Development of a Variable Camber Compliant Aircraft Tail using Structural Optimization

by

Matthew G. Good

Thesis Submitted to the Faculty of the
Virginia Polytechnic Institute and State University
In partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

July 2003
Blacksburg, Virginia

Keywords: Morphing Airplanes, Compliant Mechanism, Variable Camber, Structural Optimization, Method of Moving Asymptotes

Development of a Variable Camber Compliant Aircraft Tail using Structural Optimization

Matthew G. Good

Abstract

The objectives of the research presented in this thesis are the development of a seven degree-of-freedom morphing airplane and the design and integration of a variable camber compliant tail. The morphing airplane was designed and manufactured to study the benefits of large planform changes and flight control morphing. Morphing capabilities of each wing consist of 8 in. wing extension and contraction, 40° of wing sweep and ±20.25° of outboard wing twist in addition to 6 in. of tail extension and contraction. Initial wind-tunnel tests proved that for a large range of lift coefficients, the optimal airplane configuration changes to minimize the drag.

Another portion of this research deals with the development of a structural optimization program to design a variable camber compliant tail. The program integrates ANSYS, aerodynamic thin airfoil theory and the Method of Moving Asymptotes to optimize the shape of an airfoil tail for maximum trailing edge deflection. An objective function is formulated to maximize the trailing edge tip deflection subject to stress constraints. The optimal structure needs to be flexible to maximize the tip deflection, but stiff enough to minimize the deflection of the tip due to aerodynamic loading. The results of the structural optimization program created a compliant tail mechanism that can deflect the trailing edge tip with a single actuator ±4.27°.

Acknowledgments

I would like to thank my advisor Dr. Daniel J. Inman for his support throughout this work. Dr. Inman's knowledge and excitement throughout this research made it a wonderful experience. I would also like to thank Dr. Harry H. Robertshaw for his contributions and support of the morphing airplane program at Virginia Tech. To Dr. William H. Mason I am grateful for his insights on aerodynamics and wind-tunnel testing. A special thanks to Dr. Jae-Sung Bae for answering many of my programming questions and helping me throughout my research. I would also like to thank Mostafa Abdalla for his guidance and assistance with structural optimization.

I would especially like to thank my parents for their love and support throughout my life. Their encouragement has helped me accomplish many great things. To my brothers, sister and friends, thank you for always being there for me and giving me the opportunity to get away from my work.

To the fellow Virginia Tech graduate students who made tailgating the football games with the Hokie Grill and canoeing the New River a memorable experience. Thank you for making my time at Virginia Tech an enjoyable one.

This research has been supported by DARPA and the Goodson Research Fund.

Table of Contents

List of Tables

List of Figures

Nomenclature

W_{aero}	- aerodynamic lifting force
W_{span}	- weight of the span actuator
W_{twist}	- weight of the twist actuator
P	- fictitious load
M_{twist}	- twist moment equation
$M_{span\,cylinder\,/\,rod}$	- span cylinder and rod moment equation
x	- distance from the fictitious load to point of interest
L_1	- length of the span cylinder
L_2	- length of the span rod
L_3	- length of the twist mechanism
E	- Young's modulus
I	- moment of inertia
δ	- beam deflection
u_{Tip}	- vertical trailing edge tip displacement
σ_i	- stress in ith beam member
$\sigma_{allowable}$	- yield stress divided by the factor of safety
m	- total number of constraints
n	- total number of design variables
h_{lower}	- lower bound of the design variables
h_{upper}	- upper bound of the design variables

$x^{(o)}$	- initial starting point of the optimization program
k	- iteration index for the Method of Moving Asymptotes
$f_i\left(x^{(k)}\right)$	- convex approximation of the objective function and constraints
$\nabla f_i\left(x^{(k)}\right)$	- gradient information of the approximated objective function and constraints
$U_j^{(k)}$	- moving asymptote upper value
$L_j^{(k)}$	- moving asymptote lower value
$E_{Desired}$	- desired percent error value
Obj_{New}	- new value of the objective function
Obj_{Old}	- previous iteration's value of the objective function

Chapter 1

Introduction

1.1 Morphing Airplane Background

Designing a morphing airplane gives the vehicle the ability to change its aerodynamic characteristics while in flight. Improvement to the vehicle's performance can be accomplished through planform configuration changes and/or wing shape changes. With the capability of planform changes, called adaptive morphing, the airplane can meet multiple mission requirements. An example of a planform area change is an airplane extending and contracting its wing. Having a larger wingspan will generate more lift, so the airplane will be able loiter over a desired location. Then by contracting the wingspan, the airplane will be able to simulate a fighter jet, meant for faster speeds to attack. For flight shape changes, the airfoil shape enables the wings to be used as flight control surfaces. Flight control morphing is done dynamically in flight to control the roll, pitch and yaw of the airplane as desired, as well as for trimming the airplane. A lot of research has been accomplished to determine which morphing capability offers the most benefit and this thesis will develop a platform to help determine the aerodynamic benefits of each.

1.1.1 Mission Adaptive Morphing Literature Review

Planform morphing has been used by the military to create airplanes to meet a variety of mission objectives. One of the most successful morphing airplanes is the F-14 Tomcat. The Tomcat is able to sweep its wings from 20 to 68° (F-14 Tomcat, 2004). This range of sweep allowed the airplane to generate large amounts of lift when the wings were unswept, such as during take-off and landings, then during flight, the wings were swept back to a more optimal aerodynamic configuration at higher speeds.

In addition to sweeping capabilities, research by Blondeau *et al* has been done on extending and contracting the wingspan of an airplane. Blondeau's morphing wing uses inflatable telescopic spars to extend the wing, from 7 to 15 inches, and can support wing loadings in excess of 15 lbs/ft^2. This extension generates a change in aspect ratio of 114%, while achieving a lift to drag ratio of 9 to 10 (Blondeau *et al*, 2003).

Another morphing capability investigated was the Hyper-Elliptic Cambered Span (HECS) wing by Wiggins *et al.* The HECS wing is a hyper-elliptical swept leading and trailing edge wing that has a variable spanwise camber. Using a single-degree-of-freedom mechanism, the wing could continuously change shape form a nonplanar to a planar configuration. Then, using linear aerodynamic theory, the aerodynamic forces produced by the HECS wing during morphing could be obtained. By changing the spanwise camber, an increase of 14% lift to drag ratio is achieved compared to a planar elliptical chord wing (Wiggins *et al*, 2004).

The benefits of morphing have been investigated by Johnston *et al*, who compared the flight control energy requirements of morphing and conventionally actuated wings. An adaptive camberline was designed to generate the morphing wing shapes. Then the aerodynamic energy was derived from the camberline functions using the vortex lattice method. Applying beam theory to the morphing airfoil sections, the strain energy could be obtained along with the Lagrange multipliers. Combining the aerodynamic and strain energies with the Lagrange multipliers, the optimal energy wing deflection could be determined. Using total energy as a cost function and placing constraints on achieving changes in lift and moment coefficients, the work expressions for a wing with morphing inputs and a wing with inboard and outboard flaps, could be implemented numerically. At the conclusion of the investigation, the spanwise deflection of the morphing wing allowed for minimization of the aerodynamic energy requirements, however the morphing vehicle generated strain not produced by the conventional vehicle (Johnston *et al*, 2003).

1.1.2 Flight Control Morphing Literature Review

Flight control morphing changes the airfoil shape to achieve a desired flight characteristic. The idea originates from nature and a bird's ability to vary the geometry of its wings and tail. Currently, airplanes use flaps and ailerons to modify the wing's camber to change their lift effectiveness. The flaps and ailerons offer a way to change the camber of the wing. However, the discontinuity generated between the wing and the flap and/or aileron increases drag. By alleviating the discontinuity between the lifting surface and the flap, a morphing wing will better simulate a bird's wing. Looking at nature, birds optimize their wings and tail for all aspects of flight from take-off to landing, by changing the camber and planform area of their wings and tail. In a paper by Adrian Thomas, Thomas looked at five different shapes of birds' tails and determined their aerodynamic characteristics. As a bird flys, the tail feathers change their deflection to change the angle of attack of the bird as well as stabilize the bird during slow flight (Thomas, 2004).

To mimic the flight control capabilities of a bird, flight control morphing uses small deflections in the wings to generate the desired motion instead of large planform changes. These small deflections produce the desired affects on an airplane at high speeds. Since small deflections are all that are needed to control the airplane during high speed flight, smart materials are currently being researched to morph a wing. Kudva *et al* used smart materials as control surfaces of a wing to alleviate the need for flaps. Using a mechanical actuation system to deflect the leading and trailing edge that is enclosed in a flexible wing skin eliminates the discontinuities in the airfoil due to the flaps and ailerons. Since the wings are a continuous surface, the lift and stall angle characteristics of the airplane improve (Kudva *et al*, 1997).

A "Smart Materials and Structures Development – Smart Wing" program was started by DARPA/USAF Wright Laboratory to investigate the use of smart materials to morph a wing. Wing twist and adaptive control surfaces were investigated through the Smart Wing program. Wing twisting was accomplished using a shape memory alloy (SMA) torque tube actuator. The wing was designed to achieve a twist angle of 3 to 5°. The adaptive control surface design used an SMA based actuation system to create a continuous wing surface. Through experimentation, the trailing edge of the wing was able to rotate 10° (Kudva *et al*, 1997).

A variety of flight control morphing has been attempted with smart materials. This is because smart materials are lightweight and each type of smart material has its benefits depending on what type of motion is required. To investigate the energy requirements of active materials to morph a wing, Pettit *et al*, developed a low cost computational model. Pettit's model predicted the force, stroke and energy requirements needed for the actuator to overcome aerodynamic loading encountered during morphing changes. By using the conformal mapping technique, the model allowed the planform to be easily changed and the requirements of the actuator were quickly determined (Pettit *et al*, 2002).

The drawback to using smart materials for morphing is because they require a lot of power. Stack actuators produce a lot of force, but not much motion, whereas SMA wires produce large amounts of motion, but are slow because the SMA wires need to be heated to deform and cooled to return to their original shape. Flight control surfaces need to have quick reaction times while mission adaptive morphing need to have large deflections. Therefore, smart materials by themselves have either too slow or have too short a stroke to be effective for morphing motion. A solution to a continuous, fast acting flap is to use levers to amplify the motion of an actuator. Compliant mechanisms are optimized for leveraging the motion of an actuator and distribution of forces over the surface. This research will investigate the feasibility of a compliant tail mechanism by designing and manufacturing one for testing

1.2 Introduction to Compliant Mechanisms

Since there are many drawbacks to using smart materials for airplane morphing, compliant mechanisms offer an alternative method to achieve the desired large continuous wing deflection. To improve the pitching moment of the adaptive airplane, discussed more in-depth in Chapter 2, a variable camber tail will be mounted to the airplane. With the addition of another control surface on the morphing airplane, the variable camber tail and the twisting of the outboard wings will be used to trim the airplane during flight. The variable camber tail is similar to a flap on a fixed wing aircraft; however, the tail will be a continuous curved surface while a flap causes a discontinuity between the flap and the tail airfoil. Gern *et al* (2002) investigated the benefits of a continuous morphing surface compared to a fixed wing aircraft with a flap. The model developed showed the potential of a continuous morphing airfoil on an aircrafts performance over a conventional airplane. Therefore, instead of mounting a flap on the tail to be used to pitch the morphing airplane, a continuous variable camber tail will be designed. To design a variable camber tail, the structure will be a compliant mechanism. Given an input force on a compliant mechanism, the internal structure elastically deforms to transfer the input force to a desired motion, force or energy. In the case for the morphing airplane, a single actuator will provide an input force on the compliant tail which will cause a desired trailing edge tip deflection. A few simple examples of a compliant mechanism are shown in Figure 1.1.

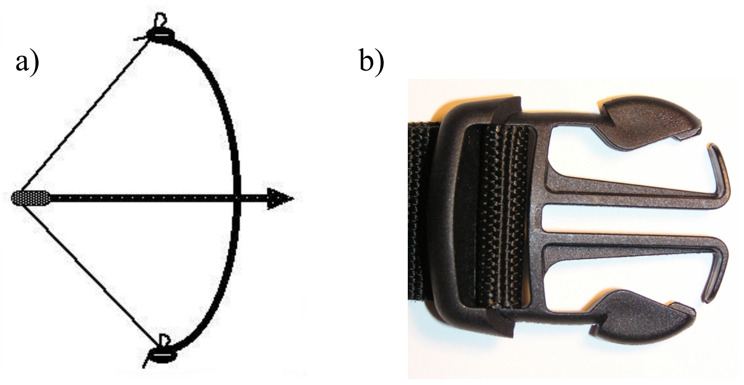

Figure 1.1: Examples of compliant mechanisms a) Bow b) Backpack clip

1.2.1 History of Compliant Mechanisms

Designing compliant mechanisms was first investigated by Paros *et al* (1965). Up until that point, compliant mechanisms were designed through trial and error. Paros developed exact and simplified equations for calculating the critical spring rates of a single-axis and two-axis flexure hinge. Flexure hinges allow for frictionless limited angular motion about the compliant hinge with zero backlash. Figure 1.2 shows a single axis flexure hinge. If a force is applied to the flexure hinge as shown, the compliant hinge will deflect about the necked down region. Today, flexure hinges are mostly used in Micro-Electro-Mechanical Systems (MEMS) applications.

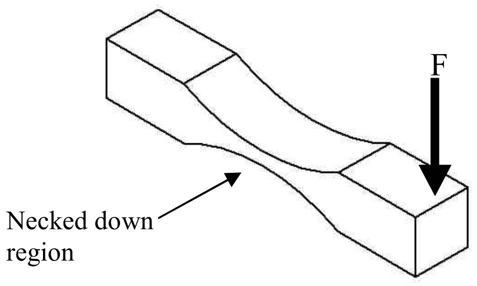

Figure 1.2: Single axis flexure hinge

Since Paro's initial research into flexible hinge design, compliant mechanisms have grown in popularity. One of the main reasons is that in most cases, no assembly is required. The mechanisms are made out of a single piece of material. Having no assembly greatly speeds up production time and decreases the manufacturing cost of the part. Also, since the mechanism is made out of a single piece of material there are no joints, which eliminates backlash. Since there is no backlash, the force input to the mechanism easily controls the output displacement. Another benefit of compliant mechanisms is that a single device using a compliant member mimics nature. As stated by Ananthasuresh *et al* (1995); "Traditionally, engineered artifacts are designed to be strong and rigid. Designs in nature, on the other hand, are strong and compliant." Compliant mechanisms offer many manufacturing advantages over equivalent assembled products; however, the design of compliant mechanisms is more involved. For small devices, like MEMS, compliant mechanisms work well because the motion is linear. However, small, linear motion will not work for a variable camber tail design. For the compliant mechanisms to achieve large deflections, nonlinearity in the members starts to occur. To account for the large nonlinear deflections, two different types of design methods are used to create a compliant mechanism; the kinematic synthesis approach and the topology optimization approach.

1.2.2 Kinematic Synthesis Approach

The kinematic synthesis approach was developed by Howell *et al* (1994) to design compliant mechanisms with large nonlinear displacements. This method models a compliant mechanism as a pseudo-rigid-body model, so rigid-link mechanism theory may be used to analyze the system, see Figure 1.3. The pseudo-rigid-body method models the compliant members as cantilever beams and the connection points of the members as pin joints. With the addition of springs to the model, the force-deflection relationships of the compliant mechanisms can be predicted.

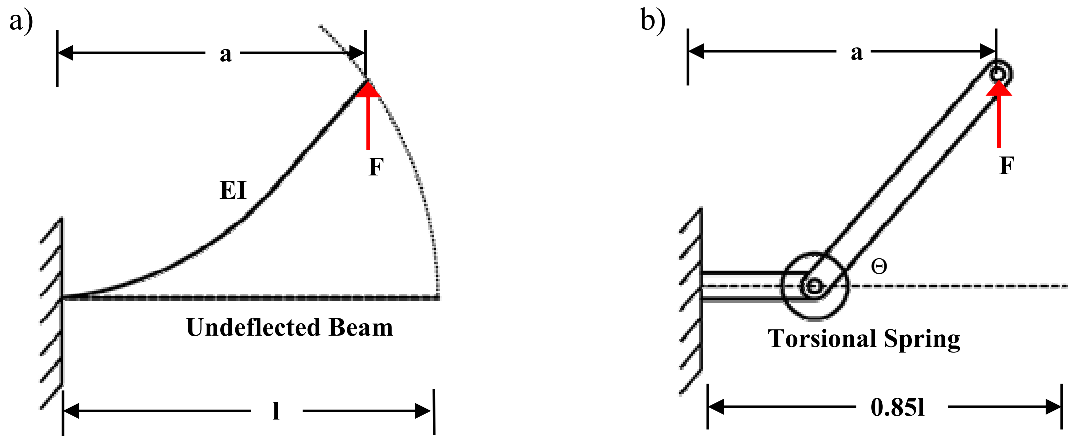

Figure 1.3: a) Compliant mechanism b) Pseudo-rigid-body mechanism model of compliant mechanism

1.2.3 Topology Optimization Approach

Topology optimization accounts for large nonlinear displacements by discretizing the design space with truss or beam elements. By discretizing the design space, each member of structure will undergo only small linear displacement; therefore, linear analysis can be used. The ground structure parameterization can be a full mesh, where every node is connected with a truss or a beam element, or a partial mesh, which only connects certain nodes (Bendsoe *et al*, 2003). Figure 1.4 demonstrates a full ground structure (a), and a partial ground structure (b). The full ground structure will allow more topology considerations for the optimization program; however, a design variable is needed for every member. As the number of design variables increase, so does the computation time for each iteration. The partial ground structure will decrease the number of design variables and computing time to determine an optimal shape. The disadvantage to a coarse mesh is the optimal compliant structure will need to be refined. The coarse optimized structure will be similar to the full ground structure, however, since the structural optimization program can only optimize the structure given, the optimized structure may have strange connections between members. For example, with a coarse optimized structure, it may have a right angle connection between two members to simulate a diagonal member to connect the nodes. Therefore, the finer the mesh, the better the optimal structural shape is designed, since the structural optimization program has more members to design with. A major benefit of topology optimization is the finite element mesh never has to be remeshed as the optimization program progresses. Since the members never actually go to zero, the mesh of the structure is always constant. The members that need to be removed go to a set minimum cross-sectional area value instead of zero so singularity in the finite element matrices do not arise (Frecker, 1997). Another benefit of topology optimization over kinematic synthesis is during optimization of the design space, the optimization program determines the topology of the structure. Once the optimization program converges, the final optimized structure is completed. The optimum shape will deflect the desired amount without any of the members going beyond the material properties capabilities.

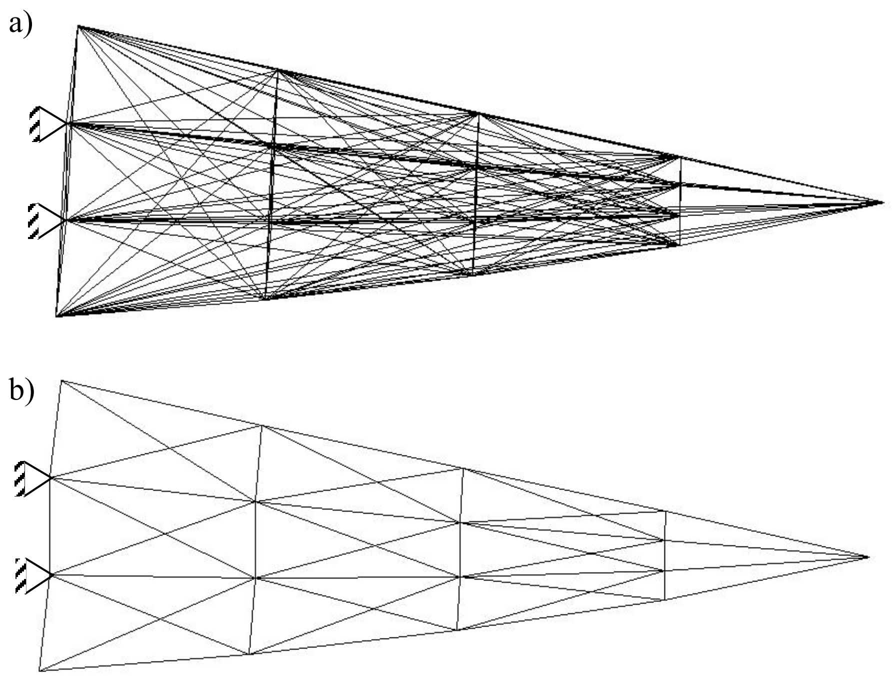

Figure 1.4: a) Full ground structure b) partial ground structure of the tail

1.2.4 Literature Review of Compliant Mechanisms

A variety of research has been done using compliant mechanisms for morphing shape changes. Saggere *et al* (1999) used a compliant mechanisms to achieve small shape changes for static shape control with a single torque actuator. Using shape control to control the position or alignment of a certain number of points on the structure, the surface can track a desired shape. To determine the position and alignment of the points on the structure, dimensional synthesis was used. As the actuator applies a torque to the elastic structural frame, the desired surface shape is formed by minimizing the least squares error of the final shape to the desired shape.

Saxena *et al* used topology optimization to develop a compliant mechanism with strength considerations by incorporating local failure conditions relating to stress constraints. By taking into account the local failure conditions, Saxena *et al* were able to design flexible, yet stiff structures such as compliant pliers and crimpers (Saxena *et al*, 2001). Taking Saxena *et al* research a step further, Frecker improved the objective function by deriving a multi-criteria formulation of the objective function. Frecker demonstrated that by maximizing a ratio of the output displacement and strain energy, the mechanical efficiency of the compliant mechanism improved (Frecker 1997).

Anusonti-Inthra *et al* used Frecker's multi-criteria formulation of the Mutual Potential Energy and Strain Energy in designing an active truss structure for a rotor airfoil. Using topology optimization, the objective of the active truss is to optimize the location of piezoelectric actuators to produce the maximum trailing edge deflection. To simplify manufacturing, a partial ground structure is used to mesh the airfoil. The results from the topology optimization of maximizing the trailing edge deflection while constraining the active material volume, the achievable deflection was 3.7° of the effective flap angle (Anusonti-Inthra *et al*, 2003).

1.3 Thesis Overview

The purpose of this research is the development of a variable camber airplane with eight degrees of freedom. Chapter 2 discusses the design and wind-tunnel testing for a seven degree-of-freedom morphing airplane that has a maximum fuselage chord length of 49 in. and a variable wing span of 46.52 in. to 61.52 in. The airplane is capable of extending and contracting each wing 8 inches along with sweeping each wing up to 40°. The outboard wings can twist ±20° and the tail can extend and contract 6 inches. Wind-tunnel analysis was completed on the airplane to determine the optimal airplane configuration for certain ranges of lift that would produce the least amount of drag.

Chapter 3 discusses the development of the structural optimization code to generate a variable camber tail. To develop a variable camber tail, finite element analysis of the structure needed to be completed along with linking the results to the Method of Moving Asymptotes optimization code. An in-depth discussion of The Method of Moving Asymptotes will be presented and the reason for choosing this particular optimization tool over others will be discussed. The structural optimization problem will be posed.

Chapter 4 presents the results and analysis of the structural optimization code. The problems with stress as constraints will be addressed and the optimal shape will be presented. Also, the manufacturing of the compliant mechanism will be discussed.

Chapter 5 wraps up this research with the conclusion along with recommendations for future work. The structural optimization code along with analysis code is presented in the Appendices.

Chapter 2

Design and Wind-Tunnel Analysis of a Morphing Airplane

The purpose for designing and constructing a morphing airplane was to investigate the aerodynamics effects produced by large scale shape changes. To evaluate multiple morphing transformations, the airplane is designed to have seven degrees-of-freedom (dof). Each wing is independently controlled to sweep, twist and extend/contract, while the tail can also extend and contract to change the chord length of the fuselage. The objective during wind-tunnel testing was to determine the benefits of morphing for multiple mission configurations and flight control. Initial wind-tunnel testing to determine minimal drag configurations showed that over a range of flight conditions, the optimal planform shape changed.

2.1 Design Requirements and Constraints

The Morphing Aircraft Structures (MAS) program was started by the Defense Advanced Research Projects Agency (DARPA). The goal of the MAS program was to design a airplane capable of performing multi-mission objectives by changing the airplane's physical characteristics in flight. To accomplish this objective, DARPA set design requirements for the morphing wing airplanes to achieve. These requirements are (DARPA, 2003):

1) 200% change in Aspect Ratio(AR)
2) 50% Change in Wing Area
3) 5° change in wing twist
4) 20° change in wing sweep.

In addition to DARPA's requirements, the 6 ft by 6 ft dimensions of Virginia Tech's wind-tunnel limited the wing span of the model. To minimize wall effects during wind-tunnel testing, the maximum wing span was designed to include space between the wing tip and the wind-tunnel wall. Besides the span constraint, weight was also considered when designing the morphing airplane. By designing the airplane to be as light as possible, the morphing airplane would generate lift at lower air speeds than if it were heavier. Therefore wind-tunnel testing could be accomplished at slower speeds.

2.2 Overall Design

The morphing airplane designed and constructed is able to accomplish seven independent planform configuration changes. With these seven dofs, the airplane can change span, sweep its wings, twist the outboard wings, and extend and contract the tail. From these capabilities, the optimal planform configuration for different mission requirements/profiles can be analyzed through wind-tunnel testing. In addition to looking at the benefits of static morphing for multi-mission objectives, improving the airplanes maneuverability over fixed wing aircraft will be investigated. Maneuverability of the airplane will be accomplished using symmetric or anti-symmetric dynamic morphing to achieve a desired flight characteristic. By comparing the different morphing configurations for dynamic maneuvering in the wind tunnel, the optimized profile can be determined. For example, what is the best way to achieve a roll rate, through twisting the outboard wings, using span extension, sweeping a single wing or is it a combination of any of these? Besides determining the optimal planform configuration to achieve a desired flight characteristic, another comparison can be made by measuring the energy required to generate the desired morphing change.

For the morphing airplane designed, the overall size of the airplane at its base configuration, wings unswept, span and tail unextended, is a length of 42.75 in. from tip to tail and has a wing span of 46.5 in. At the fully extended configuration, the airplane's length is 49 in. and has a wing span of 61.5 in. This gives a percent fuselage chord change of 12%, while the wing span changes 35%. The wings can sweep 40° and the outboard wings can twist ±20.25°. With all this morphing potential, the airplane's aspect ratios ranges from 1.4 to 3.25, this is a change in AR of 131% and a 32% increase in planform area (span, sweep and tail). Two of DARPA's morphing requirements are not met, change in AR of 200% and a change in wing area of 50%. However, if the model's wing span was not limited by the wind-tunnel dimensions, all of DARPA's requirements could easily have been met with a larger span change.

2.3 Span fabrication and design

A morphing airplane with the ability to change its span for static or dynamic uses allows the airplane to go from a loitering (reconnaissance) configuration, span out, to an attack configuration, span in. Using the span change dynamically, the airplane could generate a variety of roll rates, depending on the difference in wing extension of each wing. By designing independent span control for the morphing airplane test platform, both the static and dynamic cases will be investigated. The span of the airplane, unswept, can range from 46.52 in. to 61.52 in. and can fully extend within two seconds. Each wing of the airplane was designed only to extend and contract 8 in. due to the wind-tunnel size constraint. To achieve the 8 inches in span, a pneumatic actuator is used.

2.3.1 Span Pneumatic Actuator

Pneumatics were chosen to actuate the span due to their large force to weight ratio. The pneumatic span actuators were position feedback cylinders from Bimba. With an 8 in. stroke, the actuators are capable of an accuracy of ±0.08 in. The linear resistive transducer (LRT) or potentiometer mounted within the Bimba actuators is capable of position feedback repeatability of 0.001 in. In addition to being highly accurate, the actuator is capable of applying a force 1.7 times the air pressure. During implementation of the control for the span actuators, an air pressure of 70 psi provided the most control authority. Therefore the force generated by the span actuator is 119 lbs. To supply the span actuator with pressurized air, an external air tank is used to supply 90 psi to the airplane. Since the span actuators are best controlled at 70 psi flow control valves were used to regulate the flow to the desired pressure before being supplied to the solenoid valve.

2.3.2 Solenoid Valves

To control the pneumatic span actuators, a 3 position closed center solenoid valve was integrated between the flow control valve and actuator. A three-position closed center solenoid valve is needed to control the span extension, retraction and to hold the span actuator at any desired mid-point position. Another benefit of having the valve closed center helps prevent the actuator from yielding to the external forces acting on the system by maintaining pressure in the actuator when the valve is shut off. The valve draws 0.55 Watts at 24 VDC with a response time of 20 ms and operates between pressures of 30-100 psi with a max scfm of 12 at 100 psi. With such a fast response time, a proportional-derivate controller was designed to control the span.

Designing the ability for the morphing airplane to vary span, changes in wing area of 43.3% can be obtained. Such a large area change will allow the airplane to go from a loitering configuration, full extension, to an attack configuration, wings contracted. In addition to extending and contracting the wings symmetrically, anti-symmetric span changes could be implemented for flight control. For instance, by extending one wing out farther than the other, a rolling effect would be generated. Depending on the difference in wing span for each side of the airplane, the roll rate would be affected. A picture of the span and tail extended and contracted is shown in Figure 2.1

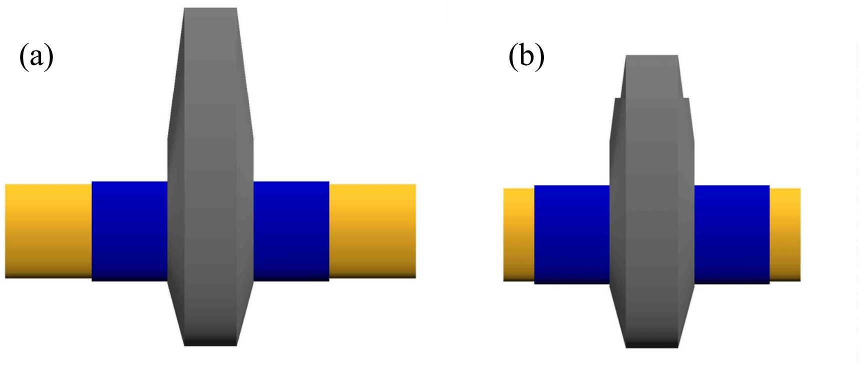

Figure 2.1: Top view of a) Span and tail extension b) Span and tail contracted

2.3.3 Hollow Wings

To have a morphing airplane with span extension, hollow inboard wings needed to be designed and fabricated to be able to incorporate the span and twist actuators along with the outboard wing. The airfoil used to create the hollow wing was a modified NACA 0020-64 airfoil with a 14 in. chord. The modifications, 64, are the 6 produces a leading edge radius of a standard 4-digit airfoil, while the chordwise position of the maximum thickness is modified to 0.4. By moving the maximum thickness of the airfoil allocates enough space for the span and twist actuators. The span of the hollow inboard wing is 16.6 in.

In addition to having the hollow wings house the actuators, the airplane's ability to sweep 40° needed to be taken into account during the design. To prevent disrupting the aerodynamic flow over the wings and shell, the hollow wing was designed to always be within the fuselage shell for any angle of sweep. Figure 2.2 shows the leading edge within the fuselage shell at full sweep for the left wing and the trailing edge is within the shell for the right unswept wing. Other design considerations for the hollow wing was that it needed to be rigid enough to keep its shape under loading, be a fixed boundary condition for the twisting outboard wings and be light in weight. To maintain the wing's airfoil shape under aerodynamic loading and to provide a rigid constraint for twist, the hollow wings were designed to be made out of 1/8 in. fiberglass. Fiberglass is not only able to hold its shape under load, but it is also light in weight. Two stationary ribs, which are connected to the span actuator, are used to attach the hollow wings to the airplane.

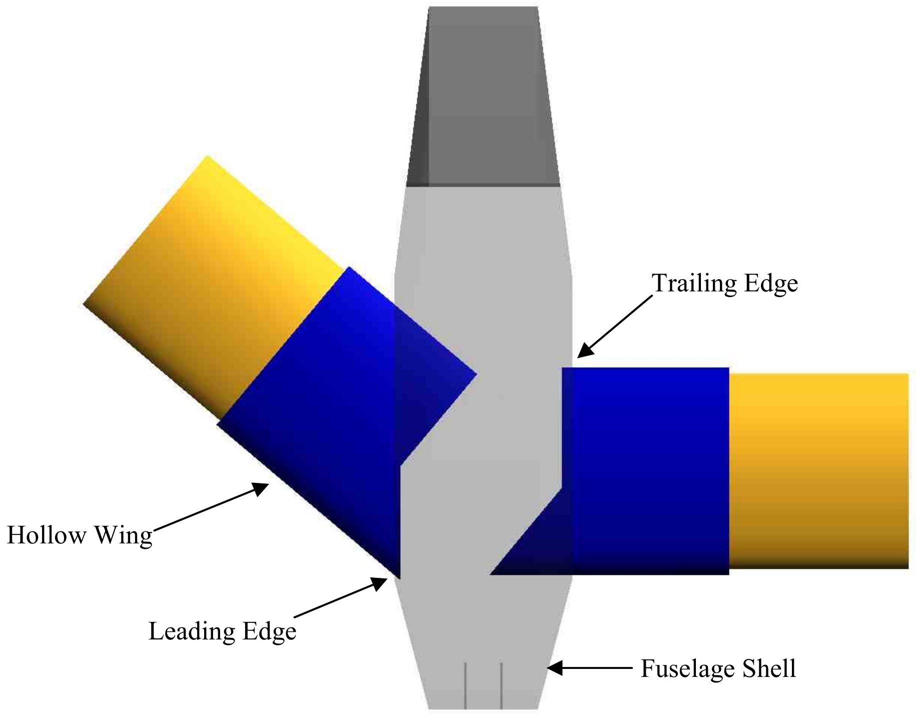

Figure 2.2: Hollow wing design to prevent gaps between fuselage shell and inboard wing for all angles of wing sweep

2.4 Twist Design and Fabrication

The objective of designing and fabricating a truly morphing airplane means eliminating all flaps and ailerons from the airplane. Therefore, to be able to roll and trim the airplane for various morphing configurations, twist was designed for the outboard wings. Using twist for trimming the airplane will allow for controlled flight for certain planform configurations. However, twist is the one capability that can set the designed morphing airplane apart from fixed wing aircraft. By using twist the airplane can generate lift by twisting its outboard wings to a positive angle of attack while the nose of the airplane is pointed at a negative angle of attack. In this configuration, nose pointed down while the airplane is climbing, offers the potential of the airplane to see and attack a target on the ground while climbing out of harms way.

Since the morphing airplane doesn’t have any ailerons, a twist mechanism was designed and assembled that will be attached to the end of the span actuator to be used for control and trimming the airplane. The twist mechanism is designed to twist the outboard wing ±20°. To show the internal structure of the twist mechanism a Solid Edge drawing along with the actual model is shown in Figure 2.3.

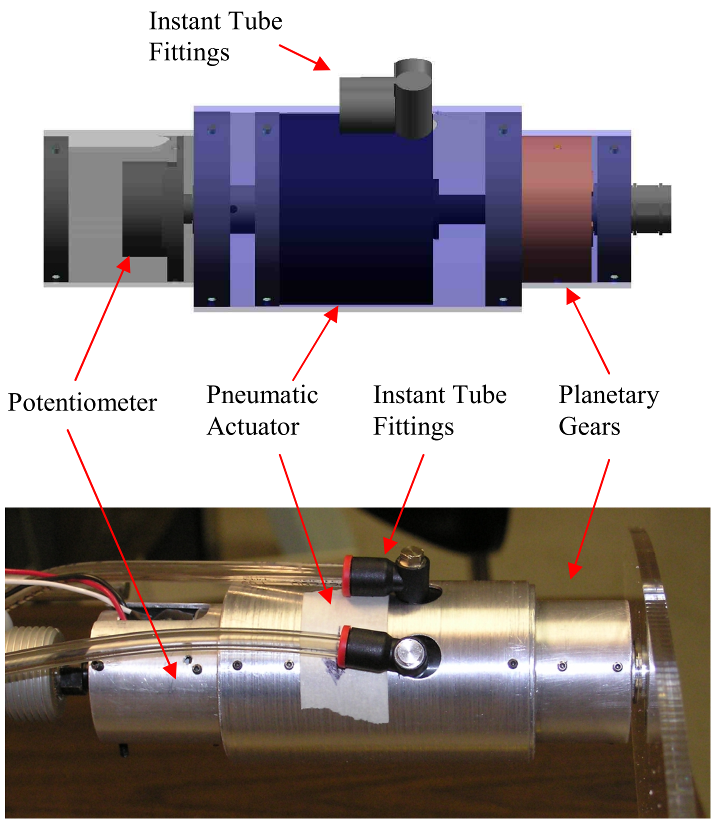

Figure 2.3: Solid Edge design and actual twist mechanism

2.4.1 Twist Actuator and Gears

The amount of torque needed to twist the outboard wing section varies as the span is extended and contracted. Therefore, the worse case scenario is when the span is contracted, resulting in only a small amount of the outboard wing to twist. This worst case scenario was used in designing the twist mechanism because it required the largest amount of torque to twist, to overcome the outboard wing's stiffness. For the twist mechanism, pneumatic rotary actuators were again chosen due to their large amount of torque output while being light in weight. However, pneumatic rotary actuators have a limited rotation. For the actuators used in the twist mechanism, the rotation is limited to 270°. At 70 psi, the torque provided by the pneumatic actuator is 15.6 in-lbs. This is not a sufficient amount of torque to twist the outboard wing sections the desired ±20°. To increase the amount of torque produced by the twist mechanism, planetary gears were added to the motor shaft output. To ensure the gears would not fail and for ease of design, the planetary gears were taken from a DeWalt cordless screwdriver. Since the planetary gears would limit output rotation by the gear ratio, only one planetary gear set was used. One stage of DeWalt's planetary gear set has a gear ratio of 1:6.67. With this gear reduction, the output torque at 70 psi is 106.6 in-lbs torque to twist the foam outboard wings, but limits the amount of twist rotation to 40.5°. As a result, the outboard wing can only twist ±20.25°.

2.4.2 Outboard Wing Design

To have a twisting outboard wing, a material had to be found that would be stiff enough to sustain the air loads without deflection and yet flexible enough so that a large amount of torque is not needed to twist it ±20.25°. The material chosen for the outboard wing section was a closed-cell polyethylene foam. The foam was cut to the shape of a modified NACA 0020-64 airfoil with a 14 in. chord length in order for it to fit within a close tolerance of the hollow inboard wing. To twist the foam with the twist mechanism, the foam will be attached to a plexiglass rib section that mounts to the end of the twist mechanism using a key. Figure 2.4 demonstrates the twisting of the outboard wing.

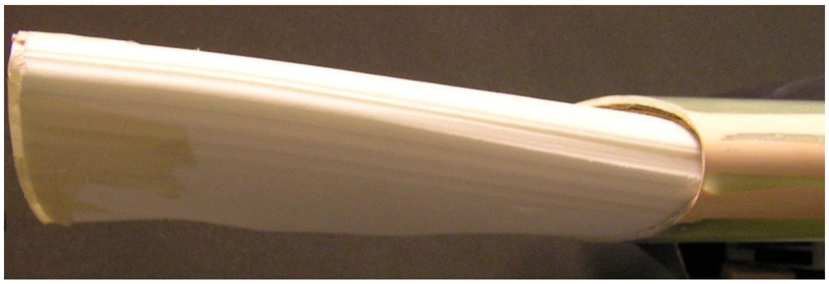

Figure 2.4: Outboard wing twisted 20°

2.4.3 Electropneumatic Regulator

To control the twist mechanism accurately, the pressure of the air supply to the solenoid values needed to be regulated. To solve this problem, off the shelf regulators were used from Marsh Bellofram, type 3110 which allow a max flow rate of 1.25 scfm through the regulator. By controlling the signal to the regulators, 0-10VDC, the pressure can be regulated to an accuracy of ±0.5%. Such precise control is needed for the twist mechanism because any deflection error can generate large aerodynamic forces.

2.5 Sweep Design and Fabrication

For a morphing UAV, being able to change the wing's sweep angle allows the airplane to go from a slow, high lift configuration to a swept wing optimized for fast flight. Using sweep as a large planform configuration change has been implemented on the F-14 Tomcat and the aerodynamic benefits are well known. However, a significant difference between the F-14 Tomcat and the morphing airplane designed here is the morphing airplanes ability to sweep each wing independently. This one change allows for an anti-symmetric planform configuration that will be used for flight control. For the morphing airplane design, sweep alone causes an area change of 7.7%.

An initial design of the sweep mechanism was to use a linear actuator along with pulleys to generate the sweeping change. This design was cumbersome and created a lot of problems trying to connect the string potentiometers and wing permanently to the wire attached to the actuator. To simplify the design, a support bar was added beside the span actuator to directly drive the sweep of the wings. This bar would also provide wing support and keep the sliding outboard wing inline with the inboard hollow wing. Figure 2.5 shows the three-bar linkage design created by the support bar along with the sweep actuator. As the wing is swept inwards around point C, the sweep actuator rotates into the fuselage about point A. By sweeping the wings, the center of gravity (*c.g.*) location of the airplane will shift. If the wings are swept back 40°, the *c.g.* location will shift towards the tail of the airplane, giving the airplane stability.

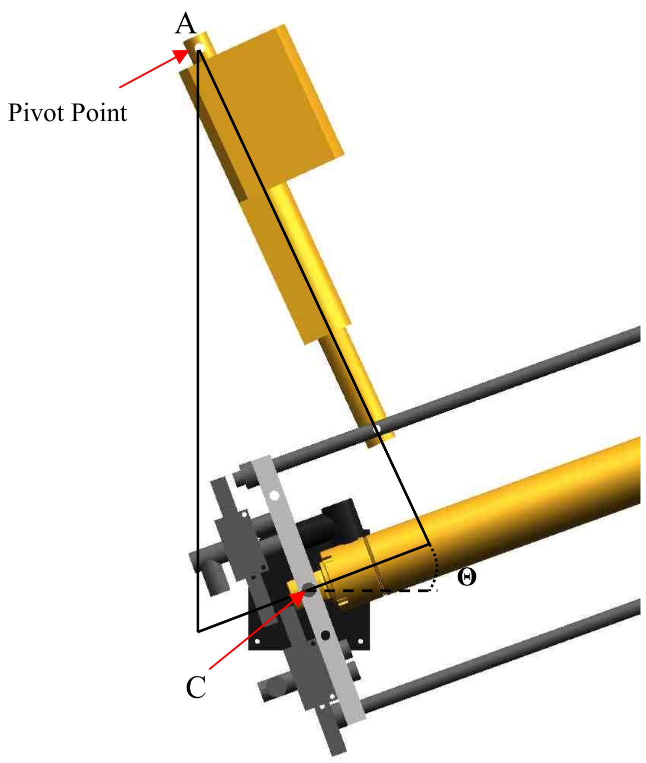

Figure 2.5: 3-Bar Linkage design for the sweep

2.5.1 Sweep Actuators

The original sweep actuators were pneumatic actuators. These actuators easily swept the wings; however, both the rate and the position were hard to control. Another major problem with pneumatic sweep actuators were the aerodynamic forces could easily overcome the internal actuator air pressure. This means that intermediate sweep angles could not be held. Therefore, to alleviate these problems, the design was changed to use nonbackdrivable electromechanical actuators. The electromechanical actuator is a four inch Acme-lead-screw driven actuator with a maximum 25 lbs push/pull force and a 25% duty cycle. The current draw of the actuator is 2.8 amps for 24 VDC. To help alleviate the friction between the rotating wing and the fuselage base, the wings are mounted on top of a turntable. With a 200 lbs load capacity and very low friction bearings, the turntable provides sweep an almost frictionless rotation. Another benefit of the turntable is it distributes the weight of the wing on the fuselage.

2.5.2 Potentiometers

Due to size limitations within the fuselage, the electromechanical actuators do not have an internal encoder. Therefore to do position feedback control, potentiometers were used to read the sweep angle directly. To ensure accurate readings from the potentiometers, the pot base had to be fixed to the fuselage base while the pot shaft had to be free to rotate. Both of these problems were solved in mounting the potentiometers, see Figure 2.6. Connecting the pot to the pivot screw of the wing was done by manufacturing an attachment to be mounted to the potentiometer shaft. The attachment is slotted with a 1/8th in. slot, while the pivot screw is machined to have an extruded key to fit the slot. Since the only force on the pot would be a rotational force, the pivot screw will be inserted into the pot attachment to transfer rotation. This means only one sensor will be used for controlling the sweep and it will be accomplished by measuring the sweep angle directly. Since the sweep angle is the only measurement, an approximation of the actuator position will be made from the sweep angle. Using this type of sensing for the sweep, accurate control of the sweep was done by using a proportional-derivative controller. The sweep angle can be controlled for ±0.3° while the sweep travels 30° per second.

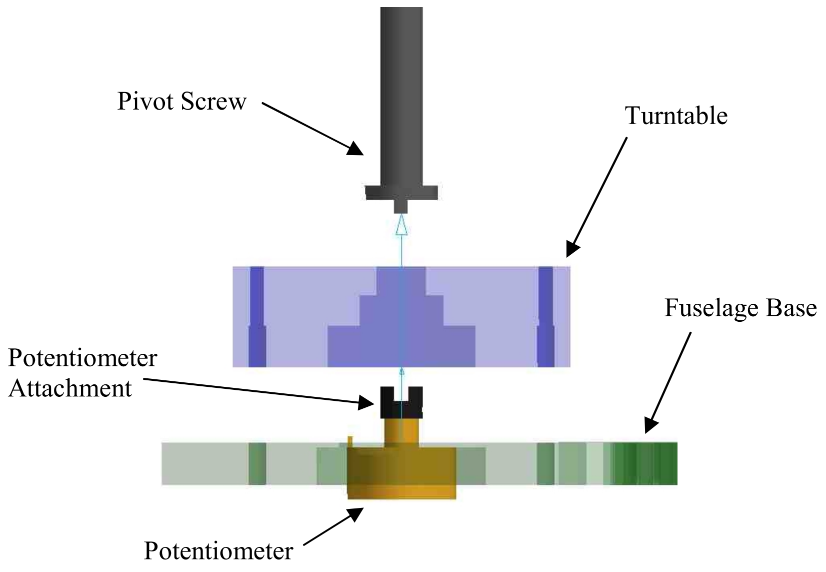

Figure 2.6: Sweep potentiometer mount

2.6 Wing Deflection Analysis

Before implementing the wing design, a simple analysis was completed to determine the deflection of the wing under aerodynamic loading. The design of the wing has the span actuator with the twist mechanism mounted to the tip. This wing design can simply be modeled as a cantilever beam with three different cross sectional areas, the span cylinder, the span rod and the twist mechanism, see Figure 2.7. For comparison to the real wing, Figure 2.8 shows the internal wing assembly. The loads occurring on the wing assembly are due to the aerodynamic lift forces, W_{aero}, the weight of the span actuator, W_{span}, the weight of the twist actuator, W_{twist}, and a fictitious load, P. For cantilever beams under distributed loading, it is known that the maximum displacement will occur at the beam's tip. Therefore, a fictitious load, P, will be placed at the tip of the twist mechanism, so the tip displacement can be calculated. To calculate the deflection, Castigliano's Theorem for beam deflection was used. The formula for Castigliano's Theorem is,

$$\delta_i = \int \left(\frac{M}{EI} \right) \frac{\partial M}{\partial P_i} dx \tag{2.1}$$

where δ_i is the deflection anywhere along the beam, M is the moment force, E is the Young's modulus and I is the moment of inertia. To use Castigliano's Theorem, three moment equations needed to be formed. The span cylinder and span rod were assumed to have the same weight and aerodynamic forces applied to them, therefore, the moment equations for the two will be the same. Thus there are only two moment equations, and they are

$$M_{twist} = (W_{aero} - W_{twist}) * \frac{x^2}{2} - Px \tag{2.2}$$

$$M_{span\,cylinder\,/\,rod} = L_3(W_{aero} - W_{twist})(x - \frac{L_3}{2}) + \frac{1}{2}(W_{aero} - W_{span})(L_3 - x)^2 - Px \tag{2.3}$$

where L_3 is the length of the twist mechanism and x is an arbitrary distance along the wing. For Equation 2.1, the $\partial M / \partial P$ needs to be calculated for Equations 2.2 and 2.3. Both equations generate the same result,

$$\frac{\partial M}{\partial P} = -x \, . \tag{2.4}$$

Before using Castigliano's Theorem, an assumption was made that each wing section would have the same Young's Modulus. This assumption is valid because the materials used to make the span actuator and twist mechanism have similar material properties. In addition to using the same Young's Modulus for the wing sections, for the analysis a wing loading of 5.3 psf was used, since this is what the loading will be during wind-tunnel testing. Using Castigliano's Theorem, the maximum displacement occurring at the tip is 0.0391 in. With such a small deflection under the wing loading of 5.3 psf, the tip deflection is negligible.

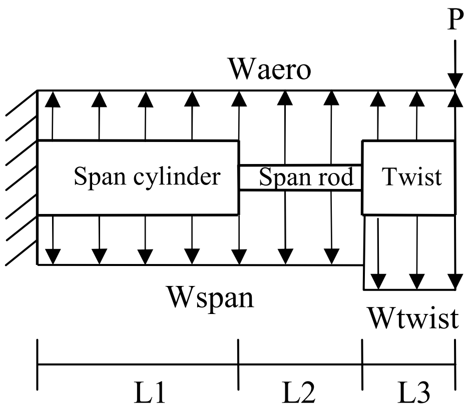

Figure 2.7: Wing model used to calculate wing deflection due to aerodynamic forces

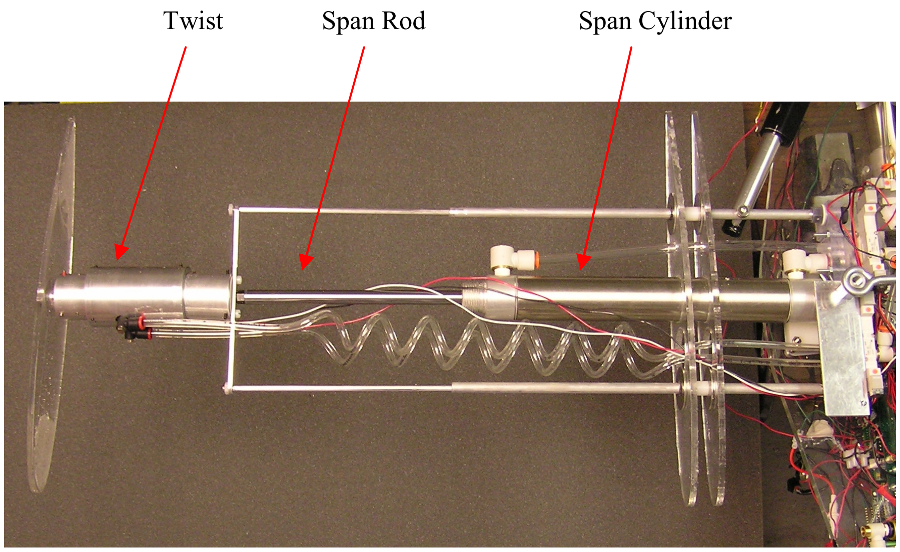

Figure 2.8: Top view of wing

Figure 2.9 shows the bending moment diagram for the wing model. The maximum bending moment occurs where the wing mounts to the base, which is expected. Figure 2.10, plots the stress due to bending in the wing. Figure 2.10 shows the maximum bending stress, $5x10^3$ psi, occurs at the connection point between the span cylinder and the span rod. This is expected since the span rod has the smallest diameter, which means that section will have higher internal stresses. To calculate the factor of safety (FOS), the exact material used to make the span actuator needs to be known. However, this is not the case, so from Norton's Machine Design book, the yield strength value for stainless steel ranges from 35 to 275 kpsi (Norton, 1998). For a conservative analysis, the smallest yield strength value for stainless steel of 35 kpsi was used and it results in a FOS of 7. Failure of the wing due to the bending stress will not occur under a wing loading of 5.3 psf. In addition, the FOS calculation is very conservative, since the actuator is stainless steel with chrome plating. The actual yield strength will be a lot larger than 35 kpsi.

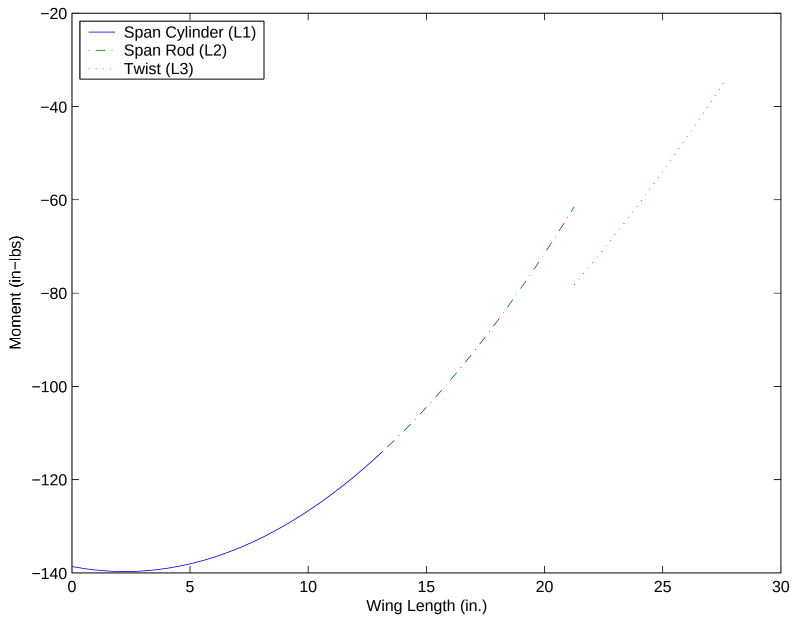

Figure 2.9: Bending Moment Diagram of the wing

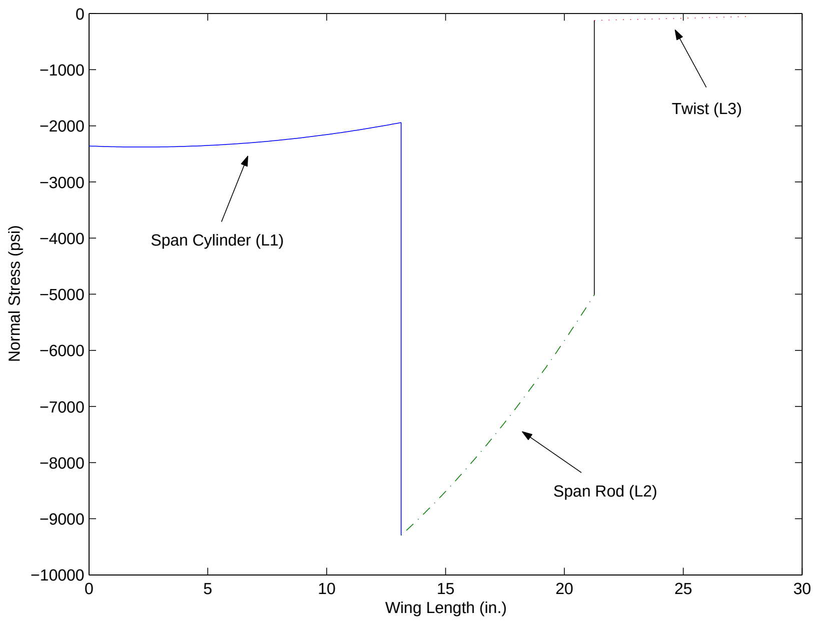

Figure 2.10: Distribution of normal stress under loading conditions

To relieve the fuselage base from handling the large moment generated by the wing due to aerodynamic forces, a turnbuckle mounts between two pivot screws that are connected to each wing; Figure 2.11. Another benefit of the turnbuckle is it prevents the wings from sagging due to slop in the turntable.

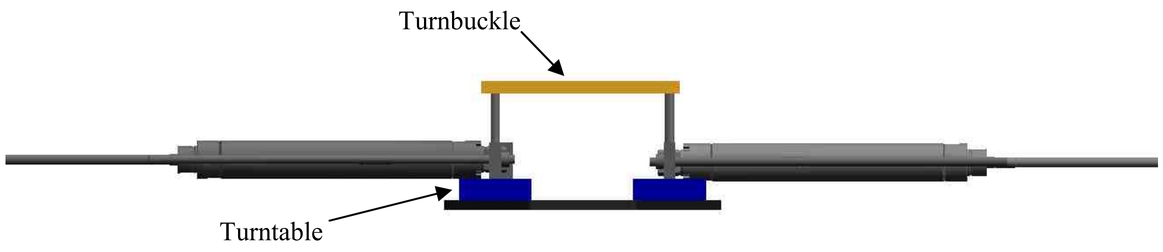

Figure 2.11: Wing connection using a turnbuckle

2.7 Tail Design and Fabrication

The tail is designed to extend and contract 6 in. and provides a 12% change in fuselage chord length. By changing the morphing airplane's chord length, the *c.g.* of the morphing airplane will shift in relation to the tail. As the *c.g.* location shifts aft, the stability of the airplane will increase. Another benefit of the tail extension/contraction is this is the only pure planform area change morphing capability the airplane has. Figure 2.1 demonstrates the tails fuselage extension capabilities.

2.7.1 Tail Pneumatic Actuator

Actuation of the tail is completed by a bi-directional 6 in. pneumatic actuator. The position feedback pneumatic actuator used for the tail is a smaller version of the pneumatic actuators used for the span. A stroke length of 6 in was selected so the actuator wouldn't interfere with other internal components within the morphing airplane's fuselage. With a 6 in. stroke, the tail actuator is capable of an accuracy of ±0.06in. In addition to being highly accurate, the actuator is capable of applying a force 0.9 times the air pressure. Therefore, with an input pressure of 90 psi being supplied to the airplane, the force produced by the tail actuator is 81 lbs.

2.7.2 Tail Regulator/Valve

To control the tail actuator, an electropneumatic regulator is used. The regulator is the same type used for the twist mechanism. In addition to the electropneumatic regulator, a solenoid valve is used to extend and contract the actuator. The solenoid valve is the same type used for the span and twist mechanisms.

2.8 Wind-Tunnel Analysis

The aerodynamic analysis was performed by Johnston and is discussed in more detail by Neal *et al* (2004). For the first wind-tunnel testing performed on the morphing airplane, the objective was to determine the affects of span, sweep and fuselage extension on the model's aerodynamic characteristics. By varying these three planform changes, the influences of these changes on the aerodynamic center location and drag force could be identified. The wind-tunnel test was performed at an airspeed of 50 mph, which produced a wing loading of 5.3 psf. For each morphing configuration, the airplane was rotated through -4° to 20° angle of attack.

2.8.1 Wind-Tunnel Results

From the testing, the aerodynamic center location shifted aft as the wings swept back for different amounts of span extension. This result was as expected and helped to verify our wind-tunnel analysis. The main result obtained from the wind-tunnel testing was the airplane's ability to reduce drag by changing its planform area. As the airplane was cycled through a large range of lift coefficients, the optimal airplane configuration changed to reduce drag. For low lift coefficient values, the optimal airplane configuration are wings unswept and unextended. Then as the lift coefficient increases to around 0.3, the optimal low drag morphing configuration are wings unswept and fully extended. Finally, as the lift coefficient gets to around 0.5, the optimal configuration is wings swept 40° and fully extended. The results are presented in Neal *et al* (2004) and are shown in Figure 2.12.

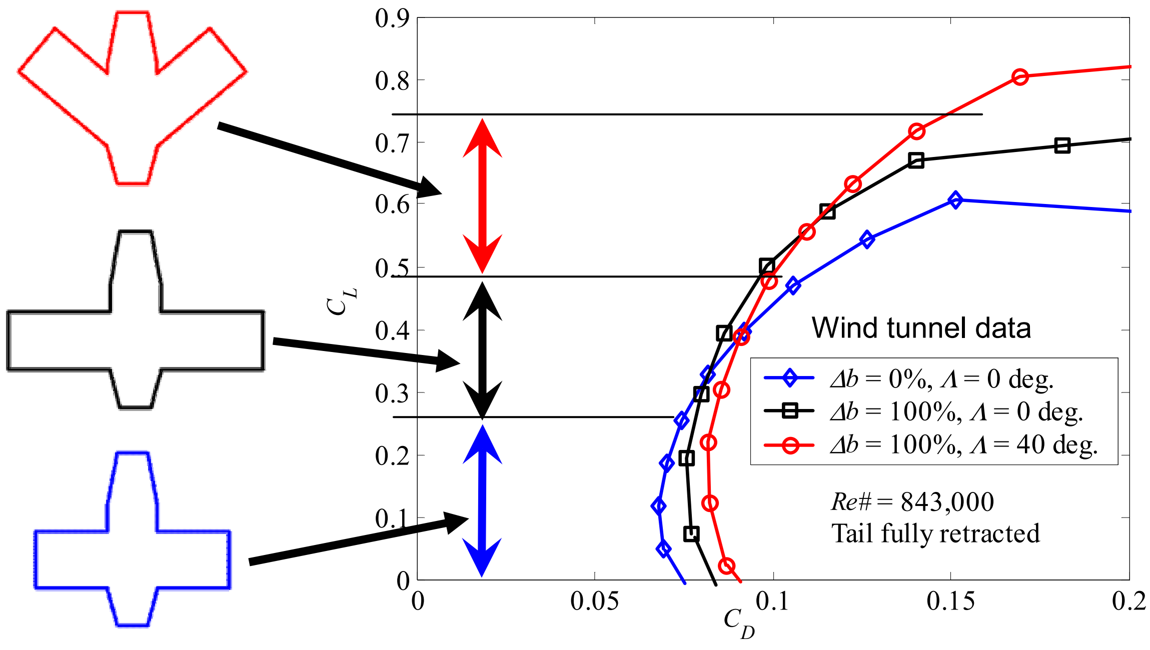

Figure 2.12: Drag reduction over a range of lift coefficients for varying morphing changes

2.9 Chapter Summary

In this chapter the design and wind-tunnel analysis of the fully adaptive aircraft configuration was discussed. Beginning with the design requirements set by DARPA for the MAS program that the morphing airplane was designed to meet. Then the span, twist and sweep designs were discussed. Following the design of the wing, an analysis was completed on the amount of aerodynamic loading the wings are capable of supporting without failure. Then the last degree of freedom designed, the tail, was discussed. The tail can be used to vary the *c.g.* and aerodynamic center location of the model airplane. Finally the wind-tunnel results were discussed and from the testing, the optimal planform configuration varies over a range of lift coefficients to optimize the drag reduction.

Chapter 3

Compliant Mechanism Design using Structural Optimization

Fixed wing aircraft currently control the pitch, roll and yaw of the airplane through the use of flaps, ailerons and a rudder. Mounting the flaps and ailerons to the wing causes a discontinuity between the airfoil and the flap or aileron. This discontinuity increases the drag of the airfoil, which decreases the effectiveness of the flap or aileron. Being able to combine the airfoil and the flap/aileron as one structure eliminates the discontinuity and produces a continuous variable camber airfoil. For the continuous surface flap to achieve similar deflections that a fixed wing flap generates, the structure needs to be compliant, yet stiff, to support itself under aerodynamic loading. Also, being able to precisely control the angle of deflections is crucial. Since most compliant mechanisms are made from a single piece of material, there is little to no backlash between joints, which allows precise control of the deflection angle. However, designing a compliant mechanism that maximizes the trailing edge tip deflection a structural optimization program needs to be created. The optimized shape of the compliant mechanism needs to be flexible to achieve the desired tip deflection and stiff enough to support itself under aerodynamic loading. In addition to providing a continuous deflectable surface, the structural optimization program will generate the desired trailing edge tip deflection using a single actuator. The linear actuator will distribute its force to two nodes of the structure to deflect the mechanism with a torque motion. This chapter will present the structural optimization program that will be used to design a variable camber compliant tail.

3.1 Analytical Model

The purpose of designing a variable camber tail is to mount the design to the morphing airplane discussed in Chapter 2. Not only will this give the morphing airplane an additional degree-of-freedom, but the variable camber tail will provide pitch control. Since the compliant tail will be mounted to the airplane, the tail size is already constrained by the fuselage design. For the morphing airplane design, the fuselage section is a NACA-0017-64 with a 49 in. chord length, while the tail section is 12.5 in. of the chord. In order to mount the compliant tail to the airplane, a decision had to be made on where to choose the boundary conditions for the tail. The boundary conditions were chosen so the tail simulated a cantilever beam, as shown in Figure 3.1. Selecting the boundary conditions at the end of the tail allowed the compliant mechanism to be easily mounted to the fuselage base.

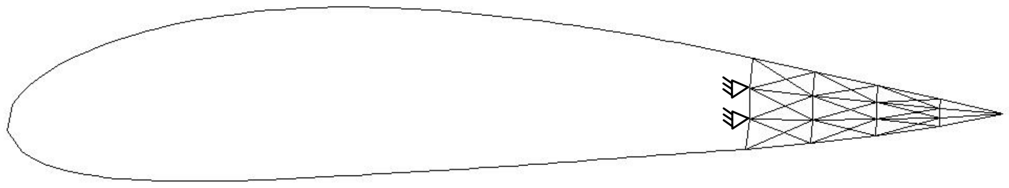

Figure 3.1: Fuselage design with discretized tail and boundary conditions

3.2 Optimization Formulation

To formulate the problem of shape change, the optimization problem is modeled as maximizing the vertical trailing edge deflection. However, if the optimization problem is only formulated to maximize the trailing edge tip deflection, the optimized structure may be too flexible to withstand the aerodynamic forces applied to it. Therefore, a multi-criteria optimization problem needs to be developed. The optimal structure needs to be flexible to maximize the tip deflection, but stiff enough to minimize the deflection of the entire tail airfoil section due to aerodynamic loading. Therefore the objective function is to maximize the i) vertical tip deflection due to an input force and ii) minimize the vertical tip deflection due to the aerodynamic loading. The constraints of the optimization problem are formulated as the combination of the stress of each member due to the input force and the stress due to the aerodynamic loading. The optimization problem can be represented as follows,

$$\max_{h_j} \quad u_{Tip} \tag{3.1}$$

subject to:

$$|\sigma_i| \le \sigma_{allowable} \quad i = 1 \ldots m \tag{3.2}$$

$$h_{lower} \le h_j \le h_{upper} \quad j = 1 \ldots n \tag{3.3}$$

where u_{Tip} is the vertical tip displacement due to the input force and the aerodynamic loading, σ_i is the stress of each truss member, i is the number of constraints, $\sigma_{allowable}$ is the allowable stress, m is the number of constraints, h_j is the design variable, height of the beam members, j is the number of design variables (DV), h_{lower} is the lower bound of the design variables and h_{upper} is the upper bound of the design variables.

Maximizing the trailing edge tip displacement subject to stress constraints yielded undesired results. As the tip displacement is maximized, the design variables can either increase or decrease within their upper and lower bounds. Once the optimal shape of the structure is found, the members with the design variables equaled to the lower bound are supposed to be removed because they do not contribute to maximizing the objective function. The stress within the member should also go to zero as the height goes towards the lower bound, since this would allow removal of the members without affecting the rest of the structure. However, with stress as constraints, this is not the case, as the member's heights decrease, the stress in the member increases. Also, since the stress is distributed about the structure, removal of members causes the stress to be redistributed. Some members are more critical than others to the structure and removal of any of these critical members will increase the stress on the other members to the point were they may violate the stress constraints. More discussion on this topic is presented in Chapter 4.

3.3 Progression of Structural Optimization

Using structural optimization, the variable camber tail topology was optimized. Figure 3.2 is a generalized flowchart of the structural optimization program written to optimize the compliant tail shape. The ground structure and initialization of the design variables was accomplished using a Matlab program that is shown in Appendix A. The Matlab program generates an ANSYS log file by discretizing an airfoil tail section into the desired number of elements. ANSYS is then run by this log file that is sent by the main C program. Having the main code written in C allows the program to integrate ANSYS, a thin airfoil theory code and an optimization program in order to optimize the shape of the initial ground structure.

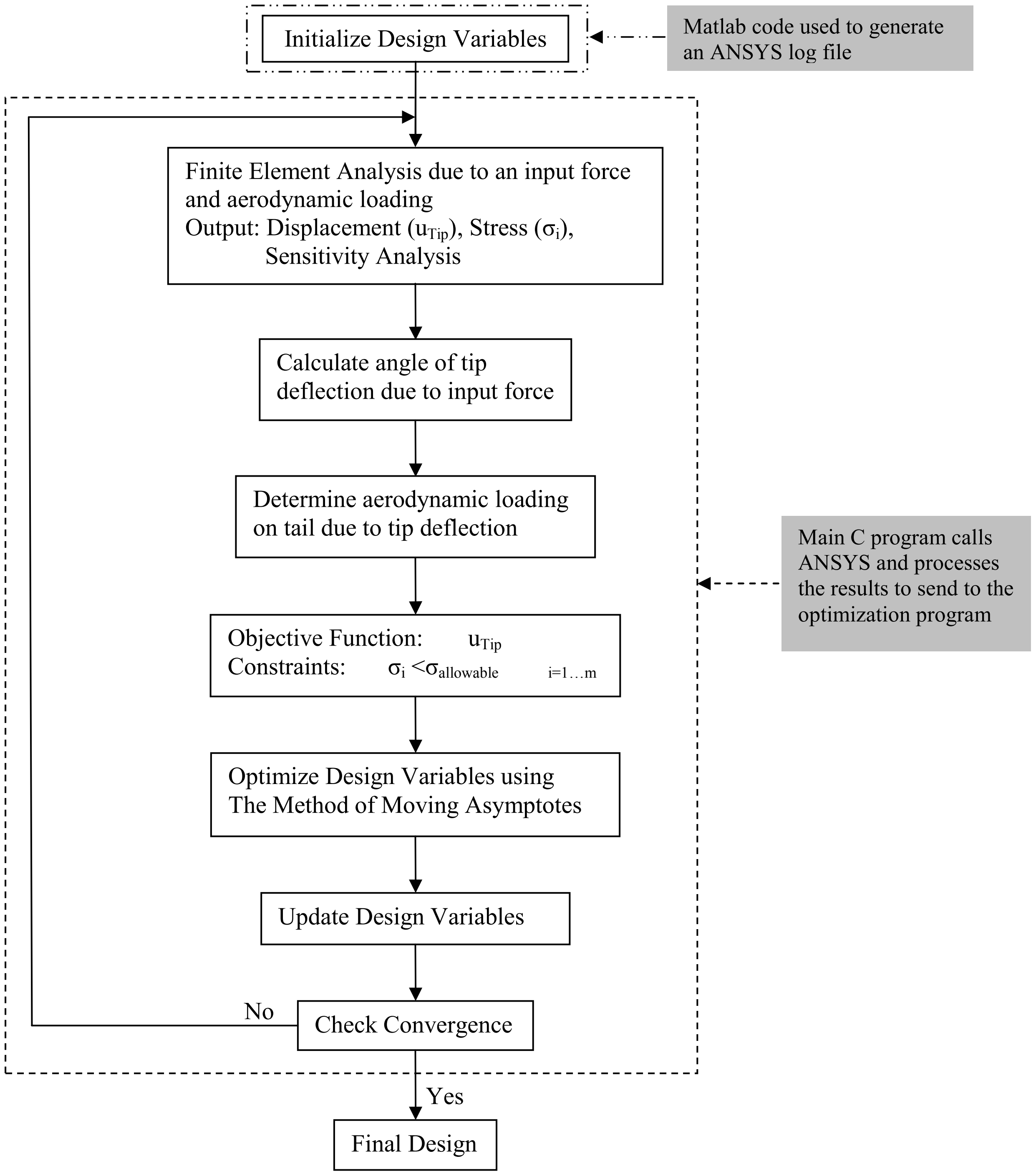
Initialize Design Variables
Matlab code used to generate an ANSYS log file
Finite Element Analysis due to an input force and aerodynamic loading
Output: Displacement (u_{Tip}), Stress (σ_i), Sensitivity Analysis
Calculate angle of tip deflection due to input force
Determine aerodynamic loading on tail due to tip deflection
Main C program calls ANSYS and processes the results to send to the optimization program
Objective Function: u_{Tip}
Constraints: $\sigma_i < \sigma_{allowable}$ $i=1\ldots m$
Optimize Design Variables using The Method of Moving Asymptotes
Update Design Variables
No
Check Convergence
Yes
Final Design

Figure 3.2: Flowchart of the structural optimization program

3.3.1 Initializing the Design Variables

Before running the structural optimization program, the design variables need to be initialized. The design variables in the optimization program are the height of each truss or beam member; the program can handle either element. In most compliant optimization programs, the design variables are the cross-sectional areas of the truss or beam members. However, since the compliant tail was going to be mounted to the morphing airplane, the cost and ease of manufacturing the part needed to be taken into consideration. The simplest and most cost effective way to manufacture the compliant mechanism was using a two-dimensional laser cutter. Therefore, the thickness of the part was held constant, while the height of the truss/beam members would vary.

In addition to initializing the design variables, a ground structure or mesh of the design space is necessary. The size of the mesh will depend on the computer speed and desired completeness of the final topology shape. To reduce the amount of computational time of the objective function in the finite element code, only a partial or coarse ground structure was developed. However, the major drawback to a coarse mesh is that the mesh may not include an optimal beam member link. Since the code can not generate beam members, it will select beams that closely mimic the desired shape. For instances, with a coarse mesh, the optimal shaped developed may have beams connected with right angles, while what the optimal shape may be the diagonal beam member connecting the two nodes. To generate the initialize ground structure, a Matlab code was written that discretizes an airfoil shape and generates an ANSYS log file.

3.3.2 Finite Element Analysis

Once Matlab generates the ANSYS log file, the file is sent to ANSYS for analysis. ANSYS was used for the finite element analysis because of the wide range of capabilities it offers compared to developing your own finite code. ANSYS has the capability of calculating the displacement, stresses and the sensitivity information all in one call to the program, which is done using the log file generated by Matlab. However, ANSYS can only handle 60 design variables, which limits the mesh size of the structure.

To have ANSYS calculate the tip displacement due to the input force and the aerodynamic forces, for the first iteration, the deflection angle is assumed to be zero, as a result the aerodynamic forces acting on the structure are zero. Once the first ANSYS analysis of the structure is completed, the angle of tip deflection is calculated based on the deflection of the structure. This angle will be used to apply the aerodynamic forces to the structure during the next iteration of the structural optimization program. Therefore, the aerodynamic forces are always an iteration behind the actual tip displacement. Having the aerodynamic forces an iteration behind does not cause a problem because as the structure converges to its optimal shape, the tip deflection will vary less than .005 in or 0.02°. Therefore, the aerodynamic forces acting on the structure will be almost exactly the same for each iteration as the program converges. ANSYS then outputs the vertical tip displacement, element stresses and gradient information to the main C program to be used for input into the optimization program. The main C program is shown in Appendix B. The main C program then processes the ANSYS analysis to calculate the aerodynamic forces acting on the structure along with the information to send to the optimization program.

3.3.3 Aerodynamic Forces

As the structural optimization program runs through its iterations, the trailing edge tip displacement will vary after each iteration, which causes the aerodynamic forces to change as well. As stated in the previous section, the reflex angle is calculated from the vertical tip deflection produced by the input force. Knowing the angle of deflection, the chord length of the airfoil and the airspeed, the aerodynamic forces acting on the tail section are generated using a thin airfoil theory program developed by Johnston (2003). Thin airfoil theory uses a thin airfoil and calculates the pressure difference between the upper and lower surface of the airfoil. Figure 3.3 shows the distributed aerodynamic pressure, Δp on the airfoil surface for the NACA-0017-64 airfoil at 0° angle of attack and a tail deflection of 10°. The aerodynamic pressure is calculated using Equation 3.4 and 3.5. The aerodynamic load can then be calculated for any point on the tail by multiplying the surface pressure by the surface area it is acting on. The formula for the dynamic pressure is,

$$q_\infty = \frac{1}{2}\rho U_\infty^2 \tag{3.4}$$

where q_∞ is the dynamic pressure, ρ is the air density and U_∞ is the free stream velocity. The free stream velocity was set to 50 mph because that is the wind-tunnel speed the morphing airplane was test at. The aerodynamic pressure can then be calculated by

$$\Delta p = q_\infty \Delta C_p(x) \tag{3.5}$$

where $\Delta C_p(x)$ is the pressure coefficient with respect to the position along the chord.

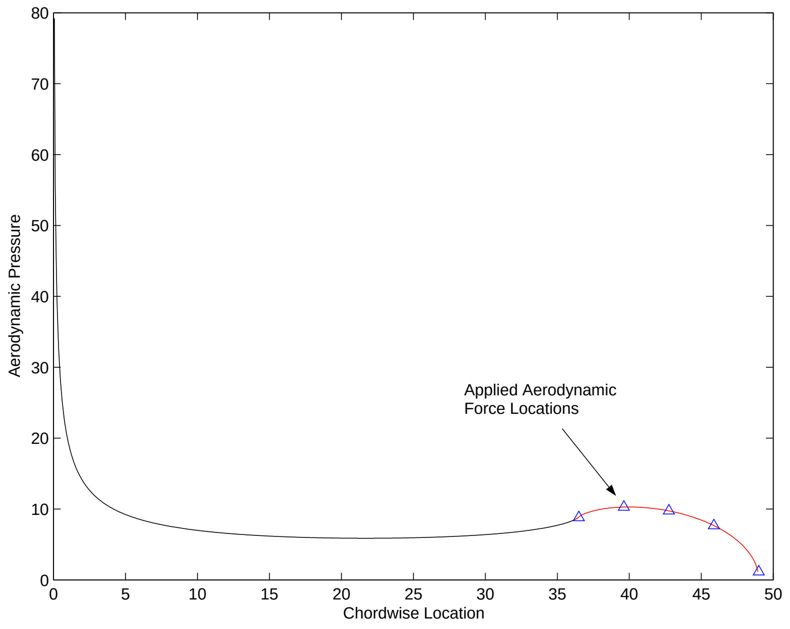

Figure 3.3: Distributed pressure along the top airfoil surface

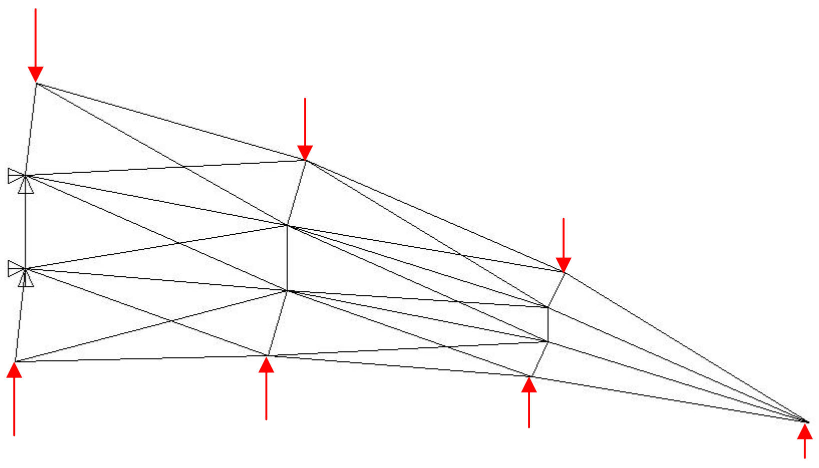

Figure 3.4: Aerodynamic load distribution acting on the tail section

3.3.4 Method of Moving Asymptotes

The optimization method chosen to optimize the design variables was the Method of Moving Asymptotes (MMA), which was developed by Svanberg (1987). The MMA program used in this research was developed by Svanberg for research use only and a contact list on how to obtain the program from him is listed in Appendix C. MMA is a variation of linear and reciprocal programming and is widely used in structural optimization problems. Linear programming approximates the objective function and constraints with a linear approximation, however, an optimal solution is not guarunteed for that approximation. Reciprocal programming, approximates the objective function and constraints with a reciprocal, however, an optimal candidate may be a saddle point, not a minimum. The Method of Moving Asymptotes is based on approximating the objective function and constraints with convex functions about the design point. The benefit of a convex function is that a single optimum is guaranteed as long as there is a feasible design area. The general approach to attacking optimization problems using MMA, is listed below.

Step 0 : Choose a starting point $x^{(o)}$, and let the iteration index k = 0

Step I : Given an iteration point $x^{(k)}$, calculate $f_i\left(x^{(k)}\right)$ and the gradients $\nabla f_i\left(x^{(k)}\right)$ for i = 0, 1,..., m

Step II : Generate a subproblem $P^{(k)}$ by replacing, in P, the (usually implicit) functions, f_i by approximating explicit functions $f_i^{(k)}$, based on the calculations from step I.

Step III : Solve $P^{(k)}$ and let the optimal solution of this subproblem be the next iteration point $x^{(k+1)}$. Let k = k + 1 and go to step I

The approximated function, f_i^k has a specific form

$$f_i^{(k)}(x) = r_i^{(k)} + \sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{U_j^{(k)} - x_j} + \frac{q_{ij}^{(k)}}{x_j - L_j^{(k)}} \right) \tag{3.6}$$

where

$$p_{ij}^{(k)} = \begin{cases} \left(U_j^{(k)} - x_j^{(k)}\right)^2 \dfrac{\partial f_i}{\partial x_j}, & if\ \dfrac{\partial f_i}{\partial x_j} > 0 \\ 0, & if\ \dfrac{\partial f_i}{\partial x_j} \le 0 \end{cases} \tag{3.7}$$

$$q_{ij}^{(k)} = \begin{cases} 0, & if\ \dfrac{\partial f_i}{\partial x_j} \ge 0 \\ -\left(x_j^{(k)} - L_j^{(k)}\right)^2 \dfrac{\partial f_i}{\partial x_j}, & if\ \dfrac{\partial f_i}{\partial x_j} < 0 \end{cases} \tag{3.8}$$

$$r_i^{(k)} = f_i\left(x^{(k)}\right) - \sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{U_j^{(k)} - x_j^{(k)}} + \frac{q_{ij}^{(k)}}{x_j^{(k)} - L_j^{(k)}} \right) \tag{3.9}$$

where $f_i\left(x^{(k)}\right)$ is the objective function and constraint approximations and $U_j^{(k)}$ and $L_j^{(k)}$ are the upper and lower moving asymptotes. By varying *U* and *L*, the optimization method codes in convergence properties. For example, if the objective function starts to oscillate, *L* and *U* will converge closer to the current iteration point to stabilize the process. However, if the objective function results are slowly improving, the moving asymptotes will expand farther from the current iteration point to help speed up the convergence process. Step II takes the approximated convex objective function and constraints about the design point and forms a subproblem, *P*, which is a separable, convex approximation of the objective function and constraints. Therefore, since the subproblem is separable and convex, the dual method is used to solve the subproblem.

The benefit of the duality method is that it is an efficient way to solve optimization problems with a large number of design variables compared to a small number of constraints. In addition to using the duality method to efficiently solve the optimization problem, faster convergence is also achieved by determining the gradient information of the objective function and constraints.

To demonstrate the Method of Moving Asymptotes, a simple example with 2 design variables will be used, so the solution can be shown graphically. The example problem is to minimize the weight of a two-bar truss with an external load applied to node 3, see Figure 3.5. The design variables are the cross-sectional area of the bars, x_1 and the half of the distance, x_2, between the two nodes, 1 and 2. The only active constraint is that the tensile stress in bar one can not be greater than a certain amount, the exact numbers are not important. For the example, both the objective function and constraint are nonlinear and are plotted in Figure 3.6 along with an initial starting point. The contour lines shown in Figure 3.6 are solutions to the objective function by varying the design variables x_1 and x_2 in the objective function. The objective function's minimum is at the origin, therefore, the contour plot decreases as the design variables converge to zero.

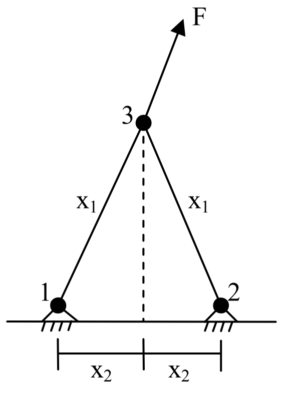

Figure 3.5: Two-bar truss problem

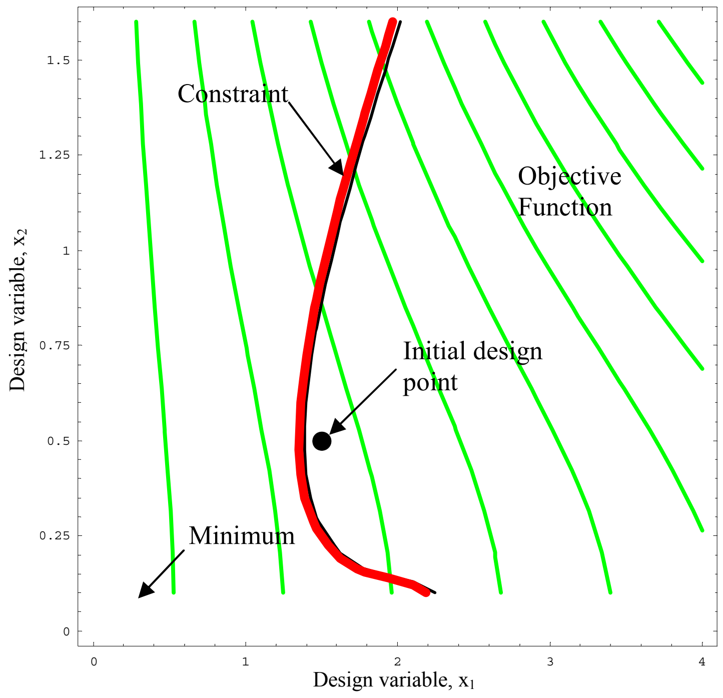

Figure 3.6: Contour plot of objective function with constraints

Using MMA, the nonlinear objective function is approximated to a convex function along with the constraints. By approximating the objective function and constraints to be a convex function about the design point, you are guaranteed an optimal point for that iteration as long there is a feasible design region. However, this does not mean that a global minimum is guaranteed. The approximated objective function and constraints, along with the optimum point for that iteration are presented in Figure 3.7.

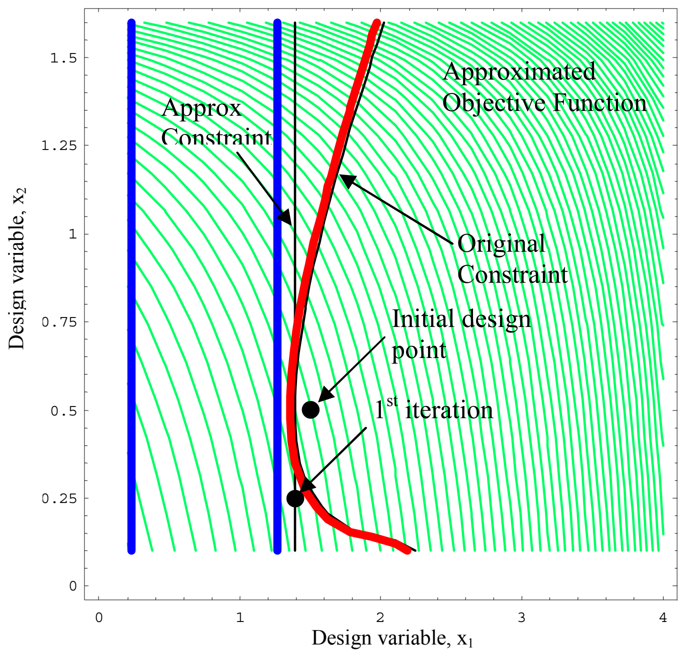

Figure 3.7: Convex approximation of objective function and constraint

Once an optimal point for that approximated function is found, the process repeats itself by approximating a new objective function and constraints about the new design point. Figure 3.8 demonstrates the iterative process by showing each iteration point until MMA converges and finds the global minimum.

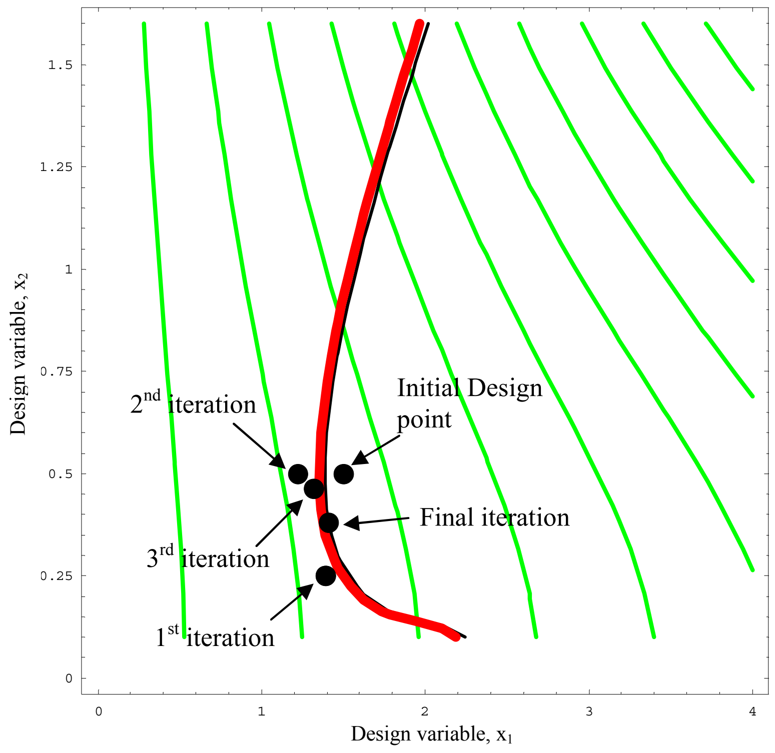

Figure 3.8: Convergence of example

3.3.5 Convergence Criteria

To determine if the topology of the structure was optimized a convergence criteria was developed. During every iteration of the program, the percent error was calculated based on the new objective function value and the previous objective function result. The formulation for the percent error is,

$$\%E_{Desired} = \frac{\left|Obj_{New} - Obj_{old}\right|}{\left|Obj_{New}\right|} \tag{3.10}$$

where *%E*$_{Desired}$ is the desired percent error value, *Obj*$_{New}$ is the new value of the objective function and *Obj*$_{old}$ is the previous iteration's objective function value. Once the percent error falls below 0.001 a counter increases by one. Through experience with running this structural optimization program, it was determined that as long as the next fifteen consecutive iterations were below the desired percent error, the structure was considered to have converged. If the calculated percent error ever went above the desired percent error, the counter was reset to zero and the new values of the design variables were looped through the structural optimization program.

3.4 Chapter Summary

The objective of this research is to design a compliant structure that will generate a continuous curved deflectable tail. Mounting the compliant mechanism to the morphing airplane will add an additional degree-of-freedom to the airplane to be used for pitch control. The steps to design the variable camber compliant tail were accomplished using structural optimization. Given a design domain, the domain was meshed using a partial ground structure and initializing the design variables. ANSYS was then used to compute the displacement, stresses and sensitivity information for each design variable due to an input force as well as to aerodynamic loading. The aerodynamic forces were determined using thin airfoil theory program that was developed by Johnston (2003). Then using the Method of Moving Asymptotes, a first order optimization method, the structure was optimized for the given design point. If the optimized structure didn't satisfy the convergence criteria, the new iteration point was sent to through the structural optimization loop again, until the convergence criteria was satisfied.

Chapter 4

Results

One of the main goals of this research was the design and implementation of a variable camber compliant tail for the morphing aircraft. In this chapter, the setup and results of the structural optimization program will be presented.

4.1 Material Selection

Material consideration is an important design aspect of compliant mechanisms. When designing a compliant mechanism, material selection is a significant decision. Depending on the mechanism's application, compliant mechanisms can be made from a variety of materials such steel, aluminum, nylon, etc. Applications of compliant mechanisms may require small predictable displacements in a harsh environment where metals would be a good material selection. However, for the variable camber tail, the application requires a flexible yet strong material to generate large displacements. To help choose a material, a simple calculation is made to determine the ratio of strength to Young's modulus. The material that will result in the largest deflection will have the highest strength to Young's modulus ratio. Table 4.1 list a variety of materials with their material properties along with the ratio. Based on the ratio of strength to Young's modulus, the material chosen to be used in the structural optimization program was Acrylonitrile Butadiene Styrene (ABS). In addition to having a large ratio, ABS can be machined at Virginia Tech using their laser cutter, so the compliant mechanism was manufactured from this material as well.

Table 4.1: Ratio of yield strength to Young's Modulus

Material	E (GPa)	S_Y (MPa)	S_Y/E*1000
Aluminum (6160-T4)	68.9	131	1.9
Dural	73	420	5.8
Steel (4140 Q&T@400)	207	1641	7.9
Nylon (type 66)	2.8	55	19.6
ABS Plastic (Molded)	2.4	71.8	29.9

4.2 Structural Optimization Setup

Having chosen the material for the compliant mechanism, a structural optimization was performed using the ground structure shown attached to the fuselage in Figure 4.1. The length of the tail section was designed to be 12.5 in. which correlates with the morphing airplane's tail section discussed in Chapter 2. Using the Matlab code attached in Appendix A, the tail section was descretized to have 46 beam members, whose heights would be the design variables. By varying the heights of the beam members, the cross-sectional areas would vary, but since the part was going to be manufactured on a two-dimensional laser cutter the thickness was held constant. Forty-six beam elements were chosen because any more divisions would put the number of design variables above ANSYS's limit of 60. The design variables were then initialized with the same randomly selected value, which was not required to be within the bound limits of the design variables shown in Table 4.2. The lower bound limit on the design variables prevent the structure from causing singularities in the stiffness matrix, if the beam heights were allowed to go to zero. While the upper bound limit prevents the beams from having to very large cross-sectional area. The input force deflecting the compliant mechanism was accomplished by using a single actuator applying a force at two locations. During the optimization run, a simulated force of 16 lbs was split between two nodes to generate a moment on the structure. The forces along with the boundary conditions are shown in Figure 4.2. Application of the input force to the compliant structure will be accomplished using the pneumatic tail actuator that was used for tail extension and contraction. The pneumatic actuator is capable of applying a force of up to 81 lbs. To help guarantee that the compliant material will not yield, the $\sigma_{allowable}$ will be equal to the yield point of the material divided by the factor safety, which for this simulation was set to 1.4. Convergence of the structural optimization program occurred when the value of the objective function did not vary more than 0.001 in. for 15 straight iterations.

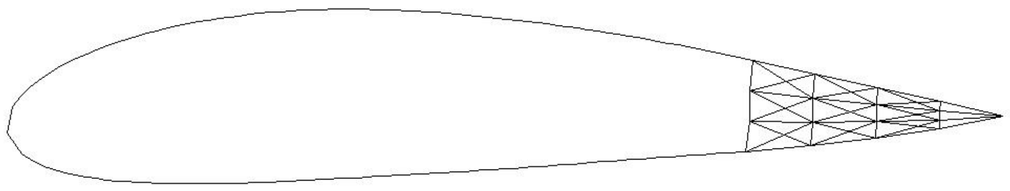

Figure 4.1: Discretized tail with 46 design variables attached to fuselage airfoil

Table 4.2: Upper and lower bounds on the design variables

Constraints	Value
h_{min}	$5.0\text{x}10^{-3}$ in
h_{max}	$6.25\text{x}10^{-2}$ in

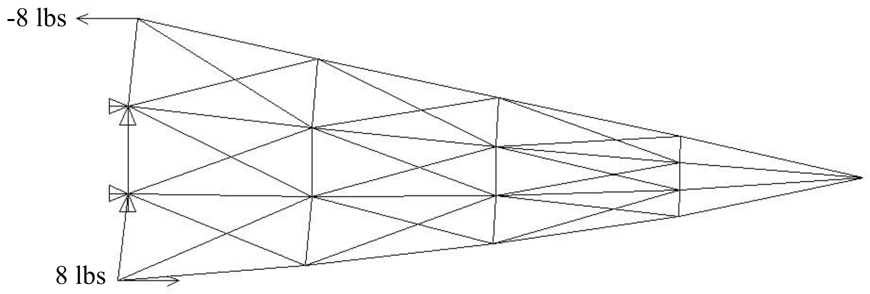

Figure 4.2: Input forces applied to tail structure

4.3 Optimized Compliant Tail Structure

The structural optimization program was run with three different initial values for the design variables. For two of the runs the initial values were within the bounds of the design variables while the other run was outside of the bound. By running the structural optimization program from a variety of initial points and having all three reach the same optimal solution helps ensure that a global minimum was found, rather than a local minimum. Figure 4.3, shows the objective function for three runs of the structural optimization program with different initial values of the design variables. In addition to just looking at the objective function results to verify convergence, the design variable values were reviewed to guarantee that they had converged as well and were not varying significantly between iterations. The results for the design variables for all three iterations are presented in Appendix D.

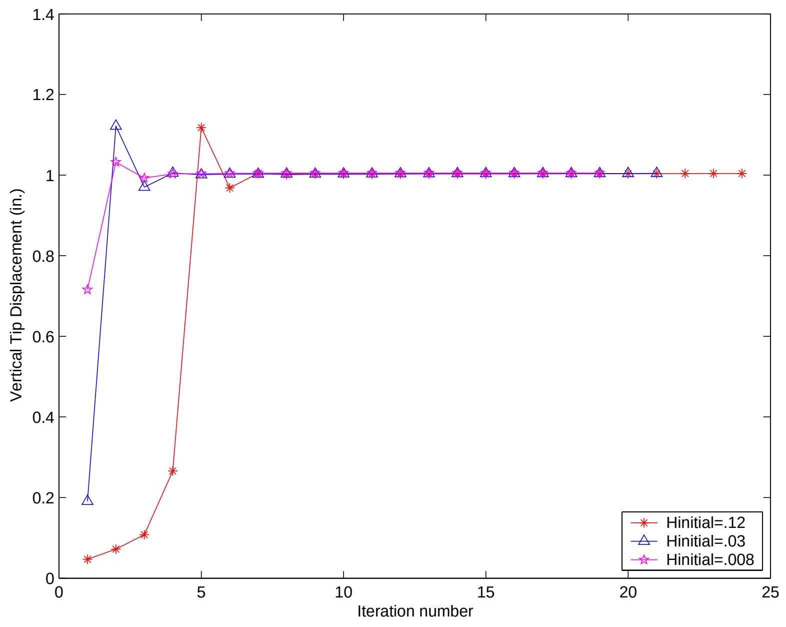

Figure 4.3: Objective function results for three separate optimization runs

All three runs, as well as other runs not presented, all converged to a vertical tip displacement of ±1 in. or an angle of deflection of ±4.57°. When the initial guess of the design variables, $h_{initial}=0.03$ or $0.008\ in.$ were within the upper and lower bounds of the design variables, the program converged quickly. However, when $h_{initial}=0.12\ in.$, which was outside of the design variables limits, the program took a few more iterations before convergence of the structure occurred. In all three cases, the vertical tip displacement reaches a peak value around 1.15 in. for two of the cases and 1.05 in. for $h_{initial}=0.008$. However, at these maximum displacement values, the constraints are violated, as shown in Figure 4.4. To guarantee that the material doesn't yield under deflection, a factor of safety of 1.4 was used for the constraint. For all three runs as well as other runs not presented, the stress constraints for each beam member were satisfied. The stress acting on the ground structure for the optimal shape is presented in Figure 4.5. Figure 4.6 displays the optimal volume of each member.

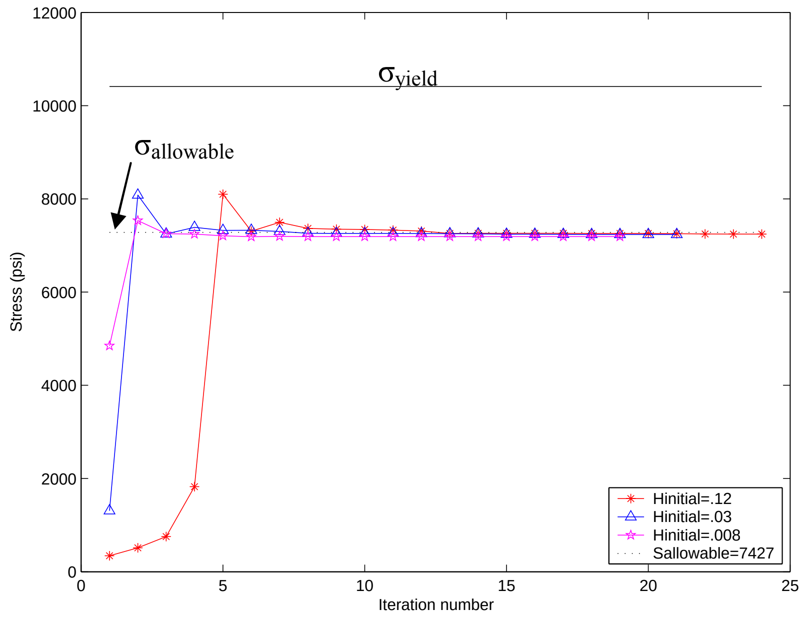

Figure 4.4: Stress constraints results for three separate optimization runs

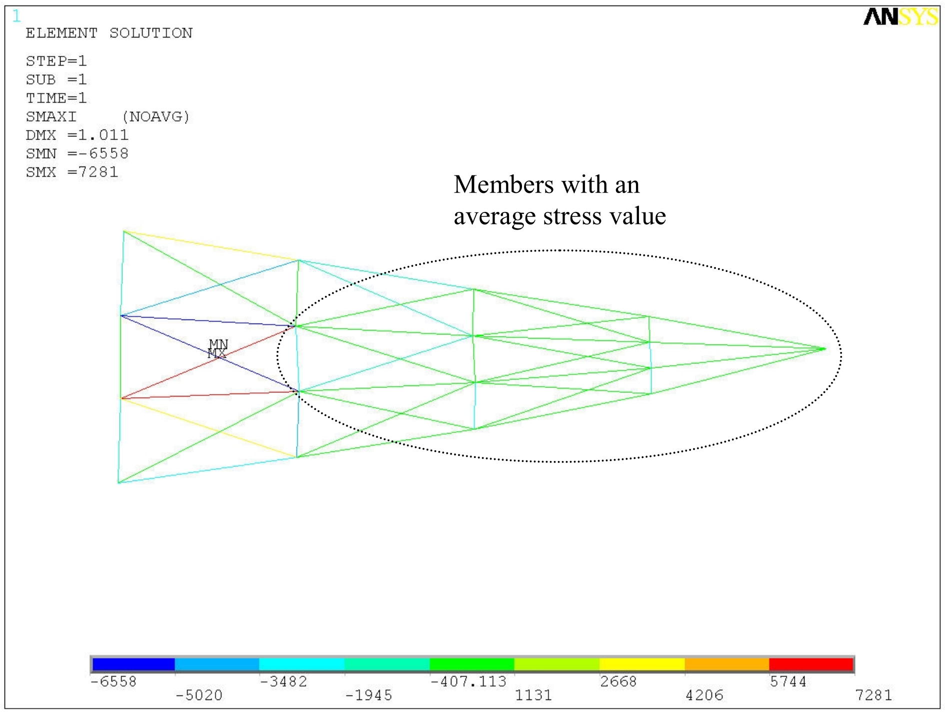

Figure 4.5: Internal beam member stresses

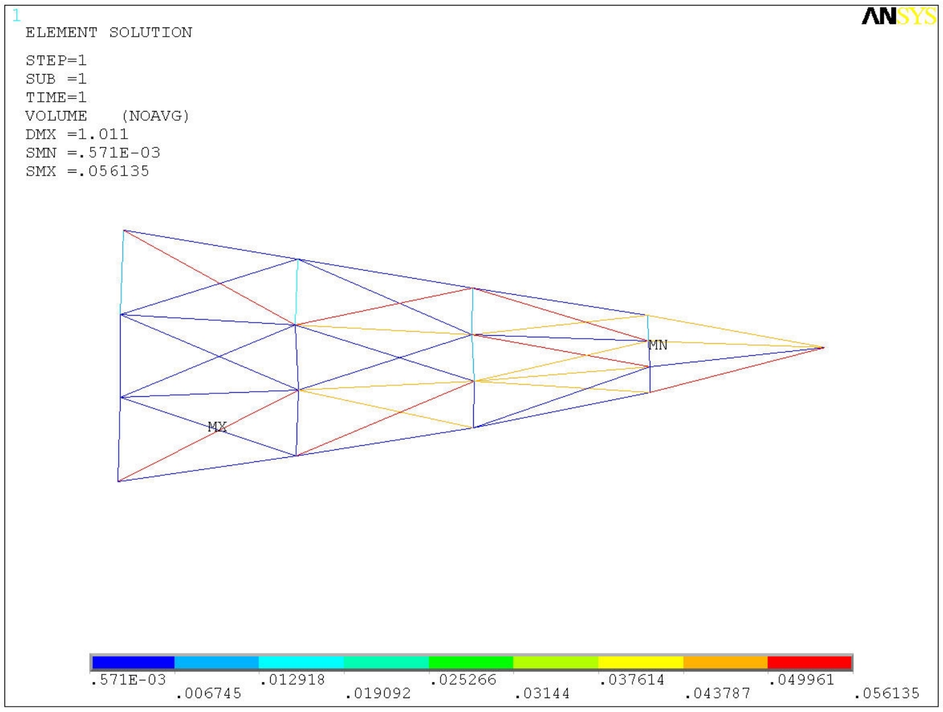

Figure 4.6: Volume of each beam member

From the results presented in Figure 4.5 and Figure 4.6, the problem with stress constraints can be shown. The purpose of the optimization program was to eliminate beam members that do not contribute to maximizing the vertical tip deflection. Beam elimination is accomplished by having the height of the undesired beams decrease to the lower bound of the design variables. However, this is not the case when stress is used as constraints because as the member's heights decrease, the stresses in the members increase. In addition, all members of the structure carry stress and by eliminating the beam members that are at the lower height limit, the stress of all the other members increase. Some members if removed will cause other beam members to violate the stress constraints, this result is shown in Figure 4.7. The maximum allowable stress was constrained to the absolute value of 7430 psi, and this constraint is violated by beam members MN and MX. In fact, the stress is so large that the material has surpassed its yield stress value of 10400 psi. This is all due to the removal of the beams with the lower limit value of the design variables. A better approach to the problem would be to use a multi-criteria objective function to maximize the ratio of Mutual Potential Energy over the Strain Energy of the structure. With this optimization formulation, the stress constraints are removed from the constraint to the objective function, while the height of the beam member's become the constraints. Using this multi-criteria formulation however, would require writing your own finite element program because ANSYS can not solve the sensitivity information of this type of objective function.

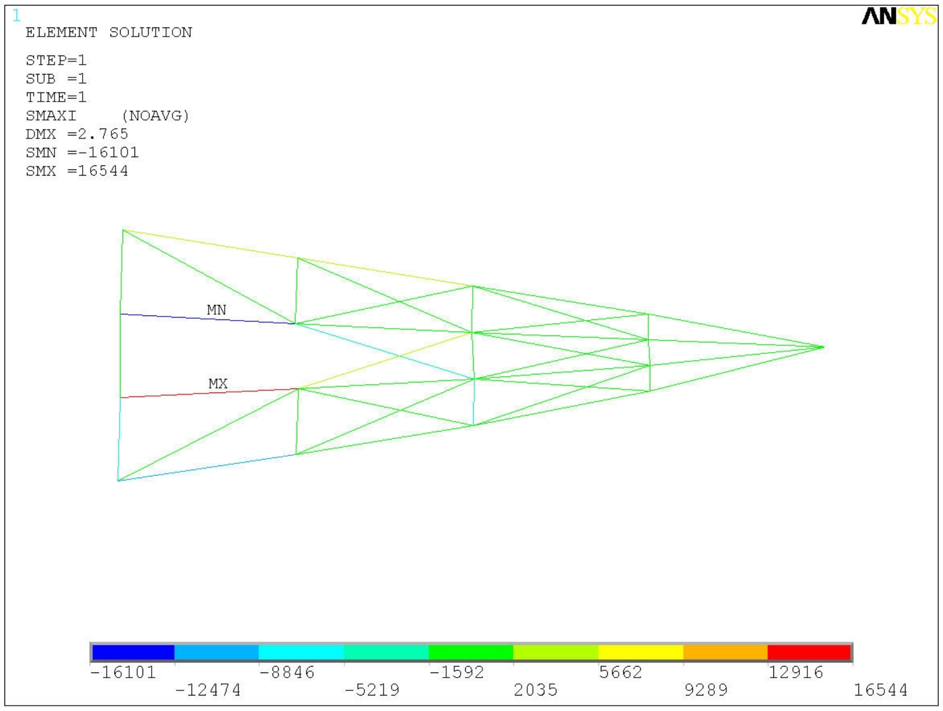

Figure 4.7: Violation of stress constraints

To achieve the optimal structural shape, the non critical members need to be removed. The non critical members are the beams that do not support the structure under loading and help maximize the vertical tip displacement. These members have a stress value that is around the average stress value. Most of the members with an average stress value are circled in Figure 4.5. The removal of some of these members reduce the weight of the structure and will not hamper the structural integrity of the tail or decrease the trailing edge tip deflection. The optimal shape of the variable camber compliant airfoil tail is shown in Figure 4.8 and is capable of a maximum tip deflection of ±0.934 in, which is an angle of deflection of ±4.27°. The optimal shape without any members removed is shown in Figure 4.9. The difference in vertical tip deflection is 0.066158 in. or 0.3° deflection between the two structures.

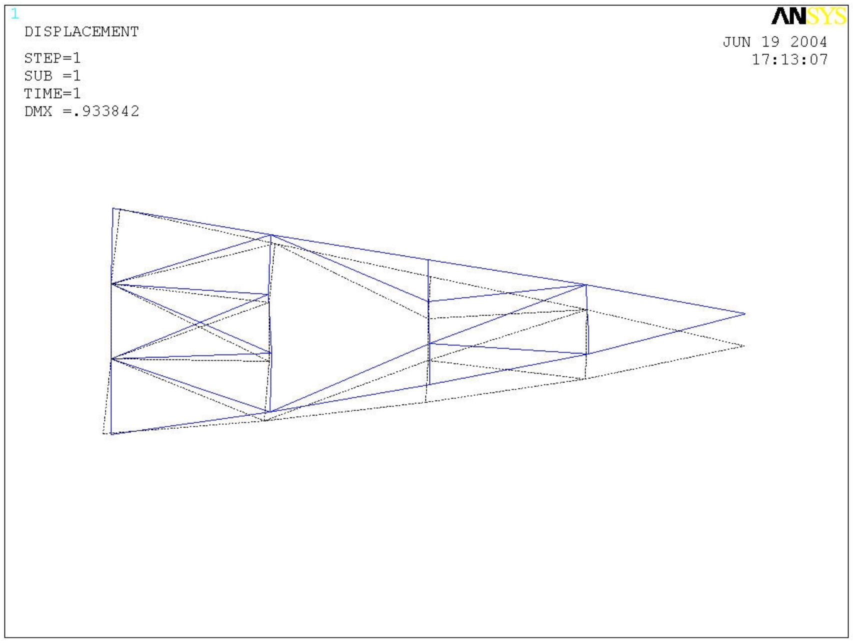

Figure 4.8: Trailing edge tip deflection of optimal compliant mechanism

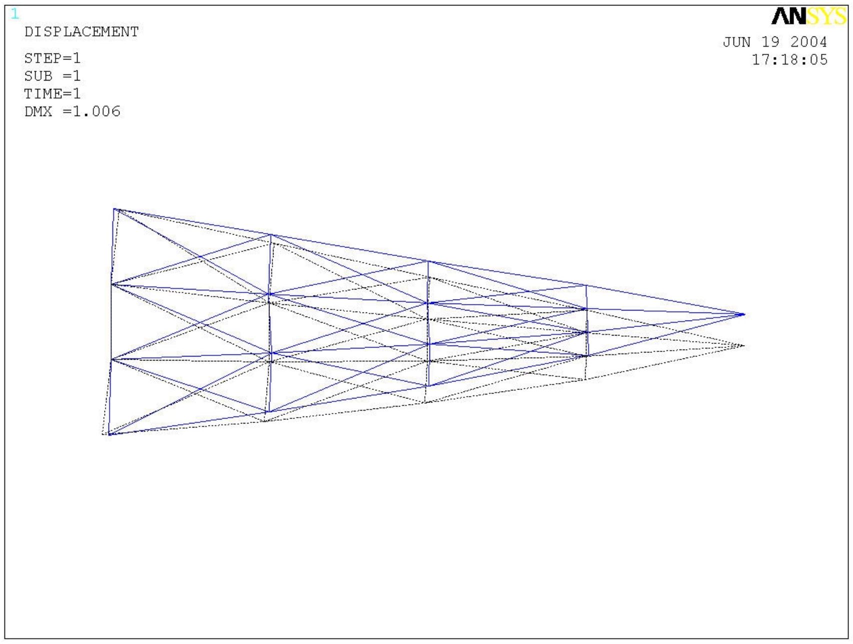

Figure 4.9: Trailing edge tip deflection without beam members removed

The member stresses of the optimal compliant mechanism are similar to the stresses for the optimal structure with out any beam members removed. The optimal structure stress values without any beam members removed are presented in Figure 4.5. The optimal shaped compliant mechanism stresses are shown in Figure 4.10.

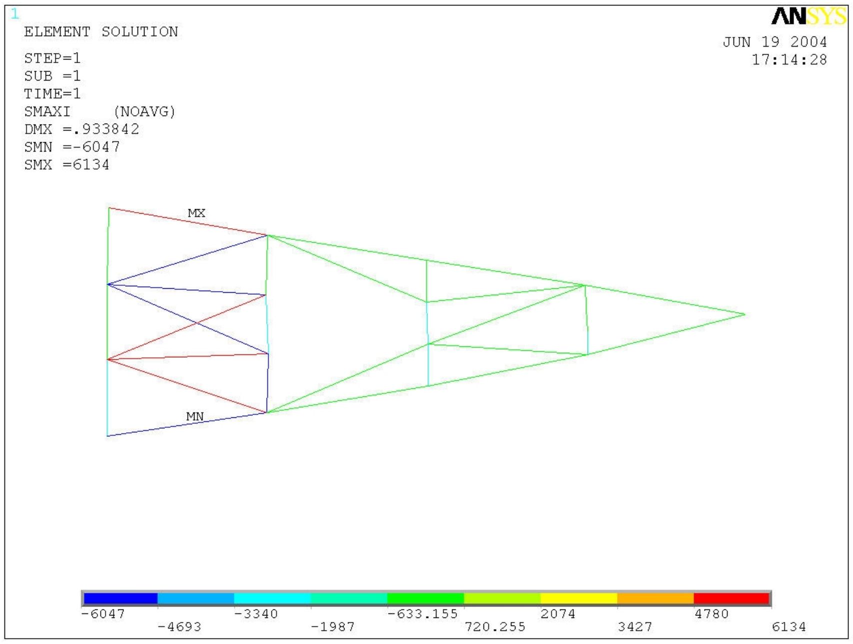

Figure 4.10: Internal stresses of the optimal tail shape

To better show the amount of deflection the variable camber tail achieves in relation to the whole fuselage, Figure 4.11 has the optimal tail undeflected while Figure 4.12 has the tail at its maximum trailing edge tip deflection.

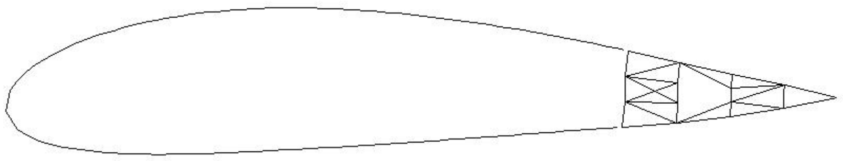

Figure 4.11: Fuselage with optimal complaint tail

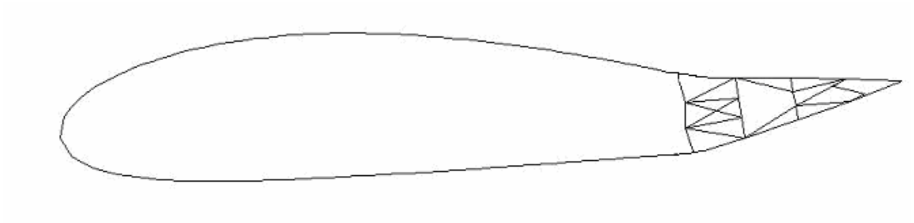

Figure 4.12: Fuselage with maximum trailing edge tip deflection

An investigation was also done to see the affects of changing the upper and lower limits of the design variables. If the lower limit was decreased, a singularity in the stiffness matrix occurred, unless the input force also decreased. As the upper bound of the design variables increased, so did the trailing edge tip deflection as well as the stress within the beam members. Therefore, to increase the upper bound, the factor of safety value would have to be decreased, so as not to violate the stress constraints. Also, an analysis was done by changing the material to one with a lower ratio of yield strength to Young's modulus value, correcting the upper and lower design variable limits to prevent a singularity in the stiffness matrix and decreasing the input force. With a different material used for the compliant mechanism, the optimal shape configuration stayed the same, while the maximum tip displacement decreased.

4.3.1 Aerodynamic Deformation of the Optimal Shaped Compliant Mechanism

The results presented in the previous section show the displacement and stresses occurring on the optimal shaped compliant mechanism due to an input force and aerodynamic forces. However, this section presents the affects the aerodynamic loading has on the compliant mechanism while at its maximum tip deflection of 4.27°. The optimal tail structure tip deflects only 0.198 in. or 0.9° under aerodynamic loading. The deformation of the structure due to the aerodynamics effects are shown in Figure 4.13. However, removal of any of the vertical beam members causes the deformation of the tail structure to exceed an unacceptable deflection of 0.5 in.

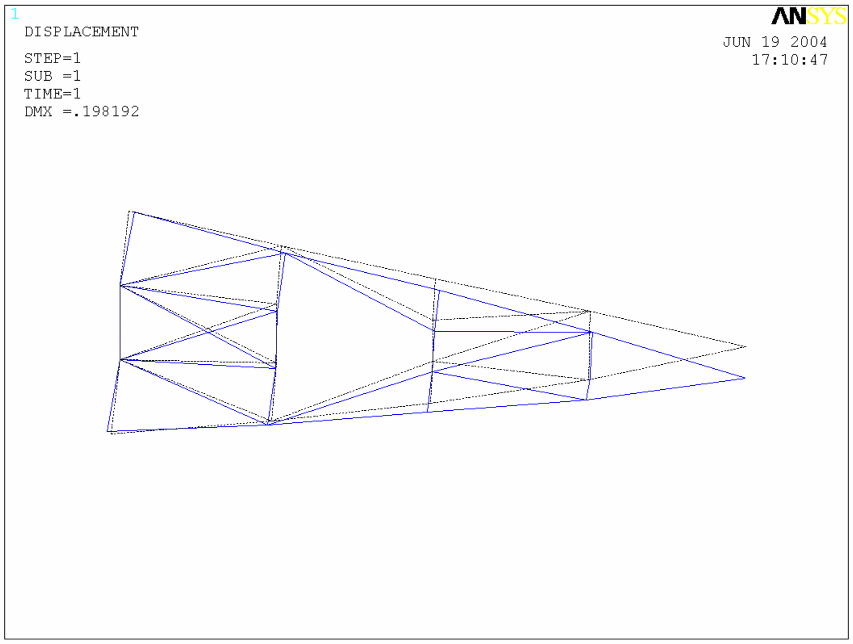

Figure 4.13: Deformation of compliant mechanism due to aerodynamic forces

4.3.1 Manufacturing the Compliant Mechanism

Manufacturing the optimal designed compliant mechanism resulted in many difficulties. First of all, the material chosen to make the compliant mechanism part from was ABS plastic, the same material the structure was designed with. However, ABS is made with a lot of rubber that easily catches on fire when the laser try's to make small precision cuts. In addition to catching on fire, when trying to cut the optimal shape, some of the beam members are very thin, and when using the laser to cut the material, the material would melt and not keep its shape. Therefore, to eliminate all of these problems, the laser cutter's power was turned down to 90% and the compliant tail members were scaled up. From experimenting with the laser cutter, the smallest height the laser would cut and still not warp the material was 0.125 in. Scaling up the model so the smallest members were 0.125 in. the manufactured piece would deflect only a small amount. The manufactured scaled-up version of the compliant mechanism is shown in Figure 4.14. Analyzing the scaled-up model in ANSYS, the trailing edge tip deflection was calculated to be 0.056296 in., which is too small to be used as a variable camber tail.

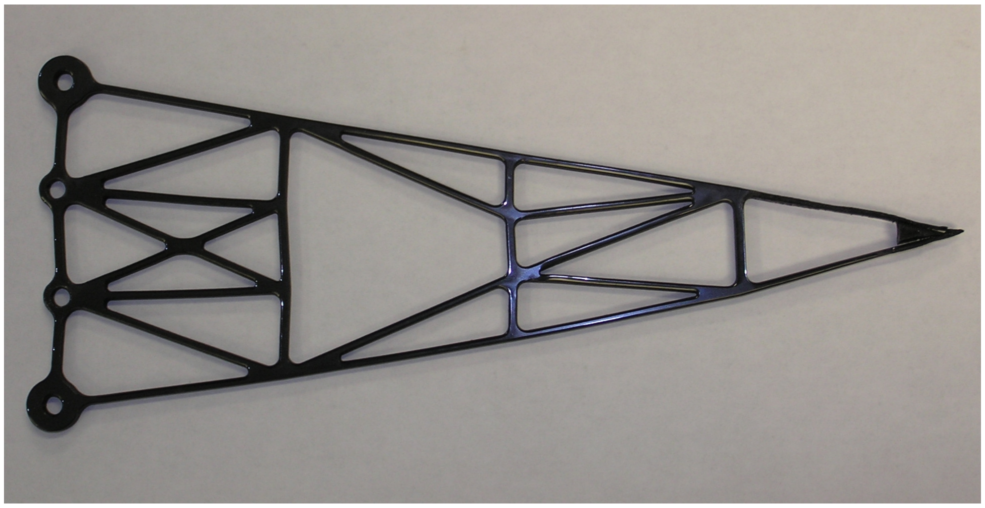

Figure 4.14: Manufactured compliant mechanism

4.4 Chapter Summary

Using the structural optimization program developed in this thesis to design an optimal shape to maximize the trailing edge tip deflection under aerodynamic loading was achieved. The optimal shaped compliant tail mechanism can obtain a maximum vertical tip deflection of ±0.934 in., which is ±4.27° angle of deflection. This result was after non-critical beam members were removed. The tip deflection due to the aerodynamic forces was 0.198 in. or 0.9° angle of deflection. However, from the results of the structural optimization program, it was determined that using stress as constraints was not a good decision. As demonstrated in this chapter, having the stress as constraints does not allow eliminating the beam members that converge to the lower bound value of the design variables. Therefore, when removing members, one must consider the stresses in the members and remove only the non-critical members that are near the average stress value. A better approach would be to use a multi-criteria objective function to maximize the ratio of Mutual Potential Energy over the Strain Energy of the structure.

Chapter 5

Conclusion

Throughout this research the primary objective has been the design of a variable camber tail using a compliant mechanism to maximize the trailing edge tip displacement. Much of the study was devoted to developing a structural optimization program to optimize the topology of a discretized tail structure. The program developed used a main C program to link ANSYS to be used to analyze the airfoil tail structure, an aerodynamic code to generate the aerodynamic forces acting on the aircraft tail as it deflects, and an optimization program to optimize the design variables, which were the height of the beam members. This chapter presents the contributions of this research along with the conclusions that were drawn from the study. Finally, section 5.2 presents recommendations for future work which could be conducted in this area.

5.1 Contribution

The purpose of the research presented in this thesis was to develop a structural optimization program to optimize the topology of an airfoil tail section. Designing a variable camber tail eliminates the discontinuity between the airfoil and the flap. This work contributes a structural optimization program that links a finite element program to an aerodynamic code and an optimization program. With this structural optimization program, the trailing edge tail deflection is achieved using a single actuator. Another benefit of this structural optimization program is that any structure that can be modeled in ANSYS can be optimized. This means that not only can a two-dimensional compliant mechanism be designed for optimal deflection; a three-dimensional compliant wing could be designed to vary the camber or twisting of the wing. Another contribution of the structural optimization program is including the affects of aerodynamic forces on the structure. As the program iterates, the aerodynamic forces get updated, since the aerodynamic forces acting on the structure change as the tip deflection increases. Therefore, the optimal structure will be flexible enough to maximize the tip displacement, but stiff enough to minimize the deflection due to aerodynamic loading. Another contribution of this research is that a single actuator is used to deflect a compliant tail mechanism ±0.934 in, which is ±4.27° angle of deflection under aerodynamic loads. Unlike most compliant mechanisms used to deflect the trailing edge of an airfoil, the mechanism designed using the structural optimization tool presented in this thesis is easily manufactured. This is due to fact that all members are designed out of the same material and only a small number of them are needed to deflect the structure.

5.2 Future Work

Future work that can be done on the morphing airplane with the variable camber tail is extensive wind-tunnel testing. The morphing airplane should be tested to see the benefits of anti-symmetric morphing as well as the benefits of dynamic morphing to be used for flight control. For the structural optimization program, the most important future work would be to remove the stress constraints. To eliminate the stress constraints, a finite element program needs to be written to handle the nonlinear objective functions. However, if a new objective function can be found that allows the use of ANSYS as the finite element program, many future capabilities can be addressed. Keeping ANSYS as the finite element program allows almost any structure that can be modeled in ANSYS to be optimized. In the current research, the compliant mechanism was designed using linear analysis for the material and the beam in bending. However, compliant mechanisms bend in a nonlinear fashion and the materials behave nonlinearly. Both nonlinear capabilities can be addressed simply by changing the ANSYS log file code to account for the nonlinearities of the material and bending. In addition to adding nonlinear analysis, a three-dimensional compliant variable camber tail or wing can be optimized. The 3D model only needs to be modeled in ANSYS and the main C code and optimization program will optimize the structure. The structural optimization program offers a wide range of future capabilities that can be easily addressed with simple modifications.

Bibliography

Ananthasuresh, G.K., and Kota, S. 1995, "Designing Compliant Mechanisms", Mechanical Engineering, vol. 37, No. 11, November, 1995, pp. 93-96.

Anusonti-Inthra, P., Gandhi, F. and Frecker, M., "Design of a conformable rotor airfoil using distruibuted piezoelectric actuation", Proceedings of IMECE'03, 2003 ASME International Mechanical Engineering Congress, Washington, D.C., November 15-21, 2003.

Barrett, R., "Active aeroelastic tailoring of an adaptive Flexspar stabilator", *Journal of Smart Materials and Structures*, vol. 5, No. 6, Bristol, UK, pp. 723-730, 1996.

Bendsoe, M.P. and Sigmund, O., *Topology Optimization Theory, Methods and Applications*, Springer-Verlag Berlin Heidelberg, New York, 2003.

Blondeau, J., Richeson, J., and Pines, D.J., "Design, development and testing of a morphing aspect ratio wing using an inflatable telescopic spar", *44th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference*, Norfolk, VI, April 7-10, 2003.

DARPA website: http://www.darpa.mil/dso/thrust/matdev/mas.htm, 15 March 2004.

F-14 Tomcat website:
http://www.geocities.com/CapeCanaveral/Lab/5139/Grumman.htm, 3 June 2004.

Frecker, M., "Optimal design of compliant mechanisms", Doctor of Philosophy in M.E. in The University of Michigan, 1997.

Gern, F.H., Inman, D.J., Kapania, R.K., "Computation of actuation power requirements for smart wings with morphing airfoils", 43rd *AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference*, Denver, CO, April 22-25, 2002.

Haftka, R.T., and Gürdal, A., *Elements of Structural Optimization*, Kluwer Academic Publishers, Netherlansds, 1992.

Hibbler, R.C, *Mechanics of Materials*, Prentice Hall, New Jersey, 1997.

Howell, L.L, *Compliant Mechanisms*, John Wiley & Sons, New York, 2001.

Howell, L.L. and Midha, A., "A method for the design of compliant mechanisms with small-length flexural pivots", *Journal of Mechanical Design*, vol. 116, March 1994, pp. 280-290.

Johnston, C.O., Neal, D.A., Wiggins, L.D., Robertshaw, H.H., Mason, W.H., and Inman, D.J., "A model to compare the flight control energy requirements of morphing and conventionally actuated wings", AIAA Paper 2003-1716, *44th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Norfolk, VA, April 7-10, 2003.

Kudva, J.N., Appa, K., Jardine, A.P., Martin, C., "Overview of Recent Progress on the DARPA/USAF Wright Laboratory Smart Materials and Structures Development – Smart Wing" Program", *SPIE Smart Structures and Materials Conference 1997: Industrial and Commercial Applications of Smart Structures Technologies*, J. M. Sater (ed.), vol. 3044, International Society for Optical Engineering, Bellingham, WA, pp 24-32, 1997.

Murphy, M.D., Midha, A., and Howell, L.L, “The topological synthesis of compliant mechanisms”, *Mechanism and Machine Theory*, 31, 185, 1996.

Norton, R.L., *Machine Design an Integrated Approach*, Upper Saddle River, NJ, Prentice-Hall Inc., 1998.

Paros, J.M. and Weisbord, L., “How do design flexure hinges”, *Machine Design*, vol. 37, pp 151-157, 1965.

Pettit, G.W., Robertshaw, H.H., Gern, F.H. and Inman, D.J., “A Model to evaluate the aerodynamic energy requirements of active materials in morphing wings”, Proceedings of DETC’01, 2001 ASME Design Engineering Technical Conferences, September, 2001.

Saxena, A. and Ananthasuresh, G.K., “Topology optimization of compliant mechanisms with strength considerations”, *Mechanics of Structures and Machines*, 29(2), pp. 199-221, 2001.

Svanberg, Krister, “The Method of Moving Asymptotes - A New Method for Structural Optimization”, *International Journal for Numerical Methods in Engineering* **24**:359-373, 1987.

Thomas, Adrian L.R., “On the Aerodynamics of Birds’ Tails”, *Philosophical Transactions: Biological Sciences*, Vol. 240, No. 1294, pp 236-380, June 29th, 1993.

Wiggins, L., Stubbs, M., Johnston, C., Robertshaw, H., Reinholtz, C., and Inman, D., A design and analysis of a morphing Hyper-Elliptic Cambered Span (HECS) wing, *45th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference*, Palm Springs, CA, April 19-22, 2004.

Appendix A

Matlab Code to Generate Initial Discretized Aircraft Tail Ground Structure

```
close all
clear all
clc

syms M R P Hw Sr Ar c BEAM3 LINK1

A=[Sr^2 Sr 1;
    2*Sr 1 0;
    2 1 0];

Z1=M/(1-P)^2*(1-2*P+2*P*Sr-Sr^2);
Z2=2*M/(1-P)^2*(P-Sr);

B=[Z1 Z2 Ar].';

single=inv(A)*B;

%All coefficients in percentage of chord
chord=49;   % chord length of fuselage
c=1;        % chord
P=0.4;      % location of max camber
M=0.04;     % max camber value
R=0.2551;   % size of reflex control surface
base_ang=2*M/(1-P)^2*(P-1)*180/pi;
Ar=0;
Hw=.17;%0.12;     % maximum thickness value
t=Hw*c;     % max thickness

Sr=c-R;
Ar=Ar*pi/180;

elem_num=4; % Number of elements along the top line of the tail
```

```
Tipnode=5; % Node number of tip, not Keypoint, if elem_num=3,
Tipnode=4;
div=4; % Number of keypoints in the vertical direction
element_type=BEAM3; % describes the element type, BEAM3 or LINK1
width=.25;  % in, width of the beam
%Force=1;%-1000;  % Force acting on the tail tip in Fy direction
Torque=8; %in-lb, Torque produced by the motors on the keypoints,
%Xsect=width*height;  % in^2, Defines the cross sectional area
E=2.4*145038; %psi, ABS Plastic molded, extruded, Matweb range(1.79-3.2
GPa, avg 2.4GPa) Defines the Modulus of elasticity of ABS Plastic,
conversion to psi is *145038
sig_yield=71.8*145; %psi, *145 conversion from MPa to psi, Matweb,
range from 47.8-107MPa; Flexural yield stress of ABS Plastic molded
DVmin=0.005; %in ; minimum height the members can be
DVmax=0.0625;%0.25; %in ; maximum height the members can be
height=.2; %in, height of the beam
houter=.125; %in, height of the outer surface beams
Izz = 1/12*width*height^3; % Defines the moment of inertia about the z
axis

a=subs(single(1));
b=subs(single(2));
d=subs(single(3));

n=1;
for x=Sr:(R/elem_num):c  % Only the reflex control surface (tail) will
be created
    if x<P*c
        yc(n)=c*M/(P^2)*(2*P*(x/c)-(x/c).^2);
        dyc(n)=2*M/(P^2)*(P-(x/c));
    elseif x>=P*c & x<Sr*c
        yc(n)=c*M/(1-P)^2*(1-2*P+2*P*(x/c)-(x/c).^2);
        dyc(n)=2*M/(1-P)^2*(P-(x/c));
    elseif x>=Sr*c
        yc(n)=c*(a*(x/c).^2+b*(x/c)+d);
        dyc(n)=(2*a*(x/c)+b);
    end
    n=n+1;
end

x=[Sr:(R/elem_num):c]; %define chord line

a0=1.4845; a1=0.63; a2=1.758; a3=1.4215; a4=0.5075;
yt=t*(a0*sqrt(x/c)-a1*(x/c)-a2*(x/c).^2+a3*(x/c).^3-a4*(x/c).^4);
%thickness distribution

theta=atan2(dyc,ones(size(dyc)));

xup=[x-yt.*sin(theta)];
yup=[yc+yt.*cos(theta)];

xlow=[x+yt.*sin(theta)];
ylow=[yc-yt.*cos(theta)];

% %average last yup and ylow to form one point
%y=(yup+ylow)/2;
```

```
%Determine the range of y
ytot=yup-ylow;
% Divide up the horizontal lines between yup and ylow by div
count=2;
for ii=div-2:-1:1
    y(count,:)=ylow+ytot*ii/(div-1);
    x_expand(ii+1,:)=x;
    count=count+1;
end

% FEM keypoints
figure(1)
% Plot the keypoints
plot(xup*chord,yup*chord,'r^',xlow*chord,ylow*chord,'r^',x*chord,y(2:en
d,:)*chord,'k^')
axis equal

figure(2)
% Plot the keypoints
plot(xup*chord,yup*chord,'r',xlow*chord,ylow*chord,'r')
axis equal

% Combine xup, xlow and x into a matrix x and also combine yup, ylow
and y
% into a matrix y
x_expand(1,:)=xup;
x_expand(end+1,:)=xlow;
x=x_expand;
y(1,:)=yup;
y(end+1,:)=ylow;

% Average the y location of the tail tip, so they meet at one point
yavg=0;
for ii=1:div
    yavg=y(ii,end)+yavg;
end
yavg=yavg/div;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Check if desired shape can be reached
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%YieldStress(E,Izz,yavg,sig_yield,R,height)

%Generate the Keypoints matrix
count=0;
for ii=1:div
    for jj=1:elem_num
        Keypoints(jj+count,1)=[x(ii,jj)*chord];
        Keypoints(jj+count,2)=[y(ii,jj)*chord];
    end
    count=count+elem_num;
end

Keypoints(end+1,:)=[1*chord,yavg*chord]
```

```
% Create a matrix of the keypoint numbers that will be used to generate
the lines
count=1;
for ii=1:div
    for jj=1:elem_num
        Lines(ii,jj)=count;
        count=count+1;
    end
end
tip=length(Keypoints);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Generate the ANSYS log file
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

OptLogfileInForce(Keypoints,Lines,div,elem_num,element_type,E,Torque,ti
p,chord,Ar,x,sig_yield,width,DVmin,DVmax,Tipnode)

fclose('all'); % some file doesn't close, so close everything



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   The function OptLogfileInForce.m will generate the initial
%      ground structure and create the ANSYS log file
%
%   This function generates a partial ground structure
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function
OptLogfileInForce(Keypoints,Lines,div,elem_num,element_type,E,Torque,ti
p,chord,Ar,x,sig_yield,width,DVmin,DVmax,Tipnode)

global counter ao

toler=0.001; %in ; tolerance
SVmin=-sig_yield; %psi ; yield point of material
SVmax=sig_yield; %psi ; yield point of material
SVolmin=.0001; %in^3 ; minimum volume of the total structure
SVolmax=.05; %in^3 ; maximum volume of the total structure
ObjToler=0.001; %in ; objective function tolerance

syms BEAM3 LINK1
%fid=fopen('C:\CompliantMech\OptLogfileInForce.txt','w');
fid=fopen('C:\CompliantMech\OptLogfileInForceTemplate.txt','w');
% fprintf(fid, ['finish\n']);
% fprintf(fid, ['/clear,start\n']);
fprintf(fid, ['/BATCH\n']);
fprintf(fid, '/UIS,MSGPOP,3\n'); % Suppresses ANSYS warnings
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Define height as variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf(fid, ['/INPUT,''C:\\CompliantMech\\Hvar'',''txt'', !file of
variables\n']);
fprintf(fid, ['W=',num2str(width),'\n']); % Declares the width

fprintf(fid, ['/PREP7\n']);

% Generate the Partial Ground Structure
for nn=1:length(Keypoints)
    fprintf(fid,
['K,',num2str(nn),',',num2str(Keypoints(nn,1)),',',num2str(Keypoints(nn
,2)),',0,\n']);
end

counter=0;

% Create horizontal lines
for ii=1:div
    for jj=1:elem_num-1
        fprintf(fid,
['l,',num2str(Lines(ii,jj)),',',num2str(Lines(ii,jj+1)),',\n']);
        counter=counter+1;
    end
    % Create lines to the tail
    fprintf(fid, ['l,',num2str(Lines(ii,end)),',',num2str(tip),',\n']);
    counter=counter+1;
end

% Create vertical lines
for ii=1:div-1
    for jj=1:elem_num
        % if(Lines(ii,jj)~=5 && Lines(ii+1,jj)~=9) % Removes beam
connecting nodes 5 and 9
        fprintf(fid,
['l,',num2str(Lines(ii,jj)),',',num2str(Lines(ii+1,jj)),',\n']);
        counter=counter+1;
        %   end
    end
end

% Create diagonal lines
for ii=1:div-1
    for jj=1:elem_num-1
        fprintf(fid,
['l,',num2str(Lines(ii,jj)),',',num2str(Lines(ii+1,jj+1)),',\n']);
        counter=counter+1;
        fprintf(fid,
['l,',num2str(Lines(ii,jj+1)),',',num2str(Lines(ii+1,jj)),',\n']);
        counter=counter+1;
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Define Element definitions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if element_type==LINK1
    % Truss
    fprintf(fid, ['ET,1,LINK1\n']); % For a truss
    for ii=1:counter
        fprintf(fid, ['R,',num2str(ii),',H',num2str(ii),'*W','\n']); %
Real constant, defines the cross sectional area
    end
    fprintf(fid, ['MP,EX,1,',num2str(E),'\n']); % Material property,
defines the Young's modulus

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %  Define Boundary Conditions, attach to Keypoints
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fprintf(fid, ['DK,',num2str(Lines(2,1)),',all,0\n']);
    fprintf(fid, ['DK,',num2str(Lines(end-1,1)),',all,0\n']);

elseif element_type==BEAM3
    % Beam
    fprintf(fid, ['ET,1,BEAM3\n']); % For a beam use BEAM3

    for ii=1:counter
        fprintf(fid,
['R,',num2str(ii),',H',num2str(ii),'*W',',W/12*H',num2str(ii),'**3,H',n
um2str(ii),',,,\n']);
    end
    fprintf(fid, ['MPTEMP,,,,,,,,\n']); % Material property, defines
the temperature
    fprintf(fid, ['MPTEMP,1,0 \n']); % Material property, defines the
temperature
    fprintf(fid, ['MPDATA,EX,1,,',num2str(E),'\n']); % Material
property, defines the Young's modulus
    fprintf(fid, ['MPDATA,PRXY,1,,.3\n']); % Material property, defines
poissons ratio

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %  Define Boundary Conditions, attach to Keypoints
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fprintf(fid, ['DK,',num2str(Lines(2,1)),', ,0, ,0,UX,UY, , , ,
,\n']);
    fprintf(fid, ['DK,',num2str(Lines(end-1,1)),', ,0, ,0,UX,UY, , , ,
,\n']);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Apply input forces
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if element_type==LINK1
    fprintf(fid, ['/PREP7\n']);
    fprintf(fid, ['/GO\n']);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %  Define Forces acting on the structure at Keypoints
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fprintf(fid,
['FK,',num2str(Lines(1,1)),',FX,',num2str(Torque),'\n']);
    fprintf(fid, ['FK,',num2str(Lines(4,1)),',FX,-
',num2str(Torque),'\n']);

elseif element_type==BEAM3

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %  Define Torque Forces, attach to Keypoints
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fprintf(fid, ['/PREP7\n']);
    fprintf(fid, ['/GO\n']);

    fprintf(fid,
['FK,',num2str(Lines(4,1)),',FX,',num2str(Torque),'\n']);
    fprintf(fid, ['FK,',num2str(Lines(1,1)),',FX,-
',num2str(Torque),'\n']);
    fprintf(fid, ['/INPUT,''C:\\CompliantMech\\AVal'',''txt'', !file of
variables\n']); % Add aerodynamic forces
end

fprintf(fid, 'FINISH\n');
fprintf(fid, ['/PREP7\n']);

%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Meshing
%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf(fid, ['LESIZE,ALL, , ,1, ,1, , ,1,  ! Makes elements whole
length of truss member,\n']);
fprintf(fid, ['                                        !  can''t have a node
where truss members cross!!!\n']);

for ii=1:counter
    fprintf(fid, ['lsel,s,line,,',num2str(ii),'\n']); % selects line to
attach attributes to,  ii is line number
    fprintf(fid, ['latt,1,',num2str(ii),',1,,\n']);  % attaches
material, real constant (ii) and type to meshed line
end

fprintf(fid, ['lsel,all\n']);
fprintf(fid, ['lmesh,all\n']);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Define Forces acting on the structure at Keypoints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%fprintf(fid, ['FK,',num2str(tip),',FY,',num2str(Force),'\n']);
fprintf(fid, ['finish\n']);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Solve the resulting system of equations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf(fid, '/SOLU \n'); % Solves the ANSYS File
fprintf(fid, 'SOLVE\n'); % Solves the ANSYS File

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Postprocessing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf(fid, '/POST1\n'); % Picks postprocessing
fprintf(fid, 'PLDISP,1\n'); % Plots undeformed and deformed shape
%fprintf(fid, 'PRRSOL,F \n'); % List Reaction Forces

% Node 6, tip, displacement due to input force
fprintf(fid, 'PRNSOL,U,Y\n'); % Tip displacement

% Volume of each element
fprintf(fid, 'ETABLE,Volume,Volu,\n');


% Displacement table
%fprintf(fid, 'ETABLE,,U,X\n'); % x-displacement
%fprintf(fid, 'ETABLE,,U,Y\n'); % y-displacement

if element_type==LINK1
    fprintf(fid, 'ETABLE,SAXL,LS, 1\n'); % Axial Stress
    fprintf(fid, 'SSUM\n');  % Summation of the volume

    fprintf(fid, 'PRETAB,SAXL,VOLUME\n'); % List Element Table
    for ii=1:counter
        %    fprintf(fid,
['*GET,Volume',num2str(ii),',ELEM,',num2str(ii),',VOLU,\n']);
        fprintf(fid,
['*GET,SAXL',num2str(ii),',ELEM,',num2str(ii),',ETAB,SAXL\n']);
    end
elseif element_type==BEAM3
    % Stress at nodes
    fprintf(fid, 'ETABLE,SMAXI,NMISC,1\n'); % stress at the ith node
    fprintf(fid, 'ETABLE,SMAXJ,NMISC,3\n'); % stress at the jth node

    fprintf(fid, 'SSUM\n');  % Summation of the volume

    fprintf(fid, 'PRETAB,SMAXI,SMAXJ,VOLUME\n'); % List Element Table

    for ii=1:counter
        %    fprintf(fid,
['*GET,Volume',num2str(ii),',ELEM,',num2str(ii),',VOLU,\n']);
        fprintf(fid,
['*GET,StressI',num2str(ii),',ELEM,',num2str(ii),',ETAB,SMAXI\n']);
        fprintf(fid,
['*GET,StressJ',num2str(ii),',ELEM,',num2str(ii),',ETAB,SMAXJ\n']);
        fprintf(fid,
['SMAX',num2str(ii),'=StressI',num2str(ii),'>StressJ',num2str(ii),'\n']
);
    end
```

```
end
fprintf(fid, ['*GET,YDISPTip,NODE,',num2str(Tipnode),',U,Y\n']);

% NEW OBJECTIVE FUNCTION DEFINED
%fprintf(fid, ['ObjTip=1/YDISPTIP !objective function is 1/uf\n']);
%%%%% New objective defined
fprintf(fid, ['ObjTip=-YDISPTIP !objective function is 1/uf\n']);
%%%%% New objective defined

fprintf(fid, 'FINISH\n');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Specify Design and State Variables and Object function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf(fid, '/OPT\n');
fprintf(fid, 'OPMAKE\n'); %creates design set

% Need this line if not running ANSYS in batch mode
%fprintf(fid, ['OPANL,C:\\CompliantMech\\AnalysisFileOptTail,txt, !
Don''t need for a batch run\n']);
if element_type==LINK1
    for ii=1:counter
        fprintf(fid,
['OPVAR,H',num2str(ii),',DV,',num2str(DVmin),',',num2str(DVmax),',',num
2str(toler),',\n']);
        %    fprintf(fid,
['OPVAR,Volume',num2str(ii),',SV,',num2str(SVolmin),',',num2str(SVolmax
),',',num2str(toler),',\n']);
        fprintf(fid,
['OPVAR,SAXL',num2str(ii),',SV,',num2str(SVmin),',',num2str(SVmax),',',
num2str(toler),',\n']);
    end
elseif element_type==BEAM3
    for ii=1:counter
        fprintf(fid,
['OPVAR,H',num2str(ii),',DV,',num2str(DVmin),',',num2str(DVmax),',',num
2str(toler),',\n']);
        %    fprintf(fid,
['OPVAR,Volume',num2str(ii),',SV,',num2str(SVolmin),',',num2str(SVolmax
),',',num2str(toler),',\n']);
        fprintf(fid,
['OPVAR,SMAX',num2str(ii),',SV,',num2str(SVmin),',',num2str(SVmax),',',
num2str(toler),',\n']);
    end
end


%fprintf(fid, ['OPVAR,YDISPTip,OBJ,,,',num2str(ObjToler),',\n']);

fprintf(fid, ['OPVAR,OBJTIP,OBJ,,,',num2str(ObjToler),',\n']);


% Choose the Gradient method for sensitivity analysis
fprintf(fid, 'OPTYPE,GRAD\n');

fprintf(fid, 'OPGRAD,LAST,0\n');
```

```
% Run Sensitivity analysis in ANSYS
fprintf(fid, 'OPEXE\n')

% Print Sensitivity results our
fprintf(fid, 'OPRGR,ALL\n');

% fprintf(fid, ['save,C:\\CompliantMech\\OptOutputdb,db,,all\n']);

fclose(fid);
```

Appendix B

Main C Program that Integrates ANSYS, Thin Airfoil Code and Method of Moving Asymptotes

```
/**********************************************************************

 This file is the main C program to run the variable camber compliant
airfoil tail code
   using structural optimization.  The ANSYS log files are generated
using the Matlab TailGenerator.m.

 File name: OptCCode.cpp
 Created By: Matthew Good
 Date: 5/11/04
 Updated: 6/8/04

**********************************************************************/

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <malloc.h>
#include "matrix.h"
#include "mmasubLib.h"

#pragma comment(lib, "libmx.lib")
#pragma comment(lib, "libmatlb.lib")
#pragma comment(lib, "libmat.lib")
#pragma comment(lib, "libmmfile.lib")
#pragma comment(lib, "libmatlb.lib")
#pragma comment(lib, "mmasubLib.lib")
```

```
/* Global Variables */
#define num_dv 46
#define num_nodes 17
#define tip_node 5
#define chord 49
#define elem_num 4
#define mm 46
#define nn 46

int main()
{

/*********************************************************************
    Define pointers used in the program
*********************************************************************/

      /* Pointers for Aerodynamic code */

      mxArray* aero_ptr;
      mxArray* chord_ptr;
      mxArray* angle_ptr;
      mxArray* elem_num_ptr;
      mxArray* keypt_ptr;

      /* Pointers for Method of Moving Asymptotes code */

      mxArray *m_ptr;
      mxArray *n_ptr;
      mxArray *xval_ptr;
      mxArray *xmin_ptr;
      mxArray *xmax_ptr;
      mxArray *xold1_ptr;
      mxArray *xold2_ptr;
      mxArray *lowin_ptr;
      mxArray *uppin_ptr;
      mxArray *a_ptr;
      mxArray *a0_ptr;
      mxArray *c_ptr;
      mxArray *d_ptr;
      mxArray *dfodx_ptr;
      mxArray *dfodx2_ptr;
      mxArray *fval_ptr;
      mxArray *dfdx_ptr;
      mxArray *dfdx2_ptr;
      mxArray *foval_ptr;
      mxArray *iter_ptr;

      mxArray *xmma_ptr;
      mxArray *ymma_ptr;
      mxArray *zmma_ptr;
      mxArray *lam_ptr;
      mxArray *xsi_ptr;
      mxArray *eta_ptr;
      mxArray *mu_ptr;
      mxArray *zet_ptr;
      mxArray *s_ptr;
      mxArray *lowout_ptr;
      mxArray *uppout_ptr;


/***********************************************************************
    Define variables used in the program
**********************************************************************/
  /* pointers */
      char *line;

line=(char *)malloc(50);  // Allow a variable length string to be
                              stored in line
      if(line==NULL)
            cerr<<"Not enough memory for malloc\n";

  double objfunc=0;
  char gap[]="1";
  char space[]="";
  char space2[]=" ";
  char line2[]="    NODE      UY    ";
  char line3[]=" CHANGE IN VALUE (DUE TO +1% CHANGE IN DV)
GRADIENT";
  char line4[]=" TABLE LABEL     TOTAL";
  char line5[]=" MINIMUM VALUES";
  char line6[]=" MAXIMUM VALUES";
  int num[num_dv];
  double SMAXI[num_dv];
  double SMAXJ[num_dv];
  double Vol[num_dv];
  double TipDispF;
  double PI=3.14;
  double objold;
  double objnew;
  char text1[100];
  char text2[70], text3[20];
  int cc=1;
  int dd=1;
  int iteration=1;
  double error=1;
  double FGrad[num_dv+1][num_dv];
  double Grad[num_dv+1][num_dv];
  double SMAXF[num_dv][1];
  double SMAX[num_dv][1];
  double Volume;
  double SAXL[num_dv];
  double chordArray[1]={49};
  double angArray[1];
  double elemArray[1]={elem_num};
  double keyptArray[1]={num_nodes};
  double foval;
  double FOS=1.4;

  /* Initial Values to Change */

  int maxiter=15;
  int begin=1
  int element=2;
  double InHval=0.12;
  double MinHval=0.005;
  double MaxHval=0.03125;
  double MaxStress=71.8/FOS*145;

  /* Define variables for the Method of Moving Asymptotes Program*/

  int iter=1;
  int m=mm; )
  int n=nn;
  double xval[nn][1];
  double xmin[nn][1];
  double xmax[nn][1];
  double xold1[nn][1];
  double xold2[nn][1];
  double dfodx[nn][1];
  double dfodx2[nn][1];
  double fval[mm][1];
  double dfdx[nn][mm];
  double dfdx2[nn][mm];
  double lowin[nn][1];
  double uppin[nn][1];
  double a0=1;
  double a[mm][1];
  double c[mm][1];
  double d[mm][1];

  double start, finish, duration;

  double * xvalp = (double*)malloc(sizeof(double)*nn);
  double * xold1p = (double*)malloc(sizeof(double)*nn);
  double * xold2p = (double*)malloc(sizeof(double)*nn);
  double * lowinp = (double*)malloc(sizeof(double)*nn);
  double * uppinp = (double*)malloc(sizeof(double)*nn);

/* Files used to record values at each iteration  */
ofstream opFuncFile("C:\\CompliantMech\\ObjFunc.txt",
ios::trunc|ios::out);
ofstream opDispFile("C:\\CompliantMech\\ObjDisp.txt",
ios::trunc|ios::out);
ofstream opFile6("C:\\CompliantMech\\FStress.txt",
ios::trunc|ios::out);
ofstream opFileXval("C:\\CompliantMech\\xval.txt",
ios::trunc|ios::out);

/*******************************************************************
   Initialize the Design Variables

*******************************************************************/
  ofstream opHFile("C:\\CompliantMech\\Hvar.txt", ios::trunc|ios::out);

   for(cc=0;cc<num_dv;cc++)
   {
         opHFile<<"H"<<cc+1<<"="<<InHval<<"\n";
   }
  opHFile.close();
```

```
/*********************************************************************
   Initialize the Aerodynamic Forces

*********************************************************************/
  ofstream opAFile("C:\\CompliantMech\\AVal.txt", ios::trunc|ios::out);

   for(cc=0;cc<elem_num+1;cc++)
   {
         opAFile<<"FK,"<<cc+1<<",FY,0,\n";
         opAFile<<"FK,"<<cc+1+3*elem_num<<",FY,0,\n";
   }
  opAFile.close();


/*********************************************************************
    Setup program to use link or beam elements
      1 if link elements
      2 if beam elements

*********************************************************************/
if(element==2)
        strcpy(line,"    ELEM    SMAXI       SMAXJ       VOLUME  ");
else
        strcpy(line,"    ELEM    SAXL       VOLUME  ");

printf("%s\n",line);

/*********************************************************************
   Deletes file, vm1.out from previous run, ANSYS doesn't write over
the file like the manual says, ANSYS will only add to the end of the
file, therefore the file must be deleted
*********************************************************************/
if(iter==1)
      {
      system("cd C:\\CompliantMech\\");
      system("del C:\\CompliantMech\\vm1.out");
      }

/*********************************************************************
    Start the timer
*********************************************************************/
            start=clock();

/* while(iter<maxiter) */    /* Use this outer while loop if you only
                                need to run the program through a
                                certain number of iterations, maxiter*/
while(iteration<maxiter)     /* Use this inner while loop if you want
                                the program to run until the
                                structure has converged */
{
```

```
/*******************************************************************
    This calls ANSYS and runs the batch file,
OptLogfileInForceTemplate.txt and outputs it as vm1.out

*******************************************************************/
system("ansys71 -p ansysrf -b -i
c:\\CompliantMech\\OptLogfileInForceTemplate.txt -o
c:\\CompliantMech\\vm1.out");


/*******************************************************************
    Transfer the values from ANSYS's out file vm1.out of the
    displacement, gradient, volume and constraint vaules to the
    corresponding text files.

      c=300, prevents an infinite loop

*******************************************************************/

  ifstream inFile("C:\\CompliantMech\\vm1.out");

  ofstream opFile("C:\\CompliantMech\\FResultsConstraints.txt",
ios::trunc|ios::out);
  ofstream opFile3("C:\\CompliantMech\\FResultsDisp.txt",
ios::trunc|ios::out);
  ofstream opFile4("C:\\CompliantMech\\FGradient.txt",
ios::trunc|ios::out);
  ofstream opFile5("C:\\CompliantMech\\Volume.txt",
ios::trunc|ios::out);


      if(!inFile)
      {
        cerr<<"File could not be opened\n";
        exit(1);
      }
    while(!inFile.eof())
      {
        inFile.getline(text1,100);
        if(strcmp(line,text1)==0)
        {

while(!inFile.eof()&&strcmp(gap,text1)!=0&&strcmp(space,text1)!=0)
              {
                     inFile.getline(text1,100);
                     if(strcmp(gap,text1)!=0)
                     {
                         opFile << text1 <<"\n";
                     }
                     cc++;
                     if(cc==300){
                           inFile.close();
                           opFile.close();
                           opFile3.close();
                           opFile4.close();
                           opFile5.close();
                           exit(1);
                     }
              }
        }
        else
        if(strcmp(line2,text1)==0)
        {
              while(!inFile.eof()&&strcmp(space,text1)!=0)
              {
                     inFile.getline(text1,100);
                     if(strcmp(space,text1)!=0)
                     {
                         opFile3 << text1 <<"\n";
                     }
                     cc++;
                     if(cc==300)
                     {
                           inFile.close();
                           opFile.close();
                           opFile3.close();
                           opFile4.close();
                           opFile5.close();
                           exit(1);
                     }
              }
        }else
              if(strcmp(line3,text1)==0)
              {
                    for(int i=1;i<=num_dv;i++)
                    {
                      inFile.getline(text2,70);
                      inFile.getline(text3,20);    //getline(text3,10);
                    opFile4<<text3<<"\n";

                          if(cc==300)
                          {
                           inFile.close();
                           opFile.close();
                           opFile3.close();
                           opFile4.close();
                           opFile5.close();
                           exit(1);
                          }
                    }
              }else
                  if(strcmp(line4,text1)==0)
                  {
                      inFile.getline(text2,11);
                      inFile.getline(text3,15);    //getline(text3,10);
                    opFile5<<text3<<"\n";
                if(cc==300)
                          {
                           inFile.close();
                           opFile.close();
                           opFile3.close();
                           opFile4.close();
                           opFile5.close();
                           exit(1);
```

```
                          }
                  }else

      if(strcmp(line5,text1)==0||strcmp(line6,text1)==0)
                        {
                              for(dd=1;dd<3;dd++)
                              {
                              inFile.getline(text1,100);
                              opFile6<<text1<<"\n";
                              }
                        }
      }

 opFile6<<"\n"; // separates each run
  inFile.close();
  opFile.close();
  opFile3.close();
  opFile4.close();
  opFile5.close();


/**********************************************************************
    Read in the values stored in FResultsConstraints.txt

**********************************************************************/

  ifstream inFile2("c:\\CompliantMech\\FResultsConstraints.txt",
ios::in);

  if(element==1)
  {
       for(cc=0;cc<num_dv;cc++)
         inFile2 >>num[cc] >>SAXL[cc] >>Vol[cc];
  }
      else
      {
            for(cc=0;cc<num_dv;cc++)
            {
                  inFile2 >>num[cc] >>SMAXI[cc] >> SMAXJ[cc] >>Vol[cc];
            }
      }

 inFile2.close();


/**********************************************************************
    Read in the values stored in FResultsDisp.txt (only need the tip
node)

**********************************************************************/

   ifstream inFile3("c:\\CompliantMech\\FResultsDisp.txt", ios::in);
   int spacefiller;

   cc=1;
 while(cc<(num_nodes))
 {
      inFile3.getline(text1,100);
    if(cc==(tip_node-1))//||cc==(tip_node+num_nodes-1))
      {
            inFile3 >>spacefiller >>TipDispF;
      }
      cc++;
 }
   inFile3.close();



/***********************************************************************
    Read in the total volume of the structure stored in Volume.txt

***********************************************************************/

  ifstream inFile5("c:\\CompliantMech\\Volume.txt", ios::in);

        inFile5 >>Volume;

  inFile5.close();



/***********************************************************************
     This section determines the angle of tip deflection of the tail
calculated from ANSYS
         that will be used in the aerodynamic analysis

***********************************************************************/

/*  Need to calculate the tip angle from tip deflection due to input
    force and aerodynamic forces */
    double angle=0;
      angle=atan(TipDispF/12.5)*180/PI;
      if(TipDispF>=0)   //Tail is rotated counter-clockwise
            angle=-1*angle;

      angArray[0]=angle;


/***********************************************************************
    This section creates the code to call the standalone Matlab
Aerodynamics code and generates
      the Aerodynamic forces for the ANSYS log file in AVal.txt, ANSYS
has a limit of file name sizes

***********************************************************************/

      chord_ptr=mxCreateDoubleMatrix(1,1, mxREAL);
      memcpy(mxGetPr(chord_ptr),chordArray,1*sizeof(double));

      angle_ptr=mxCreateDoubleMatrix(1,1, mxREAL);
      memcpy(mxGetPr(angle_ptr),angArray,1*sizeof(double));

      elem_num_ptr=mxCreateDoubleMatrix(1,1, mxREAL);
      memcpy(mxGetPr(elem_num_ptr),elemArray,1*sizeof(double));
      keypt_ptr=mxCreateDoubleMatrix(1,1, mxREAL);
      memcpy(mxGetPr(keypt_ptr),keyptArray,1*sizeof(double));

      mmasubLibInitialize();

      /* Call the Matlab compiled function */

aero_ptr=mlfChrisaerocode(chord_ptr,angle_ptr,elem_num_ptr,keypt_ptr);

      mmasubLibTerminate();

      mxDestroyArray(aero_ptr);
      mxDestroyArray(chord_ptr);
      mxDestroyArray(angle_ptr);
      mxDestroyArray(elem_num_ptr);
      mxDestroyArray(keypt_ptr);

 //mlfPrintMatrix(aero_ptr);/*use to print pointers to command window*/


/**********************************************************************
    Objective function evaluation,

      Displacement of tip due to an input force and the aerodynamic
loading

**********************************************************************/
   objfunc=-TipDispF;
   foval=objfunc;       /* Value of the objective function */

   opDispFile<<TipDispF<<"\n";  /* Record Tip displacement due to input
                                    force */

/**********************************************************************
    Constraint function evaluations,

      stress due to force input and aerodynamic loading

**********************************************************************/

   if(element==1)  /* if element is a link */
  {
         for(cc=0;cc<num_dv;cc++)
                SMAX[cc][0]=fabs(SAXL[cc]);
  }
      else /* if element is a beam */
      {
             for(cc=0;cc<num_dv;cc++)
             {
                  if(fabs(SMAXI[cc])>fabs(SMAXJ[cc]))
                  //fabs, absolute value of a float variable
                  {
                          SMAXF[cc][0]=SMAXI[cc];
                  }
                  else
                  {
```

```
                              SMAXF[cc][0]=SMAXJ[cc];
                        }
                  }
      }


/*********************************************************************
    Read in the values stored in FGrad.txt

*********************************************************************/

   ifstream inFile4("c:\\CompliantMech\\FGradient.txt", ios::in);
   ofstream opFileG("C:\\CompliantMech\\Gradient.txt",
ios::trunc|ios::in);

for(cc=0;cc<(num_dv+1);cc++)
{
      for(int dd=0;dd<num_dv;dd++)
      {
            inFile4 >>FGrad[cc][dd];
            Grad[cc][dd]=FGrad[cc][dd];
          opFileG <<Grad[cc][dd]<<"  ";
            if(dd==(num_dv-1))
                  opFileG<<"\n";
      }
 }

inFile4.close();
opFileG.close();


/*********************************************************************
    Assign values to the Method of Moving Asymptote variables

*********************************************************************/

if(iter==1&&begin==1)
{
    for(cc=0;cc<nn;cc++)
   {
         xval[cc][0]=InHval;
         xmin[cc][0]=MinHval;
         xmax[cc][0]=MaxHval;
         xold1[cc][0]=xval[cc][0];
         xold2[cc][0]=xval[cc][0];
         lowin[cc][0]=MinHval;
         uppin[cc][0]=MaxHval;
      }

   for(cc=0;cc<mm;cc++)
   {
         a[cc][0]=0;
         c[cc][0]=1000;
         d[cc][0]=0;
   }
}
```

```
if(iter==1&&begin==2)
{
      /* Read in the values */
      ifstream inFileXval("C:\\CompliantMech\\xvalbegin.txt");

       for(cc=0;cc<num_dv;cc++)
       {
        inFileXval >>num[cc] >>xval[cc][0] >> xold1[cc][0]
>>xold2[cc][0] >>lowin[cc][0] >>uppin[cc][0];
        xmin[cc][0]=MinHval;
        xmax[cc][0]=MaxHval;
       }
   inFileXval.close();

   for(cc=0;cc<mm;cc++)
   {
         a[cc][0]=0;
         c[cc][0]=1000;
         d[cc][0]=0;
   }
}

/*********************************************************************
    This section creates the code to call the standalone Matlab Method
of Moving Asymptotes
      code and generates the optimal solution for the given point

*********************************************************************/

 for(cc=0;cc<num_dv;cc++)
 {
         dfodx[cc][0]=Grad[0][cc];
         dfodx2[cc][0]=0;
 }

  for(cc=1;cc<(mm+1);cc++)
   {
         fval[cc-1][0]=SMAX[cc-1][0]/MaxStress-1;
         for(dd=0;dd<nn;dd++)
         {
                     dfdx[dd][cc-1]=Grad[cc][dd]/MaxStress;
                     dfdx2[dd][cc-1]=0;
         }
   }

   m_ptr=mxCreateDoubleScalar(m);
   n_ptr=mxCreateDoubleScalar(n);
   a0_ptr=mxCreateDoubleScalar(a0);
   iter_ptr=mxCreateDoubleScalar(iter);
   foval_ptr=mxCreateDoubleScalar(foval);

/*Prints out the pointer value to the command window, if need to view*/
/*
   mlfPrintMatrix(m_ptr);
   mlfPrintMatrix(n_ptr);
   mlfPrintMatrix(a0_ptr);
   mlfPrintMatrix(iter_ptr);
   mlfPrintMatrix(foval_ptr);
*/

    cerr<<"iter= ";
    mlfPrintMatrix(iter_ptr);
      cerr<<"\n";

      xmin_ptr=mxCreateDoubleMatrix(nn,1, mxREAL);
      memcpy(mxGetPr(xmin_ptr),xmin,nn*sizeof(double));
//    mlfPrintMatrix(xmin_ptr);

      xmax_ptr=mxCreateDoubleMatrix(nn,1, mxREAL);
      memcpy(mxGetPr(xmax_ptr),xmax,nn*sizeof(double));
//    mlfPrintMatrix(xmax_ptr);

      xold1_ptr=mxCreateDoubleMatrix(nn,1, mxREAL);
      memcpy(mxGetPr(xold1_ptr),xold1,nn*sizeof(double));
//    mlfPrintMatrix(xold1_ptr);

      xold2_ptr=mxCreateDoubleMatrix(nn,1, mxREAL);
      memcpy(mxGetPr(xold2_ptr),xold2,nn*sizeof(double));
//    mlfPrintMatrix(xold2_ptr);

      lowin_ptr=mxCreateDoubleMatrix(nn,1, mxREAL);
      memcpy(mxGetPr(lowin_ptr),lowin,nn*sizeof(double));
//    mlfPrintMatrix(lowin_ptr);

      uppin_ptr=mxCreateDoubleMatrix(nn,1, mxREAL);
      memcpy(mxGetPr(uppin_ptr),uppin,nn*sizeof(double));
//    mlfPrintMatrix(uppin_ptr);

      xval_ptr=mxCreateDoubleMatrix(nn,1, mxREAL);
// Creates a matrix nnx1 and has the pointer, xval_ptr point to it
      memcpy(mxGetPr(xval_ptr),xval,nn*sizeof(double));
// Assigns the value, xval to the pointer, xval_ptr
//    mlfPrintMatrix(xval_ptr);

      a_ptr=mxCreateDoubleMatrix(mm,1, mxREAL);
      memcpy(mxGetPr(a_ptr),a,mm*sizeof(double));
//    mlfPrintMatrix(a_ptr);

      c_ptr=mxCreateDoubleMatrix(mm,1, mxREAL);
      memcpy(mxGetPr(c_ptr),c,mm*sizeof(double));
//    mlfPrintMatrix(c_ptr);

      d_ptr=mxCreateDoubleMatrix(mm,1, mxREAL);
      memcpy(mxGetPr(d_ptr),d,mm*sizeof(double));
//    mlfPrintMatrix(d_ptr);

      dfodx_ptr=mxCreateDoubleMatrix(nn,1, mxREAL);
      memcpy(mxGetPr(dfodx_ptr),dfodx,nn*sizeof(double));
//    mlfPrintMatrix(dfodx_ptr);

      dfodx2_ptr=mxCreateDoubleMatrix(nn,1, mxREAL);
      memcpy(mxGetPr(dfodx2_ptr),dfodx2,nn*sizeof(double));
//    mlfPrintMatrix(dfodx2_ptr);
```

```
      fval_ptr=mxCreateDoubleMatrix(mm,1, mxREAL);
      memcpy(mxGetPr(fval_ptr),fval,mm*sizeof(double));
//    mlfPrintMatrix(fval_ptr);

      dfdx_ptr=mxCreateDoubleMatrix(mm,nn, mxREAL);
      memcpy(mxGetPr(dfdx_ptr),dfdx,mm*nn*sizeof(double));
//    mlfPrintMatrix(dfdx_ptr);

      dfdx2_ptr=mxCreateDoubleMatrix(mm,nn, mxREAL);
      memcpy(mxGetPr(dfdx2_ptr),dfdx2,mm*nn*sizeof(double));
//    mlfPrintMatrix(dfdx2_ptr);


/**********************************************************************
    Call the Method of Moving Asymptotes and get optimized values

**********************************************************************/

   mmasubLibInitialize();   // Initializes mmasubLib libraray

   xmma_ptr=mlfMmasub(&ymma_ptr, &zmma_ptr, &lam_ptr, &xsi_ptr,
&eta_ptr, &mu_ptr, &zet_ptr, &s_ptr, &lowout_ptr, &uppout_ptr, m_ptr,
n_ptr, iter_ptr, xval_ptr, xmin_ptr, xmax_ptr, xold1_ptr, xold2_ptr,
foval_ptr, dfodx_ptr, dfodx2_ptr, fval_ptr, dfdx_ptr, dfdx2_ptr,
lowin_ptr, uppin_ptr, a0_ptr, a_ptr, c_ptr, d_ptr);

   mmasubLibTerminate();    // Terminates mmasubLib libraray

/*  mlfPrintMatrix(xmma_ptr);
      mlfPrintMatrix(ymma_ptr);
      mlfPrintMatrix(zmma_ptr);
      mlfPrintMatrix(lam_ptr);
      mlfPrintMatrix(xsi_ptr);
      mlfPrintMatrix(eta_ptr);
      mlfPrintMatrix(mu_ptr);
      mlfPrintMatrix(zet_ptr);
      mlfPrintMatrix(s_ptr);
      mlfPrintMatrix(lowout_ptr);
      mlfPrintMatrix(uppout_ptr);
*/


/**********************************************************************
Update design variables for next iteration

**********************************************************************/

      mxDestroyArray(xold2_ptr);
      xold2p=mxGetPr(xold1_ptr);
//    xold2_ptr=xold1_ptr;

      mxDestroyArray(xold1_ptr);
      xold1p=mxGetPr(xval_ptr);
//    mlfPrintMatrix(xold1_ptr);

      mxDestroyArray(xval_ptr);
      xvalp=mxGetPr(xmma_ptr);
```

```
//    mlfPrintMatrix(xval_ptr);

      mxDestroyArray(lowin_ptr);
      lowinp=mxGetPr(lowout_ptr);
//    lowin_ptr=lowout_ptr;

      mxDestroyArray(uppin_ptr);
      uppinp=mxGetPr(uppout_ptr);
//    uppin_ptr=uppout_ptr;

        for(cc=0;cc<nn;cc++)
   {
              xold2[cc][0]=xold2p[cc];
              xold1[cc][0]=xold1p[cc];
              xval[cc][0]=xvalp[cc];
            lowin[cc][0]=lowinp[cc];
              uppin[cc][0]=uppinp[cc];

opFileXval<<cc+1<<"\t"<<xval[cc][0]<<"\t"<<xold1[cc][0]<<"\t"<<xold2[cc
][0]<<"\t"<<lowin[cc][0]<<"\t"<<uppin[cc][0]<<"\n";
   }
        opFileXval<<"\n";


/**********************************************************************
   Display results before the next iteration

**********************************************************************/

/* Write Objective Function value to ObjFunc.txt */
    opFuncFile<<foval<<"\n";

/*  Calculate the error in the objective function to be used to
      determine convergence */
      if(iter==1)
      {
            objold=foval;
            objnew=foval;
            error=1;
      }
      else
      {
            objold=objnew;
            objnew=foval;
            error=fabs(objnew-objold)/fabs(objnew);
      }

      if(error<0.001)
        iteration++;
        else
        iteration=0;

  cerr<<"TipF Disp="<<TipDispF<<"
F="<<foval<<"\t"<<"Error="<<error<<"\t"<<"Iteration="<<iteration<<"\n";

      mxDestroyArray(iter_ptr);
      mxDestroyArray(foval_ptr);
      mxDestroyArray(dfodx_ptr);
      mxDestroyArray(dfodx2_ptr);
      mxDestroyArray(fval_ptr);
      mxDestroyArray(dfdx_ptr);
      mxDestroyArray(dfdx2_ptr);

      mxDestroyArray(xmma_ptr);
      mxDestroyArray(ymma_ptr);
      mxDestroyArray(zmma_ptr);
      mxDestroyArray(lam_ptr);
      mxDestroyArray(xsi_ptr);
      mxDestroyArray(eta_ptr);
      mxDestroyArray(mu_ptr);
      mxDestroyArray(zet_ptr);
      mxDestroyArray(s_ptr);
      mxDestroyArray(lowout_ptr);
      mxDestroyArray(uppout_ptr);

      mxDestroyArray(m_ptr);
      mxDestroyArray(n_ptr);
      mxDestroyArray(xmin_ptr);
      mxDestroyArray(xmax_ptr);
      mxDestroyArray(a0_ptr);
      mxDestroyArray(a_ptr);
      mxDestroyArray(c_ptr);
      mxDestroyArray(d_ptr);

      if(iter==1) /* saves vm1.out from first iteration to vm1bck.out
                        to allow quick re-analysis */
      {
            system("cd C:\\CompliantMech\\");
            system("copy C:\\CompliantMech\\vm1.out
C:\\CompliantMech\\vm1bck.out");
            cerr<<"\n";
      }

//    if(iter<(maxiter-1))  /* Outer while loop */
    if(iteration<maxiter) /* Inner while loop */
      {     /* Deletes vm1.out for next iteration */
      system("cd C:\\CompliantMech\\");
      system("del C:\\CompliantMech\\vm1.out"); }

iter++;

}

  finish=clock();
  duration = (double)(finish - start) / CLOCKS_PER_SEC;
  cerr<<"Time elapsed is "<<duration<<" in seconds or "<<duration/60<<"
minutes\n";

  opFuncFile.close();
  opFile6.close();
  opFileXval.close();
  opDispFile.close();
```

```
/*********************************************************************
   Process Stress File to easily plot using the Results.m file

*********************************************************************/

    char text1R[100],text2R[70], text3R[20];
        char lineR[]=" VALUE";
        int counter=1;

      ifstream inFileR("C:\\CompliantMech\\FStress.txt");

      ofstream
opFileR("C:\\CompliantMech\\ResultsMinusU\\ABSDV46\\FStressMatF8mn005mx
03125bc4.txt", ios::trunc|ios::out);

        if(!inFileR)
      {
        cerr<<"File (a) could not be opened\n";
        exit(1);
      }
    while(!inFileR.eof())
      {
        inFileR.getline(text1R,7);
        if(strcmp(lineR,text1R)==0)
        {
              if(counter>1)
              {
                    inFileR.getline(text2R,14);
                  inFileR.getline(text3R,14);
                    opFileR<<text2R<<"\t"<<text3R<<"\n";
                    counter=1;
               }
              else
              {
                    inFileR.getline(text2R,14);
                  inFileR.getline(text3R,14);
                    opFileR<<text2R<<"\t"<<text3R<<"\t";
                    counter++;
              }
        }
      }

      inFileR.close();
      opFileR.close();

  return 1;
}
```

Appendix C

Method of Moving Asymptotes

The Method of Moving Asymptotes program was developed by Svanberg () and is distributed freely for research use. To obtain a copy of the program Svanberg's contact information is listed below.

Written in May 1999 by
Krister Svanberg <krille@math.kth.se>
Department of Mathematics
SE-10044 Stockholm, Sweden.

Appendix D

Design Variable Values

In this appendix, the design variable values for each iteration is presented for the all three optimization runs presented in Chapter 4; $h_{initial} = 0.03, h_{initial} = 0.008$ and $h_{initial} = 0.12$.

Design Variables	**Initial starting point = 0.03 in. Iterations**										
	1	2	3	4	5	6	7	8	9	10	11
1	0.0300	0.0060	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
2	0.0300	0.0056	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
3	0.0300	0.0162	0.0414	0.0573	0.0597	0.0625	0.0616	0.0447	0.0193	0.0050	0.0050
4	0.0300	0.0311	0.0562	0.0625	0.0617	0.0625	0.0622	0.0625	0.0625	0.0625	0.0625
5	0.0300	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
6	0.0300	0.0531	0.0625	0.0625	0.0621	0.0625	0.0624	0.0625	0.0625	0.0625	0.0625
7	0.0300	0.0372	0.0576	0.0625	0.0610	0.0625	0.0622	0.0625	0.0625	0.0625	0.0570
8	0.0300	0.0281	0.0497	0.0624	0.0608	0.0625	0.0621	0.0625	0.0625	0.0625	0.0625
9	0.0300	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
10	0.0300	0.0524	0.0303	0.0472	0.0551	0.0625	0.0622	0.0625	0.0625	0.0625	0.0625
11	0.0300	0.0302	0.0509	0.0303	0.0119	0.0050	0.0052	0.0050	0.0188	0.0273	0.0364
12	0.0300	0.0269	0.0488	0.0625	0.0595	0.0625	0.0568	0.0511	0.0430	0.0331	0.0208
13	0.0300	0.0054	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
14	0.0300	0.0062	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
15	0.0300	0.0415	0.0605	0.0434	0.0252	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
16	0.0300	0.0341	0.0590	0.0625	0.0621	0.0625	0.0624	0.0625	0.0625	0.0625	0.0625
17	0.0300	0.0115	0.0336	0.0220	0.0336	0.0264	0.0320	0.0348	0.0309	0.0337	0.0331
18	0.0300	0.0052	0.0050	0.0234	0.0445	0.0625	0.0622	0.0625	0.0625	0.0625	0.0625
19	0.0300	0.0431	0.0620	0.0625	0.0615	0.0625	0.0623	0.0625	0.0625	0.0625	0.0625
20	0.0300	0.0306	0.0519	0.0625	0.0583	0.0625	0.0615	0.0625	0.0625	0.0625	0.0625
21	0.0300	0.0067	0.0050	0.0050	0.0051	0.0050	0.0051	0.0050	0.0050	0.0050	0.0050
22	0.0300	0.0242	0.0054	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
23	0.0300	0.0259	0.0095	0.0299	0.0470	0.0625	0.0619	0.0625	0.0625	0.0625	0.0625
24	0.0300	0.0300	0.0227	0.0066	0.0051	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
25	0.0300	0.0145	0.0300	0.0119	0.0056	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
26	0.0300	0.0052	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
27	0.0300	0.0417	0.0184	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
28	0.0300	0.0301	0.0092	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
29	0.0300	0.0540	0.0625	0.0625	0.0623	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
30	0.0300	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
31	0.0300	0.0078	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
32	0.0300	0.0541	0.0624	0.0625	0.0622	0.0625	0.0624	0.0625	0.0625	0.0625	0.0625
33	0.0300	0.0172	0.0413	0.0279	0.0356	0.0433	0.0519	0.0625	0.0625	0.0625	0.0625
34	0.0300	0.0408	0.0611	0.0625	0.0598	0.0625	0.0617	0.0625	0.0625	0.0625	0.0625
35	0.0300	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
36	0.0300	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
37	0.0300	0.0187	0.0441	0.0574	0.0463	0.0337	0.0167	0.0050	0.0050	0.0050	0.0050
38	0.0300	0.0315	0.0058	0.0050	0.0051	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
39	0.0300	0.0358	0.0541	0.0625	0.0582	0.0625	0.0613	0.0625	0.0625	0.0625	0.0625
40	0.0300	0.0310	0.0532	0.0625	0.0602	0.0625	0.0622	0.0625	0.0625	0.0625	0.0625
41	0.0300	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
42	0.0300	0.0548	0.0625	0.0625	0.0621	0.0625	0.0624	0.0625	0.0625	0.0625	0.0625
43	0.0300	0.0531	0.0625	0.0625	0.0622	0.0625	0.0624	0.0625	0.0625	0.0625	0.0625
44	0.0300	0.0519	0.0263	0.0442	0.0562	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
45	0.0300	0.0229	0.0470	0.0625	0.0619	0.0625	0.0624	0.0625	0.0625	0.0625	0.0625
46	0.0300	0.0407	0.0506	0.0625	0.0616	0.0625	0.0621	0.0625	0.0566	0.0524	0.0468

Design Variables	Initial starting point = 0.03 in. Iterations										
	12	13	14	15	16	17	18	19	20	21	22
1	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
2	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
3	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
4	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
5	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
6	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
7	0.0495	0.0398	0.0276	0.0125	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
8	0.0625	0.0625	0.0625	0.0625	0.0614	0.0591	0.0557	0.0510	0.0450	0.0374	0.0279
9	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
10	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
11	0.0429	0.0280	0.0176	0.0061	0.0237	0.0121	0.0050	0.0050	0.0050	0.0050	0.0050
12	0.0057	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
13	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
14	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
15	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
16	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
17	0.0337	0.0336	0.0337	0.0337	0.0337	0.0338	0.0338	0.0338	0.0339	0.0339	0.0339
18	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
19	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
20	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
21	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
22	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
23	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0616
24	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
25	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
26	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
27	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
28	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
29	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
30	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
31	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
32	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
33	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
34	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
35	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
36	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
37	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
38	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
39	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
40	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
41	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
42	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
43	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
44	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
45	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
46	0.0398	0.0310	0.0204	0.0098	0.0282	0.0203	0.0280	0.0334	0.0398	0.0473	0.0562

Design Variables	**Initial starting point = 0.008 in. Iterations**									
	1	2	3	4	5	6	7	8	9	10
1	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
2	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
3	0.0080	0.0053	0.0310	0.0131	0.0054	0.0050	0.0050	0.0050	0.0050	0.0050
4	0.0080	0.0202	0.0458	0.0625	0.0620	0.0625	0.0622	0.0625	0.0625	0.0625
5	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
6	0.0080	0.0337	0.0594	0.0625	0.0624	0.0625	0.0624	0.0625	0.0625	0.0625
7	0.0080	0.0292	0.0524	0.0625	0.0619	0.0625	0.0621	0.0624	0.0625	0.0625
8	0.0080	0.0089	0.0328	0.0584	0.0613	0.0625	0.0586	0.0615	0.0625	0.0625
9	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
10	0.0080	0.0336	0.0117	0.0050	0.0051	0.0050	0.0051	0.0051	0.0050	0.0050
11	0.0080	0.0158	0.0393	0.0098	0.0054	0.0050	0.0054	0.0051	0.0050	0.0050
12	0.0080	0.0077	0.0325	0.0486	0.0578	0.0625	0.0615	0.0624	0.0625	0.0625
13	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
14	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
15	0.0080	0.0314	0.0561	0.0625	0.0593	0.0360	0.0069	0.0050	0.0050	0.0050
16	0.0080	0.0264	0.0517	0.0625	0.0623	0.0625	0.0618	0.0624	0.0625	0.0625
17	0.0080	0.0276	0.0516	0.0212	0.0418	0.0272	0.0367	0.0309	0.0354	0.0334
18	0.0080	0.0050	0.0307	0.0488	0.0616	0.0625	0.0618	0.0624	0.0625	0.0625
19	0.0080	0.0319	0.0537	0.0625	0.0616	0.0625	0.0585	0.0616	0.0625	0.0625
20	0.0080	0.0190	0.0431	0.0625	0.0606	0.0625	0.0610	0.0623	0.0625	0.0625
21	0.0080	0.0053	0.0051	0.0050	0.0051	0.0050	0.0052	0.0050	0.0050	0.0050
22	0.0080	0.0066	0.0053	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
23	0.0080	0.0071	0.0050	0.0183	0.0056	0.0050	0.0050	0.0050	0.0050	0.0050
24	0.0080	0.0159	0.0376	0.0092	0.0052	0.0193	0.0335	0.0526	0.0625	0.0625
25	0.0080	0.0326	0.0117	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
26	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
27	0.0080	0.0314	0.0066	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
28	0.0080	0.0156	0.0056	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
29	0.0080	0.0335	0.0590	0.0625	0.0624	0.0625	0.0624	0.0625	0.0625	0.0625
30	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
31	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
32	0.0080	0.0338	0.0571	0.0625	0.0620	0.0625	0.0598	0.0619	0.0625	0.0625
33	0.0080	0.0054	0.0309	0.0130	0.0053	0.0050	0.0050	0.0050	0.0050	0.0050
34	0.0080	0.0311	0.0557	0.0625	0.0619	0.0625	0.0622	0.0625	0.0625	0.0625
35	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
36	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
37	0.0080	0.0055	0.0310	0.0194	0.0082	0.0050	0.0051	0.0050	0.0050	0.0050
38	0.0080	0.0204	0.0051	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
39	0.0080	0.0281	0.0484	0.0625	0.0606	0.0625	0.0611	0.0623	0.0625	0.0625
40	0.0080	0.0199	0.0448	0.0625	0.0611	0.0625	0.0618	0.0624	0.0625	0.0625
41	0.0080	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
42	0.0080	0.0303	0.0553	0.0625	0.0623	0.0625	0.0623	0.0625	0.0625	0.0625
43	0.0080	0.0337	0.0591	0.0625	0.0624	0.0625	0.0624	0.0625	0.0625	0.0625
44	0.0080	0.0335	0.0584	0.0625	0.0624	0.0625	0.0624	0.0625	0.0625	0.0625
45	0.0080	0.0061	0.0316	0.0476	0.0614	0.0625	0.0618	0.0624	0.0625	0.0625
46	0.0080	0.0311	0.0161	0.0336	0.0452	0.0601	0.0621	0.0625	0.0625	0.0625

Design Variables	**Initial starting point = 0.008 in. Iterations**									
	11	12	13	14	15	16	17	18	19	20
1	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
2	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
3	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
4	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
5	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
6	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
7	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
8	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
9	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
10	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
11	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
12	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
13	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
14	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
15	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
16	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
17	0.0348	0.0344	0.0346	0.0346	0.0346	0.0346	0.0346	0.0346	0.0346	0.0346
18	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
19	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
20	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
21	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
22	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
23	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
24	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
25	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
26	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
27	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
28	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
29	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
30	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
31	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
32	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
33	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
34	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
35	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
36	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
37	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
38	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
39	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
40	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
41	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
42	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
43	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
44	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
45	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
46	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625

Design Variables	**Initial starting point = 0.12 in. Iterations**									
	1	2	3	4	5	6	7	8	9	10
1	0.1200	0.0782	0.0539	0.0243	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
2	0.1200	0.0783	0.0537	0.0239	0.0051	0.0050	0.0050	0.0050	0.0050	0.0050
3	0.1200	0.0783	0.0625	0.0589	0.0604	0.0568	0.0625	0.0619	0.0625	0.0622
4	0.1200	0.0783	0.0625	0.0608	0.0612	0.0610	0.0625	0.0617	0.0625	0.0622
5	0.1200	0.0782	0.0523	0.0213	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
6	0.1200	0.0783	0.0625	0.0618	0.0621	0.0620	0.0625	0.0620	0.0625	0.0623
7	0.1200	0.0783	0.0625	0.0549	0.0502	0.0544	0.0625	0.0586	0.0625	0.0622
8	0.1200	0.0783	0.0625	0.0568	0.0560	0.0568	0.0625	0.0607	0.0625	0.0623
9	0.1200	0.0782	0.0523	0.0212	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
10	0.1200	0.0783	0.0625	0.0616	0.0619	0.0602	0.0625	0.0598	0.0625	0.0623
11	0.1200	0.0783	0.0624	0.0494	0.0364	0.0123	0.0050	0.0053	0.0050	0.0051
12	0.1200	0.0783	0.0625	0.0568	0.0568	0.0567	0.0625	0.0606	0.0625	0.0608
13	0.1200	0.0782	0.0529	0.0225	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
14	0.1200	0.0783	0.0552	0.0264	0.0051	0.0050	0.0050	0.0050	0.0050	0.0050
15	0.1200	0.0783	0.0625	0.0595	0.0593	0.0606	0.0341	0.0225	0.0050	0.0051
16	0.1200	0.0783	0.0625	0.0608	0.0610	0.0611	0.0625	0.0619	0.0625	0.0624
17	0.1200	0.0782	0.0539	0.0261	0.0542	0.0293	0.0250	0.0447	0.0297	0.0388
18	0.1200	0.0782	0.0525	0.0219	0.0050	0.0050	0.0256	0.0618	0.0625	0.0621
19	0.1200	0.0783	0.0625	0.0598	0.0605	0.0605	0.0625	0.0614	0.0625	0.0623
20	0.1200	0.0783	0.0625	0.0553	0.0542	0.0536	0.0625	0.0578	0.0625	0.0610
21	0.1200	0.0783	0.0526	0.0226	0.0052	0.0051	0.0050	0.0051	0.0050	0.0051
22	0.1200	0.0783	0.0623	0.0460	0.0213	0.0057	0.0079	0.0050	0.0050	0.0050
23	0.1200	0.0783	0.0624	0.0474	0.0330	0.0575	0.0310	0.0489	0.0624	0.0621
24	0.1200	0.0783	0.0624	0.0492	0.0363	0.0173	0.0050	0.0051	0.0050	0.0050
25	0.1200	0.0782	0.0540	0.0262	0.0064	0.0498	0.0187	0.0060	0.0050	0.0050
26	0.1200	0.0783	0.0525	0.0218	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
27	0.1200	0.0783	0.0623	0.0452	0.0194	0.0056	0.0050	0.0050	0.0050	0.0050
28	0.1200	0.0783	0.0624	0.0471	0.0257	0.0079	0.0050	0.0050	0.0050	0.0050
29	0.1200	0.0782	0.0625	0.0621	0.0623	0.0618	0.0625	0.0623	0.0625	0.0625
30	0.1200	0.0782	0.0523	0.0213	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
31	0.1200	0.0783	0.0574	0.0308	0.0053	0.0050	0.0050	0.0050	0.0050	0.0050
32	0.1200	0.0783	0.0625	0.0621	0.0622	0.0621	0.0625	0.0622	0.0625	0.0624
33	0.1200	0.0783	0.0623	0.0486	0.0530	0.0365	0.0243	0.0432	0.0576	0.0617
34	0.1200	0.0783	0.0625	0.0571	0.0525	0.0581	0.0625	0.0369	0.0149	0.0438
35	0.1200	0.0782	0.0523	0.0213	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
36	0.1200	0.0782	0.0523	0.0213	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
37	0.1200	0.0783	0.0623	0.0474	0.0570	0.0607	0.0625	0.0594	0.0624	0.0469
38	0.1200	0.0783	0.0624	0.0497	0.0234	0.0060	0.0552	0.0213	0.0050	0.0050
39	0.1200	0.0783	0.0624	0.0528	0.0413	0.0492	0.0625	0.0484	0.0624	0.0612
40	0.1200	0.0783	0.0624	0.0525	0.0494	0.0247	0.0624	0.0362	0.0596	0.0617
41	0.1200	0.0782	0.0523	0.0213	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
42	0.1200	0.0783	0.0625	0.0623	0.0624	0.0622	0.0625	0.0620	0.0625	0.0624
43	0.1200	0.0783	0.0625	0.0619	0.0622	0.0621	0.0625	0.0622	0.0625	0.0624
44	0.1200	0.0783	0.0625	0.0614	0.0389	0.0056	0.0554	0.0620	0.0625	0.0625
45	0.1200	0.0783	0.0624	0.0555	0.0560	0.0423	0.0599	0.0615	0.0625	0.0624
46	0.1200	0.0783	0.0625	0.0583	0.0569	0.0598	0.0625	0.0613	0.0625	0.0622

Design Variables	**Initial starting point = 0.12 in. Iterations**									
	11	12	13	14	15	16	17	18	19	20
1	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
2	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
3	0.0625	0.0625	0.0619	0.0601	0.0579	0.0551	0.0516	0.0472	0.0417	0.0348
4	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
5	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
6	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
7	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0623	0.0502
8	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
9	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
10	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
11	0.0050	0.0050	0.0050	0.0166	0.0302	0.0461	0.0624	0.0625	0.0625	0.0625
12	0.0624	0.0580	0.0536	0.0470	0.0388	0.0287	0.0162	0.0050	0.0050	0.0050
13	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
14	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
15	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
16	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
17	0.0320	0.0354	0.0327	0.0336	0.0330	0.0331	0.0332	0.0332	0.0333	0.0334
18	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
19	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
20	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
21	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
22	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
23	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
24	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
25	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
26	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
27	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
28	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
29	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
30	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
31	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
32	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
33	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
34	0.0624	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
35	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
36	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
37	0.0351	0.0203	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
38	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
39	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
40	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
41	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050
42	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
43	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
44	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
45	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625
46	0.0625	0.0625	0.0625	0.0577	0.0513	0.0430	0.0321	0.0182	0.0050	0.0050

Design Variables	**Initial starting point = 0.12 in. Iterations**				
	21	22	23	24	25
1	0.0050	0.0050	0.0050	0.0050	0.0050
2	0.0050	0.0050	0.0050	0.0050	0.0050
3	0.0262	0.0154	0.0050	0.0050	0.0050
4	0.0625	0.0625	0.0625	0.0625	0.0625
5	0.0050	0.0050	0.0050	0.0050	0.0050
6	0.0625	0.0625	0.0625	0.0625	0.0625
7	0.0287	0.0050	0.0050	0.0050	0.0050
8	0.0625	0.0625	0.0625	0.0608	0.0586
9	0.0050	0.0050	0.0050	0.0050	0.0050
10	0.0625	0.0625	0.0625	0.0625	0.0625
11	0.0625	0.0625	0.0625	0.0625	0.0625
12	0.0050	0.0050	0.0050	0.0050	0.0050
13	0.0050	0.0050	0.0050	0.0050	0.0050
14	0.0050	0.0050	0.0050	0.0050	0.0050
15	0.0050	0.0050	0.0050	0.0050	0.0050
16	0.0625	0.0625	0.0625	0.0625	0.0625
17	0.0334	0.0335	0.0336	0.0338	0.0339
18	0.0625	0.0625	0.0625	0.0625	0.0625
19	0.0625	0.0625	0.0625	0.0625	0.0625
20	0.0625	0.0625	0.0625	0.0625	0.0625
21	0.0050	0.0050	0.0050	0.0050	0.0050
22	0.0050	0.0050	0.0050	0.0050	0.0050
23	0.0625	0.0625	0.0625	0.0625	0.0625
24	0.0050	0.0050	0.0050	0.0050	0.0050
25	0.0050	0.0050	0.0050	0.0050	0.0050
26	0.0050	0.0050	0.0050	0.0050	0.0050
27	0.0050	0.0050	0.0050	0.0050	0.0050
28	0.0050	0.0050	0.0050	0.0050	0.0050
29	0.0625	0.0625	0.0625	0.0625	0.0625
30	0.0050	0.0050	0.0050	0.0050	0.0050
31	0.0050	0.0050	0.0050	0.0050	0.0050
32	0.0625	0.0625	0.0625	0.0625	0.0625
33	0.0625	0.0625	0.0625	0.0625	0.0625
34	0.0625	0.0625	0.0625	0.0625	0.0625
35	0.0050	0.0050	0.0050	0.0050	0.0050
36	0.0050	0.0050	0.0050	0.0050	0.0050
37	0.0050	0.0050	0.0050	0.0050	0.0050
38	0.0050	0.0050	0.0050	0.0050	0.0050
39	0.0625	0.0625	0.0625	0.0625	0.0625
40	0.0625	0.0625	0.0625	0.0625	0.0625
41	0.0050	0.0050	0.0050	0.0050	0.0050
42	0.0625	0.0625	0.0625	0.0625	0.0625
43	0.0625	0.0625	0.0625	0.0625	0.0625
44	0.0625	0.0625	0.0625	0.0625	0.0625
45	0.0625	0.0625	0.0625	0.0625	0.0625
46	0.0050	0.0050	0.0050	0.0050	0.0050

Vita

Matthew G. Good was born on September 12, 1978 to Tim and Mary Good of Madison, Wisconsin. He graduated from Hudson High School in June 1997 and began his college career at Ohio University. During his undergraduate years, he completed three quarters of co-op experience with Structural Dynamics Research Corporation (SDRC). After graduating from Ohio University with a B.S. in Mechanical Engineering in 2002, he began graduate school at Virginia Tech later that fall. At Virginia Tech, Matt completed his research under Dr. Daniel J. Inman with the Center of Intelligent Material Systems and Structures (CIMSS) in the morphing airplane program.