Spring 2021

Design and Testing of a Feed-Forward Control System for Deployable Vortex Generators Dependent on Angle of Attack

Solomon B. Whitmire
The University of Akron, sw110@zips.uakron.edu

Christopher J. Chapanar
The University of Akron, cjc172@zips.uakron.edu

Kirklin M. Anderson
The University of Akron, kma145@zips.uakron.edu

Nickalus R. Amon
The University of Akron, nra23@zips.uakron.edu

Daniel W. Chech
The University of Akron, dwc35@zips.uakron.edu

DESIGN AND TESTING OF A FEED-FORWARD CONTROL SYSTEM FOR DEPLOYABLE VORTEX GENERATORS DEPENDENT ON ANGLE OF ATTACK

By

Solomon Whitmire

Christopher Chapanar

Kirklin Anderson

Nickalus Amon

Daniel Chech

Final Report for 4600:497 / 4900:490 Spring 2021

Faculty/Honors Sponsor: Dr. Manigandan Kannan

Faculty/Honors Advisor: Dr. Scott Sawyer

Faculty/Honors Reader 1: David Warther III

Faculty/Honors Reader 2: Ray Roos

30th April 2021

Abstract

A vortex generator (VG hereafter) is a common feature of an aircraft wing that disturbs the flow on the leading edge of the wing, thus energizing the boundary layer and reducing flow separation. For an aircraft experiencing flow separation, VGs can increase the lift-to-drag ratio of the wing and prevent stall; however, if flow separation isn't an issue, the unnecessary frontal area of the VGs has the potential to produce parasitic drag. This study seeks to determine whether the use of a deployment system can improve the performance of VG's by raising or lowering them depending on the angle of attack of the wing. Using wind tunnel testing, a feed-forward control deployment system was developed which improved the lift to drag ratio for some angles of attack, and it was determined that further development could potentially produce a system with significant improvements in aircraft efficiency.

Contents

1. Introduction

1.1 Boundary Layer Separation

When a fluid is flowing around a body, it forms a boundary layer, which is a region around the body where flow must be treated as viscous [8]. This region of viscous flow, under certain conditions, can separate from the body in an occurrence known as boundary layer separation. Separation occurs when the fluid loses energy due to friction, causing the pressure gradient along the surface of the body to decrease until it eventually reverses at a point known as the point of separation [8]. When the body being considered is a lift producing airfoil, boundary layer separation can severely decrease the amount of lift produced by the airfoil in an occurrence known as stall [8].

1.2 Vortex Generators

As previously explained, boundary layer separation can be catastrophic to an airfoil. Thus, solutions are often implemented to reduce the likelihood of separation. One of the most common methods of delaying separation is to cause turbulence in the boundary layer [8]. Because turbulent flow contains more kinetic energy than laminar flow, a turbulent boundary layer can potentially provide the energy necessary to avoid the formation of the adverse pressure gradient that causes separation [8]. This delay in separation can be observed in figure 1 below.

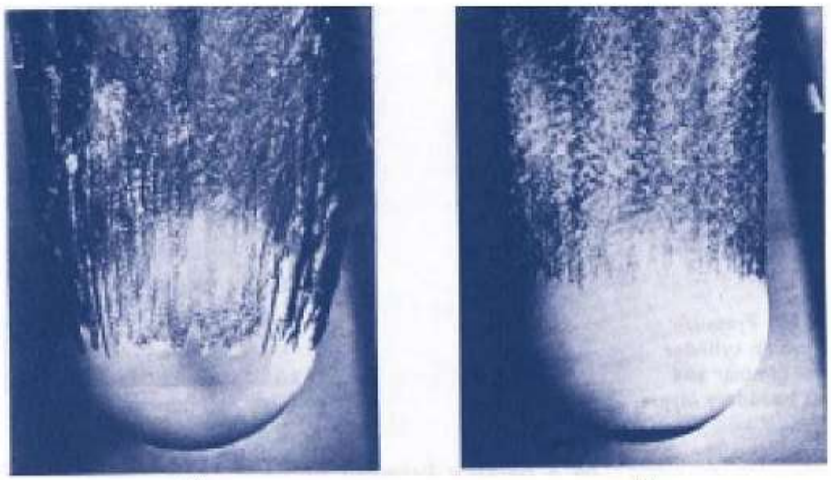

Figure 1 The effect of a turbulent boundary layer on separation. The image on the left side of this figure shows flow separation around a sphere with a laminar boundary layer. In the image to the right, the boundary layer has been made turbulent, thus visibly delaying separation [8]

In order to achieve a turbulent boundary layer, vortex generators are often placed along the wing of an aircraft. Vortex generators are a type of boundary layer control device, which is a general term for any device which can be used to influence the boundary layer [10]. Typically, vortex generators consist of small, inclined vanes, typically rectangular or triangular in shape, which create vortices that mix high energy free stream flow into the lower energy boundary layer [2]. An example of a vortex generator array on a wing can be seen in figure 2 below.

Figure 2 Vortex Generators on the wing of a 737-800 [11]

1.3 Important Factors

As stated in *Fundamentals of Fluid Mechanics,* separation occurs over an airfoil because the angle of attack of the airfoil is too large [8]. The angle of attack of an airfoil is the angle between the free stream flow and the chord line of the airfoil [10]. In other words, if flow is horizontal, the angle of attack is the angle at which an airfoil is inclined.

It is known that vortex generators are beneficial when this angle is too large because they can delay separation; however, it is also known that a turbulent boundary layer creates more skin friction drag than a laminar boundary layer [10]. Because of this, it can be assumed that, when the angle of attack of the wing is not large enough to cause separation, vortex generators actually do more harm than good, by producing friction drag without benefitting the lift of the airfoil. If this is the case, it should be observable in the lift-to-drag ratio of the wing, which is a performance parameter that is simply the amount of lift (vertical force) produced by the wing, divided by the amount of drag (horizontal force) produced. If lift is not improved but drag is increased, then the lift to drag ratio of a wing with unnecessary vortex generators should be smaller than that of a wing with no vortex generators because the latter would have the same numerator (lift) with a smaller denominator (drag).

1.4 Proposed Solution

Throughout the course of any flight, the wing will typically experience many different angles of attack. For example, the wing may need to be at a steeper angle at takeoff and landing than when cruising. For this reason, there are likely to be some flight conditions where vortex generators improve the lift to drag ratio and others where they decrease it, all within the same flight. Even if this increase in drag is not very large, it can be assumed that, over the course of hundreds of hours of flight, a large loss could be taken in terms of fuel economy because of the need to overcome the drag with additional thrust from the power plant. However, simply removing the vortex generators could cause safety concerns due to their effectiveness at reducing wing stall, outlined previously.

For this reason, a system was designed which deploys vortex generators to the optimal height depending on the angle of attack of the wing, which could potentially reduce any negative effects due to skin drag. The system was tested in a wind tunnel in an attempt to determine what level of deployment, from 0-100 percent deployed in increments of 10% was optimal at angles of attack ranging from -12 to 6 degrees in increments of 2 degrees, and from 6 degrees to 24 degrees in increments of 1 degree. Then, a curve fit was generated for percent deployment as a function of angle of attack, and the final feed-forward control system was again tested and compared to a control with no system deployed.

2. Design

An initial survey of available resources and skill proficiencies guided the overall design process. The physical structure needed to be stiff enough to withstand forces in the wind tunnel, while also providing a large enough interior for the necessary components. Additionally, the structure needed to be manufacturable, cost effective, and be completed quickly to facilitate adequate time for testing. It was decided that the wing structure would be made by use of a FDM 3D printer, since this option had the most desirable compromises. The question was then turned to computational and mechanical power of the system, which had similar requirements to that of the wing structure. It was determined that a simple microcontroller, breadboard, and small servo motors would suffice. The system needed only one input, that of the pitch of the system, which would be achieved by a gyroscope wired into the microcontroller via the breadboard.

To achieve the desired deployment effect, a lifting mechanism was developed that would utilize two servo motors connected in tandem by a rod, see figure 3 below. The gyroscope would detect the attitude of the system and send the system to the microcontroller. The microcontroller would take that value, compare it to a known curve of deployment height versus angle of attack and select an appropriate value of deployment. That value would then be converted to a servo command that would move the vortex generators up to the appropriate position.

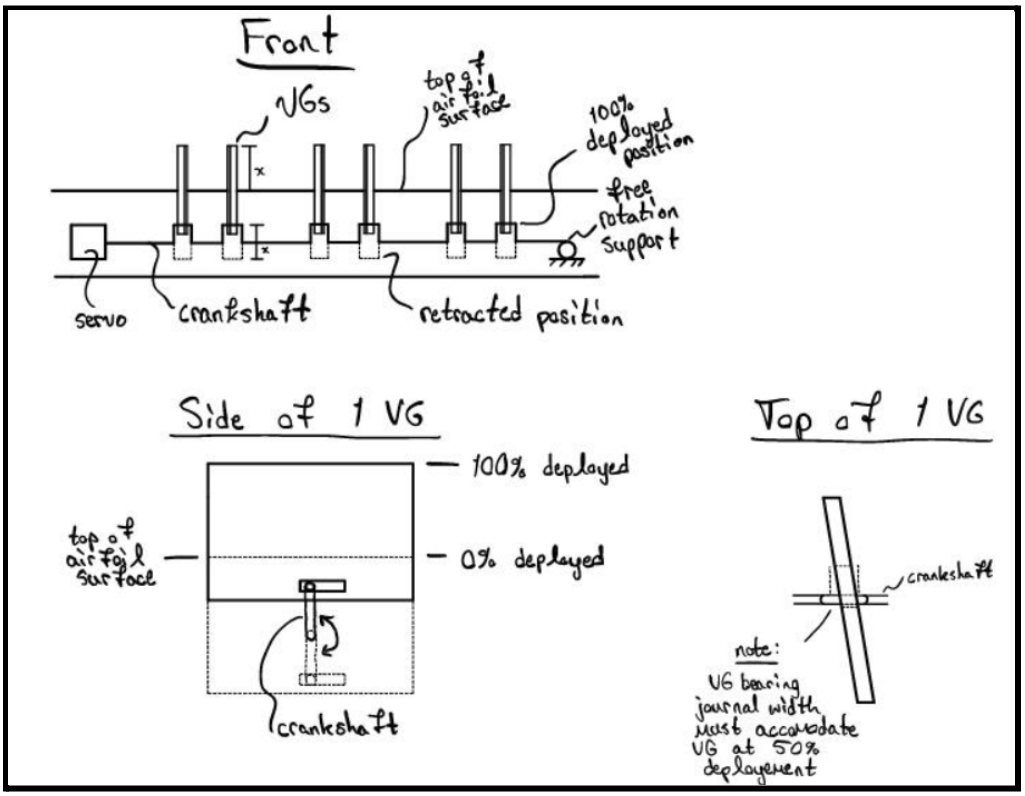

Figure 3 Initial sketch of the lifting mechanism

2.1 Theoretical Basis and Sizing

It was desired that the concept of deployable VGs incorporated a baseline. In order to achieve this the team decided that rather than performing tests to find an airfoil that required VGs, it was more efficient to select an aircraft that already has VGs in use. The airfoil selected was the USA-35B, which is the airfoil used on the Piper PA-18 Carbon Cub SS. The overall span of the wing for testing was maximized to the walls of the wind tunnel testing area. This was done for the assumption of recording data with an infinite wing, as well as to mitigate the effects of wing tip vortices. The team sized the chord of the wing and chose the airspeed for data recording so as not to overload the wind tunnel sting, or damage the wing test section, as it was additively manufactured. The thickness, dependent on the chord of the wing, was also taken into account such that we needed to be able to fit the electronics within the overall design as well. The final parameters for the wing turned out to be a 12 inch chord with an 18 inch span. The airspeed of the tunnel was determined to be 30 mph giving a Reynold's number of 287,727 which was much lower than desired.

The VGs were sized by referencing multiple sources [5] see also: [6]; [7]; [12]. The chord of the VGs was set at a value of 7% of the overall wing chord. This value was given by the flitetest article as we were not able to find other data sources. The VGs were placed such that the leading edge of the VGs were located at the point where the boundary layer transitioned from laminar to turbulent flow for an angle of attack slightly less than the critical angle of attack for the wing. The transition point was found using XFOIL which can be seen as the pressure drop on the upper surface below in figure 4. It is also precisely defined in the XFOIL command window, see figure 5.

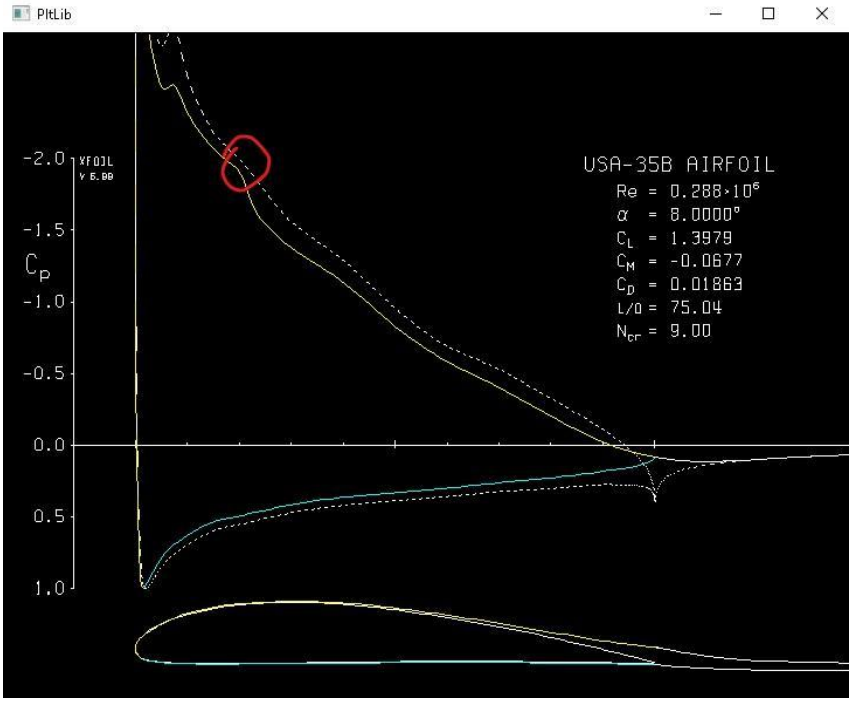

Figure 4 XFOIL visual for 8 degree angle of attack, indicating boundary layer transition point in red circle

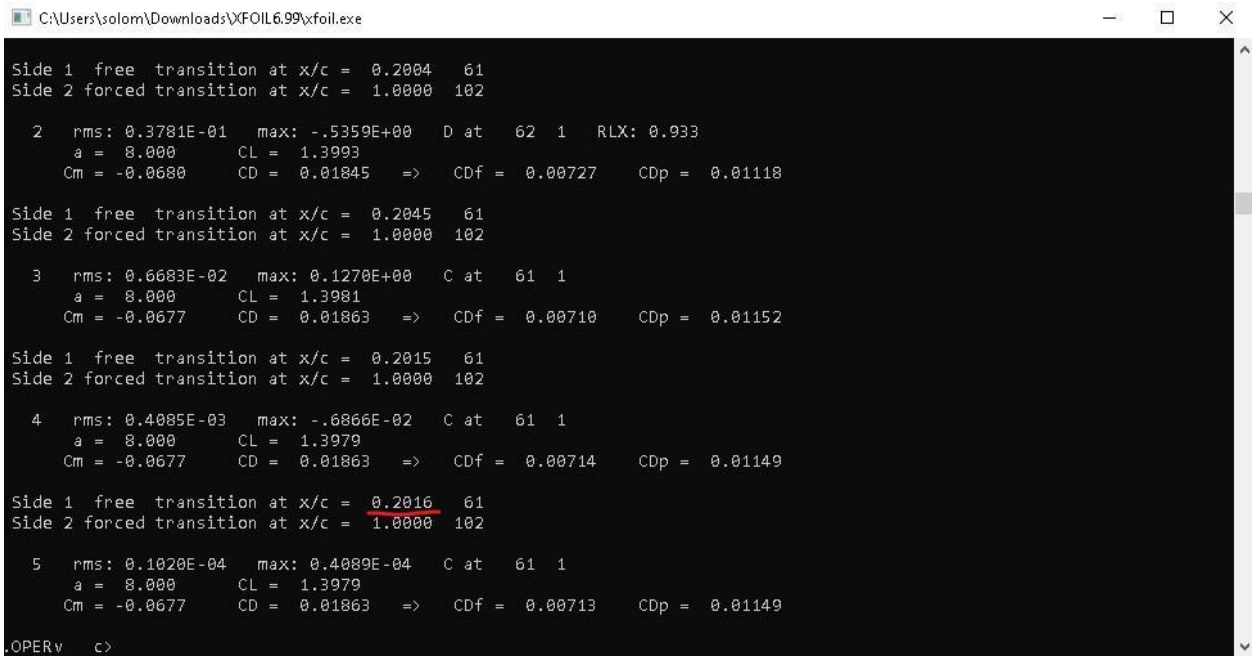

Figure 5 XFOIL results for 8 degree angle of attack, indicating boundary layer transition point in red underline

This value is in percent of the chord. This allowed the height of the VGs to be equal to the thickness of the boundary layer right before the transition. This thickness, delta, can be found using the equation below for a laminar boundary layer [7].

$$\delta = \frac{5.0x}{\sqrt{Re}}$$

Where x is designated as the location of the transition point along the chord, and Re is the Reynold's number. Even though this value was calculated to be 0.023 inches, a variable height was desired and with the servo motors and manufacturing methods available it was decided that the overall deployment of the VGs would be 0.25 inches. A research paper on the effects of VG installation angle [6] was used to find the incidence angle for the vortex generators. The method described in the paper uses the ratio of the distance from the trailing edge of the VG to the point of boundary layer separation. The separation point of the boundary layer was found using XFOIL, as seen below in figure 6.

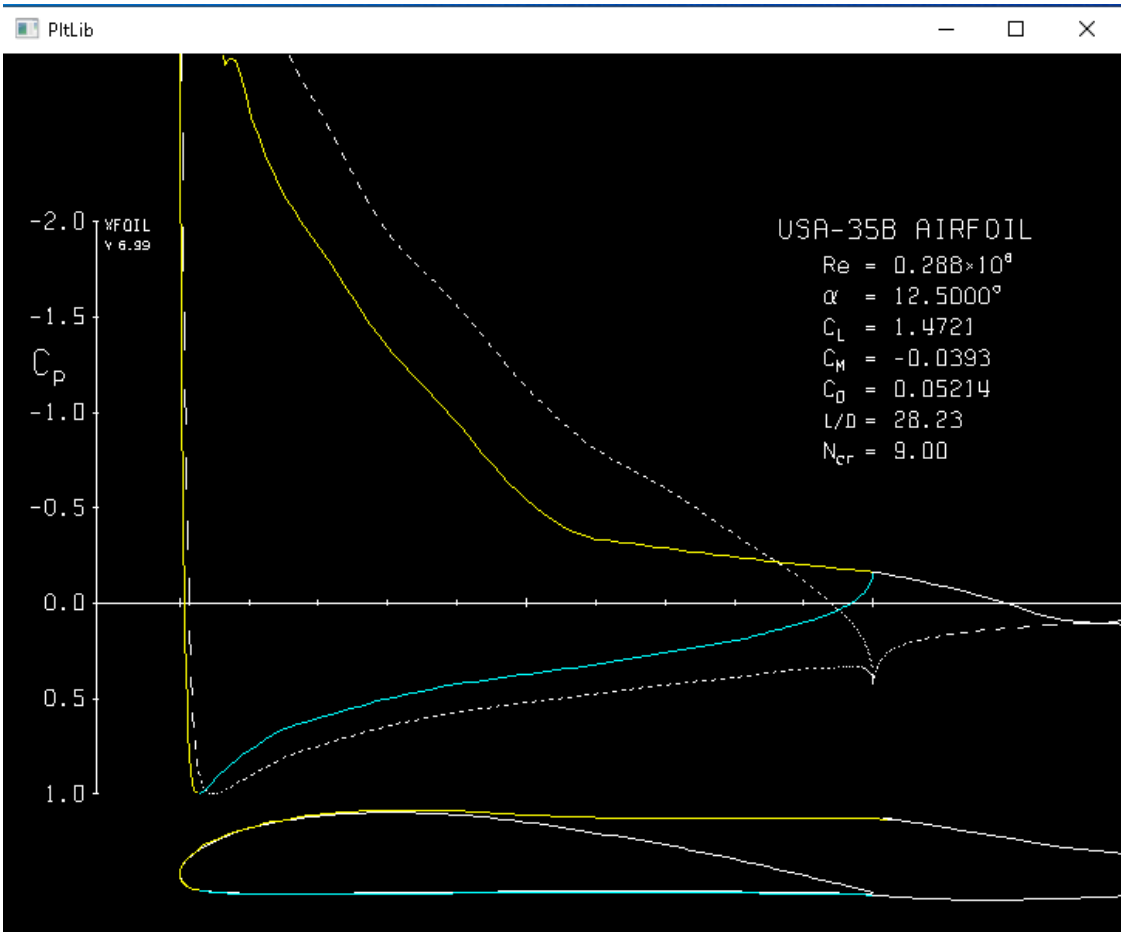

Figure 6 XFOIL visual for 12.5 degree angle of attack, indicating boundary layer separation

With the separation occurring around 0.5 times the length of the chord, the ratio was found to be 117.7. When referring to the research paper [6], the installation angle should be 20 degrees off center from being in line. The final placement parameter for the VGs is the span wise spacing. Two methods were used to find this value, one from the Flitetest article [12] and one from a research paper [5]. The research paper suggested a spacing equal to five times the thickness of the boundary layer. For our case this would have resulted in 0.116 inches between each of the vortex generators. From a native intelligence and feasibility standpoint this value appeared to be too small. The Flitetest article suggested that VGs be spaced no more than two times the radius of the vortex generated by each VG. The radius, r, of the vortex generated was calculated using the following equation.

$$r = \frac{10SC_L}{\pi^2 b}$$

Where S is designated as the planform area of the VG and b is the height of the VG. While Cl is the coefficient of lift of the VG (a flat plate) mounted at an angle, α, or in our case, the installation angle, β.

$$C_l = 2\pi \sin(\beta)$$

This method gave a span wise spacing of 3.658 inches, which appeared much more reasonable. For the sized wing span, the maximum even amount of four VGs were then spaced 3.658 inches apart, from trailing edge to trailing edge, and centered along the span of the wing. The sizing code was written in MATLAB and can be seen in Appendix B.

2.2 Computer Aided Design

After deciding on the design concept and determining the proper sizing of the major components, the system prototype was designed using SolidWorks. First, the main body was designed by extruding the airfoil to the span of 18 inches and extrude-cutting an offset airfoil 17.5 inches (.25" from each edge of the first extrusion) in order to hollow out the inside. The inner airfoil was offset by .180" to form the shell of the wing. Then, the mounting point for the wind tunnel was added using a revolve (for the outer shell) and a revolve cut (for the hollow inside). Next, the wing was divided into a leading edge (LE) section and a trailing edge (TE) section, and a female to female linkage was designed to join the two (shown below). This division of LE and TE allowed access to the hollow inside of the wing, so that the deployment system could be removed, adjusted, and replaced. This was especially helpful for attaching the microcontroller to the computer to upload a new code.

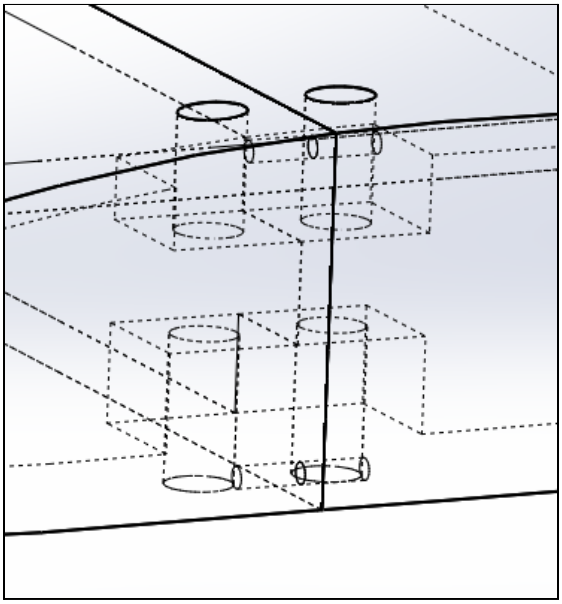

Figure 7 Female Mating Surfaces between leading edge and trailing edge section

Finally, each section was divided into pieces to ensure that each individual piece could fit into the print space of the available 3D printers. To accomplish this, the LE was divided into 2 equally spaced pieces, and the TE was divided into 3 pieces. Because the stress due to moments on a wing is known to be larger as distance from center span decreases, the two divisions of the TE were kept farther from center span. This resulted in the three sections of the TE being 4.5", 9", and 4.5", from left to right. The LE was only split into two sections so that the split lines on the LE and TE did not align,and so that no single cross

section of the wing was entirely compromised by a split line. Two forms of added rigidity were used to mate the split lines: a 0.295” carbon fiber spar in both the LE and TE section, and alignment rods between each split face, including the LE to TE face. These mating methods will be discussed in greater detail in Section 3.0 - Manufacturing. All of the holes for each of these components, as well as the sections of the wing, can be seen in figure 8 below.

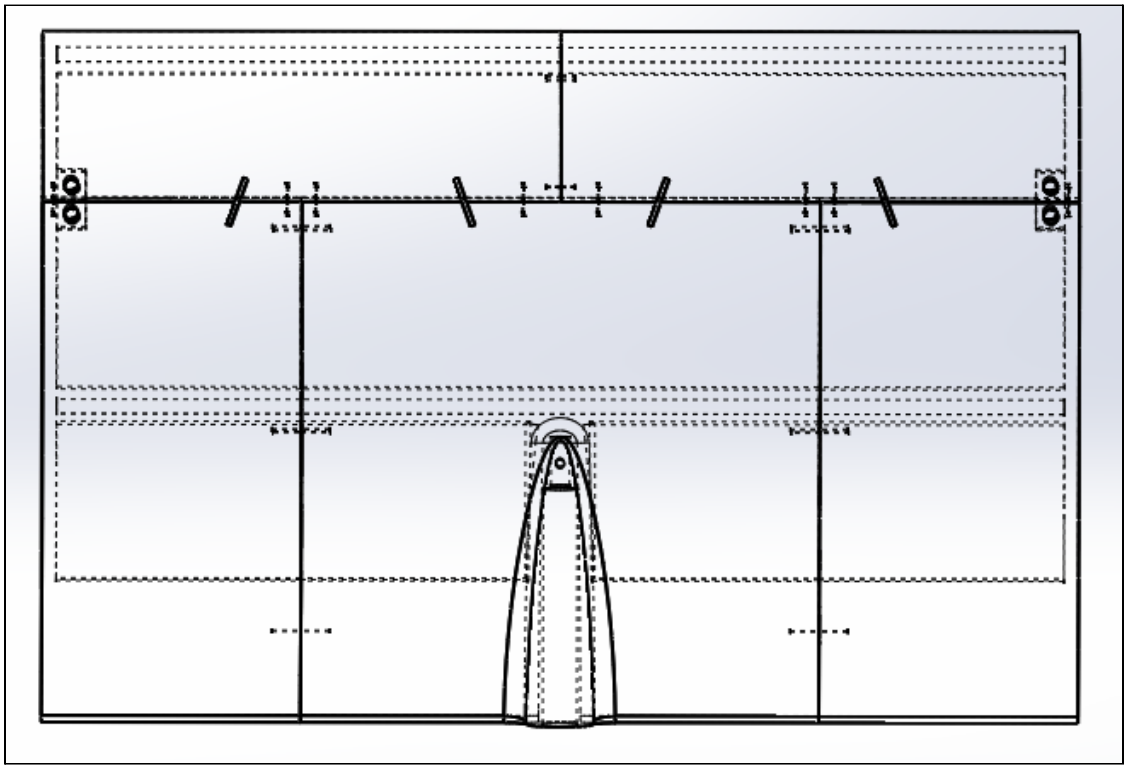

Figure 8 Top view of wire frame wing section. Here, each hole for spars (front and rear horizontal cutouts) and guidance pins (small horizontal and vertical cylinders at each mating surface) can be seen, as well as the 5 major sections of the wing and the wind tunnel mount (bottom center).

Lastly, mounting surfaces for each of the components were added. This included cut-outs for the vortex generators which were modeled in SolidWorks, as well, blocks for the servos, and a mount for the accelerometer.

2.2 Mechanical and Electrical Components

The two servo motors used were EMAX ESO8As and provided 1.5kg-cm of stall torque with a 5 volt input, and the rod that connected the two was a 0.125 inch diameter carbon fiber rod. This lifting system was capable of withstanding the applied forces of the wind tunnel and would maintain a reasonable amount of accuracy of vortex generator deployment under said forces. This mechanism was controlled by a Teensy 3.2 microcontroller, which was mounted on a standard breadboard. The controller received attitude inputs by a mpu-6050 3-axis accelerometer and gyro sensor that was mounted in the direct center of the wing structure to provide accurate angular readings. Additionally, an auxiliary button and two LEDs were wired into the system and mounted on the side of the structure to allow for greater flexibility and simplification of code during the initial data collection phase. The entire

system, including the motors, were powered by a single 5V, 5000mAh battery interfaced directly to the Teensy 3.2, the very same kind used for portable device charging.

Research was then conducted to integrate the circuit into a more compact form. The circuit took various forms upon multiple breadboards in attempts to condense and consolidate it to make room for the other internal components. Development started on a custom printed circuit board design, PCB, and plans to make accommodations and mounting changes to organize the interior as much as possible. A PCB manufacturer was found with the appropriate capabilities and team members began learning how to program the circuit into the company's builder. The idea for a PCB had to be scrapped, however, due to time constraints on both developmental and deliverable fronts. The extra time needed to build and debug the PCB would take members off of other essential duties that would prevent a working model from being tested on time. Further, the time it would take to have the PCB submitted, manufactured and delivered would have exceeded the allotted time for this project.

2.3 Initial Data Collection and Initial Analysis

Initial wind tunnel data needed to be collected in order to optimize vortex generator deployment with respect to angle of attack. The team needed to know the particular effects of the generators for any given deployment state and how it related to the lift and drag characteristics. A program was written that would cycle through ten levels of deployment so that the system could be continuously tested at every angle of attack without need of external adjustment. These ten levels of deployment, spanning from 0% to 100%, would give a fine enough resolution to find the most efficient level of deployment for each degree of angle of attack. The code utilized the two LEDs and button to achieve this, see figure 9 below. Each level of deployment was assigned a light code to provide positive feedback and assurance as to the level being tested. At the end of testing at one angle of attack, the wind tunnel was opened and a single press of the button would cycle to the next level of deployment and its accompanying light code. The light codes were programmed in such a way that an accidental double press, or a button bounce, would be noticeable, see Appendix C for the data collection code. That data was then analyzed, verified, and then regressed to produce an equation for optimal vortex generator deployment for the final system. Special attention was paid to the data collected from the 0% deployed configuration, as this set of data could be compared to published results of the airfoil and be used to confirm the reliability of the wing structure. Further detail on the preliminary data collection and analysis can be seen in section 4.1.1.

The preliminary design called for a small hole to be drilled out of the back of the system so that the vortex generator deployment level could be precisely controlled by a laptop while the wind tunnel was running. This idea would have allowed for continuous data collection and more precise results. An inspection of the final parts revealed far less physical space was available for the wire, and a work around needed to be developed. This led to the development of the button and LED code. To minimize the number of times that the wind tunnel had to be stopped to reconfigure the system, a compromise was made between data fidelity and practicality which were the 10% deployment intervals. It was important that the steps of deployment remained relatively small because of the small size of the vortex generators themselves.

Figure 9 System being tested with button and LED. Light code indicating 40% deployment

2.4 Design Constraints

The four largest constraints to the project were: The time available to complete the project, the limitations of the windtunnel, the manufacturability of the system, and the overall cost to build the system.

2.4.1 Time Constraint

Of the four main constraints affecting this project, the most influential was time. The system had to be designed and manufactured quickly in order to meet deadlines set by both the group and the course syllabus, which meant that producing a durable, manufacturable design had to take a higher priority than extensive data gathering and refinement. Additionally, access to the wind tunnel was limited due to constraints from the University, such as Mechanical Engineering Laboratories requiring use of the tunnel and the availability of the student assistants who are responsible for its use. These limited access windows meant not only that less testing could be performed than desired which limited the total accuracy of the collected data but also meant less results to analyze and average. This limited windtunnel access also hindered the initial design of the system, as the group had to evaluate the available test space dimensions, mounting method, and testing procedures later in the time schedule than expected. These factors of prioritizing a manufacturable design combined with the limited wind tunnel access meant that time was the most important constraint for this project.

2.4.2 Wind Tunnel Constraint

The second highest constraint was the test space and mounting method of the wind tunnel used for testing. The wind tunnel's usable testing space only had a cross section of 20 by 28 inches, and a length of 48 inches. This limited the overall wingspan of the airfoil and alongside this cramped space, the system had to be largely designed around the mounting sting in order to be properly secured and mitigate the risk of the system moving during testing or otherwise dislodging itself and ruining either the data results or the wind tunnel. As mentioned in section 2.2, it was decided that the airfoil shell would attach to the sting via a cylindrical tunnel in the central rearward section, which in turn meant that the full testing length could not be utilized and the wing could only take up about half two thirds of the testing space at maximum. This limited chord reduced the Reynolds number experienced by the airfoil

which also contributed to the inaccuracy of the collected data. On top of the already limited chord length the forces applied to the sting via the airfoil cannot exceed a normal force of 25 pounds, and axial force of 10 pounds, and an applied pitching moment of 50 inch-pounds, meaning that the size of the chord had to be reduced even further in order to not damage the tunnel. This in turn reduced the reynolds number even further and created even more inaccuracy in the collected data. The cylindrical mounting tube built into the airfoil shell also likely had a negative impact on the airflow of the system.

2.4.3 Manufacturability Constraint

The previous two major constraints both impacted the manufacturability of the project, which in itself was already a constraint set by the group. In order to complete the project in a reasonably fast timeline without the use of precision tools or machinery, 3D printing was selected as the production method. By using the FDM 3D printer with already owned PLA filament, the group could begin manufacturing the system as soon as the design was finalized, with the caveat that neither the filament nor the printer were designed with particularly high precision in mind which meant that multiple attempts to print suitable parts had to be made. As a result, in the system design, generous tolerances had to be placed on any dimension that interacted with other parts or components. As described in section 2.2, the airfoil shell had to be manufactured in multiple sections due to the largest possible print dimension being 400mm (15.75”). Care had to be taken to design the connection points in a way that gaps between parts were minimized without taking up excessive amounts of internal space so that the deployment mechanism had room for its full range of motion and all components could still fit inside the shell. Due to the cramped conditions within the shell, the internal components also had to be chosen to minimize their size without falling below minimum performance requirements.

2.4.4 Cost Constraint

Going into the project, the group attempted to make use of manufacturing equipment and hardware resources that the group already possessed in order to minimize costs. The largest expense covered by already possessed equipment was the 3D printer so if the airfoil shell was printed large expenses could be avoided in purchasing or renting equipment. Combined with the idea of rapid manufacturing this cost saving potential effectively constrained the manufacturing technique to 3D printing alone. Outside of this however, the performance requirements of the internal hardware meant that regardless of which specific part was purchased to fulfill each hardware need the cost remained roughly the same. It was of course still the group's goal to minimize costs wherever possible, such as the use of parts donated by the Aero Design Team and the use of the University’s wind tunnel. Because of this relative difference in parts costs that weren’t already possessed or donated were so low, the pricing and cost constraint of the project was debatably the lowest priority of the constraints.

3. Manufacturing

3.1 Design for Additive Manufacturing and Test Prints

It was decided that the parts would be 3D printed because of the complex surfaces of the airfoil and internal structure for the mounting and locating of the electronics. The print orientation was taken into consideration during the design process. Even though the critical surfaces were going to be sanded and waxed, it was desired that the layer lines of the prints be in line with the streamlines of the moving fluid. The parts also fit the best into the 300mm x 300mm x 400mm (11.8in x 11.8in x 15.7in) build volume by being oriented this way. Both the trailing edge section and the leading edge section were fitted with a carbon fiber spar to help laterally strengthen the sub-assemblies. Any complex internal structure was also given 45 degree supports to help reduce the amount of support material needed, especially where it would have been needed in the tight fit areas. An example of this can be seen in figure 10 below for the built in mounting of one of the servo motors.

Figure 10 Example of features for additive manufacturing

Once the full system was modeled, some fine tuning of the size of the parts needed to be tested due to the nature of the parts produced by the desktop 3D printer. The parts that come off the desktop 3D printer generally have holes that are smaller than nominal and protrusions that are larger than nominal. This is because of the nozzle of the printer slightly over extruding on these lower end machines. The simplest way to mitigate this problem was to oversize or undersize the respective features which required doing so. The critical features, seen in figure 11 below, were the mounting section for the sting (top-left), the VG slots in the airfoil surface (top-right), the holes for the spars and alignment pins (bottom-left), and the geometry used for attaching the leading edge section to the trailing edge section of the wing (bottom-right).

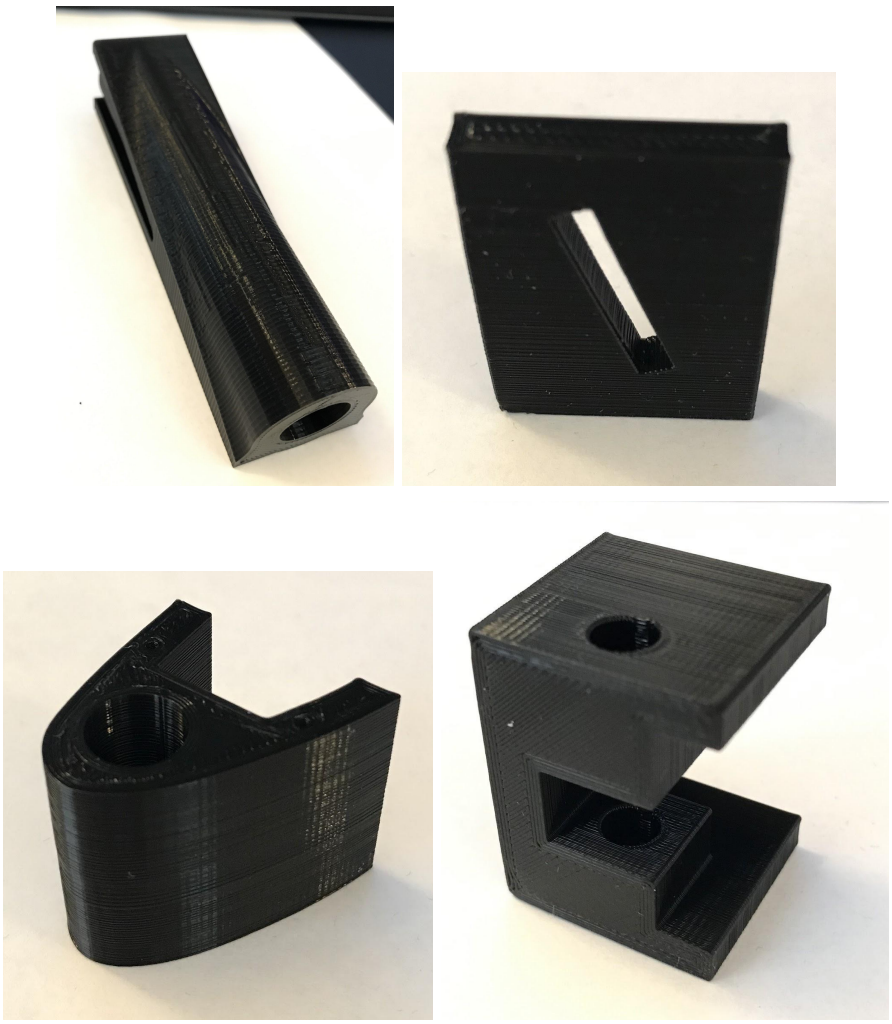

Figure 11 3D printed test pieces to confirm dimensional accuracy

Most of these features were printed undersize; this resulted in offsetting the faces by approximately 0.01 inches, depending on the size of the feature. The most critical of these features was the mounting section to the stingl and the VG slots. The sting mounting section needed to fit tight in order to reduce the vibrations throughout the test model, leading to inaccurate angle of attack readings from the accelerometer. This would have led to very inaccurate results. The set screw also needed to be in a precise location so as to not damage the wiring or sensors within the sting.

3.2 Final Printed Parts

The full system was divided into two sections to allow for a simple method to be used to insert and mount the internal electronics; these sections were the leading edge section, and the trailing edge section. The leading edge section was then split into two halves, the port and starboard sides, so that it could fit within the build volume of the desktop 3D printer. The trailing edge section was not able to be split directly in half because of the geometry used to mount the system to the wind tunnel sting. The trailing edge section was split into three sections so that it could also fit within the print volume of the 3D printer. The team was originally supplied with a 1 kg spool of SUNLU Brand PLA filament from Dr.

Kannan for printing the sections. The first couple of prints with this filament failed due to clogged nozzles at approximately 80%-90% completion. After consulting with Dr. Kannan, and Aaron Trexler on this issue, we found that SUNLU brand had recently changed their manufacturing process for the filament resulting in more particles being present in the roll of filament, which eventually led to nozzle clogs. Some of the failed prints can be seen in figure 12 below. It can be seen towards the top of the print where the nozzle became partially clogged.

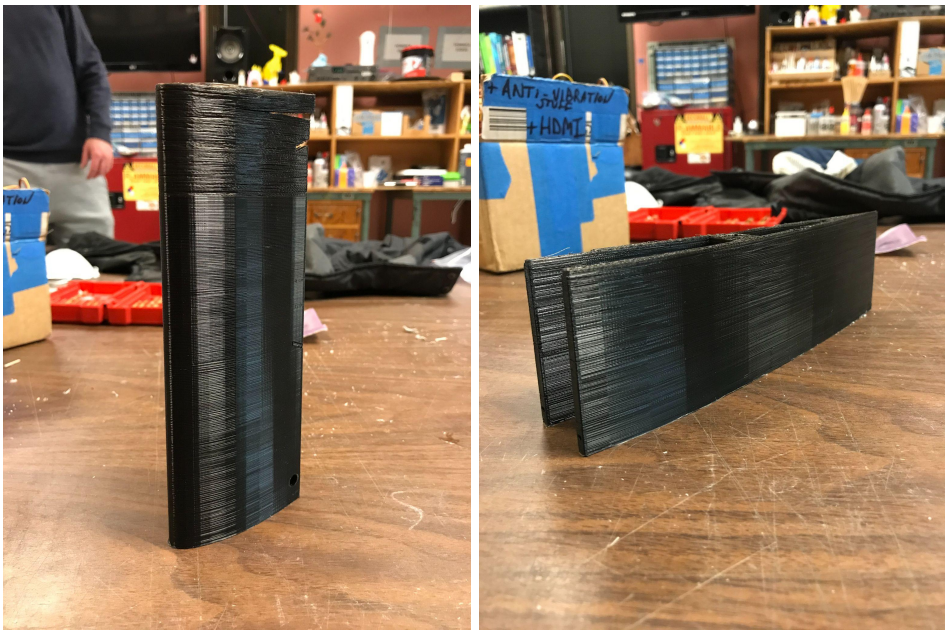

Figure 12 Failed 3D prints using black SUNLU brand PLA material

These issues pushed the timeline back by approximately one week. A 1 kg roll of Hatchbox Brand PLA filament was purchased from Amazon and used for the remainder of the manufacturing process. This new roll of filament posed no issues and provided all successful prints. The Zips Aero Design Team was kind enough to allow us to have 24/7 access to their CR10s Pro V2 3D printer by Creality, pictured below.

Figure 13 Zips Aero Design Team CR10s Pro V2 Desktop 3D Printer

The printer has an enclosure with sliding doors to help keep the chamber warm to mitigate the effects of print warpage. The printer is also hooked up to a Raspberry Pi running Octoprint. This system allows for the printing process to be supervised from any laptop with an internet connection and the credentials to login to the account. The system is also running an add-on called “The Spaghetti Detective” which is an AI system that watches the prints and notifies the user if the print has failed. This is how we were able to stop our first couple prints so as not to waste as much plastic or ruin the 3D printer’s hotend.

3.3 Assembly and Post Processing

The assembly of the wing was completed in several stages. After the parts were 3D printed, they were sanded down to make them smooth, which would reduce surface drag in the wind tunnel and give more accurate results. After each of the parts were sufficiently sanded down, they were prepared to be put together. All of the pieces that were to be glued together first had toothpicks, or alignment pins, protruding out of the sides of them, so as to aid in aligning the sections prior to glue up. These alignment pins also helped to strengthen the parts from shear forces, since epoxy is much stronger in the normal direction than what it is in shear, and would then fail in shear. In addition to toothpicks being used as alignment pins, two 0.295 inch arrow shafts ran through most of the length of the wing. One was located in the leading edge and the other was located inside of the trailing edge. An unsanded leading edge piece as well as the rear pieces with toothpicks protruding out of them can be seen in figures 14 and 15, respectively, below:

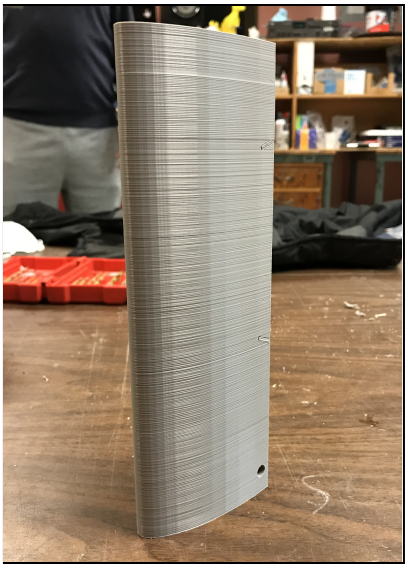

Figure 14 Unsanded leading edge section

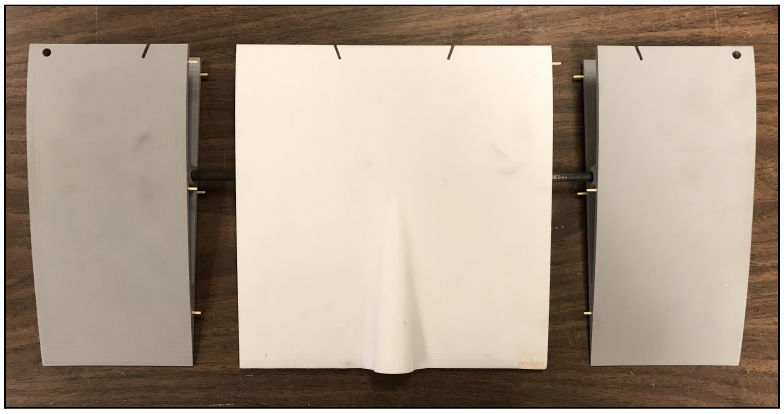

Figure 15 Rear sections with toothpicks protruding

After the parts were sanded and aligned, they were glued together using five minute epoxy glue. In order to glue them, the mixed epoxy was applied to the surfaces that were to be glued together. The pieces were clamped together to make sure that they stayed in place while the epoxy cured. It takes the epoxy five minutes to harden, but a full hour to fully cure. The clamping process included the parts in between two pieces of styrofoam in order to give the parts soft surfaces holding them together. This eliminates the issue of possibly damaging the parts from being clamped to a hard surface for a long period of time. The clamps were connected directly to two pieces of wood that bordered the styrofoam pieces. The purpose of the wood was to take the concentrated force of the clamps and spread it out over a larger area to make sure the pieces glued together evenly. The process of gluing the rear three pieces together is shown below in figure 16:

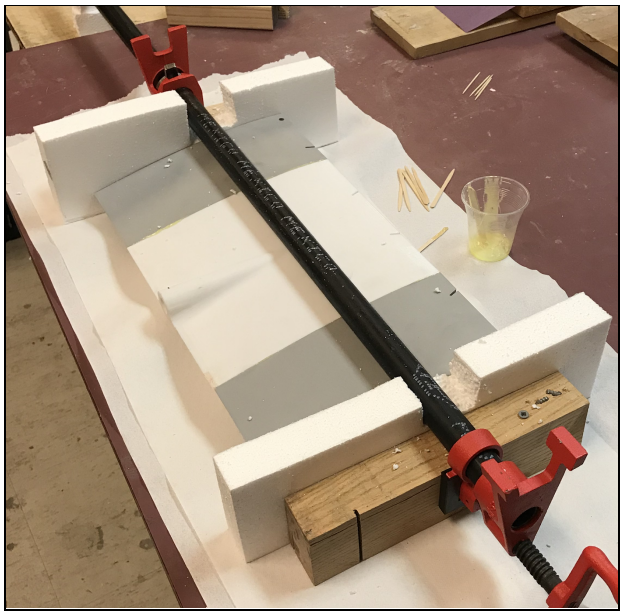

Figure 16 The rear sections being glued together with the clamp, wood, and styrofoam

After the epoxy was finished curing, the areas where the parts were glued together had to be sanded down, since the epoxy dried around the sides. This would create more friction/drag, so it had to be sanded more. In order to get the smoothest finish around the sections, they were sanded down with sandpaper with increasing grit levels to reduce friction as much as possible. Initially they were sanded using 180 grit sandpaper, then 220 grit, 400 grit, and then finally 800 grit. Paste wax was then applied to the edges to be buffed out afterwards. Two ¼ inch wooden dowel pins would be placed inside of the female linkages to keep them in place.

In order to put the wing together, the servos, the carbon fiber rod, the vortex generators, and the rest of the avionics were placed inside of the rear section of the wing with the vortex generators placed in the slots. The leading edge section would be attached to the rear section using two 3D printed female to female linkages at either side of the wing. When the wing was about to be assembled, it was discovered that it was impossible to assemble the system. Due to the geometry of the vortex generators alternating their directions to the left and right, simply attaching the leading edge to them was impossible. This was solved by using a small hand saw and cutting a straight line from the ends of the slots in the leading edge to the back of the leading edge. This allowed the vortex generators to be placed in the rear slots and still be able to attach the leading edge to the trailing edge sections. In order to eliminate the extra drag from the holes in the wing next to the vortex generators, tape was placed along them to eliminate as much extra drag as possible. A picture of the modified slots, as well as the vortex generators sitting in their slots, is shown below in figure 17.

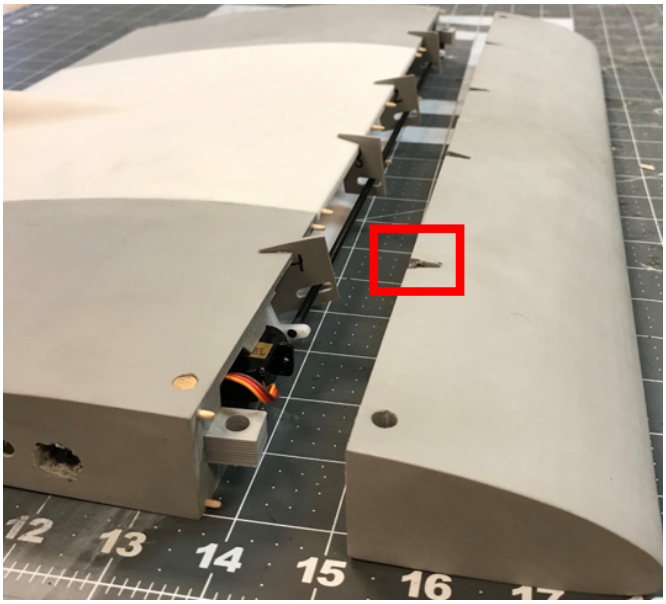

Figure 17 The cut in the leading edge that allowed it to fit around the vortex generators

In order for the wing to be able to be tested in the wind tunnel, it needed to be able to fit onto the sting inside of the wind tunnel. The sting would be inserted into the hole in the rear of the wing. In order to do this, another smaller hole was drilled into the center of the underside of the wing where the sting would be in. This hole would accommodate a 4-40 set screw that could be tightened into the wing and would hold the sting against the screw and the top of the hole. During testing, however, as the 4-40 screw was being tightened against the sting, the threaded hole became stripped. Another hole was drilled in the same place, but it was a larger hole that could accommodate a 10-32 bolt. Since this bolt had a larger diameter, more material was being used to hold the weight of the wing in place, so it did not become stripped like the smaller diameter screw did.

4. Design Verification

Design verification was completed in three phases. First, data was gathered from wind tunnel testing of various angles of attack at various levels of vortex generator deployment. That data was then compiled into an equation to optimize the most efficient deployment level at a given angle of attack. Finally, that equation was programmed into the system and tested to produce the final results.

4.1 Testing

Below in section 3.1.1 is the methodology used to attain preliminary data which was then used to test the final prototype of the system by the methods explained in section 4.1.2.

4.1.1 Preliminary Data Gathering

The first step in testing the system was to run a wind tunnel analysis on the wing from angles of attack (AOA) of -12° to 20° with the deployment system off. When the system is off or outside of the operational AOA, the vortex generators will revert to the minimum deployment level which allows them to effectively mimic the natural curve of the airfoil. This in turn means that when the system was off, the assembly mimicked the base airfoil and gave data to test against. The team then performed the same analysis with the vortex generators at different levels of deployment (in steps of 10%) and entered all of these performance results into an excel spreadsheet. After outliers were removed, the lift to drag ratio was calculated for each set of data and plotted versus AOA. At each AOA, the deployment level with the highest lift to drag ratio was selected. Then, a curve fit was produced of percent deployment versus AOA, and the equation of this fit was used to control the final system. Based on the results of this preliminary testing, the system only had a major effect between AOAs of -6 and 6 degrees, so the system was set to lower the vortex generators outside these bounds. Within these bounds, the generators followed the curve fit equation

$$\%\,d = 0.0559\alpha^5 - 0.0628\alpha^4 - 2.5234\alpha^3 + 1.3102\alpha^2 + 21.299\alpha + 46.203$$

4.1.2 Final System Testing

Once the optimal deployment level at each AOA was determined, an equation was developed that changed the deployment level of the vortex generators dynamically as the accelerometer detected the airfoil's current AOA, as well as setting the deployment level to 0% when turned off or outside of the operational AOA. A source code was found and modified to suit this application, see code in Appendix D [9]. The airfoil was run through the wind tunnel analysis for all AOA with the new system turned off three times, then run three more times with the system on. All six sets of data were sent to Excel where an outlier analysis was performed by removing extreme data points, and both sets of data (system off and on) were averaged. A curve fit was then applied to both sets of averaged performance data and they were plotted against each other. The results of this comparison with and without outliers included can be seen below in section 4.2.

4.2 Data Analysis

Analysis of the final set of data gathered by the feed-forward system consisted of comparing results from the fully implemented system with the system with 0% vortex generator deployment. As stated, the system's initial testing at 0% vortex generator deployment was compared with respect to the

published results of the standard USA 35B airfoil [1]. Results from this comparison were favorable and allowed for analysis of the system itself to proceed with confidence, See figure 18 below.

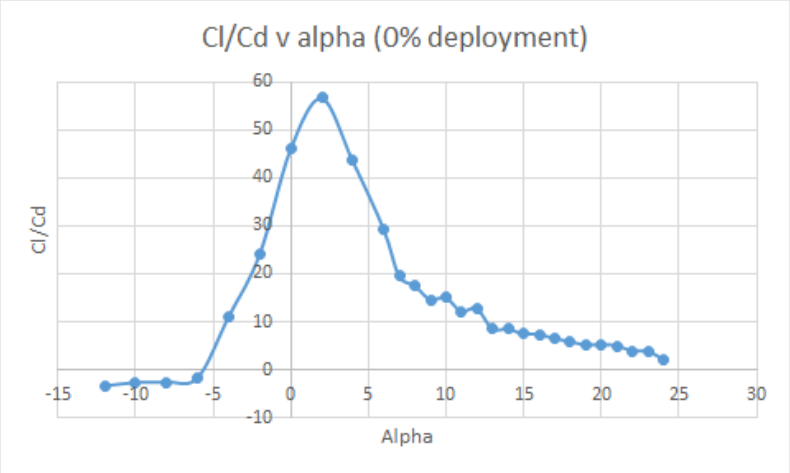

Figure 18 Graph Displaying Cl/Cd vs. Alpha for 0% Deployment

4.2.1 Verification

A Requirements and Verification Table can be found as Table 2 in Appendix A detailing how each requirement was verified.

4.3 Standards Utilized

The standard most utilized during this project was ISO/ASTM 52910-18 for additive manufacturing methods. The standard lays out requirements, guidelines, and recommendations for additive manufacturing and specifics as to its appropriate use. Section 6 of this document was applied during the initial design and CAD steps of design, in particular section 6.9.2 which discusses design intent, practicality and purpose of additive manufacturing over other manufacturing methods [4]. Special consideration was also given to section 7, which deals with the layer lines left behind as a result of the process. Debate and study was involved as to how large of an impact that they would have on the results. The layer lines ran parallel to the streamline path which indicated that they should have a negligible effect, however, the wing was sanded smooth as a precaution.

Questions were raised early on in the design process about the structural integrity of the system under the loads of the wind tunnel and every effort was made to mitigate the risk of a catastrophic break up during testing. If the system were to fail during testing, the resulting debris of the system would most certainly damage the fan blades of the wind tunnel and render it inoperable. To ensure that every effort was made to prevent this, all hardware used in the system was required to have a DFARS 252.225-7009 compliance [3]. This includes the most critical component, the set screw, which is the only thing besides friction holding the system to the sting of the wind tunnel.

4.4 Results

Results from testing were inconclusive. The system did have a definite positive effect on the performance of the airfoil at some angles of attack, but is not consistent with the expected results with

static vortex generators, so it is difficult to draw a conclusion. Based on the results shown in figure 19, the system had anywhere from a 60% improvement to a 20% detrimental effect on the lift to drag ratio. It is suspected that the deviations and anomalies present in the results are due to three main causes: errors brought about from the wind tunnel itself, mechanical tolerance issues discussed in section 4.4.3, and variations in atmospheric conditions during each of the different days of testing. Extreme outliers were removed from the data to confirm that the error was in the data gathering methods and not the analysis itself. Doing so provided a graph with incongruous results which confirmed that physical factors produced the results, see figure 20.

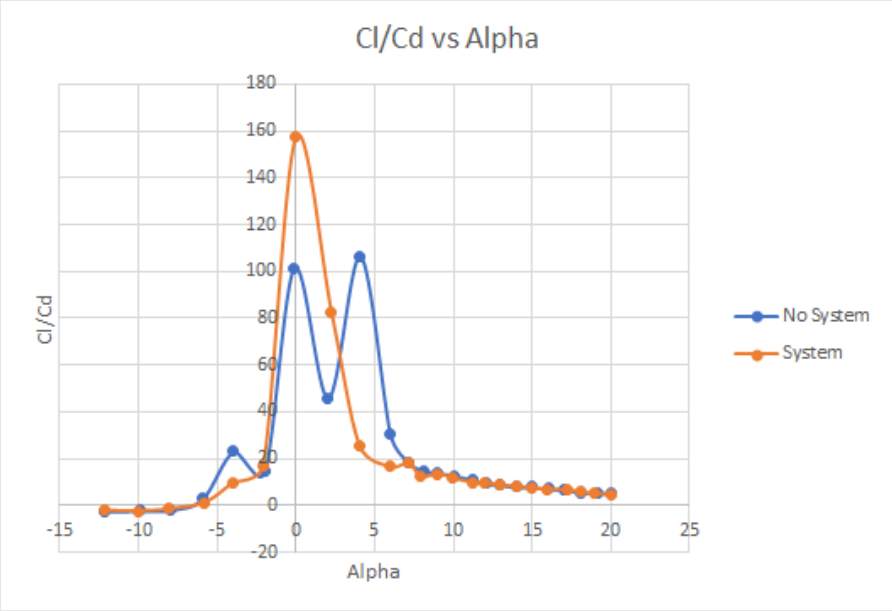

Figure 19 Graph comparing Cl/Cd vs Alpha between the system implement and not implemented

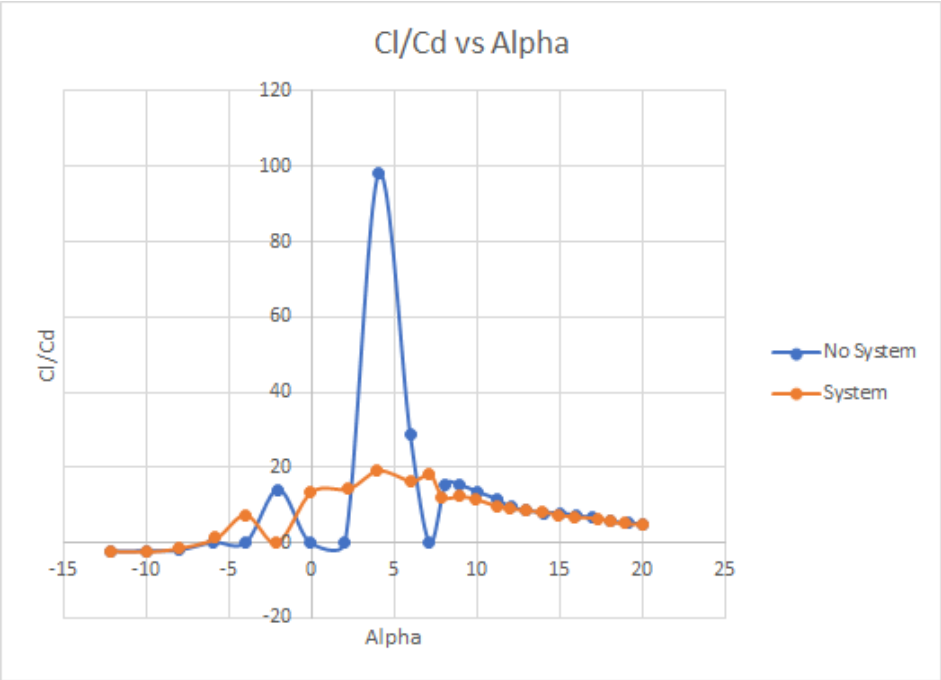

Figure 20 Graph comparing Cl/Cd vs Alpha between the system implement and not implemented using outlier analysis

5. Costs

According to Glassdoor, as of the 10th of April, 2021, the average Aerospace Engineer makes $93,392 per year. If one were to break that salary down to 50 weeks of working (excluding two weeks for vacation) and 40 hours per week, that averages out to $46.70 per hour of working. If all five members work a combined average of 10 hours per week, and the duration of the project is 34 weeks, then the total cost in labor totals to about $15,878 over the course of the project. In addition to this cost, there are many other costs associated with this project, which can be seen in Table 1. After considering these costs, the total cost of the project in labor and materials, is $16,658. Since this was a student project, there was no cost in labor. Many of the electronics were already owned by the students, such as the Teensy 3.2, breadboard, and wiring that was used. The Zips Aero Design Team generously donated many materials to the project as well, such as the 3D Printer, the servos, the wooden dowel rods, and the carbon fiber rod. After considering all these costs, the total actual cost to the team was $27.98.

5.1 Parts

Table 1: Parts Costs

Part	Manufacturer	Retail Cost ($)	Actual Cost ($)
CR-10s Pro V2 3D Printer	Creality	700.00	0.00
Gray PLA	Hatchbox	25.99	25.99
Teensy 3.2	PJRC	19.80	0.00
Breadboard	Microcenter	3.99	0.00
Wiring	Amazon	6.99	0.00
Servos	EMAX	8.20	0.00
Accelerometer	Amazon	1.99	1.99
Wooden Dowel Rods	Home Depot	5.29	0.00
Carbon Fiber Rod	Dragon Plate	8.11	0.00
Total	N/A	780.36	27.98

5.2 Labor

The labor was split into many different sections headed by different members of the project group. All members participated in planning, designing, and scheduling the project. The preliminary design was primarily done by Kirklin Anderson. The CAD Modeling was completed by everyone in the group, but primarily by Chris Chapanar. The 3D printing and manufacturing was completed by Solomon Whitmire, with the help of Dr. Manigandan Kannan and Aaron Trexler in the university's 3D printing lab. The programming of the Teensy was primarily done by Dan Chech. The manufacturing and assembly was

completed by Solomon Whitmire, Kirklin Anderson, and Nick Amon. All members of the group helped with testing the system in the wind tunnel. Data analysis was completed primarily by Nick Amon. The collective time dedicated to this project weekly was about 10 hours per week, on average. The total time from start to finish of the project was 34 weeks.

6. Conclusion

6.1 Accomplishments

The system was initially designed to be able to deploy the VGs to an optimum height in order to energize the boundary layer, keeping it attached to the surface of the wing, only when required. Upon testing the final system it was found that the VGs had more of an effect on the characteristics of the wing section at much lower angles of attack than what was expected. With this data a feed-forward control system was designed to deploy the VGs to the most efficient height at specified angles of attack. With a verification test prior to the wind tunnel testing, the team was able to prove that the goal of designing a feed-forward control system for deploying VGs dependent on angle of attack was successful. The second part of the project was to prove or disprove that the system positively affected the characteristics of wing section by increasing the stall angle or increasing the Cl to Cd ratio. The results from this experiment were inconclusive. This was determined by the fact that a baseline test with no VGs deployed was run prior to any testing during the data collection and final testing phases, and the results were inconsistent.

6.2 Uncertainties

The results from testing could have been inconsistent for a number of reasons. The uncertainties that could have played a significant role in the final results from testing could have been caused by the inaccuracy from the wind tunnel measurements, surface trueness and roughness, VG and VG slot fitting and tolerances, tolerances of the servo deployment mechanism, vibrations and values read by the accelerometer, and/or the wired connections.

The most notable uncertainty was the inaccuracy of the wind tunnel. Upon consulting with one of our former professors, Garrett McHugh, he notified us that the wind tunnel is not the most accurate as the sting has been overloaded before. If the sting had been overloaded before, this could have permanently damaged the force readout sensors for the wind tunnel. The surface trueness of the wing section could have been scanned using a laser scanner had we initially thought about this and had access to this sort of technology. If the surface of the airfoil manufactured did not match that of what was originally used to create the CAD model, this could have also varied our final results. Once the system was assembled the critical surfaces were sanded and waxed as described in section 3.0 Manufacturing above. It was hypothesised that this would provide a surface roughness that would be acceptable for the purposes of this project. It could have been possible to use a profilometer to find the exact average roughness value; this could then have been compared to the standards for wind tunnel testing. A feature that could have led to a large amount of uncertainty was the deployment of the VGs. During testing the loose fitting of the VGs in the VG slots allowed them to vibrate which could have led to incorrect results. The servos could have also been jittering while under load which would have induced more vibrations into the VGs themselves. The angular readings from the accelerometer could have varied, as well, if the wing section vibrated enough on the sting. This was mitigated by mounting the accelerometer as close to the sting tip as possible. Had the accelerometer been mounted at the wing tips it would have been displaced even more causing larger changes in the angular readings. The final uncertainty that was considered was the wiring. This could have posed a problem because of the use of a breadboard. If the

wires were moving around the connections could have become temporarily loose during operation. This could have been mitigated by using a custom PCB with soldered connections. This option was under consideration for the majority of the project, but seeing as no one on the team had previous experience with this it would have taken a significant amount of time to do this, and so it was decided against.

6.3 Ethical considerations

When working on any form of aerospace system, safety is always the number one ethical concern. Because this system is meant to reduce likelihood of wing stall, and wing stall can cause an airplane to crash, the system has the potential to improve the safety of an airplane. However, as with any added electronic system, a failure of the system has the potential to be catastrophic. If, for example, this system was implemented on an airplane, and the pilot were to believe that the airplane was safe for a specific range of angles of attack, but the system was to fail, it could cause wing stall, with the potential of endangering lives. Because of this, if any version of this system were to be implemented on an airplane, it would require extensive testing to establish safe ranges of use, as well as redundancies in sensors to ensure the system is operational and to notify the pilot if it fails.

Another ethical consideration for this system is the carbon emission of airplanes. Reduction of emissions has become an important ethical concern in modern society, and because this system has the potential to reduce drag on an aircraft, it has the potential to reduce the amount of fuel expended. This reduction would come from a reduced need for thrust output to overcome the drag of the aircraft.

6.4 Future work

Although the results of this study are inconclusive, they do show potential that could justify further investigation. There are a few key aspects to the development of this system that could be improved in order to yield more reproducible results. First, a higher Reynolds number should be used for the next prototype, since it is commonly known that higher Reynold's numbers tend to create smoother performance curves for an airfoil. This added "smoothness," or predictability, could potentially yield a much more accurate curve fit function for the controller. A prototype with a higher Reynold's number would require a bigger wing section, which would therefore require a larger wind tunnel. Additionally, the Reynold's number can be increased by performing the tests at a higher wind speed. The wind speed used for testing was limited by the wind tunnel as well, due to the limit on the amount of force that the wind tunnel can safely handle on the test section, which is a maximum normal force, axial force, and moment of 25lbs, 10lbs, and 50inch-lbs, respectively.

Another method that could be used to improve upon this study in future work would be to gather more data in the development phase of testing. Rather than testing each angle of attack with each level of deployment one time, the same test could be done multiple times and on multiple different days to account for changes in atmospheric pressure. Then, the data could be averaged before determining the optimal percent deployment for each angle of attack. This could potentially create a more optimal curve fit for the percent deployment versus angle of attack.

References

[1] Airfoiltools.com. USA-35B AIRFOIL (usa35b-il). Retrieved from http://airfoiltools.com/airfoil/details?airfoil=usa35b-il. Accessed April 2021.

[2] A. Seshagiri, E. Cooper, and L. W. Traub, "Effects of Vortex Generators on an Airfoil at Low Reynolds Numbers," *Journal of Aircraft,* vol. 46, no. 1, pp.116-122, January 2009.

[3] FAR (2021, February). *252.225-7009 Restriction on Acquisition of Certain Articles Containing Specialty Metals.* Acquisition.gov. https://www.acquisition.gov/dfars/252.225-7009-restriction-acquisition-certain-articles-containg-specialty-metals

[4] International Standard Organization. (2018). *Additive manufacturing — Design — Requirements,* guidelines and recommendations (Standard No. 52910). Retrieved from https://compass.astm.org/EDIT/html_annot.cgi?ISOASTM52910+18

[5] Li, X., Liu, W., Zhang, T., Wang, P., & Wang, X. (2019). Analysis of the Effect of Vortex Generator Spacing on Boundary Layer Flow Separation Control. Applied Sciences, 9(24), 5495. https://doi.org/10.3390/app9245495

[6] Li, X.-K., Liu, W., Ting-Jun Zhang, & Xiao-dong Wang. (2019, December 2). Experimental and Numerical Analysis of the Effect of Vortex Generator Installation Angle on Flow…ResearchGate; MDPI. https://www.researchgate.net/publication/337708648_Experimental_and_Numerical_Analysis_of_the_Effect_of_Vortex_Generator_Installation_Angle_on_Flow_Separation_Control

[7] Li, X., Yang, K., & Xiaodong Wang. (2019, March 12). Experimental and Numerical Analysis of the Effect of Vortex Generator Height on Vortex Characteristics and... ResearchGate; MDPI. https://www.researchgate.net/publication/331712096_Experimental_and_Numerical_Analysis_of_the_Effect_of_Vortex_Generator_Height_on_Vortex_Characteristics_and_Airfoil_Aerodynamic_Performance

[8] P. M. Gerhart, A. L. Gerhart, and J. I Hochstein, *Munson, Young and Okiishi's Fundamentals of Fluid Mechanics,* 8th ed. John & Wiley Sons Inc, 2016, pp. 489-514.

[9] Rowberg, J (2012) I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (Version 10.0) [Source code]. https://github.com/jrowberg/i2cdevlib

[10] T. R. Yechout, S. L. Morris, D. E. Bossert, W. F. Hallgren, *Introduction to Aircraft Flight Mechanics,* American Institute of Aeronautics and Astronautics, 2003

[11] Udris, A. (2015, September 1). Vortex generators: Preventing stalls at high and low speeds. Available at: https://www.boldmethod.com/learn-to-fly/aerodynamics/vortex-generators/. Accessed April 2021.

[12] Vortex Generator Design Tips and Experimentation. (2015). Flite Test; FliteTest. https://www.flitetest.com/articles/vortex-generator-design-tips-and-experimentation

Appendix A Requirement and Verification Table

Table 2 - System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
1. System can withstand wind tunnel forces a. Airfoil shell remains fully together/intact b. Deployment mechanism can deploy in direct wind	1. Visual Inspection **a. Visual Inspection** Shell was inspected after test runs to ensure no cracks had formed and minor bend test was performed to ensure strength **b. Visual Inspection** Vortex generators were deployed and stowed during wind tunnel test multiple times to ensure consistent deployment	Y Y Y
2. System is entirely feed-forward control and needs no outside connection to function (during tests) a. Accelerometer is able to accurately measure angle of attack b. System is able to deploy vortex generators as desired based on angle of attack c. System lowers vortex generators outside of operational angles of attack	2. Experimentation **a. Experimentation/Data Analysis** While mounted on wind tunnel sting system was connected to laptop with its arduino code to check that the accelerometer's angle output matched what was displayed by the wind tunnel **b. Experimentation/Visual Inspection** While previous verification was taking place, deployment value output and vortex generators were examined to ensure proper deployment values at each AOA **c. Experimentation** Once previous verifications were complete, system was angled out of operational range to ensure that vortex generators returned to zero deployment	Y Y Y Y
3. Systems battery cell is able to sufficiently power hardware throughout experiments	**1. Experimentation** System was left on without charging through multiple rounds of preliminary and verification testing and lasted roughly an hour and a half or longer (depending on battery cell) before cell change was required	Y

Appendix B Vortex Generator Sizing MATLAB Code

Contents


```
%Senior Design 2020-2021
%11/17/2020
clear
clc
close all
format shortEng
```

Initial Values

```
c=12; %chord (in)
b=18; %span (in)
v=30; %wind velocity (mph)
```

Airfoil Data

```
%USA-35B
crit_aoa=12.75; %stall angle at 200k-500k (deg)
```

Wind Tunnel Limits

```
N_max=25; %(lbf)
A_max=10; %(lbf)
Mp_max=50/12; %(ft-lbf)
```

Calcs

```
rho=0.00230812; %sl/ft^3
mu=3.76378e-7; %lbf s/ft^2
c=c/12; %chord (ft)
b=b/12; %span (ft)
S=c*b %planform area (ft^2)
v=v*(5280/3600); %velocity (ft/s)
Re=(rho*v*c)/mu
Cl_max=1.5270; %Cl max b/w 200k and 500k (at 500k 12.75deg (stall point))
Cd_max=0.09155; %Cd max b/w 200k and 500k (at 200k 15.75deg)
Cm_max=0.0980; %Cm max b/w 200k and 500k (at 200k -5.75deg)
L=0.5*rho*(v^2)*S*Cl_max;
D=0.5*rho*(v^2)*S*Cd_max;
Mp=0.5*rho*(v^2)*S*c*Cm_max;
aoa=-crit_aoa:0.25:crit_aoa+15;
N=(L*cosd(aoa))+(D*sind(aoa));
A=-(L*sind(aoa))+(D*cosd(aoa));
FS=2;
```

```
N_max=max(N)*FS %(lbf)
A_max=max(abs(A))*FS %(lbf)
Mp_max=Mp*12*FS; %(in-lbf)
A_dist=1.146; % (in)
xac_from_TE=12-(0.25*(c*12));
L_of_sting=5.1;
C_dist=xac_from_TE-L_of_sting;
Mp_bal_point=(Mp_max-((A_dist+C_dist)*N_max))
```

```
S =

     1.5000e+000


Re =

   269.8279e+003


N_max =

    10.2535e+000


A_max =

     4.2226e+000


Mp_bal_point =

   -43.8568e+000
```

VG Sizing

```
c_VG=0.07*c*12 % chord (length) of ea. VG (in)
l_VG=0.2016*c*12 % distance from LE_wing to LE_VG (in)
    % at transition point of BL at 8 deg using Ncrit=9
delta=((5*l_VG)/sqrt(Re)) %laminar BL thickness right before transition (in)
fprintf('Choose t, the maximum deployment of VGs in inches, noting \n')
fprintf('that max may not be reached in this experience. \n')
t=0.25 %choose 1/4in max deployment, note may not need full
              %deployment (in)
S_VG=t*c_VG; %area of VG (in^2)
x_sep=0.5*c*12;
xfromTEVGtoSep=x_sep-(l_VG+c_VG);
xoverH=xfromTEVGtoSep/delta;
if xoverH<=20 %incidence angle of VG to LE of wing (degrees)
    beta=30
elseif xoverH>20 && xoverH<=80
    beta=25
elseif xoverH>80
    beta=20
end
Cl_VG=2*pi*sind(beta); %Cl of flat plate at beta
spacing_research_paper=5*delta %spacing of vortex generators (in)
fprintf('Spacing recommended from research paper seems pretty \n')
```

```
fprintf('small, probably should use spacing that is recommended \n')
fprintf('by flite test forum. \n')
r=((10*S_VG*Cl_VG)/((pi^2)*t)); %radius of generated vortex (in)
spacing_flite_test=2*r %spacing of vortex generators (in)
```

```
c_VG =

   840.0000e-003


l_VG =

     2.4192e+000


delta =

    23.2862e-003

Choose t, the maximum deployment of VGs in inches, noting
that max may not be reached in this experience.

t =

   250.0000e-003


beta =

    20.0000e+000


spacing_research_paper =

   116.4309e-003

Spacing recommended from research paper seems pretty
small, probably should use spacing that is recommended
by flite test forum.

spacing_flite_test =

     3.6580e+000
```

Published with MATLAB® R2020b

Appendix C Data Collection Teensy Program


```
#include <Servo.h>

 // pushbutton pin
 const int buttonPin = 3;

 // servo pin
 const int servoPin = 9;
 const int servoPin2 = 10;  ////Added
 Servo servo, servo2; ////Added servo2

int led = 13; //activating pin 13 for LED
int led2 = 15;

//create a variable to store a counter and set it to 0
int counter = 0;
void setup()
{
  servo.attach (servoPin);
  servo2.attach (servoPin2);                    ////Added
  // Set up the pushbutton pins0 to be an input:
  pinMode(buttonPin, INPUT);
  pinMode(led , OUTPUT); //setting LED as an output
  pinMode(led2 , OUTPUT); //setting LED as an output
}

void loop()
{
 // local variable to hold the pushbutton states
  int buttonState;
  //read the digital state of buttonPin with digitalRead() function and store the value in buttonState variable
  buttonState = digitalRead(buttonPin);
  //if the button is pressed increment counter and wait a tiny bit to give us some time to release the button
  if (buttonState == LOW) // light the LED
  {
    counter++;
    delay(150);

  }
  if(counter == 0)
  {
    servo.write (61.26);  // zero degrees
    servo2.write (118.74); ////Added
    digitalWrite(13,HIGH);              // 13 is the Red LED
    digitalWrite(15,HIGH);              // 14 is the Green LED
    digitalWrite(13,LOW);               // Code: Red/Blue
    digitalWrite(15,LOW);
  }
  else if(counter == 1)
 {
    servo.write (67.38);  // zero degrees
    servo2.write (112.62); ////Added
    digitalWrite(13,HIGH);              // 13 is the Red LED
    digitalWrite(13,LOW);               // Code: Red
  }
  else if(counter == 2)
    {
    servo.write (73.23);  // zero degrees
    servo2.write (106.77); ////Added
    digitalWrite(15,HIGH);              // 14 is the Green LED
    digitalWrite(15,LOW);               // Code: Blue
  }
  else if(counter == 3)
    {
    servo.write (78.91);  // zero degrees
    servo2.write (101.09); ////Added      // Code: {No light}
  }
  else if(counter == 4)
    {
```

```
      servo.write (87.48);  // zero degrees
      servo2.write (95.52); ////Added
      digitalWrite(13,HIGH);              // 13 is the Red LED
      digitalWrite(15,HIGH);              // 14 is the Green LED
      digitalWrite(13,LOW);               // Code: Red/Blue
      digitalWrite(15,LOW);
    }
    else if(counter == 5)
      {
      servo.write (90);  // zero degrees
      servo2.write (90); ////Added
      digitalWrite(13,HIGH);              // 13 is the Red LED
      digitalWrite(13,LOW);                // Code: Red
    }
    else if(counter == 6)
      {
      servo.write (95.52);  // zero degrees
      servo2.write (84.48); ////Added
      digitalWrite(15,HIGH);              // 14 is the Green LED
      digitalWrite(15,LOW);              // Code:Blue
    }
    else if(counter == 7)
      {
      servo.write (101.09);  // zero degrees // Code: {No light}
      servo2.write (78.91); ////Added
    }
    else if(counter == 8)
      {
      servo.write (106.77);  // zero degrees
      servo2.write (73.23); ////Added
      digitalWrite(13,HIGH);              // 13 is the Red LED
      digitalWrite(15,HIGH);              // 14 is the Green LED
      digitalWrite(13,LOW);                // Code: Red/Blue
      digitalWrite(15,LOW);
    }
    else if(counter == 9)
      {
      servo.write (112.62);  // zero degrees
      servo2.write (67.38); ////Added
      digitalWrite(13,HIGH);              // 13 is the Red LED
      digitalWrite(13,LOW);               // Code: Red
    }
    else if(counter == 10)
      {
      servo.write (118.74);  // zero degrees
      servo2.write (61.26); ////Added
      digitalWrite(15,HIGH);              // 14 is the Green LED
      digitalWrite(15,LOW);               // Code: Blue
    }
    //else reset the counter to 0 which resets thr servo to 0 degrees
    else
     counter = 0;
  }
```

Appendix D Final System Teensy Program


```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//      2012-06-20 - improved FIFO overflow handling and simplified read process
//      2012-06-19 - completely rearranged DMP initialization code and simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//      2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//      2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                 - add 3D math helper file to DMP6 example sketch
//                 - add Euler output and Yaw/Pitch/Roll output formats
//      2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//      2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//      2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "math.h"
// coeff. from Chris' 5th order polynomial curve fit
float p5 = 0.0559;
float p4 = -0.0628;
float p3 = -2.5234;
float p2 = 1.3102;
float p1 = 21.299;
float p0 = 46.203;

float pd = 0;
float aoa;
float arm_throw;
float servo_angle;

int led = 3;


#include "MPU6050_6Axis_MotionApps20.h"
#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

#include "Servo.h"

Servo myservo_one, myservo_two;  // create servo object to control a servo// ADDED SERVO OBJECT

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* =========================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 * ========================================================================= */

/* =========================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
```

```
    release. For more info, see these links:

    http://arduino.cc/forum/index.php/topic,109987.0.html
    http://code.google.com/p/arduino/issues/detail?id=958
 * ================================================================ */



// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
//#define OUTPUT_TEAPOT



#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };



// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}



// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
    myservo_one.attach(9); //Added this right herer~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    myservo_two.attach(10);
    //myservo_one.write(0);
    //myservo_two.write(0);

  pinMode(led, OUTPUT);


    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
```

```
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);


    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
    // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}



// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // other program behavior stuff here
        // .
        // .
        // .
        // if you are really paranoid you can frequently test in between other
        // stuff to see if mpuInterrupt is true, and if so, "break;" from the
        // while() loop to immediately process the MPU data
        // .
        // .
        // .
    }

    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();
```

```
    // check for overflow (this should never happen unless our code is too inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt (this should happen frequently)
    } else if (mpuIntStatus & 0x02) {
        // wait for correct available data length, should be a VERY short wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;

        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("\t");
            Serial.print(q.x);
            Serial.print("\t");
            Serial.print(q.y);
            Serial.print("\t");
            Serial.println(q.z);
        #endif

        #ifdef OUTPUT_READABLE_EULER
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetEuler(euler, &q);
            Serial.print("euler\t");
            Serial.print(euler[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(euler[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(euler[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[2] * 180/M_PI);
            //Output to servo
           //myservo_one.write(180-(90+ ypr[2] * 180/M_PI)); // added thiss ~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~SEI
           // myservo_two.write(180-(90+ ypr[2] * 180/M_PI)); // ypr is [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector. M_PI is Pi (3.14...)
          /////~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Control Loop End~~~~~~~~~~~~~~~~~~~~
                 aoa = (-1*(ypr[2] * (180/M_PI)))+86;
             if((aoa >= -6) && (aoa <= 6)) {
                 pd = (p5*pow(aoa,5)) + (p4*pow(aoa,4)) + (p3*pow(aoa,3)) + (p2*pow(aoa,2)) + (p1*aoa) + p0;
                 arm_throw = ((pd/100)*0.25)-0.125;
                 servo_angle = (180/M_PI)*asin(arm_throw/0.26);
                myservo_one.write (servo_angle+90);
                myservo_two.write ((90-servo_angle));
                digitalWrite(led, HIGH);
                }
              else
                {
                myservo_one.write (90-28.74);
                myservo_two.write (90+28.74);
                digitalWrite(led, HIGH);
                delay(100);
                digitalWrite(led, LOW);
                delay(100);
                }

                 Serial.print("\t");
                 Serial.print(aoa);
                 Serial.print("\t");
                 Serial.print(pd);
                 Serial.print("\t");
                 Serial.print(arm_throw);
                 Serial.print("\t");
                 Serial.println(servo_angle);

            /////~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Control Loop End~~~~~~~~~~~~~~~~~~

        #endif

        #ifdef OUTPUT_READABLE_REALACCEL
            // display real acceleration, adjusted to remove gravity
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            Serial.print("areal\t");
            Serial.print(aaReal.x);
```

```
            Serial.print("\t");
            Serial.print(aaReal.y);
            Serial.print("\t");
            Serial.println(aaReal.z);
        #endif

        #ifdef OUTPUT_READABLE_WORLDACCEL
            // display initial world-frame acceleration, adjusted to remove gravity
            // and rotated based on known orientation from quaternion
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```