# PROJECT BIRD BRAIN:

## DEVELOPMENT OF AN AVIONICS PACKAGE FOR A ROBOTIC ORNITHOPTER

JACK MONACO ‘22

SUBMITTED TO THE
DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING
PRINCETON UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF
UNDERGRADUATE INDEPENDENT WORK

FINAL REPORT

APRIL 23, 2022

PROFESSOR AIMY WISSA
PROFESSOR ANIRUDHA MAJDUMDAR
MAE 440
57 PAGES
FILE COPY

This thesis represents my own work in accordance with University regulations.

# Abstract

Small UAVs have many civilian and military applications. Flapping wing UAVs combine multiple modes of flight, enabling desirable performance in cruising conditions as well as low Reynolds number flight. Flapping UAVs are also safer and quieter than fixed wing and rotary wing equivalents but have been the subject of less research. In order to further equip an open-loop robotic ornithopter for dynamics analysis and the implementation of control, an on-board avionics package was designed to collect data on orientation, accelerations, tail position, and wing position during flight. The system was designed to integrate with existing hardware while providing a platform for expansion to different forms of communication and control. The package was fabricated, and recorded data was validated and calibrated using tests performed in a motion capture environment. Analysis revealed highly accurate orientation estimation in aircraft pitch and roll, while yaw estimation suffered from magnetic variability in the testing environment. Although further model development is necessary to make use of results, the devised solutions for sensing tail and wing position were shown to provide accurate and useful data. The functional capabilities of the package make it an excellent platform for further examination of the in-flight dynamics experienced by the ornithopter and for future efforts to implement control.

# Acknowledgments

I must start by giving glory to Jesus, from whom and by whom I have received all things, including the head on my shoulders, my love for engineering, and the privilege of attending this amazing institution for the past four years.

I would then like to express my gratitude to everyone who has taught me, loved me, pushed me, and given of their time and energy to help me become a better human and engineer during my time at Princeton. Thank you to my Advisor, Professor Aimy Wissa for her assistance, humor, and encouragement throughout all portions of this project. Thanks also to Paul Lee for his mentorship and to all the members of the Bio-inspired Adaptive Morphology lab for their thoughtful contributions and support during this semester of research.

This project would not have been possible without the generous and frequent assistance of Jonathan Prevost, as well as the wealth of information that lives in his brain. I cannot thank Jon enough for his constant willingness to help.

A very special thank you to my dad, Peter Monaco, for his prayers, his cheerful encouragement, and his delightful eagerness to jump on a Zoom call to help me debug my code.

Thanks also to John Porter from VICON for his assistance and his MATLAB scripts, and finally, thank you to Warren for the generous donation of his micro-USB cable.

# Table of Contents

# List of Figures

# Chapter 1: Introduction

## 1.1 Background on Flapping Wing UAVs

Small unmanned aerial vehicles, or small UAVs have growing applicability in both civilian and military fields. Due to their small size, relatively low cost, and autonomous operation, small UAVs are natural replacements for existing methods of surveillance, crop monitoring, disaster response, search and rescue, and environmental data collection. Small UAVs can be classified into one of three categories: fixed wing, rotary wing, and flapping wing UAVs [1]. Flapping UAVs, or ornithopters, take inspiration from avian flight. Ornithopters combine the flapping mode of flight, which often has a high cost of transport but offers advantages at lower speeds, with gliding, which has a low cost of transport. Some of the primary features distinguishing flapping from fixed wing and rotary wing UAVs are maneuverability, low Reynolds number flight performance, safety, and volume [1]. Small fixed wing UAVs suffer from high drag penalties when flying in low Reynolds number flight regimes, and also lack agility. Rotary UAVs, such as drones, can hover easily and perform well at slow flight speeds, but they are noisy, power-intensive, and are not as efficient at cruising speeds when compared to fixed wing aircraft. The ability of an ornithopter to glide takes advantage of some of the efficiency of fixed wing design, while flapping enables better low Reynolds number performance by increasing the local Reynolds number, among other effects [2]. Like birds, ornithopters have the potential to hover, make high agility maneuvers, and cruise using natural air currents. Flapping UAVs have been the subject of less research and development than their fixed wing and rotary counterparts, but this combination of features makes ornithopters a relevant area of study and robotic development for further expansion of the applications of small UAVs.

## 1.2 Research Platform

This project centers on Shadow, a robotic ornithopter provided by Dr. Aimy Wissa, as a research platform for studying avian flight dynamics and characteristics. In the past, this ornithopter design has been used by Professor Wissa to examine the effects of implementing a passively compliant mechanism into the leading edge of the wings to improve steady-level flight performance [1].

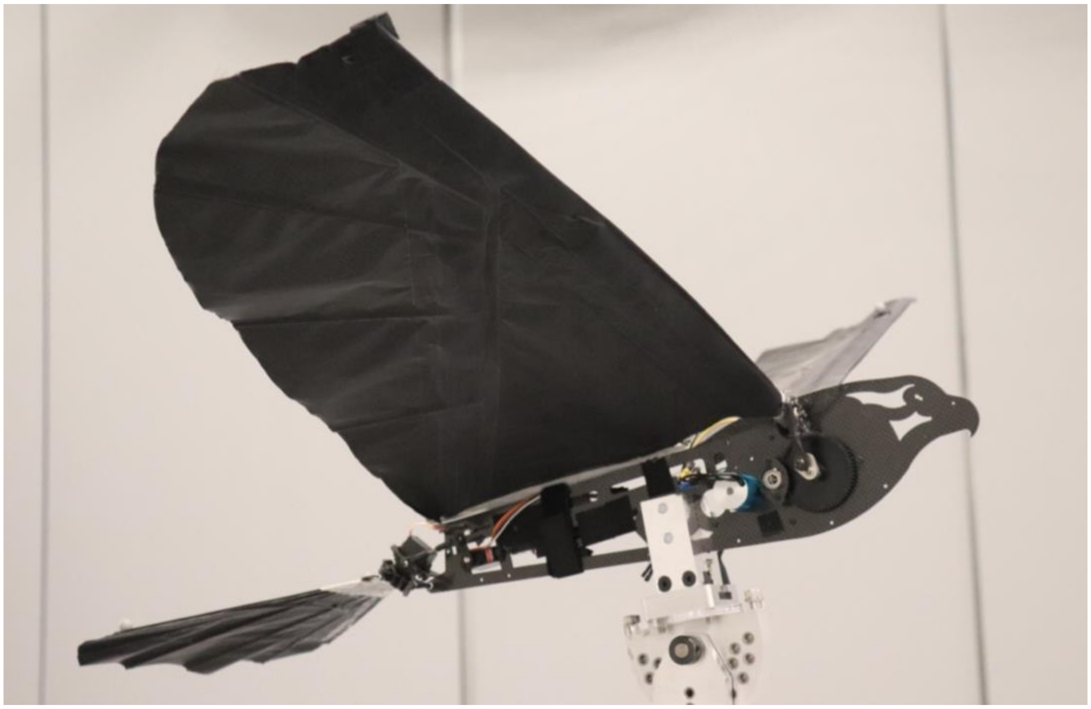


Figure 1.1: Shadow the Ornithopter

The ornithopter body is constructed of carbon fiber, and the wings are a combination of nylon ripstop, nylon tape, Dacron, and carbon fiber rods [3]. The leading edge of each wing is a 4mm carbon fiber rod that is fixed into a shoulder joint with a collet nut and the wingspan is also fixed to the length of the body in two locations. Flapping motion is generated with a two-stage gearbox that sits beneath the shoulder joints. A 4-pole brushless motor with a 9-tooth geared pinion turns an 84-tooth gear, whose 6-tooth shaft turns a second 64-tooth gear, resulting in an 896 to 9 gearing ratio. Mounted on each end of the shaft of the second larger gear is an eccentric standoff that connects to the shoulder joint via two swivel ball joints (figure 1.2), translating rotational motion into sinusoidal motion of the shoulder joint.

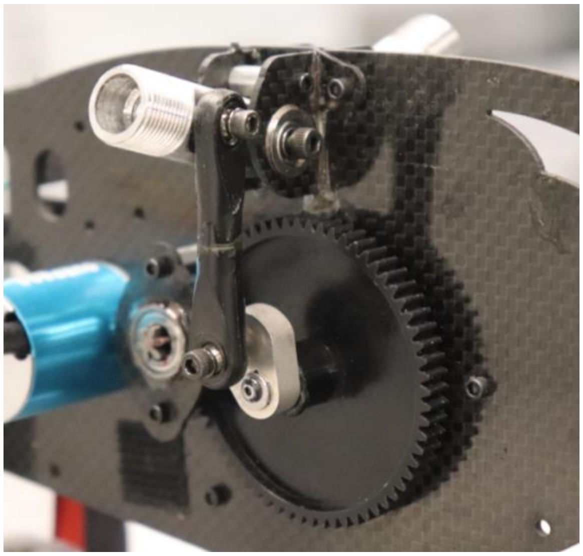


Figure 1.2: Shoulder joint and standoff mechanism, without wing attached

The tail of the ornithopter is also constructed of nylon, Dacron, and carbon fiber rods. Its position is controlled by two servos, one mounted on the body that adjusts pitch via a mechanism similar to a four-bar linkage, and one mounted on the rocker of the four-bar linkage that adjusts yaw (figure 2.3). For a full detailed report on the ornithopter construction and mechanics, see the report referenced in appendix A.

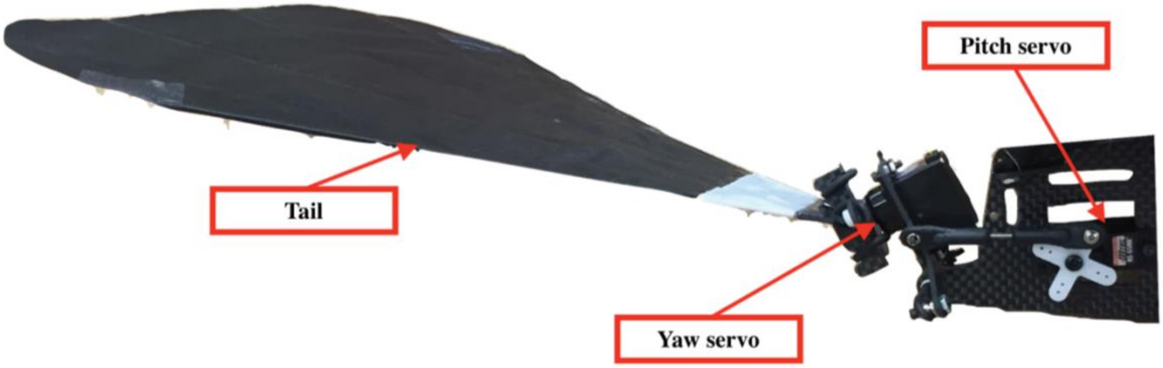


Figure 1.3: Ornithopter tail mechanism [3]

The ornithopter electronics system is open loop, and essentially functions the same as a remote-controlled model airplane. Signals from a remote control are receiver by a 2.4 GHz radio receiver which outputs pulse-width-modulation signals (PWM) to each of the two tail servos, as well as to a 30 Amp electronic speed controller (ESC) which in turn controls the brushless motor driving the wings. The system is powered by a 11.1 Volt, 1300 mAh, 3-cell Lithium Polymer battery, The receiver is powered via a 5V connection to the ESC and the servos are in turn powered through the receiver. A full list of parts can be found in appendix A.

## 1.3 Project Focus

The goal of this project is to equip Shadow with a central processor and a suite of sensors, also referred to here as an avionics package, to enable in-flight data collection, as well as provide a platform for dynamics research and for the development and implementation of feedback control on the ornithopter.

In order to expand the capabilities of this robotic platform for further investigation of avian inspired flight dynamics, a reliable solution for collecting in-flight data is necessary. Such a platform would enable the development of an in-flight dynamics model, which is required in order to develop a control system for the robotic platform. In addition to sensing motor position and orientation, this project will also provide a platform for controlling the actuators without need for a human operator to send instructions via a transmitter. For experimental purposes, this will enable the conduction of controlled, repeatable tests, and in order to implement a control system in the future, both sensing and actuation capabilities are required.

## 1.4 Expansion on Existing Work

The development of an avionics package for this ornithopter was begun by Kevin Murphy, an Undergraduate at the University of Illinois Urbana-Champaign. The main contributions of this project included research into part selection, development of a CAD model for the ornithopter, outfitting of a magnetic encoder on first gear shaft, and development of models relating encoder position and control inputs to wing and tail positions [4]. These contributions will be discussed in more detail in further sections.

While the previous project made good headway on understanding the demands and limitations of the avionics package, it remained in a prototyping phase, and lacked several of the capabilities desired of the final avionics package. First, the system was only equipped to collect data and sample motor positions and was therefore not equipped to also handle control of the actuators. This would have prohibited the platform from being used for controlled, repeatable testing and from expansion to implement control. Another small shortcoming of the system was that it lacked a method for determining servo position in any way other than sampling the servo input signal, which, as will be shown in later sections, is an insufficient indicator of position. Finally, the package had many separate components and loose wires, and it lacked a clean formfactor to optimize for space, weight, and ease of use.

In order to expand on this work and produce a robust final product, the objectives of this project are as follows:

- Successfully intercept the ornithopter electronics system in a way that retains the capability for control by a human operator while providing a platform for expansion to control.
- Devise solutions for collecting in-flight data on servo positions, orientation, accelerations, and wing position, to be stored with on-board removable storage.
- Develop a clean, self-contained, and easy to work with package with minimal separate components or loose wires. Design sustainable mounting solutions for all components.
- Confirm the validity of collected data by comparison with motion capture data
- Experiment with wireless communication for data transfer, calibration, and debugging.

## 1.5 Report Format

In the following chapter (Chapter 2) the design and fabrication of the avionics package is described in detail with references to existing work and comments on the selected methods. In Chapter 3, tests are presented to confirm the reliability of the package, and Chapter 4 outlines the conclusions and implications of this project, as well as suggestions for further development.

This report and the attached appendices also aim to provide the necessary background and technical details necessary for future collaborators to easily continue working with this ornithopter and avionics package.

# Chapter 2: Avionics Package Design

## 2.1 Big Picture of the Brain

The avionics package, affectionately referred to as the brain of the ornithopter, will be responsible for handling a number of tasks, which inform component selection and design. The required functions of the brain are listed below:

- Read PWM signals from the radio receiver and transmit PWM signals to actuators.
- Read and process data from an inertial measurement unit (IMU)
- Take readings from analog sensors, such as internal servo potentiometers and current sensors
- Track the position of one or more rotary encoders
- Store sensor data to an SD card at fixed time intervals
- Transmit data and debugging information wirelessly

The figure below depicts the signal flow of the required tasks handled by the microcontroller.

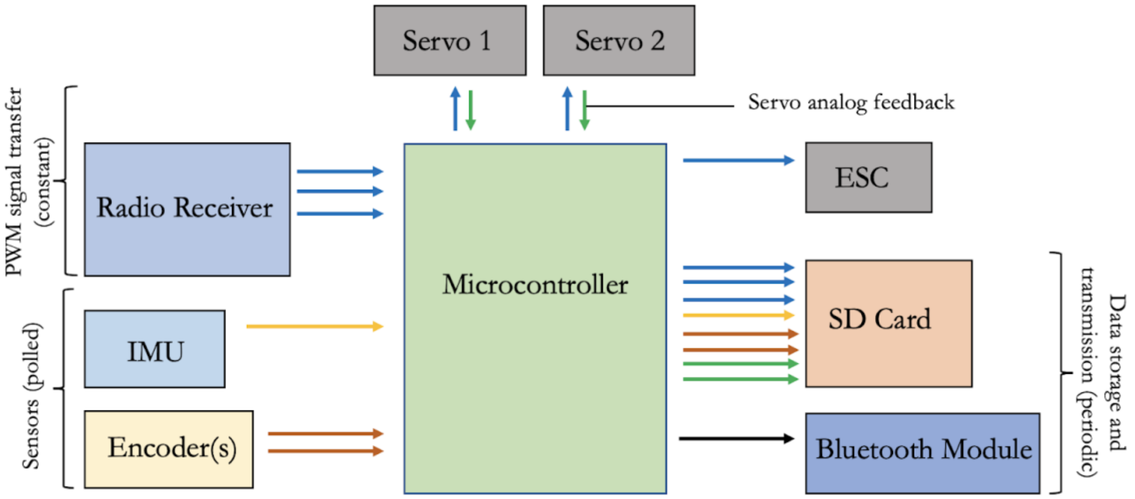


Figure 2.1: Summary of inputs and outputs to be managed by the microcontroller

## 2.2 Microcontroller Selection

The design of the brain centers heavily on the selection of microcontroller to use as the central processor for the package. In the first attempt to outfit the ornithopter with a sensor suite, an Arduino Uno was used as the microcontroller. The Arduino was selected for its capability to interface with I2C and SPI devices, as well as its easy-to-use platform and IDE. However, this system relied on a separate encoder buffer board to count the steps of the incremental encoder, as well as a separate data logging device to interface with an SD card. Beyond that, while the Arduino Uno is a good prototyping tool, its bulky form factor, limited pin availability, and low clock speed (16MHz) make it less ideal for a permanent installment on the ornithopter.

For this project, a Teensy 4.1 was selected as the microcontroller of choice. Teensy is a line of microcontrollers developed by Paul Stoffregen of PJRC which is highly regarded in the hobby electronics community for its wide range of capability and high processing power. The Teensy 4.1 uses an ARM Cortex-M7 processor and runs at a clock speed of 600MHz, leaving most of its competitor processors in the digital dust [5]. Using the Teensy ensures that the processing and timing demands of future onboard operations will be met. The Teensy is priced between $27 and $40 depending on the vendor and if the pins are pre-soldered, and its small form factor (2.4 by 0.7 inches) makes it an ideal selection for a small UAV.

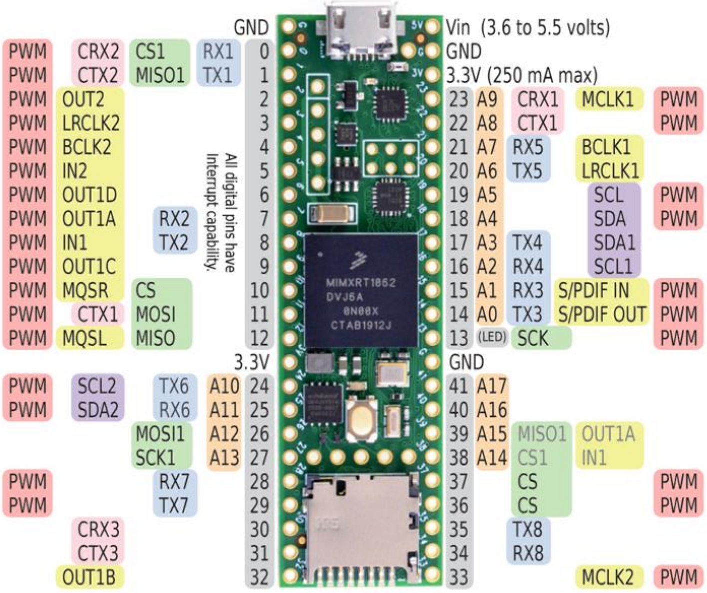


Figure 2.2: Teensy 4.1 pinout diagram [5]

With regard to pinout requirements, the avionics package developed in this project makes use of three analog inputs, eight digital inputs and outputs, and three pulse width modulation outputs, in addition to interfacing with up to five devices through a combination of I2C, UART, and SPI communication. This combination of demands would have necessitated a larger Arduino Board, such as the Arduino Mega, but are easily met with the 55 configurable input/output pins on the Teensy. Another highly desirable feature of the Teensy 4.1 is the onboard microSD card slot, eliminating the need for an external data logger and decreasing the overall number of components.

Programming and interfacing with Teensy microcontrollers are also virtually the same as with Arduino controllers. An extension is available from PJRC that enables the Teensy to be programmed and communicated with through the Arduino IDE. In addition to compatibility with all Arduino libraries, there also exists a large selection of libraries optimized for Teensy microcontrollers. See appendix C for details on how to install and use the Teensyduino extension.

The main design constraint introduced by using the Teensy 4.1 is that, while powered by a 5 Volt power supply, the microcontroller used 3.3 Volt logic, meaning that it is not compatible with 5V inputs and outputs. In this project, this proved to not be a major constraint, but it did require the sourcing of 3.3V compatible sensors, and the final circuit board includes two separate power circuits. Another important way in which the Teensy differs from Arduino products is that it cannot be powered by both the USB cable and a 5V power source simultaneously. To avoid damaging the board, a trace on the back of the teensy must be cut to power the board via battery and use the USB cable simultaneously (see appendix B for details). The Teensy 4.1 also lacks native wireless capabilities, such as those available on Wi-Fi-compatible controllers, such as the ESP32, however its other features, especially the built in microSD slot, make it the best choice for this project.

## 2.3 Monitoring PWM Input

One of the primary tasks of the avionic package is to read, record, and pass along incoming signals from the ornithopter's radio receiver, in order to enable both onboard control and piloting by a human operator.

The control input from the radio transmitter consists of three values, corresponding to yaw, pitch, and throttle. The combined signal is received and decoded by the radio receiver, which in turn sends the three values to the servos and ESC in the form of pulse width modulation signals. Pulse width modulation, or PWM is a basic form of digital communication where a square pulse is sent with a constant period and the value being sent corresponds to the width of the pulse, or the percentage of the duty cycle it fills (figure 2.3). In typical model aircraft communication, pulses range from 1 to 2 milliseconds and occur on a 20 ms period.

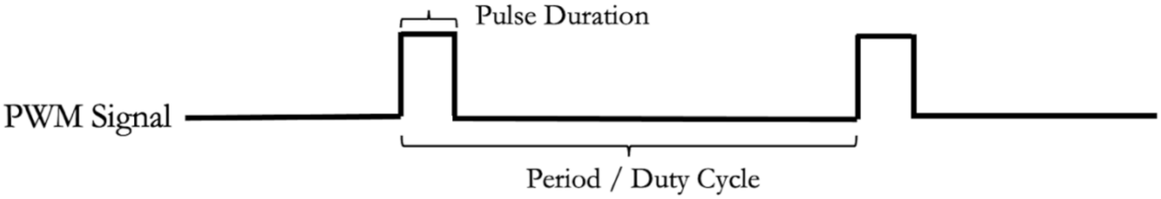


Figure 2.3: Simple pulse width modulation signal

In the previous avionics package, pulse widths were sampled by connecting the controller to each of the servo PWM lines and using interrupt routines to look for rising and falling edges to time the duration of each pulse. In an effort to simplify code and make use of optimized libraries, a different method was attempted for reading the PWM values in this version of the project.

FreqMeasureMulti() is a Teensy library for detecting the frequencies of multiple input signals, but it can be configured to return the duration of high pulses. The library uses software interrupts to poll the lines in question to detect rising and falling edges. The library only works on pins enabled with "FlexPWM" and it stores readings in a buffer, which must be read out to get the most recent recording [5]. Using FreqMeasureMulti() objects to track the pulse widths in the background worked nicely at first, but gradually resulted in several problems. On occasion the function would return an extremely high value out of the range of expected pulse durations. This would throw both servos to their maximum positions and, if the motor was connected, would result in the ornithopter attempting to flap at full speed, potentially damaging the mechanism. After investigation, this problem was traced to the library's use of interrupts.

Interrupts, while optimal for programs that require a large number of short, immediate operations, such as the monitoring of an incremental encoder, are generally dangerous to use in combination with larger programs and tend to result in bugs. Specifically, an expensive operation, such as a serial write or the opening and closing of a file from an SD card can often be disrupted by hardware interrupts, so interrupts are disabled during the operation. However, if the interrupt condition occurs during the operation, the interrupt bit will still be flipped, resulting in the interrupt routine being executed after interrupts are re-enabled [5]. In interrupt routines that relate to precise timing operations, or for interrupt conditions that may occur multiple times during the prioritized operation, this disabling of interrupts may seriously affect the performance of the interrupt routine. In the case of FreqMeasureMulti(), interrupt disabling resulted in inaccurate timing, and ultimately the library's method of constantly polling to look for signal edges was not an efficient way of timing high pulses.

As a result of these findings, a new method of pulse timing that did not rely on interrupt usage was pursued.

One option briefly experimented with was to take 20 milliseconds after each data recording and poll each of the three input lines continuously while counting the number of high recordings. At the end of this period the number of highs out of total samples could be mapped to a duration of the 20ms period. This method failed to account for the fact that the clock of the microcontroller is not synchronous to the clock of the radio receiver, as well as the fact that the period of the PWM signal is not perfectly uniform, and also tends to be closer to 21 milliseconds. Because the start of polling operation was not synchronized with frequency of the pulses, the polling would periodically begin mid-pulse, resulting in a failure to capture the correct pulse duration and sending one to two inaccurate readings to the actuator. Figure 2.4 shows this effect over time.

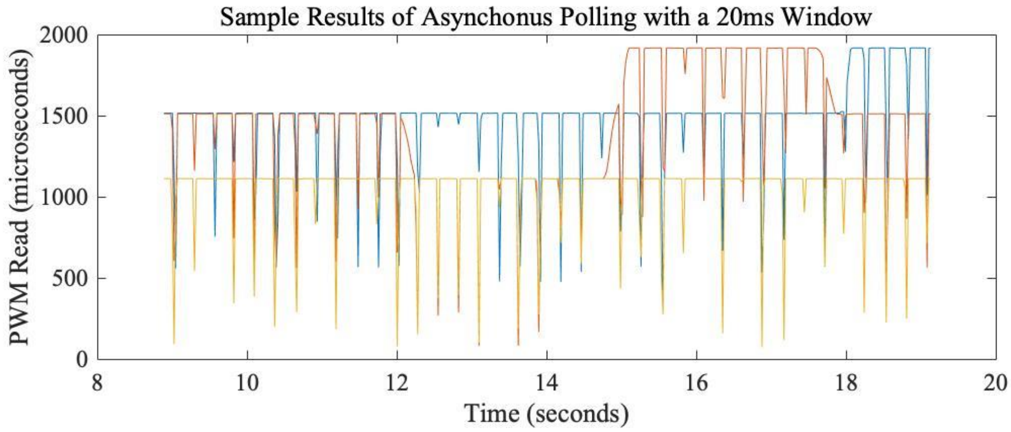


Figure 2.4: Three PWM inputs being timed with asynchronous polling

The method that ultimately resulted in successful pulse timing takes advantage of the pulse sequence and their predictable width ranges. Figure E.1 in appendix E shows the three pulses tracked using a digital logic analyzer, and the structure of this sequence is also represented in figure 2.5. Depending on the values being sent, the order in which the pulses arrive varies, but they are always clumped together at the start of the duty cycle. The tracking method for the final avionics package is as follows: within the main loop of the program, the processor checks on the status of several buttons, records a sampling of data, and then spends the remainder of the loop handling the PWM signal. Upon startup, this section calls a method named alignPWM() which simply looks for a high signal on any of the three input lines and then records the time at which the first high appears, returning if none are found in the allotted search window. Once a pulse is found, the method readWritePWM() is called on subsequent loops. This method uses the previous pulse start time to predict at what time the next group of pulses will arrive, and strategically polls the lines during the expected window. The function returns once the starts and stops of all three pulses have been detected and timed, and also records the start time of the first pulse to inform the next search window. In this way, the main loop is synchronized with the duty cycle of the radio receiver, and the pulses are timed efficiently and accurately in about six milliseconds, leaving the remaining fifteen milliseconds of the period for other operations such as sensor reading and data recording. Figure 2.5 shows the timing layout of the operations of the main loop, and the full Teensy/Arduino code can be found in appendix D.

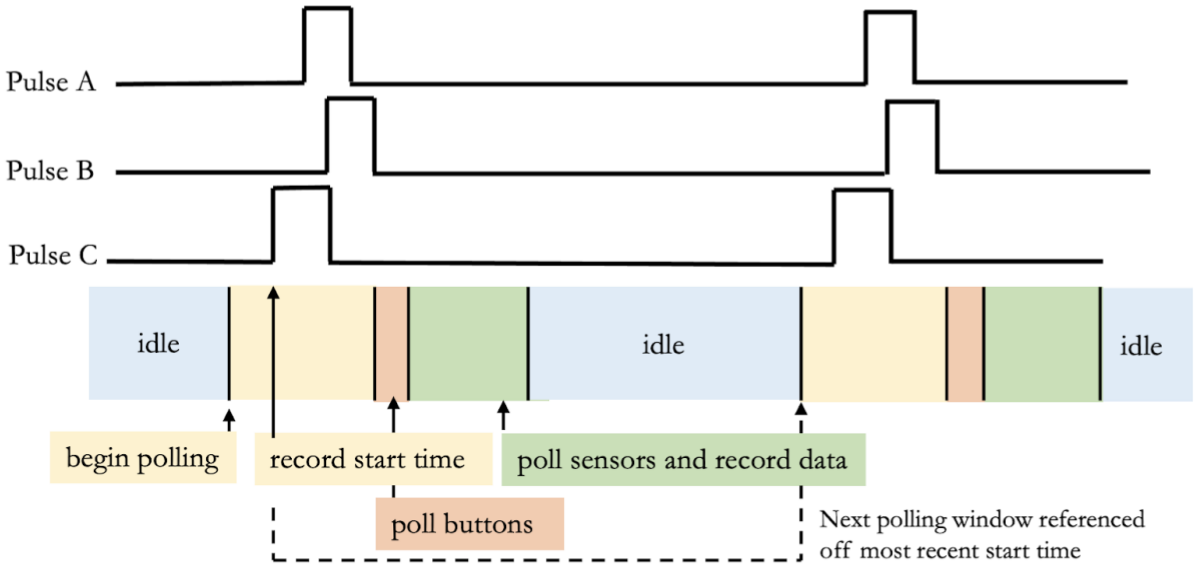

Pulse A
Pulse B
Pulse C
idle
idle
idle
begin polling
record start time
poll sensors and record data
poll buttons
Next polling window referenced off most recent start time

Figure 2.5: Timing layout of synchronous programming approach

This method for pulse timing relies on the fact that the pulses are clean and there is no chance of stray high signals, which would throw off the timing operation. This proves to be a safe assumption as the PWM signals are internally debounced by the receiver. By fitting sensor reading and data recording, which generally only takes 1-2 milliseconds, in between the pulses, data is reliably recorded every 21ms without inhibiting flight instructions.

This simple task of timing and relaying pulses proved to be one of the most challenging problems of the project, and while a robust solution was arrived at, it is important to note that the existing system could be improved by adopting a different form of radio communication. This solution was created to interface with the existing hardware, and although the microcontroller still has plenty of time to perform all the necessary operations, transitioning to a better form of communication would free up a significant amount of processor time and enable data sampling and recording on custom intervals.

## 2.4 Shaft Encoder

An important function of the avionics package is to report information on flapping rate and wing angle. In previous work done by Kevin Murphy, this was accomplished by outfitting the first gear shaft with a magnetic encoder and developing a relationship between the shaft angle and wing angle. The selection of a magnetic encoder for this application is a very good choice. Magnetic encoders work by placing a diametrical magnet (magnetized in the radial direction) in front of a sensor that uses multiple Hall elements to detect changes in magnetic flux in different directions. Through a combination of these readings, the sensor can detect the position of the magnet at a high resolution [6]. Magnetic encoders are advantageous for use in small devices that require fast and accurate angle readings because of their extremely compact size, compatibility with high RPM, and completely frictionless operation. In the previous project, a diametrical magnet had been outfitted into the first shaft of the gearbox and a magnetic encoder, an RLS R4MK encoder chip was mounted on a custom breakout board in a small 3D printed fixture attached to the ornithopter with hot glue (figure 2.6). The encoder used an incremental output, whose steps were counted using an encoder buffer board, which the Arduino sampled using SPI communication. In order to improve this solution for use in the final avionics package, the first two objectives were to update the encoder to a standard, replaceable part, and to design a sustainable mounting solution.

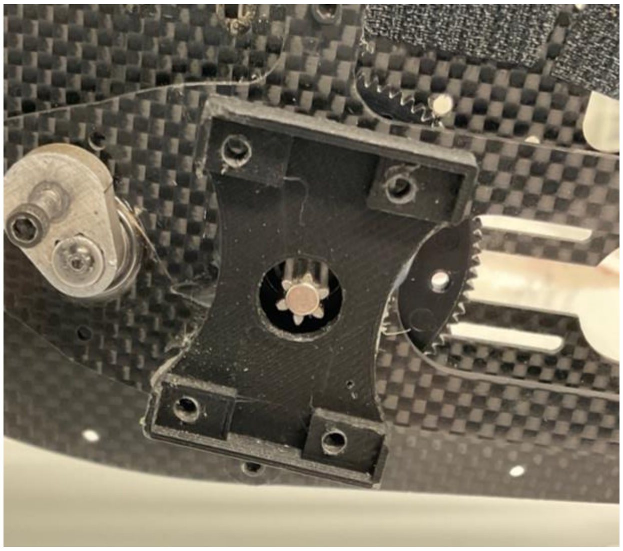


Figure 2.6: Previous encoder mount and gear shaft with imbedded magnet

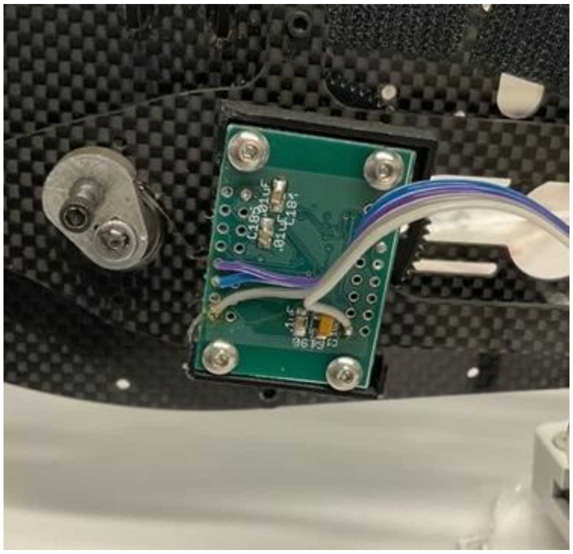


Figure 2.7: Encoder board in mount

Sourcing from RLS, the original manufacturer of the IC encoder proved to be difficult due to shipping delays, so a different brand of encoder magnetic encoder was sourced. The final part used in this iteration of the project was the AS5047D, a magnetic encoder chip from AMS, compatible with the magnets sourced in the previous project. This was mounted on a breakout board sourced from Digi-Key (see appendix B for full parts list). In addition to being an inexpensive and reliably source-able part, this encoder has multiple output types, including a normal incremental output as well as an angular output readable through PWM or SPI communication. These alternative output formats were desirable when designing a system with only one central processor. Incremental (AB or ABZ) encoders typically rely on high volume interrupt usage to track position movements, which, as detailed in the previous section, can significantly disrupt simultaneous operations, and can be disrupted themselves by operations that disable, or pause interrupt usage. In the previous project an encoder buffer was used to solve this problem, but in this project, to minimize extra components, the SPI output of the encoder was taken advantage of.

SPI, or serial peripheral interface, is a form of data transfer commonly used in communication between microcontrollers and basic integrated circuits, as it is simple and faster than UART or I2C communication. SPI uses four lines: a chip-select (CSn) and a clock line (CLK) coming from the microcontroller, a master-out-slave-in (MOSI) line, and a master-in-slave-out line (MISO). Each communication starts by pulling the chip select line low and then transfers a one- or two-byte instruction with an associated address. Each data transfer simultaneously sends and receives data, and the instruction sent in one transfer is executed on the next. For instance, to read the value in a given register on the chip, a read instruction with that address is sent in one transfer, and the following transfer receives the requested data while sending the next instruction. The figure below shows the SPI schematic of a sequence of read-only commands.

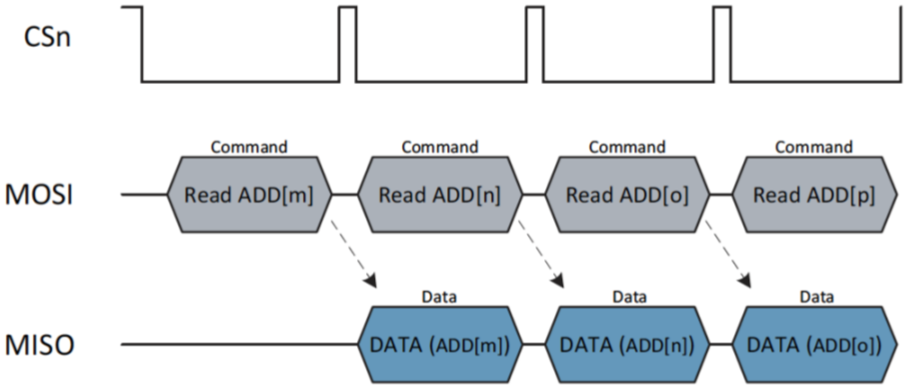


Figure 2.8: Simple sequence of read commands through SPI [7]

In the case of the AS5047D, the command to read the address that stores the 14-bit instantaneous angle measurement is, in hexadecimal, FFFF, which is the same as if the line was just tied high. Therefore, if the only desired action is to read this angle, the MOSI line can simply be tied high, and every SPI transfer after the first one will return the value in the angle register. On the final circuit board for the project, both options—full MOSI/MISO communication and read-only operation—were wired for (see appendix B for details). By using SPI to read the encoder, the need for a buffer was eliminated and a precise angle measurement was available at any time.

To mount the encoder on the first gear shaft, a 3D-printed mount was designed and printed at 60% fill density. This mount makes use of the existing hardware that mounts the port-side plate to the ornithopter body, enabling easy removal and durable attachment. The board is positioned at a relief angle of 15 degrees to allow more room for the shoulder joint and standoff to move without the danger of colliding with the mount, and the encoder is fixed to the mount with M2 bolts and nuts (figures 2.9, 2.10, and 2.11).

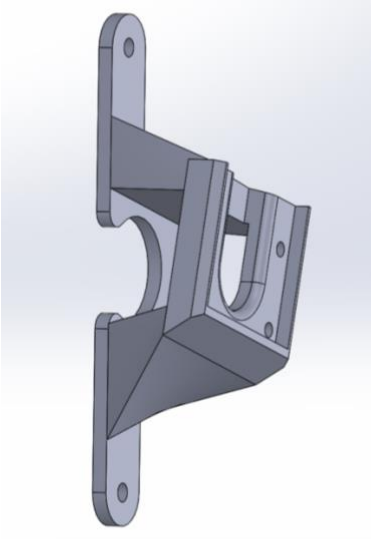


Figure 2.9: New encoder mount model

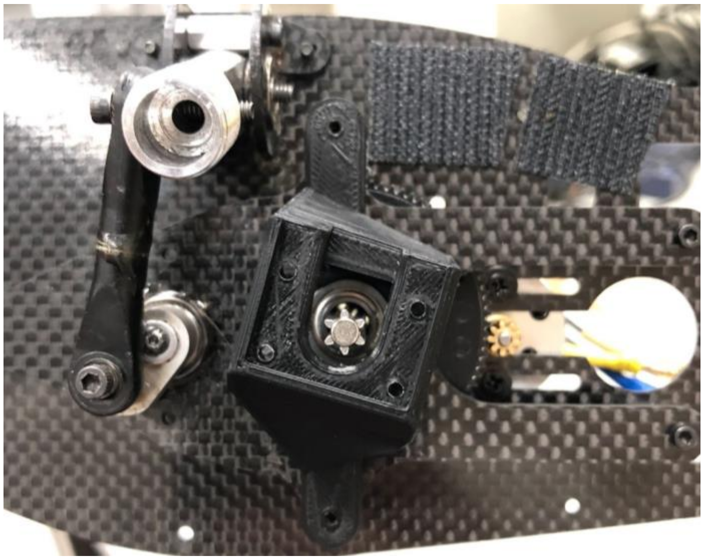


Figure 2.10: Mount positioned on shaft with imbedded magnet

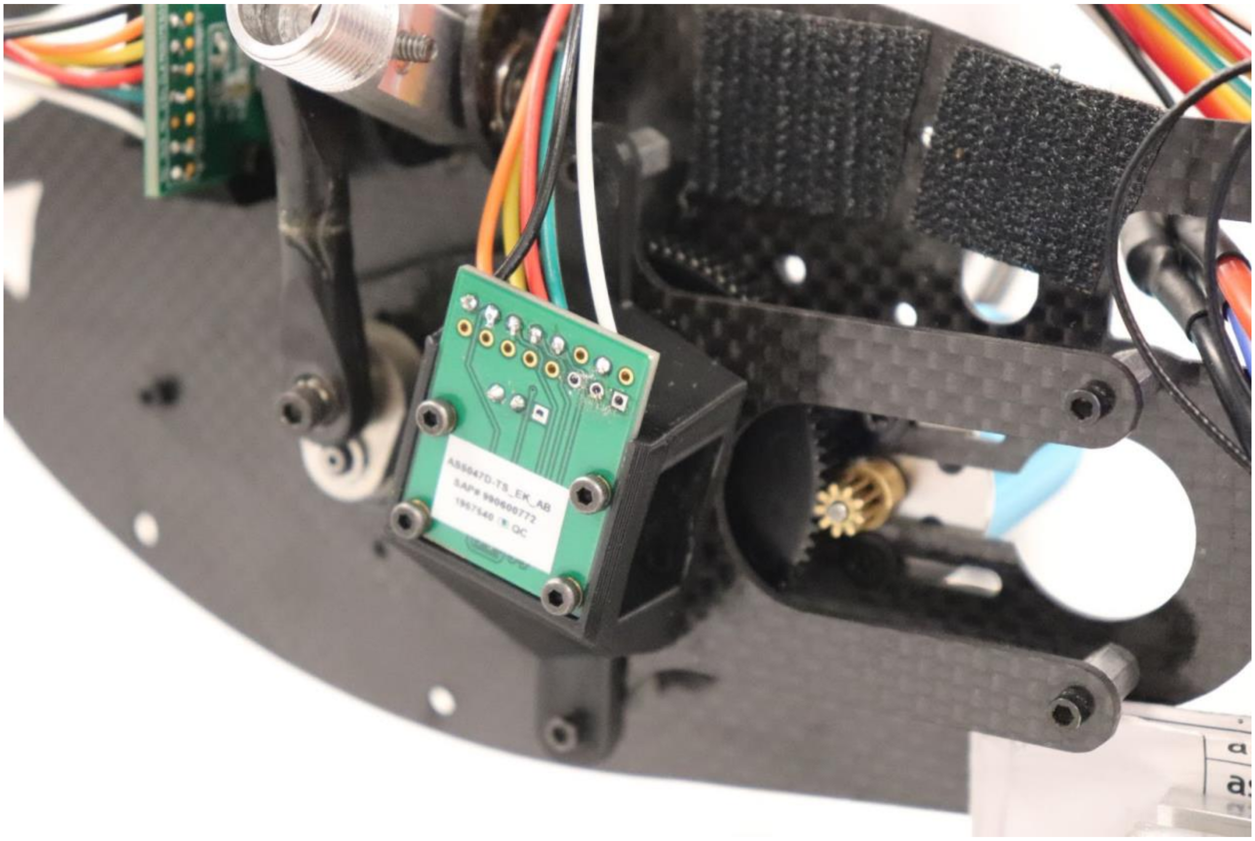


Figure 2.11: Encoder mounted on ornithopter gear shaft

## 2.5 Shoulder Encoder

In the previous project, a CAD model was used to model the relationship between the shaft rotation and wing position. While the shaft encoder provides a very accurate measure of motor speed, from which flapping frequency can be calculated, it is an indirect measure of wing position for several reasons. First, the wing angle cannot be predicted based on the angle of the first gear shaft upon power-up due to the gearing ratio. Additionally, due to play in the standoff mechanism and the loads experienced by the wings while flapping, the movement of the shoulder joint cannot be perfectly predicted using the position of either gear shaft. In response to this deficiency, an exploration was begun into the possibility of mounting a second encoder on the shoulder joint.

The shoulder joint of the ornithopter consists of a bearing and a machined shoulder piece that receives the collet nut for the carbon fiber rod. The bearing and shoulder are connected with a bolt that threads through the shoulder and rides in the bearing (see figure 1.2 in introduction). As an experiment, this bolt was threaded in from the tail-side of the joint and a small 3D-printed magnet housing was designed to thread onto the protruding end of the bolt (figure 2.12). A simple mounting piece that clamps onto the ornithopter body and holds an AS5047D board in front of the magnet was modeled and printed at 40% infill (Figure 2.13). The shoulder encoder was wired for SPI communication in the same manner as the shaft encoder. SPI communication with a device is only enabled when the chip select pin is pulled low, therefore, it is possible for both encoders to share the same MOSI/MISO/CLK lines as long as each has its own chip select (see wiring diagram in appendix B for more details).

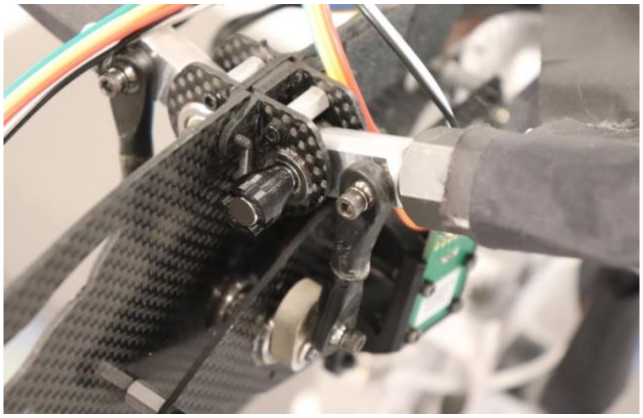


Figure 2.12: Magnet holder threaded onto end of bolt

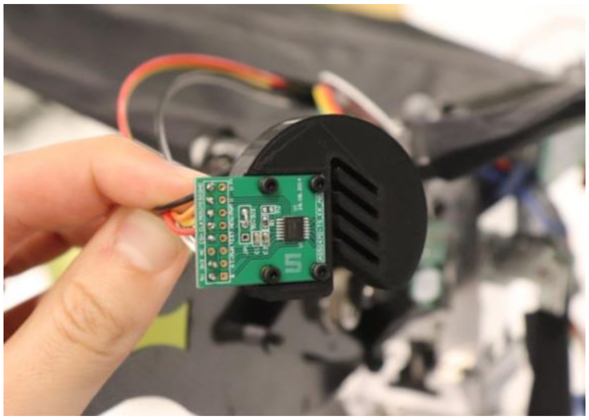


Figure 2.13: AS5047D board installed in mount

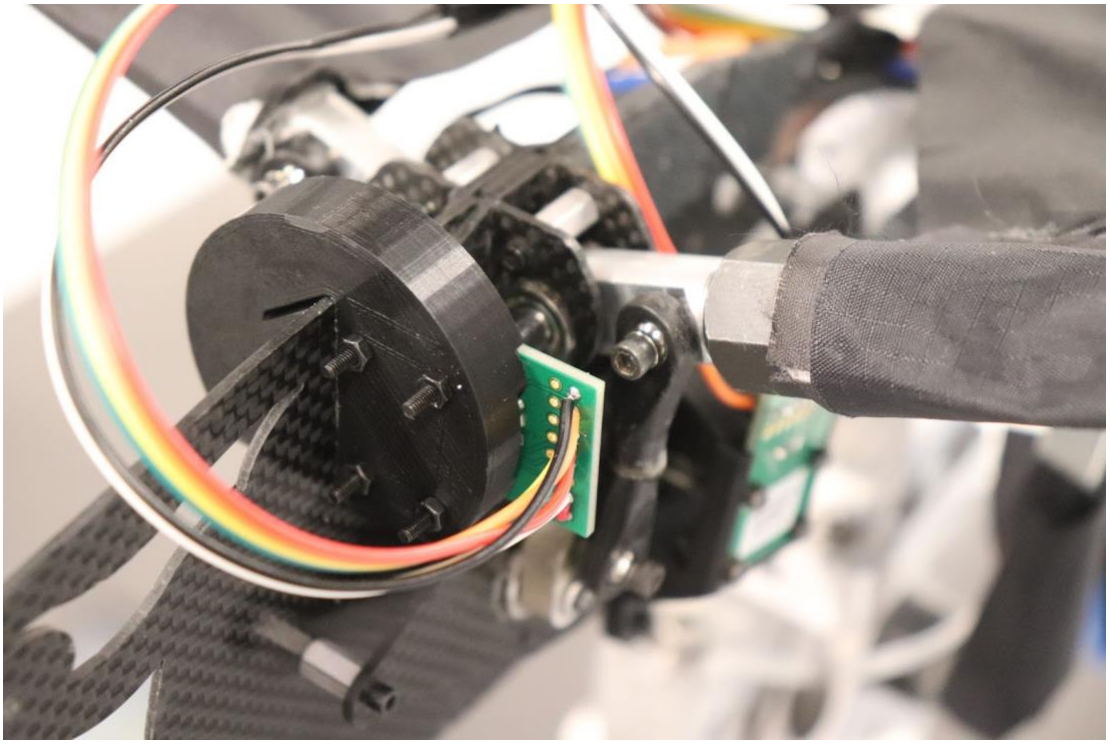


Figure 2.14: Shoulder encoder and magnet assembled and in place

Readings from this encoder configuration are shown in Chapter 3.3. This solution is not a long term one, as it relies on friction to keep the magnet and bolt rotating in sync with the shoulder joint, but it demonstrates the usefulness and effectiveness of a shoulder mounted encoder. As will be discussed in the results, a simple modification to the shoulder joint could enable this to be a lasting position sensing solution.

## 2.6 Inertial Measurement Unit

A primary aim of any avionics package is to be able to sense changes in acceleration and orientation. For this purpose, an inertial measurement unit, or IMU was included in the package. An IMU is a sensor that includes three-axis accelerometers, gyroscopes, and magnetometers, from which linear and angular accelerations, as well as heading with respect to the earth's magnetic field, can be sensed. The process of combining these readings to produce an estimate of absolute orientation is called sensor fusion. In brief, the gravity vector calculated from the accelerometer is used to reduce two degrees of freedom, and the magnetic field direction detected by the magnetometer is used to align the axes with respect to the earth. Because the estimate of the earth's magnetic field is affected by the hard iron and soft iron elements in the sensor's surroundings, calibration is necessary to sense absolute north. Gyroscopes are added to the configuration to filter out the effects of non-gravity accelerations on the accelerometer [8].

For this project, the BNO055 inertial measurement unit from Bosch was selected, mounted on a board supplied from Adafruit Electronics. The BNO055 includes built-in sensor fusion and calibration models, and Adafruit provides an extensive Arduino library for easy integration of the sensor. An in-depth discussion of the process of interfacing with the IMU and verifying orientation readings is provided in the data verification section. For information on how to calibrate the IMU, which is required after every power-up, see appendix C.

## 2.7 Wireless Communication

To aid in the useability of the package, a simple Bluetooth module was added to the package to aid in debugging and data transfer. The HC-06 Bluetooth module is initialized using the Arduino SoftwareSerial library and written to as a normal UART (universal asynchronous receiver/ transmitter) serial port. For information on how to connect to the Bluetooth device and view outputs in the Arduino IDE serial monitor, see appendix C.

In its current form, the avionics package makes use of the Bluetooth module in a number of applications. When entering calibration mode for the IMU, calibration status and sample data are monitored via Bluetooth to allow for the IMU to be calibrated without being tethered to the computer. The Bluetooth monitor is also used to print error messages and to transfer data without removing the SD card from the package. While Bluetooth is a useful communication method for these purposes, it is limited in its usability for real time communication. Printing via serial communication is notoriously time consuming, and so repeated usages of the Bluetooth module are not sustainable while also monitoring PWM and recording data. Additionally, Bluetooth communication does not have a real-time serial protocol, which means this form of communication couldn't be used for instantaneous data processing or feedback.

## 2.8 Servos and Other Hardware Updates

As described in Section 1.2, the tail mechanism of the ornithopter is controlled with two servos that take PWM signals as input. Servos provide reliable position control over a small range of motion by using an internal potentiometer and gear box to correct position via an internal feedback loop. As described in section 2.3, the previous avionics package conflated control input with the actual position of the servo arm, failing to account for the tracking error between input and output. To solve this problem, both servos were replaced with analog feedback servos, which function identically, but also provide a direct line to the servo's internal potentiometer which can be sampled for a more accurate reading of position.

The analog outputs of the servos were stepped down from 5V range to 3.3V range using voltage dividers, each with one 10kΩ and one 20kΩ resistor. However, analog readings during testing subsequently indicated that voltage readings from the potentiometers were spanning a lower voltage range than expected. Further testing has revealed that the analog output of each servo takes values between 0.25 and 2.49 volts, indicating that the voltage dividers are not necessary, and removing them (by replacing the 10kΩ resistors with jumpers) would result in a higher resolution signal. If this modification is attempted in future work, see appendix B for instructions.

Several small 3D-printed parts were developed to aid in mounting the servos securely. In addition to servo replacement, the ESC and motor of the ornithopter were both replaced with readily source-able parts to eliminate bottlenecks in the future, should either need replacing. The leads and battery connection on the ESC were shortened and replaced to cut weight. See appendix A for more information on existing electronics components.

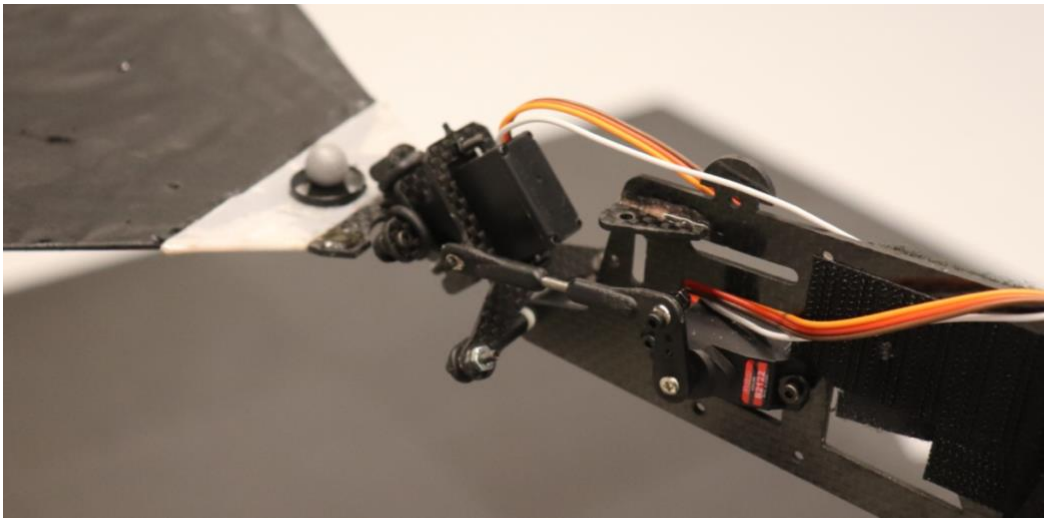


Figure 2.15: Analog feedback servos installed in tail mechanism

## 2.9 Final Package

One of the goals of the project was to produce a robust final package that was clean and easy to use. In the process of prototyping and testing, a circuit board was designed to house the microcontroller, IMU, Bluetooth module, and other necessary components. The final package is shown in figure 2.16. The single-sided board was designed in Eagle CAD and milled on a Bantam desktop mill out of an FR-1 circuit board blank. See Appendix B for more information on the board design, as well as links to the wiring diagram, board schematic, and Eagle board file. In addition to the components already discussed, the final board includes three push-buttons to put the microcontroller into recording mode, IMU calibration mode, or data transfer mode. Two LEDs are also available for use to indicate the mode of operation. In addition to pins for connection to the ESC, servos, radio receiver, and encoders, there is also an extra 3 pin header for an analog current sensor, to be implemented in a future project. The microcontroller and IMU are mounted using female headers and the back of the board is insulated using Kapton tape. Figure 2.16 shows all components of the package wired into the main circuit board, and figure 2.17 shows the board in positioned on the ornithopter, currently held in place with Velcro.

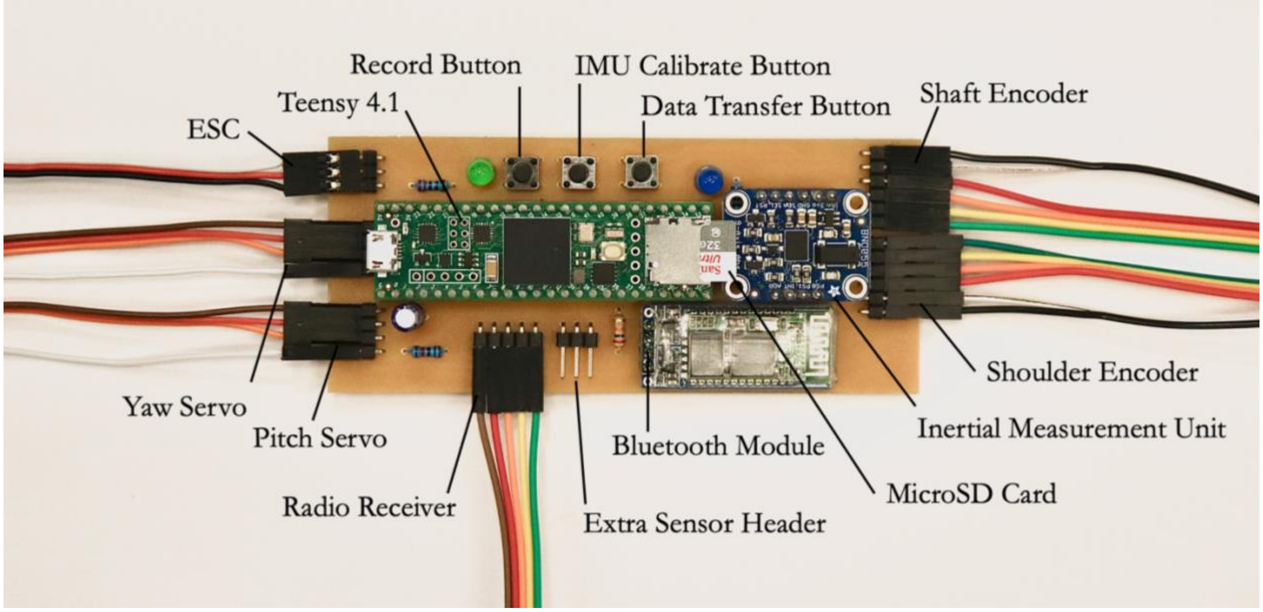


Figure 2.16: Final board with components and connections labeled

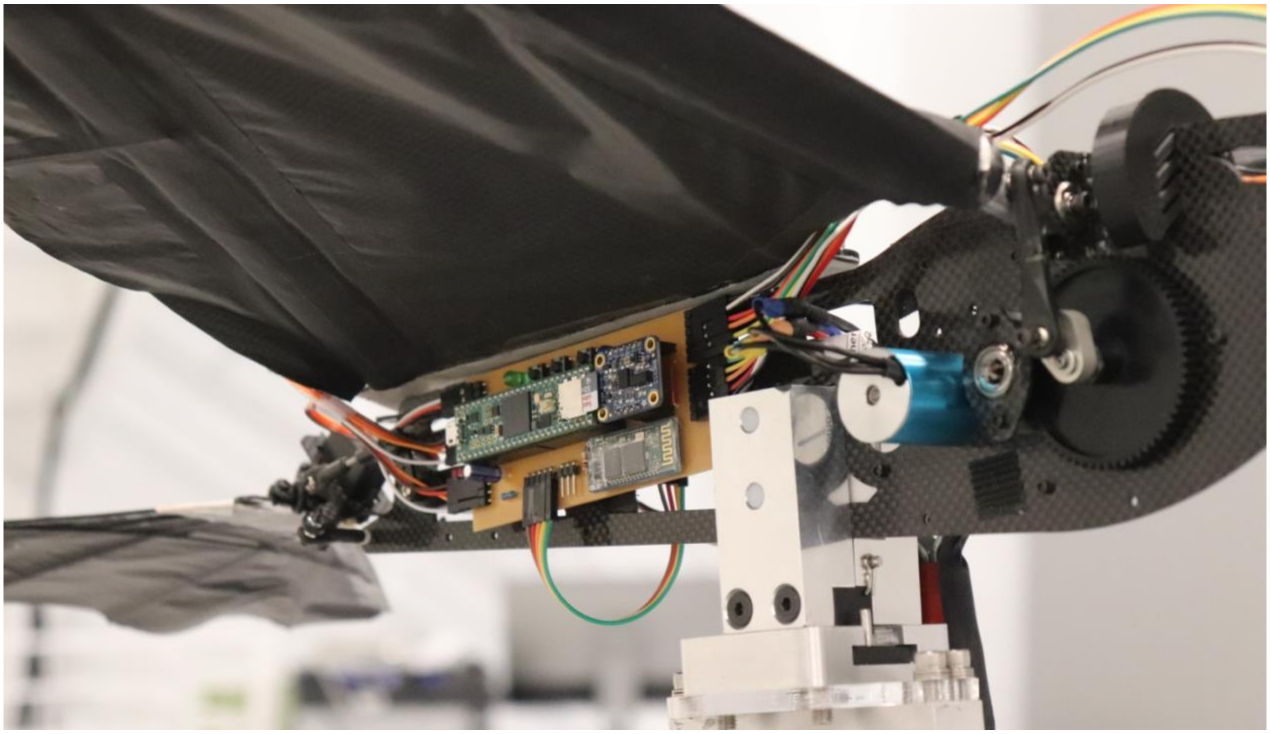


Figure 2.17: Avionics package mounted on ornithopter

# Chapter 3: Testing and Verification

## 3.1 Orientation - IMU Verification

In order to test the validity of output of the inertial measurement unit, data was collected from the avionics package while tracking its motion in a motion capture system (system provider: VICON). Specifically, the sensor's estimation of absolute orientation was tested, as it is the most relevant to the operation of a UAV, and trajectory cannot be calculated from accelerations without an accurate orientation.

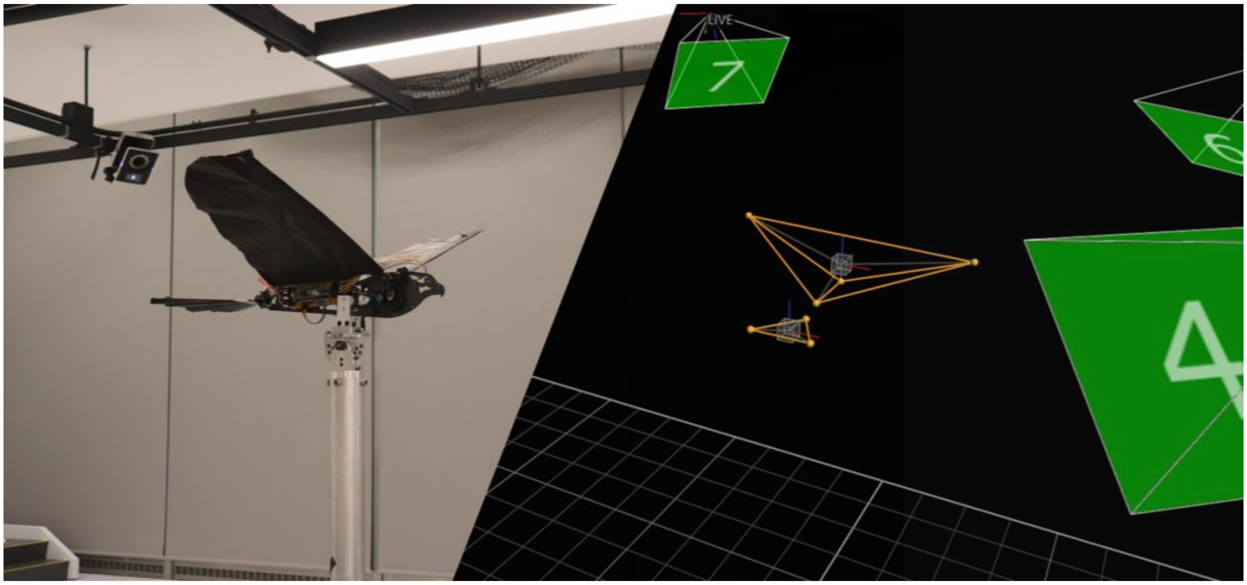


Figure 3.1: Experimental setup tracked in VICON virtual environment

There are several methods for representing orientation in 3D space. In this data verification process, orientations were represented in yaw, pitch, and roll. When using the standard convention for aircraft body coordinate axes (shown in figure 3.2), these correspond to Euler angles, rotating around the Z, Y, and X axes, in that order.

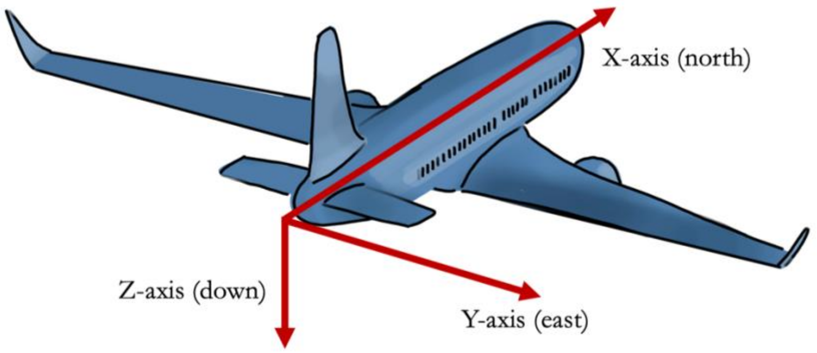


Figure 3.2: Standard aircraft coordinate axes

The BNO055 IMU has two available angular output formats, Euler angles, and quaternions, both of which return absolute orientation of the sensor with respect to the earth. By monitoring the IMU Euler outputs while adjusting its orientation, the sensor's internal coordinate axes were determined (figure 3.3). It is worth noting that this axis frame is different from the one indicated by readings from the acceleration vector output, which does not follow the right-hand rule. A comparison of body frame axis conventions is shown below.

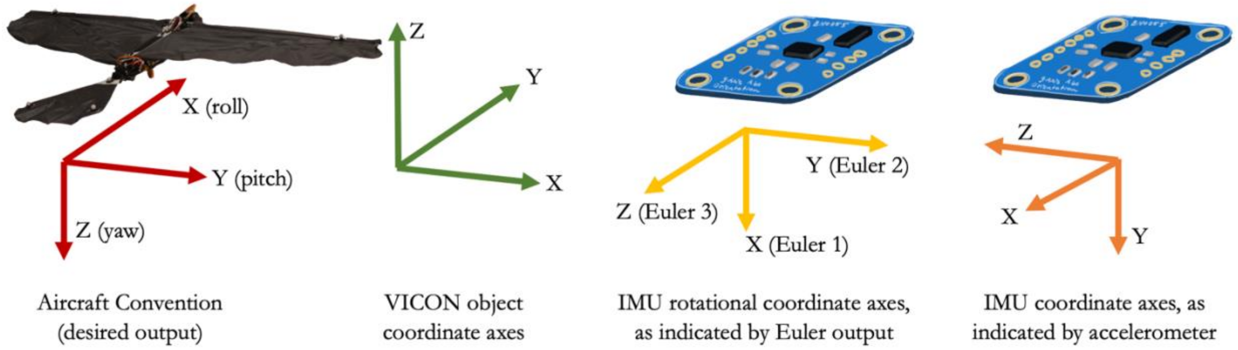


Figure 3.3: Axis conventions of VICON and IMU data in comparison with the standard aircraft frame

In this avionics package, the sensor is positioned on the aircraft rotated 90 degrees around the aircraft x-axis. Testing showed that in this configuration, the Euler angle output of the IMU is not a reliable measure of orientation. This is unsurprising, given that Euler angles have the potential to suffer from a loss of one degree of freedom when rotated by 90 degrees on one axis, a problem known as gimbal lock. Additionally, recommendations from the sensor manufacturer indicate that the Euler angle output makes use of "automatic orientation detection" which is unreliable when pitch or roll exceeds 45 degrees, and that the quaternion output is recommended for absolute orientation [9]. Therefore, while both Euler angle and quaternion outputs were used for testing and debugging, the quaternion output is the only one viable for in-flight usage.

In order to test the orientation output, the ornithopter was tracked as a rigid body in VICON using four markers on the body and wings tips (figure 3.1). Without flapping, the ornithopter was held and rotated by hand to change its orientation in each of the three axes independently, and then in compound orientations making use of all three axes. To aid in post processing, the coordinate frame of the VICON system was set using the calibrated IMU to be at 0 degrees in all directions, and the ornithopter was initialized as an object while at (0, 0, 90) as indicated by the IMU.

The VICON system provides orientation data in either quaternion output or helical (scaled angle-axis rotation) output. Comparing quaternion output from VICON and from the IMU did not initially yield good comparisons, so the quaternion IMU data was manipulated to Euler angles for interpretability. The equations in Table 1 are sourced from the Adafruit sensor library source code and convert quaternions into the equivalent XYZ Euler angles with respect to the sensor. In most tests, the Euler angle corresponding to yaw, or rotation around the Z-axis showed unexpected displacement, potentially an effect of internal recalibration, over the course of the test. Unreliable yaw readings were also noticed when attempting to set the VICON coordinate system using the IMU. Drift around one axis will affect the comparability of more than one value when using a non-sequence-dependent method of angle representation, such as quaternions or angle axis rotations, so in order to obtain comparable results, both the quaternions from the IMU and the helical output from VICON were converted into ZYX Euler angles, with respect to the aircraft frame. By using the Z-axis as the primary Euler axis, the effects of yaw drift are filtered out from the other values. Additionally, due to the orientation of the sensor on the ornithopter body, an extra 90-degree rotation around the aircraft X-axis is reported by the sensor, so by using the X-axis as the final Euler axis, the roll displacement can be corrected for by simply subtracting 90 degrees from the third value.

As indicated by figure 3.3, the axes of the VICON object in this testing setup differ from the axes of the IMU, neither of which correspond to the desired output, which uses the standard aircraft convention (see figure 3.3 for details). Therefore, in order to calculate ZYX Euler angles that correspond to (yaw, pitch, roll), a ZXY Euler sequence is used for the VICON transformation, an

XYZ Euler sequence is used for the IMU quaternion transformation, and several signs are flipped to reflect the directions of the aircraft axes.

Table 3.1: Transformations for obtaining yaw, pitch, roll from IMU output [10]

| Target Euler Angle with respect to aircraft | Corresponding IMU Euler output | Euler calculated from quaternion output |
|---|---|---|
| *Yaw* | $\mathrm{Euler.x(\,)}$ | $-\tan^{-1}\left(\frac{2\,(xy\ +\ zw)}{x^2-y^2-z^2+w^2}\right)$ |
| *Pitch* | $\mathrm{Euler.y(\,)}$ | $\sin^{-1}(2\,(xy\ -yw))$ |
| *Roll* | $-\mathrm{Euler.z(\,)}+90°$ | $\tan^{-1}\left(\frac{2\,(yz\ +\ xw)}{-x^2-y^2+z^2+w^2}\right)-\frac{\pi}{2}$ |

The described transformations were performed on IMU quaternion data, and the VICON angle-axis output was transformed, as described, using MATLAB scripts provided by VICON technical support (see appendix D). Graphing the results yields the following comparison.

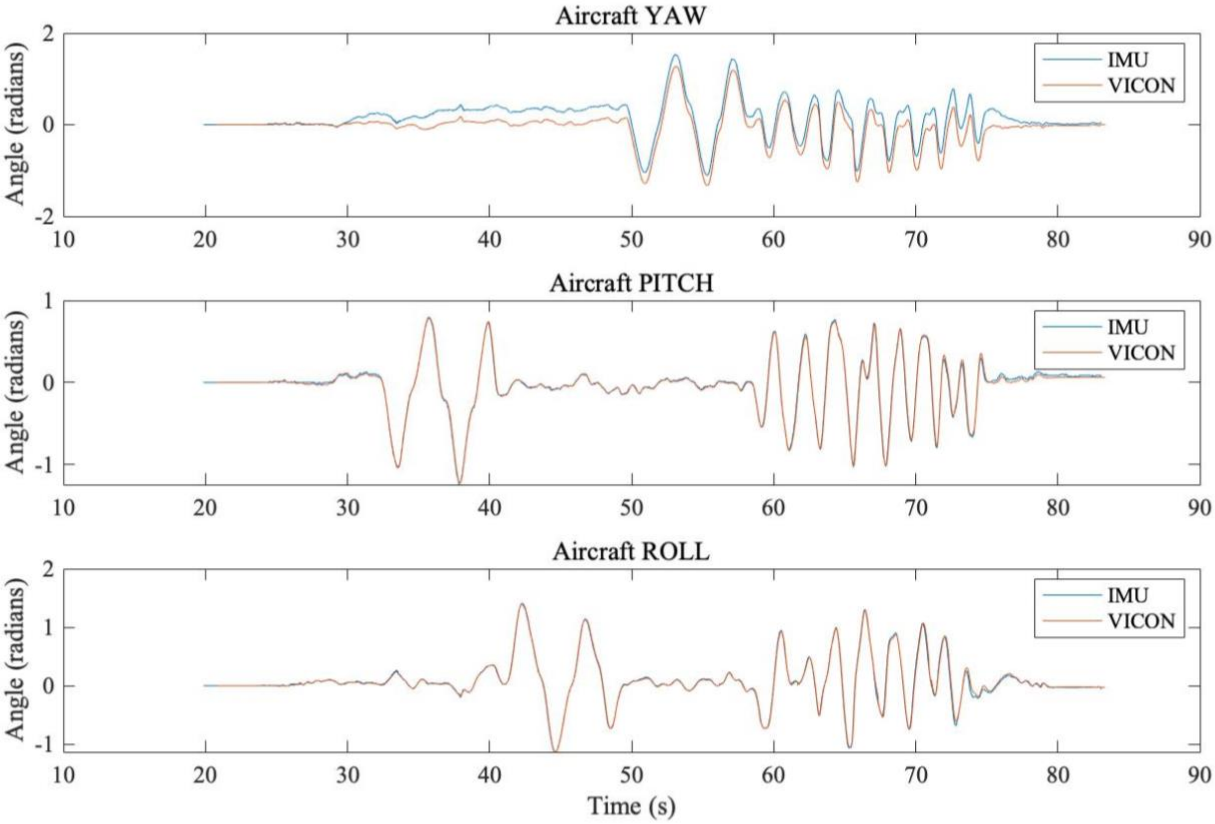


Figure 3.4: Yaw, pitch, roll as calculated from both the quaternion output of the IMU and the angle-axis output from the VICON tracker during handheld maneuvering of the ornithopter

This comparison demonstrates the unexpected changes in the IMU reading for yaw, as calculated from the quaternion output. While this trial shows a small and consistent offset, drift is seen on most trials and varies in extremity. Additionally, in comparison to the 90-degree sensor orientation on the ornithopter, tests with the IMU in its flat orientation tend to see larger drift. See figure E.2 in appendix E for an example. However, by using the Z-axis as the primary axis in the Euler angle rotation, the effects of the drift are filtered out of the other two values, revealing a very accurate tracking in both pitch and roll.

One possible explanation for the drift seen in yaw is the fact that the IMU's sense of heading is completely dependent on its estimation of the earth's magnetic field using the magnetometer, which can be impaired by large steel structures or affected by changing magnetic fields from nearby equipment. The sensor calibrates actively, and recalibration while moving within the environment may result in changing heading estimations. To test if the environment was resulting in heading uncertainty, a simple test was performed. While recording orientation, the IMU was placed in a fixed position, rotated 90 degrees, as on the ornithopter. It was then picked up and rotated around all three axes before being returned to the original fixed position. This was repeated five times in a row and the test was repeated inside the motion capture stage as well as outside of the building. The results are shown below.

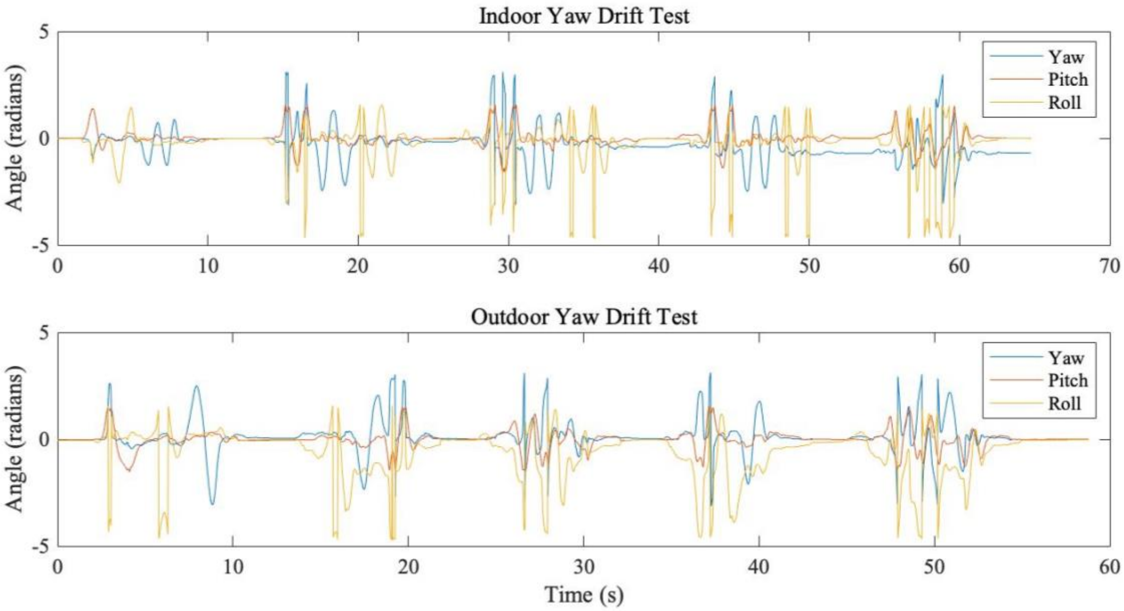


Figure 3.5: Basic test to detect drift in yaw indoors versus outdoors

Inside the motion capture stage, the baseline value for yaw changes gradually over the course of the test, but this effect is not observed when testing outdoors, indicating that the physical environment is affecting the reading (a second test, in the flat orientation is shown in appendix E). Based on these tests, the current inertial measurement unit is a reliable source of pitch and roll data, but yaw data can be unreliable during indoor data collection.

## 3.2 Tail Position and Servo Feedback

As described briefly in the introduction, the tail of the ornithopter is controlled using two servos, one that adjusts the pitch of the tail via a four-bar-linkage, and a second servo mounted on the four-bar-linkage that adjusts yaw. In this project, analog feedback servos were installed to monitor the position of each servo in addition to recording the control input. Analog readings taken by the microcontroller are integers that range from 0 to 1023, although with the servos output range and the voltage divider, readings only range from 257 to 426. Control input pulse width from the radio receiver ranges from 1107 to 1913 microseconds. In order compare control input to actual servo arm position, relationships were first derived to map servo feedback values to control input. Each servo was held at five fixed positions, and the average feedback value and control input were both recorded. Using these data points, the following linear relationships were derived:

$$4.727(yaw\ feedback) - 113.8 = corresponding\ yaw\ control\ input \qquad 1$$

$$4.728(pitch\ feedback) - 124.0 = corresponding\ pitch\ control\ input \qquad 2$$

In the following figure, these relationships are used to rescale feedback readings to the control input range, enabling a comparison of the actual servo position to control input during quick tail motion.

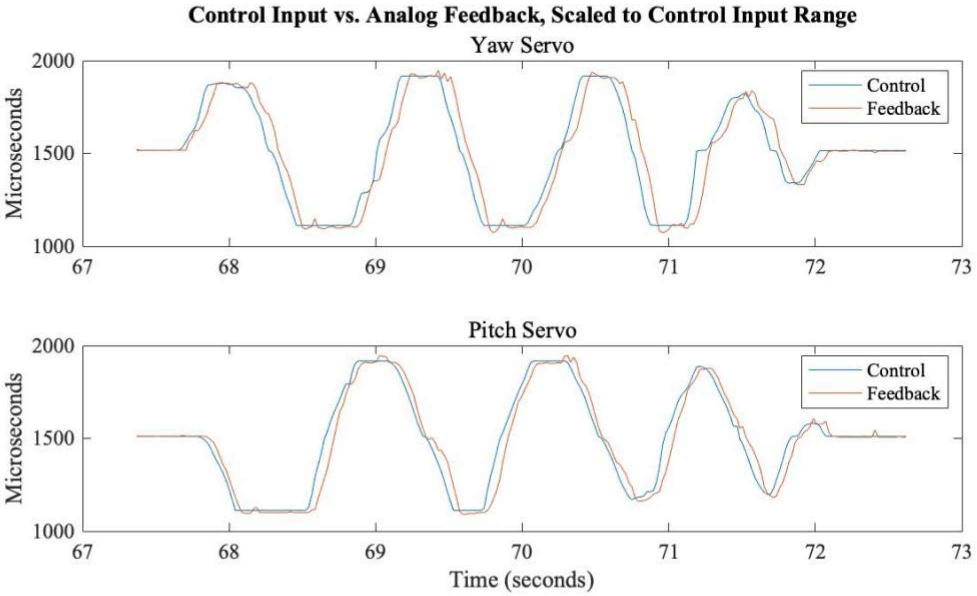


Figure 3.6: Tail servo control input plotted against feedback readings to demonstrate tracking lag

This comparison demonstrates the short lag between control input and the servo actually moving to the specified position. The next step was to map the control input range to the actual tail position, which was demonstrated using the tail pitch. By tracking both the ornithopter body and the ornithopter tail as rigid objects in VICON, the difference in pitch between the two could be tracked while changing the control input and reading the feedback. Similar to the initial feedback mapping, the average tail pitch and servo feedback were recorded at five fixed points and a relationship was derived between the two. In this case, the crank angle and rocker angle of a four-bar linkage are being compared, so the relationship is not expected to be linear. On this range of inputs, a quadratic fit provides a good estimate of the relationship.

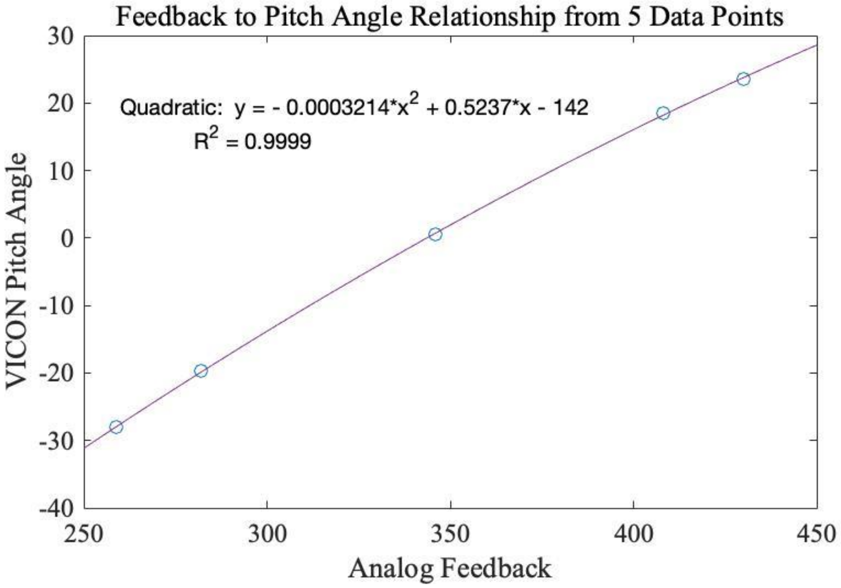


Figure 3.7: Using quadratic fit to obtain relationship between analog feedback values and tail pitch angle

This relationship maps feedback readings to the expected tail position, with zero degrees corresponding to the baseline (mid-point) tail position. To demonstrate the robustness of this relationship, the tail pitch, as expected indicated by servo feedback can be compared with the tail pitch as recorded in motion capture (figure 3.8).

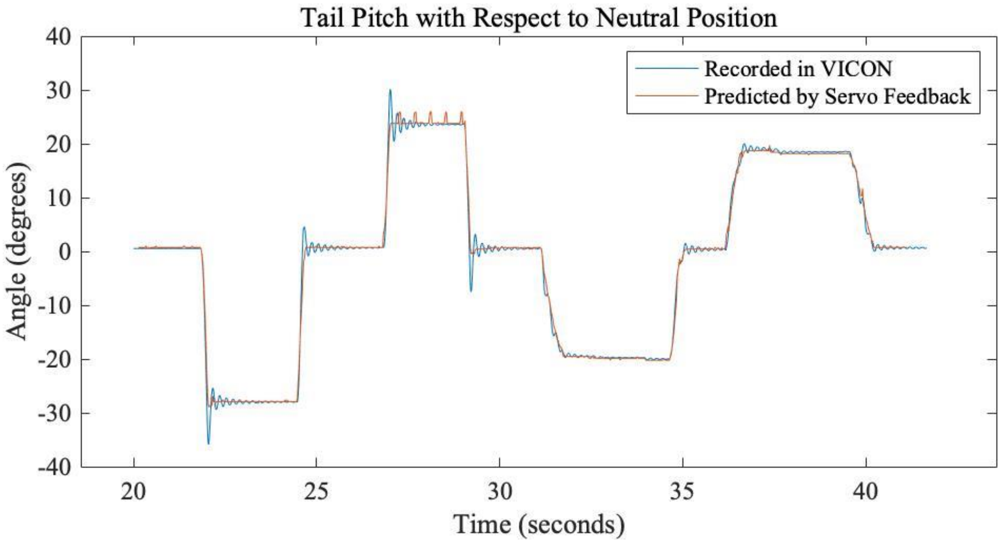


Figure 3.8: Tail pitch calculated from servo feedback, compared with VICON data

In this comparison, flex in the tail results in small oscillations after quick position changes, which is observed by the VICON object tracker as overshoot. Aside from this effect, the model shows reliable position estimation.

This method of tail position sampling is expected to be more accurate than predicting position based on the control input, especially while under actual in-flight loads. To demonstrate this, pressure was applied by hand to the top side and then underside of the tail to imitate load during flight. This test undoubtedly exceeded the limits of realistic load experienced during flight, and the servo position experienced what looked like jitter as it attempted to maintain its original position. Additionally, the tail position recorded by VICON was not necessarily comparable to the one predicted by servo feedback due to the flex in the tail and the positioning of the markers. Nevertheless, the unstable servo movement was reflected in the data recorded from the servo feedback, while the control input indicated no changes. While more testing and modeling is necessary to be able to fully estimate tail position during flight, these tests demonstrate the usefulness of analog feedback for detecting when the tail position deviates from its expected position. A plot of the reference load test can be found in appendix E.

## 3.3 Flapping Rate and Wing Position

The final objective of the avionics package is to provide accurate estimations of wing angle and motor speed during flapping. In the following tests, the ornithopter was placed in a stand and VICON markers were affixed to the leading edge of the left wing. Data was recorded from both the gear shaft encoder and the shoulder encoder while flapping the ornithopter, and the test was also tracked in VICON and recorded with high-speed video. The following figure shows the wing position recorded from the shoulder encoder overlaid with the vertical motion (z-coordinate) of a mid-wing VICON marker. The timing of the trials was aligned by giving the wing a sharp tap before the flapping sequence and manually aligning this spike in both data sets.

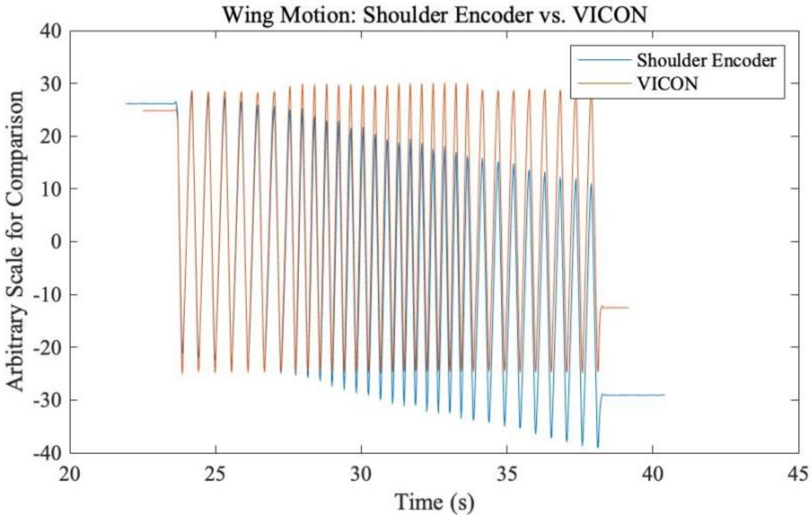


Figure 3.9: Wing oscillatory pattern extracted from VICON and shoulder encoder data during three periods of constant throttle

While the vertical motion of the marker and angular motion of the joint are not directly comparable, the shoulder encoder clearly detects the same motion and flapping frequency as observed in VICON. The drift seen in shoulder angle readings is a result of the temporary design of the encoder apparatus, which relies on friction to ensure that the magnet rotates in sync with the shoulder joint (see Chapter 2.5). With a simple modification to the shoulder joint, such as replacing the bolt with a pin and set screw, this setup could provide a very reliable way of sensing wing angle.

The gear shaft encoder was also sampled during the same test. As discussed in the Chapter 2.4 and 2.5, to eliminate interrupt usage, the avionics package does not keep track of the absolute position of the encoders, but rather samples their instantaneous angular position during data recording using SPI. This is a good solution for the low frequency oscillatory motion of the shoulder, but initial testing revealed that the gear shaft rotates too quickly to be accurately tracked using angular position data on 21ms intervals. Instead of changing the programing approach to attempt a higher sample rate, a short routine was written that takes a short time interval (2 milliseconds in this test) and samples the encoder position to make an estimate of rotations per second during that sampling period. The following figure show a section of the data recorded from both encoders. A plot of the full trial can be seen in appendix E.

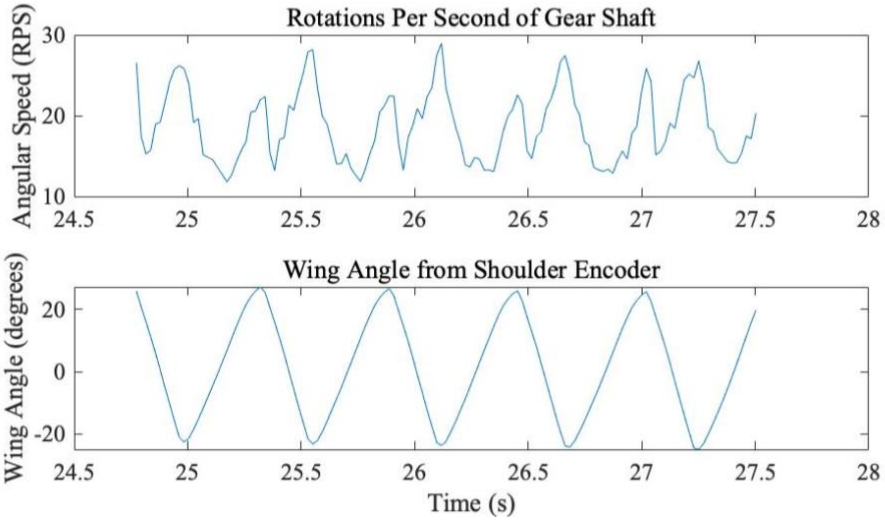


Figure 3.10: Data from shoulder and shaft encoders while flapping at constant throttle

The rotational speed of the gear shaft encoder shows high variability depending on the position of the wing, and this pattern indicates two main effects. As this a stationary test, and the ornithopter body is being supported by a stand, gravity assists the downstroke of the wing, so rotational speed of the motor increases during downstroke, as shown by the peaks in the angular speed during decrease in wing angle. The second effect relates to the shape of both the angular speed and wing angle plots. While the expected motion of the wing joint would be neatly sinusoidal over time, there is a relatively sharp transition from upstroke to downstroke reflected in the wing angle plot. This pattern is also reflected in the VICON data, and high-speed video analysis shows the upstroke continuing even while the eccentric standoff on the gear shaft has rotated past 12 o'clock and begun downward motion. At a certain point, the standoff connection becomes taut and yanks the wing into the downstroke. This indicates play in the mechanism that allows the momentum of the upstroke to momentarily unload the gear shaft, resulting in a small spike in rotational speed, before the downstroke engages, which briefly slows the gear box down, before being sped up again due to gravity.

In addition to the wing angle returned by the shoulder encoder, the pattern observed by sampling rotational speed of the gear shaft also provides a very reliable indicator of wing position during constant throttle. Further research is necessary to examine the relationship between wing position and rotational speed during free flight, or while changing throttle, but these results show that both encoder solutions can provide useful and accurate information on both flapping frequency and wing position.

# Chapter 4: Summary and Conclusions

## 4.1 Summary of Results

The goal of this project was to develop a robust, easy to use, and adaptable system for in-flight data collection on a robotic ornithopter and to verify the validity of the sensory output.

The avionics package designed and fabricated in this project successfully intercepts the existing ornithopter electronics, enabling control by either the onboard processor or by a remote human operator. Sensor data is collected and recorded roughly 48 times per second, which still leaves much of the processor time available for other tasks. Automatic file generation, on board removable storage, and Bluetooth capabilities make data easily accessible, and the package is small, organized, and lightweight.

The three main sensory focuses of the ornithopter avionics package are 3D orientation, tail servo position, and wing position. Solutions were devised for collecting data in each of these areas, and testing was begun to verify their efficacy.

Orientation data taken from an on-board inertial measurement unit showed excellent tracking in pitch and roll when compared against motion capture data. The unit's estimation of yaw, or heading proved unreliable within the motion capture stage, which further testing indicated was due to the magnetic properties of the environment and the sensor's constant recalibration. The observed effect would not be expected to pose problems when flying outdoors.

The installment of analog feedback servos in the tail mechanism enabled a more direct method of sampling actual servo position, which was shown to differ from the control input during fast tail motion. Initial tests were conducted to map servo position readings to tail pitch, and a relationship between the two was derived. In order to achieve the ultimate goal of estimating tail orientation based on servo position, similar testing and analysis must be done with the yaw servo, and data should be collected under realistic aerodynamic loads. However, this project provides a good starting point and demonstrates the usefulness of analog feedback for position sensing.

The magnetic encoder on the gear shaft of the ornithopter was replaced with a more reliable mounted solution, and a second encoder was installed experimentally on the shoulder joint.

Reading angular position via SPI instead of tracking incremental outputs simplified the package and increased reliability by eliminating interrupt usage. Flapping rate data in the form of angular position and RPS estimates was confirmed using motion capture data and also revealed a two-peaked load pattern experienced by the ornithopter motor, revealing wing motion that deviated from the ideal sinusoidal pattern. This development identifies areas for improvement in the wing mechanism and the current encoder apparatus provides multiple ways of detecting wing angle and flapping frequency.

## 4.2 Suggestions for Future Work

The broad next phase of this project is to perform many of the same tests but during free and constrained flight. This will enable the development of relationships between inputs and outputs that can directly inform the control model for the ornithopter. For instance, characterizing the in-flight load pattern experienced by the motor can provide data on how to reliable return to gliding position or execute a single flap at a given speed. However, in the short term, several tests and modifications can be carried out to improve the avionics package.

If accurate yaw orientation indoors is a requirement of future experiments, then steps should be taken to further examine the heading problem. First, another BNO055 sensor should be tested to see if the part is defective. Another magnetometer could also be tested in the same environment. Additionally, when the ornithopter transitions out of the motion capture stage, in may be useful to reconfigure the internal axes of the sensor by programming the chip using the Adafruit library. This could simplify data processing and eliminate future confusion by removing the need to perform transformations on outputs.

As described, a model is still needed for predicting the tail orientation from the combination of both servo positions, and this model should be tested under flight-like conditions.

To further improve the ornithopters ability to sense wing position, a second, more reliable version of the shoulder encoder should be developed as described in Chapter 3.3. Additionally, the standoff-shoulder joint mechanism should be examined for ways to reduce the play observed in flapping tests.

With regard to the avionics package itself, the primary limitation of the system is its current form is its dependence on the timing cycle dictated by the radio receiver pulse-width inputs. As detailed in Chapter 2.3, this solution is effective for robust data collection during human operated flight and is only necessary while using the RC controller. However, transitioning to a different method of radio communication would free up significant processor time and introduce flexibility for custom data recording intervals, as well as provide a better platform for inflight communication when implementing control.

## 4.3 Conclusion

While areas remain for further improvement of the avionics package, the stated objectives for this project have been met. The ornithopter is now equipped with a reliable platform for in-flight data collection which will enable analysis of actuator motion and physical orientation during flight, providing a tool for dynamics research and for the development of control on the ornithopter.

# Reference List

[1] Wissa, A., 2014, “Analytical Modeling and Experimental Validation of a Passively Morphing Ornithopter Wing,” Ph.D. Dissertation, Department of Aerospace Engineering, University of Maryland, College Park.

[2] Harmon, R. L., 2008, “Aerodynamic Modeling of a Flapping Membrane Wing Using Motion Tracking Experiments,” M. S. Thesis, Department of Aerospace Engineering, University of Maryland, College Park.

[3] Heimerdinger, M., 2016, “Final Report,” ME 497, University of Illinois Urbana-Champaign.

[4] Murphy, K., 2019 “Development of a Flight Avionics Package Addition for an Ornithopter,” ME 497, University of Illinois Urbana-Champaign.

[5] Stoffregen, P., 2022, ‘Teensy 4.1 Development Board’ [online]. Available: https://www.pjrc.com/store/teensy41.html. [accessed: April 23, 2022]

[6] AKM, 2022, ‘Principle and Advantages of Magnetic Encoder’ [online]. Available: https://www.akm.com/us/en/products/rotation-angle-sensor/tutorial/magnetic-encoder/. [accessed: April 20, 2022].

[7] AMS, 2016, “AS5047D 14-Bit On-Axis Magnetic Rotary Position Sensor with 11-Bit Decimal, Binary, and Incremental Pulse Count.”

[8] Douglas, B., MATLAB, 2019, ‘Understanding Sensor Fusion and Tracking, Part 2: Fusing a Mag, Accel, & Gyro Estimate’ [online]. Available: https://www.youtube.com/watch?v=0rlvvYgmTvI. [accessed: April 18, 2022]

[9] Townsend, K., 2015, ‘Adafruit BNO Absolute Orientation Sensor – FAQ’ [online]. Available: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/faqs. [accessed March 20, 2022].

[10] Adafruit, 2021, ‘Adafruit_BNO055/utility/quaternion.h’ [online]. Available: https://github.com/adafruit/Adafruit_BNO055/blob/master/utility/quaternion.h. [accessed: April 22, 2022].

# Appendix A: Ornithopter Construction and Previous Work

In the following appendices, references are made to documents contained in a project folder, which can be obtained by contacting the Bio-inspired Adaptive Morphology Lab at Princeton University.

## A.1 Ornithopter Construction and Assembly

See report and instructions created by Madison Heimerdinger: Shadow Project Folder 2022 → Previous Work and Reports → Madison Heimerdinger - ME 497 Final Paper.pdf

Shadow Project Folder 2022 → Previous Work and Reports → Mech Assembly – Body.dox

## A.2 Open Loop Electronics

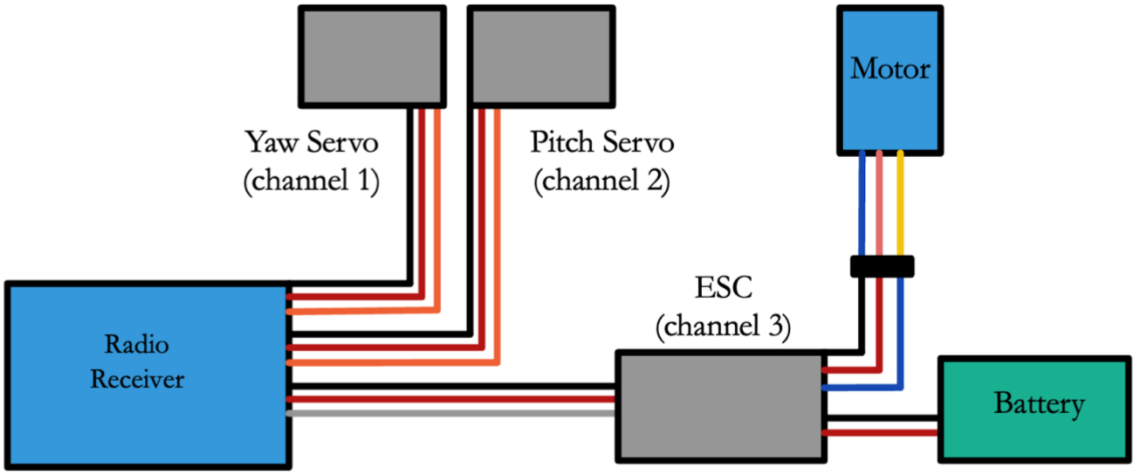


Figure A.1 Open Loop wiring diagram for ornithopter (avionics package not installed)

Table A.1: Components list for open loop ornithopter

| Component | Part Number |
|---|---|
| Analog Feedback Servos | Batan analog s1213 (Adafruit product ID 1404) |
| Radio Receiver | Minima 6E 2.4 GHz 6 Channel aircraft receiver |
| Electronic Speed Controller | UBEC 30A RC ESC |

| | |
|---|---|
| Battery | PKZ1033 11.1V 1300 mAh 3-cell Lithium Polymer |
| Motor | Surpass Hobby Blue 2435 4800 kV brushless motor |

For information on programming the ESC using the remote controller, see the following link: https://www.rcelectricparts.com/esc-user-guide.html#04

# Appendix B: Avionics Package Construction

## B.1 Components and Wiring

Table A.1: Components list for avionics package and encoders

| Component | Part Number |
|---|---|
| Microcontroller | PJRC Teensy 4.1 |
| IMU | Adafruit BNO055 |
| Bluetooth Module | HC-06 |
| Encoder board | Digi-Key AS5047D-TS_EK_AB |

Useful data sheets can be found in Shadow Project Folder 2022 → Data Sheets

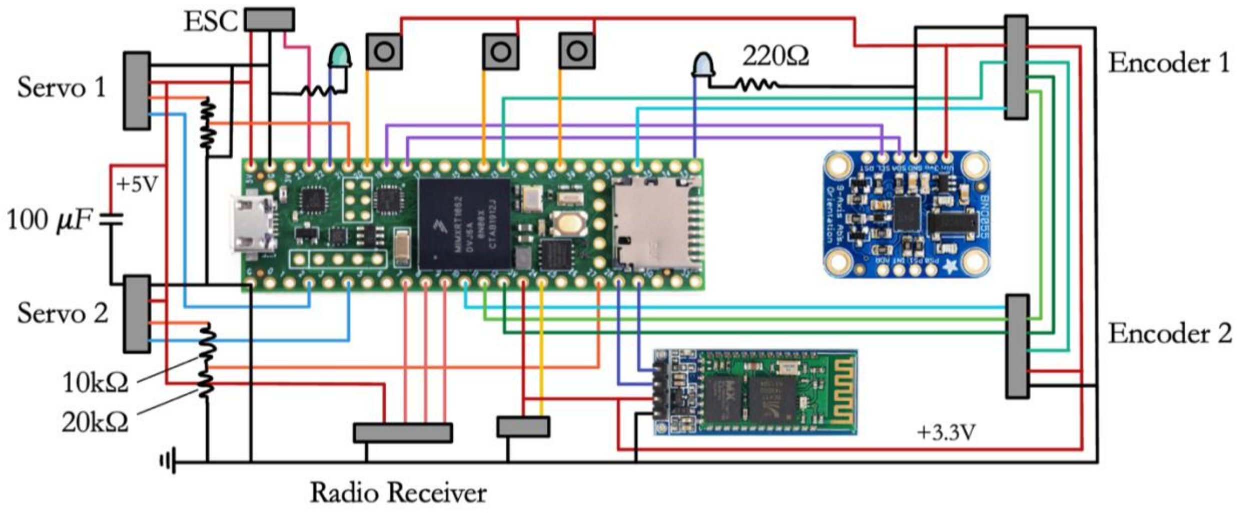


Figure B.1: Wiring diagram for board components

## B.2 Package Construction

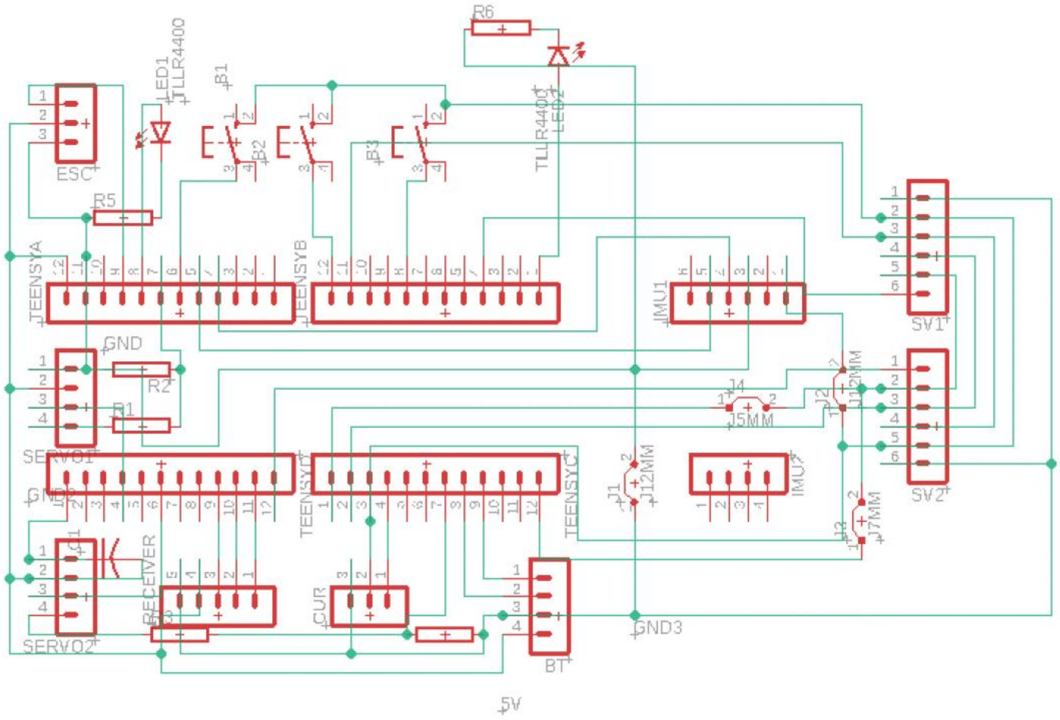


Figure B.2: Eagle schematic of Avionics Package

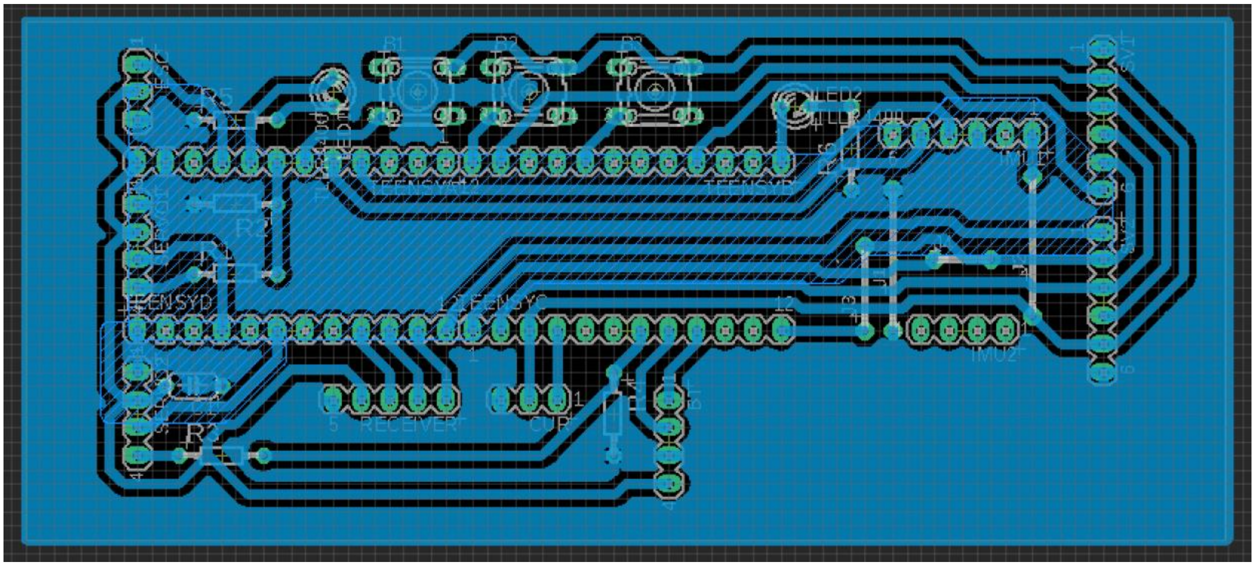


Figure B.3: board design in Eagle

Eagle Schematic and Board Files can be found in Shadow Project Folder 2022 → Board Files

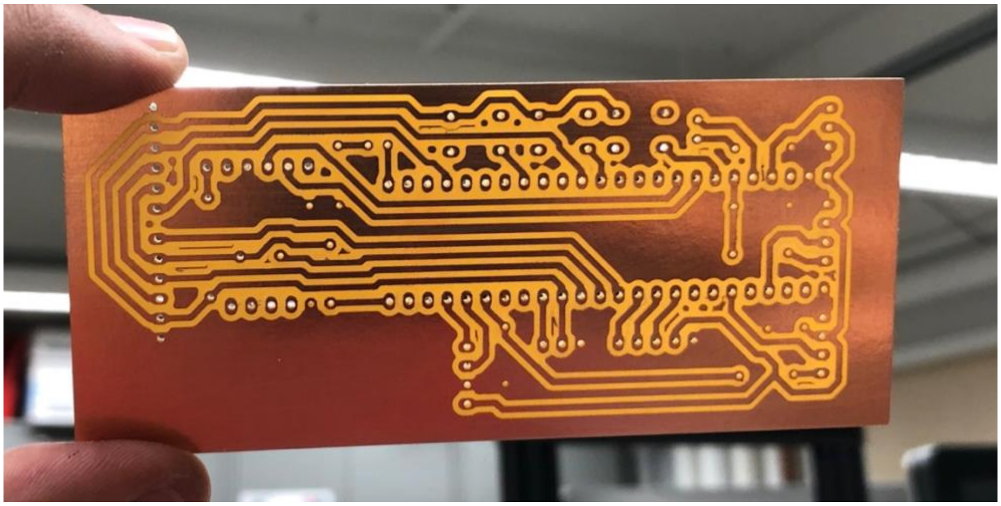


Figure B.4: Milled circuit board

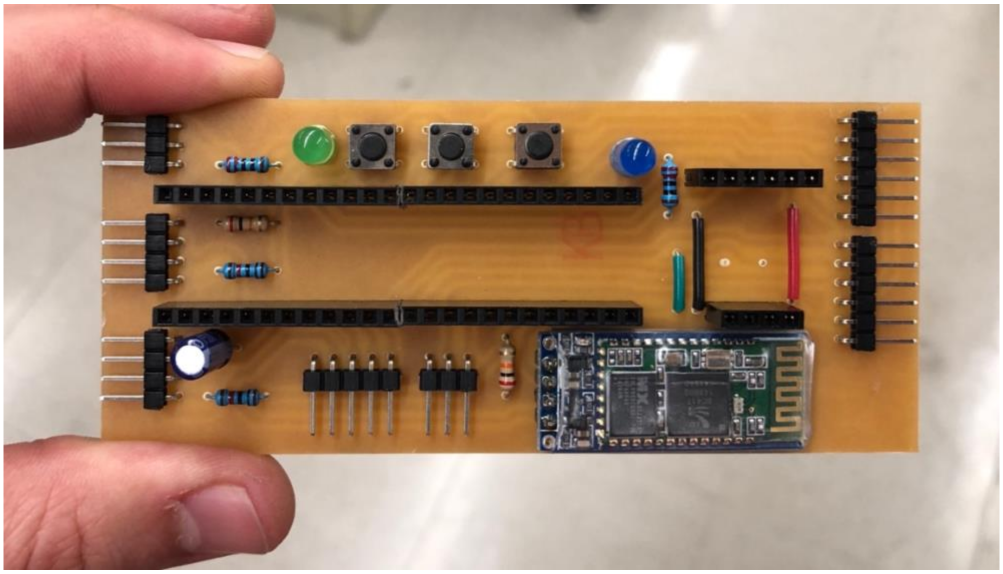


Figure B.5: Assembled circuit board

As mentioned in chapter 2.4, the microcontroller currently reads the angle registers of the encoders by connecting the MOSI pin to pin 32 on the microcontroller and writing it high all the time, but the circuit board has been designed to enable full SPI communication if desired. By removing the green jumper wire and soldering a jumper between the two holes located between the black and red jumpers, the encoder MOSI lines will be connected to the MOSI pin of the microcontroller.

Chapter 2.8 mentions the use of voltage dividers to step down the voltage from the analog feedback servos and how this solution was proved unnecessary. In order to obtain higher resolution from the analog feedback, the 10kΩ resistor could be removed from both voltage dividers and replaced with a jumper. The 10kΩ resistors are the two blue resistors on the lower left. This modification will necessitate a repeating of the tests performed in chapter 3.2 to develop new relationships to map feedback values to servo position.

As noted, the package also includes a header for an analog current sensor. A sensor (Part number: ACS711EX) was selected and purchased for this purpose, but it has not yet been implemented.

Finally, if the Teensy 4.1 should ever need to be replaced, the replacement teensy board will need to be modified by cutting the trace between the two pads indicated in figure B.6. This can be done with a razor blade and will enable the teensy to be powered from the circuit board while the USB cable is connected.

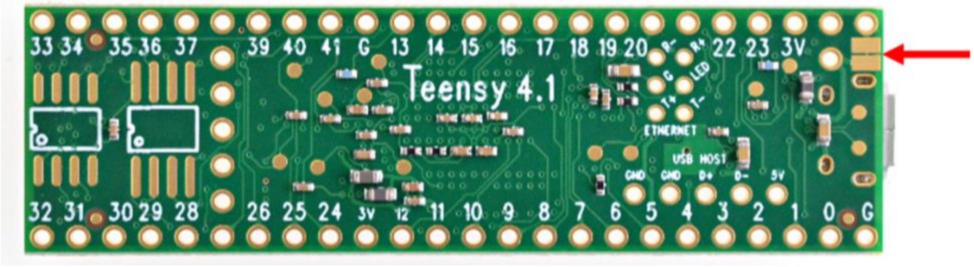

33 34 35 36 37 39 40 41 G 13 14 15 16 17 18 19 20 22 23 3V
Teensy 4.1
LED
ETHERNET
USB HOST
GND GND D+ D- 5V
32 31 30 29 28 26 25 24 3V 12 11 10 9 8 7 6 5 4 3 2 1 0 G

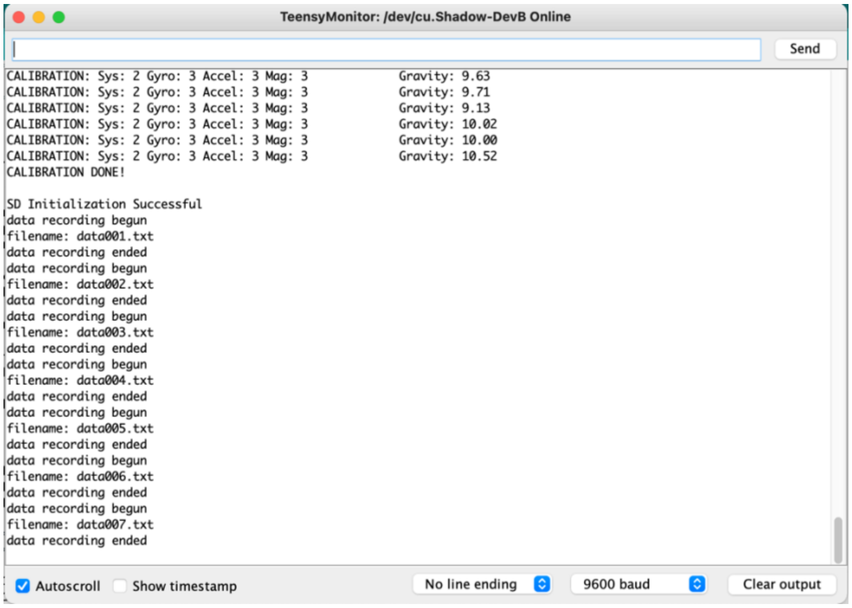
Figure B.6: Trace to be cut on Teensy board [5]

# Appendix C: Using the Avionics Package

## C.1 Teensyduino and Bluetooth

To program and interact with the Teensy 4.1 microcontroller, first install the Arduino IDE, available here: https://www.arduino.cc/en/software. Then, install the Teensyduino add-on available from PJRC: https://www.pjrc.com/teensy/td_download.html. Also install the teensy loader, available here: https://www.pjrc.com/teensy/loader.html. Launching the Teensyduino application will launch the Arduino IDE, from which programs can be written and the teensy can be interfaced with. Be sure to select the Teensy board from the tools menu. When programming the board, Teensyduino will launch the teensy loader application. A small red LED on the teensy board will flash dimly and then brightly when the program is loaded. If having trouble connecting to the board, check the micro-USB cable. In this project, a brief survey of all the micro-USB cables in the land revealed that most cables in existence are charging-only cables and do not have data lines.

The avionics package Bluetooth port is discoverable as “Bird Brain”. To connect to the board, connect to Shadow as a Bluetooth device using the passcode 1234. To monitor the Bluetooth output using the Arduino IDE serial monitor, connect to the board, and then, in the tools – port menu, select the Bluetooth device. Open the serial monitor and from the drop-down menus select “no line ending” and “9600 baud.”



```
CALIBRATION: Sys: 2 Gyro: 3 Accel: 3 Mag: 3            Gravity: 9.63
CALIBRATION: Sys: 2 Gyro: 3 Accel: 3 Mag: 3            Gravity: 9.71
CALIBRATION: Sys: 2 Gyro: 3 Accel: 3 Mag: 3            Gravity: 9.13
CALIBRATION: Sys: 2 Gyro: 3 Accel: 3 Mag: 3            Gravity: 10.02
CALIBRATION: Sys: 2 Gyro: 3 Accel: 3 Mag: 3            Gravity: 10.00
CALIBRATION: Sys: 2 Gyro: 3 Accel: 3 Mag: 3            Gravity: 10.52
CALIBRATION DONE!

SD Initialization Successful
data recording begun
filename: data001.txt
data recording ended
data recording begun
filename: data002.txt
data recording ended
data recording begun
filename: data003.txt
data recording ended
data recording begun
filename: data004.txt
data recording ended
data recording begun
filename: data005.txt
data recording ended
data recording begun
filename: data006.txt
data recording ended
data recording begun
filename: data007.txt
data recording ended
```

Figure C.1: Arduino serial monitor receiving messages from the Bluetooth unit

An LED on the HC-06 begins flashing upon power-up. When the serial monitor is opened, the LED should stop flashing. When the package is powered off and then on again the serial monitor should be closed and reopened. The name, passcode, and baud rate of the Bluetooth unit can also be programmed using AT commands. See the following article for more information: http://www.martyncurrey.com/arduino-and-hc-06-zs-040/

## C.2 Assembling and Using the Package

To assemble the avionics package on the ornithopter, the ESC, motor, and battery are all connected as shown in the open loop wiring diagram (figure A.1). Then the servos, esc, radio receiver, and encoders are connected to the board as shown in figure 2.16 (also included below). The board will power on once connected to power via the ESC, and power-up can be indicated by the LED on the HC-06. To connect the radio receiver, the brown and red wires can be connected to any of the ground and 5V connections on the receiver (red = 5V). The orange, yellow, and green lines are connected to the PWM output lines of channels 1, 2, and 3, respectively, on the receiver.

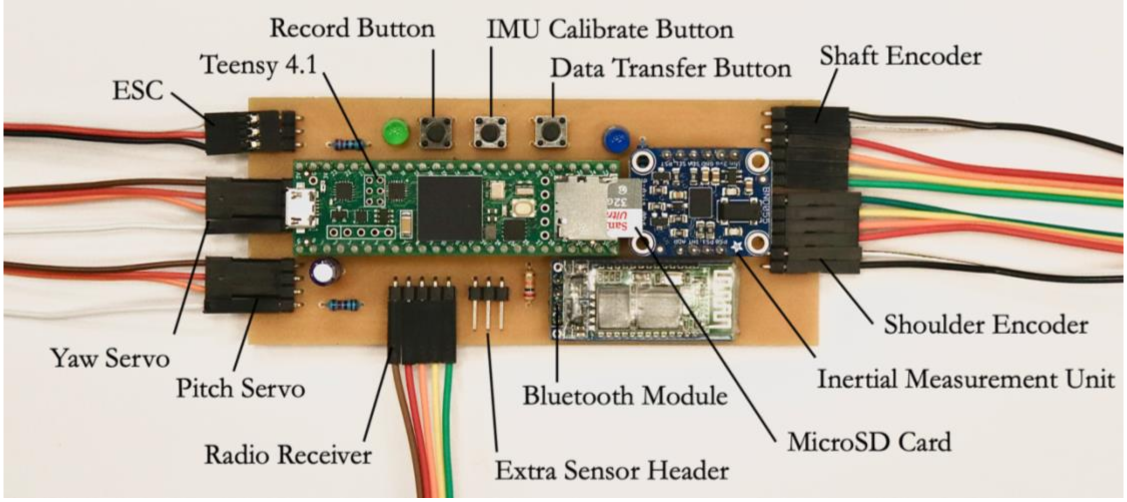


Figure 2.16, reinserted

If using the IMU, calibration will need to be performed after every power-up. To monitor calibration status, press the IMU calibrate button, and the green LED on the board should begin flashing. At this point the calibration readout, as well as the estimation of gravity magnitude and the Euler angle output will be displayed every 250 milliseconds. Calibration values range from 0 to 3, with 3 being fully calibrated. To calibrate the gyroscope, leave the package stationary for several seconds, then move the package around in a figure-8 motion to calibrate the magnetometer. Finally, to calibrate the accelerometer, hold the package for five seconds with one of the IMU axes pointed down, then rotate to have a different axis pointed down, continue this pattern of 90-degree rotations to give the accelerometer calibration time in all orientations. This process will take the most time. Once all values read 3, the IMU is calibrated. The button can be pushed to exit the calibration monitoring routine at any point.

To record data, the record button is pushed, and the green LED is turned on. Data will not be recorded if the SD card is not in place. Pressing the button a second time ends data recording. With each data recording, a new file is generated titled “data001.txt” with a unique number. To transfer data from the SD card, the data transfer button can be used, which will read out the most recently created data file since power-up and print its contents to the serial monitor. However, in its current state this routine is very slow and therefore is not recommended for large data files. It is ideal for taking a small data recording and viewing it immediately, which can be helpful for debugging purposes. The best way to transfer data is to remove the microSD card.

# Appendix D: Code

The full Arduino code for the avionics package can be found in Shadow Project Folder 2022 → Code → Avionics_Package_v16

Several MATLAB functions were provided by John Porter at VICON for use in data processing. Specifically, the RotationMatrixFromAngleAxix( ) and EulerFromMatrix( ) functions were used to convert the helical output from the VICON system to ZXY Euler angles, with respect to the VICON coordinate system. The script provided can be found in Shadow Project Folder 2022 → Code → ViconUtils.m

# Appendix E: Extra Figures

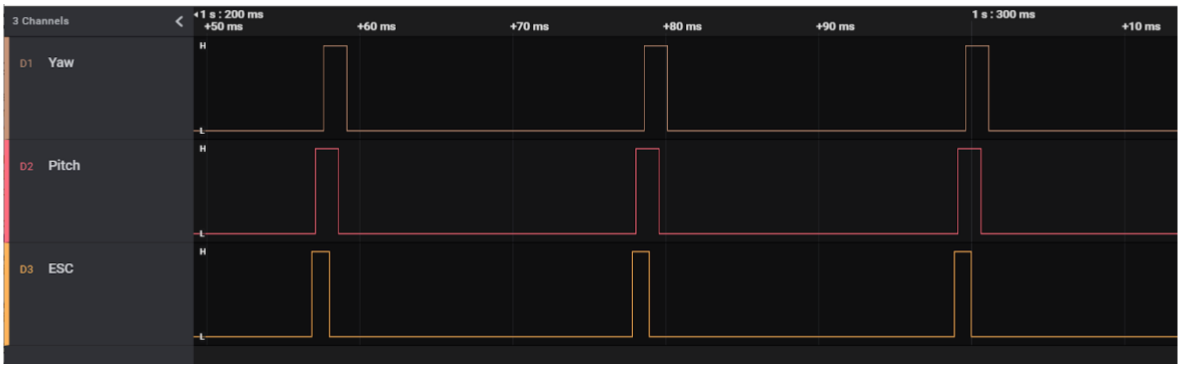


Figure E.1: Pulses from radio receiver tracked using a digital logic analyzer

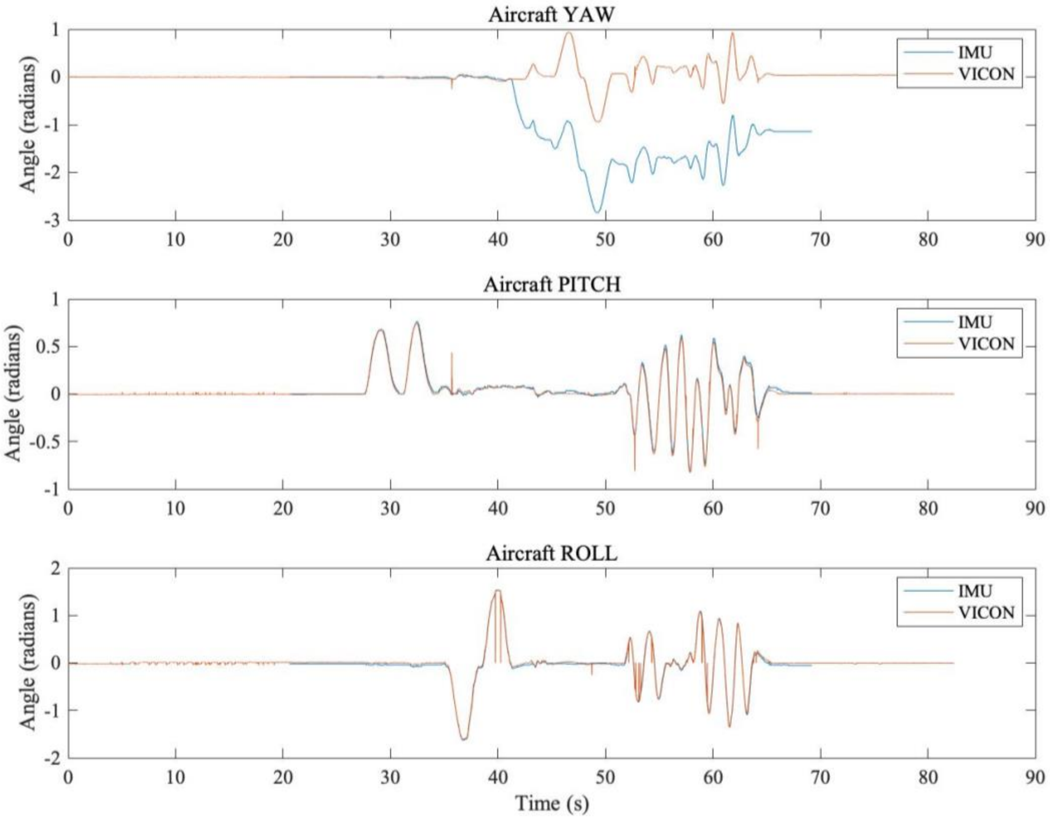


Figure E.2: Yaw, pitch, roll comparison for a trial using the IMU in its flat position. IMU was affixed to the VICON active wand, which was tracked as a rigid body

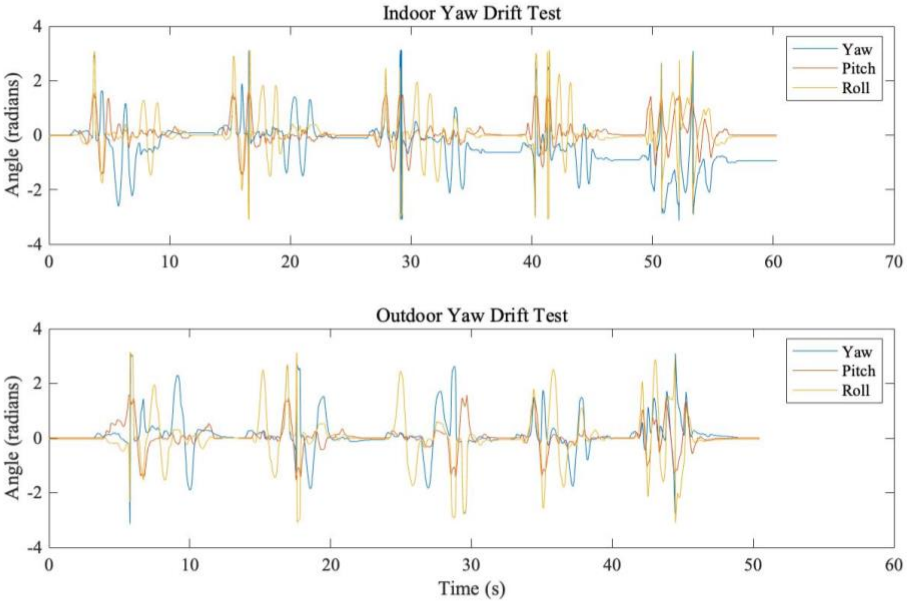


Figure E.3 Indoor versus outdoor yaw drift test with IMU in the flat orientation

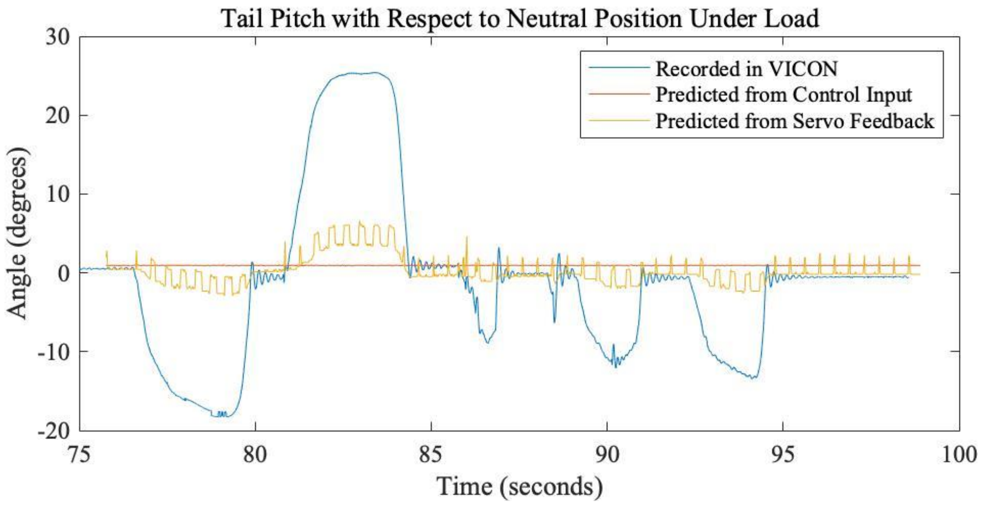


Figure E.4: Tail pitch under load compared with input and feedback reading, referenced in chapter 3.2

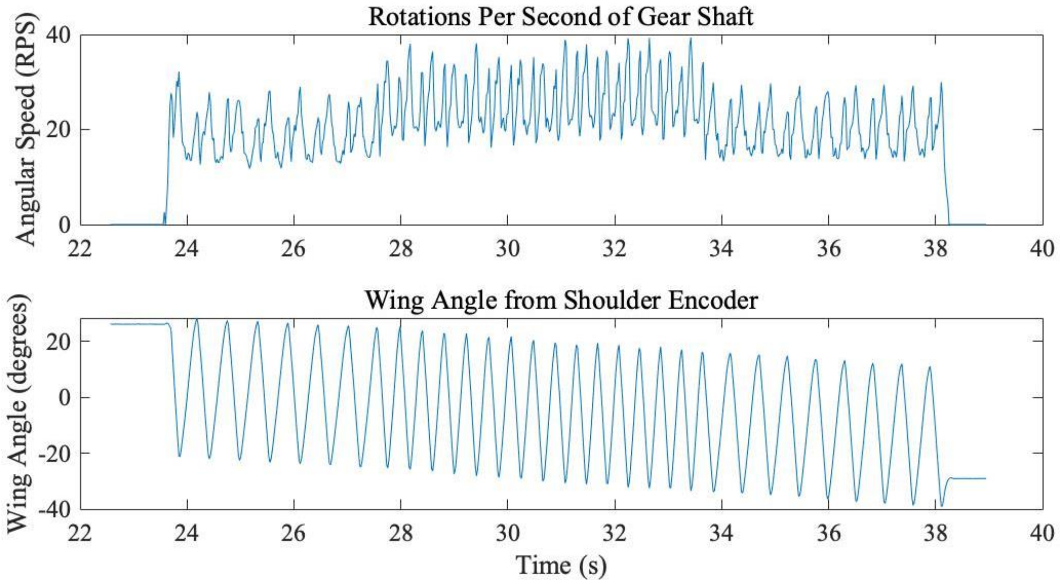


Figure E.5: Data from shoulder and shaft encoders while flapping during three periods of constant throttle