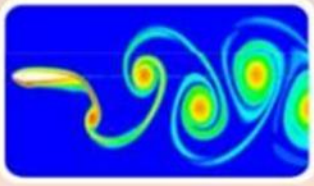
VT-Robotic Bird



Mechanical Design

- Brandon Horton (Sub-team Leader)
- Joseph Amaya
- Zach Collie
- Michael Kossa

Computational Analysis

- Mohomed Elseikh (Sub-team Leader)
- Paul Asbury (Finance Manager)
- Rachel Nichols
- Greg Gadell

Controls

- Brandon Galbraith (Team Facilitator)
- Josh Marino (Sub-team Leader)
- Chris Nesaw

Facilitator email: Bgalbrai@vt.edu
Facilitator phone: (717)-875-4710

Project 12 Advisors: Dr. Bayandor, Dr. Kurdila, Dr. Battaglia, Dr. Mueller

May 9, 2014

Executive Summary

Over the years flapping flight has been researched by several scientists, but never fully understood. Most aerial vehicles to date make use of fixed wing flight, while effective this form of flight offers less maneuverability. With our project this year, we intend to offer a better understanding of flapping wing flight by improving and redesigning a prototype of a robotic seagull. The seagull will be a bio-inspired design and is intended to match the flight patterns and dimensions as closely as possible. No previous iterations of this project have been successful in attaining flight. Our task this year will be to evaluate the existing prototype of the previous robotic bird design, apply advanced control techniques in order to devise a feedback controlled closed-loop kinematic model, and analyze the flight kinematics of the bird using computational fluid dynamics.

Since so little is known about creating flight by flapping wings with high biological accuracy, our design project will provide a unique perspective into this area of study. To accomplish this task it was necessary to divide our design team into three main subgroups: Mechanical Design, Computational Analysis, and Controls. The main semester objective for the mechanical design team consisted of improving the design of the previous year's iteration and to generate ideas for future improvements to the model. The controls team was responsible for the development of a system to accurately depict the flapping motion of the bird. In order to validate the design decisions the computational analysis team began analyzing unsteady flight patterns over a two dimensional airfoil to be used for the entire wing structure.

Table of Contents

Background

Flapping flight has been investigated for hundreds of years, dating back to the 14th century with Leonardo da Vinci's famous ornithopter. Inspired by the birds soaring around us, many attempts have been made throughout history to create flight with flapping wings. Only few attempts have come close to creating an aerial vehicle which not only mimics bird flight, but does so with a high degree of biological accuracy and correct articulation.

Currently, the specialized maneuverability and stability of flying creatures is far more refined than the fixed wing vehicles of modern technology and is poorly understood by researchers today. Furthermore, recent bird flight studies have shown that optimized flapping wings could provide greater efficiency by requiring much less power than its fixed-wing counterpart [1]. By truly understanding bird flight, scientists and engineers can design flying vehicles with more knowledge that can lead to higher efficiency and meet specialized tasks.

With our project this year, we intended to offer a better understanding of optimized flapping flight by improving and redesigning last year's prototype of a robotic seagull. Previous iterations of this project have yet to be successful in creating a fully functional robotic bird. Since so little is known about creating flight by flapping wings with high biological accuracy, our design project will provide a unique perspective into this area. The information obtained from our design project will be able to be used for future scientists and engineers to design highly maneuverable and efficient flying mechanisms.

Design Evaluation Criteria

Customer Needs. It was a main focus of the design team to provide a prototype that was as biologically accurate as possible. The most important customer needs in order to bring the project to fruition are provided in Table 1. This includes having a wingspan that matches that of a seagull, as well as similar body characteristic dimensions. Not only should the physical model accurately represent a seagull, but the flight patterns and kinematics should be equivalent. In order to be a true bio-inspired design, the prototype will follow the wing motion of a seagull as precisely as possible. A key component in flight was being able to generate sufficient lift to overcome the weight of the bird. By reducing the overall bird weight the model can reduce flight speed, such that a lightweight model was a high priority. Similarly it was desired to generate enough lift to maintain steady flight and be self-propelled for the entire motion with only flapping motion. During flight, the joints will be under extreme pressure and load forces, especially at the top and bottom of the flap motion, and will need to remain intact and sturdy.

Table 1
Needs vs. Metrics Matrix of Key Project Components

	Need	Metric	Unit mfg. cost	Fully extended wingspan	Total robot weight	Total lift generated	Self-propelled flight time	Maximum airspeed	Wing flap rate	Wing joint cycles to failure
3	Has a wingspan similar to that of a seagull			●						
4	Flap rate is similar to that of a seagull								●	
5	Kinematics are similar to that of a seagull			●	●	●		●	●	
11	Joints can withstand normal flight loads									●
16	Is lightweight				●		●			
17	Design generates sufficient lift					●	●	●		
18	Performs at sufficient flight speed						●	●		

Target Specifications. Our target specifications for the overall bird model were essential design factors when developing our physical prototype. The lift generated should be at least 7.5 N and the thrust should be slightly positive to maintain flapping flight. For flapping flight conditions, our ideal flight velocity was between 7 m/s and 10 m/s for a flap rate of 2 Hz. As for the physical prototype, the ideal weight of the model should remain under 0.7 kg, reducing the necessary lift generated, while maintaining a factory of safety of at least 5 for all parts. Another main challenge was for feedback control such that actuation of the joints occurred at the proper time during the flap cycle to ensure the proper flapping flight kinematics.

Biology. For our project we decided to base our design off the Glaucous winged gull. We chose this bird in particular because of its large wingspan, and that it is capable of producing lift throughout its entire flap cycle. To fully understand our design it is necessary to understand the biology of the wing of a seagull. As seen from Figure 1, the secondary section of the wing extends from the elbow to the wrist joint. This section of the wing is primarily responsible for generating lift throughout the entire flap cycle. The primary section of the wing, which can be seen extending from the wrist to the end of the wing, is responsible for lift and thrust. During a flap cycle, the primary wing section undergoes a backsweep motion on the upstroke. While in this backsweep motion, the drag experienced on the wing is lessened due to a reduction of surface area.

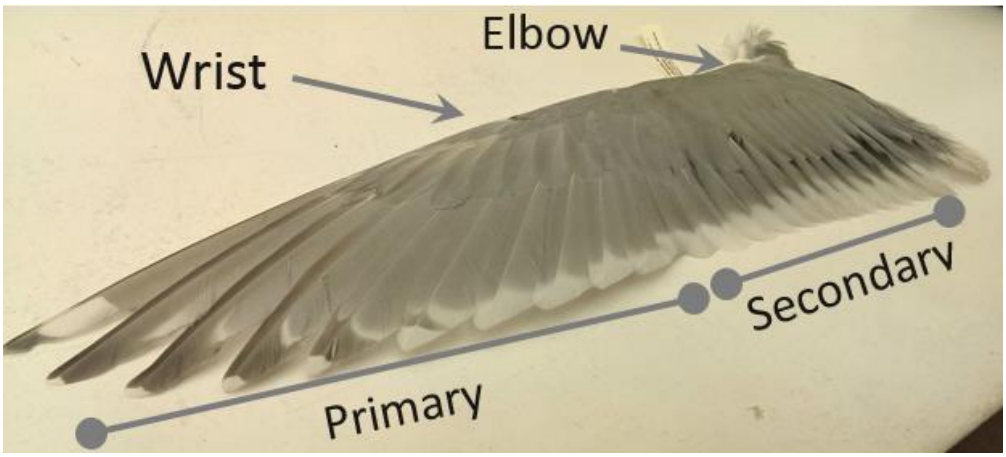


Figure 1. This figure contains the elbow and wrist joint of a seagull.

Robotic Model. To better understand what is happening during a flap cycle we created a robotic model of our wing, shown in Figure 2, using the Denavit-Hartenburg convention. What makes our robot model unique is that it incorporates four degrees of freedom. Similar work by other groups, such as Festo, only incorporate a three degree of freedom design.

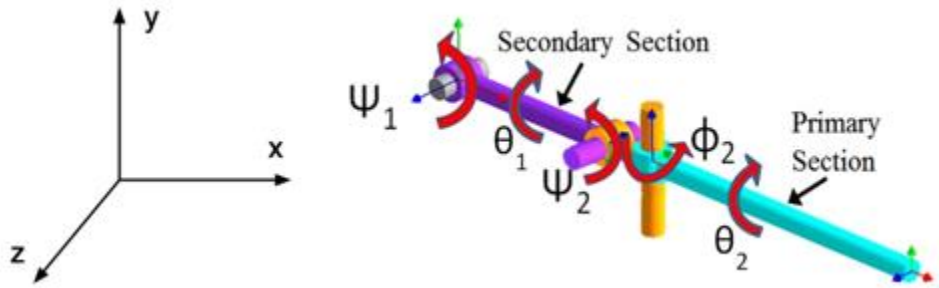


Figure 2. In this figure, Ψ_1 and Ψ_2 represent the vertical motion about the YZ plane, θ_1 and θ_2 represent the torsional motion in the XZ plane, and Ψ_2 represents the backsweep motion.

For our secondary section, the design is slightly different than how it is in nature because we only have one DOF. In nature, a bird has two degrees of freedom about its secondary wing section which, but based on preliminary CFD results we found it was easier to simplify the secondary section into one degree of freedom. The primary wing section has three degrees of freedom, one that controls the vertical motion, another which controls the torsional motion, and the last that contains the backsweep motion. The backsweep motion is one of the features of our project that make it unique, and is responsible for the extra degree of freedom when compared to other designs.

Computational Fluid Dynamics Analysis

The following sections will explore the setup of the models, meshing, and equations used to actuate a 2D case study of the secondary and primary wing sections, as well as a full 3D wing.

Secondary Wing Section. The CFD simulations in this study were run using the commercial package ANSYS Fluent v14.5 which uses two-dimensional Navier-Stokes equations to solve transient simulations. The S1223 airfoil was used for all dynamic analysis because it matches the cross-section of a seagull wing most accurately and is capable of generating lift during the entire kinematic motion of the bird's wing. Data points for a two-dimensional S1223 airfoil were imported from Airfoil Tools [2] into ICEM CFD v14.5 and a two-dimensional geometry of this airfoil was created. A rectangular boundary was defined around the airfoil and the left face was defined as a velocity inlet, the top and bottom faces were defined as symmetry, the right face was defined as a constant pressure outlet, and the airfoil was defined as a rigid body with a no-slip wall. Using an appropriate number of nodes per area, a rectangular mesh was generated around the airfoil geometry with a minimum cell volume of 2.0e-5 m^3 and a maximum cell volume of 5.0e-4 m^3. The number of nodes was determined based on a grid resolution study, which analyzed the smallest number of nodes necessary to maintain the integrity of the results. The mesh was then changed from a rectangular mesh to a triangular mesh to allow for dynamic mesh manipulation and re-meshing while running each case.

The mesh and boundary conditions were further manipulated so that each case could be analyzed. To update the dynamic mesh, the Laplacian Smoothing Method was used. This method is built into the Fluent code and works using Equations 1-3 [3]. The location of each mesh vertex is adjusted based upon its bordering vertices. This method can only be used in a triangular mesh.

$$\overrightarrow{x_i^m} = \frac{\sum_j^{n_i} \vec{x}_j^m}{n_i} \tag{1}$$

$$\vec{x}_i^{m+1} = \vec{x}_i^m(1-\beta) + \vec{x}_i^m\beta \tag{2}$$

$$SF_I = \left(\frac{\sum \frac{1}{D_J}\Delta s_J}{\sum \frac{1}{D_J}}\right) \tag{3}$$

Local re-meshing equations based upon sizing functions were then used to define the resolution and size of the background grid created by Fluent. These equations are shown in Equations 4-10 [3].

$$size_b = \left(\frac{\sum SF_I \frac{1}{L_I}}{\sum \frac{1}{L_I}}\right) \tag{4}$$

$$d_b = \frac{d_{min}^P}{d_{max}} \tag{5}$$

$$size_P = size_b \times \left(1 + \alpha \times d_b^{1+2\beta}\right) = size_b \times \gamma \tag{6}$$

$$size_{P,max} = size_b \times (1+\alpha) = size_b \times \gamma_{max} \tag{7}$$

$$\gamma = 1 + \alpha d_b^{\frac{1}{1-\beta}} \qquad if\ \alpha < 0 \tag{8}$$

$$\gamma = 1 + \alpha d_b^{1+2\beta} \qquad if\ \alpha > 0 \tag{9}$$

$$size \notin \left[\frac{4}{5}\gamma size_b, \frac{5}{4}\gamma size_b\right] \tag{10}$$

All equations shown are built into the Fluent code and were employed based upon research on proper dynamic meshing parameters in 2-D unsteady flight simulation. The inlet velocity and optimal angle of incidence (θ_1) were varied to determine the optimal velocity and angle of incidence for maximum lift. A user-defined function (UDF) was input to Fluent and acted as the velocity component on the airfoil. The angles Φ_1, Ψ_1 and θ_1, in relation to the airfoil, are shown in Figure 3. Φ_1 is the forward and backward motion of the airfoil, Ψ_1 is the vertical movement of the airfoil and θ_1 is the angular movement of the airfoil (varying angle of incidence). For the cases studied, $\Psi_1 = 30°$, θ_1 was varied from 0°-45°, and $\Phi_1 = 0°$. A value of 30° for Ψ_1 was used because this was the overall angle the secondary section of the wing a seagull went through during one stroke of the wing. The inlet velocity of the air over the airfoil was then varied to simulate bird flight at different speeds through stagnant air. Equations 11-13 show the equations used to govern the motion of the secondary wing section within the UDF. Each inlet velocity was tested against each θ_1 value, and the value for the average coefficient of lift (C_L) and coefficient of drag (C_D) on the airfoil were determined for two seconds of flight.

$$V_x = \phi_{\ 1} fr_1 cos(ft) \tag{11}$$

$$V_y = fr_1 \Psi_1 cos\ (ft) \tag{12}$$

$$V_\theta = \theta_1 f cos(ft) \tag{13}$$

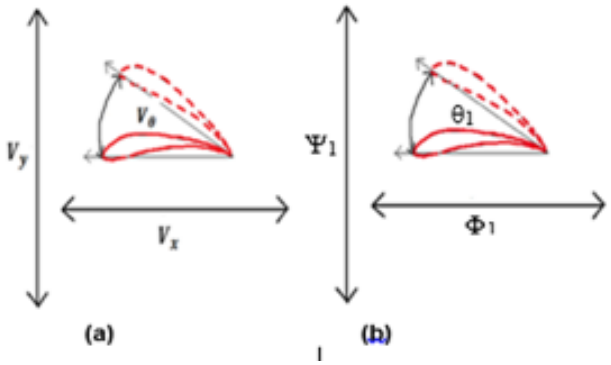


Figure 3. Shown is a representation of the angles Ψ_1, Φ_1 and θ_1. Each angle in (b) corresponds to the magnitude of the velocities shown in (a).

For each case, the inlet velocity was varied before the case was run. The magnitude of the velocity was normal to the left boundary and the velocity was uniform over the inlet. The pressure at the outlet was set to 0 Pa (gauge pressure), and the top and bottom boundary conditions were not changed from their setting of Symmetry within ICEM. The interior fluid was air at a density of 1.225 kg/m^3 and viscosity of 1.7894E-5 kg/m-s. The material of the airfoil was aluminum and the airfoil was defined as a rigid body.

By defining the airfoil as a rigid body, the user-defined function could be used to manipulate the airfoil using the specified values of Φ_1, Ψ_1 and θ_1. To run a case, a laminar, transient analysis was used with a Reynolds number varying from 6,000-140,000, depending on the inlet velocity setting. The analysis was run for six full flaps of the bird's wing at a rate of 3 Hz, for a total flight time of two seconds. When running a case, after every two iterations, the data for C_L and C_D were saved and the mesh was updated. After each case was run, plots of the pressure contours and vorticity contours were generated, and the total lift and drag values were calculated based on the C_L and C_D for each time step of flight.

Primary Wing Section. A second study was done to determine the ideal amplitudes of the three degrees of freedom actuated by the primary portion of the wing, namely the vertical, backsweep and torsional motions. The same commercial package Fluent v14.5 was used to solve the transient, two-dimensional study completed on the primary portion of the wing. The user-defined function was altered to accurately represent how the primary portion of the wing moved in relation to the secondary portion of the wing. The same case and data files used for the secondary wing study were used for the primary wing study. A different user-defined function was applied to the airfoil using equations derived from the secondary wing motion and are shown as Equations 14-16.

$$V_x = \phi_1 f r_1 cos(ft) + \phi_2 f r_2 cos(ft - \tfrac{2\pi}{9}) \quad (14)$$

$$V_y = f r_1 \Psi_1 cos\,(ft) + f r_2 \Psi_2 cos\,(ft - \tfrac{2\pi}{9}) \quad (15)$$

$$V_\theta = \theta_2 f cos(ft) \quad (16)$$

Cases were run to determine the optimal θ_2 and Φ_2 angles shown in Figure 4. The angles shown above are in relation to the magnitude of the velocities within the user-defined function applied to the airfoil. All cases were run by varying the inlet velocity and the angles Φ_2 and θ_2. A table of these values is provided in Appendix A.

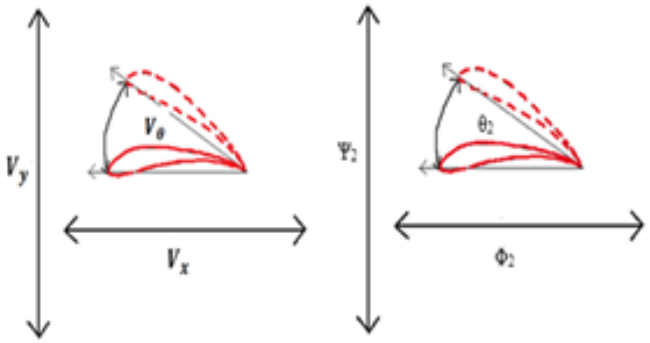


Figure 4. Shown are the magnitudes of the velocities (Ψ_2, Φ_2, θ_2) for the motion of the primary portion of the wing.

3-Dimensional Wing. A three dimensional geometry of the bird's wing was created using actual dimensions from the prototype bird. This geometry accurately represented the dimensions of the wing on the prototype of the bird as well as the specific secondary and primary sections. The overall geometry was divided into a secondary section and primary section, both defined as rigid bodies. Both secondary and primary sections were created so that specific user-defined functions could be applied and move the wing with the appropriate amplitudes. A base section was attached to the 3-D geometry and acted as a stationary axis around which the wing rotated. Figure 5 shows the geometry of the wing used to analyze the wing in three dimensions.

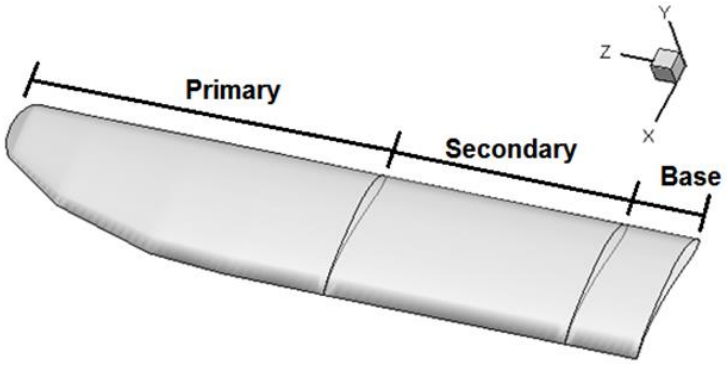


Figure 5. Overall geometry of the wing used for 3-D analysis.

A triangular mesh was defined around the wing with approximately 1,000,000 nodes and dynamic mesh settings were applied so that re-meshing was possible during the analysis of the wing in Fluent. To define the motion of the wing, rotational and translational velocity equations were applied to both the secondary and primary sections of the wing independently. The secondary section utilized one rotational velocity equation about the x-axis to define its motion and is shown in Equation 17:

$$\omega_x = \left(\frac{\Psi_1}{2}\right)\cos(ft)\, f, \tag{17}$$

where Ψ_1 is the amplitude of the motion and f is the frequency of the flap in Hz. For the primary portion of the wing, translational as well as rotational equations were applied. The translational equations move the primary portion of the wing within the x-z plane with respect to the secondary

motion described by Equation 17. These equations are shown in Equations 18 and 19 as translational velocities:

$$V_y = -r_1 \cos(\alpha)\dot{\alpha} \tag{18}$$

$$V_z = r_1 \sin(\alpha)\dot{\alpha} \tag{19}$$

where r_1 is the length of the secondary section of the wing, α is the position of the connection axis between the secondary and primary sections, $\dot{\alpha}$ is the angular velocity of the secondary section of the wing and *f* is the frequency of the flap in Hz. The equations for α and $\dot{\alpha}$ are shown in Equations 20 and 21.

$$\alpha = -\left(\frac{\Psi_1}{2}\right)\sin(ft) + \left(\frac{\Psi_1}{2}\right) \tag{20}$$

$$\dot{\alpha} = \left(\frac{\Psi_1}{2}\right)\cos(ft)\, f \tag{21}$$

The equations that define the rotational motion of the wing are shown in Equations 22-24 as angular velocities:

$$\omega_x = \left(\frac{\Psi_2}{2}\right)\cos(ft)\, f \tag{22}$$

$$\omega_y = -\left(\frac{\phi_2}{2}\right)\cos\left(ft + \left(\frac{\pi}{2}\right)\right) f \tag{23}$$

$$\omega_z = \left(\frac{\theta_2}{2}\right)\cos(ft)\, f \tag{24}$$

where Ψ_2 is the amplitude of the vertical flap, Φ_2 is the amplitude of the back-sweeping motion of the primary wing section in the horizontal plane, and θ_2 is the amplitude of the angle of incidence, which varies dynamically throughout the flap cycle.

Results of CFD Analysis

Once the setup was complete, a case study was performed on the primary and secondary wing sections independently to determine the peak performance of the wing. Preliminary results of a 3-D wing analysis are also provided as well as the limitations to the current design setup.

Secondary Wing Section. Three evaluations were completed to study the secondary section of the wing. The first study was used to validate the accuracy of the results calculated by Fluent for a wing at low Reynolds numbers. The next study was used to determine if one degree of freedom was sufficient to provide enough lift for sustained flight, and the last study determined the necessary velocity to maintain this lift.

Validation Study. In order to ensure that the ANSYS Fluent solver produced accurate results, a second airfoil with known coefficients of lift and drag values was evaluated. This airfoil was a NACA 0012 Airfoil, which was evaluated for a coefficient of lift as a function of the angle of incidence in a static flow case. Results of the validation case study were compared to known results taken from a study performed by NASA [4]. The same turbulent system and Reynolds number were modeled using ANSYS Fluent 14.5 and evaluated at an angle of incidence of 10°, 0°, and -10°. The results for the value of C_L calculated in each case are provided in Table 2.

Table 2
Results of the value of Cl calculated from NASA and the validation study [3]

Angle (deg.)	C_L - NASA	C_L - Fluent	Percent Error (%)
10	1.09	1.0093	7.4
0	0	-0.000143	0.01
-10	-1.08	-1.014	6.1

A percent error of 7.4% was calculated between the results obtained from the test case and the case run by NASA at an angle of incidence of 10°. A percent error of 0.01% and 6.1% were calculated for an angle of incidence of 0° and -10°, respectively. The differences in the NASA simulation and the ANSYS Fluent method are small enough to be considered negligible, suggesting that the ANSYS Fluent approach is a viable method to simulate the fluid effects of air over the wing.

One Degree of Freedom Study. The key objective of this study was to determine if one degree of freedom on the secondary portion of the wing was sufficient to supply enough lift for the bird, or if adding a second degree of freedom was necessary. This case study evaluated the performance of the secondary wing section only. In seagull flight, the secondary portion of the wing generates a majority of the lift necessary for flight while the primary wing section generates the required thrust. Currently, the designed secondary wing portion experiences one degree of freedom in the vertical direction, and can be adapted to include a varying angle of incidence, θ_1. The ratio of the coefficient of lift to the coefficient of drag (C_L/C_D) was examined for a range of inlet velocities, based on the average coefficients of lift and drag values calculated for each case. The largest ratio of C_L/C_D indicates the bird will generate a maximum amount of lift in relationship to drag. The velocity at this maximum value is necessary to determine the speed at which the bird must fly to maintain the maximum amount of lift per drag over the secondary portion of the wing.

Figure 6 displays the vortices generated by the airfoil during the up sweep motion of the secondary wing section undergoing an incoming flow velocity of 7 m/s. The current position of the airfoil is the top of the flap cycle. There is a zero angle of incidence for this example. There are no significant vortices produced by the airfoil in the current position, nor at any other point of the flap cycle. There are two different steam traces of air coming from the top and bottom of the airfoil independently. These stream traces merge and generate a low drag environment through which the airfoil travels. There are no leading edges vortices produced to hinder the path of the airfoil, and no trailing edge vortices produced other than the parallel stream traces which would cause any adverse effects to the lift generation.

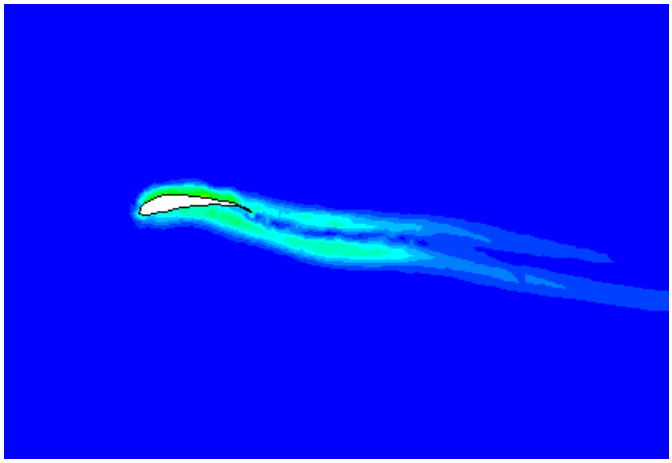


Figure 6. Airfoil vorticity of a steady flap rate of 2 Hz and an angle of incidence of zero degrees.

Shown in Figure 7 is a graph of inlet velocity versus C_L/C_D for each varying angle of incidence (θ_1). At an angle of incidence of 0°, the maximum ratio of the coefficient of lift to the coefficient of drag is attained. For an incident velocity of approximately 9 m/s, the coefficient of lift to drag ratio was greatest and represents the velocity at which the bird must fly to attain the maximum amount of lift and minimum drag over the secondary portion of the wing. Inlet velocities ranging from 0.5 m/s to 10 m/s cover the spectrum of possible flight speeds that our current bird prototype can sustain. The range of flight velocities was measured in order to determine the minimum flight velocity necessary to maintain steady flight. The lower the velocity the more likely the prototype could attain and sustain this velocity.

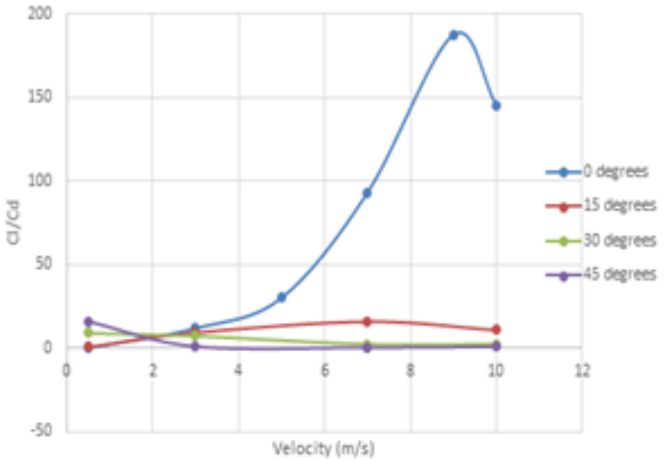


Figure 7. Ratio of the coefficient of lift to the coefficient of drag vs. velocity for a varying angle of incidence over both wings.

The airfoil motion is modeled as a sinusoidal function and provides a varying amount of lift as a function of time. Figure 8 is a plot of the lift generated by the airfoil motion, in Newtons, versus each time step analyzed during the two second flapping motion. Since the airfoil naturally generates lift at a neutral position, lift was generated regardless of the current airfoil position. The peak lift generation occurs where the airfoil returns to the starting position of the flap cycle, during the down sweep of the wing. The minimum lift occurs when the airfoil reaches its starting point

during the upsweep motion of the wing. The results display a consistent peak level of lift through each flap for the entire time of flight. A greater number of time steps will produce a more refined pattern, and remove any unwanted noise.

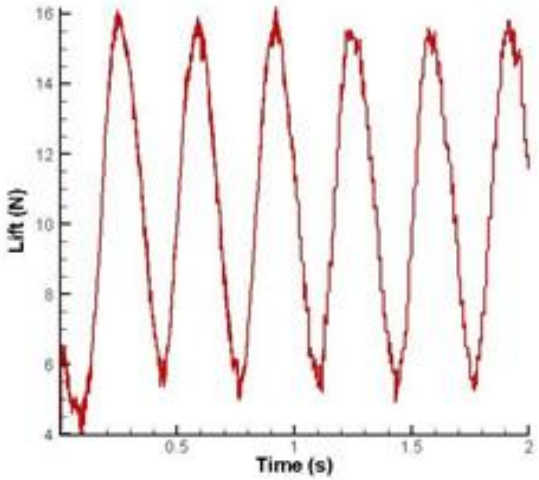


Figure 8. Analysis of the lift generated as a function of time during multiple sinusoidal flaps of the airfoil.

Ideal Velocity Study. A second study was done to determine the minimum velocity the bird should fly at to maintain the appropriate amount of lift. This study was done at an angle of incidence, θ_1, of 0°. \From the previous results shown in the "One Degree of Freedom Study", without an angle of incidence the greatest overall lift with the lowest amount of drag will be produced. The vertical rotation, Ψ_1, was approximated as a sine wave with an amplitude of 42°. This is different than the previous value of Ψ_1 used and therefore all inlet velocities must be re-evaluated for this portion of the study. The airfoil was re-evaluated at inlet velocities of 3, 5, 7, 10, and 12 m/s to provide a range of possible lift values per velocity with $\Psi_1 = 42°$, $\theta_1 = 0°$, and $\Phi_1 = 0°$.

It was determined, as expected, that with an increasing velocity, the lift generated also increased. The lift to weight ratio (*L/mg*) for the secondary portion of the wing was calculated in order to determine the minimum flight velocity to generate enough lift to keep the bird in a sustained flight. A lift to weight ratio of at least one was necessary in order to generate sufficient lift to overcome the gravitational forces on the bird. The target mass specification for the bird prototype was less than 1 kg and therefore a lift force of at least 9.81 N was required to maintain flight. However, anticipating that 1 kg is the worst case scenario for the mass of the bird prototype a number of different mass options were compared in Figure 9. The results displayed in Figure 9 are based on only the lift generated over the secondary wing section on the bird prototype.

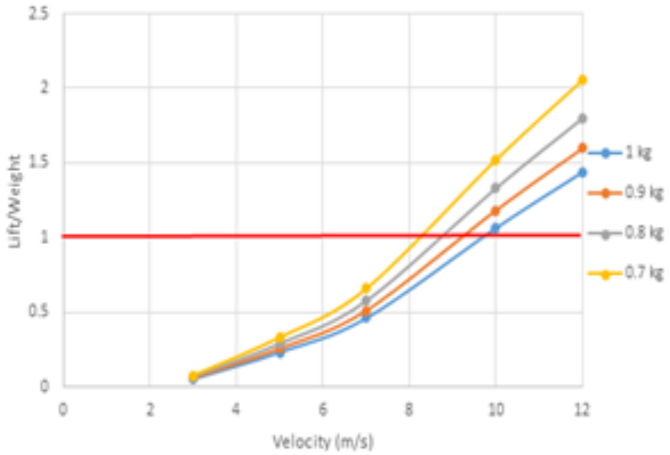


Figure 9. Ratio of the lift generated to the weight of the bird as a function of the flight velocity. The ratio of lift to weight must be greater than 1 in order to maintain sufficient lift.

After analyzing the results, a velocity of at least 9.7 m/s is required for sustained flight, based on the 1 kg mass of the bird. This velocity is much greater than the anticipated final velocity required to keep the bird in flight because the lift generated from the primary wing section has not been included. While the primary section will not produce a comparable lift force to the secondary section, it will be significant enough to reduce the required flight velocity.

Primary Wing Section. Once the evaluation of the secondary section of the wing was completed, the primary section of the wing was analyzed on a similar case study basis to determine lift and thrust produced. By determining the lift generated from both wing sections, the target velocity could be calculated based on the maximum allowable weight of the bird prototype. An expanded C code was written in order to take into account the angular rotation across all three degrees of freedom, as compared to the single degree of freedom system for the secondary wing portion. The Ψ_2 value was determined to be a sine wave with approximately 38 degrees of amplitude. This amplitude is based off of the findings of Liu [5], and is a comparable match to the vertical range of a seagull. Ψ_2 and θ_2 do not maintain any consistent rotational patterns during the flight of a seagull; however the motion of the prototype wing will maintain one set path. The goal of the primary wing section case study was to determine the combination of angles of Ψ_2 and θ_2 that produced the maximum lift and thrust values. θ_2, as similar to the secondary wing portion, can range from 0 to 45 degrees from a horizontal neutral position. Ψ_2, which is the retraction of the wing towards the body during the upsweep of the flap cycle in order to reduce air resistance through a surface area reduction, can effectively range from 0 to 55 degrees, in which 0 degrees is a fully extended wing. It was also desired to measure a range of possible flight speeds as the final mass of the bird model has yet to be determined. A sample of the flight path that was considered during the case study is provided as Figure 10.

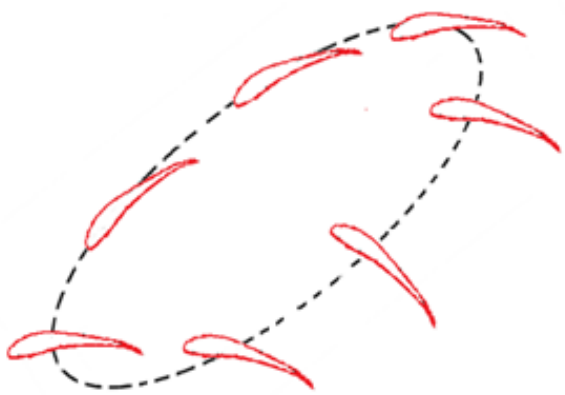


Figure 10. Proper flight path of the primary wing section.

Also taken into account when constructing the C code for the primary wing section is the fact that the amplitude of the wing tip will be exaggerated by the motion of the secondary wing portion. The wing tip has a longer moment arm from the body of the bird. The amplitude of the secondary wing section was 42 degrees of rotation. Since the secondary and primary section rotate in the vertical plane independently, if the secondary portion remained neutral, the primary potion would only translate the 38 degrees of a full flap. In order to further simplify the design of the C code it is assumed that the airfoil maintains a consistent distance from the body of the bird, while in reality a semi-circular pattern would be followed as the wing wraps around the bird. It was necessary at this point in the research process to maintain a 2-D model for simplicity. Since only a 2-D airfoil was used as a measuring device, a triangular surface area was used to estimate the effective area of the wing section rather than a rectangular section which was employed for the secondary wing. While the true prototype wing is not a perfect triangle, an underestimation of the surface area during CFD calculations will only increase the effectiveness of the end design.

Optimum Angular Rotation Study. When examining the primary wing portion, it was desired to determine the optimal combination of angles to generate a maximum overall lift and minimize drag at a variety of different flight velocities. Test velocities consisted of 7 and 10 m/s, which were estimated to be the limits of attainable velocity for the model, as well as the velocities necessary to produce enough lift based on the estimated prototype weight. Ψ_2 was examined at values of 15, 30, 45, and 55 degrees, while θ_2 was evaluated at 15, 30, and 45 degrees. Lift and thrust were calculated at each possible combination of parameters consisting of 24 different test cases, while maintaining a consistent vertical rotation, Ψ_2, of 38 degrees. The setup for this case study can be found in Appendix A where each combination of angles examined during the case study has been defined explicitly.

The drag force results of the case study are provided in Figure 11 below and were the main focus of the case study, as all of the possible thrust production occurs in primary wing portion. Any combination of angles under or at a zero Newton force are considered a thrust force rather than a drag force. Only two test cases produced a noticeable negative drag force. It was decided that the optimum angular combination would occur with θ_2 at 15 degrees and Ψ_2 at 45 degrees at a flight velocity of 7 m/s. While the test case of 55 degrees in the back sweep motion produced a greater negative drag force than when a 45 degree back sweep is implemented, it was determined that the prototype materials could not sustain such an extreme range of motion during flight and would be ineffective for the project. There were no test cases at 10 m/s of flight velocity that

generated a thrusting force, and for this reason the velocity of the prototype must remain at or below 7 m/s to maintain steady flapping flight.

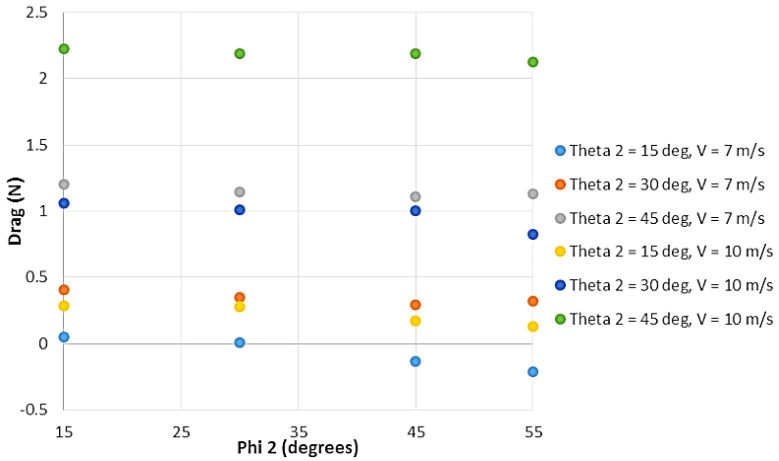


Figure 11. Results of the case study comparing the drag force experienced by both wings for each combination of back sweep and torsional angles.

Figure 12 is a pressure contour of the primary wing section undergoing 7 m/s of incident velocity from left to right. The angular rotation about the z axis for this case is 15 degrees, and the back sweep angle is 45 degrees. The current position of this wing is at the full inflection of 15 degrees torsion, and is fully extended in the neutral position as the wing reaches the full downward position of the flap cycle. There is a low pressure system generated above the airfoil, and in front of the leading edge. This allows for additional lift generation which can be combined with the lift generated from the secondary wing section. There is also a high pressure pocket developed below the airfoil during the down sweep motion. This high pressure system will push up on the airfoil as well as forward into the low pressure leading edge system. In this manner the thrust is produced for the entire flap cycle. The upsweep of the airfoil is the most significant source of drag during the flap cycle, however, the drag is not substantial enough to overcome the thrust produced.

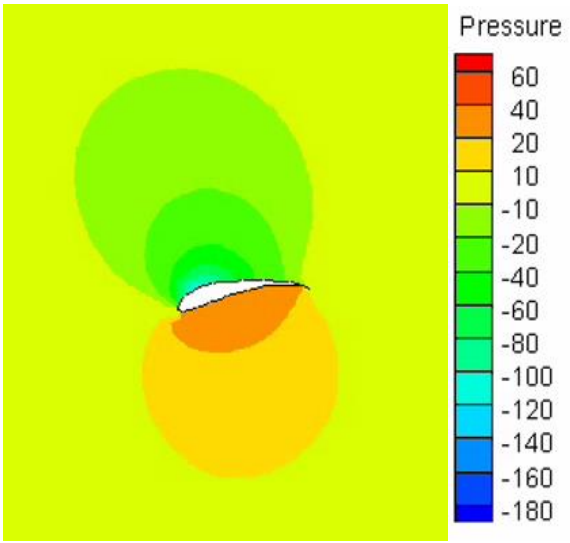


Figure 12. Relative pressure contour of the S1223 airfoil during the down sweep of the flap cycle.

The drag as a function of time data displayed in Figure 13 experiences a Ψ_2 of 45 degrees and a θ_2 value of 15 degrees with a flow velocity of 7 m/s. The majority of the flight time experiences a negative drag, meaning that it is actually a thrust. In order to maintain steady flight it is not necessary to have a thrust during the entire flap cycle, only enough to overcome the overall drag of the bird body and wing.

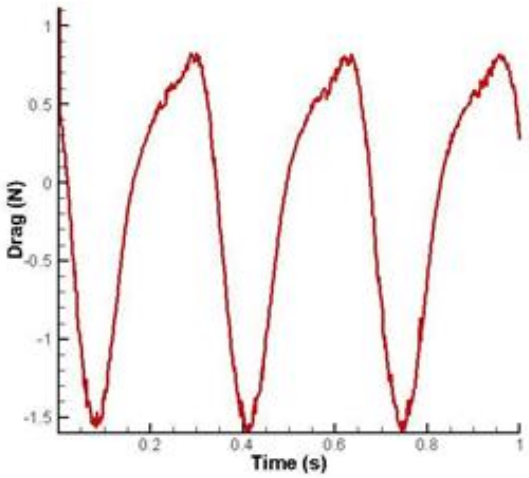


Figure 13. Drag and thrust forces generated by the flight motion of the airfoil during each position in time. Negative drag forces are considered thrust.

The lift force produced by the primary wing sections was greatest when θ_2 was at 15 degrees, regardless of the effects of the back sweep motion. It was determined in the secondary wing section case study that a zero degree value for θ_2 would generate greater lift, however since the primary section is needed for the thrust, some lift forces were sacrificed to generate a significant thrust force through the torsion around the z axis.

***3-Dimensional Wing*.** Results of the three-dimensional analysis were inconclusive. This is because limitations with ANSYS licensing, computational time and access to a super-computer capable of running these simulations was not available. The super-computer available did not have proper licensing to run ANSYS Fluent and the allotted budget did not allow for a full purchase of this package. Computational time to run one case utilizing all four degrees of freedom actuated in the wing would require at least one month of constant computing. By utilizing the ANSYS license provided by Virginia Tech, one case was run with only two degrees of freedom actuated. After analyzing this case, it was determined that the number of cells and nodes used within the mesh was not sufficient for any meaningful results. Figure 14 shows the difference between the original mesh generated and the mesh structure after one quarter of the flap cycle.

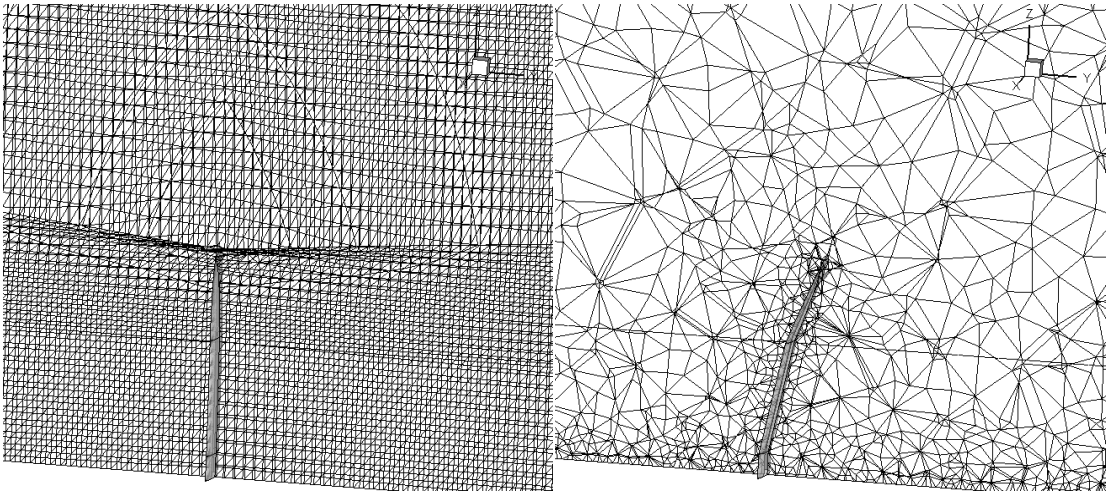


Figure 14. Comparison of the mesh density and re-meshing function before and after one quarter of the wing flap.

The original mesh is uniform with 1,000,000 nodes and would provide sufficient data across the entire wing. However, the altered mesh experienced an exponential decrease in nodes to less than 100,000 in a limited number of time steps. The initial parameters set for the dynamic re-meshing were insufficient in maintaining the integrity of the mesh resolution during a 3-D simulation. Figure 15 below is the result of a simulation based on the above grid resolution.

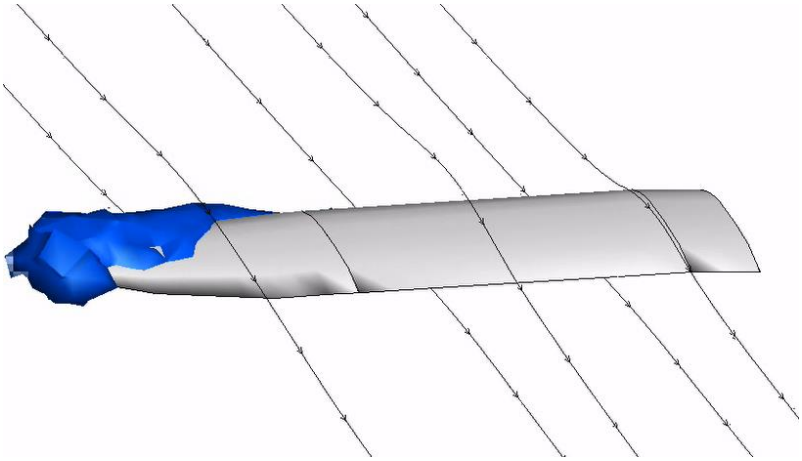


Figure 15. Shown are the results of the vortices produced after completing a 2 degree of freedom study with rotation in the vertical direction on both the primary and secondary wing sections.

Preliminary results show the majority of vortices form at the leading edge and wing tip. For this reason it can be confirmed that the other two degrees of freedom, namely backsweep and torsion actuated by the primary portion of the wing, need to be included in the overall kinematics for flapping flight. With the inclusion of these two degrees of freedom, drag and vortices should be reduced over the course of the flap cycle. From Figure 15 it can also be assumed that the mesh resolution and dynamic mesh parameters need to be refined when running future cases in 3-D. Further analysis should be done on a super-computer with proper licensing and computing capabilities. When completing this analysis, it is suggested that 10,000,000 cells be used within the mesh to accurately determine aerodynamic forces present on the wing over the specified

motion. It is also suggested that complex dynamic mesh parameters including Laplacian smoothing and sizing functions be utilized to reduce dynamic mesh deformations over the course of the flap cycle.

Through the work completed by the CFD team, unsteady flapping flight can effectively be modeled using computational fluid dynamics in a 2-D study and applied to actual bird flight kinematics. After completing both the secondary and primary case studies in 2-D, optimal amplitudes for the vertical rotation of the secondary section, the vertical rotation of the primary section, the back sweep of the primary section and the torsional rotation of the primary section were determined. These amplitudes were given to the CAD and Controls teams to actuate the four degrees of freedom system needed for sustained flight.

Mechanical Design

Research at the Smithsonian and Virginia Falconers. In order to validate the data taken by Dr. Liu, as well as to explore the flight kinematics of other birds, the team took a trip to the Smithsonian and Virginia Falconer's Club. For our studies, the team chose a red tail hawk. The team chose this bird because they have a flap rate of approximately 3 Hz, which closely matches the flap rate of the seagull. Also, these birds' wings are optimized for lift production, which is one of our top metric requirements.

At the Smithsonian, an ornithologist showed us a recently deceased red tail hawk that still had full joint and tendon flexibility. By manipulating the hawk wing and using a ruler, the team took measurements for each of the primary bones: humerus, radius/ulna, and wrist. In addition, the team measured the lengths of both the primary and secondary feathers. Lastly, a protractor was used to measure the length angles of maximum rotation at the shoulder, elbow, and wingtip.

After taking our measurements of the hawk body, the team was shown fully extended wings from both a red tail hawk and a seagull. Our goal was to create a 3-D model of the wing using the techniques of stereovision, as shown in Figure 16. First, the team covered each wing with several circular yellow markers. By taking pictures of the top and bottom surface of each wing as well as the calibration matrix, the team generated a point cloud that describes the shape of the wing. After converting the point cloud to the 3-D model, the team could extract airfoil profiles extending from the shoulder connection to the edge of the primary feathers.

Figure 16. A depiction of the stereo vision setup used to gather data at the Smithsonian Ornithology department.

Furthermore, we organized a visit with the Falconer's Club, shown in Figure 17, to get kinematic data of the red tail hawk's flight to compare it to the kinematic data from Liu's model and the kinematics of the previous year's model. As in the Smithsonian trip, the team used stereovision with the intent of creating a 3-D kinematic model of the hawk flight. In this test, however, the team used two GoPro Hero 3's instead of the NEX-6 camera and took multiple videos of the hawk at frame rates of 120 and 240 Hz. Unfortunately, however, when processing the data we found that there was too much point occlusion during the hawk's flap cycle to build an accurate kinematic trajectory. Therefore, we were forced to base our kinematics on those modeled by Dr. Liu.

Figure 17. Stereo vision setup used to collect flight kinematics data for the red tailed hawk.

Design and validation of the CAD model kinematics. Because we were unable to derive our own biological flapping flight kinematics, we based our work on Dr. Liu's work [5]. Figure 18 demonstrates Ψ_1 and Ψ_2 (secondary and primary rotation angles respectively) versus non-dimensional time for a full flap cycle.

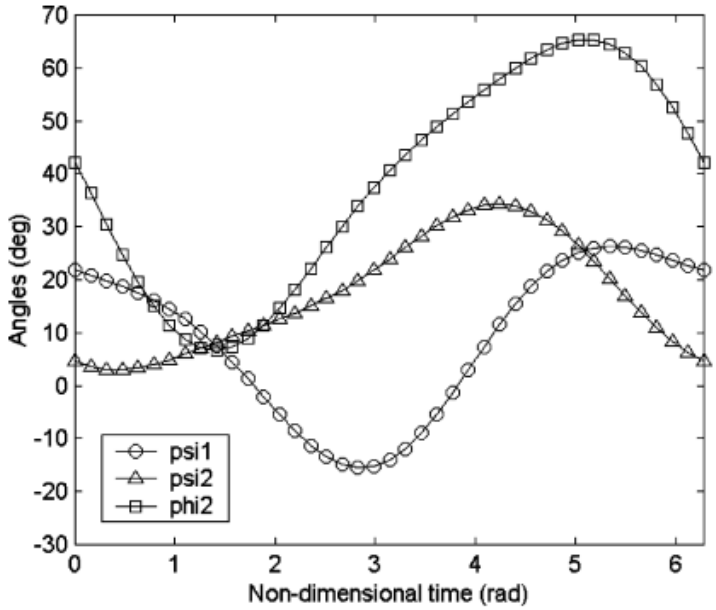


Figure 18. Rotation angles versus non-dimensional time for the secondary and primary sections (Ψ_1 and Ψ_2 respectively) from Dr. Liu's work.

Because this is the only kinematic data that we could reference, we used this as the kinematic basis for our CAD model. From our research, we found that the primary take-away from this graph is the phase difference between Ψ_1 and Ψ_2. This phase difference describes the timing of the primary section downstroke with respect to the secondary section and remains relatively constant over multiple flap rates. From Figure 19, the phase difference is almost exactly one unit of non-dimensional time. Therefore, we iterated the linkages of the CAD model until our phase difference between Ψ_1 and Ψ_2 was also exactly one unit of non-dimensional time as shown in Figure 19.

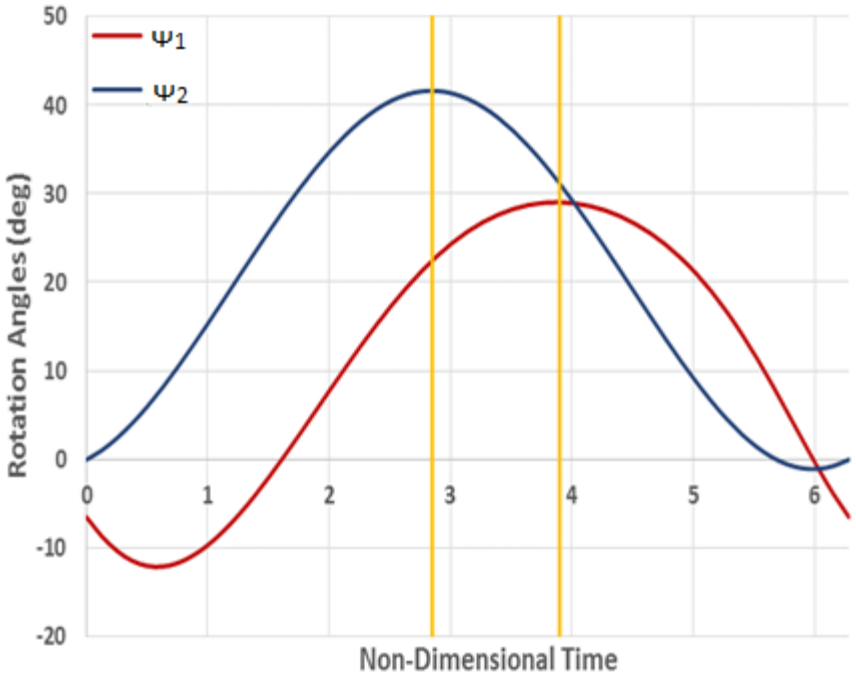


Figure 19. Rotation angles versus non-dimensional time for the secondary and primary sections (Ψ_1 and $\Psi_2$respectively) of our CAD model. Phase difference is exactly *one* unit of non-dimensional time.

It is also important to note that the amplitude of the secondary section angular rotation is slightly higher in our CAD model's kinematics than in Figure 19. This is because Dr. Liu's data was taken for a seagull flying at a high velocity, which requires a smaller flap amplitude in order to maintain level flight. Also, our CAD model was modeled to use the optimized 45˚ backsweep angle that our 2-D CFD simulations predicted. This is in contrast to the 65˚ backsweep angle shown in Dr. Liu's data. This difference is also due to the high velocity case that Dr. Liu modeled. At high velocities, birds increase their backsweep angle to minimize their wing surface area on the upstroke and thus minimize drag. This backsweep motion is illustrated in Figure 20, along with the 15˚ amplitude of torsion that was also predicted by our 2-D CFD simulations.

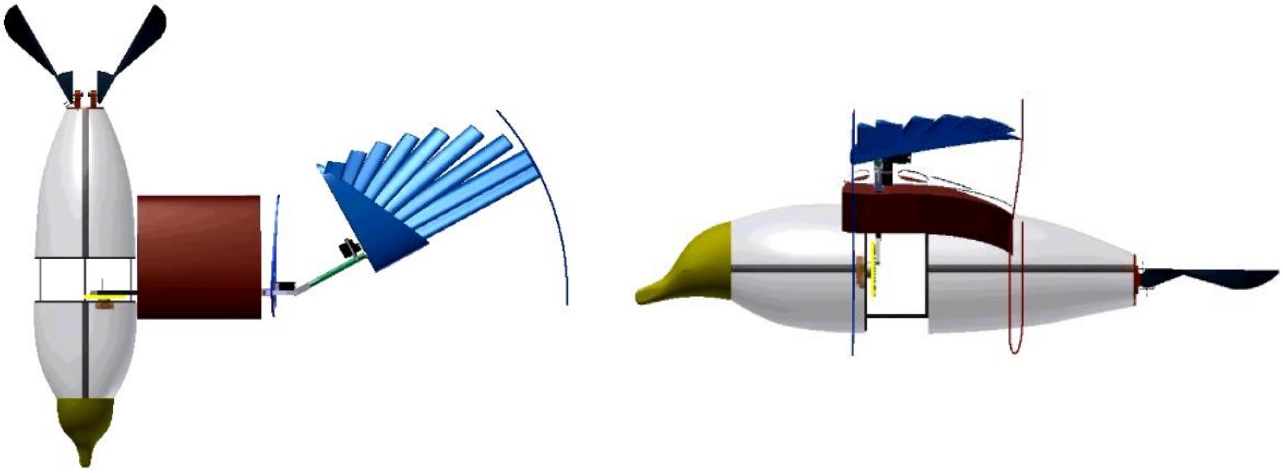


Figure 20. Illustration of backsweep (left) and torsion (right) motions that were produced in the CAD model.

Each of these motions were created using Dynamic Simulation in Autodesk Inventor. This program also allows us to combine all of these degrees of freedom into a single flap cycle. By tracing the wingtip of our CAD model for a flap cycle with all four degrees of freedom actuated, we produced the following wingtip trace in Figure 21.

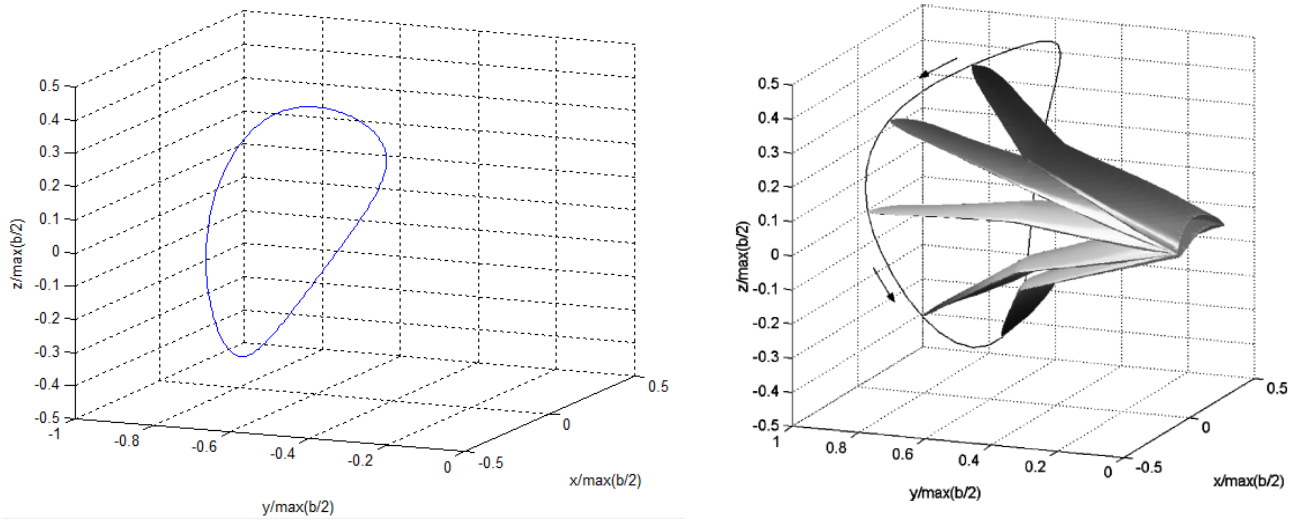


Figure 21. Wingtip trace of our CAD model (right) and Dr. Liu's wingtip trace. Upon visual inspection these two traces are very similar in magnitude and shape.

Upon visual inspection, our wingtip trace appears to be very similar in both shape and magnitude to the wingtip trace produced by Dr. Liu's data. Because we do not have access to the raw data Dr. Liu used to make his wingtip trace, we can only compare our data through visual inspection. By the comparison of these two traces, we were able to validate that the kinematics of our CAD model closely match those of a real bird.

Manufacturing Process. The manufacturing process for many components found on the bird model involved numerous steps. Specifically, the two wingtips were made in the same way. First, the shape was designed and printed using an FDM additive manufacturing machine. In each case, the team used the machine available to us with the smallest resolution, and with the smallest workable infill. As the additive part would not be the final piece used on the model, strength was not a concern, only obtaining the proper shape. Once the part was printed, shown in Figure 22, it was then used to create a mold, which was done simply using silicone. Once the mold was cured, we then used an expanding two part liquid urethane foam with a 4 lb/ft^3 density to cast the final parts. The four pound density was chosen based on the limited knowledge of the team as most likely to be able to withstand the forces associated with flapping flight, while at the same time still minimizing weight.

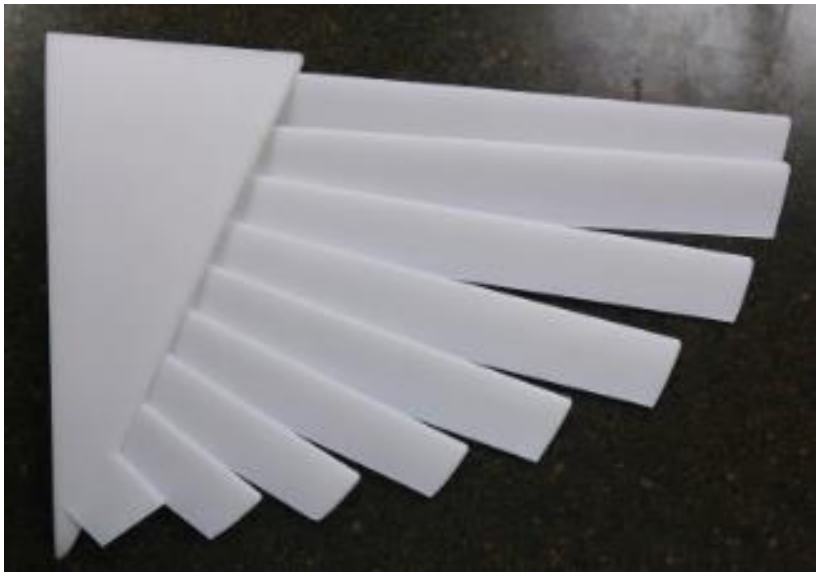


Figure 22. Silicone printed mold for the wing tips.

The purpose for printing the final shape of the desired parts, creating a mold, and then using that mold to cast the final production part was in order to minimize the amount of material needed by the additive machine. This was a concern for the larger wingtip pieces, as the size prevented us from using machines available to students at no cost through Virginia Tech. However, this was not the case for the head of the bird. Again the four pound urethane foam was used for its' creation, but the mold used for its casting, rather than the head shape itself, was printed on a MakerBot Replicator 2 machine.

The casting process itself was relatively straightforward. The interior surface of the mold was first prepared with a release agent, and the two parts of foam were then combined. The liquid urethane mixture was then poured directly into the mold, and allowed to expand and cure for roughly seven minutes. The mold halves were then gently pried apart and the final shape was removed, then briefly sanded, shown in Figure 23.

Figure 23. The head mold has been created from a liquid urethane mixture and the mold halves have been separated from the head.

Beyond this, the airfoils used on the secondary section of the bird wing were also made entirely of foam. In this case, instead of molding a shape, the team made use of a Model Maker rapid prototyping machine. This machine CNC's a shape directly from foam, which could then be used directly on the final bird model.

The old frame comprising the body of the bird supports the motor, drive train, and shoulders used was constructed entirely of carbon fiber. It was determined that this level of strength was not necessary. Instead, the team moved to a more lightweight composite made from balsa wood sandwiched between thin carbon fiber laminate, shown in Figure 24. Through this change, and a reduction in the total amount of carbon fiber used in the design approximately, 51 percent of the weight from the body frame was lost. The underlying structure of the bird wings and joints remains entirely carbon fiber and aluminum, however.

Figure 24. Old frame shown on left and the updated frame shown on right. The updated frame has a 51 percent reduction in mass over the old frame.

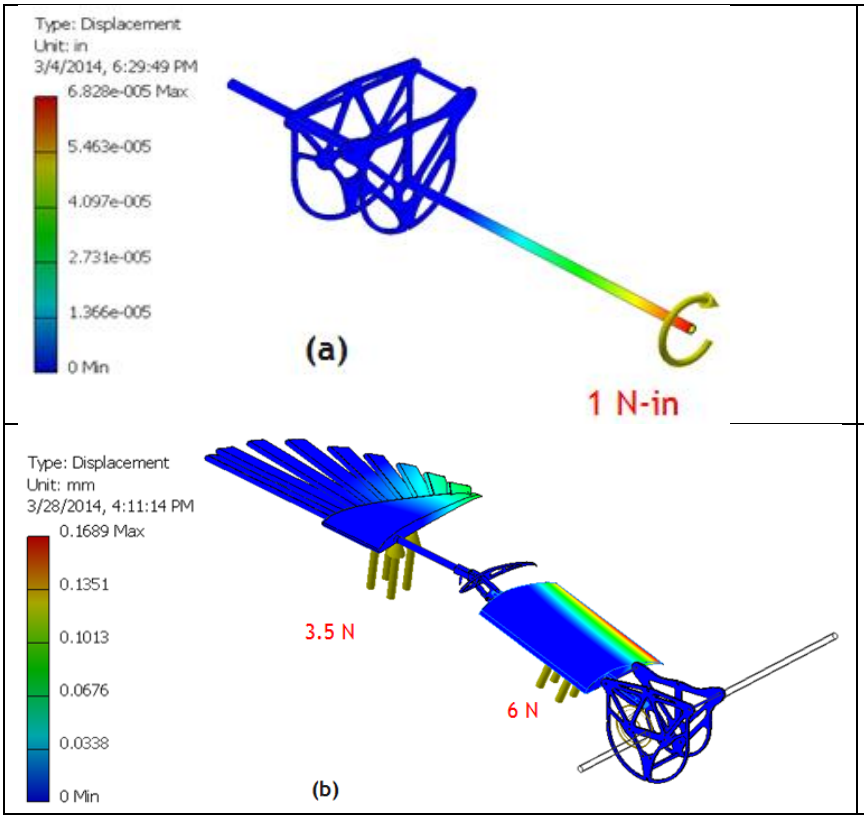
The center rod running down the updated bird body was attached securely to the frame using screws and epoxy resin. In order to accomplish this, the screw was fed through a hole in

the frame, and placed within the end of the carbon fiber rod. From there, epoxy was mixed and used to fill the space within the rod around the screw threads. The epoxy cured around the screw and adhered to the rod, providing an excellent connection.

As mentioned, the joints of the bird are made from machined aluminum, as they previously were. However, changes were made to the way in which the pieces are connected; in place of the bushings added previously, press fit rods were used instead. This change had the effect of simplifying and streamlining construction while still keeping friction at a minimum.

Finite Element Analysis Results. To reduce the weight of the previous model, the body of the bird was re-designed to reduce the amount of material used, also shown in Figure 25. However, the reduction in material raised concerns about the body's structural soundness during use. To determine if our re-designed body continued to be structurally sound, the team used the FEA features built into Autodesk Inventor.

The first concern was from the amount of stress that the body rod would experience around the tail. By reducing the tail piece from four rods down to a single piece, the team worried about the torsional stress on the structure. Another concern was about the stresses that the foam pieces would experience during typical flight loads. From the FEA results for these scenarios, we found that the redesigned body was enough to withstand the stresses. Further cases were developed to model stresses that the robotic bird would have to endure should it crash during flight. The cases are illustrated in Figure 26. Even for the worst case scenario, Figure 26(d), the structure still managed to have a minimum factor of safety of 3.52. Because of this, the team is confident in the structural soundness of the re-designed body.



Torsional stress near the tail.

Finite Element Analysis for the primary and secondary foam sections on the wing.

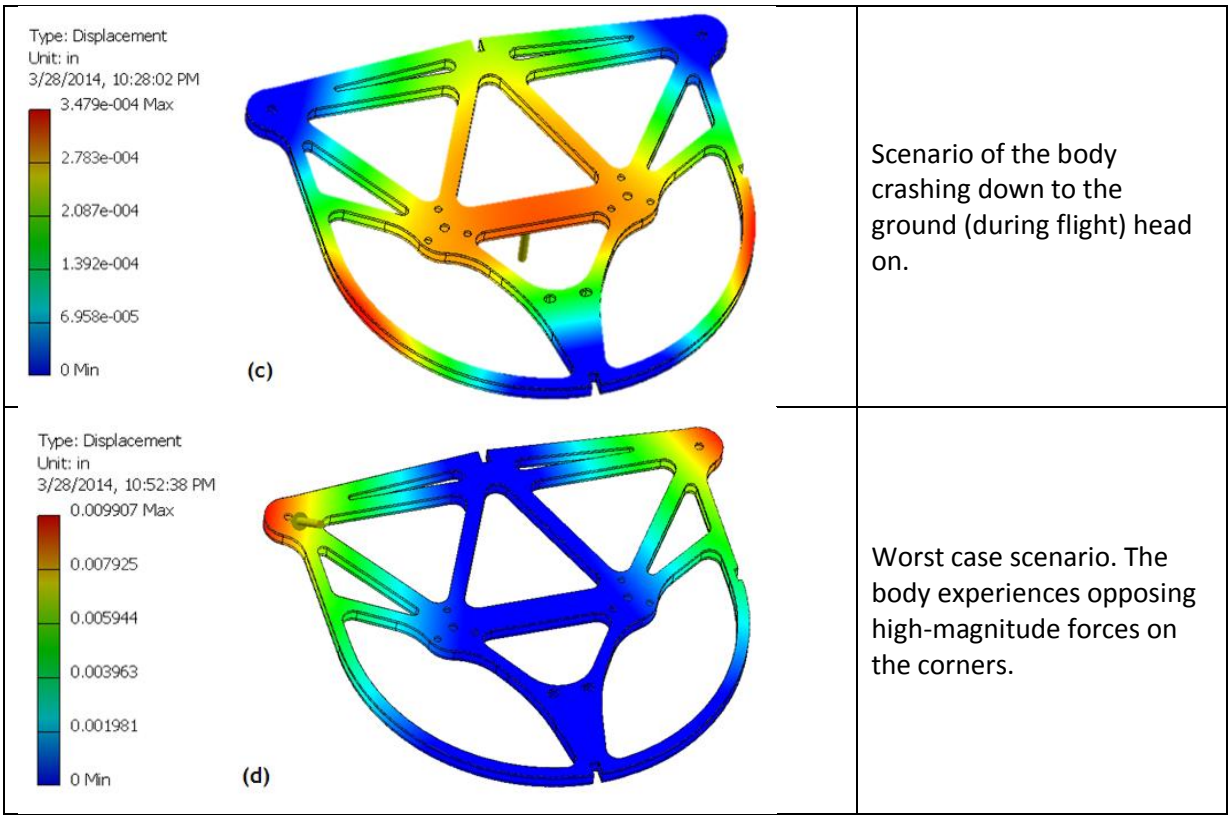


Figure 26. The FEA results showing the areas of maximum stress for different scenarios.

Actuation of physical prototype. To actuate our physical prototype, we used a Maxon EC 16 motor to drive the vertical flapping motion of the secondary and primary sections and micro-servos to drive the backsweep and torsion of the primary section. Each of these motor types is depicted in Figure 27.

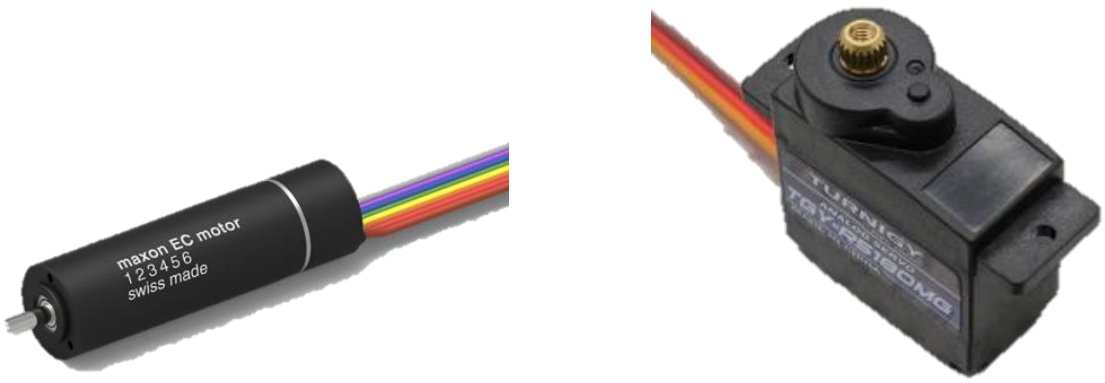


Figure 27. Maxon EC 16 motor (left) and Turnigy micro-servo (left).

A single Maxon motor was used to drive both flaps at the same time and frequency by meshing two gears of the same size together. While this eliminated the need for a second motor and therefore reduced the weight of the body, it doubled the load that the Maxon motor had *to carry*. Using Dynamic Simulation, we analyzed the torque required to flap the wing and generated the plot shown in Figure 28 for flap rates of 1, 2, and 3 Hz.

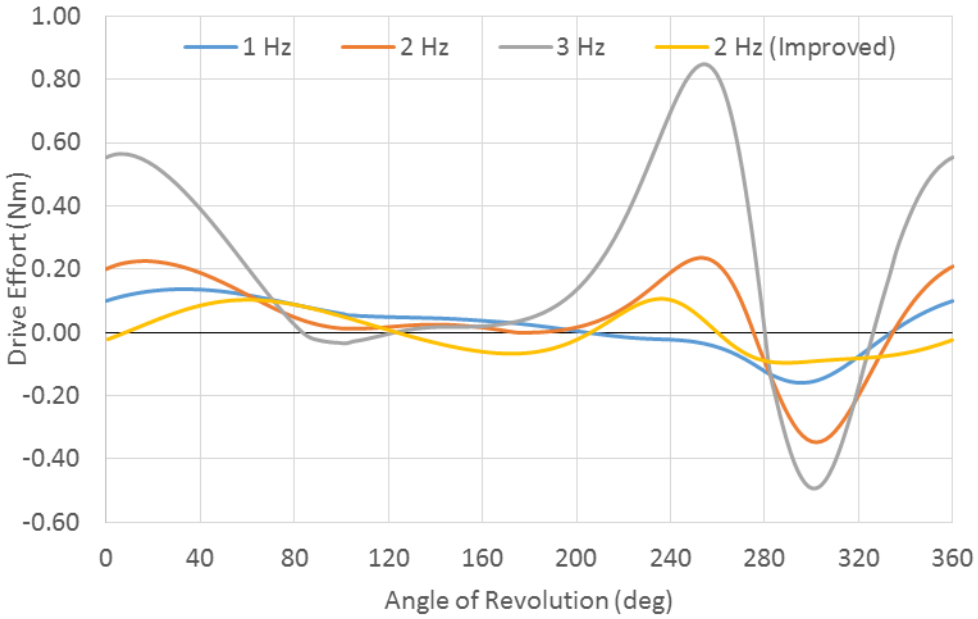


Figure 28. Torque versus drive gear angle for flap rates of 1, 2, and 3 Hz. Also shows how the addition of a spring significantly reduced the drive effort required for a 2 Hz flap rate, "2 Hz (Improved)".

Figure 28 indicates that a significant change in the torque required to maintain a constant flap rate occurs at approximately 280˚ drive gear angle (the bottom of the flap). At our target flap rate of 2 Hz, this torque "spike" has an amplitude of 0.6 N-m. However, the maximum continuous torque of the Maxon EC 16 is only 0.13 N-m; this suggests that our motor would struggle or possibly never achieve our target flap rate. In order to alleviate the stress on the motor, we used Dynamic Simulation to introduce the effects of a compression spring on the torque requirements of the motor. This configuration is illustrated in Figure 29.

Figure 29. Example of introducing a compression spring to alleviate the torque load on the drive motor. The spring properties that we simulated were k = 500 N/m and free length = 40 mm.

The addition of this compression spring reduced the torque load on the drive motor by 66%, as shown in Figure 28 as "2 Hz (Improved)". By reducing the torque "spike" to only 0.2 N-m as well as inertial effects from the wing, the Maxon motor was able to produce a constant 2 Hz flap rate.

We used a separate micro-servo to drive the back sweep and torsion of each primary section, a total of four micro servos. Each servo was placed according to Figure 30, with the first mounted at the base of our molded wingtip and the second mounted onto the wrist joint itself. Because each of these servos could rotate up to 60˚ in 0.1 seconds, they were plenty fast enough to drive our primary section's degrees of freedom. Also, due to their light weight of only 13 grams, they did not significantly contribute to the load of the drive motor despite having a relatively long moment arm.

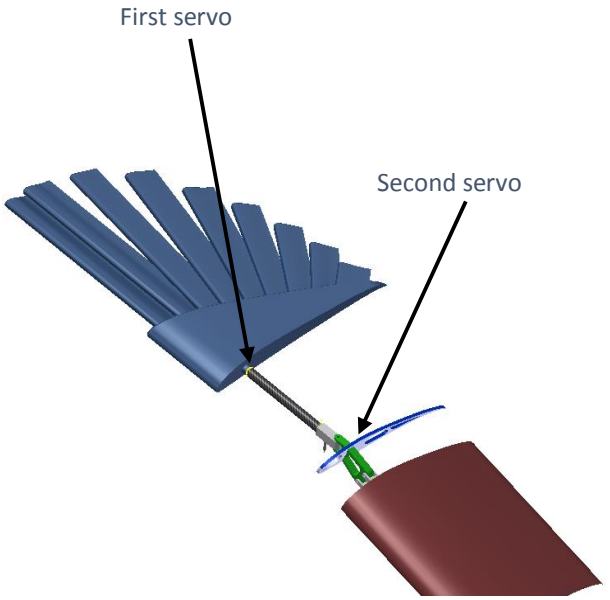


Figure 30. Placement of micro-servos to drive backsweep and torsion. First servo drives torsion and is mounted at base of wingtip, whereas second servo drives backsweep and is mounted on the wrist joint itself.

Design of a Microcontroller for a Flapping Wing Micro Aerial Vehicle

This section first presents the procedures and assessment of the design to control wing kinematics with the ATmega2560. It includes implementation of a motor driver, the addition of an absolute encoder to measure the exact position of the secondary wing section during a flap cycle, and the use of servos for the actuation of the backsweep and wingtip rotation of the primary wing section using data obtained from the secondary wing section. The procedures for and assessment of adding serial output to the ATmega2560 design to communicate the kinematic data for validation is then presented.

Introduction. Presented in the following sections are the design of a microcontroller for the control and data collection system of a flapping wing micro aerial vehicle. The system uses the Atmel ATmega2560 8-bit AVR microcontroller mounted on an Arduino Mega 2560 board.

The design makes use of the ATMega2560's digital input/output pins, the Master/Slave SPI Serial Interface, the fifteen 8-bit PWM channels, and the Programmable Serial USART interface. Servo and DC motor control circuits were built and control software was written using the Arduino language platform. Additionally, the design included a serial UART connection between the ATmega2560 and a remote personal computer. This connection served to monitor wing angles and control motor speed using a National Instruments LabVIEW VI.

The design had two main objectives. The first objective was to use the ATmega2560 to properly control the actuators of the robot. Part of this objective was to ensure a kinematically accurate representation of bird flight. The second objective was to use the ATmega2560 to provide measurements and feedback of system performance.

Connecting a Motor Driver Circuit to the ATmega2560. Hardware was added to control the main drive motor responsible for the flapping of the secondary wing section. This hardware included a motor driver attached to a brushless DC motor and the ATmega2560. This circuit was designed according to the specifications outlined by the Maxon DEC Datasheet [6]. For the complete motor driver circuit, see Appendix B.

Central to the circuit was the Maxon DEC 24/2 Motor Driver Module (DEC). The DEC Module is a small 1-quadrant digital controller for the control of a Maxon EC16 16 mm 30 W Brushless DC motor with built-in Hall sensors (EC16). The DEC provides the +11.1 volt signal to each of the three motor windings and internally regulates a +5 volt signal to the three internal hall sensors of the EC16.

The DEC provides digital speed control using the internal hall sensors of the EC16. This control is done via the digIN1 and digIN2 input pins of the DEC. These were connected to pins PA2 and PA3 of the ATmega2560. The Enable input of the DEC enables or disables the power stage of the motor controller. Finally, the Ready output can be used to report the state of operational readiness or a fault condition. During normal operation the output of this pin reads logic high. In the cause of a fault condition, the output is switched to ground. This pin was connected to reset pin of the ATmega2560. Pin connections for the DEC are shown in Figure 31.

To control the added hardware, we programmed the ATmega2560 following the pseudo code and program listing given in Appendices C and D, respectively. The program shown in Appendix D introduces the subroutine initDECModule in the main loop, which was used to initialize and enable the motor drive module. First the digIN1, digIN2, setValueSpeed, and Enable pin are set to outputs. Next, the digIN1 and digIN2 are set to logic high and logic low, respectively. This puts the DEC into closed loop control with a 500 to 20,000 rpm speed range. Finally, the subroutine sets the Enable pin high to enable the power stage of the motor.

The set value input is performed through an external analog voltage (0 to +5 V). In the main routine, the state of the DEC is checked for any fault conditions. As long as there are no limiting conditions, the motor speed is written to PE4 (digital pin 2) using pulse width modulation (PWM). The set value voltage is given by the following equation:

$$V_{set} = \left(\frac{n - n_{\min}}{n_{\max} - n_{\min}} \cdot 4.9[V]\right) + 0.1, \tag{25}$$

where V_{set} is the set value voltage, n is the desired speed, n_{min} is the minimum speed, and n_{max} is the maximum speed.

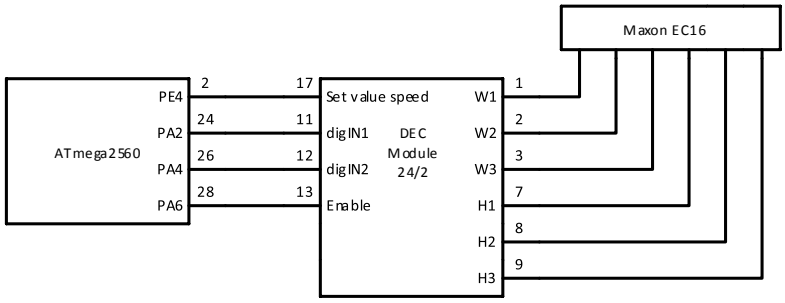


Figure 31. Motor driver circuit developed for the ATmega 2560 microcontroller using the Maxon DEC Module to interface with the Maxon EC16 DC Motor.

To test the operation of the hardware installation and the initDEC subroutine, we simply observed the approximate flap rate of the secondary wing section. Initial settings of the speed control pins digIN1 and digIN2 allowed for motor speeds up to 80,000 rpm. At these speeds, the EC16 was unable to produce enough torque to flap the wings. This caused a fault condition which required the DEC to be reset by pulling the Enable pin low. Afterwards, the motor speed was limited to 500-20,000 rpm which produced the necessary torque for continuous operation. There were no other successive problems with the motor driver circuit. Future work may include open loop speed control to provide a varying flap speed at different points in the cycle.

Positional Measurement of the Secondary Wing Section. In order to measure the angular position of the secondary wing section, the addition of hardware to the ATmega2560 controller was necessary. This hardware included an absolute quadrature encoder mounted to the right-hand main drive gear and attached to the SPI lines of the ATmega2560. For the complete circuit, see Appendix B.

The hardware used for this design was a CUI AMT 203 Modular Encoder (AMT203). This encoder transmits an absolute 12 bit angular location (4096 positions) of the main drive gear. This allows for an accurate measurement of the location of the secondary wing section at any point during a single flap cycle. As shown in Figure 32, the chip select bit (CSB) used for SPI communication was connected to the PL2 pin of the ATmega2560 (digital pin 47 on the Arduino Mega 2560 board).

To control the quadrature encoder, we programmed the ATmeg2560 following the pseudo code and program listing given in Appendices C and D, respectively. The program shown in Appendix D consists of three subroutines that were called from main. The three subroutines were named SPI_T, initCUI, and setZeroCUI. The subroutine SPI_T was added to simplify the sending and receiving of data over the SPI serial connection. The subroutine first declares an 8-bit unsigned integer which holds the received data. Next, it pulls the CSB pin low to allow communication with the AMT203. The subroutine then sends the message and stores the new message in the temporary space. The subroutine then pulls the CSB pin high to cease communication before returning the received message.

The subroutine initCUI was used to initialize the encoder. This subroutine sets the slave select pin for the AMT203 to an output and pulls the pin high to ensure no erroneous communication with the encoder. As shown in Figure 33, the most significant bit (msb) data out on MISO is valid soon after CSB goes low. The MOSI data is valid soon after the falling edge of

SCK. The Encoder drives data out on MISO as long as the CSB is low. Normally, CSB goes low, then after 8 clocks the command is interpreted. CSB high resets the clock counter and terminates any command sequence. In accordance with the AMT203 datasheet [7], the bit order is set to send the most significant bit first using the command SPI.setBitOrder(MSBFIRST). Next, the data mode sets the clock phase (CPHA) which shifts data in and out on the rising edge of the data clock signal and the clock polarity (CPOL) which determines whether the clock is idle when low. The SPI bus of the AMT203 runs full duplex and transfers multiples of 8 bits in a frame. The SPI type is the most common (CPOL = 0, CPHA = 0), also known as Microwire. Data is captured on the rising edge of SCK and the output data is changed after the falling edge of SCK, shown in Figure 33. This was done using the command setDataMode (SPI_MODE0). Finally, the command setClockDivider (SPI_CLOCK_DIV32) was executed to reduce the SPI communication clock from 16 MHz to 500 kHz.

All of wing movements are based upon the non-dimensionalized time for a full flap cycle (0 to 360 degrees). The wing motion begins from the bottom of the flap cycle. To calibrate this position, the subroutine setZeroCUI was used to zero the encoder at the bottom of the flap cycle. In this subroutine, SPI_T is used to issue the set_zero_point (0x70) command to the encoder. The code waits to receive an acknowledgement (0x80) as confirmation of the set zero execution.

Similarly, to read the absolute position of the secondary wing section, SPI_T is used to issue drupes (0x10) command from the ATmega2560. The microcontroller receives a series of wait sequences (0xA5) then a reflected rd_pos (0x10). This is immediately followed by the MSB data and then by the LSB data. MSB data is subsequently bit shifted left 8 places and added to the LSB in order to provide the 12-bit position. The binary position is then converted to degrees by the equation:

$$\deg = -(ABSposition * 0.08789 - 360), \tag{26}$$

where *deg* is the angular position of the gear and *ABSposition* is the 12 bit integer representation of the absolute position of the gear. This angular position was critical in subsequently determining the actuation of the primary wing section.

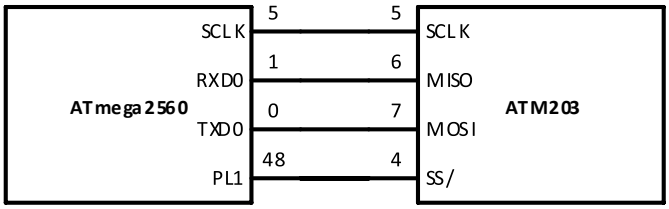


Figure 31. AMT203 Modular Encoder uses an SPI Interface to communicate absolute position of the secondary wing section to the ATmega2560.

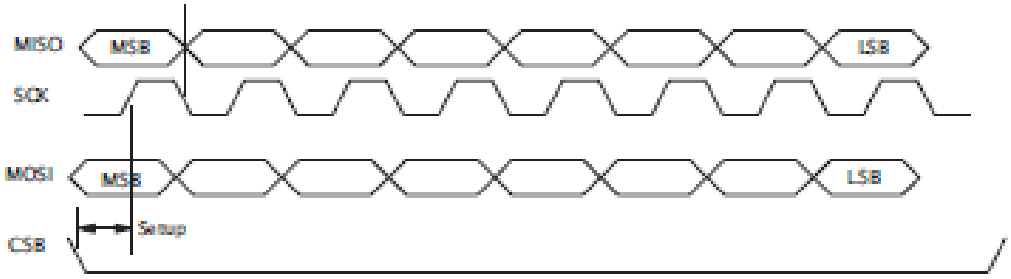


Figure 32. SPI BUS Timing Diagram for the AMT203 Modular Encoder.

While developing the design, there was some initial difficulty in the SPI communication of the absolute position. Originally the code would pull the CSB high and send all the data commands. The CUI encoder, however, will only send and receive 8 bits per data command. This was unclear in the datasheet and corrected in the code. The CSB is pulled high, a single byte is shifted in, and then the CSB is pulled low again. Further readings of the encoder showed to be reliable.

Control of the Backsweep and Wing Tip Rotation of Primary Section. Micro servos were used to manipulate each of the wrist joints which connect the primary and secondary wing sections. These servos provided the backsweep articulation of the primary wing section. Two additional servos at the wingtip were used to rotate the feathered wingtip of the primary section. Included in this section, and listed in Appendix E, is the addition of the calcTorsionAngle subroutine to the existing software developed.

The hardware used for this design was four Turnigy TGY-90S Micro Servos. The position control wires of these servos were connected to the ATmega2560. Using the preloaded Arduino bootloader on the ATmega2560, these pins offer built-in PWM capability to accurately position the servos using the Arduino servo library <Servo.h>.

To control the micro servos, we programmed the ATmega2560 following the pseudo code and program listing given in Appendices C and D, respectively. The program shown in Appendix D introduces the calcTorsionAngle subroutine. This subroutine uses the angle of the secondary wing section determined from the previous section to calculate the required rotation of the wingtip servos. The equation this subroutine uses was developed from computational fluid dynamic analysis and dynamic simulation to maximize lift and thrust production from the primary wing section. The subroutine calcTorsionAngle employs an if-else-if case structure which relates to the piecewise function shown in Figure 34 to determine what rotation angle to return.

The first case is an input angle less than 140 degrees. At this position, the rotation maintains a constant 30 degree angle of attack. The next case is an angle from 140 degrees to less than 195. In this case, the angle of attack is given by the equation:

$$\alpha = 3.810 \times 10^{-4} t^3 - 1.913 \times 10^{-1} t^2 + 31.19t - 1.648 \times 10^3, \qquad (26)$$

where α is the angle of attack (degrees) and t is the non-dimensionalized time (degrees). In the third case, the input angle is greater or equal to 195 degrees and less than 291 degrees. In this case, the secondary wing section has neared the top of the flap cycle. Finally, in the last portion of the flap cycle, the angle of attack is given by the equation:

$$\alpha = -1.874 \times 10^{-4} t^3 + 1.832 \times 10^{-1} t^2 - 59.02t + 6.267 \times 10^3, \qquad (27)$$

where again, α is the angle of attack (degrees) and t is the non-dimensionalized time (degrees). The subroutine calcTorsionAngle returns the calculated angle of attack. This angle is written as a PWM signal to the Turnigy Micro Servos using the servo.write command.

The micro servos attached to the wrist joint, which are responsible for the backsweep motion, are controlled using a sine wave function with an amplitude of 45 degrees and a vertical shift of 22.5 degrees. Similar to the servos responsible for wingtip rotation, the signal is written as a PWM signal using the servo.write command from the <Servo.h> library.

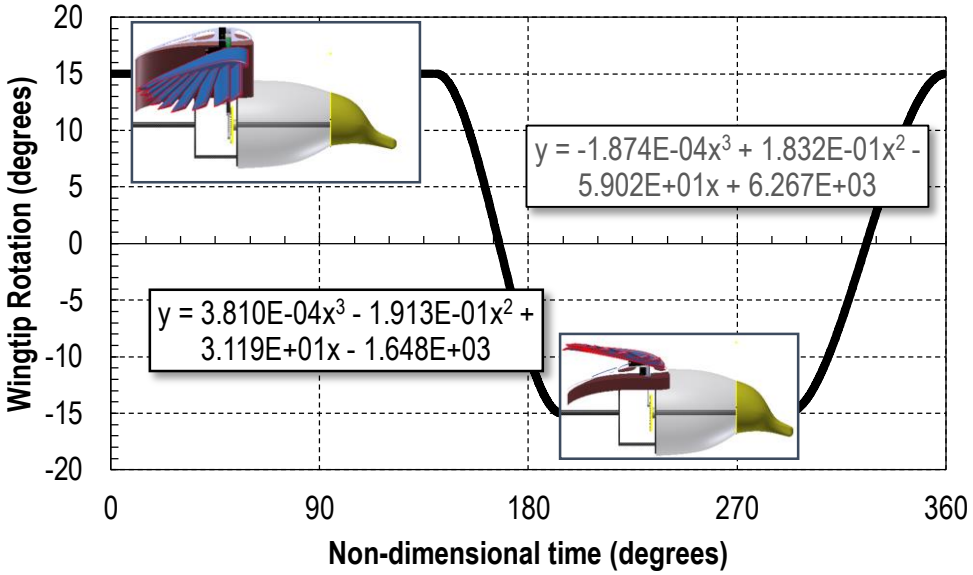


Figure 34. Function of wingtip rotation over a single flap cycle using a non-dimensionalized time domain.

While developing the design presented in this section, several mistakes and difficulties were encountered. Initial calculations of torsionAngle were done as an integer data type for writing to the rotation servo. This caused propagating errors which caused a choppy rotation of the wingtip. This was alleviated by changing the variable to a float data type and casting it is an integer before writing it to the servo. It was found, however, that the servo requires an unsigned integer. By repositioning the servo and adding 15 degrees to the calculation, we were able to simulate a -15 to 15 degree sweep by changing the movement from 0 to 30 degrees.

Finally, due to symmetrical mounting of the servos on different sides of the robot, the rotation of the wingtip was found to be opposite. By subtracting the rotation angle from 180 degrees, the movement was mirrored across the robot midline. A similar adjustment was made for the wrist servo responsible for the backsweep motion.

Adding Serial Communication to the ATmega2560. In order to provide communication with external monitoring software, the addition of one subroutine, serialEvent, was added to the existing software. The added subroutine is listed in Appendix D.

Initialization of the UART serial port was performed using the command Serial.begin(9600). This command sets the PE0 and PE1 pin of the ATmega2560 (digital pin 0 and 1 on the Arduino Mega board) into RXD0 and TXD0 mode, respectively. This opens up communication between microcontroller and the PC via the USB port at a baud rate of 9600 bits per second.

The software first multiplies the secondary wing section angle and primary section wing tip rotation calculation by 100. This allows the data to be transferred to the PC as an integer and post-processed to a floating point number with two significant digits.

After the outgoing data is converted to 2-byte integers, the software writes a 2-byte header (0xB562) to the TXD0 serial register using Serial.write() to indicate the beginning of data transmission. Next, the highByte and lowByte commands split the rotation and secondary angle integers into 1-byte blocks for transmission using Serial.write() command. The complete packet is post-processed by the PC using a National Instruments LabVIEW VI interface.

In addition to sending positional data, the ATmega2560 uses the RXD0 pin to receive desired changes in the motor speed. Continuous reading and changing of the motor speed can adversely affect processing times. In order to alleviate this, the interrupt service routine (ISR) serialEvent() was used to change the motor speed variable only when the microcontroller receives a signal on the RXD0 pin.

The serialEvent() subroutine simply monitors the RXD0 register for new data in a while loop using the Serial.available() command from the Arduino serial library. When new data is available on the RXD0 pin, the serial data is read using the Serial.read() command and assigned to the motorSpeed variable.

While developing the design presented in this section, several mistakes and difficulties were encountered. Originally, the baud rate was set at 115,200 to collect data as quickly as possible. However, at this rate, there was a significant amount of data loss. The baud rate was reduced to 9600 to account for this loss.

Refinement of the code in future work should include error checking to evaluate serial strings for data loss. These data packets should be discarded to provide a more accurate representation of robot articulation. Additionally, serial packets should contain software flow control to prevent overflow of serial registers.

Synopsis of Control Effort for Actuation. The objectives of this design were to use the ATmega2560 microcontroller to actuate the robot with kinematic accuracy and provide measurements of system performance. Both objectives were met. By using a motor controller and internal hall sensors of the EC16, proper flap rates were obtained from the secondary section. The addition of an absolute encoder accurately measured the angle of this wing section. These measurements were used to actuate the primary wing section movements using equations developed from fluid dynamics research and dynamic simulations. This information was successfully transmitted to a remote PC via the UART serial ports of the microcontroller.

Product Evaluation

Testing Structure. Creating a test structure for a dynamic, flapping bird presented many challenges. We created a number of ideas early on, but none of them were viable designs in their entirety. It was important for us to be able to obtain qualitative results from our prototype. We

decided the most important qualitative result that we could measure would be lift, and we began to tune our test structure plans around this idea. We went through many iterations of radically different ideas, and in the end created the design that is shown below in Figure 35.

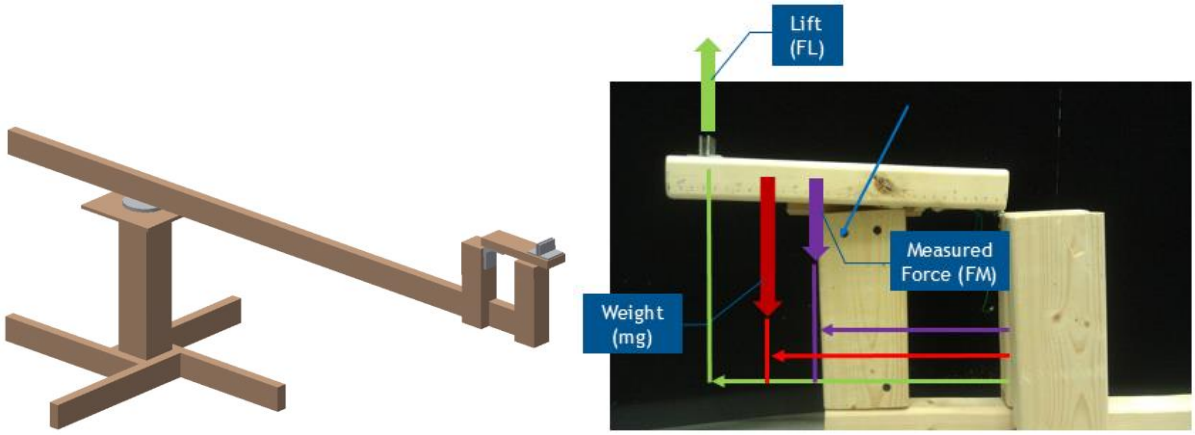


Figure 35. A CAD drawing of the test structure and present forces on top board. The main structural components are made of wood and the bird is placed six feet from the base.

This test structure elevates the bird a few feet from the ground on a main pillar, with a lazy susan bearing centered on top. This lazy susan bearing is attached to a wooden beam that extends six feet away from the main platform. At the tip of this beam, there is a structure with a hinged board that the bird is mounted onto. Below this hinged piece is a sensor that will be able to measure the force of the top board and weight of the bird. Since the top board is hinged, we can perform a static moment analysis on the bird to find what the lift forces are. Using Equation 28, we can take the measured force in the sensor and use that to find lift:

$$F_L = \frac{(mg * x_1 - F_m * x_2)}{x_3}, \tag{28}$$

where F_L is the lift force, mg is the force due to gravity, x_1 is the distance along the board from the hinge to the center of gravity of the combination of the bird and top hinged board with mounting hardware, F_m is the measured force that is read by the sensor, x_2 is the distance along the board from the hinge to the location of the sensor, and x_3 is the distance along the board from the hinge to where the bird is mounted.

The original sensors that we used on this test structure were 10 N force sensitive resistors. These sensors were picked out for an earlier prototype, but we decided that if the specs were good enough for that design, they should work well for this new hinged lazy susan bearing design as well. This was not the case unfortunately. For reasons that are still unknown, and after a few hours in the lab trying to get them to work, we have concluded that force sensitive resistors will not work for this application. After roughly fifty trials with many different independent variables being adjusted, we could not get a steady reading from the sensors that could be used to calibrate them.

Testing Structure Future Work. With the realization that the force sensitive resistors would not work, it is recommended that any future team that is to complete this test stand should use a higher quality load cell. It would be preferable to use one that has an operating range of 0-5

N with a sensitivity of at least 0.01 N. It is also recommended that a motor assist be researched to aid the bird with overcoming the extra friction and weight that is present due to the test rig, such that thrust may be measured as well.

Stereo Vision. Another method in which we plan on testing is through the kinematics of the physical prototype. This will be accomplished with two cameras using stereo vision for flap cycles. Traces will be added to the end of the wing tips while two cameras are recording. Post-processing of the videos will occur in which we determine the wing tip trajectory. The kinematic data will then be compared to our design kinematics, as well as Liu's model [5].

Technology Readiness Level. Our robotic bird design received a TRL 6. We initially began researching flapping flight, the anatomy of birds, aerodynamics of flight, and many other topics. CFD simulations were then completed based on existing research to determine the amplitude of the motion for our four degrees of freedom. A physical prototype was then developed after the design kinematics for the CAD model were validated. Once the prototype was constructed, the actuation of the joints was controlled correctly and at the proper time. Currently we are working on testing our prototype under relevant conditions to determine the amount of lift produced.

Conclusion

Flapping flight has been studied by numerous groups, but has not been kinematically correct up to this point. Our project involved designing a fully articulated bio-fidelic robotic bird that produces positive lift for flapping flight. We initially began researching flapping flight, the anatomy of birds, aerodynamics of flight, and many other topics. Robotic protocols were then developed using the DH convention in order to model the four degrees of freedom about the bird's wing. CFD analyses were run in 2-D and 3-D to validate that nature was correct about one degree of freedom about the secondary portion of the wing. Also through case studies, the amplitude for each degree of freedom was optimized to best produce lift and marginal thrust.

With these results, a CAD model was then developed in which the weight of the bird was significantly reduced while still maintaining structural integrity. One important aspect throughout the entire design phase was that our kinematics were validated by Liu's model [5]. Before construction of our CAD model started, we first validated our design kinematics against Liu's model. While the construction of the physical prototype was underway, control designs were being implemented on the robotic bird. The main effort from the control designs was to actuate the backsweep motion and wing tip torsion correctly, as well as at the correct time during the flap cycle. After the design kinematics were validated, construction of our physical prototype began with ideas for a test rig that was able to measure lift and thrust. Our current testing structure comprises of a beam attached to lazy susan bearings such that the motion is circular about the base.

Through our project we have made many contributions to flapping flight. We developed a fully and correctly articulated bio-fidelic robotic bird that produces positive lift for flapping flight. Flapping flight kinematics were optimized through CFD simulations run in 2-D and 3-D case studies. A physical prototype was created in which we minimized the prototype weight while still maintain structural integrity.

References

[1] Hoepffner, J. and Y. Naka (2011) “Oblique Waves Lift the Flapping Flag.” Physical Review Letters 107(19): 194502.

[2] "S1223" *Airfoil Tools*. N.p., n.d. Web.<http://airfoiltools.com/airfoil/details?airfoil=s1223-il>.

[3] Fluent, Inc. (2006) *Fluent 6.3 UDF Manual*. Lebanon, New Hampshire.

[4] "2D NACA 0012 Airfoil Validation." *2D NACA 0012 Airfoil Validation*. N.p., n.d. Web. 8 Dec. 2013. <http://turbmodels.larc.nasa.gov/naca0012>.

[5] Liu, Tianshu, Shirley Jones, Kathryn Kuykendoll, and Ray Rhew. "Avian Wing Geometry and Kinematics." AIAA Journal 44.5 (2006): 954-963. Print.

[6] Maxon Motor, *1-Q-EC Amplifier DEC Module 24/2 Datasheet*, rev. May 2010 (Maxon Motor 2010), p. 100.

[7] CUI, Inc., *AMT20 Series Modular Encoder Datasheet*, rev. 1.08 (Tualatin, OR: CUI, Inc. 2013), p. 100.

[8] Atmel Corporation, *Atmel ATmega640/V-1280/V-2560/V-2561/V Datasheet*, rev. 2549Q-AVR-02/2014 (San Jose, CA: Atmel Corporation 2014), p. 100.

[9] LSI Computer Systems, Inc., *LS7366R 32-bit Quadrature Counter with Serial Interface Datasheet* (Melville, NY: LSI Computer Systems, Inc. 2007), p. 100.

[10] Digi International, Inc., *Product Manual v1.xEx – 802.15.4 Protocol* (Minnetonka, MN: Digi International Inc. 2012), p. 100.

APPENDIX A: Primary Wing Case Study Results

Table A-1
Shown are the specific parameters used for the primary wing case study

Case	Velocity (m/s)	Φ_2 (horizontal motion) [Degrees]	Ψ_2 (vertical motion) [Degrees]	θ_2 (angle of incidence) [Degrees]
C7_15	7	15	38	15
C10_15	10	15	38	15
C7_30	7	30	38	15
C10_30	10	30	38	15
C7_45	7	45	38	15
C10_45	10	45	38	15
C7_55	7	55	38	15
C10_55	10	55	38	15
C7_15B	7	15	38	30
C10_15C	10	15	38	30
C7_30B	7	30	38	30
C10_30C	10	30	38	30
C7_45B	7	45	38	30
C10_45C	10	45	38	30
C7_55B	7	55	38	30
C10_55C	10	55	38	30
C7_15E	7	15	38	45
C10_15F	10	15	38	45
C7_30E	7	30	38	45
C10_30F	10	30	38	45
C7_45E	7	45	38	45
C10_45F	10	45	38	45
C7_55E	7	55	38	45
C10_55F	10	55	38	45

Appendix B: Hardware Schematic

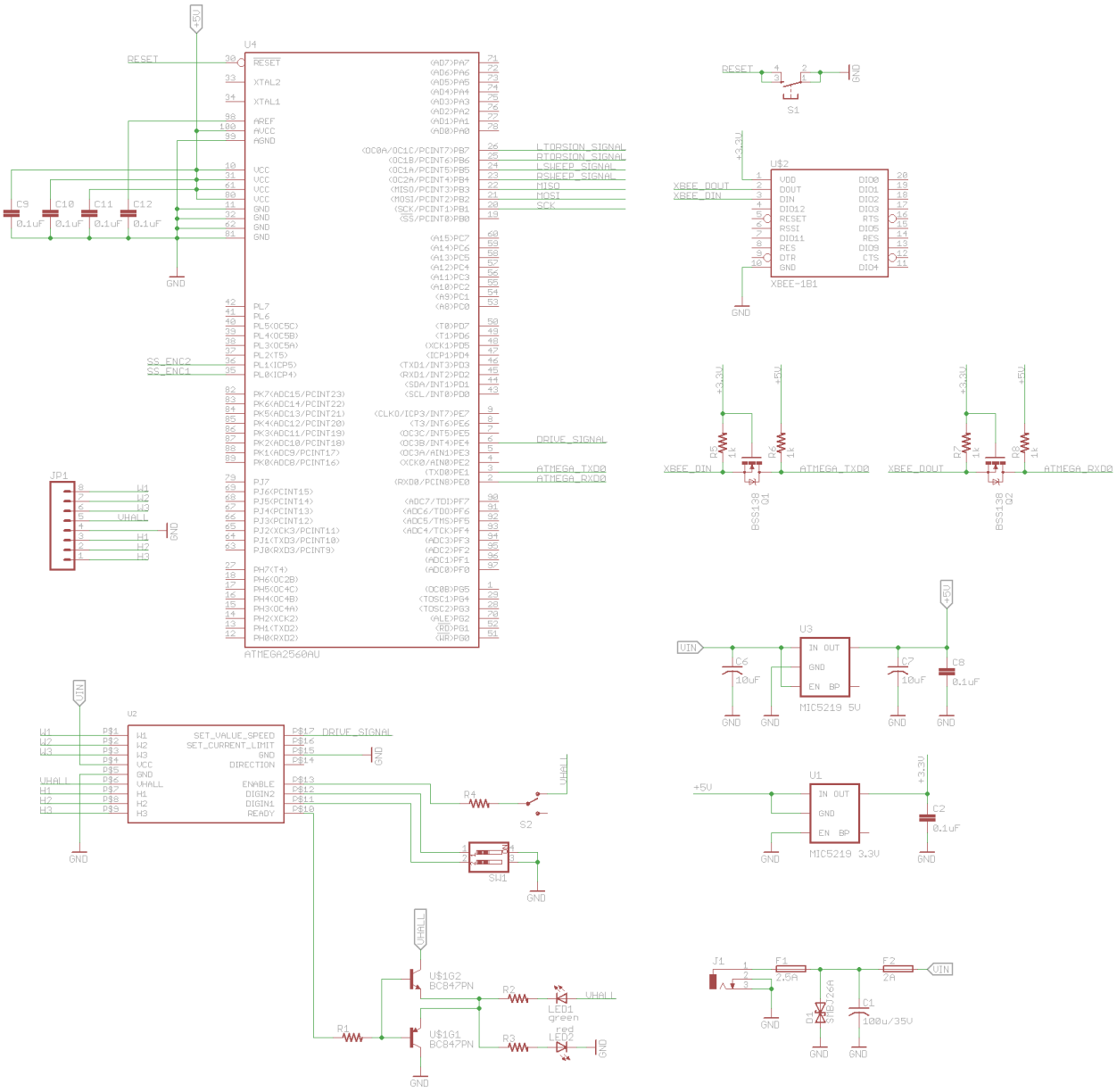


Figure B-1. Hardware schematic for the ATmega2560 control and data collection circuit.

Appendix C: Pseudo Code for the Software Developed

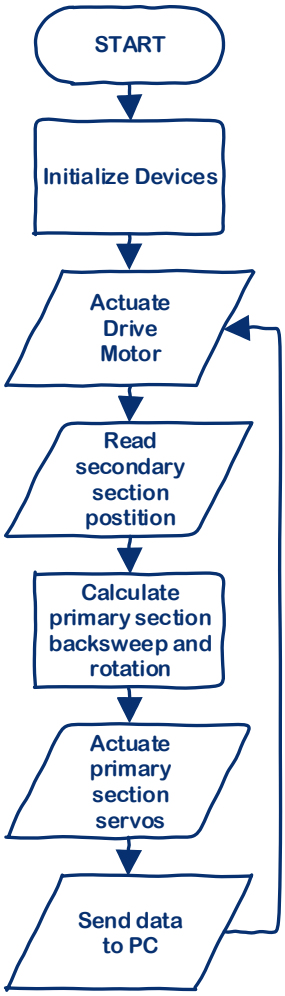


Figure C-1. Flowchart for ATmega2560 data processing.

Appendix D: Program Listing

```
//=======================================
// VT Robotic Bird Driver Code
// AUTHOR: Brandon Galbraith
// DATE: May 7, 2014
// Rev. 4.02
//=======================================
// Hardware:
// Arduino Mega 2560 (ATmega2560)
// CUI AMT 203 Modular Encoder
// LS7366R Quadrature Counter
// Maxon DEC Module 24/2
// Maxon EC16 Brushless DC Motor
// Turnigy TGY-90S Micro Servo
//=======================================

#include <Servo.h>
#include <SPI.h>

#define rd_pos 0x10
#define nop_a5 0x00
#define set_zero_point 0x70

uint16_t ABSposition = 0;
uint8_t temp[1];

long packet;
float deg = 0.00;

// Declare servos
Servo leftWingServo;
Servo rightWingServo;
Servo leftWristServo;
Servo rightWristServo;

// Define pins for DEC Module
const int digIN1 = ;    // pin[] <<digIN1>>
const int digIN2 = ;    // pin[] <<dinIN2>>
const int setValueSpeed = ; // pin[] <<Set value speed>>
const int enablePin = 2;   // pin[] <<Enable>>

// Define pins for backsweep and wingtip rotation servos
const int leftWingServoPin = ; // PH3(OC4A)
const int rightWingServoPin = ; // PH4(OC4B)
const int leftBacksweepServoPin = ; // PH5(OC4C)
const int rightBacksweepServoPin = ; // PH6(OC2B)
```

```
// Slave select pins SPI devices
const int slaveSelectEnc1 = ; // S1 on LS7366 Breakout Board
const int slaveSelectEnc2 = ; // S2 on LS7366 Breakout Board
const int slaveSelectCUI = ; // AMT203 CSB Pin

// Hold current encoder count
unsigned int encoder1count = 0;
unsigned int encoder2count = 0;
unsigned int modEncoderCount;

// Hold angle values
float nonDimensionalTime;
float plungeAngle;
float torsionAngle;

// Timing Variables
unsigned long startTime = 0;
unsigned long elapsedTime;

// Motor Control Variables
int motorSpeed = 150;
boolean readyState = true;
int measuredSpeed;

void setup() {
 // Initialize serial communications
 Serial.begin(115200); // USB communication to serial terminal

 initCUI(); // Initialize CUI Modular Encoder

 //initEncoders(); // Initialize LS7366R Quadrature Counter
 //clearEncoderCount(1);
 //clearEncoderCount(2);

 initDECModule(); // Initialize Maxon Motor Controller

 setZeroCUI(); // Set absolute zero position of the AMT203

 // Mount servos
 leftWingServo.attach(leftWingServoPin);
 rightWingServo.attach(rightWingServoPin);
 leftWristServo.attach(leftBacksweepServoPin);
 rightWristServo.attach(rightBacksweepServoPin);

}

void loop() {
```

```
//==================================
// Actuation of the Main Drive Motor
//==================================

// Check for motor fault condition
if(readyState == false) {
 motorSpeed = 0;
}

// Set speed
analogWrite(setValueSpeed, motorSpeed);

//=====================================================
// Positional Measurement of the Secondary Wing Section
//=====================================================

// Read secondary section position
uint8_t recieved = ;
ABSposition = 0;

SPI.begin();
digitalWrite(slaveSelectCUI, LOW);

SPI_T(rd_pos);

recieved = SPI_T(nop_a5);

while (recieved != rd_pos) {
 recieved = SPI_T(nop_a5);
 delay(2);
}

temp[0] = SPI_T(nop_a5);
temp[1] = SPI_T(nop_a5);

digitalWrite(slaveSelectCUI, HIGH);
SPI.end();

temp[0] &=~ 0xF0;

ABSposition = temp[0] << 8;
ABSposition += temp[1];

deg = -(ABSposition * 0.08789 - 360);

//=================================================================
// Control of the Backsweep and Wing Tip Rotation of Primary Section
//=================================================================
```

```
// Calculate Required Wingtip Rotation
torsionAngle = calcTorsionAngle(deg);

// Execute wingtip rotation
rightWingServo.write(torsionAngle);
leftWingServo.write(-(torsionAngle-180));

// Execute backsweep movement
rightWristServo.write( (int)(23.5*sin(deg*3.1415/180) + 23.5) );
leftWristServo.write( (int)(175 - (23.5*sin(deg*3.1415/180) + 23.5)) );

//================================
// Quadrature Counter Speed Control
//================================

//================================
// Serial communications to LabVIEW
//================================

//packet = (int)torsionAngle*100 << 16;
//packet |= (int)deg*100;
int intDeg = deg*100;
int intTorsion = torsionAngle*100;

// output readings to serial port
Serial.write(0x);
Serial.write(0x);
Serial.write(highByte(intTorsion));
Serial.write(lowByte(intTorsion));
Serial.write(highByte(intDeg));
Serial.write(lowByte(intDeg));
Serial.write(highByte((int)readyState));
Serial.write(lowByte((int)readyState));

delay(10);

}

//================================
// Initialize DEC Module Subroutine
//================================
void initDECModule() {

// DEC Module inputs
pinMode(digIN1, OUTPUT);
pinMode(digIN2, OUTPUT);
pinMode(setValueSpeed, OUTPUT);
```

```
 pinMode(enablePin, OUTPUT);

 // Enable 500...5000 rpm speed range
 digitalWrite(digIN1, HIGH);
 digitalWrite(digIN2, LOW);

 // Enable the power stage
 digitalWrite(enablePin, HIGH);
}

//=========================================
// Initialize CUI Modular Encoder Subroutine
//=========================================
void initCUI() {
 pinMode(slaveSelectCUI, OUTPUT);
 digitalWrite(slaveSelectCUI, HIGH);
 SPI.begin();
 SPI.setBitOrder(MSBFIRST);
 SPI.setDataMode(SPI_MODE0);
 SPI.setClockDivider(SPI_CLOCK_DIV32);
 SPI.end();
}

//================================================
// Initialize LS7366R Quadrature Counter Subroutine
//================================================
void initEncoders() {

 // set slave selects as outputs
 pinMode(slaveSelectEnc1, OUTPUT); // S1 on LS7366R Breakout Board
 pinMode(slaveSelectEnc2, OUTPUT); // S2 on LS7366R Breakout Board

 // Raise select pins
 digitalWrite(slaveSelectEnc1, HIGH);
 digitalWrite(slaveSelectEnc2, HIGH);

 // Set encoder 1 to non-quad modulo-n count mode
 digitalWrite(slaveSelectEnc1, LOW);
 SPI.transfer(0x); // WR MDR0
 SPI.transfer(0x); // non-quad modulo-n count mode
 digitalWrite(slaveSelectEnc1, HIGH);

 // Set encoder 1 to 1-byte counter mode
 digitalWrite(slaveSelectEnc1, LOW);
 SPI.transfer(0x); // WR MDR1
 SPI.transfer(0x); // 1-byte counter mode
 digitalWrite(slaveSelectEnc1, HIGH);
```

```
 // Set encoder 1 to modulo-15 mode
 digitalWrite(slaveSelectEnc1, LOW);
 SPI.transfer(0x); // WR DTR
 SPI.transfer(0x); // Set n=15
 digitalWrite(slaveSelectEnc1, HIGH);

 // Set encoder 2 to non-quad free-running mode
 digitalWrite(slaveSelectEnc2, LOW);
 SPI.transfer(0x); // WR MDR0
 SPI.transfer(0x); // non-quad free running count mode
 digitalWrite(slaveSelectEnc2, HIGH);

 // Set encoder 2 to 1-byte counter mode
 digitalWrite(slaveSelectEnc2, LOW);
 SPI.transfer(0x); // WR MDR1
 SPI.transfer(0x); // 1-byte counter mode
 digitalWrite(slaveSelectEnc2, HIGH);
}

//======================================
// Clear LS7366R Encoder Count Subroutine
//======================================
void clearEncoderCount(int encoder) {

 // Set encoder 1 data register to 0
 if (encoder == 1) {
  digitalWrite(slaveSelectEnc1, LOW);
  SPI.transfer(0x); // CLR CNTR
  digitalWrite(slaveSelectEnc1, HIGH);
 }

 // Set encoder 2 data register to 0
 if (encoder == 2) {
  digitalWrite(slaveSelectEnc2, LOW);
  SPI.transfer(0x); // CLR CNTR
  digitalWrite(slaveSelectEnc2, HIGH);
 }
}

//======================================
// Read LS7366R Encoder Count Subroutine
//======================================
int readEncoder(int encoder) {

 SPI.begin();
 // Initialize temporary variables for SPI read
 int count;
```

```
 // Read encoder 1
 if (encoder == 1) {
  digitalWrite(slaveSelectEnc1, LOW);
  SPI.transfer(0x); // Transfer CNTR to OTR, then output OTR serially on TXD(MISO)
  count = SPI.transfer(0x);
  digitalWrite(slaveSelectEnc1, HIGH);
 }

 // Read encoder 2
 if (encoder == 2) {
  digitalWrite(slaveSelectEnc2, LOW);
  SPI.transfer(0x); // Transfer CNTR to OTR, then output OTR serially on TXD(MISO)
  count = SPI.transfer(0x);
  digitalWrite(slaveSelectEnc2, HIGH);
 }
 SPI.end();
 return count;
}

//==================================
// Serial Event Interrupt Subroutine
//==================================
void serialEvent() {
 while (Serial.available()) {
  motorSpeed = Serial.read();
 }
}

//=======================================================
// Calculate Primary Section Wingtip Rotation Subroutine
//=======================================================
int calcTorsionAngle(float angle) {
 float torsionAngle;

 if(angle < 140) {
  torsionAngle = 30;
 }
 else if (angle >= 140 && angle < 195) {
  torsionAngle = 0.000381*angle*angle*angle - 0.191*angle*angle + \
  31.2*angle - 1650 + 15;
 }
 else if (angle > 173 && angle < 291) {
  torsionAngle = 0;
 }
 else {
  torsionAngle = -.000187angle*angle*angle + 0.183*angle*angle - \
  59.0*angle + 6270+15;
 }
```

```
 return (int)torsionAngle;
}

//====================================
// SPI Data Transfer to CUI Subroutine
//====================================
uint8_t SPI_T (uint8_t msg) {
 uint8_t msg_temp = 0;
 digitalWrite(slaveSelectCUI, LOW);
 msg_temp = SPI.transfer(msg);
 digitalWrite(slaveSelectCUI, HIGH);
 return(msg_temp);
}

//==============================
// CUI Set Zero Point Subroutine
//==============================
void setZeroCUI() {

 int8_t idleChar = 0x;

 SPI.begin();
 digitalWrite(slaveSelectCUI, LOW);

 idleChar = SPI_T(set_zero_point);

 while (idleChar != 0xFFFFFF80) {
  idleChar = SPI_T(nop_a5);
  delay(20); // mfg recommended 20us gap between reads
 }

 digitalWrite(slaveSelectCUI, HIGH);
 SPI.end();
}
```

Appendix E: Code Revision Summary

VT Robotic Bird Driver Code Revision Summary
Author: Brandon Galbraith
Hardware:
Arduino Mega 2560

Rev. 2.00b
April 2, 2014
*Added servo control for LH wingtip torsion

Rev. 3.00b
April 3, 2014
*Added code for LS7366R Quadrature Counter
*Added call function for initialization of DEC Module
*Added ISR for fault condition from DEC Module

Rev. 3.01b
April 5, 2014
*Revised code comments
*Added serial output of encoder count from LS7366
*Fixed break statement error in fault condition ISR
*Motor speed is reduced to zero on fault condition
*User receives serial message to manually reset Arduino on fault condition

Rev. 3.02b
April 17, 2014
*Added quad counter 2 to measure actual motor speed
*Change quad counter 1 to 1-byte counter mode to increase processing speed
*Change quad counter 1 to modulo-n mode to derive wing plunge angle
*Added servo control for RH wingtip torsion
*Calculate wingtip torsion angle as a function of plunge angle
*Limited motor speed to 500...5000 rpm
*Output measured motor speed to the serial port
*Output plunge angle to the serial port
*Output wingtip torsion angle to the serial port

Rev. 3.03b
April 19, 2014
*Added ISR for a serial event input from LabVIEW which controls motor speed
*Added serial output of ready state of DEC module
*Changed serial output of readings to hexadecimal format

Rev. 3.04b
April 20, 2014
--> Need to increase baud rate to 115200
--> Need to change wingtip torsion to subroutine
--> Problem uploading motor speed
--> Problem with data resolution
--> need to change plungeAngle to a float (or other data type)

Rev. 3.05
April 23, 2014
*Moved eqn for wingtip torsion to a subroutine
*Change torsionAngle to float data type

Rev. 4.00
April 30, 2014
*Added CUI Modular Encoder
*Revised torsionAngle calculation based on absolute zero position
*Revised problem with negative torsion angles

Rev. 4.01
May 1, 2014
*Removed readyLED ouput from code (hardware controlled)
*Changed pin assignment for servo attachments
*Fixed SPI communications with CUI Mod Encoder
*Added serial output of nonDimTime, rotationAngle, and readyState

Rev. 4.02
May 7, 2014
*Connected <<Ready>> from DEC to Reset pin on Arduino and removed faultInterrupt() ISR

Rev. 4.03
May 8, 2014
*Corrected an error in the calcTorsionAngle in which the case structures were inconsistent, causing a flutter in the wingtip rotation