

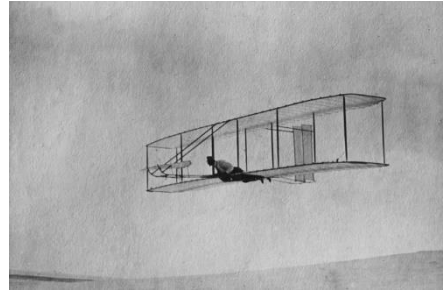
By the early 1970s, the usage of composite materials became more frequent in that period. The benefits of higher wing loading had been proved, so sailplanes with less wing areas were designed in order to increase empty weights. The SB-10 (Fig. 2.1-f) is a good example. It took advantage of the advancement in materials and managed to have 29-m wingspan and to be the first sailplane to achieve a glide ratio over 50 [6]. The advances in composite materials had a huge influence on sailplane design and allowing larger wingspans and aspect ratios. The increasing in span and performance had continued, as in Eta (Fig. 2.1-g), the two-seat, self-launching sailplane, which was capable of achieving maximum glide ratio of 50 with 31-m span. It is worth mentioning that the eta ranked as the highest performance sailplane to date.

The evolution in sailplane wing design is still the passion of designers and researchers in seek of higher performance. The existing of modern technologies in material science and computational fluid dynamics opens various options to make progress. In addition, employing multidisciplinary design optimization in sailplane design has great influence on exploring the trade-off among different disciplines and obtaining the optimum design.

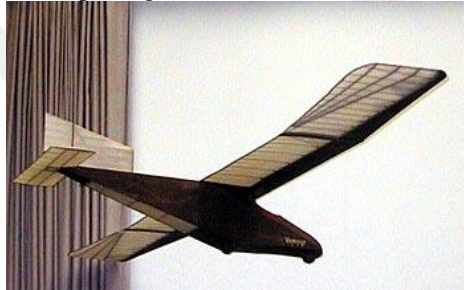
One of the great indications that our ambition has no end is the *Airbus Perlan 2* project [7]. Perlan 2 (Fig. 2.1-h) is a pressurized sailplane designed to fly at the edge of space at altitude of 90,000 ft (about 27.5 km) to explore the science of giant mountain waves. At this altitude, the air density is less than 2% of what it is at sea level and the temperatures is minus 70 degrees. The first flight of this sailplane was on September 23, 2015.



a. Monoplane glider of otto Lilienthal (1894).[8]



b. The Wright brother's glider (1902). [9]



c. The Vampyr (1921). [10]



d. The Fafnir (1930). [11]



e. FS-24 Phoenix (1957). [12]



f. SB-10 sailplane (1972). [13]



g. Eta sailplane (2001). [14]



h. Perlan II sailplane (2015). [Photo by James Darcy, Airbus Americas]

Figure 2.1. Evolution of sailplane wing design.

2.3 Aero-structural Design Optimization in the Literature

The emergences of MDO started with researches on structural optimization during the period from 1960 until 1980s, by Schmit [15], [16], [17], [18] and Haftka et al. [19], [20], [21], [22]. Then the application of MDO had extended to other disciplines instead of focusing only on structure. The wing design was the first application of MDO as the wing design combines three main disciplines; aerodynamics, structures, and controls. First steps in MDO in an aeroelastic wing design is found to be in the 70s.

In 1977, Haftka [23] used sequential optimization to solve the coupled aerostructural system of fighter wing. The aerostructural model coupled a lifting line aerodynamic model with a simple structural Finite Element Analysis (FEA). The same model for combined aerodynamic and structural optimization was also used by McGeer [24], who investigated wing design characteristics that offer minimum drag. Afterwards, Grossman et al. [25], used aerostructural model that coupled a lifting-line theory with beam analysis. They applied two different optimization approaches to design a sailplane wing, sequential and simultaneously integrated optimization. As a result, they found that the optimized wing resulted from integrated optimization approach was superior in terms of performance or weight to other obtained using sequential approach. Two years later, in 1990, Grossman et al. [26] used another aerostructural model to optimize a subsonic transport wing with more focusing on alleviating the computational cost of the optimization. Similarly, Wakayama and Kroo [27], conducted wing planform optimization for minimum drag with constraints on structural weight and maximum lift. They showed how much the optimum is dependent by compressibility drag, aeroelasticity and structural design conditions. In middle 1990s, utilization of high-fidelity methods started with works, like Borland et al. [28], Chattopadhyay and Pagaldipti [29] and Baker and

Giesing [30]. However, limitations on the computational resources have restricted the number of design variables. In continuation, Manning [31] demonstrated collaborative optimization with large-scale design using industry-standard analyses and showed that collaborative optimization is ready for real-world implementation. At the same year, Reuther et al. [32] presented a new high-fidelity modelling framework for the coupled optimization of aero-structural systems. An adjoint solver used to obtain the aerodynamic sensitivities. In addition, to discussion on evaluating trade-offs between aerodynamic performance and structural weight for complete aircraft configurations.

In 2001, Maute et al. [33] presented an aero-structural analysis that coupled the Euler equations to a linear finite-element model. The adjoint and direct methods were used for computing the sensitivities of the coupled aero-structural system. In addition, the solver was a nonlinear block Gauss–Seidel method with relaxation. Later in 2002, Martins [34] developed a framework couples a finite-volume Euler CFD solver with a linear finite-element structural model. The framework uses an accurate and efficient method to calculate the sensitivities of aerodynamic and structural cost functions with respect to both aerodynamic shape and structural variables. In 2004, Maute and Allen [35] formulated an aero-structural optimization problem to determine the topology of the optimal structure. Barcelos et al. [36] presented, in 2006, a class of Newton–Krylov–Schur methods for solving the coupled nonlinear fluid-structure-mesh movement problem. It showed improvements in terms of robustness and efficiency. In 2008, the same authors, Barcelos and Maute [37], Compared the optimization results for Euler and Navier–Stokes flow models and showed the importance of accounting for viscous laminar and turbulent effects on the optimization results.

Kennedy and Martins (2010) presented a comparison of methods for aero-structural analysis and optimization. They used an aerostructural model that combines a panel method and a finite-element solver. In this study, they used variety of nonlinear solvers including; a nonlinear block Gauss-Seidel, nonlinear block Jacobi, Newton-Krylov or approximate Newton-Krylov approach. The aerostructural problem was induced drag minimization for a typical subsonic turboprop aircraft wing.

In 2011, Jorge [3] conducted aerostructural design optimization for sailplane wing using high-fidelity MDO framework. He employed a multi-disciplinary feasible architecture. The aerostructural model combined a panel method coupled with a finite-element solver.

Many other researches and studies were published on aerostructural optimization in few past years with a variety of architectures, solvers, optimization algorithms [38] [39] [2].

Although MDO emerged firstly in aeronautics applications, it was extended to other areas of applications. Such as bridges, buildings, rail way cars, microscopes, automobiles, ships, propellers, rotorcraft, wind turbines, and spacecraft [40]. With all these various applications, MDO is still promising more in the future.



CHAPTER 3

SOARING AND SAILPLANE

3.1 Introduction

This chapter introduces the soaring flight and sailplanes. It begins with a brief history of soaring and its basics. Then, it discusses the three phases of the soaring flight and the main characteristics in sailplane performance. After that, a classification of the sailplanes is presented.

3.2 Soaring Flight

3.1.1 Brief History

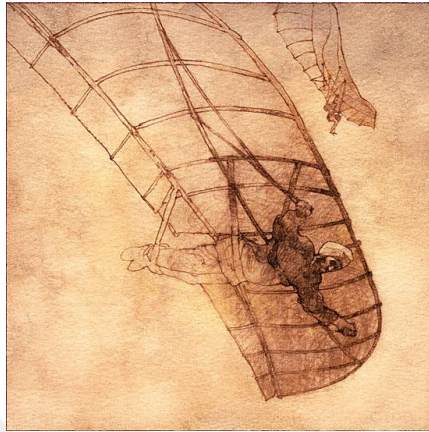
The dream of flying associated with people since ancient times, they spared no effort to make this dream come true. The early attempts of flying were basically a simulation of birds flying, like what Abbas ibn Firnas did in the 800s in Cordoba, Al Andalus (today South Spain). He was a polymath, who studied birds, falling seeds, leaves, feathers, and bats and devoted more than 20 years to building what seems to have been the first modern glider, with wings sewn of silk, wood, and actual feathers. In order to examine his invention, he jumped off the hillside into the wind and remained airborne for 10 minutes. Only upon landing, he discovered that he had no control on descent so he hit the ground fast and injured himself.

In the 15th century and as a step forward in glider design, Leonardo Da Vinci drew his first impressions of what a flying machine might have looked like based on the wing of a bat [41].

Later on, in the 16th century, an attempt carried out in Istanbul by another polymath Hezarafen Ahmed Çelebi, he built a set of glider wings that he could attach to his body. Hezarafen jumped from the top of the Galata Tower in Istanbul with his wings and managed to glide about 2 miles away crossing of the Bosphorus successfully [7].

In the 18th century, Sir George Cayley, an English engineer, conceived a craft which integrated the basic aerodynamic principles. This craft came with fixed wings to provide

lift, flappers to provide thrust, and a movable tail to provide control. After that, nearly 100 years later, Otto Lilienthal the German pioneer followed the experimental approach established earlier by Sir George Cayley. Otto was the first person to conduct successful gliding flights beginning in 1891 [8].



a. Ibn Firnas's gliding machine foreground, Leonardo Da Vinci's at rear [42]



b. Hezarafen taking off from atop the Galata Tower in Istanbul [43]

Figure 3.1. Early gliding attempts.

Wright Brothers were experimenting with gliders and gliding flight. They managed to develop a series of gliders from 1900 to 1902. Their experiments associated with aerodynamics in order to attain doable control system. By 1903, Orville and Wilbur Wright succeeded to achieve powered flight of just over a minute by putting an engine on their best glider design [8].

In 1911, Orville Wright had set a world duration record of flying his motorless craft for 9 minutes and 45 seconds. This record was broken after ten years by Dr. Wolfgang Klemperer with a soaring duration of 13 minutes [8].

By 1920, the gliding as a sport was increasing in line with developing more efficient gliders. Where the World War I Treaty of Versailles restricted flying power aircraft in Germany. As a result, German engineers focused more on developing efficient sailplane design and using the atmospheric resources to attain more lift. At this time, many soaring duration records were set and some of them reached hours. A turning point in soaring history was in 1928, Austrian Robert Kronfeld proved that thermal lift could be exploited by a sailplane to gain altitude by making a short out and return flight [5]. During the World War II, gliders played a role in carrying military troops and heavy supplies within the battlefield. After the war, in the 1950s, many gliders clubs and factories established in line with holding World Soaring Championships.

The period of the 1960s and 1980s witnessed a rapid growth in soaring, as a result of advances in the related field, like; structural engineering, computational fluid dynamics, materials, and electronics. In addition, the usage of composite materials, efficient airfoils and Global Positioning System (GPS) had major advancements on modern soaring. Nowadays, sailplanes are a wonderful combination of performance and efficiency.

3.1.2 Basics of Soaring and Gliding Flight

Soaring (or gliding) flight is heavier-than-air flight without the use of thrust. Thus, only three forces act on the sailplane; Lift, Drag and Weight as Fig. 3.2 illustrates.

Although a sailplane is not powered, there is still a need to obtain thrust. The sailplane does this by converting the potential energy that it has accumulated into kinetic energy as it glides downward, trading height for distance.

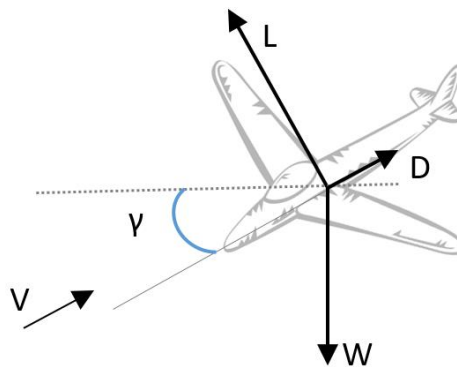


Figure 3.2. Forces on a sailplane.

A sailplane is always descending in the air. Where, it is constantly losing height in order to maintain forward speed. The more the nose is tilted downward, the more speed is given to the sailplane, but at the same time, the more height is lost in the process [44]. Fig. 3.2 shows the forces acting on a sailplane during the flight and also shows the resultant “nose down” flight path, which is measured by an angle called glide angle. This angle is a bit exaggerated in the Figure to show the principle.

As mentioned, only three forces affect a sailplane during the flight. Let us discuss these forces on more details.

Lift

Lift Force is the vertical component of the aerodynamic resultant force, which acts on the aerodynamic center. It is perpendicular on the velocity direction and opposes the downward force of the weight. The wing produces the major fraction of the sailplane lift.

The shape of the airfoil creates a difference in pressure distribution between the upper and lower surfaces of the wing. This difference is what produces the lift. At the same time, this pressure difference also varies with the relative speed and direction between the wing and airflow direction.

Eq. (3.1) is the mathematical relationship of lift. It shows that the lift is related to the angle of attack (AOA), airspeed, altitude, and the area of the wing.

$$L = \frac{1}{2} \rho V^2 S C_L \quad (3.1)$$

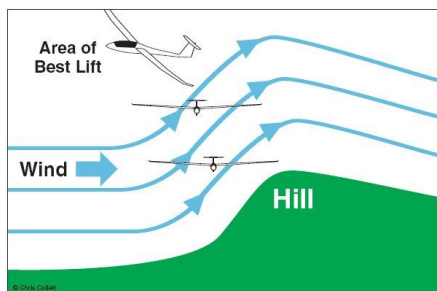
Where, ρ is air density, V is the airflow speed, S is the reference wing area, and C_L is the lift coefficient.

Sailplanes need lift to remain airborne, yet the absence of power on a sailplane made that difficult until discovering outer energy sources to gain lift. These sources are natural atmospheric phenomena and there are three of them as following:

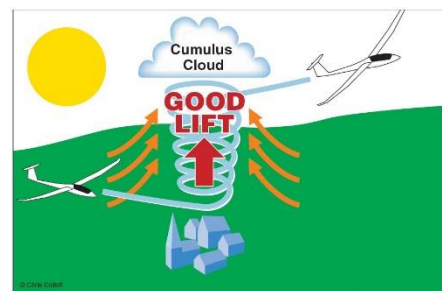
Ridge Lifts: It is created when a wind approaches an obstacle, like a mountain ridge, that is enough to direct the wind upward (Fig. 3.4-a). If the wind is strong enough, the ridge lift offers a good amount of upward force for sailplane to stay airborne or even gain more lift.

Thermals: As the sun heats up differently colored land areas at different rates, this creates bubbles or streams of rising warm air, called thermals (Fig. 3.4-b). When encountered, it is necessary for the sailplane to circle inside the thermal to gain lift. It was discovered in 1928 and still the most common method of soaring [44].

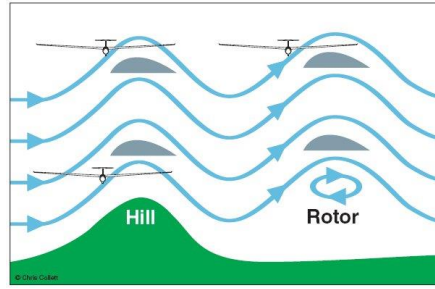
Wave Lifts: It occurs when the wind blows over mountains or ridges, this often bounces and sets up one or more standing waves downwind of the ridge or mountain (Fig. 3.4-c). Thus, by flying back and forth along the wave a sailplane can climb to great heights.



a. Soaring using Ridge Lifts.



b. Soaring using Thermals.



c. Soaring using Wave Lifts.

Figure 3.3. Natural atmospheric phenomena to gain lift [bookergliding.co.uk].

Drag

Drag Force is the horizontal component of the aerodynamic resultant force, also acts on the aerodynamic center and parallel to the velocity direction. Drag minimization is a major concern for designers. Two different types of drag on a sailplane, parasite and induced.

Parasite Drag: This type is composed of three types as following:

- Form Drag, which is air resistance to motion due to the shape of the sailplane,
- Skin Friction Drag, which is due to the smoothness or roughness of the sailplane surfaces, and
- Interference Drag, which is due to the interaction between surfaces with different characteristics (e.g. wing and fuselage).

The wing of the sailplane alone has very low parasite drag, but when the total drag of the sailplane is added to it, the amount of drag becomes significant. Parasite drag can be reduced by streamlining the non-lifting parts of the sailplane, like fuselage, and keeping all sailplane surfaces as smooth as possible.

Induced Drag: this part of drag arises from the development of lift and is typically written as:

$$C_{Di} = \frac{C_L^2}{\pi AR e} \quad (3.2)$$

Where, AR is the aspect ratio, e is Oswald's efficiency factor. As Eq. (3.2) indicates, the larger the aspect ratio of the wing is, the lesser the induced drag is. Therefore, sailplane designers always aim to reduce drag by increasing the aspect ratio of the glider.

The two types of drag have different relationships with airspeed, where, parasitic drag increases with the square of the airspeed. On other hand, induced drag, as a function of

lift, is decreasing as the airspeed increases. Fig. 3.5, illustrates the relationship of parasite and induced drag with airspeed.

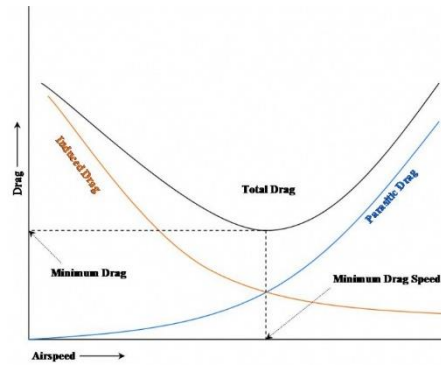


Figure 3.4. Relationship of drag components with airspeed.

The total drag is written mathematically as following:

$$C_D = C_{D0} + C_{Di} \quad (3.3)$$

Where, the first term is the form drag and the second is the induced drag.

In this thesis, minimizing the induced drag is one of the main objectives. That is to be achieved through obtaining an optimized wing design with elliptical lift distribution.

Weight

Weight is the third force that acts on a sailplane during flight. Weight opposes lift vertically and acts on the center of gravity of the sailplane (Fig. 3.3). In addition, the other portion of the weight force, the one parallel to the flight path, is what makes the sailplane moving forward.

The weight of a sailplane is kept as low as possible consistent with adequate strength. The lift produced by the wings acts in opposition to the weight of the whole sailplane. It will, therefore, be apparent that the lighter the structure and the bigger the wing, the lower will be the resultant rate of sink [9] as Eq 3.10 indicates.

One of the main objectives of this thesis is minimizing the weight of sailplane through minimizing the wing structural weight.

3.1.3 Phases of Soaring Flight

Take –off and Climbing

The attitude of the glider is gently and smoothly graduated to the full climb attitude at this stage. By definition, a sailplane has no power (engine). Therefore, it needs an outer

launching method to take off and climb. A number of methods used to do so, for example [45];

1. *Launch by Elastic Cord.* This is the simplest method, where an elastic chord attached to the glider's nose is used in a certain angle in order to obtain proper tension to launch the glider after releasing the chord.
2. *Launch by Ground Winch.* This method consists of a large rotating drum driven by a powerful motor. The glider is pulled by a steel cable, of approximately 1000 meters in length, that winds onto the drum. With this method, gliders could gain altitudes of 200 -250 m.
3. *Launch by Automobile Tow.* In this method, an automobile is used to tow the glider on the ground using cable, until the glider reaches enough speed to fly. Normally, the cable is of 1000 to 3000 meters in length.
4. *Launch by Airplane Tow.* This method is used for launching large sailplanes, when altitudes from 500 to 1200 m are targeted. In this method, other airplane using a cable of 60 to 100 meters in length tows the sailplane. When the preferred height and settings are reached, the sailplane releases the cable.

Cross-Country Flight

A cross-country flight is defined as one in which the sailplane has flown beyond gliding distance from the local soaring site [46]. In other words, it is a distance flying using up-currents to gain altitude for extending flight time of a sailplane.

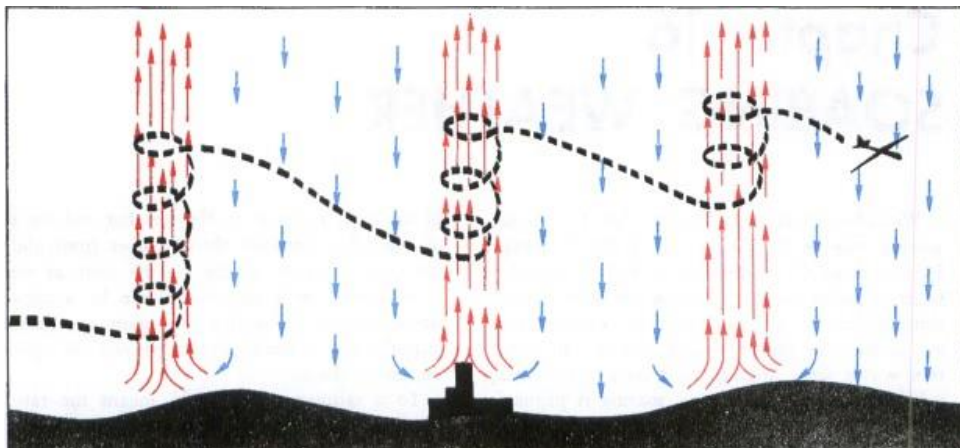


Figure 3.5. An example of cross-country soaring using thermals [aviationweather.ws]

This phase of flight could take hours and the sailplane could travel for hundreds of kilometers during it. Where, the sailplane is repeatedly climbing and then gliding, trying

always to get enough altitudes for more soaring. The sailplane in this phase depends mainly on the natural lift sources mentioned earlier.

Landing

At final approach, each of direction, speed and sink speed should be carefully managed. Generally, sailplanes have high L/D ratio, they need to activate the airbrakes at specific angles in order to reach the landing speed.

3.2 Sailplane Flight Performance

The Performance of sailplane can be evaluated by its maximum range or maximum endurance. However, each one of these two figures is related to performance parameter. The maximum range corresponds to maximum glide ratio. At the same time, maximum endurance corresponds to minimum sink speed.

In order to understand these parameters better, we assume a sailplane at equilibrium ($L=W$), following the equations of motion as:

$$D = W \sin(\gamma), \quad (3.4)$$

$$L = W \cos(\gamma). \quad (3.5)$$

Where D is drag force, W is sailplane weight, L is the lift force, and γ is the flight path angle, which in a steady state glide called the *glide angle*. Dividing one equation by the other:

$$\tan(\gamma) = \frac{D}{L} = \frac{CD}{CL} \quad (3.6)$$

This angle is independent of the weight and its lowest value corresponds to the higher L/D ratio:

$$\frac{L}{D} = \frac{CL}{CD} \quad (3.7)$$

The lift-to-drag ratio is known as the *glide ratio*. This is the most important parameter in sailplane performance. Most sailplanes have maximum glide ratios between 20 and 50. The corresponding glide angles lie between approximately 3 and 1 Deg [47].

Glide Range:

Assuming the sailplane descends from initial height (h_1) to another height (h_2), we can define the *glide range*, R , as the horizontal distance a sailplane covers between two given altitudes. It is calculated as following:

$$R = \frac{h_1 - h_2}{\tan(\gamma)} = \frac{CL}{CD} (h_1 - h_2) \quad (3.8)$$

The equation (3.8) states clearly that maximum range (R_{\max}) is corresponded to maximum glide ratio $(CL/CD)_{\max}$.

Glide Endurance:

The *glide endurance*, T, is the time that takes a sailplane to descent between two given altitudes and it is calculated as following:

$$T = \frac{\Delta h}{V_s} \quad (3.9)$$

The maximum endurance in still air is obtained by maintaining the sink speed minimum [48].

With the assumption that the glide angle is small, the sink speed can be written as following:

$$V_s = -V \frac{D}{W} \approx -V \frac{D}{L} = \sqrt{\frac{W}{\frac{1}{2}\rho s}} \frac{C_D}{C_L^{3/2}} \quad (3.10)$$

This equation, which relates the airspeed to the wing loading (W/S) and the lift coefficient CL, is of fundamental importance to sailplane performance analysis. To have minimum sink speed, both of the two quantities wing loading (W/S) and $CD/CL^{3/2}$ should be minimum.

Usually, the sailplane performance is determined using speed polar or flight polar, which shows the sink speed as function of the airspeed (Fig. 3.6).

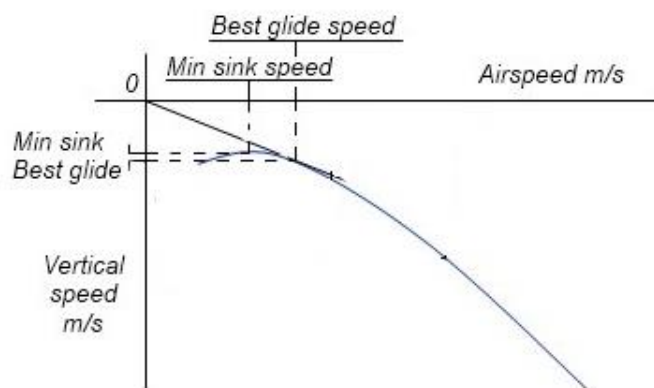


Figure 3.6. Typical speed polar of a sailplane.

Using this polar, gives information on important speed values, such as; the speed-to-fly which is the speed that gives maximum gliding range. Also, one can use this polar to

obtain the speed value which gives minimum sink rate. The important thing to know that the speed for maximum range is different in value from the speed for minimum sink speed.

3.3 Sailplanes Classification

The sailplanes come in vast range of types since their beginnings. This depends on the mission definition of the sailplane, such training, sport, competition..etc. In addition to nature of updrafts which the sailplane is designed to operate in. However, the most popular classification is the *Competition Classification* by The Fédération Aéronautique Internationale (FAI). FAI is the world governing body for air sports, aeronautics and world records in aeronautics. FAI classification is as listed in Table 3.1 [48].

Table 3.1. Competition sailplanes classification by FAI.

| Class Name | Description |
|----------------------------------|--|
| Open Class | No restrictions. Max mass (MTOV) is 850 kg. |
| Standard Class | Max wingspan 15 m , no camber changing devices. Max mass (MTOV) is 525 kg. |
| 15 meter Class | Max wingspan 15 m, camber changing devices permitted, max mass (MTOV) is 525 kg. |
| 18 meter Class | Max wingspan 18 m, max mass (MTOV) is 600 kg. |
| 20 meter Two-Seater Class | Max wingspan 20 m, max mass (MTOV) is 750 kg. |
| World class | Limited class for a single type of sailplane – the PW-5. |
| Club Class | For older and smaller gliders within a specified range of performances. |

In the case study of this thesis, a wing design for *15-meter class* sailplane is used.

CHAPTER 4

MULTIDISCIPLINARY DESIGN OPTIMIZATION

4.1 Introduction

This Chapter introduces a brief of multidisciplinary design optimization approach and formulation. In addition, the optimization algorithms are classified and briefly discussed. Furthermore, a definition of Extensions to the Design Structure Matrix (XDSM) is presented in order to understand the main differences in MDO architectures, which are discussed in this Chapter as well.

4.2 Multidisciplinary Design Optimization Approach

4.2.1 MDO Definition and Application

Multidisciplinary Design Optimization (MDO) is a field of engineering that utilizes the techniques of numerical optimization for engineering systems design that involves multiple disciplines or subsystems (e.g., Aerodynamics, Structure, Control, ...etc.). The need for MDO came from the fact that the performance of a multidisciplinary system is driven not only by the performance of the individual disciplines but also by their interactions.

MDO began in the 1980s following the success of the application of numerical optimization techniques to structural design in the 1970s. Aircraft design, as a complex engineering system, was one of the first applications of MDO because there is much to be gained by the simultaneous consideration of the various disciplines involved (structures, aerodynamics, propulsion, stability and controls, etc.), which are tightly coupled [49]. Recently, MDO application has been extended to the design of complete aircraft, as well as a wide range of other engineering systems, such as bridges, constructions, automobiles ...etc.

In the process of design optimization, a well-defined optimization problem is one of the important roles of the designer besides other important numerical choices.

4.2.2 MDO Problem Formulation

Problem formulation is the selection of design variables, constraints, objectives. Normally, this is the difficult part of the optimization. These three entities are defined as follows:

4.2.2.1 Objective Function

The objective function, f , is the numerical value that is to be maximized or minimized. The choice of objective function is governed by the nature of the problem. Many methods work only with a single objective. For our case, we work on minimization of drag coefficient and weight.

4.2.2.2 Design Variables

The design variables, x , are the parameters that are varied in a specific range by the optimizer in order to maximize/minimize the objective function. For instance, the thickness of a structural element and twist angle of a wing can be considered design variables. Design variables are often bounded by upper and lower values.

Many types of design variables could be used in an optimization problem, such as quantitative and qualitative, local and global, continuous and discrete.. etc.

4.2.2.3 Constraints

The constraints are conditions to be satisfied in order for the design to be logical and feasible. One of the most popular constraint in aircraft design is for the lift produced by the wing to equal the weight of the aircraft. Two main types of constraints, the first one is the equality constraint, which is the condition is restricted equally to a fixed quantity. The other type is, inequality constraint, which is the condition is required to be greater/smaller or equal to a certain quantity.

4.2.3 Optimization Problem Statement

Mathematically, the Optimization problem is given as:

Minimize $f(x)$ by varying x ; from x_1 to x_2

subject to $C_1 = 0$ & $C_2 \geq 0$

f : Objective function, (e.g. structural weight or drag coefficient).

x : Vector of design variables, (e.g. twist, thickness ..); x_1 is the lower bound and x_2 is the upper bounds.

C_1 : Vector of equality constraints (e.g. lift = weight);

C_2 : Vector of inequality constraints (e.g. structural failure).

4.3 Optimization Algorithms

The integrated components of the optimization process in any optimization problem are the optimization algorithm, which is an efficient numerical simulator and a realistic representation of the physical processes we wish to model and optimize. A good attention should be paid on the subject of formal optimization methods and theories when combining multiple disciplines in one single model. Selecting a suitable optimization algorithm is as much as important of selecting variables and constraints. Yang [48] has mentioned three main issues in the simulation-driven optimization and modeling, which are; 1) the efficiency of an algorithm, 2) the efficiency and accuracy of a numerical simulator, and 3) assign the right algorithms to the right problem.

Two main categories of numerical optimization algorithms, the gradient-based and gradient-free algorithms. However, there is a third category called the hybrid method, which is a combination of the two methods mentioned. Although there are many existed optimization algorithms in the literature, there is no one-algorithm suits all problems.

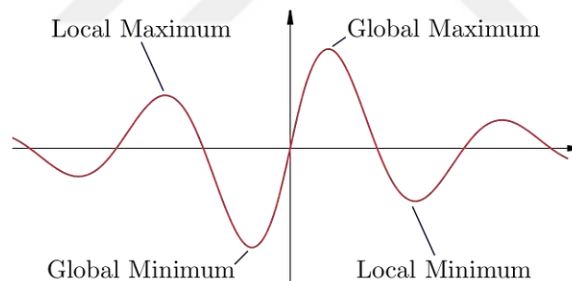


Figure 4.1. Local versus global minima/maxima.

4.3.1 Gradient-based Algorithms

This type of algorithms uses the gradient of the objective function to direct the design solution into the direction of improvement. The gradient-based algorithm determines the direction of steepest descent through the gradient information of the objective function. After that, one move or whole line search is conducted in the direction of steepest direction determined. Then, one more time, the optimizer determines the direction of the steepest descent and makes moves according to it. This process is conducted repeatedly until obtaining the optimum solution based on the optimization criteria. One of the main drawbacks of the gradient-based method that it more often drive the solution to a local

optimum, which is not necessarily to be the same of global optimum. Thus, this method guarantees only the convergence to the local minimum. Knowing that the selection of initial design points to be in the surrounding of the global minimum would be of great importance. However, this could not be the practice for real-life optimization problems. So, gradient-based methods could output a quality performance in theory, yet less quality in practice [50].

Examples of gradient-based methods are Newton methods, Quasi-Newton methods, steepest descent, trust region method, conjugate gradients ... etc. The main advantages of this method comparing to the gradient-free method are it is significantly faster in convergence and easier in implementation.

4.3.2 Gradient-free Algorithms

The gradient-free algorithms are more suitable for obtaining the global optimum, where they are able to explore the whole design space, however, require a large number of function evaluations. Many gradient-free algorithms are designed to find global optimum and thus are able to find multiple local optima until finding global optimum [49]. One of the disadvantages of this method comparing to the gradient-based methods is the slow convergence. The most common examples of gradient-free methods are Nelder–Mead Simplex, Simulated Annealing, Divided Rectangles method, Genetic Algorithms, Particle Swarm Optimization ...etc.

4.4 Extensions to the Design Structure Matrix (XDSM)

XDSM is a diagram based on extending the standard design structure matrix (DSM) for visualizing MDO processes. It simultaneously shows data dependency and process flow on a single diagram. This diagram developed by Lambe and Martins and more detailed information can be found at their publication [51].

Here is the main syntax used to define (and understand) XDSM Diagram:

Shapes, which represent the component of the model as following:

- Rectangles for generic processes,
- Parallelograms for data input and output,
- Rounded rectangle for MDA (Multidisciplinary Design Analysis).

Connections, which distinguish the track of data connection, form the other track of computational analysis process:

- Thick gray lines for data connections,
- Thin black lines for process connections.

Both computational process and data transfer are executed in an order defined by numbering components, starting from zero. XDSM illustrates the process in numerical order, from 0 to 5 for example, unless there is a loop during the process. A loop between certain components still running until the condition defined by the driver is satisfied. Also, Table 4.1 define each symbol used.

Table 4.1. List of symbols used in XDSM.

| <i>Symbol</i> | <i>Description</i> |
|----------------|--|
| x | Vector of design variables. |
| y^t | Vector of design variables targets. |
| y | Vector of coupling variable responses. |
| f | Objective function. |
| c | Vector of design constraints. |
| c^c | Vector of consistency constraints. |
| N | Number of disciplines. |
| $(\)_0$ | Shared Functions or variables by more than one discipline. |
| $(\)_i$ | Functions or variables of only discipline i . |
| $(\)^*$ | Optimal value of Functions or variables. |
| $\tilde{(\)}$ | Approximation of a given function or vector of functions. |

As a simple example of XDSM application, Fig. 4.2 illustrates the solution of a standard nonlinear programming problem.

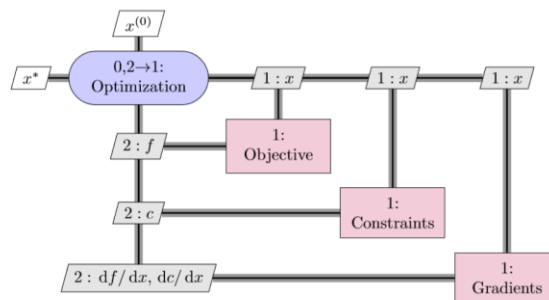


Figure 4.2. Example of XDSM representation of gradient-based optimization problem [40].

4.5 MDO Frameworks and Architecture

MDO architecture is the combination of the components of the design problem and the organizational framework used to find the optimal solution [51]. Many MDO architectures have been developed and evaluated since the emergence of MDO. The

examples of these architectures include All-at-once (AAO) architecture [52], shown in Fig. 4.3, individual discipline feasible (IDF), Shown in Fig. 4.4, multidisciplinary feasible [52], shown in Fig. 4.5, concurrent subspace optimization (CSSO) architecture [53], shown in Fig. 4.6, collaborative optimization (CO) architecture [54], shown in Fig. 4.7, bi-level integrated system synthesis (BLISS) architecture [55], shown in fig. 4.8, and others.

Determining which most effective architecture for a certain optimization problem is an active field of research [51]. However, the selection of an architecture is subjected to a number of aspects, like for instance, the number of the disciplines involved, the number and type of design variables, the level of fidelity and the optimization methods. Currently, most studies use gradient-based methods [56]. Set of requirements have been put forward by [57] for MDO framework to have, including Architectural Design, Problem Formulation, Problem Execution and Information Access.

The MDO architectures can be classified in: single-level methods and multilevel methods. Single-level methods, like Individual Discipline Feasible (IDF) or Multi-Disciplinary Feasible design (MDF), include only one optimizer at a system-level, which runs a system analysis in each step and has authority over the global system [52].

Illustrations of the mentioned architectures are provided here using XDSM diagram.

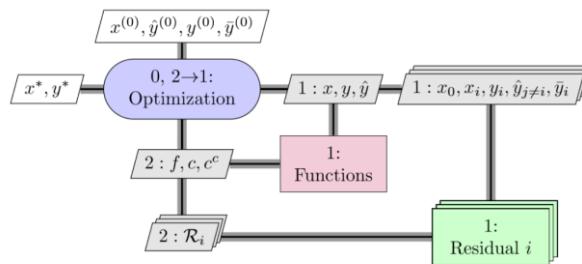


Figure 4.3. XDSM diagram of AAO Architecture [51].

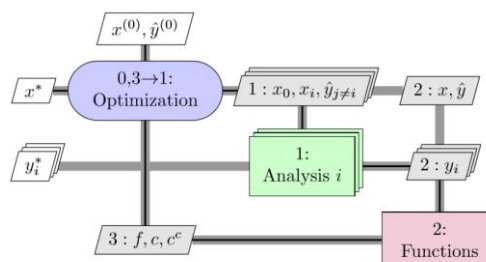


Figure 4.4. XDSM diagram of IDF Architecture [40].

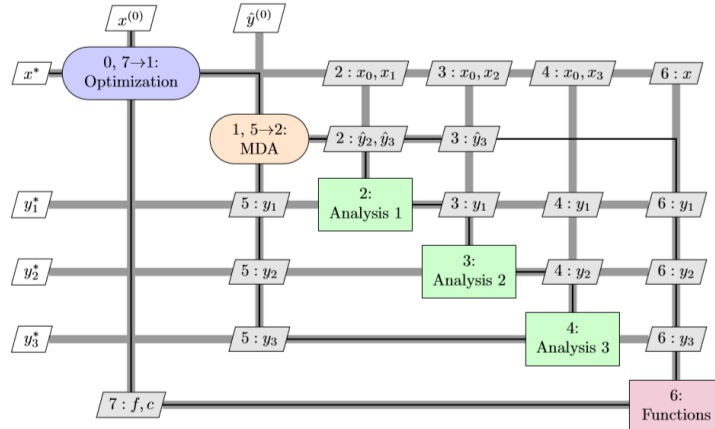


Figure 4.5. XDSM diagram of MDF architecture with a Gauss–Seidel multidisciplinary analysis [40].

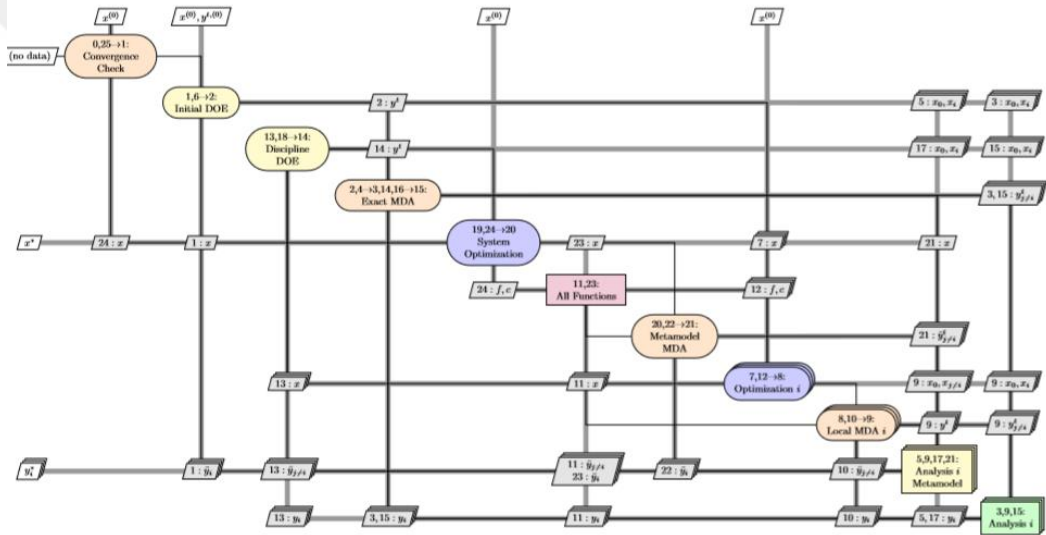


Figure 4.6. XDSM diagram of CSSO Architecture [40].

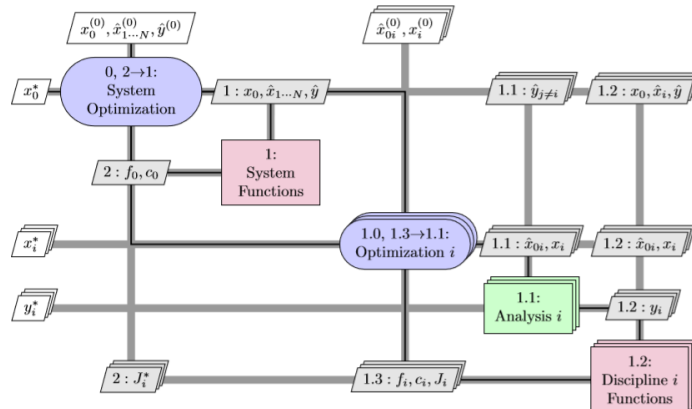


Figure 4.7. XDSM diagram of CO Architecture [51].

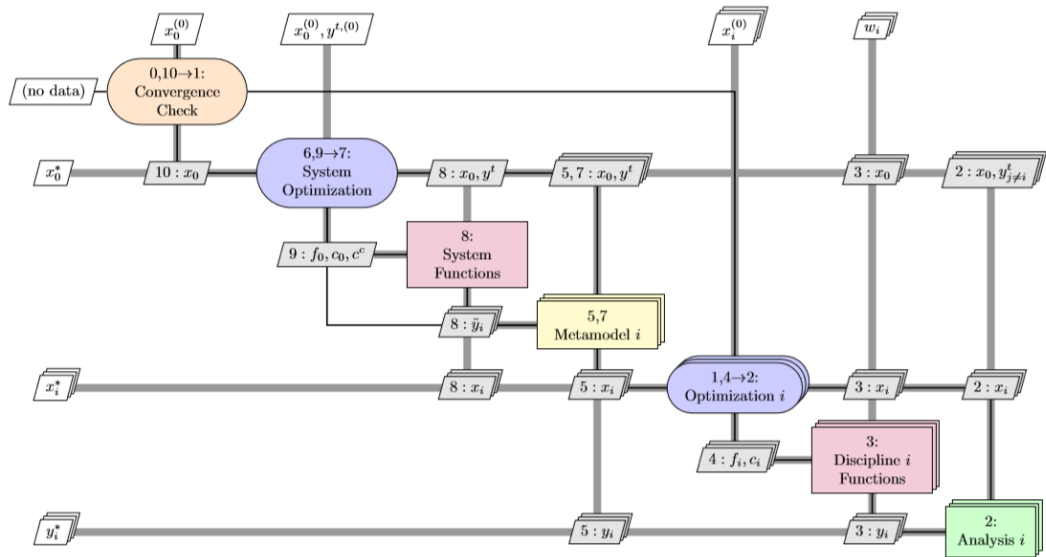


Figure 4.8. XDSM diagram of BLISS-2000 Architecture [51].

CHAPTER 5

AEROSTRUCTURAL IDEALIZATION OF THE SAILPLANE WING

5.1 Introduction

In this Chapter, descriptive details of the aerostructural components are presented. Starting with introduction of the MDO frameworks used to carry out the aero-structural analysis and optimization. Then, presenting both aerodynamic and structural theories, and aero-structural coupling mechanism. After that, a case study for sailplane wing is provided.

5.2 Aerostructural Optimization Framework

In this research, an aerostructural optimization tool, called OpenAeroStruct (OAS) [58]. OAS is an open-source coupled aerostructural analysis and design optimization tool, developed within the OpenMDAO framework, a NASA-developed open-source software framework for multidisciplinary design optimization (MDO). Following, an introduction of these two frameworks is presented.

5.2.1 Open MDAO Framework

OpenMDAO (MDAO stands for Multidisciplinary Design Analysis and Optimization) is an open-source high-performance computing platform for systems analysis and multidisciplinary optimization, written in Python and NASA-developed. The main motivation for developing this framework was to aid in the design of unconventional aircraft. However, the general structure and methods may be applied to solve any number of engineering-related design problems [59].

OpenMDAO provides a library of sparse solvers and optimizers, and works with both gradient-free (e.g., genetic algorithm, particle swarm) and gradient-based optimization methods. One other hand, OpenMDAO has the ability to subdivide a problem into groups or components and link their inputs and outputs, making it easier to decompose large and complex problems.

OpenMDAO performs data transfers between components, across processors in parallel settings, and includes nonlinear and linear solvers with support for matrix-free or sparse linear algebra. Any nonlinear or linear solver can be applied in each group or component in the hierarchy tree. This means that OpenMDAO in general uses hierarchical solution approaches. For example, if we have a problem with many components, we can solve a component with Newton solver while the group containing that component can use a GMRES solver.

OpenMDAO has been used to solve MDO problems in satellite design [59], wind turbine design [60], aircraft design [61] and aircraft trajectory optimization [62].

5.2.2 Open AeroStruct Framework

OpenAeroStruct (OAS), an open-source coupled aerostructural analysis and design optimization tool. It couples the Vortex-Lattice Method (VLM) and finite-Element Analysis (FEA) using six degree-of-freedom (DOF) spatial beam elements with axial, bending, and torsional stiffness. It is mostly implemented in Python, but some of the more intensive computations use FORTRAN. The main objective of OAS is to provide a physically meaningful multidisciplinary model that can be used to obtain low-order approximations of aircraft performance or to compare solution algorithms [56].

OAS is developed within the OpenMDAO framework, so, it computes derivatives for the aerostructural system using the coupled adjoint method. A variety of solvers can be used to converge the aerostructural optimization problem, solvers like block Gauss–Seidel, GMRES, or LU decomposition for the linear system, and nonlinear block Gauss–Seidel or Newton for the nonlinear system [56].

The modular implementation and the efficiency in gradient computation made OAS a unique tool. The model is broken down into many low-level computations. As OAS is built upon OpenMDAO, these computations are executed within OpenMDAO. Then, it calculates the whole model-level derivatives given component-level derivatives of the outputs with respect to the inputs of each component.

OAS code serves as educational and researches MDO tool. It has been used in graduate-level MDO courses at ISAE-SUPAERO (the University of Toulouse’s Institute for Aerospace Engineering) and the University of Michigan [56]. As well as, It has been used in research filed as has been used as a realistic testbed application by Bons et al. [63], Cook et al. [64], and Friedman et al. [65].

The default aerostructural optimization problem in OAS is a fuel-burn minimization using the Breguet Range Equation. However, the modular implementation of the code makes it easy to reformulate the optimization problem and define other objective and constraints functions. As in our case, a function for sink speed was added as the objective function. The sink speed Eq. (3.10), is the function of both structural variables i.e. weight and aerodynamic coefficients i.e. lift and drag coefficients. Thus, the minimum sink speed corresponds with minimum weight and minimum drag at the fixed lift.

5.3 Aerostructural Idealization of the Wing

OAS couples the vortex-lattice method (VLM) as aerodynamic solver and 1-D finite-element analysis (FEA) as a structural solver. These two models are linked with each other in a system with Multidisciplinary analysis solver (MDA) and Optimizer. The following sections describe these components in more details.

5.3.1 Aerodynamic Idealization

The aerodynamic model in OAS, calculates the total lift and drag. However, the induced drag is calculated using VLM, whereas, the viscous drag is calculated using flat-plate-based estimation.

Vortex Lattice Method (VLM) is used to compute the aerodynamic forces and moments acting on the wing. Specifically, it computes lift and induced drag. VLM models the wing as a lifting surface with superimposing infinite horseshoe vortices on that surface. This combines multiple modern numerical lifting-line theory (LLT) models [58]. The velocities induced by each horseshoe vortex at a specified control point are calculated using the Biot-Savart law [66].

A summation is performed for all control points on the wing to produce a set of linear algebraic equations for the horseshoe vortex strengths that satisfy the boundary condition of no flow through the wing i.e. velocity normal to the panel is zero. The vortex strengths are related to the wing circulation and the pressure differential between the upper and lower wing surfaces. The pressure differentials are integrated to yield the total forces and moments.

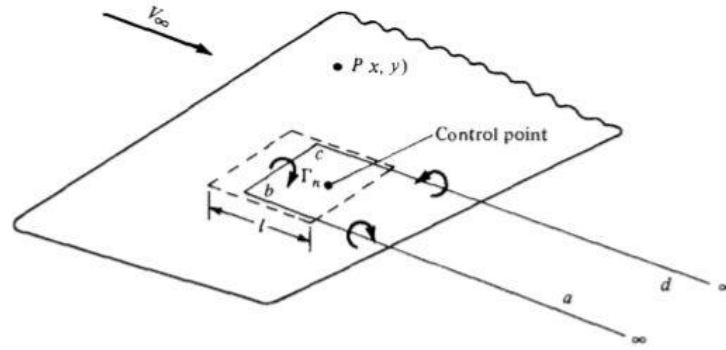


Figure 5.1. Single horseshoe vortex on lifting surface [67].

As a vortex grid in VLM represents the wing, each horseshoe vortex consists of a bound vortex in the spanwise direction and two trailing vortices that extend into the freestream direction [67]. Fig. 5.1 shows a single horseshoe vortex on a lifting surface. The circulation induced by the flow field is what produces the lift. It represents the strength of the vortex filament.

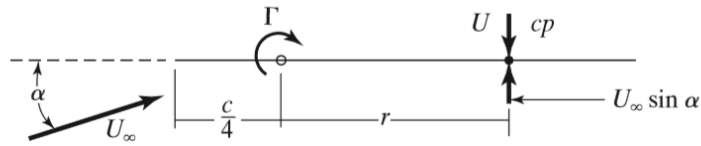


Figure 5.2. Planar airfoil section indicating location of control point and bound vortex ($r=c/2$) [66].

The velocity induced by a vortex filament of strength Γ and a length of $d\ell$ is given by the Biot-Savart law [66]:

$$d\mathbf{V} = \frac{\Gamma}{4\pi} \frac{d\boldsymbol{\ell} \times \mathbf{r}}{|\mathbf{r}|^3} \quad (5.1)$$

Where, r is the distance of the control point from the vortex filament. By integrating for a semi-infinite straight vortex element, yields:

$$V = \frac{\Gamma}{4\pi d} \quad (5.2)$$

Where d is the distance from point P to the finite start point of the vortex filament. For a panel with length ℓ , the control point locates on the centerline of the panel and at a distance of $\frac{3}{4}\ell$ of the front of this panel. While, the bound vortex is at a distance of $\frac{1}{4}\ell$ from the front of the panel as well.

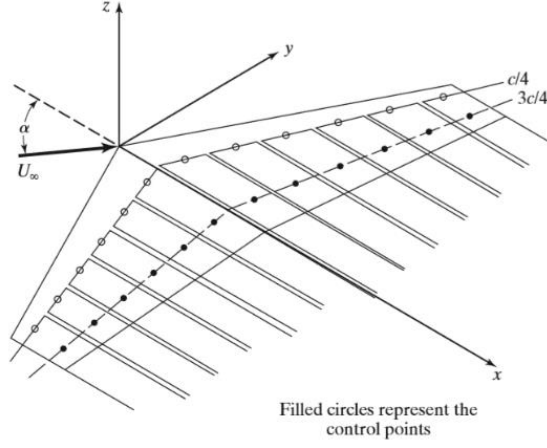


Figure 5.3. Distributed horseshoe vortices over a wing [66].

To form a lifting surface, multiple horseshoe vortices should be placed over the span. Similar, to LLT model (vortices aligned in spanwise direction) except vortices in the chordwise direction, are superimposed as well, Fig. 5.3. Now, enforcing a flow tangency condition at this control point that specifies that the velocity normal to the panel must be zero. Physically, this means that flow cannot go through the lifting surface. Through applying the flow tangency conditions at each of control points of superimposed vortices, we obtain the linear system as follows:

$$\mathbf{A} \boldsymbol{\Gamma} = -\mathbf{V}_\infty \cdot \mathbf{n}. \quad (5.3)$$

Where \mathbf{A} is the aerodynamic influence coefficients matrix, \mathbf{V}_∞ is the freestream velocity, and \mathbf{n} is the normal to the panel. By solving this linear system, the circulation strengths for each horseshoe vortex on the lifting surface are obtained.

To obtain the total force vector \mathbf{F} and total moment vector \mathbf{M} about the origin, a summation of the contributions of all the forces \mathbf{F}_i is performed on all the individual horseshoe vortices, with ρ being the air density.

$$\mathbf{F}_i = \rho \Gamma_i (\mathbf{V}_\infty + \mathbf{v}_i) \times \mathbf{l}_i. \quad (5.4)$$

$$\mathbf{F} = \sum_{i=1}^N \mathbf{F}_i. \quad (5.5)$$

$$\mathbf{M} = \sum_{i=1}^N \mathbf{F}_i \times \mathbf{r}_i. \quad (5.6)$$

Where, \mathbf{l}_i is the bound vortex's vector, \mathbf{v}_i is the induced velocity at the center of the bound vortex with the distance \mathbf{r}_i .

The lift and drag are computed using these forces, which they are corresponding to force vector components. Drag is the freestream direction component and lift is the perpendicular component.

On the other hand, OAS aerodynamic model computes the skin friction drag using flat-plate-based estimates [58]. This skin friction drag estimate is based on the airfoil thickness-to-chord ratio, the Reynolds number, and other aircraft and flow properties. The drag estimate is adjusted using a form factor, which accounts for pressure drag due to flow separation. The semi-empirical models used for drag estimation are considered valid up to certain mach number, called drag-divergence Mach number [68].

5.3.2 Structural Idealization

OAS simulates the wing structurally as a spatial beam element. A tubular spar is used to represent the structure side of the sailplane wing. Mathematically, a finite element method (FEM) approach that uses spatial beam elements is employed, resulting in six degree-of-freedom (DOFs) per node. The spatial beam element (Fig. 5.4) is a combination of truss, beam, and torsion elements. In other words, It carries three types of loads at the same time, which are axial, bending, and torsional loads.

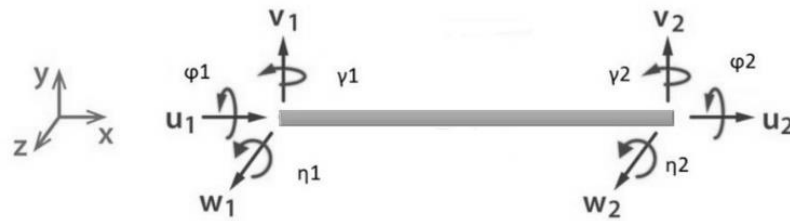


Figure 5.4. 6-DOF spatial beam element.

Each spatial beam element has 12 DOFs in total, as shown in Fig. 5.4 The displacements (both translational and rotational) illustrated are in the local coordinate frame. As shown, each end of the element has three translational DOFs; u, v, w aligned with x, y, z axis respectively. As well as, three rotational DOFs; φ, γ, η with respect to x, y, z axis, respectively.

OAS solves the linear system $\mathbf{K}\mathbf{u} = \mathbf{f}$, where \mathbf{K} is the global stiffness matrix, \mathbf{u} is the vector of displacements and rotations at the nodes, and \mathbf{f} are the forces and moments acting at the nodes.

The global stiffness matrix is given by:

$$K = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} \\ 0 & 0 & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 \\ 0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 \\ 0 & 0 & -\frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 \\ 0 & \frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} \\ -\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & 0 & 0 & -\frac{6EI_z}{L^2} \\ 0 & 0 & \frac{12EI_y}{L^3} & 0 & 0 & 0 & 0 & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 & 0 \\ 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 & 0 \\ 0 & \frac{2EI_y}{L} & 0 & 0 & \frac{4EI_y}{L} & 0 & 0 & 0 & 0 & \frac{4EI_y}{L} & 0 & 0 \\ 0 & \frac{2EI_z}{L} & 0 & 0 & 0 & \frac{4EI_z}{L} & 0 & 0 & 0 & 0 & \frac{4EI_z}{L} & 0 \end{bmatrix}$$

Symmetrical

Where, A is the cross-section area of the beam, L is the beam length, E is Young modulus, G is shear modulus, J is the polar moment of inertia, I_y , I_z is the torsional moment of inertia.

In OAS, spatial beam elements are always connected end-to-end in a single sequence, so the resulting global stiffness matrix exhibits a banded structure where stiffness submatrices are added on block diagonals. Once the stiffness matrix has been assembled [58].

5.3.3 Aero-Structural Coupling

Now, we need to go through the coupling mechanism of aerodynamic and structure models, and see how they will be linked to serve as one aerostructural model.

5.3.3.1 Load and Displacement Transfer Scheme

The load and displacement transfer process should grantee the accurate translation of the nodal displacement of the structure model to the aerodynamic mesh point displacements. As well as, transferring the loads on the aerodynamic panel to the structural nodes.

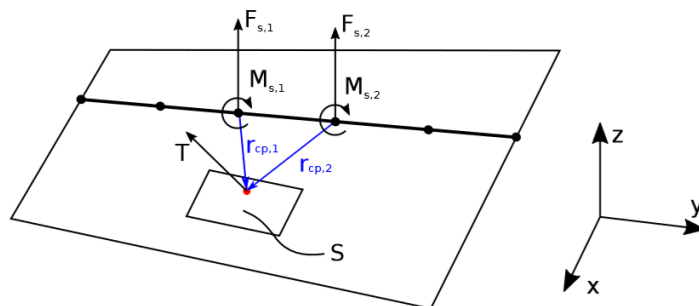


Figure 5.5. Load transfer from a panel on the aerodynamic mesh to adjacent structural nodes [58].

In OAS, the load and displacement transfer is simplified as aerodynamic and structural models have the same spanwise discretization. For the load transfer, each lifting surface produces a traction (T) equivalent to the net force computed by VLM and this force to be transferred to the structural nodes at the edges of this panel.

In addition, the center of pressure of the panel, is assumed to be at the centerline of the panel in the spanwise direction and at a quarter-length from the front of panel in the chordwise direction.

Thus, the transferred force and moment on each structural node is given as:

$$\mathbf{F}_{s,i} = \int_{panel} \frac{1}{2} \mathbf{T} dS = \frac{1}{2} \mathbf{T} S, \quad (5.7)$$

$$\mathbf{M}_{s,i} = \int_{panel} \mathbf{r}_i \frac{1}{2} \mathbf{T} dS = \frac{1}{2} \mathbf{r}_{cp,i} \times \mathbf{T} S. \quad (5.8)$$

where the subscript s refers to the structural mesh, $i = 1,2$ indicates the left or right node, S is the panel area, \mathbf{r}_i is the vector pointing from the structural node to a point on the panel, and $\mathbf{r}_{cp,i}$ points from the structural node to the aerodynamic center of pressure for the panel.

For the displacement transfer, the displacements on the structural mesh, which are resulted from the load transferred, need to be transferred back as deflections on the aerodynamic mesh (\mathbf{u}_a). These displacements are both translational displacements ($\mathbf{d}_{s,i}$) and rotations ($\boldsymbol{\theta}_{s,i}$). The displacement transfer is given by:

$$\mathbf{u}_a = \frac{1}{2} \sum_{i=2}^2 (\mathbf{d}_{s,i} + \boldsymbol{\theta}_{s,i} \times \mathbf{r}_i). \quad (5.9)$$

Since there are contributions from the left and right structural nodes, wherever they located on the panel, a $1/2$ factor is used. This valid since the aerodynamic mesh is evaluated only at the midpoint in the spanwise direction.

The used transfer scheme satisfies the requirements of being consistent and conservative [34]. *Consistency* requirement, is that the resultant nodal forces and moments are the same of forces and moments resulted from the pressure distribution for each element. *Conservativeness* requirement, defined as the virtual work resulted by the load vector over virtual displacements of the structural model, equals the work performed by pressure filed on the aerodynamic mesh.

Consistency is satisfied by the construction of the scheme, where the nodal forces and moments are defined to be the equivalent resultants from the traction field.

For *Conservativeness* to be satisfied, the virtual work done on the structural mesh should equal the virtual work done on the aerodynamic mesh. To demonstrate that, we write the equation of virtual work performed by structural mesh:

$$\delta W_s = \sum_{i=1}^2 (\mathbf{F}_{s,i} \cdot \delta \mathbf{d}_{s,i} + \mathbf{M}_{s,i} \cdot \delta \boldsymbol{\theta}_{s,i}) \quad (5.10)$$

By substitute Eq. (5.7) & (5.8):

$$\delta W_s = \frac{1}{2} \sum_{i=1}^2 (\mathbf{T} \cdot \delta \mathbf{d}_{s,i} + \mathbf{r}_{cp,i} \times \mathbf{T} \cdot \delta \boldsymbol{\theta}_{s,i}) S. \quad (5.11)$$

On the other hand, the virtual work performed by aerodynamic mesh, we have:

$$\delta W_a = \int_{panel} \mathbf{T} \delta \mathbf{u}_a dS \quad (5.12)$$

From Eq. (5.9):

$$\delta W_a = \frac{1}{2} \sum_{i=1}^2 \int (\mathbf{T} \cdot \delta \mathbf{d}_{s,i} + \mathbf{T} \cdot \delta \boldsymbol{\theta}_{s,i} \times \mathbf{r}_i) dS. \quad (5.13)$$

Since \mathbf{T} , $\mathbf{d}_{s,i}$ and $\boldsymbol{\theta}_{s,i}$ are constant values, the integration returns as follows:

$$\delta W_a = \frac{1}{2} \sum_{i=1}^2 (\mathbf{T} \cdot \delta \mathbf{d}_{s,i} + \mathbf{T} \cdot \delta \boldsymbol{\theta}_{s,i} \times \mathbf{r}_{cp,i}) S. \quad (5.14)$$

Considering the algebra of vectors:

$$\begin{aligned} \mathbf{T} \cdot \delta \boldsymbol{\theta}_{s,i} \times \mathbf{r}_{cp,i} &= \delta \boldsymbol{\theta}_{s,i} \times \mathbf{r}_{cp,i} \cdot \mathbf{T} \\ &= \delta \boldsymbol{\theta}_{s,i} \cdot \mathbf{r}_{cp,i} \times \mathbf{T} = \mathbf{r}_{cp,i} \times \mathbf{T} \cdot \delta \boldsymbol{\theta}_{s,i} \end{aligned} \quad (5.15)$$

Now, with respect to Eq. (5.15), we can write $\delta W_s = \delta W_a$.

5.3.3.2 Aerostructural Optimization Mechanism

The aerostructural model is a combination of the aerodynamic and structural models described above. These two models are structured within the aerostructural system as shown in Fig. 5.6. The aerodynamic model receives the mesh and produces loads, whereas the structural model receives the aerodynamic loads and outputs displacements. This continues until the analysis converges.

The default iteration method in OAS is Gauss–Seidel fixed-point iterations which covers the multidisciplinary analysis (MDA). What distinguishes this iteration method

that each analysis is running using the most recent output from the previous analysis until returning a consistent set of state variables [58].

Fig. 5.6 shows the XDSM diagram [51] corresponding to aerostructural optimization. Here, the input and output vectors, x and y , respectively, vary depending on the design variables, objectives, and constraints selected.

The x vectors are design variables and the y vectors are states, where $*$ represents the values at the design optimum. The MDA is Gauss–Seidel iterations method, which controls the iterations through appropriate distribution of the state target information and, at the same time, checking the convergence of the system.

The optimization process is running with the numerical order. The process starts from (0), where an initial guess of the design variables, $x^{(0)}$, is passed to the optimizer. The optimizer passes the state target information to MDA. In this stage, (2), MDA passes the aerodynamic inputs and variables to the aerodynamic solver, which, in turn, outputs aerodynamic loads. Now, the outputted loads pass to the structural solver as inputs with other structural variables. Then, the structural solver outputs corresponding displacements. That means a new mesh (deformed mesh), this deformed mesh is the new input for the aerodynamic solver. These iterations continue until a convergence is achieved, returning the optimal solution x^* .

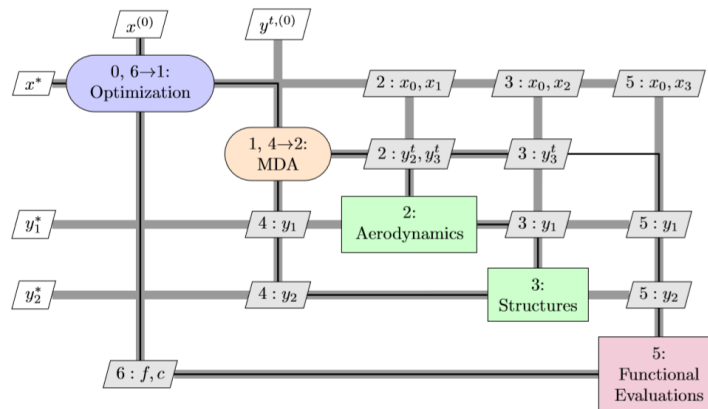


Figure 5.6. XDSM diagram for default aerostructural optimization [58].

It is worth mentioning that the objective and constraints are evaluated and an updated set of design variables is computed in each iteration.

5.3.4 The Optimizer

Currently, most MDO studies use gradient-based methods [56] as the optimizer used in this research. This optimizer is gradient-based algorithm called SLSQP. SLSQP is a sequential least squares programming algorithm [69] that evolved from the least squares solver of Lawson and Hanson. It uses a Quasi-Newton Hessian approximation and an L1-test function in the line search algorithm. SLSQP uses Sequential Least Squares Programming to minimize the objective function of variables and their bounds, and equality and inequality constraints. SLSQP Optimizer moves from a starting point in the direction of the steepest slope. Similar to Newton's Method, a sequence of one-dimensional optimization - involving quadratic approximation functions- are solved. These functions (Objective and constraints functions) are solved over and over until an optimum solution is found.

5.4 Setting up the Aerostructural Optimization Problem

The objective of this thesis is to obtain an efficient wing design for sailplanes, which contributes to high performance in terms of range and endurance. Thus, this requires maximum glide ratio (CL/CD) for the range, and minimum ($CD/CL^{1.5}$) with minimum weight for sink speed, as mentioned in the performance section. Therefore, we have objectives from two disciplines: glide ratio in aerodynamic and weight – precisely, the structural wing weight - in structure. Given that these two disciplines are deeply intersected, the multidisciplinary design optimization approach is selected to do the aerostructural analysis and optimization.

The objective function of the aerostructural optimization is the *sink speed*, Eq. (3.10). It corresponds to minimize the weight and maximizing the glide ratio (CL/CD). To do that, an objective function component for sink speed was added to OAS code.

The aerostructural optimization for the sailplane wing is as following:

$$\begin{aligned} & \text{Minimize} && \sqrt{\frac{2W}{\rho s}} \frac{C_D}{C_L^{1.5}} \\ & \text{w.r.t.} && \text{Twist, angle of attack, Taper, Beam Thickness} \\ & \text{subject to} && L = W, \text{ Stress failure} \end{aligned} \quad (5.16)$$

Where; W is the sailplane weight, C_D : Drag coefficient, C_L : Lift coefficient, S : reference area, ρ : is air density and L : lift force.

5.5 Case Study of a Sailplane Wing

For the case study, an initial wing design is selected as shown Table 5.1 and Fig. 5.7. This wing represents the baseline design for the aerostructural optimization.

Table 5.1. Initial sailplane wing specifications.

| <i>Specification</i> | <i>Value/Description</i> | <i>Unit</i> |
|----------------------|--------------------------|----------------|
| Planform | Rectangular | - |
| Root chord | 0.7 | m |
| Span | 15 | m |
| Reference Area | 10.5 | m ² |
| Aspect ratio | 21.4 | - |
| Twist | 0 | degree |
| Taper Ratio | 1 | - |
| Airfoil | NACA 63 ₃ 618 | - |

This wing has simple geometry, in order to capture any changes will take place after the optimization. However, the span is fixed on 15 and the initial aspect ratio is 21.4.

As explained in section 3.2, to minimize the aerostructural objective function i.e. sink speed, a set of design variables were selected in association with two main constraints.

These design variables are:

- *Taper Ratio*: it varies within the range of 0.3 to 1.
- *Twist Angle*: five control points over the wing span were set to examine spanwise twist distribution, the angle at each control point ranges from -10 to 15 degree.
- *Beam Thickness*: five control points were set in spanwise direction. The thickness is bounded in the range 0.001 to 0.1 m.
- *Angle of attack*: it varies within the range -10 to 10 degree.

Thus, the aerostructural optimization problem has twelve design variables in total. In addition to two constraints:

- *Lift equals weight*: means that the wing should produce lift that is able to carry the sailplane weight. Also, it is worth mentioning that this constraint is used to derive the objective function relation in the first place.
- *Failure constraint*, which is the aggregated spar failure using a Kreisselmeier-Steinhauser (KS) function [70] [71].

For the aerodynamic model, the flight conditions of a cross-country flight used, Table 5.2. At the same time, the mechanical properties used for structural model is based on Aluminum 7075, Table 5.2.

Table 5.2. Aerodynamic and structural models specifications.

| <i>Model</i> | <i>Parameter</i> | <i>Value</i> | <i>Unit</i> |
|--------------|------------------|--------------|-------------------|
| Aerodynamic | Speed | 25 | m/s |
| | Height | 1,000 | m |
| | Angle of attack | 3 | degree |
| | Air Density | 1.112 | Kg/m ³ |
| Structure | Young modulus | 71.7 | GPa |
| | Shear Modulus | 26.9 | GPa |
| | Material Density | 2810 | Kg/m ³ |

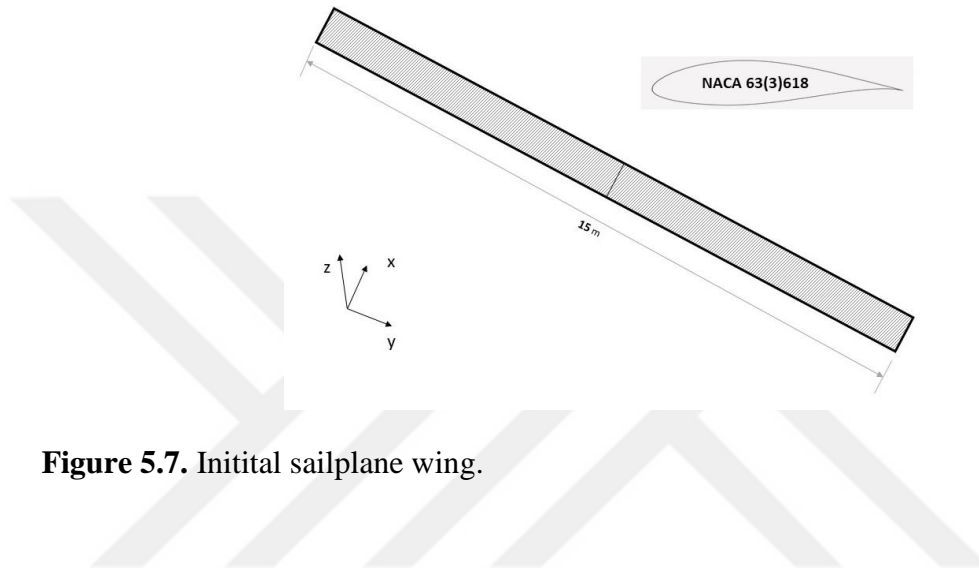


Figure 5.7. Initial sailplane wing.

CHAPTER 6

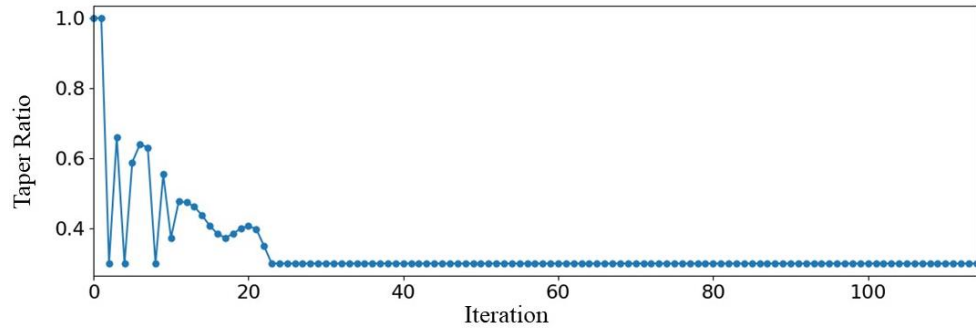
THE RESULTS

6.1 Introduction

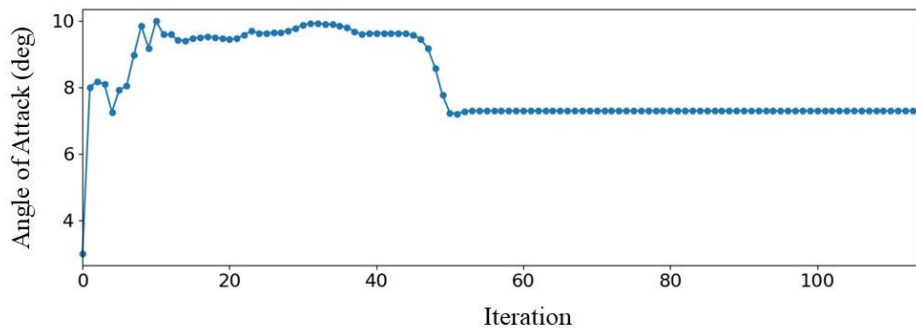
Given that all the conditions for aerostructural problem have been set up in sections 5.4 and 5.5, an aerostructural optimization for the case study wing is all set to be ran using OAS code. By running the case study problem using the OAS code, the system converged after 59 iterations and 115 function evaluations. The results of the aerostructural optimization show that the objective was achieved, where the sink speed was reduced about 37% less than the initial value. This reduction is resultant of increasing the glide ratio about 25% and reducing the structural weight to more than 50% of the initial value. Equally important, the two constraints also have been fulfilled in the optimization. The following sections highlight the results of each discipline.

6.2 Aerodynamic Optimization Findings

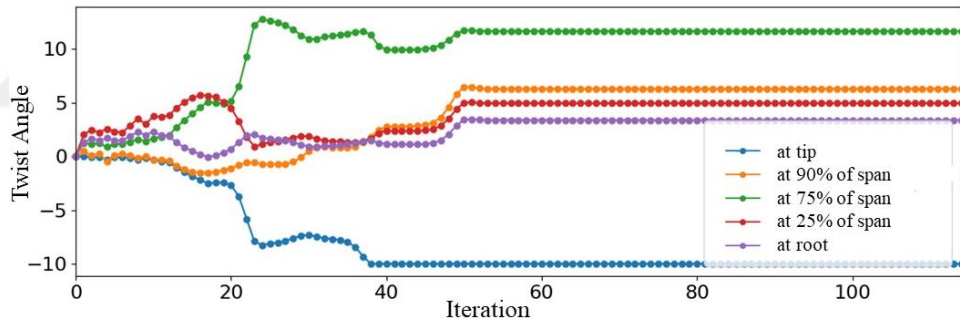
The results from the aerodynamic aspect of the optimization showed that the optimizer has managed to obtain elliptical lift distribution and to reduce the drag coefficient. Also, one can notice that the taper ratio has been reduced from 1 to 0.3 (Fig. 6.1-a). For the angle of attack, the optimizer tried firstly to increase it to nearly 10 degrees, then it decreased the angle to 7.3 degrees (Fig. 6.1-b). On the other hand, the variation of twist angle was different based on the location over the span. The initial values of the angles of the all five locations over the span were zero. After the optimization, the greatest values can be found at the tip with -10 degrees and at $\frac{3}{4}$ of the span with 11.64 degrees (Fig. 6.1-c).



a. Taper ratio during the optimization process.



b. Angle of attack during the optimization process.



c. Twist angles for the five control points along the span during the optimization process.

Figure 6.1. Aerodynamic design variables during the optimization process.

6.3 Structural Optimization Findings

On the structural aspect of the optimization, one can notice the decreasing in structural wing weight, which was decreased more than 50% of the initial value. This reduction basically a result of decreasing the thickness over the tubular beam. As mentioned, the thicknesses of five control points over the span were selected as design variables. The thicknesses of those five control points were changed differently. Where, the thickness increased a bit at the root, while it decreased significantly at the tip. Fig. 6.2 shows the variation of thickness values during the optimization problem. Regarding the structural

constraint, the optimizer managed to satisfy the structural constraint in order to avoid any structural failure during the thickness reduction process.

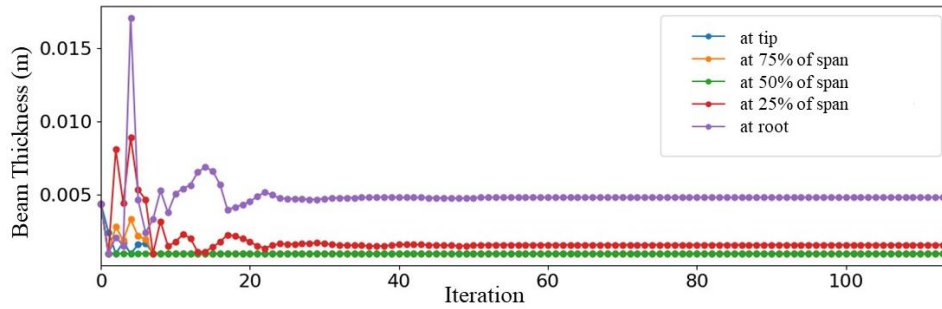
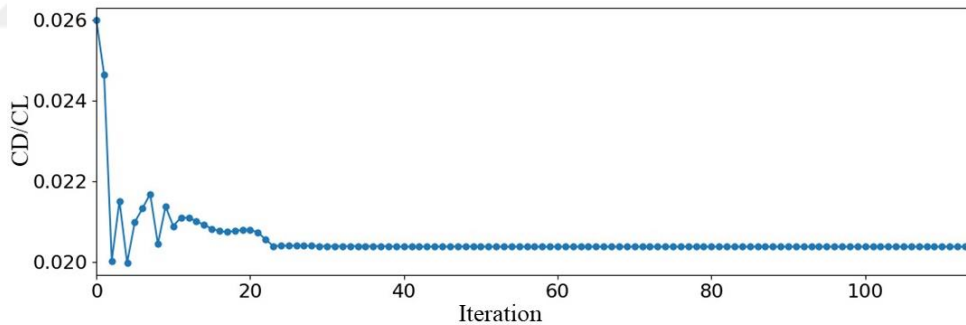


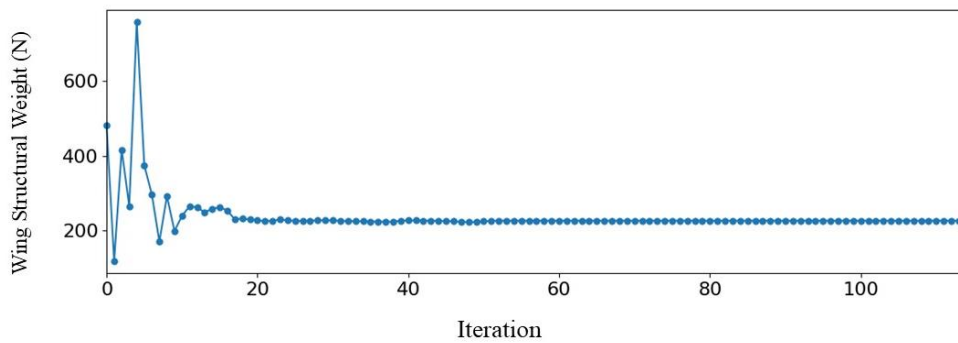
Figure 6.2. Beam thickness for the five control points along the span.

6.4 Aero-structural Optimization Findings

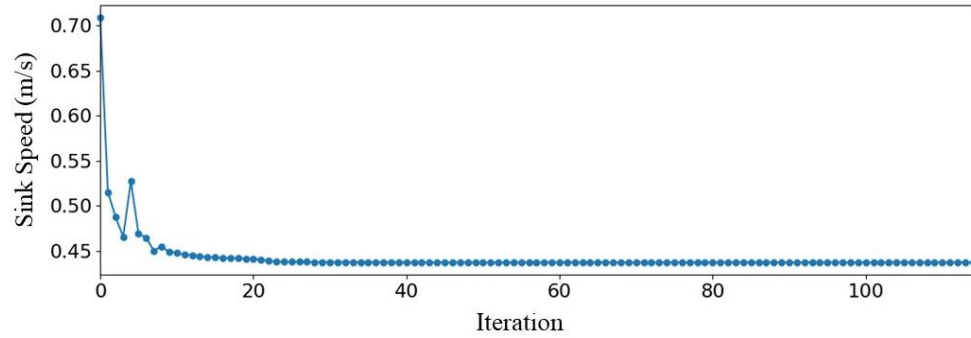
The results from aerostructural optimization showed that the sink speed, the aerostructural objective, have been minimized. Where the optimizer minimized it from 0.7 m/s to 0.44 m/s. The sink speed minimization was associated with the minimization of drag and structural weight (Fig. 6.3). The aerostructural optimization results are showed in Table 6.1. In addition, Fig. 6.4 illustrate the evolution of the optimization process on the wing through four main points of the iterations.



a. Inverted glide ratio variation during the Optimization.



b. Wing structural weight variation during the Optimization.



c. Sink Speed (the objective) variation during the Optimization.

Figure 6.3. The objective function during the optimization process.

Table 6.1. Aerostructural optimization results

| <i>Function/Variable</i> | <i>Initial Value</i> | <i>Optimized Value</i> | <i>Lower bound</i> | <i>Upper Bound</i> | <i>Unit</i> |
|--------------------------|----------------------|------------------------|--------------------|--------------------|-------------|
| Sink Speed | 0.7 | 0.44 | - | - | m/s |
| Glide ratio CL/CD | 38.46 | 49 | - | - | - |
| Aspect ratio | 21.4 | 32.65 | - | - | - |
| Angle of attack | 3 | 7.3 | -10 | 10 | deg |
| Taper ratio | 1 | 0.3 | 0.3 | 1 | - |
| Twist (root) | 0 | 3.37 | -10 | 15 | deg |
| Twist (25% span) | 0 | 4.94 | -10 | 15 | deg |
| Twist (75% span) | 0 | 11.64 | -10 | 15 | deg |
| Twist (90% span) | 0 | 6.28 | -10 | 15 | deg |
| Twist (tip) | 0 | -10 | -10 | 15 | deg |
| Thickness (root) | 0.0044 | 0.0048 | 0.001 | 0.1 | m |
| Thickness (25% span) | 0.0044 | 0.003 | 0.001 | 0.1 | m |
| Thickness (50% span) | 0.0044 | 0.0016 | 0.001 | 0.1 | m |
| Thickness (75% span) | 0.0044 | 0.0011 | 0.001 | 0.1 | m |
| Thickness (tip) | 0.0044 | 0.001 | 0.001 | 0.1 | m |
| Wing struct weight | 49 | 23.16 | - | - | kg |

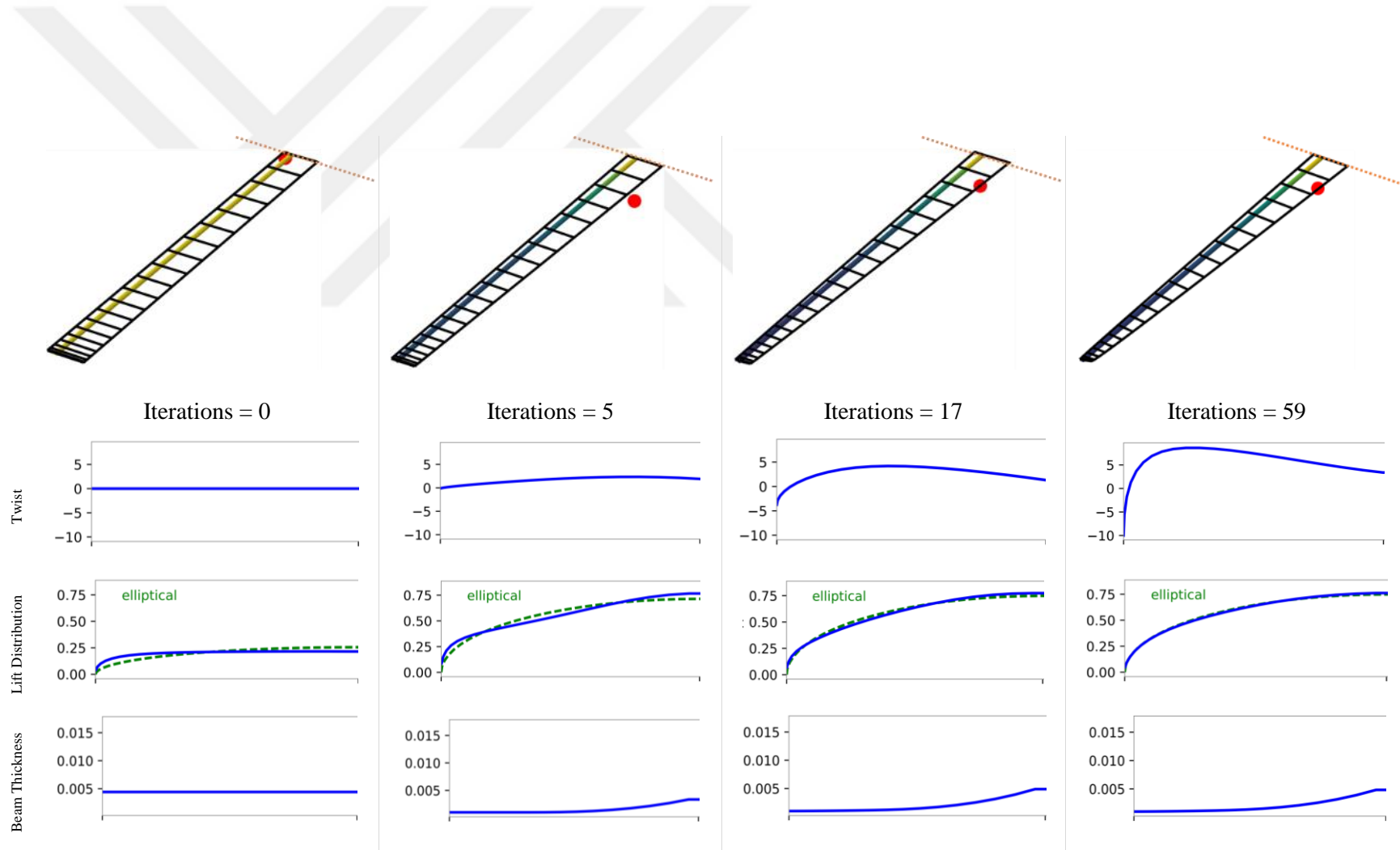


Figure 6.4. Aerostructural optimization at four iteration points, starting from the initial design (on the left) and ending at the optimized design (on the right).

CHAPTER 7

DISCUSSION

7.1 Aerodynamic Optimization

For the aerodynamic aspect of the optimization, the optimizer worked on minimizing CD through ensuring an elliptical lift distribution over the wing, where the elliptical distribution is precisely the optimum distribution for minimum induced drag [47]. For the purpose of that, the optimizer has changed the taper ratio from 1 to 0.3 and associated this with varying the twist angle distribution along the span. As a matter of fact, changing the geometry from rectangular to tapered is quite interesting finding, where in reality, tapered wing is the planform found most frequently on gliders [46].

On the other hand, even though tapering a wing planform gives a number of significant aerodynamic and structural advantages, it also could be challenging regarding wing tip stall unless an appropriate twist distribution is considered. For that reason, after settling on the taper ratio, the optimizer kept changing the twist distribution over the span in a pattern that achieves an elliptical lift distribution, and thereby, minimizing the induced drag. Moreover, the variation of twist was associated with changing the angle of attack as well. Where, the optimizer increased the angle of attack about 4 degrees to meet the required amount of lift. Another interesting finding, the optimizer used the twist angle, and benefited from the cosine-spaced mesh, to bend up the last panel at the tip, forming a winglet-like shape (Fig. 7.1). At the early preliminary design, this could be a good indication to consider at following design stages of the wing.

However, the benefits that a winglet provides to the performance and handling qualities are being repeatedly proved [72], [73]. In summary, the optimizer has managed to increase CL/CD ratio about 27% more than the initial value of the baseline wing with fulfilling the *lift-equals-weight* constraint.

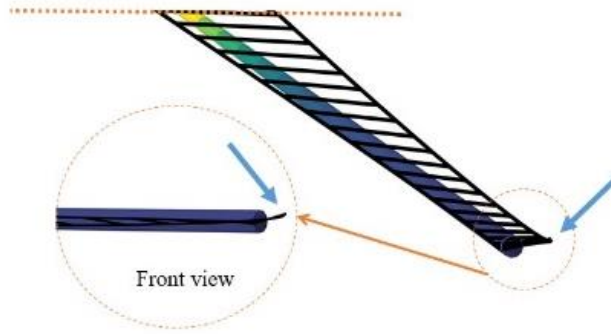


Figure 7.1. Bent-up tip of the optimized wing.

7.2 Structural Optimization

For the structural aspect of optimization, the objective of this optimization is clearly the weight minimization. Given that the simplicity of the structural model, the main factor in weight minimization is reducing thickness over span with fulfilling the failure constraint. The initial thickness of the structure model is 0.0044 m, which is constant over the span. This thickness distribution gives an initial wing weight of 49 kg. As the structural weight is affected by the lift distribution, the thickness was varying throughout the optimization unevenly from the root to the tip. The largest thickness value appeared at the root as expected, where the bending moment is the highest. Then, it was gradually decreasing from root to tip as Fig. 6.4 illustrates. As a result, the wing structural weight has been reduced more than 50% with fulfilling the failure constraint.

7.3 Aero-structural Optimization

As the objective of the aero-structural optimization is minimization of sink speed, this implies minimization of two sub-objectives, the drag and the weight. However, the interdependencies among the aerodynamic and structural design variables drive the optimizer to vary the design variables in a consistent way, which leads to minimum objective function and, at the same time, satisfies the constraints.

By observing the optimization process, one can notice that the optimizer reduced the taper ratio aiming to increase the aspect ratio, where increasing aspect ratio leads to less induced drag. At the same time, reducing taper ratio leads to reduce the wing area moving outboard from the root. Correspondingly, this reduces the lift as moving to tip. Thus, unloading the wing as moving to tip yields a consistent decrease in beam thickness i.e. less structural weight.

Similarly for twist, the optimizer changed the twist distribution in a consistent way aiming to have elliptical lift distribution, which evidently leads to minimum induced drag and less structural weight. This is more obvious at the wing tip, where the tip is twisted down (washout). Wings often have less incidence at the tip than the root (washout) to reduce structural weight and improve stalling characteristics [74]. As (Fig. 6.1-c) shows, the optimizer twisted down the tip by 10 degrees, leading to decrease the local angle of attack at the tip, which means less local lift at the tip. As a consequence, lesser beam thickness at the tip would be enough to provide adequate stiffness. This, in turn, leads to reduce the structural weight.

Using range equation, Eq. (3.8) and endurance equation, Eq. (3.9), one can capture the improvements of the optimized wing on the sailplane performance in cross-country flight, Table 6.1. For the range, the optimized wing showed an improvement of 27% - i.e. additional 10.5 km - better than the baseline wing. Likewise, for the endurance, the optimized wing increased the endurance by 37% - i.e. additional 14 minutes – better than the baseline wing.

CHAPTER 8

CONCLUSION

8.1 Thesis Conclusion

This study has introduced multidisciplinary design optimization for sailplane wings. Two main disciplines were included, aerodynamic and structure. The design optimization has been running using aerostructural design optimization tool, called OAS. This tool has been adapted to suit the design objectives of sailplane wings.

The proposed methodology used in this research was to start from simple wing design, yet relatively with good design configurations. Then, running the coupled aerostructural optimization on the baseline design. For the aerodynamic analysis, Vortex Lattice Method has been used to calculate the aerodynamic loads and moment. On the other hand, finite element analysis has been used for the structural analysis, stresses, and deformations. The objective of the aerostructural design optimization was to obtain a wing design with better performance characteristics.

The findings of the aerostructural optimization indicated improvements in the performance characteristic of the optimized wing. All the changes that optimizer made on the baseline, supported, in a way, the theories and good practices in wing design for both disciplines; aerodynamic and structural. For example, increasing the aspect ratio, the washout twist distribution, and beam thickness distribution. Taken these findings into consideration, will offer interesting design perspectives in advanced design phases.

8.2 Future Work

The future work of this study can be continued in many aspects. For structure, we can represent the wing structurally using the effective wing box, and consider the use of composite materials. For aerodynamic, using another aerodynamic solver like panel method and comparing the results, also, having finer aerodynamic discretization. For aerostructural optimization, it is preferable to expand the work through using a variety of optimization algorithms (gradient-based and gradient free) and MDO architectures and make comparison among the results.

Also, we can run high fidelity analysis and optimization based on the findings of our study and by using other tools. Another important thing to consider is involving more design and flight conditions to the aerostructural optimization problem, like different speeds, altitudes, flight phases (climb, thermalling ...etc.).



REFERENCES

- [1] AIAA. American Institute of Aeronautics and Astronautics. (1999). White Paper on Industrial Experience with MDO. *Symposium on Multidisciplinary Analysis and Optimization*. 426.
- [2] J. Mariens. (2012). Wing Shape Multidisciplinary Design Optimization. M.SC. Thesis. TU Delft.
- [3] B. Jorge. (2011). Aero-Structural Optimization of Sailplane Wings. M.SC. Thesis. Instituto Superior Técnico Sintra, Portugal.
- [4] M. Tomac. (2011). Adaptive-fidelity CFD for Predicting Flying Qualities in Preliminary Aircraft Design. M.SC. Thesis KTH.
- [5] A. Welch. (1980). The Story of Gliding. Second Edition.
- [6] M. Maughmer. (2003). The Evolution of Sailplane Wing Design. in *AIAA International Air and Space Symposium and Exposition: The Next 100 Years*.
- [7] Perlan II Project. 2018. Available: <http://www.perlanproject.org/aircraft>. [Accessed: 29-Apr-2018].
- [8] Otto Lilienthal Museum. 2018. Available: <http://www.lilienthal-museum.de/olma/ehome.htm>. [Accessed: 29-Apr-2018].
- [9] The Wright Brothers. 2018. Available: <https://airandspace.si.edu/exhibitions/wright-brothers/online/fly/1902/gliders.cfm>. [Accessed: 29-Apr-2018].
- [10] Performance Increase Possibilities of Gliders. 2018. Available: <https://www.dg-flugzeugbau.de/en/library>. [Accessed: 29-Apr-2018].
- [11] The Fafnir. 2018. Available: <https://scalesoaring.co.uk/VINTAGE/Documentation/Fafnir/Lippisch.html>. [Accessed: 29-Apr-2018].
- [12] A short history of gliding. 2018. Available: <http://www.saintex.fr/histoirevolavoile.htm>. [Accessed: 29-Apr-2018].
- [13] Metroliner / SB 10 / ATRA. 2018. Available: <https://forum.luftfahrtclubbraunschweig.de/viewtopic.php?t=4383>. [Accessed: 29-Apr-2018].
- [14] C. V Pilcher. (2011). Preliminary Sailplane Design Using MDO And Multi-Fidelity Analysis. M.SC. Thesis. Ryerson University.
- [15] L. Schmit. (1960). Structural Design by Systematic Synthesis. in *2nd Conference on Electronic Computation*.
- [16] L. A. Schmit and W. A. Thornton. (1964). Synthesis of an airfoil at supersonic mach number. *National Aeronautics and Space Administration*.
- [17] L. A. Schmit. (1981). Structural synthesis - Its genesis and development. *AIAA J.*, **19**, 1249–1263.

- [18] L. A. Schmit. (1984). Structural Synthesis - Precursor and Catalyst. *Recent Experiences in Multidisciplinary Analysis and Optimization*.
- [19] R. T. Haftka. (1986). Automated Procedure for Design of Wing Structures to Satisfy Strength and Flutter Requirements. *Computers & Structures*, **24**, 799 - 808.
- [20] R. T. Haftka, J. H. S. Jr., F. W. Barton, and S. C. Dixon. (1975). Comparison of Two Types of Structural Optimization Procedures for Flutter Requirements. *AIAA J.*, **13**, 1333–1339.
- [21] R. T. Haftka. (1977). Optimization of Flexible Wing Structures Subject to Strength and Induced Drag Constraints. *AIAA J.*, **15**, 1101–1106.
- [22] R. T. Haftka and C. P. Shore. (1979). Approximate Methods for Combined Thermal/Structural Design. *NASA Technical Paper*. 1428.
- [23] R. T. Haftka. (1977). Optimization of flexible wing structures subject to strength and induced drag constraints. *AIAA J.*, **15**, 1101–1106.
- [24] T. Mcgeer. (1984). Wing design for minimum drag with practical constraints. *J. Aircr. - J Aircr.*, **21**, 879–886.
- [25] B. Grossman, Z. Gurdal, R. T. Haftka, G. J. Strauch, And W. M. Eppard. (1988). Integrated Aerodynamic/Structural Design of a Sailplane Wing. *J. Aircr.*, **25**, no. 9, 855–860.
- [26] B. Grossman, R. T. Haftka, J. Sobieszczanski - Sobieski, P.-J. Kao, D. M. Polen, and M. Rais-Rohani. (1990). Integrated aerodynamic-structural design of a transport wing. *J. Aircr.*, **27**, 1050–1056.
- [27] S. Wakayama and I. Kroo. (1995). Subsonic wing planform design using multidisciplinary optimization. *J. Aircr.*, **32**, 746–753.
- [28] C. J. Borland, J. R. Benton, P. D. Frank, T. J. Kao, R. A. Mastro, and J. F. M. Barthelemy. (1994). Multidisciplinary design optimization of a commercial aircraft wing – an exploratory study. in *AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 1358–1368.
- [29] A. Chattopadhyay and N. Pagalapati. (1995). A multidisciplinary optimization using semi-analytical sensitivity analysis procedure and multilevel decomposition. *Comput. Math. with Appl.*, **29**, 55–66.
- [30] M. Baker and J. Giesing. (1995). A practical approach to MDO and its application to an HSCT aircraft. in *Aircraft Engineering, Technology, and Operations Congress Los Angeles, CA, U.S.A.*
- [31] V. Michelle Manning. (1999). Large-scale design of supersonic aircraft via collaborative optimization. 1st Edition.
- [32] J. Reuther, J. Alonso, J. Martins, and S. Smith. (1999). A coupled aero-structural optimization method for complete aircraft configurations. *37th Aerosp. Sci. Meet. Exhib.*
- [33] K. Maute, M. Nikbay, and C. Farhat. (2001). Coupled analytical sensitivity analysis and optimization of three-dimensional nonlinear aeroelastic systems. *AIAA J.*, **39**, 2051–2061,.
- [34] J. R. R. A. Martins. (2005). a Coupled-Adjoint Method for High-Fidelity Aero-Structural Optimization. *Optimization and Engineering*, **6**, 33-62.

- [35] K. Maute and M. Allen. (2004). Conceptual design of aeroelastic structures by topology optimization. *Struct. Multidiscip. Optim.*, **27**, 27–42.
- [36] M. Barcelos, H. Bavestrello, and K. Maute. (2006). A Schur–Newton–Krylov solver for steady-state aeroelastic analysis and design sensitivity analysis. *Comput. Methods Appl. Mech. Eng.*, **195**, 2050–2069.
- [37] M. Barcelos and K. Maute. (2008). Aeroelastic design optimization for laminar and turbulent flows. *Comput. Methods Appl. Mech. Eng.*, **197**, 1813–1832.
- [38] C. V Pilcher. (2011). Preliminary Sailplane Design Using MDO And Multi-Fidelity Analysis. M.SC. Thesis. Ryerson University.
- [39] J. E. K. Hoogervorst. (2015). Wing aerostructural optimization using the Individual Discipline Feasible architecture.
- [40] J. R. R. A. Martins and A. B. Lambe. (2013). Multidisciplinary Design Optimization: A Survey of Architectures. *AIAA J.*, **51**, 2049–2075.
- [41] USA Soaring, History of gliding & soaring. 2004. Available: [http://www.ssa.org/files/member/br soaring history v5 04.pdf](http://www.ssa.org/files/member/br%20soaring%20history%20v5%2004.pdf). [Accessed: 29-Apr-2018].
- [42] Qatar Foundation, Arab Science : A Journy of Innovation. 2009. Available: <http://www.grouporigin.com>. [Accessed: 29-Apr-2018].
- [43] A. Firas, Pioneers of Aviation. 2013. Available: http://lostislamichistory.com/pioneers_of_aviation/. [Accessed: 29-Apr-2018].
- [44] Gliding Federation of Australia. (2001). Basic Gliding Knowledge. 1st Edition, Australia.
- [45] S. Frati. (1946). The glider, 1st Edition, Milano.
- [46] Federal Aviation Administration. (2013). Glider Flying Handbook. 1st Edition, U.S.
- [47] F. Thomas and J. Milgram (1999), Fundamentals of Sailplane Design. 1st Edition. Germany.
- [48] S. Gudmundsson. (2014). General Aviation Aircraft Design. Applied methods and procedures. 1st Edidtion. U.S.
- [49] J. Martins. (2012). A Short Course on Multidisciplinary Design Optimization. 1st Edition. U.S.
- [50] A. J. Keane and P. B. Nair. (2005). Computational Approaches for Aerosp. Design: The Pursuit of Excellence. 1st Edition, U.S. WILEY.
- [51] A. Lambe and J. Martins. (2012). Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes. *Struct. Multidiscip. Optim.*, **46**, 273–284.
- [52] E. J. Cramer, J. E. Dennis, P. D. Frank, G. R. Shubin, and R. M. Lewis. (1994). Problem Formulation for Multidisciplinary Optimization. *SIAM J. Optim.*, **4**, 754–776.
- [53] C. L. Bloebaum, P. Hajela, and J. Sobieszczanski-Sobieski. (1992). “Non-hierarchical system decomposition in structural optimization,” *Eng. Optim. - ENG Optim.*, **19**, 171–186.

- [54] R. D. Braun and I. M. Kroo. (1996). Development and Application of the Collaborative Optimization Architecture in a Multidisciplinary Design Environment. *Multidiscip. Des. Optim. State Art*, **80**.
- [55] J. Sobieszczanski-Sobieski, J. S. Agte, and R. R. Sandusky. (1998). Bi-Level (BLISS) Integrated System Synthesis.
- [56] M. H. NM Alexandrov. (1997). Multidisciplinary design optimization: State of the art. 1st Edition.
- [57] A. Salas and J. Townsend. (1998). Framework requirements for MDO application development,” in *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*.
- [58] J. P. Jasa, J. T. Hwang, and J. R. R. A. Martins. (2018). Open-source coupled aerostructural optimization using Python. *Structural and Multidisciplinary Optimization*, **57**, 1815-1827.
- [59] J. T. Hwang, D. Y. Lee, J. W. Cutler, and J. R. R. A. Martins. (2014). Large-Scale Multidisciplinary Optimization of a Small Satellite’s Design and Operation. *J. Spacecr. Rockets*, **51**, 1648–1663.
- [60] J. S. Gray, T. A. Hearn, K. T. Moore, J. Hwang, J. Martins, and A. Ning. (2014). Automatic Evaluation of Multidisciplinary Derivatives Using a Graph-Based Problem Formulation in OpenMDAO. in *15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*.
- [61] J. Hwang and J. Martins. (2016). Allocation-mission-design optimization of next-generation aircraft using a parallel computational framework. in *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*.
- [62] R. D. Falck, J. Chin, S. L. Schnulo, J. M. Burt, and J. S. Gray. (2017). Trajectory Optimization of Electric Aircraft Subject to Subsystem Thermal Constraints. *18th AIAA/ISSMO Multidiscip. Anal. Optim. Conf.*, 1–16.
- [63] N. Bons, X. He, C. A. Mader, and J. Martins. (2017). Multimodality in Aerodynamic Wing Design Optimization,” in *35th AIAA Applied Aerodynamics Conference*.
- [64] L. W. Cook, J. P. Jarrett, and K. E. Willcox. (2017). Extending Horsetail Matching for Optimization Under Probabilistic, Interval, and Mixed Uncertainties,” *AIAA J.*, 1–13.
- [65] S. Friedman, S. F. Ghoreishi, and D. L. Allaire. (2017). Quantifying the Impact of Different Model Discrepancy Formulations in Coupled Multidisciplinary Systems,” *19th AIAA Non-Deterministic Approaches Conf.*, 1–9.
- [66] John.J.Bertin. (2001). Aerodynamics for Engineers. 4th Edition.
- [67] J. Anderson Jr. (1985). Fundamentals of Aerodynamics. 3rd Edition.
- [68] D. P. Raymer. (1992). Aircraft Design: A Conceptual Approach. 5th Edition. AIAA.
- [69] D. Kraft. (1988). A software package for sequential quadratic programming.
- [70] G. Kreisselmeier and R. Steinhauser. (1980). Systematic Control Design By Optimizing a Vector Performance Index. *Comput. Aided Des. Control Syst.*, **12**,

113–117.

- [71] A. B. Lambe, G. J. Kennedy, and J. R. R. A. Martins. (2017). An evaluation of constraint aggregation strategies for wing box mass minimization. *Struct. Multidiscip. Optim.*, **55**, 257–277.
- [72] P. C. Masak. (1993). Design of Winglets for Sailplanes. *Soaring*, 21–27.
- [73] M. D. Maughmer. (2003). The Design of Winglets for High-Performance. *J. Aircr.*, **40**, 1099–1106.
- [74] I. Kroo. (2007). *Applied Aerodynamics*. 5th Edition.



APPENDIX A: MODIFIED SECTIONS OF OAS CODE

In this appendix, the aerostructural run-file of the code is presented in addition to the modified sections in the main code, which are *run_classes.py* and *functional.py*. The rest of the code is the same of the original and available on: <https://github.com/mdolab/OpenAeroStruct>

A1 - The Run-file Script for the Aerostructural Optimization

```
# Set problem type
prob_dict = {'type' : 'aerostruct',
            'optimizer' : 'SLSQP',           # default optimizer
            'force_fd' : False,             # if true, we FD over the whole model
            'with_viscous' : True,          # if true, compute viscous drag
            'Re' : 1.e6,                    # Reynolds number
            'reynolds_length' : 0.5,       # characteristic Reynolds length
            'alpha' : 3.,                   # [degrees] angle of attack
            'M' : 0.075,                    # Mach number at cruise
            'rho' : 1.112,                  # [kg/m^3] air density at 1,000 m
            'a' : 336.6,                    # [m/s] speed of sound at 1,000 m
            'g' : 9.8,                      # [m/s^2] acceleration due to gravity

            # Aircraft properties
            'W0' : 300 }                    # [kg] weight of the sailplane without
                                           # wing structural weight

# Instantiate problem and add default surface
OAS_prob = OASProblem(prob_dict)

# Create a dictionary to store options about the surface
surf_dict = {
    # Wing definition
    'name' : 'wing',                       # name of the surface
    'num_x' : 2,                             # number of chordwise points
    'num_y' : 41,                             # number of spanwise points
    'span_cos_spacing' : 1.,                 # 0 for uniform spanwise panels
                                           # 1 for cosine-spaced panels
                                           # any value between 0 and 1 for
                                           # a mixed spacing
    'chord_cos_spacing' : 0.,               # 0 for uniform chordwise panels
                                           # 1 for cosine-spaced panels
                                           # any value between 0 and 1 for
                                           # a mixed spacing
    'wing_type' : 'rect',                   # initial shape of the wing
                                           # either 'CRM' or 'rect'
                                           # 'CRM' can have different options
                                           # after it, such as 'CRM:alpha_2.75'
                                           # for the CRM shape at alpha=2.75
    'symmetry' : True,                      # if true, model one half of wing
                                           # reflected across the plane y = 0
    'S_ref_type' : 'wetted',                # how we compute the wing area,
                                           # can be 'wetted' or 'projected'

    # Simple Geometric Variables
    'span' : 15.,                            # full wingspan, even for symmetric cases
    'root_chord' : 0.7,                      # root chord
    'dihedral' : 0.,                         # wing dihedral angle in degrees
                                           # positive is upward
    'sweep' : 0.,                            # wing sweep angle in degrees
                                           # positive sweeps back
    'taper' : 1.,                           # taper ratio; 1. is uniform chord
    'S_ref' : None,                          # [m^2] area of the lifting surface

    # Aerodynamic performance of the lifting surface at
```

```

# an angle of attack of 0 (alpha=0).
# These CL0 and CD0 values are added to the CL and CD
# obtained from aerodynamic analysis of the surface to get
# the total CL and CD.
# These CL0 and CD0 values do not vary wrt alpha.
'CL0' : 0.5,          # CL of the surface at alpha=0 (NACA
                    # 63(3)-618)
'CD0' : 0.01,        # CD of the surface at alpha=0 (NACA
                    # 63(3)-618)

# Airfoil properties for viscous drag calculation
'k_lam' : 0.05,      # percentage of chord with laminar
                    # flow, used for viscous drag
't_over_c' : 0.126,  # thickness over chord ratio (NACA 63(3)-
                    # 618)
'c_max_t' : .243,    # thickness at 34.7% for (NACA 63(3)-618)

# Structural values are based on aluminum 7075
'E' : 71.7e9,        # [Pa] Young's modulus of the spar
'G' : 26.9e9,        # [Pa] shear modulus of the spar
'yield' : 572.e6 / 1.5, # [Pa] yield stress divided by 1.5 for
                    # limiting case
'mrho' : 2.81e3,     # [kg/m^3] material density
'fem_origin' : 0.243, # normalized chordwise location of the
                    # spar
'loads' : None,      # [N] allow the user to input loads
'disp' : None,       # [m] nodal displacements of the FEM model

}

# Add the specified wing surface to the problem
OAS_prob.add_surface(surf_dict)

# Add design variables, constraint, and objective on the problem
OAS_prob.add_desvar('alpha', lower=-10., upper=10.)
OAS_prob.add_constraint('L_equals_W', equals=0.)
OAS_prob.add_objective('sink_speed', scaler=1e2)

# Setup problem and add design variables, constraint, and objective
OAS_prob.add_desvar('wing.twist_cp', lower=-10., upper=15.)
OAS_prob.add_desvar('wing.thickness_cp', lower=0.001, upper=0.1, scaler=1e2)
OAS_prob.add_desvar('wing.taper', lower=0.3, upper= 1.0)
OAS_prob.add_constraint('wing_perf.failure', upper=0.)
OAS_prob.add_constraint('wing_perf.thickness_intersects', upper=0.)
OAS_prob.setup()

st = time()
# Actually run the problem
OAS_prob.run()

print("Time elapsed: {} secs".format(time() - st))
print("\nWing CL:", OAS_prob.prob['wing_perf.CL'])
print("Wing CD:", OAS_prob.prob['wing_perf.CD'])
print("\nStructural Weight:", OAS_prob.prob['wing_perf.structural_weight']/
      OAS_prob.prob_dict['g'], " kg")
CL = OAS_prob.prob['wing_perf.CL']
CD = OAS_prob.prob['wing_perf.CD']
Range = 1*(CL/CD) # Altitude difference is 1 Km.
print("\nRange      :", Range, " Km")
print ("Sink Speed: ", OAS_prob.prob['sink_speed']," m/s")

```

A2 - Script of Modified *functionals.py*

```
from __future__ import division, print_function
import numpy as np

from openmdao.api import Component, Group

class SinkSpeedFunction(Component): """
    Computes the sink speed using the computed CL, CD, weight and other parameters.

    Note that we add information from each lifting surface.

    Parameters
    -----
    CL : float
        Total coefficient of lift (CL) for the lifting surface.
    CD : float
        Total coefficient of drag (CD) for the lifting surface.
    weight : float
        Total weight of the structural spar.

    Returns
    -----
    Sink Speed : float

    """

    def __init__(self, surfaces, prob_dict):
        super(SinkSpeedFunction, self).__init__()

        self.surfaces = surfaces
        self.prob_dict = prob_dict

        for surface in surfaces:
            name = surface['name']
            self.add_param(name+'S_ref', val=0.)
            name = surface['name']
            self.add_param(name+'structural_weight', val=0.)

        self.add_param('CL', val=0.)
        self.add_param('CD', val=0.)

        self.add_output('sink_speed', val=0.)

        self.deriv_options['type'] = 'cs'
        self.deriv_options['form'] = 'central'

    def solve_nonlinear(self, params, unknowns, resids):
        rho = self.prob_dict['rho']
        W0 = self.prob_dict['W0'] * self.prob_dict['g']

        Ws = 0.
        for surface in self.surfaces:
            name = surface['name']
            Ws += params[name+'structural_weight']

        for surface in self.surfaces:
            name = surface['name']
            S_ref = params[name+'S_ref']

        CL = params['CL']
        CD = params['CD']
        wing_load = np.sum(W0 + Ws)/S_ref
        unknowns['sink_speed'] = (CD/(CL**1.5))*(wing_load/(0.5*rho))**0.5
```

```

class FunctionalEquilibrium(Component):
    """
    Lift = weight constraint.

    Note that we add information from each lifting surface.

    Parameters
    -----
    L : float
        Total lift for the lifting surface.
    structural_weight : float
        Total weight of the structural spar.

    fuelburn : float
        Computed fuel burn in kg based on the Breguet range equation.

    Returns
    -----
    L_equals_W : float
        Equality constraint for lift = total weight. L_equals_W = 0 for the constraint
to be satisfied.
    total_weight : float
        Total weight of the entire aircraft, including W0, all structural weights,
        and fuel.

    """

    def __init__(self, surfaces, prob_dict):
        super(FunctionalEquilibrium, self).__init__()

        self.surfaces = surfaces
        self.prob_dict = prob_dict

        for surface in surfaces:
            name = surface['name']

            self.add_param(name+'L', val=0.)
            self.add_param(name+'structural_weight', val=0.)

            #self.add_param('fuelburn', val=0.) #removed fuelburn as param,
because it is not relevant in sailplane case [Ahmad]
            self.add_output('L_equals_W', val=0.)
            self.add_output('total_weight', val=0.)

            self.deriv_options['type'] = 'cs'
            self.deriv_options['form'] = 'central'

        def solve_nonlinear(self, params, unknowns, resids):
            structural_weight = 0.
            L = 0.
            W0 = self.prob_dict['W0'] * self.prob_dict['g']
            for surface in self.surfaces:
                name = surface['name']
                structural_weight += params[name+'structural_weight']
                L += params[name+'L']

            tot_weight = structural_weight + W0

            unknowns['total_weight'] = tot_weight
            unknowns['L_equals_W'] = (tot_weight - L) / tot_weight

class ComputeCG(Component):
    """
    Compute the center of gravity of the entire aircraft based on the inputted W0
    and its corresponding cg and the weighted sum of each surface's structural
    weight and location.

    Note that we add information from each lifting surface.

    Parameters
    -----
    nodes[ny, 3] : numpy array
        Flattened array with coordinates for each FEM node.
    structural_weight : float

```

```

        Total weight of the structural spar for a given surface.
cg_location[3] : numpy array
        Location of the structural spar's cg for a given surface.

total_weight : float
        Total weight of the entire aircraft, including W0, all structural weights,
        and fuel.
fuelburn : float
        Computed fuel burn in kg based on the Breguet range equation.

Returns
-----
cg[3] : numpy array
        The x, y, z coordinates of the center of gravity for the entire aircraft.
"""

def __init__(self, surfaces, prob_dict):
    super(ComputeCG, self).__init__()

    self.prob_dict = prob_dict
    self.surfaces = surfaces

    for surface in surfaces:
        name = surface['name']

        self.add_param(name+'nodes', val=0.)
        self.add_param(name+'structural_weight', val=0.)
        self.add_param(name+'cg_location', val=np.zeros((3), dtype=data_type))

    self.add_param('total_weight', val=0.)
    self.add_output('cg', val=np.zeros((3), dtype=complex))

    self.deriv_options['type'] = 'cs'
    self.deriv_options['form'] = 'central'

def solve_nonlinear(self, params, unknowns, resids):
    g = self.prob_dict['g']
    W0 = self.prob_dict['W0']
    W0_cg = W0 * self.prob_dict['cg'] * g

    spar_cg = 0.
    structural_weight = 0.
    for surface in self.surfaces:
        name = surface['name']
        spar_cg = params[name + 'cg_location'] * params[name +
'structural_weight']
        structural_weight += params[name + 'structural_weight']
    tot_weight = structural_weight + W0

    unknowns['cg'] = (W0_cg + spar_cg) / (params['total_weight'])

class ComputeCM(Component):
    """
    Compute the coefficient of moment (CM) for the entire aircraft.

    Parameters
    -----
    b_pts[nx-1, ny, 3] : numpy array
        Bound points for the horseshoe vortices, found along the 1/4 chord.
    widths[ny-1] : numpy array
        The spanwise widths of each individual panel.
    chords[ny] : numpy array
        The chordwise length of the entire airfoil following the camber line.
    S_ref : float
        The reference area of the lifting surface.
    sec_forces[nx-1, ny-1, 3] : numpy array
        Contains the sectional forces acting on each panel.
        Stored in Fortran order (only relevant with more than one chordwise
        panel).

    cg[3] : numpy array
        The x, y, z coordinates of the center of gravity for the entire aircraft.
    v : float
        Freestream air velocity in m/s.
    rho : float
        Air density in kg/m^3.

```

```

Returns
-----
CM[3] : numpy array
       The coefficient of moment around the x-, y-, and z-axes at the cg point.
       """

def __init__(self, surfaces, prob_dict):
    super(ComputeCM, self).__init__()

    tot_panels = 0
    for surface in surfaces:
        name = surface['name']
        ny = surface['num_y']
        nx = surface['num_x']

        self.add_param(name+'b_pts', val=np.zeros((nx-1, ny, 3), dtype=data_type))
        self.add_param(name+'widths', val=np.zeros((ny-1), dtype=data_type))
        self.add_param(name+'chords', val=np.zeros((ny), dtype=data_type))
        self.add_param(name+'S_ref', val=0.)
        self.add_param(name+'sec_forces', val=np.zeros((nx-1, ny-1, 3),
dtype=data_type))

        self.add_param('cg', val=np.zeros((3), dtype=data_type))
        self.add_param('v', val=10.)
        self.add_param('rho', val=3.)

        self.add_output('CM', val=np.zeros((3), dtype=data_type))

        self.surfaces = surfaces
        self.prob_dict = prob_dict

    if not fortran_flag:
        self.deriv_options['type'] = 'cs'
        self.deriv_options['form'] = 'central'

def solve_nonlinear(self, params, unknowns, resids):
    rho = params['rho']
    cg = params['cg']

    S_ref_tot = 0.
    M = np.zeros((3), dtype=data_type)

    for j, surface in enumerate(self.surfaces):
        name = surface['name']
        nx = surface['num_x']
        ny = surface['num_y']

        b_pts = params[name+'b_pts']
        widths = params[name+'widths']
        chords = params[name+'chords']
        S_ref = params[name+'S_ref']
        sec_forces = params[name+'sec_forces']

        panel_chords = (chords[1:] + chords[:-1]) / 2.
        MAC = 1. / S_ref * np.sum(panel_chords**2 * widths)

        if surface['symmetry']:
            MAC *= 2

        if fortran_flag:
            M_tmp = OAS_API.oas_api.momentcalc(b_pts, cg, chords, widths, S_ref,
sec_forces, surface['symmetry'])
            M += M_tmp

        else:
            pts = (params[name+'b_pts'][:, 1:, :] + \
params[name+'b_pts'][:, :-1, :]) / 2
            diff = (pts - cg) / MAC
            moment = np.zeros((ny - 1, 3), dtype=data_type)
            for ind in range(nx-1):
                moment += np.cross(diff[ind, :, :], sec_forces[ind, :, :], axis=1)

            if surface['symmetry']:
                moment[:, 0] = 0.
                moment[:, 1] *= 2
                moment[:, 2] = 0.
            M += np.sum(moment, axis=0)

```

```

        # For the first (main) lifting surface, we save the MAC
        if j == 0:
            MAC_wing = MAC
            S_ref_tot += S_ref

    self.M = M

    # Use the user-provided reference area from the main wing;
    # otherwise compute the total area of all lifting surfaces.
    if self-surfaces[0]['S_ref'] is None:
        self.S_ref_tot = S_ref_tot
    else:
        self.S_ref_tot = self-surfaces[0]['S_ref']

    unknowns['CM'] = M / (0.5 * rho * params['v']**2 * self.S_ref_tot * MAC_wing)

def linearize(self, params, unknowns, resids):

    jac = self.alloc_jacobian()

    cg = params['cg']
    rho = params['rho']
    v = params['v']

    for j in range(3):
        CMb = np.zeros((3))
        CMb[j] = 1.

        for i, surface in enumerate(self-surfaces):
            name = surface['name']
            ny = surface['num_y']

            b_pts = params[name+'b_pts']
            widths = params[name+'widths']
            chords = params[name+'chords']
            S_ref = params[name+'S_ref']
            sec_forces = params[name+'sec_forces']

            if i == 0:
                panel_chords = (chords[1:] + chords[:-1]) / 2.
                MAC = 1. / S_ref * np.sum(panel_chords**2 * widths)
                if surface['symmetry']:
                    MAC *= 2
                temp1 = self.S_ref_tot * MAC
                temp0 = 0.5 * rho * v**2
                temp = temp0 * temp1
                tempb = np.sum(-(self.M * CMb / temp)) / temp
                Mb = CMb / temp
                Mb_master = Mb.copy()
                jac['CM', 'rho'][j] += v**2*temp1*0.5*tempb
                jac['CM', 'v'][j] += 0.5*rho*temp1*2*v*tempb
                s_totb = temp0 * MAC * tempb
                machb = temp0 * self.S_ref_tot * tempb
                if surface['symmetry']:
                    machb *= 2
                chordsb = np.zeros((ny))
                tempb0 = machb / S_ref
                panel_chordsb = 2*panel_chords*widths*tempb0
                widthsb = panel_chords**2*tempb0
                sb = -np.sum(panel_chords**2*widths)*tempb0/S_ref
                chordsb[1:] += panel_chordsb/2.
                chordsb[:-1] += panel_chordsb/2.
                cb = chordsb
                wb = widthsb

            else:
                cb = 0.
                wb = 0.
                sb = 0.
                Mb = Mb_master.copy()

            bptsb, cg_b, chordsb, widthsb, S_refb, sec_forcesb, _ =
OAS API.oas.api.momentcalc b(b pts, cg, chords, widths, S_ref, sec forces,
surface['symmetry'], Mb)

            jac['CM', 'cg'][j, :] += cg_b

```

```

        jac['CM', name+'b_pts'][j, :] += bptsb.flatten()
        jac['CM', name+'chords'][j, :] += chordsb + cb
        jac['CM', name+'widths'][j, :] += widthsb + wb
        jac['CM', name+'sec_forces'][j, :] += sec_forcesb.flatten()
        jac['CM', name+'S_ref'][j, :] += S_refb + sb + s_totb

    return jac

class ComputeTotalCLCD(Component):
    """
    Compute the coefficients of lift (CL) and drag (CD) for the entire aircraft.

    Parameters
    -----
    CL : float
        Coefficient of lift (CL) for one lifting surface.
    CD : float
        Coefficient of drag (CD) for one lifting surface.
    S_ref : float
        Surface area for one lifting surface.
    v : float
        Freestream air velocity.
    rho : float
        Air density in kg/m^3.

    Returns
    -----
    CL : float
        Total coefficient of lift (CL) for the entire aircraft.
    CD : float
        Total coefficient of drag (CD) for the entire aircraft.

    """
    def __init__(self, surfaces, prob_dict):
        super(ComputeTotalCLCD, self).__init__()

        for surface in surfaces:
            name = surface['name']
            self.add_param(name+'CL', val=0.)
            self.add_param(name+'CD', val=0.)
            self.add_param(name+'S_ref', val=0.)

        self.add_param('v', val=10.)
        self.add_param('rho', val=3.)

        self.add_output('CL', val=0.)
        self.add_output('CD', val=0.)

        self.surfaces = surfaces
        self.prob_dict = prob_dict

    def solve_nonlinear(self, params, unknowns, resids):
        rho = params['rho']
        v = params['v']

        CL = 0.
        CD = 0.
        computed_total_S_ref = 0.
        for surface in self.surfaces:
            name = surface['name']
            S_ref = params[name+'S_ref']
            CL += params[name+'CL'] * S_ref
            CD += params[name+'CD'] * S_ref
            computed_total_S_ref += S_ref

        if self.surfaces[0]['S_ref'] is not None:
            S_ref_total = self.surfaces[0]['S_ref']
        else:
            S_ref_total = computed_total_S_ref

        unknowns['CL'] = CL / S_ref_total
        unknowns['CD'] = CD / S_ref_total
        self.S_ref_total = S_ref_total

    def linearize(self, params, unknowns, resids):
        jac = self.alloc_jacobian()

```



```

    for surface in self-surfaces:
        name = surface['name']
        S_ref = params[name+'S_ref']
        jac['CL', name+'CL'] = S_ref / self.S_ref_total
        jac['CD', name+'CD'] = S_ref / self.S_ref_total

        dCL_dS_ref = 0.
        surf_CL = params[name + 'CL']
        dCD_dS_ref = 0.
        surf_CD = params[name + 'CD']
        for surface_ in self-surfaces:
            name_ = surface_['name']
            if not name == name_:
                S_ref_ = params[name_ + 'S_ref']
                dCL_dS_ref += surf_CL * S_ref_
                dCL_dS_ref -= params[name_ + 'CL'] * S_ref_
                dCD_dS_ref += surf_CD * S_ref_
                dCD_dS_ref -= params[name_ + 'CD'] * S_ref_

        jac['CL', name + 'S_ref'] = dCL_dS_ref / self.S_ref_total**2
        jac['CD', name + 'S_ref'] = dCD_dS_ref / self.S_ref_total**2

    return jac

class TotalPerformance(Group):
    """
    Group to contain the total aerostructural performance components.
    """

    def __init__(self, surfaces, prob_dict):
        super(TotalPerformance, self).__init__()

        self.add('sink_speed',
                 SinkSpeedFunction(surfaces, prob_dict),
                 promotes=['*'])
        self.add('CD_over_CL',
                 RangeFunction(surfaces, prob_dict),
                 promotes=['*'])
        self.add('L_equals_W',
                 FunctionalEquilibrium(surfaces, prob_dict),
                 promotes=['*'])
        self.add('CG',
                 ComputeCG(surfaces, prob_dict),
                 promotes=['*'])
        self.add('moment',
                 ComputeCM(surfaces, prob_dict),
                 promotes=['*'])
        self.add('CL_CD',
                 ComputeTotalCLCD(surfaces, prob_dict),
                 promotes=['*'])

class TotalAeroPerformance(Group):
    """
    Group to contain the total aerodynamic performance components.
    """

    def __init__(self, surfaces, prob_dict):
        super(TotalAeroPerformance, self).__init__()

        self.add('moment',
                 ComputeCM(surfaces, prob_dict),
                 promotes=['*'])
        self.add('CL_CD',
                 ComputeTotalCLCD(surfaces, prob_dict),
                 promotes=['*'])

```

A3 - SCRIPT OF MODIFIED *run_classes.py*

```
from __future__ import division, print_function
import sys
from time import time
import numpy as np

# =====
# OpenMDAO modules
# =====
from openmdao.api import IndepVarComp, Problem, Group, ScipyOptimizer, Newton,
ScipyGMRES, LinearGaussSeidel, NLGaussSeidel, SqliteRecorder, profile, CaseReader,
DirectSolver
from openmdao.api import view_model
from six import iteritems

# =====
# OpenAeroStruct modules
# =====
from .geometry import GeometryMesh, Bspline, gen_crm_mesh, gen_rect_mesh,
MonotonicConstraint
from .transfer import TransferDisplacements, TransferLoads
from .vlm import VLMStates, VLMFunctionals, VLMGeometry
from .spatialbeam import SpatialBeamStates, SpatialBeamFunctionals, SpatialBeamSetup,
radii
from .materials import MaterialsTube
from .functionals import TotalPerformance, TotalAeroPerformance, SinkSpeedFunction,
RangeFunction, FunctionalEquilibrium
from .gs_newton import HybridGSNewton

class OASProblem(object):
    """
    Contain surface and problem information for aerostructural optimization.

    Parameters
    -----
    input_dict : dictionary
        The problem conditions and type of analysis desired. Note that there
        are default values defined by `get_default_prob_dict` that are overwritten
        based on the user-provided input_dict.
    """
    def __init__(self, input_dict={}):
        print('Fortran =', fortran_flag)

        # Update prob_dict with user-provided values after getting defaults
        self.prob_dict = self.get_default_prob_dict()
        self.prob_dict.update(input_dict)

        # Set the airspeed velocity based on the supplied Mach number
        # and speed of sound
        self.prob_dict['v'] = self.prob_dict['M'] * self.prob_dict['a']
        self-surfaces = []

        # Set the setup function depending on the problem type selected by the user
        if self.prob_dict['type'] == 'aero':
            self.setup = self.setup_aero
        if self.prob_dict['type'] == 'struct':
            self.setup = self.setup_struct
        if self.prob_dict['type'] == 'aerostruct':
            self.setup = self.setup_aerostruct

        # Set up dictionaries to hold user-supplied parameters for optimization
        self.desvars = {}
        self.constraints = {}
```

```

self.objective = {}

def get_default_prob_dict(self):
    """
    Obtain the default settings for the problem description. Note that
    these defaults are overwritten based on user input for the problem.

    Returns
    -----
    defaults : dict
        A python dict containing the default problem-level settings.
    """

    defaults = {
        # Problem and solver options
        'optimize' : False, # flag for analysis or optimization
        'optimizer' : 'SLSQP', # default optimizer
        'force_fd' : False, # if true, we FD over the whole model
        'with_viscous' : False, # if true, compute viscous drag
        'print_level' : 0, # int to control output during
            # optimization
            # 0 for no additional printing
            # 1 for nonlinear solver printing
            # 2 for nonlinear and linear solver
            # printing
        'previous_case_db' : None, # name of the .db file for warm
            # restart
            # example: 'aerostruct.db'
        'record_db' : True, # True to output .db file
        'profile' : False, # True to profile the problem's time
            # costs
            # view results using `view_profile
            # prof_raw.0`
        'compute_static_margin' : False, # if true, compute and print the
            # static margin after the run is
            # finished

        # Flow/environment properties
        'Re' : 1e6, # Reynolds number
        'reynolds_length' : 1.0, # characteristic Reynolds length
        'alpha' : 5., # [degrees] angle of attack
        'M' : 0.84, # Mach number at cruise
        'rho' : 0.38, # [kg/m^3] air density at 35,000 ft
        'a' : 295.4, # [m/s] speed of sound at 35,000 ft
        'g' : 9.80665, # [m/s^2] acceleration due to gravity

        # Aircraft properties
        'CT' : 9.80665 * 17.e-6, # [1/s] (9.80665 N/kg * 17e-6 kg/N/s)
            # specific fuel consumption
        'R' : 11.165e6, # [m] maximum range (B777-300)
        'cg' : np.zeros((3)), # Center of gravity for the
            # entire aircraft. Used in trim
            # and stability calculations.
        'W0' : 0.4 * 3e5, # [kg] weight of the airplane without
            # the wing structure and fuel.
            # The default is 40% of the MTOW of
            # B777-300 is 3e5 kg.
        'beta' : 1., # weighting factor for mixed objective
    }

    return defaults

def get_default_surf_dict(self):
    """
    Obtain the default settings for the surface descriptions. Note that
    these defaults are overwritten based on user input for each surface.
    Each dictionary describes one surface.

    Returns
    -----
    defaults : dict
        A python dict containing the default surface-level settings.
    """

    defaults = {
        # Wing definition
    }

```

```

'name' : 'wing', # name of the surface
'num_x' : 3, # number of chordwise points
'num_y' : 5, # number of spanwise points
'span_cos_spacing' : 1, # 0 for uniform spanwise panels
# 1 for cosine-spaced panels
# any value between 0 and 1 for
# a mixed spacing
'chord_cos_spacing' : 0., # 0 for uniform chordwise panels
# 1 for cosine-spaced panels
# any value between 0 and 1 for
# a mixed spacing
'wing_type' : 'rect', # initial shape of the wing
# either 'CRM' or 'rect'
# 'CRM' can have different options
# after it, such as 'CRM:alpha_2.75'
# for the CRM shape at alpha=2.75
'offset' : np.zeros((3)), # coordinates to offset
# the surface from its default location
'symmetry' : True, # if true, model one half of wing
# reflected across the plane y = 0
'S_ref_type' : 'wetted', # how we compute the wing area,
# can be 'wetted' or 'projected'

# Simple Geometric Variables
'span' : 10., # full wingspan, even for symmetric cases
'root_chord' : 1., # root chord
'dihedral' : 0., # wing dihedral angle in degrees
# positive is upward
'sweep' : 0., # wing sweep angle in degrees
# positive sweeps back
'taper' : 1., # taper ratio; 1. is uniform chord
'S_ref' : None, # [m^2] area of the lifting surface

# B-spline Geometric Variables. The number of control points
# for each of these variables can be specified in surf_dict
# by adding the prefix "num" to the variable (e.g. num_twist)
'twist_cp' : None,
'chord_cp' : None,
'xshear_cp' : None,
'yshear_cp' : None,
'zshear_cp' : None,
'thickness_cp' : None,
'radius_cp' : None,

# Geometric variables. The user generally does not need
# to change these geometry variables. This is simply
# a list of possible geometry variables that is later
# filtered down based on which are active.
'geo_vars' : ['sweep', 'dihedral', 'twist_cp', 'xshear_cp',
'yshear_cp', 'zshear_cp', 'span', 'chord_cp', 'taper',
'thickness_cp', 'radius_cp'],

# Aerodynamic performance of the lifting surface at
# an angle of attack of 0 (alpha=0).
# These CL0 and CD0 values are added to the CL and CD
# obtained from aerodynamic analysis of the surface to get
# the total CL and CD.
# These CL0 and CD0 values do not vary wrt alpha.
'CL0' : 0.0, # CL of the surface at alpha=0
'CD0' : 0.0, # CD of the surface at alpha=0

# Airfoil properties for viscous drag calculation
'k_lam' : 0.05, # percentage of chord with laminar
# flow, used for viscous drag
't_over_c' : 0.12, # thickness over chord ratio (NACA0012)
'c_max_t' : .303, # chordwise location of maximum (NACA0012)
# thickness

# Structural values are based on aluminum 7075
'E' : 70.e9, # [Pa] Young's modulus of the spar
'G' : 30.e9, # [Pa] shear modulus of the spar
'yield' : 500.e6 / 2.5, # [Pa] yield stress divided by 2.5 for
limiting case

'mrho' : 3.e3, # [kg/m^3] material density
'spar'
'fem_origin' : 0.35, # normalized chordwise location of the

'loads' : None, # [N] allow the user to input loads

```

```

        'disp' : None,          # [m] nodal displacements of the FEM model

        # Constraints
        'exact_failure_constraint' : False, # if false, use KS function
        'monotonic_con' : None, # add monotonic constraint to the given
                                # distributed variable. Ex. 'chord_cp'
    }
    return defaults

def add_surface(self, input_dict={}):
    """
    Add a surface to the problem. One surface definition is needed for
    each planar lifting surface.

    Parameters
    -----
    input_dict : dictionary
        Surface definition. Note that there are default values defined by
        'get_default_surface' that are overwritten based on the
        user-provided input_dict.
    """

    # Get defaults and update surface with the user-provided input
    surf_dict = self.get_default_surf_dict()
    surf_dict.update(input_dict)

    # Check to see if the user provides the mesh points. If they do,
    # get the chordwise and spanwise number of points
    if 'mesh' in surf_dict.keys():
        mesh = surf_dict['mesh']
        num_x, num_y = mesh.shape[:2]

    # If the user doesn't provide a mesh, obtain the values from surface
    # to create the mesh
    elif 'num_x' in surf_dict.keys():
        num_x = surf_dict['num_x']
        num_y = surf_dict['num_y']
        span = surf_dict['span']
        chord = surf_dict['root_chord']
        span_cos_spacing = surf_dict['span_cos_spacing']
        chord_cos_spacing = surf_dict['chord_cos_spacing']

    # Check to make sure that an odd number of spanwise points (num_y) was
    # provided
    if not num_y % 2:
        Error('num_y must be an odd number.')

    # Generate rectangular mesh
    if surf_dict['wing_type'] == 'rect':
        mesh = gen_rect_mesh(num_x, num_y, span, chord,
                             span_cos_spacing, chord_cos_spacing)

    # Generate CRM mesh. Note that this outputs twist information
    # based on the data from the CRM definition paper, so we save
    # this twist information to the surf_dict.
    elif 'CRM' in surf_dict['wing_type']:
        mesh, eta, twist = gen_crm_mesh(num_x, num_y, span, chord,
                                         span_cos_spacing, chord_cos_spacing, surf_dict['wing_type'])
        num_x, num_y = mesh.shape[:2]
        surf_dict['crm_twist'] = twist

    else:
        Error('wing_type option not understood. Must be either a type of ' +
              '"CRM" or "rect".')

    # Chop the mesh in half if using symmetry during analysis.
    # Note that this means that the provided mesh should be the full mesh
    if surf_dict['symmetry']:
        num_y = int((num_y+1)/2)
        mesh = mesh[:, :num_y, :]

    else:
        Error("Please either provide a mesh or a valid set of parameters.")

    # Compute span. We need .real to make span to avoid OpenMDAO warnings.
    quarter_chord = 0.25 * mesh[-1] + 0.75 * mesh[0]

```

```

surf_dict['span'] = max(quarter_chord[:, 1]).real - min(quarter_chord[:,
1]).real
if surf_dict['symmetry']:
    surf_dict['span'] *= 2.

# Apply the user-provided coordinate offset to position the mesh
mesh = mesh + surf_dict['offset']

# We need to initialize some variables to ones and some others to zeros.
# Here we define the lists for each case.
ones_list = ['chord_cp', 'thickness_cp', 'radius_cp']
zeros_list = ['twist_cp', 'xshear_cp', 'yshear_cp', 'zshear_cp']
surf_dict['bsp_vars'] = ones_list + zeros_list

# Loop through bspline variables and set the number of control points if
# the user hasn't initialized the array.
for var in surf_dict['bsp_vars']:
    numkey = 'num_' + var
    if surf_dict[var] is None:
        if numkey not in input_dict:
            surf_dict[numkey] = np.max([int((num_y - 1) / 5), min(5, num_y-
1)])
    else:
        surf_dict[numkey] = len(surf_dict[var])

# Interpolate the twist values from the CRM wing definition to the twist
# control points
if 'CRM' in surf_dict['wing_type']:
    num_twist = surf_dict['num_twist_cp']

    # If the surface is symmetric, simply interpolate the initial
    # twist_cp values based on the mesh data
    if surf_dict['symmetry']:
        twist = np.interp(np.linspace(0, 1, num_twist), eta,
surf_dict['crm_twist'])
    else:
        # If num_twist is odd, create the twist vector and mirror it
        # then stack the two together, but remove the duplicated twist
        # value.
        if num_twist % 2:
            twist = np.interp(np.linspace(0, 1, (num_twist+1)/2), eta,
surf_dict['crm_twist'])
            twist = np.hstack((twist[:-1], twist[::-1]))

        # If num_twist is even, mirror the twist vector and stack
        # them together
        else:
            twist = np.interp(np.linspace(0, 1, num_twist/2), eta,
surf_dict['crm_twist'])
            twist = np.hstack((twist, twist[::-1]))

    # Continue to use the user-defined twist_cp if inputted to the
    # surface dictionary. Otherwise, use the prescribed CRM twist.
    if surf_dict['twist_cp'] is None:
        surf_dict['twist_cp'] = twist

# Store updated values
surf_dict['num_x'] = num_x
surf_dict['num_y'] = num_y
surf_dict['mesh'] = mesh

radius = radii(mesh, surf_dict['t_over_c'])
surf_dict['radius'] = radius

# Set initial thicknesses
surf_dict['thickness'] = radius / 10

# We now loop through the possible bspline variables and populate
# the 'initial_geo' list with the variables that the geometry
# or user provided. For example, the CRM wing defines an initial twist.
# We must treat this separately so we add a twist bspline component
# even if it is not a desvar.
surf_dict['initial_geo'] = []
for var in surf_dict['geo_vars']:

    # Add the bspline variables when they're needed

```

```

if var in surf_dict['bsp_vars']:
    numkey = 'num_' + var
    if surf_dict[var] is None:

        # Add the initialized geometry variables to either ones or zeros.
        # These initial values do not perturb the mesh.
        if var in ones_list:
            surf_dict[var] = np.ones(surf_dict[numkey], dtype=data_type)
        elif var in zeros_list:
            surf_dict[var] = np.zeros(surf_dict[numkey], dtype=data_type)
        else:
            surf_dict['initial_geo'].append(var)

    # If the user provided a scalar variable (span, sweep, taper, etc),
    # then include that in the initial_geo list
    elif var in input_dict.keys():
        surf_dict['initial_geo'].append(var)

if 'thickness_cp' not in surf_dict['initial_geo']:
    surf_dict['thickness_cp'] *= np.max(surf_dict['thickness'])

if surf_dict['loads'] is None:
    # Set default loads at the tips
    loads = np.zeros((surf_dict['thickness'].shape[0] + 1, 6),
                    dtype=data_type)
    loads[0, 2] = 1e4
    if not surf_dict['symmetry']:
        loads[-1, 2] = 1e4
    surf_dict['loads'] = loads

if surf_dict['disp'] is None:
    # Set default disp if not provided
    surf_dict['disp'] = np.zeros((surf_dict['num_y'], 6), dtype=data_type)

# Throw a warning if the user provides two surfaces with the same name
name = surf_dict['name']
for surface in self-surfaces:
    if name == surface['name']:
        OASWarning("Two surfaces have the same name.")

# Append '_' to each repeated surface name
if not name:
    surf_dict['name'] = name
else:
    surf_dict['name'] = name + '_'

# Add the individual surface description to the surface list
self-surfaces.append(surf_dict)

def setup_prob(self):
    """
    Short method to select the optimizer. Uses pyOptSparse if available,
    or Scipy's SLSQP otherwise.
    """

    try: # Use pyOptSparse optimizer if installed
        from openmdao.api import pyOptSparseDriver
        self.prob.driver = pyOptSparseDriver()
        if self.prob_dict['optimizer'] == 'SNOPT':
            self.prob.driver.options['optimizer'] = "SNOPT"
            self.prob.driver.opt_settings = {'Major optimality tolerance': 1.0e-8,
                                            'Major feasibility tolerance': 1.0e-8,
                                            'Major iterations limit': 400,
                                            'Minor iterations limit': 2000,
                                            'Iterations limit': 1000}

        elif self.prob_dict['optimizer'] == 'ALPSO':
            self.prob.driver.options['optimizer'] = 'ALPSO'
            self.prob.driver.opt_settings = {'SwarmSize': 40,
                                            'maxOuterIter': 200,
                                            'maxInnerIter': 6,
                                            'rtol': 1e-5,
                                            'atol': 1e-5,
                                            'dtol': 1e-5,
                                            'printOuterIters': 1}

        elif self.prob_dict['optimizer'] == 'NOMAD':
            self.prob.driver.options['optimizer'] = 'NOMAD'
            self.prob.driver.opt_settings = {'maxiter': 1000,

```

```

        'minmeshsize':1e-12,
        'minpollsize':1e-12,
        'displaydegree':0,
        'printfile':1}
    elif self.prob_dict['optimizer'] == 'SLSQP':
        self.prob.driver.options['optimizer'] = 'SLSQP'
        self.prob.driver.opt_settings = {'ACC' : 1e-10}

except: # Use Scipy SLSQP optimizer if pyOptSparse not installed
    self.prob.driver = ScipyOptimizer()
    self.prob.driver.options['optimizer'] = 'SLSQP'
    self.prob.driver.options['disp'] = True
    self.prob.driver.options['tol'] = 1.0e-10

# Actually call the OpenMDAO functions to add the design variables,
# constraints, and objective.
for desvar_name, desvar_data in iteritems(self.desvars):
    self.prob.driver.add_desvar(desvar_name, **desvar_data)
for con_name, con_data in iteritems(self.constraints):
    self.prob.driver.add_constraint(con_name, **con_data)
for obj_name, obj_data in iteritems(self.objective):
    self.prob.driver.add_objective(obj_name, **obj_data)

# Use finite differences over the entire model if user selected it
if self.prob_dict['force_fd']:
    self.prob.root.deriv_options['type'] = 'fd'

# Record optimization history to a database.
# Data saved here can be examined using `plot_all.py` or `OptView.py`
if self.prob_dict['record_db']:
    recorder = SqliteRecorder(self.prob_dict['prob_name']+".db")
    recorder.options['record_params'] = True
    recorder.options['record_derivs'] = True
    self.prob.driver.add_recorder(recorder)

# Profile (time) the problem
if self.prob_dict['profile']:
    profile.setup(self.prob)
    profile.start()

# Set up the problem
self.prob.setup()

# Use warm start from previous db file if desired.
# Note that we only have access to the unknowns, not the gradient history.
if self.prob_dict['previous_case_db'] is not None:

    # Open the previous case and start from the last iteration.
    # Change the -1 value in get_case() if you want to select a different
    # iteration.
    cr = CaseReader(self.prob_dict['previous_case_db'])
    case = cr.get_case(-1)

    # Loop through the unknowns and set them for this problem.
    for param_name, param_data in iteritems(case.unknowns):
        self.prob[param_name] = param_data

def add_desvar(self, *args, **kwargs):
    """
    Store the design variables and later add them to the OpenMDAO problem.
    """
    self.desvars[str(*args)] = dict(**kwargs)

def add_constraint(self, *args, **kwargs):
    """
    Store the constraints and later add them to the OpenMDAO problem.
    """
    self.constraints[str(*args)] = dict(**kwargs)

def add_objective(self, *args, **kwargs):
    """
    Store the objectives and later add them to the OpenMDAO problem.
    """
    self.objective[str(*args)] = dict(**kwargs)

def run(self):
    """

```



```

Method to actually run analysis or optimization. Also saves history in
a .db file and creates an N2 diagram to view the problem hierarchy.
"""

# Have more verbose output about optimization convergence
if self.prob_dict['print_level']:
    self.prob.print_all_convergence()

# Save an N2 diagram for the problem
if self.prob_dict['record_db']:
    view_model(self.prob, outfile=self.prob_dict['prob_name']+".html",
               show_browser=False)

# If `optimize` == True in prob_dict, perform optimization. Otherwise,
# simply pass the problem since analysis has already been run.
if not self.prob_dict['optimize']:
    # Run a single analysis loop. This shouldn't actually be
    # necessary, but sometimes the .db file is not complete unless we do this.
    self.prob.run_once()
else:
    # Perform optimization
    self.prob.run()

# If the problem type is aero or aerostruct, we can compute the static margin.
# This is a naive temporary implementation that currently finite differences
# over the entire model to obtain the static margin.
if self.prob_dict['compute_static_margin'] and 'aero' in
self.prob_dict['type']:

    # Turn off problem recording (so nothing for these computations
    # appears in the .db file) and get the current CL and CM.
    self.prob.driver.recorders._recorders = []
    CL = self.prob['wing_perf.CL']
    CM = self.prob['CM'][1]
    step = 1e-5

    # Perturb alpha and run an analysis loop to obtain the new CL and CM.
    self.prob['alpha'] += step
    self.prob.run_once()
    CL_new = self.prob['wing_perf.CL']
    CM_new = self.prob['CM'][1]

    # Un-perturb alpha and run a single analysis loop to get the problem
    # back to where it was before we finite differenced.
    self.prob['alpha'] -= step
    self.prob.run_once()

    # Compute, print, and save the static margin in metadata.
    static_margin = -(CM_new - CM) / (CL_new - CL)
    print("Static margin is:", static_margin)
    self.prob.root.add_metadata('static_margin', static_margin)

# Uncomment this to check the partial derivatives of each component
# self.prob.check_partial_derivatives(compact_print=True)

def setup_struct(self):
    """
    Specific method to add the necessary components to the problem for a
    structural problem.
    """

    # Set the problem name if the user doesn't
    if 'prob_name' not in self.prob_dict.keys():
        self.prob_dict['prob_name'] = 'struct'

    # Create the base root-level group
    root = Group()

    # Create the problem and assign the root group
    self.prob = Problem()
    self.prob.root = root

    # Loop over each surface in the surfaces list
    for surface in self.surfaces:

        # Get the surface name and create a group to contain components

```

```

# only for this surface.
# This group's name is whatever the surface's name is.
# The default is 'wing'.
name = surface['name']
tmp_group = Group()

# Strip the surface names from the desvars list and save this
# modified list as self.desvars
desvar_names = []
for desvar in self.desvars.keys():

    # Check to make sure that the surface's name is in the design
    # variable and only add the desvar to the list if it corresponds
    # to this surface.
    if name[:-1] in desvar:
        desvar_names.append('.'.join(desvar.split('.')[:1]))

# Add independent variables that do not belong to a specific component.
# Note that these are the only ones necessary for structural-only
# analysis and optimization.
# Here we check and only add the variables that are desvars or a
# special var, radius, which is necessary to compute weight.
indep_vars = [('loads', surface['loads'])]
for var in surface['geo_vars']:
    if var in desvar_names or 'thickness' in var or var in
    surface['initial_geo']:
        indep_vars.append((var, surface[var]))

# Add structural components to the surface-specific group
tmp_group.add('indep_vars',
              IndepVarComp(indep_vars),
              promotes=['*'])
tmp_group.add('mesh',
              GeometryMesh(surface, self.desvars),
              promotes=['*'])
tmp_group.add('tube',
              MaterialsTube(surface),
              promotes=['*'])
tmp_group.add('struct_setup',
              SpatialBeamSetup(surface),
              promotes=['*'])
tmp_group.add('struct_states',
              SpatialBeamStates(surface),
              promotes=['*'])
tmp_group.add('struct_funcs',
              SpatialBeamFunctionals(surface),
              promotes=['*'])

# Add bspline components for active bspline geometric variables.
# We only add the component if the corresponding variable is a desvar
# or special (radius).
for var in surface['bsp_vars']:
    if var in desvar_names or var in surface['initial_geo'] or 'thickness'
    in var:
        n_pts = surface['num_y']
        if var in ['thickness_cp', 'radius_cp']:
            n_pts -= 1
        trunc_var = var.split('_')[0]
        tmp_group.add(trunc_var + '_bsp',
                      Bspline(var, trunc_var, surface['num_'+var], n_pts),
                      promotes=['*'])

# Add tmp_group to the problem with the name of the surface.
# The default is 'wing'.
root.add(name[:-1], tmp_group, promotes=[])

root.add_metadata(surface['name'] + 'yield_stress', surface['yield'])
root.add_metadata(surface['name'] + 'fem_origin', surface['fem_origin'])

# Actually set up the problem
self.setup_prob()

def setup_aero(self):
    """
    Specific method to add the necessary components to the problem for an
    aerodynamic problem.
    """

```

```

# Set the problem name if the user doesn't
if 'prob_name' not in self.prob_dict.keys():
    self.prob_dict['prob_name'] = 'aero'

# Create the base root-level group
root = Group()

# Create the problem and assign the root group
self.prob = Problem()
self.prob.root = root

# Loop over each surface in the surfaces list
for surface in self.surfaces:

    # Get the surface name and create a group to contain components
    # only for this surface
    name = surface['name']
    tmp_group = Group()

    # Strip the surface names from the desvars list and save this
    # modified list as self.desvars
    desvar_names = []
    for desvar in self.desvars.keys():

        # Check to make sure that the surface's name is in the design
        # variable and only add the desvar to the list if it corresponds
        # to this surface.
        if name[:-1] in desvar:
            desvar_names.append('.'.join(desvar.split('.')[:-1]))

    # Add independent variables that do not belong to a specific component
    indep_vars = [('disp', surface['disp'])]
    for var in surface['geo_vars']:
        if var in desvar_names or var in surface['initial_geo']:
            indep_vars.append((var, surface[var]))

    # Add aero components to the surface-specific group
    tmp_group.add('indep_vars',
                  IndepVarComp(indep_vars),
                  promotes=['*'])
    tmp_group.add('mesh',
                  GeometryMesh(surface, self.desvars),
                  promotes=['*'])
    tmp_group.add('def_mesh',
                  TransferDisplacements(surface),
                  promotes=['*'])
    tmp_group.add('vlmgeom',
                  VLMGeometry(surface),
                  promotes=['*'])

    # Add bspline components for active bspline geometric variables.
    # We only add the component if the corresponding variable is a desvar.
    for var in surface['bsp_vars']:
        if var in desvar_names or var in surface['initial_geo']:
            n_pts = surface['num_y']
            if var in ['thickness_cp', 'radius_cp']:
                n_pts -= 1
            trunc_var = var.split('_')[0]
            tmp_group.add(trunc_var + '_bsp',
                          Bspline(var, trunc_var, surface['num_'+var], n_pts),
                          promotes=['*'])

    # Add monotonic constraints for selected variables
    if surface['monotonic_con'] is not None:
        if type(surface['monotonic_con']) is not list:
            surface['monotonic_con'] = [surface['monotonic_con']]
        for var in surface['monotonic_con']:
            tmp_group.add('monotonic_' + var,
                          MonotonicConstraint(var, surface), promotes=['*'])

    # Add tmp_group to the problem as the name of the surface.
    # Note that is a group and performance group for each
    # individual surface.
    name_orig = name.strip('_')
    root.add(name_orig, tmp_group, promotes=[])
    root.add(name_orig+' perf', VLMFunctionals(surface, self.prob_dict),

```

```

        promotes=["v", "alpha", "M", "re", "rho"])

# Add problem information as an independent variables component
if self.prob_dict['Re'] == 0:
    Error('Reynolds number must be greater than zero for viscous drag ' +
        'calculation. If only inviscid drag is desired, set with_viscous ' +
        'flag to False.')

prob_vars = [('v', self.prob_dict['v']),
             ('alpha', self.prob_dict['alpha']),
             ('M', self.prob_dict['M']),
             ('re', self.prob_dict['Re']/self.prob_dict['reynolds_length']),
             ('rho', self.prob_dict['rho']),
             ('cg', self.prob_dict['cg'])]

root.add('prob_vars',
        IndepVarComp(prob_vars),
        promotes=['*'])

# Add a single 'aero_states' component that solves for the circulations
# and forces from all the surfaces.
# While other components only depends on a single surface,
# this component requires information from all surfaces because
# each surface interacts with the others.
root.add('aero_states',
        VLMStates(self-surfaces),
        promotes=['circulations', 'v', 'alpha', 'rho'])

# Explicitly connect parameters from each surface's group and the common
# 'aero_states' group.
# This is necessary because the VLMStates component requires information
# from each surface, but this information is stored within each
# surface's group.
for surface in self-surfaces:
    name = surface['name']

    # Perform the connections with the modified names within the
    # 'aero_states' group.
    root.connect(name[:-1] + '.def_mesh', 'aero_states.' + name + 'def_mesh')
    root.connect(name[:-1] + '.b_pts', 'aero_states.' + name + 'b_pts')
    root.connect(name[:-1] + '.c_pts', 'aero_states.' + name + 'c_pts')
    root.connect(name[:-1] + '.normals', 'aero_states.' + name + 'normals')

    # Connect the results from 'aero_states' to the performance groups
    root.connect('aero_states.' + name + 'sec_forces', name + 'perf' +
        '.sec_forces')

    # Connect S_ref for performance calcs
    root.connect(name[:-1] + '.S_ref', name + 'perf' + '.S_ref')
    root.connect(name[:-1] + '.widths', name + 'perf' + '.widths')
    root.connect(name[:-1] + '.chords', name + 'perf' + '.chords')
    root.connect(name[:-1] + '.lengths', name + 'perf' + '.lengths')
    root.connect(name[:-1] + '.cos_sweep', name + 'perf' + '.cos_sweep')

    # Connect S_ref for performance calcs
    root.connect(name[:-1] + '.S_ref', 'total_perf.' + name + 'S_ref')
    root.connect(name[:-1] + '.widths', 'total_perf.' + name + 'widths')
    root.connect(name[:-1] + '.chords', 'total_perf.' + name + 'chords')
    root.connect(name[:-1] + '.b_pts', 'total_perf.' + name + 'b_pts')
    root.connect(name + 'perf' + '.CL', 'total_perf.' + name + 'CL')
    root.connect(name + 'perf' + '.CD', 'total_perf.' + name + 'CD')
    root.connect('aero_states.' + name + 'sec_forces', 'total_perf.' + name +
        'sec_forces')

root.add('total_perf',
        TotalAeroPerformance(self-surfaces, self.prob_dict),
        promotes=['CM', 'CL', 'CD', 'v', 'rho', 'cg'])

# Actually set up the problem
self.setup_prob()

def setup_aerostruct(self):
    """
    Specific method to add the necessary components to the problem for an
    aerosturctural problem.
    """

```

```

# Set the problem name if the user doesn't
if 'prob_name' not in self.prob_dict.keys():
    self.prob_dict['prob_name'] = 'aerostruct'

# Create the base root-level group
root = Group()
coupled = Group()

# Create the problem and assign the root group
self.prob = Problem()
self.prob.root = root

# Loop over each surface in the surfaces list
for surface in self.surfaces:

    # Get the surface name and create a group to contain components
    # only for this surface
    name = surface['name']
    tmp_group = Group()

    # Strip the surface names from the desvars list and save this
    # modified list as self.desvars
    desvar_names = []
    for desvar in self.desvars.keys():

        # Check to make sure that the surface's name is in the design
        # variable and only add the desvar to the list if it corresponds
        # to this surface.
        if name[:-1] in desvar:
            desvar_names.append('.'.join(desvar.split('.')[:-1]))

    # Add independent variables that do not belong to a specific component
    indep_vars = []
    for var in surface['geo_vars']:
        if var in desvar_names or var in surface['initial_geo'] or 'thickness'
            in var:
            indep_vars.append((var, surface[var]))

    # Add components to include in the surface's group
    tmp_group.add('indep_vars',
                  IndepVarComp(indep_vars),
                  promotes=['*'])
    tmp_group.add('tube',
                  MaterialsTube(surface),
                  promotes=['*'])
    tmp_group.add('mesh',
                  GeometryMesh(surface, self.desvars),
                  promotes=['*'])
    tmp_group.add('struct_setup',
                  SpatialBeamSetup(surface),
                  promotes=['*'])

    # Add bspline components for active bspline geometric variables.
    # We only add the component if the corresponding variable is a desvar,
    # a special parameter (radius), or if the user or geometry provided
    # an initial distribution.
    for var in surface['bsp_vars']:
        if var in desvar_names or var in surface['initial_geo'] or 'thickness'
            in var:
            n_pts = surface['num_y']
            if var in ['thickness_cp', 'radius_cp']:
                n_pts -= 1
            trunc_var = var.split('_')[0]
            tmp_group.add(trunc_var + '_bsp',
                          Bspline(var, trunc_var, surface['num_'+var], n_pts),
                          promotes=['*'])

    # Add monotonic constraints for selected variables
    if surface['monotonic_con'] is not None:
        if type(surface['monotonic_con']) is not list:
            surface['monotonic_con'] = [surface['monotonic_con']]
        for var in surface['monotonic_con']:
            tmp_group.add('monotonic_' + var,
                          MonotonicConstraint(var, surface), promotes=['*'])

# Add tmp group to the problem with the name of the surface.

```

```

name_orig = name
name = name[:-1]
root.add(name, tmp_group, promotes=[])

# Add components to the 'coupled' group for each surface.
# The 'coupled' group must contain all components and parameters
# needed to converge the aerostructural system.
tmp_group = Group()
tmp_group.add('def_mesh',
              TransferDisplacements(surface),
              promotes=['*'])
tmp_group.add('aero_geom',
              VLMGeometry(surface),
              promotes=['*'])
tmp_group.add('struct_states',
              SpatialBeamStates(surface),
              promotes=['*'])
tmp_group.struct_states.ln_solver = LinearGaussSeidel()
tmp_group.struct_states.ln_solver.options['atol'] = 1e-20

name = name_orig
coupled.add(name[:-1], tmp_group, promotes=[])

# Add a loads component to the coupled group
coupled.add(name_orig + 'loads', TransferLoads(surface), promotes=[])

# Add a performance group which evaluates the data after solving
# the coupled system
tmp_group = Group()

tmp_group.add('struct_funcs',
              SpatialBeamFunctionals(surface),
              promotes=['*'])
tmp_group.add('aero_funcs',
              VLMFunctionals(surface, self.prob_dict),
              promotes=['*'])

root.add(name_orig + 'perf', tmp_group, promotes=["rho", "v", "alpha",
                                                "re", "M"])

root.add_metadata(surface['name'] + 'yield_stress', surface['yield'])
root.add_metadata(surface['name'] + 'fem_origin', surface['fem_origin'])

# Add a single 'aero_states' component for the whole system within the
# coupled group.
coupled.add('aero_states',
            VLMStates(self-surfaces),
            promotes=['v', 'alpha', 'rho'])

# Explicitly connect parameters from each surface's group and the common
# 'aero_states' group.
for surface in self-surfaces:
    name = surface['name']

    root.connect(name[:-1] + '.K', 'coupled.' + name[:-1] + '.K')

    # Perform the connections with the modified names within the
    # 'aero_states' group.
    root.connect('coupled.' + name[:-1] + '.def_mesh', 'coupled.aero_states.'
                + name + 'def_mesh')
    root.connect('coupled.' + name[:-1] + '.b_pts', 'coupled.aero_states.'
                + name + 'b_pts')
    root.connect('coupled.' + name[:-1] + '.c_pts', 'coupled.aero_states.'
                + name + 'c_pts')
    root.connect('coupled.' + name[:-1] + '.normals', 'coupled.aero_states.'
                + name + 'normals')

    # Connect the results from 'aero_states' to the performance groups
    root.connect('coupled.aero_states.' + name + 'sec_forces', name + 'perf'
                + '.sec_forces')

    # Connect the results from 'coupled' to the performance groups
    root.connect('coupled.' + name[:-1] + '.def_mesh', 'coupled.' + name +
                'loads.def_mesh')
    root.connect('coupled.aero_states.' + name + 'sec_forces', 'coupled.' +
                name + 'loads.sec_forces')

```

```

# Connect the output of the loads component with the FEM
# displacement parameter. This links the coupling within the coupled
# group that necessitates the subgroup solver.
root.connect('coupled.' + name + 'loads.loads', 'coupled.' + name[:-1] +
            '.loads')

# Connect aerodynamic mesh to coupled group mesh
root.connect(name[:-1] + '.mesh', 'coupled.' + name[:-1] + '.mesh')

# Connect performance calculation variables
root.connect(name[:-1] + '.radius', name + 'perf.radius')
root.connect(name[:-1] + '.A', name + 'perf.A')
root.connect(name[:-1] + '.thickness', name + 'perf.thickness')

# Connection performance functional variables
root.connect(name + 'perf.structural_weight', 'total_perf.' + name +
            'structural_weight')
root.connect(name + 'perf.L', 'total_perf.' + name + 'L')
root.connect(name + 'perf.CL', 'total_perf.' + name + 'CL')
root.connect(name + 'perf.CD', 'total_perf.' + name + 'CD')
root.connect('coupled.aero_states.' + name + 'sec_forces', 'total_perf.' +
            name + 'sec_forces')

# Connect parameters from the 'coupled' group to the performance
# groups for the individual surfaces.
root.connect(name[:-1] + '.nodes', name + 'perf.nodes')
root.connect('coupled.' + name[:-1] + '.disp', name + 'perf.disp')
root.connect('coupled.' + name[:-1] + '.S_ref', name + 'perf.S_ref')
root.connect('coupled.' + name[:-1] + '.widths', name + 'perf.widths')
root.connect('coupled.' + name[:-1] + '.chords', name + 'perf.chords')
root.connect('coupled.' + name[:-1] + '.lengths', name + 'perf.lengths')
root.connect('coupled.' + name[:-1] + '.cos_sweep', name +
            'perf.cos_sweep')

# Connect parameters from the 'coupled' group to the total performance
# group.
root.connect('coupled.' + name[:-1] + '.S_ref', 'total_perf.' + name +
            'S_ref')
root.connect('coupled.' + name[:-1] + '.widths', 'total_perf.' + name +
            'widths')
root.connect('coupled.' + name[:-1] + '.chords', 'total_perf.' + name +
            'chords')
root.connect('coupled.' + name[:-1] + '.b_pts', 'total_perf.' + name +
            'b_pts')
root.connect(name + 'perf.cg_location', 'total_perf.' + name +
            'cg_location')

# Set solver properties for the coupled group
coupled.ln_solver = ScipyGMRES()
coupled.ln_solver.preconditioner = LinearGaussSeidel()
coupled.aero_states.ln_solver = LinearGaussSeidel()
coupled.nl_solver = NLGaussSeidel()

# This is only available in the most recent version of OpenMDAO.
# It may help converge tightly coupled systems when using NLGS.
try:
    coupled.nl_solver.options['use_aitken'] = True
    coupled.nl_solver.options['aitken_alpha_min'] = 0.01
    # coupled.nl_solver.options['aitken_alpha_max'] = 0.5
except:
    pass

if self.prob_dict['print_level'] == 2:
    coupled.ln_solver.options['iprint'] = 1
if self.prob_dict['print_level']:
    coupled.nl_solver.options['iprint'] = 1

# Add the coupled group to the root problem
root.add('coupled', coupled, promotes=['v', 'alpha', 'rho'])

# Add problem information as an independent variables component
prob_vars = [('v', self.prob_dict['v']),
            ('alpha', self.prob_dict['alpha']),
            ('M', self.prob_dict['M']),
            ('re', self.prob_dict['Re']/self.prob_dict['reynolds_length']),
            ('rho', self.prob_dict['rho'])]

```

```
root.add('prob_vars',
        IndepVarComp(prob_vars),
        promotes=['*'])

# Add functionals to evaluate performance of the system.
# Note that only the interesting results are promoted here; not all
# of the parameters.
root.add('total_perf',
        TotalPerformance(self.surfaces, self.prob_dict),
        promotes=['L_equals_W', 'sink_speed', 'CD_over_CL', 'CM', 'CL', 'CD',
        'v', 'rho', 'cg', 'total_weight'])

# Actually set up the system
self.setup_prob()
```

