Flight Characterization of Small Scale UAV

by
Jacob Scheer

A THESIS

submitted to
Oregon State University
University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mechanical Engineering
(Honors Associate)

Presented May 27, 2016
Commencement June 2016

AN ABSTRACT OF THE THESIS OF

Jacob Scheer for the degree of Honors Baccalaureate of Science in Mechanical Engineering presented on May 27, 2016. Title: Flight Characterization of Small Scale UAV .

Abstract approved: _____

Dr. Nancy Squires

The purpose of this thesis was to understand the relation between the actual performance and the design performance of the manufacturing plane which was designed for the 2016 Design/Build/Fly competition. The actual performance was determined using an Arduino Uno based system with data from an accelerometer, gyroscope and pitot tube. While the data did not represent a statistically significant sample size it did suggest a few things:

- The cruise speed and takeoff speed while similar were slightly lower than the analytical results.
- The turning radius was slightly higher than expected, but at a lower airspeed.
- The aircraft is statically stable, but improvements in trim can be used to change the steady state settling angle.

Key Words: UAV, Airplane, Arduino, Flight Characterization

Corresponding e-mail address: scheerj@oregonstate.edu

Flight Characterization of Small Scale UAV

by
Jacob Scheer

A THESIS

submitted to
Oregon State University
University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mechanical Engineering
(Honors Associate)

Presented May 27, 2016
Commencement June 2016

Honors Baccalaureate of Science in Mechanical Engineering project of Jacob Scheer presented on May 27, 2016.

APPROVED:

Nancy Squires, Mentor, representing MIME

Roberto Albertani, Committee Member, representing MIME

Nathaniel Osterberg, Committee Member, representing MIME

Toni Doolen, Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

Jacob Scheer, Author

Table of Contents

Introduction

Engineers use scientific theory to produce designs and models of predicted performance. However, as important as it is to produce these models, it is just as important to validate them. This thesis project is meant to validate the model of a small scale unmanned aerial vehicle (UAV) through flight characterization, specifically for the 2015-2016 Design/Build/Fly (DBF) manufacturing plane. Flight characterization means getting data on the expected performance of the plane. The performance variables are determined by the set of engineering specifications which are developed for this plane. Validation is typically done through a series of testing parameters. Below is the set of engineering specifications that this thesis is meant to test:

Thesis objectives

- Cruise speed
- Turning radius
- Takeoff characteristics
    - Takeoff speed
    - Takeoff time
    - Takeoff acceleration
- Stability

Performance characteristics were predicted based on the design. At the same time a sensor package was designed using an Arduino Uno based system to measure the inertial and airspeed properties of the plane in flight. The sensor package was then put inside the plane during flight tests to gather data. Once the data was gathered, it was analyzed to interpret the results into flight characteristics which could be compared to the predicted values.

The first section of the report will cover the background information and scope of the project. During the second section, there is a review of the technical analysis. During the third section, the sensor package will be reviewed starting with the hardware selection and development of the code that the Arduino ran. During the fourth section, the results will be presented with raw data interpreted and analyzed. The fifth section will cover the statistical and error analysis. Conclusions from the data including comparisons between analytical and experimental and recommendations for next steps will conclude this thesis.

Background information

Modern commercial airplanes are outfitted with a range of sensors that monitor pressure, temperature, stress, accelerations, rotations, and positional data [1].These sensors all help to give the aircraft and operators an idea of how it is performing at that moment. Data gathered from these sensors are crucial to the autonomous functions of aircraft control as well as monitoring the health of the aircraft. The data gathered in commercial airplanes is much more extensive than what would be feasible in this thesis, therefore a set of testable flight characteristics was chosen. However, to do that it is first imperative to understand the purpose of the plane which is to be analyzed.

The DBF team was tasked with designing two planes for the 2016 DBF competition: a manufacturing plane and a production airplane. The airplane that was analyzed in this thesis was designed as the manufacturing plane in the competition. The manufacturing plane (M-plane) had to carry the subassemblies of the production plane (P-plane), not including the battery or payload of the P-plane. More points were awarded for being able to store the P-plane as fewer pieces inside the M-plane; therefore, the team elected to store the P-plane as one piece inside of the M-plane. The design choice to store the P-plane as one piece presented unique design constraints in the size constraints of the M-

plane. An example of the design constraints was the airfoil selection of the M-plane. This will be discussed in more detail in the design process.

One of the two rules that affected aerodynamic performance was the requirement of a maximum takeoff distance of 100 ft. The maximum takeoff distance was tested in Corvallis, OR. However, the environment at the competition location, in Wichita, KS had much higher wind speeds than Corvallis. The higher wind speed allowed the maximum takeoff distance shown at Corvallis to be higher than 100 ft. It also caused the team to require a minimum cruise speed of at least 8.94 m/s (20 mph). A minimal turning radius at cruise speed was also desirable because of its impact on the flight time. If the plane took longer to turn, the flight time would increase which would increase the required minimum battery capacity.

Aerodynamic performance is important for a fast and efficient airplane, however, if the airplane is not stable it won't fly no matter how efficient the design is. Stability is one area for which it is difficult to develop engineering requirements. The base requirement is that the plane must be statically stable. The wings of the plane were also designed to withstand the wing tip test. This is the second rule which directly affected the aerodynamic performance of the plane. The wing tip test simulates a 2.5 g wing loading which is a typical loading experienced by the wings during a steep turn. The wing tip test is performed by supporting the plane by the wingtips when testing the center of gravity, which was chosen to be ¼ chord of the wing from the leading edge to minimize the influence of the wing on the stability.

Design Process

Now that customer requirements and engineering specifications of the plane were defined, the sizing of the parts was calculated using Matlab. The plane already had a defined boom length and minimum cord based on the sizing constraints imposed by the P-plane. The code follows the logic diagram in Figure 1

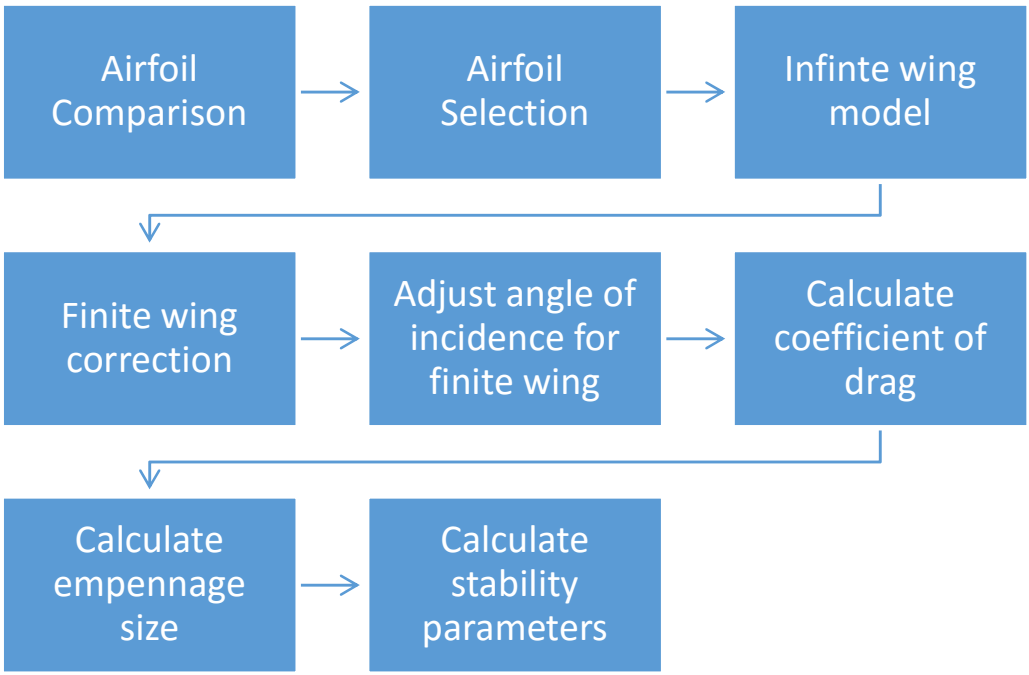

Figure 1: Design process for the M-plane.

The fuselage shape for the M-plane was already defined because it was based on the P-plane. An airfoil comparison was completed to choose the most efficient airfoil. Six NACA airfoils (1410, 2412, 2408, 23012, 23112, 24112 and 63(2)-215) were chosen based on their low coefficients of lift and large size. The lower coefficients of lift allowed for a larger wing area which helps to increase the span for a set chord length. The larger airfoil size made it easier to accommodate the P-plane wing and helped to minimize the M-plane wing chord length. The NACA 2412 was chosen because it was the most efficient and had a maximum efficiency which occurred at a lower angle of attack than the other airfoils. A lower angle of attack is beneficial because the angle of incidence of the M-plane must be close to that of the P-plane to minimize the chord of the M-plane wing.

All the data for the airfoils was taken from the Airfoil Tools website [2], which draws data from Xfoil. Xfoil is a reputable source for calculating infinite wing coefficients, and uses the panel method to determine the pressure distribution over the airfoil for lift, drag and pitching. The cruising condition for an infinite wing using the NACA 2412 was defined at an angle of attack of zero degrees.

Once the airfoil was chosen, the area of the infinite wing was then defined. The minimum chord length required to accommodate the P-plane wing was used to increase the aspect ratio. The higher aspect ratio decreased the drag, decreased the load on the batteries and motor, and allowed the team to design a lighter system.

A taper ratio of 0.826 was chosen because it increased the effective aspect ratio further without decreasing the tip Reynolds number below 150,000 during takeoff. A more significant taper decreased the induced drag and keeping the Reynolds number above 150,000 limited the possibility of the wing undergoing transition to turbulent flow while in the air as well as keeping the skin friction drag to a minimum [3]. Then the wing was corrected for its finite parameters by altering the slope of the coefficient of lift curve. To compensate for the loss of lift between a finite wing and an infinite wing the angle of incidence was adjusted to 1.5 degrees. The adjusted angle of incidence also let the team push the wing into its more efficient state to get a higher lift over drag ratio. The coefficient of drag was also corrected for the induced drag and the take-off speed was calculated based on the new coefficient of lift for the finite wing at 8.5 degrees. This completed the lift calculations and stability characteristics were then calculated.

Stability

While stability is an important consideration when designing an airplane, it is difficult to characterize in a physical manner. In full scale airplanes, this is typically accomplished by a stick-force curve which correlates the effort required to fly at different speeds and center of gravity locations [3]. However, a

full analytical simulation of the aircraft stability is beyond the scope of this thesis and the static stability derivatives are acceptable substitutes of performance estimations. The longitudinal derivative is typically expressed as a static margin which is a percent of the wing chord length.

Drag

The drag of the airplane was calculated by an aggregate approach where the coefficient of drag of each component is tabulated and expressed in terms of a common reference area. In this case it is the reference area of the wing. Referencing a common area enables one to sum all of the individual coefficients to create a single coefficient of drag for the entire airplane. The base parasitic drag data of the wings were already given from Xfoil, which was then adjusted for the induced drag from lift on a finite wing. The wing was treated as the only body which had lift dependent terms in this analysis. A comparison showing the contribution to drag of each major component at cruise is shown in Figure 2.

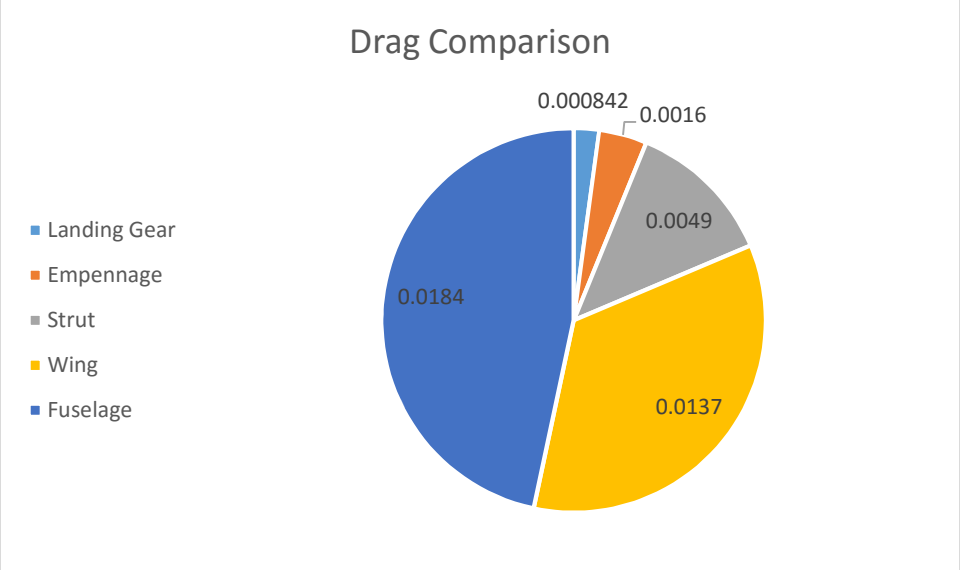

Figure 2: Drag comparison for the manufacturing plane broken down into major components. This is for the cruise condition. The numbers shown here are the coefficients of drag (C_d).

The overall aircraft coefficient of drag is then easily converted into the thrust required for steady flight at the specified cruise speed. The minimum amount of thrust to overcome the drag during takeoff is also calculated and is defined at an angle of attack of seven degrees (the wing will be at 8.5 degrees). It is important to note that while this is the minimum thrust required to reach takeoff speed, the actual thrust required for takeoff is much higher. This is because of the takeoff distance limitation of 100 ft. at Wichita, KS.

Handling

The aircraft score was penalized in the competition for adding weight to the airplane. Therefore, there were compromises in efficiency for the sake of a simple and light design. One of these was the controlled surfaces on the wing. The airplane had flaperons, which are a combination of high lift devices (flaps) and control surfaces (ailerons). The flaperons were a simple flap design with hidden hinge in the suction side of the airfoil.

Technical Analysis

Once the design of the airplane was specified, the next of getting performance predictions could begin. These were based on technical analysis performed in Matlab. The analysis makes several assumptions:

- No wind
- Airplane is being flown close to limits of performance
- Takeoff angle is nine degrees
- Takeoff is done at maximum power
- Maximum thrust is constant through takeoff and is the same for handling maneuvers
- Runway is flat and level
- Aircraft is flown at level flight (i.e. not in dive nor ascent)

Handling

The turning radius was calculated as a constant radius turn where there was no loss in altitude of the aircraft. The requirement of level flight (not losing altitude) limited the banking angle and was defined by the maximum coefficient of lift of the airplane during turn, which was initially estimated to be: 0.945. The bank angle can be related to maximum lift by the following equation:

$$\emptyset = \cos^{-1}\left(\frac{2 * m * g}{C_l * \rho * v^2 * SA}\right)$$

From there the turning radius can be found by the following equation:

$$R = \frac{W/_{SA}}{0.5 * \rho * g * C_l * \sin\emptyset}$$

However, at cruise speed at the maximum coefficient of lift, the aircraft would be generating 90 N of lift which is more than 4 times the aircraft's weight. This goes beyond the structural characteristics for which the wings were designed. Furthermore, the drag on the plane at that angle of attack at cruise speed increased to almost 9.8 N. Static thrust is typically the maximum thrust for an electric motor [4], and the drag at maximum lift and cruise speed was close the designed static thrust of the airplane (11.8 N). Therefore, the turning performance was defined by the 2.5g loading which is tested by the wing tip test.

Takeoff Simulation

The takeoff simulation was done for level ground and assuming that the maximum angle of attack on the runway was 7.5 degrees. The pilot is not to takeoff before the takeoff speed is reached and therefore the takeoff will be one continuous maneuver.

The landing gear on the M-plane is a tricycle formation resulting in a different takeoff sequence than one with a tail-dragger configuration. A tricycle landing gear lets the aircraft sit at zero degrees (or close

to it) angle of attack on the runway which helps it to accelerate faster due to an optimal thrust vector of the motor and lower drag. When the plane reaches takeoff speed, the pilot then pitches the nose up, increasing its angle of attack and lifts the airplane off of the runway. The thrust used for this analysis is based on standard flight conditions. While most batteries have a slight surge in power the first time that they are discharged, this factor is ignored in the technical analysis due to the difficulty in modelling it without extensive battery testing.

The takeoff simulation was done in multiple steps. The first step is the time up until the plane reaches takeoff speed on the runway. It is represented by the following differential equations.

$$A_y = \frac{Lift}{m} - g$$

$$A_x = \frac{Thrust - Drag - Friction}{m}$$

In the above equations, g is the acceleration due to gravity, m is the airplane mass, and $A_{x,y}$ is the aircraft acceleration. Equations above are two 2nd order coupled differential equations. They are coupled because the lift has a direct effect on the rolling resistance. The two forces which are resisting the thrust from the motor, drag and friction are cause by aerodynamic forces and the friction in the wheels respectively. During an actual takeoff, there would be two more steps to the takeoff. One, when the aircraft pitches up to leave the ground and then second, when the aircraft transitions from climb and levels off as the speed increases. These last two steps were not performed in this calculation due to the unknown pitch rate during takeoff. A maximum pitch rate could be used for this estimation, but based on the previous flight tests this would not be accurate. The pitch rate normally flown by the pilot was much gradual to avoid stall and have the minimal takeoff distance. Table 1 shows the expected performance for the plane.

Table 1: Detailed testable performance parameters for the M-plane.

Parameter	Value
Cruise Speed (m/s)	17.7 - 18.5
Takeoff Speed (m/s)	11.5
Static Longitudinal Stability	21.72%
Turning Radius (m)	18.534
Radial Acceleration (m/s^2)	21.56

Sensor Package

The purpose of the sensor package is to capture the necessary data during the flight of the plane to be able to calculate the flight parameters identified in the technical analysis. A list of requirements was developed for the sensor package:

- Must be small enough to fit inside the plane
- Must be light enough so as to not impede the flight ability of the plane
- Must be able to operate as a physical closed system
- Must be able to run for at least 10 minutes
- Must be able to record data at a high frequency
- Must have a physical filtering system to limit the noise recorded in the raw data

The first three requirements are dealing with the system being inside a small moving object (i.e an UAV). The system cannot require a physical attachment to a computer or outside device (such as power supply) for operation; otherwise, the plane would not be able to move. On full scale airplanes, the first

two requirements are not as important since their size is insignificant compared to the airplane. On a small scale airplane, the electronics represent a much larger portion of the weight and size of the airplane, therefore it can be a limiting factor. Most of the prior flight tests for the plane were less than nine minutes. From testing it was revealed that the airplane only needs a little over one minute to complete one lap of the competition course. However, a requirement of ten minutes allows for a large amount of leeway in the flight time in case the flight does not start when anticipated. The last two requirements are to ensure that the data which is recorded is useful. Useful, meaning that it is not too noisy nor is it too low of a frequency to be filtered. A higher sampling rate will experience less errors in the measurement, especially when it is integrated.

Physical Components

An Arduino Uno was selected to serve as the computer for the sensor package. Arduinos have a large selection of sensors from which to choose, their online support is exemplary, and they require very little power to operate. A low power device is key for a low system weight, as most of the weight is from the battery. Cost was also a consideration when making this decision since the entire system was self-funded. While cost and flexibility are both great benefits for a do-it-yourself solution such as the Arduino, it does require some basic coding knowledge. The coding knowledge required was one of the challenges in this project, and it will be discussed later in the text.

The Arduino had to record data without a direct computer connection. To solve this problem, an SD card shield was used to connect the Arduino to an SD card. Another possible option was having a wireless connection sending data to the computer, however, an SD card is a more robust solution for data recording.

At first, a standard 9V battery was used as the power source for the Arduino on account of its accessibility, low weight and low cost. It worked for a few short runs, however, it soon proved to be

unreliable as the voltage was not well controlled. For example, the 5V out pin which gave power to the pitot tube was supplying more than 5V. This rendered the data from the pitot tube unusable.

The Arduino has two main ways that it can be powered, through the DC barrel jack or through the USB port. The DC battery power can also run through the Ground and Vin headers of the power connector on the Arduino. Power which runs through the barrel jack port also goes through an onboard regulator. The purpose of the regulator is to condition the power input from 9V to a standard 5V electrical signal. The acceptable input range for this port is 7-12V. Through testing it was discovered that if the Arduino was powered through the USB port by a computer then the system acted normally and was very stable. Therefore, the goal was to bypass the DC barrel jack port on the Arduino. However, the power which comes through the USB port does not go through a regulator, and must already be conditioned to 5V. To solve this problem, another power source was required. A 2200 mAhr Lithium Polymer (LiPo) battery was used instead of the 9V battery. The LiPo battery has an onboard regulator to ensure that the output is always 5V, so it can power the Arduino through the USB port. Other benefits of this battery include: it is lasts much longer, and is rechargeable so it did not have to be replaced and each run could start off with a fully or nearly fully charged battery, which ensures that each test is run under similar conditions.

A button and LED were also tested with the system which would replace the delay function on the Arduino, so that the Arduino could be started from the exterior of the aircraft. Every time that the data was saved to the SD card the Arduino would send power to the red LED. This served two purposes: it signaled the user when the Arduino was recording, and it served as a warning notification if the data had failed to write to the SD card. In preliminary testing the system worked, however, when the wires were soldered together the system bypassed the button and started before it was pressed. The button start was discarded after that, but the code is included as comments in Arduino code appendix.

Sensors

Given the requirements in the previous section, it was chosen that the sensor package should have a gyroscope, accelerometer, and pitot tube sensors. Together, the gyroscope and accelerometer make an inertial measurement unit (IMU). The Adafruit 10-dof breakout board was selected for this purpose. It has an accelerometer and magnetometer(LSM303DLHC), gyroscope (L3DG20H), as well as a barometric/temperature sensor (BMP180). The axis directions for this sensor are shown in Figure 3.

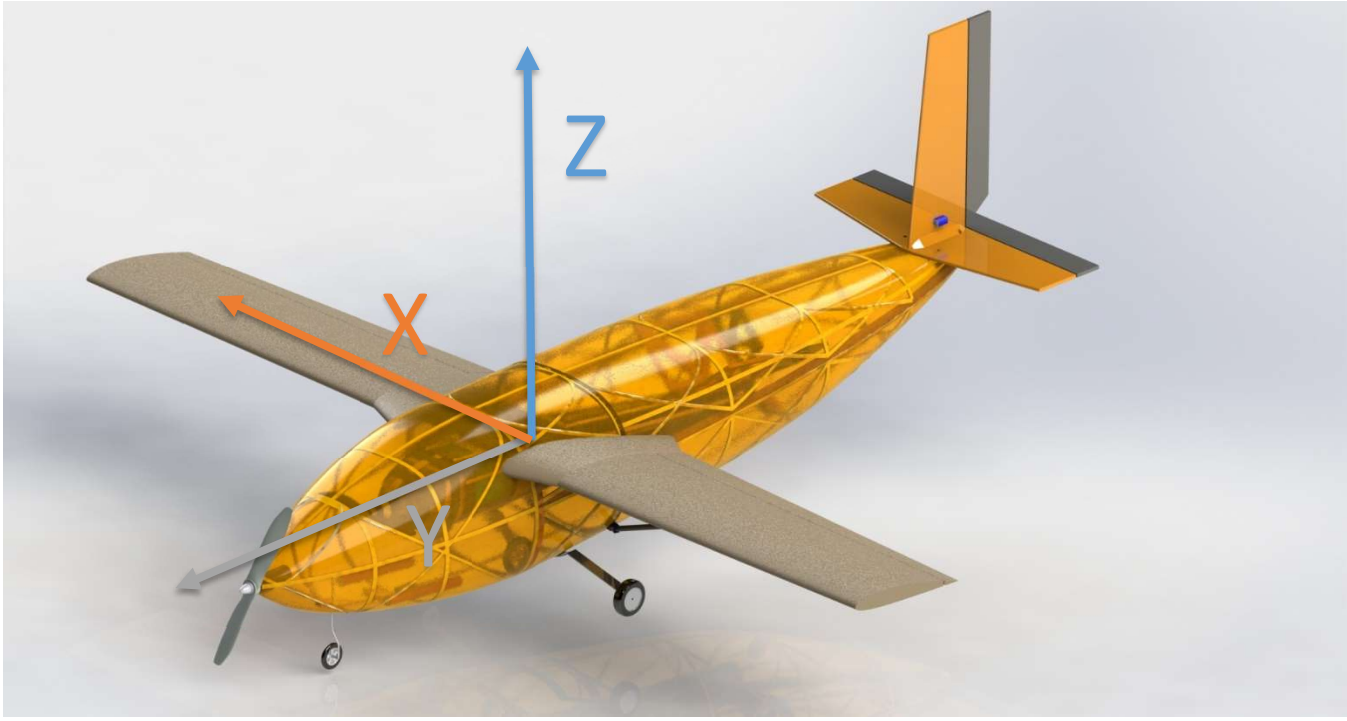

Figure 3*:* Sensor axis as shown on the M-plane

The accelerometer has a user selectable range from +/- 2 g to +/-16 g, as well as a user selectable refresh rate up to 400 hz. This airplane was designed to experience 2.5 g (24.5 m/s^2) during a steady turn. However, the maximum accelerations could exceed this in due to noise, or windy conditions. Given that the instantaneous accelerations might exceed predictable values; the accelerometer was coded to have a +/- 16 g range. The gyroscope has a range from +/-250 deg/s to +/- 2000 deg/s, but unlike the accelerometer it will automatically select the range based on the initial calibration. The rotation rates

experienced were much lower than 250 deg/s, so the gyroscope automatically selected the lowest range (+/- 250 deg/s). The magnetometer has a user selectable range of +/- 1.3 gauss to +/- 8.1 gauss. The +/- 1.3 range is used for typical conditions for determining the direction of the magnetic field. The breakout board communicates with the Arduino through an I2C connection. This allows complex data to be sent from multiple sensors with just 4 wires.

The pitot tube measures the airspeed of the airplane which can also be used to determine the turning radius if the accelerations are known. The pitot tube relies on a MPXV7002 analog pressure transducer. It has a differential pressure range of +/- 2 kPa. This gives a theoretical maximum airspeed range of 0 - 57.14 m/s (127.5 mph). Unlike the 10 degree of freedom breakout board, the pitot tube is an analog sensor, meaning that it only has one output as a voltage range. The voltage has to then be converted into a meaningful number, in this case airspeed which requires calibration.

Pitot Tube Calibration

The pitot tube was calibrated two different ways: one in which the pitot tube is only exposed to clean air, and another where the pitot tube is setup on a cutaway of the wing. The wing section was used to mimic the flow conditions that the pitot tube would see during the flight test. The pitot tube was setup centered on the wing spanwise, with the end of the pitot tubes two inches (5.08 cm) in front of the leading edge of the wing.

The test stands were then put in a wind tunnel with a pre-calibrated wind speed measurement system. The pitot tube was tested at speeds up to 60 mph (26.82 m/s) in increments of 5 mph (2.23 m/s) starting at 10 mph (4.47 m/s). The lower limit was defined by the accuracy of the wind tunnel. The wind speed is very unstable below speeds of 10 mph. The upper limit was defined by approximately 50 % greater than the cruise speed to account for any sudden gusts. 12 data points were taken for each wind speed to reduce the precision error in the calibration process.

System Damping

The first test with the sensor package was run on the outside of the plane, and with the sensor directly attached to the body of the plane. The mounting location for the first test was due to the limitations imposed by the fuselage of the plane initially tested. The data was expected to be noisy, and the accelerometer readings were completely unusable and filled the entire +/- 2 g scale that was initially selected. The results from the first test are also what led to the final selection of a +/- 16 g range. The gyroscope data did not exhibit excessive noise and a clear trend could be seen during turns.

During the next tests the noise was reduced by the use of physical dampers. A major factor in noise in an aircraft is the motor vibration. For an electric motor which is rotating at 8000 rpm, the forcing frequency is quite high (133 hz). Therefore, the goal of the damping system was to reduce the response to high frequency vibrations. A low natural frequency will decrease the response to high frequency vibrations, while a higher damping ratio will increase the time constant, or the time it takes to reach 63.2% of its steady state value. The natural frequency of a system is determined by the below equation:

$$\omega_n = \sqrt{k_{eq}/m_{eq}}$$

Where k_{eq} is the equivalent stiffness, and m_{eq} is the equivalent mass of the system. Therefore, the natural frequency can be decreased by either decreasing the spring stiffness or by increasing the mass. The sensor itself is very light at 3.5 g. As mentioned in the prior section, one of the requirements was that the system be lightweight. Therefore, significant weight should not be added to the system just for the purpose of damping. The sensor and Arduino were combined to increase the effective mass of the sensor which helped to lower the natural frequency. A polyurethane foam sponge was chosen for its low stiffness. The foam has a density of 16 kg/m^2 and a spring constant of 144 N/m. Combined with the Arduino system mass of 88.5g, the natural frequency of the system is 40 Hz.

Wiring

The sensors were connected to the Arduino with a custom made wiring harness to simplify the setup (each cable is labelled and only has one location where it can go), decrease weight, and make an overall cleaner device. As the Arduino is limited in the number of power ports, the pitot tube and IMU had to share a 5V power port. Figure 4 shows the wiring schematic.

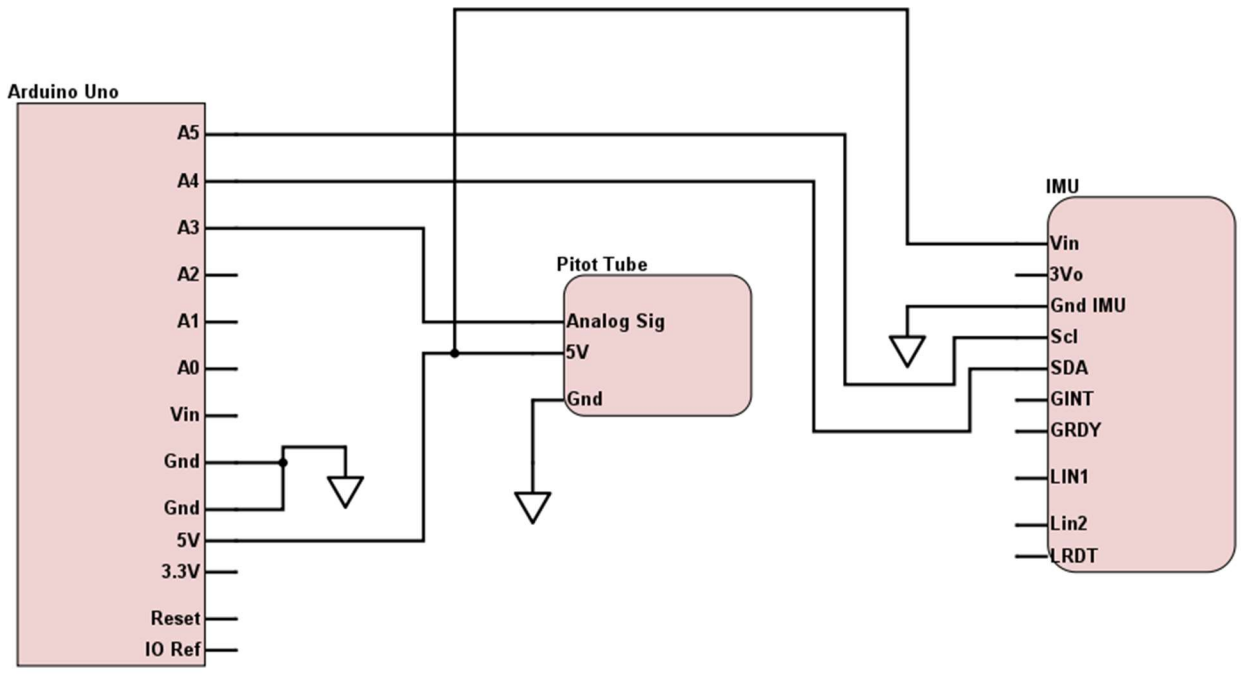

Figure 4: Wiring schematic for the Arduino, IMU and pitot tube. The ground wires are connected together, but the wiring was abbreviated with the arrow to avoid cluttering the schematic.

Software

The code that was running on the Arduino was built for speed to decrease the recording error. However, the speed at which the code runs is not just a factor of its efficiency, but also of the capability of the hardware. This presents a problem for the Arduino, because it is a low power device. For reference, the

Arduino Uno runs at 16 Mhz which is about 100 times slower than the clock speed of each core of a smart phone [4].

Therefore, all data processing was offloaded to a Matlab program, and the Arduino was used entirely for raw data collection. A logic diagram of the Arduino program is show below in Figure 5.

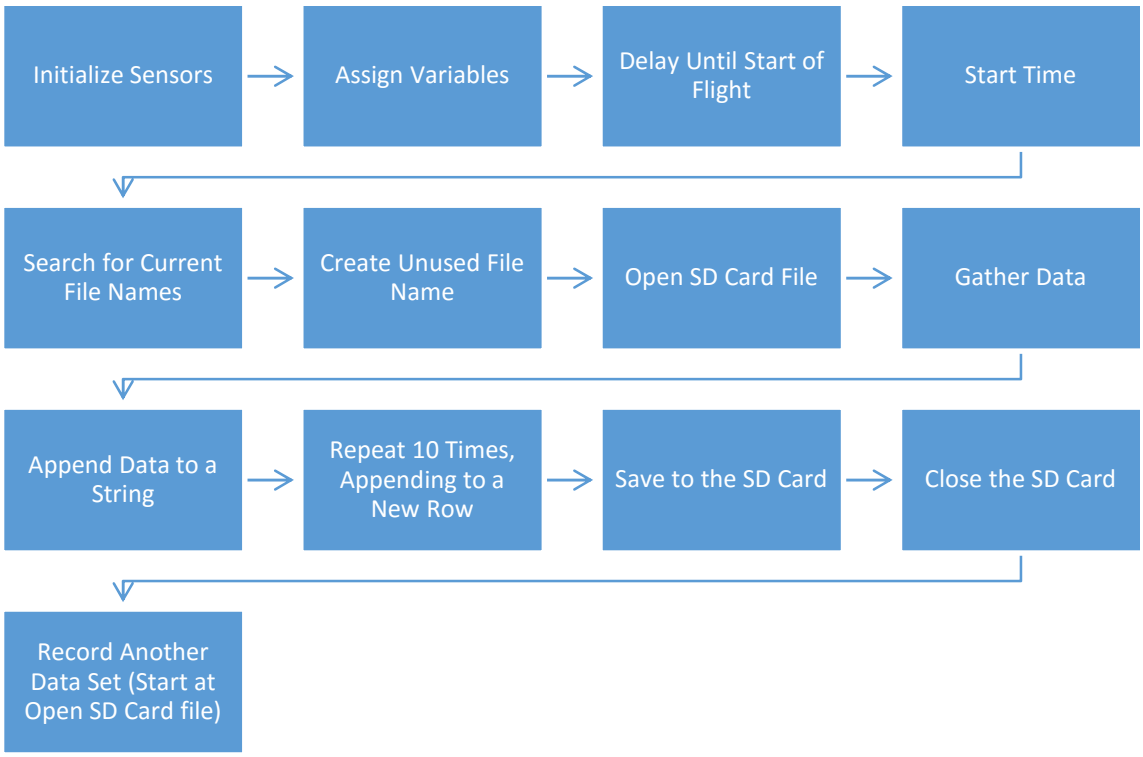

Figure 5: Logic Diagram for the Arduino code. It is a loop function which takes 10 data points in between every save to the SD card. This is one of the tools used to increase the sampling rate of the Arduino.

The accelerometer and gyroscope both rely on a unified driver which is referenced in the code by an include statement. The driver communicated with the sensors and converted the digital signals to SI units automatically. The time at each data measurement was also recorded with the internal clock on the Arduino. The pitot tube measurement voltage was collected using an AnalogRead command. The 11 data points (nine from the IMU, one from the pitot tube, and then time) are then compiled in a string.

During the primary design of the system, the data string was then immediately saved to a .csv file on the SD card. The code was simple, and it had minimal data loss in case of a crash landing since only one line of data would be lost. During testing, this yielded a sampling rate of 10 Hz. From testing and experimentation, it was revealed that the act of opening and closing the SD card file represented the largest use of processor power and was slowing down the Arduino. The code was adjusted so that it read the sensors ten times, saving the data to the data string, before appending it to the file and saving it. Further improvements were made in the code by changing the variable types for basic data, such as time. Saving the sensor data directly to the data string instead of having an intermediate variable also helped to improve the sampling rate since it was found that the original program was using more than 75% of the onboard memory of the Arduino. This increased the sampling rate to 135 Hz. The one drawback of the new logic flow for the code was the non-uniform time step between data points. Originally, each data point was around 100 ms apart, but by saving the data every 10 data points the time step would either be five or 20 ms depending on if it was saving the data or just collecting it in the data string. The non-uniform time step effected the way that the data was analyzed, because one could not use simple methods which assume a constant time step.

Results

This section will be used to go over the results from the flight tests. There are three main categories in which data was collected: takeoff, turning, and stability. The specific flight characteristics which were tested are listed below:

- Cruise speed
- Turning radius

- Takeoff characteristics
    - Takeoff speed
    - Takeoff time
    - Takeoff acceleration
- Stability

Three flight test had successful data recording, however, the data from the first flight exhibited a low frequency (1 hz) high amplitude oscillation in the z direction accelerometer data. This anomaly was not repeatable, and the root cause was never identified. It might be due to a magnetic field produced by poor battery shielding, because that flight test used home built Nickel-Metal-Hydride batteries to power the motor. These batteries were the same that were used for competition. The other two flight tests used LiPo batteries to power the motor, because they last longer and are able to give a more constant power output when they are discharged. The second test was run with a three cell, 4000 mAhr battery that ran at a lower voltage of 11.1 V (3.7V per cell). The three cell battery was thought to give enough power to get controllable flight, but it was windier than expected and the flight had to be cut short. There was some good data that was gathered from the flight as all of the sensors were acting nominally. The third flight was setup with a large 5000 mAhr, four cell battery. The higher power capability of the four cell battery gave the plane a very long run time and allowed all of the previous tests to be run as well as some tests for longitudinal stability.

It is important to note the conditions for flight during these tests. While the tests were performed in the best circumstances available, they could not be done in dead wind (i.e. zero wind) conditions. The difference between no wind and a nine mph crosswind can introduce errors and noise into the data. The pilot would wait until the wind stabilized to begin a test, but even then errors arose.

Each flight test was recorded on video so that the period of time for each of the maneuvers discussed in this section could be drawn out of the data. The video and data were synced up by blowing into the pitot tube, this was uniquely matched up with the audio from the video file.

Before any calculations with the data were performed, the data was first filtered. The accelerometer and gyroscope data were filtered with a 6^{th} order low pass Butterworth filter with a cutoff frequency of 10 hz. Unlike the accelerometer and gyroscope, which are MEMS devices, the pitot tube is a large physical sensor which impacts its practical response time. This impacted the fast fourier transform of the pitot tube data as the frequencies were much lower than the accelerometer. Therefore, the pitot tube was filtered using a cutoff frequency of 3.7 hs.

Takeoff

The acceleration data is adjusted for the change in pitch and roll, but not yaw. Therefore, the data shown is the accelerations in the x-y plane. By not compensating for yaw, if the airplane does turn in yaw, the forward acceleration due to thrust will be easier to read.

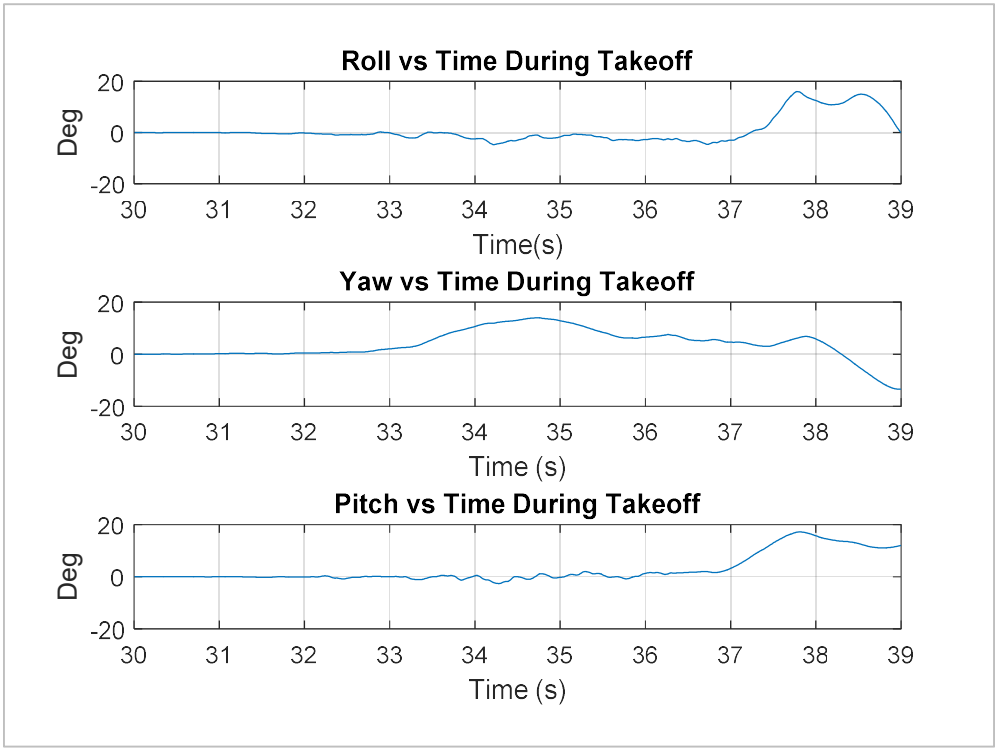

Figure 6: Rotations of the plane during takeoff of flight 1.

Figure 6 shows the accelerations during takeoff of flight one. This data is corrected for rotations by the gyroscope and is rotated into the x-y plane so to help the interpretation. At 37 seconds into the recording, the airplane pitches up to increase the coefficient of lift and takeoff. The plane does exhibit a change in the yaw direction while on the runway. The roughness in the runway can be seen by the slight variations in the pitch and roll plots between times of 31.5 and 37 seconds.

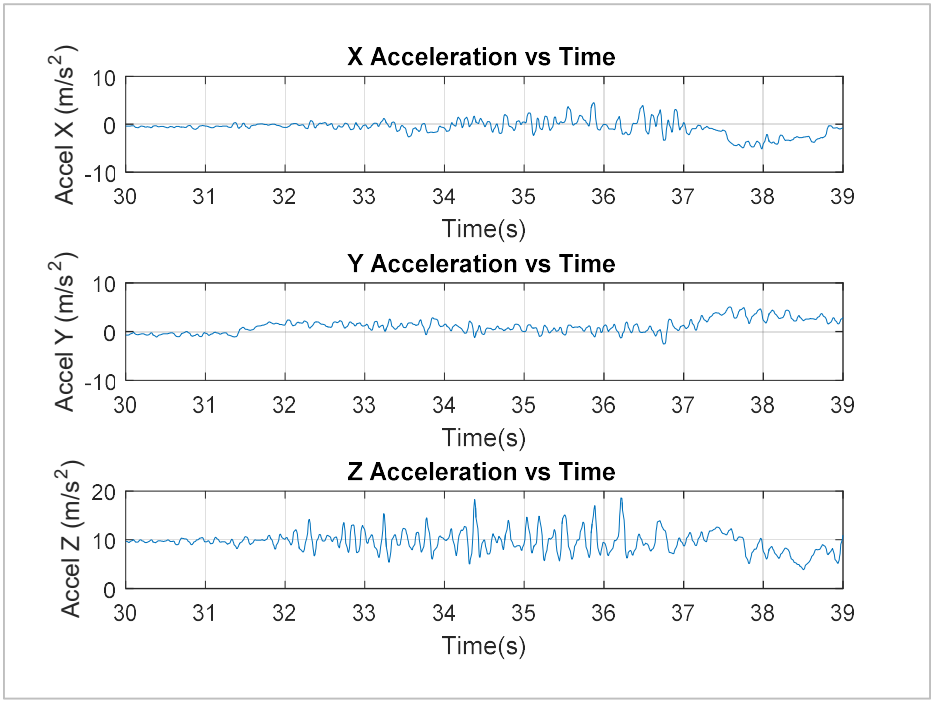

Figure 7: Rotated body accelerations expressed in the X-Y plane of the ground for flight one.

Takeoff starts at 31 seconds and liftoff is at just over 37 seconds.

Figure 7 shows the rotated acceleration data for takeoff during flight one. The takeoff starts at 31.5 seconds, and the plane exhibits maximum forward acceleration on the runway of 1.9 m/s^2 at just before 32 seconds into the recording. There is a sharp spike just after leaving the runway where the forward acceleration increases to 4.2 m/s^2 at 38.28 seconds.

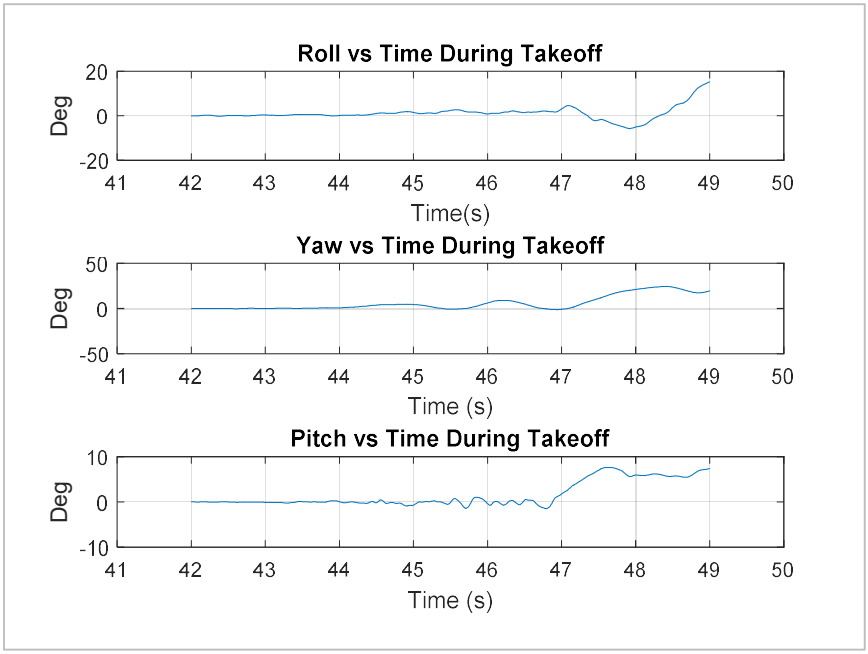

Figure 8: Roll, pitch and yaw of the airplane during the takeoff of the second flight.

Figure 8 shows the rotations of the body during takeoff of the second flight. Takeoff begins at 43.5 seconds and lift off is at just over 47 seconds. The airplane was much more stable on the runway during this takeoff when compared to flight one. The pitch was also much more stable. The increase in stability can be contributed to the increase in power which allowed the pilot to have more control at lower speeds. A higher static thrust will push more airflow over the control surfaces at low speeds which increases the effect of the control surfaces. The aircraft can be seen to stabilize at about seven degrees for takeoff when climbing. This matches the designed angle of attack for takeoff.

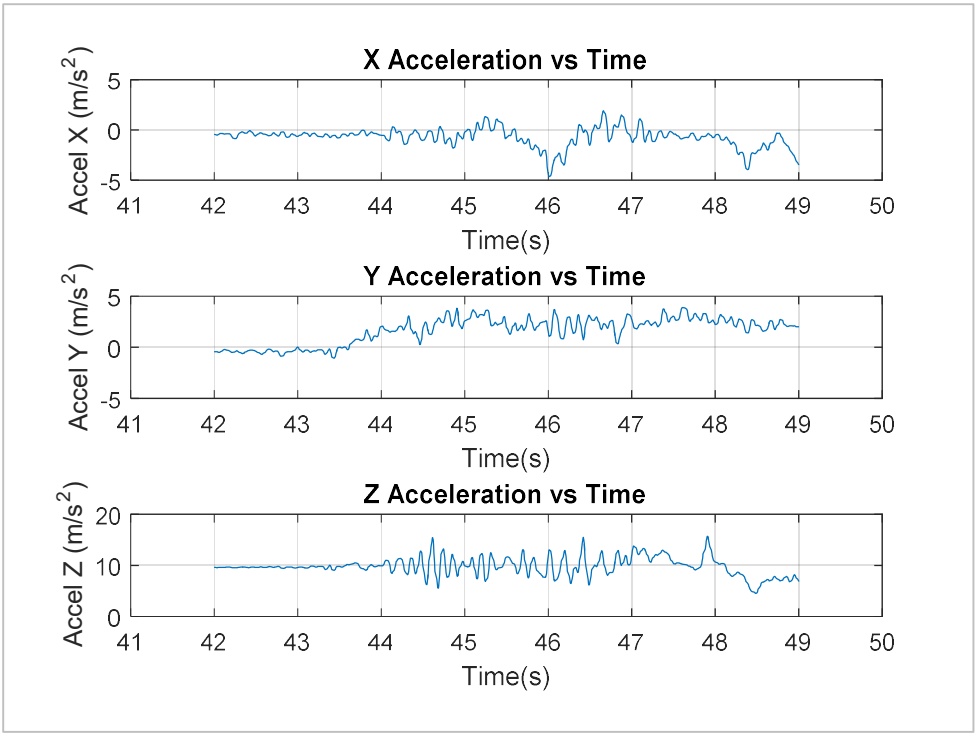

Figure 9: Airplane accelerations rotated into the plane parallel to the ground during takeoff.

Figure 9 shows the acceleration of the airplane during takeoff of flight two. One can see that the takeoff starts at roughly 43.5 seconds into the recording, as that is when the y (forward) acceleration starts to increase. There is a spool up period during the first few seconds when the acceleration is still increasing. The spool up period can be decreased by holding onto the back of the plane while the motor starts to spin, and then releasing it. At about 45.7 seconds, there is a sideways drift, which is most likely caused by a slight gust. 3.5 seconds after the motor starts spinning, the plane is airborne, which is shown as an increase in the acceleration in the z direction. At about the same time, there is a slight increase in the forward acceleration. A possible explanation for this is the drop off of rolling resistance in the wheels. The maximum forward acceleration is 3.9 m/s^2 at 47.8 seconds.

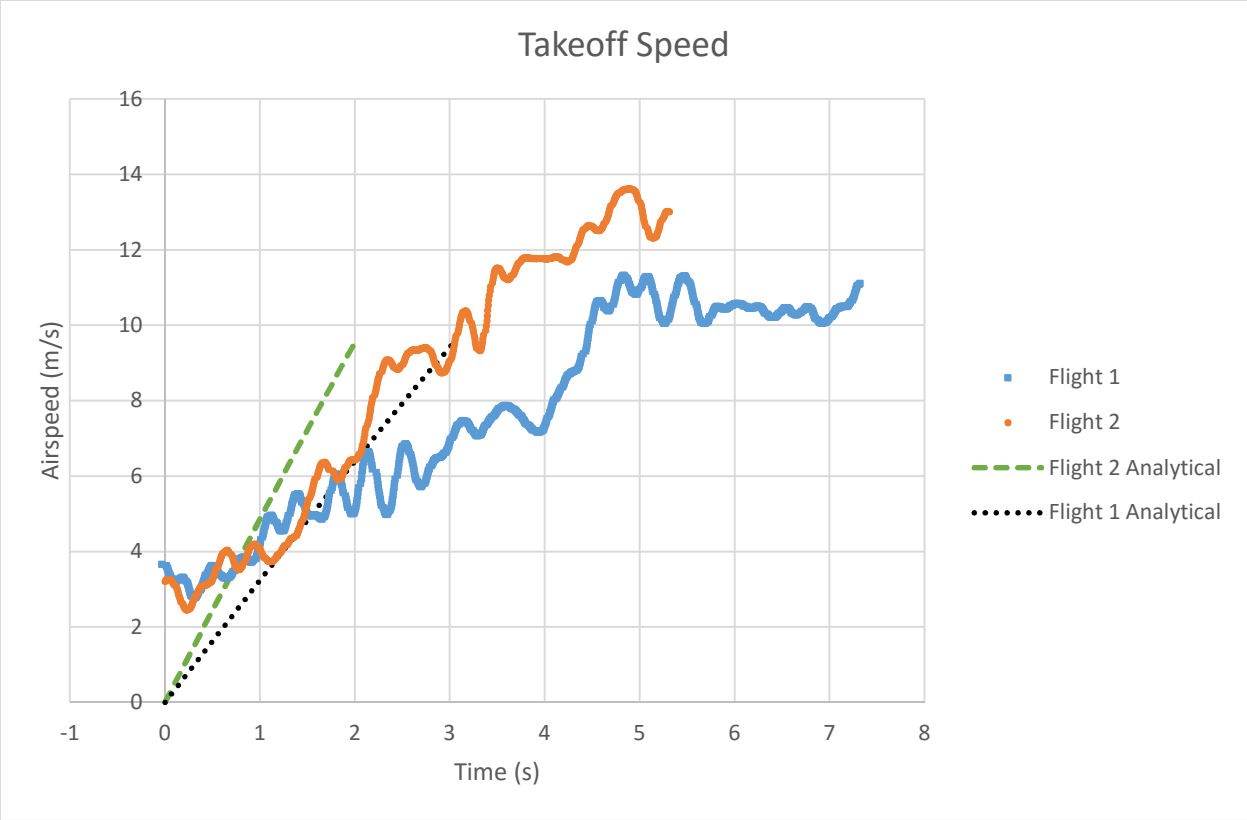

Figure 10: Air speed during takeoff for both flights.

Figure 10 shows the airspeed variation during takeoff as calculated by the pitot tube. It is also compared to the theoretical values for flight one and flight two. The difference between the two flights is the battery. The first battery was a three cell Li-Po battery at 3.7 V per cell and a maximum discharge rate of 80 amps total. Static thrust values were derived from ecalc [6] and through static thrust testing. While static thrust testing was not conducted for every battery option, the values from ecalc were compared with these options to those which were thrust tested for an experimental validation of ecalc's results. The expected static thrust for this combination was 800g (7.84 N). The next flight was done with a four cell Li-Po battery also at 3.7 V per cell and a maximum discharge of 125 Amps. While the maximum amperage is much higher for this battery, the max amperage overall was not much higher as it was limited by the esc at 30 amps. From ecalc and thrust testing this battery/prop/motor combination gave 50 % more thrust and was rated at 1.2 kg (11.76 N) of static thrust. The plot is misleading at low speeds

because the best fit curve for the pitot tube is not accurate at speeds below 4 m/s (8.9 mph). The deviation from the best fit will be discussed more in depth in the pitot tube calibration section.

Turning

During the flights, the pilot was instructed to do several long complete turns. The starting condition for each turn was level flight with the wind at a minimum. The turns did take some time and sometimes there would be a gust of wind halfway through the turn. The wind would push the aircraft around and make the IMU data more difficult to interpret, therefore these turns were disregarding from the results.

The first flight, while short and underpowered did give a very steady turn which could be analyzed. The accelerometer data for the turns was also rotated into the x-y plane of the airfield with yaw being untouched. The yaw data was not adjusted so that the gravitational component of the accelerometer reading could be separated and the radial acceleration calculated for the turns.

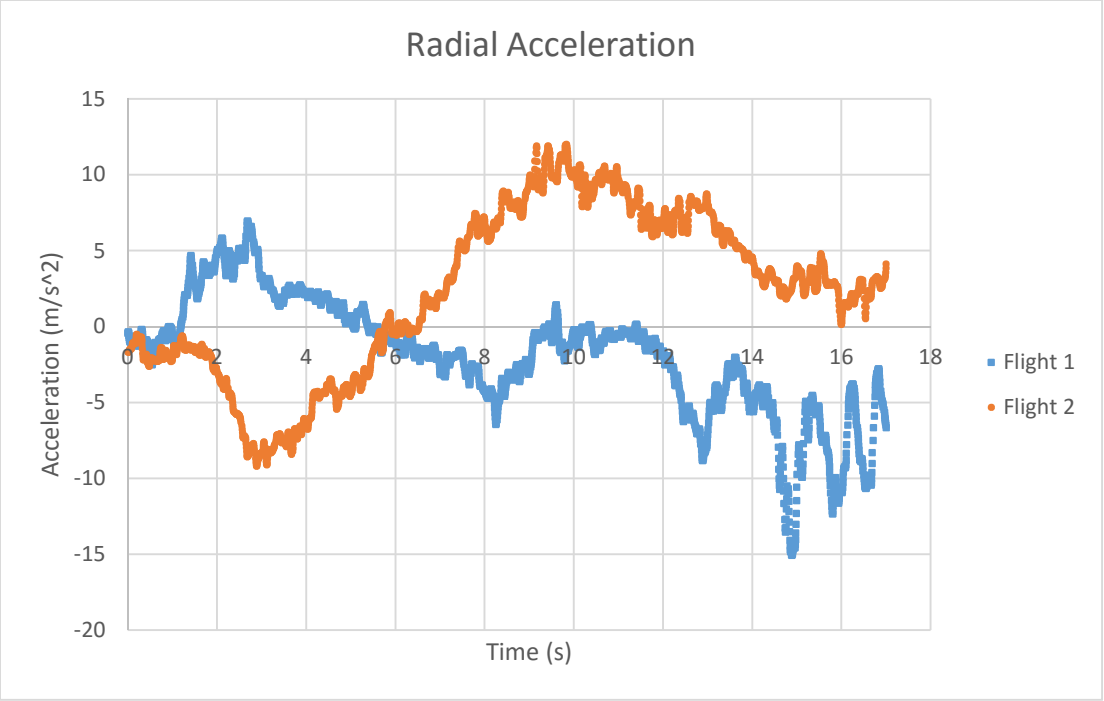

Figure 11: Radial acceleration during turn for flight one and two. This is assuming that the airplane is undergoing level flight.

Figure 11 shows the radial acceleration of the plane during the turn for both flights. The turn during flight one was a counterclockwise turn, while the turn during flight two was counterclockwise. There was a slight crosswind at the beginning portion of flight one. This explains the initial change in radial acceleration which is opposite from the direction one would expect from each turn. The turn during the second flight initially began with an adverse roll (away from the center of the turn) before rolling into the turn. This produced the trend which is seen in Figure 11 where the radial acceleration switches sign. The acceleration during flight two is much more consistent than that of flight one. This can be expected because of the lower thrust for flight one.

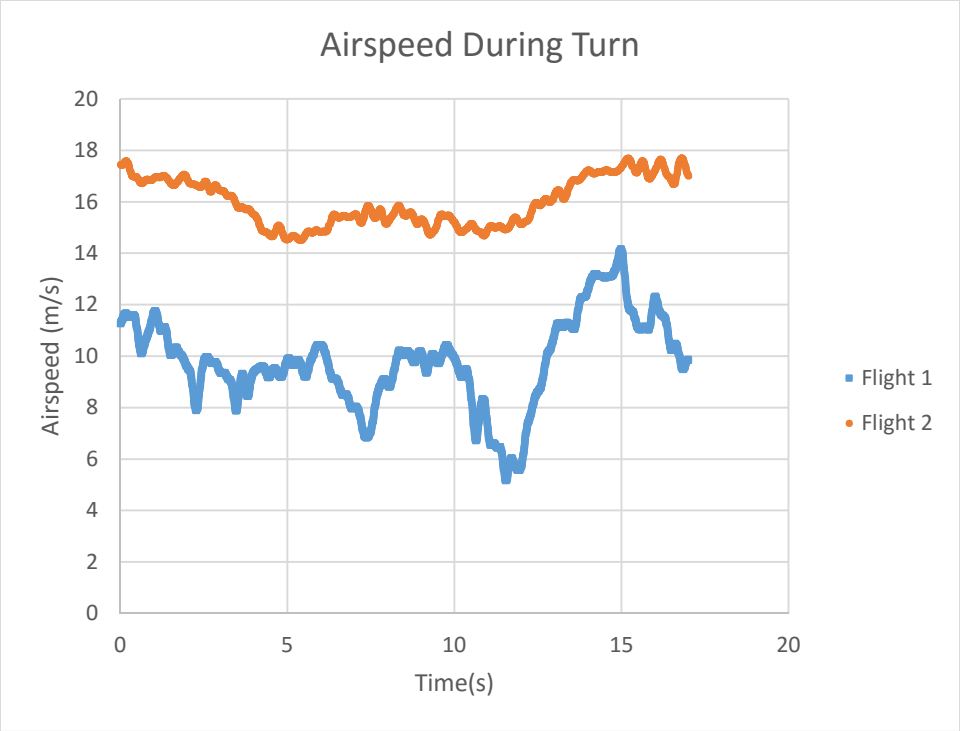

Figure 12: Air speed during the turn for flight one and two.

Figure 12 shows the airspeed during each of the turns. The turn during the second flight was fairly steady. The airspeed reading from the pitot tube did drop a bit from 17.8 m/s to 14.9 m/s, however, part of this can be attributed to the angle of the pitot tube. During a turn, the airflow over the pitot tube will not be from straight on due to the rotation of the plane and the higher angle of attack. The first flight is much more unstable. This is most likely due to the low power which made the aircraft difficult to control and led to erratic plane movements. Previous flight tests have shown that it is more difficult to control an aircraft which is underpowered. This is especially true for turns, because they require more power than cruise conditions. Given the maximum radial acceleration of 10 m/s^2 and an average turning airspeed of 14.9 m/s, the calculated turning radius is 22 m for flight two.

Perturbations

The airplane was also tested for longitudinal stability. This was done using a “flick test” where the pilot pushes the airplane nose up with full elevator and then takes their hands off of the controls to see how quickly it returns to its original position.

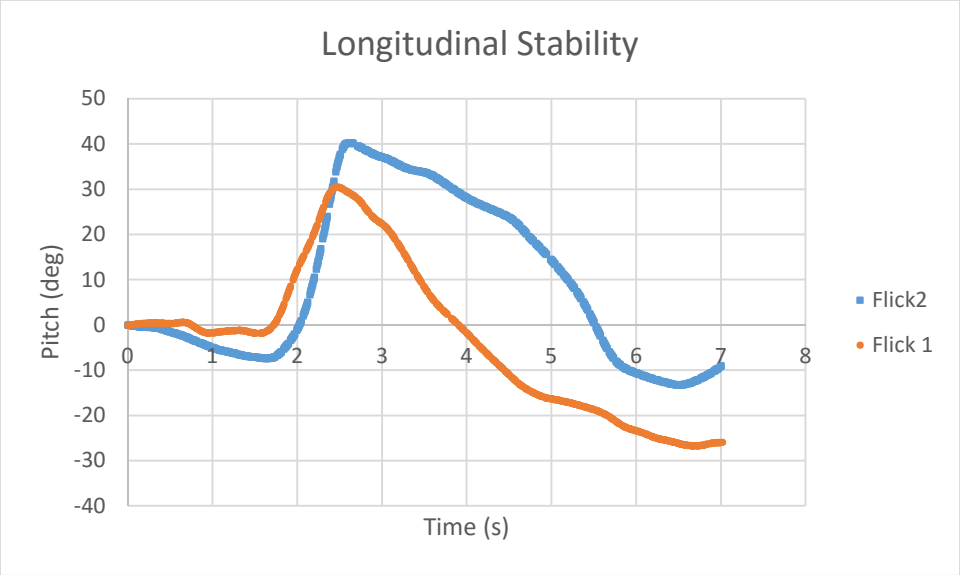

Figure 13: Pitch of the airplane during two tests of longitudinal stability.

Figure 13 shows the change in pitch during two tests of the perturbation response. During the first test, the full elevator stick input began at time 1.7 seconds and lasted until 2.4 seconds. During the next few seconds, the plane did not have any inputs at all. Therefore, any change in the movement of the plane is due to its inherent aerodynamics. The M-plane has a calculated longitudinal stability margin of 21.71 %. This is demonstrated by the down pitching moment right after the elevator is let go. The video that was taken during the flight corroborates this evidence, and four seconds after the controls were released the plane had settled down and was no longer pitching. The maximum pitch rate during the first potion of the flick test was 50 deg/s, while the maximum pitch rate for returning to stable was 31.5 deg/s in the opposite direction. During the second test, the full elevator stick input began at time 1.7 seconds and

lasted until 2.5 seconds. The maximum pitch rate during the first portion of the flick test was 100 deg/s. The maximum pitch rate for returning to stable was 40 deg/s in the opposite direction.

Uncertainty Analysis

Since only two flights were analyzed, there was not enough data to generate a statistical model of the results. Therefore, the uncertainty analysis contains only the design stage uncertainty and does not take into account statistical uncertainty. The pitot tube and IMU have a different procedure for the uncertainty analysis.

IMU

The IMU did not need to be calibrated as it is a digital sensor and arrived pre-calibrated. The uncertainty in the data is from the bias error as there were not enough data points to develop a large sample size. The accelerometer was set to a range of +/- 16 g, and the gyroscope was auto range set to +/- 250 deg/s. The sensor has a 16-bit data output which gives a possible 65,536 steps in resolution. However, some of the full scale is not represented due to calibrations. Unlike the accelerometer, the gyroscope has linearity error which is the deviation of the actual results from the straight best fit line. The linearity error accounts for most of the error in the gyroscope, however, typically this is only significant near the ends of the full scale. The design stage uncertainty of the IMU is summarized in Table 2:

Table 2: Design Stage Uncertainty of IMU

	Accelerometer (m/s^2)	**Gyroscope (deg/s)**
Base Unit	0.06	0.50
Percentage of FS	0.04%	0.20%

Pitot Tube Calibration

The Arduino Uno has a 10-bit analog to digital converter which means that the voltage input (which reads from 0-5V) is converted to a number from 1-1025 which is referred to as the Arduino units. This is the number that is recorded by the Arduino on the SD card. It is important to note that while the scale on the Arduino goes from 1-1025, a value of 1 is not zero airspeed. The voltage from the pitot tube is normalized at half of the full scale, plus an offset voltage, meaning that a reading of zero airspeed should be around 530 Arduino units.

There is one limitation in the calibration of the pitot tube. Since the pitot tube is an analog sensor it was imperative that the conditions in the wind tunnel be as close to the conditions on the plane as possible. This means that the power source must be the same, which ruled out the option of connecting the computer to the Arduino for instantaneous voltage readouts. The data was instead recorded on the SD card during the same time period that the data points for the wind tunnel were taken. While the data points were taken during the same time period and flow conditions one cannot match up individual points. Instead the averages of the points were taken and that is what was compared.

The pitot tube was calibrated two ways: one with the clean air and one where the pitot tube was mounted on the wing to simulate the flight test environment.

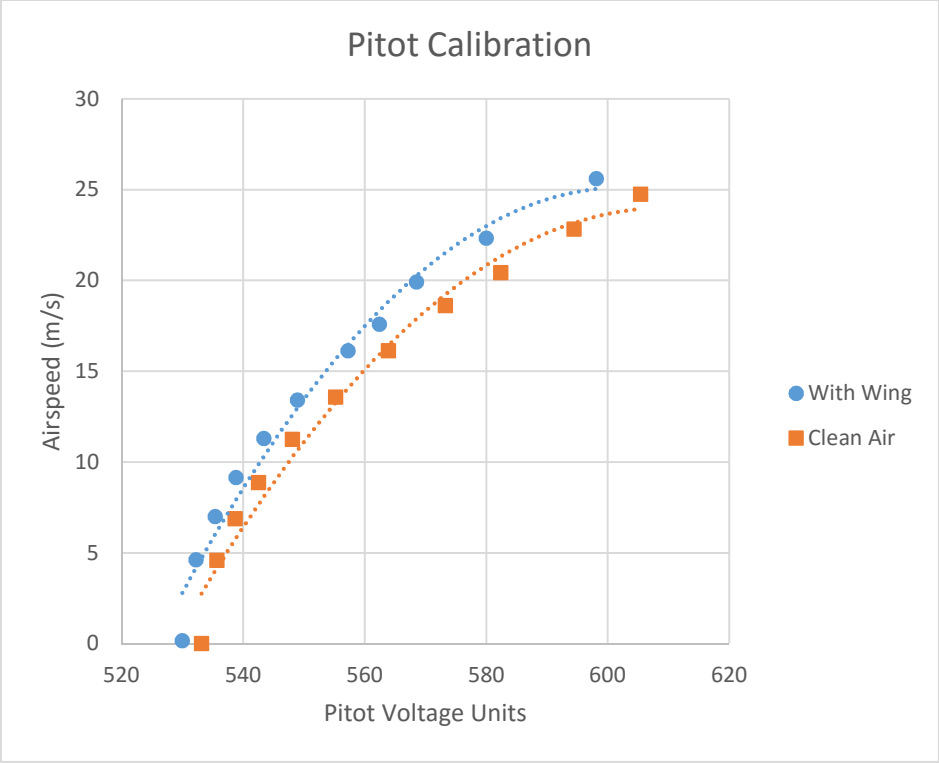

Figure 14: Arduino voltage units for each airspeed recording from the wind tunnel.

Figure 14 shows the calibration curve for both the clean air and wing mounted conditions. The slopes are similar, but the clean air has an increasingly higher voltage reading from the pitot tube. This is to be expected as the wing should slow the air in front of it, which would lead to a lower differential pressure reading from the pitot tube. The pitot tube has a full scale span of 4.5 volts. From the data sheet [7] this gives an accuracy of 0.11 volts. The 10 bit analog to digital converter has a discretization error. With a 10-bit encoder the analog to digital converter can measure as low as 0.0049 volts. This makes the total error of the pitot tube voltage a range from 0.12 – 0.26 volts with the error increasing with the airspeed which yields a maximum airspeed error of 0.067 m/s.

Conclusions/Recommendations

While there were many analyses done on the data that was gathered, the data is not 100% conclusive. To be able to say that the data is 100 % conclusive would require many more tests during similar

conditions to get a statistically significant amount of sample sets for each test performed during the flight tests. In the meantime, any conclusions drawn from this data should be considered as what the data is suggesting.

The airspeed calculations matched well with the experimental data with a cruise speed of 17 m/s for flight two. This compares to the design cruise speed of 18.5 m/s. The takeoff speed at 9.5 m/s was also close to what was designed for (11.5 m/s) which is most likely accounted for by the thrust from the motor helping to pull the plane up and effectively lowering the required lift by the wings. The takeoff angle of 7.68 degrees is also very close to what was designed (7 degrees). The turning radius, while larger than expected was offset by the decrease in the airspeed during the turn. The decrease in the airspeed during the turn contributed to a significantly smaller maximum radial acceleration than what was expected.

The perturbation tests revealed that the plane was statically stable, however, it was heavily affected by the direction of the wind, and would alter course like a weather vane. Both times, the aircraft settled to nose down attitude relative to where it began, which might be a result of a trim issue in the elevator. This limited the length of time that the flick tests could be run. Therefore, conclusions about dynamic stability cannot be made.

There is room for improvement in the design of the sensor package. The first limitation of this system is that it relied entirely on an IMU for position and body information. While that is accurate in the short term, these sensors often experience drift and a buildup of error if integrated. The error buildup could be resolved by incorporating a GPS sensor into the package. Decreasing the error buildup would help to give a more accurate depiction of the turning radius of the plane as well as give land speed information. The land speed could be combined with the data from the pitot tube to determine the effect of the wind at precise points in time.

While the attempt to integrate a switch into the Arduino system failed upon soldering, it remains a worthwhile endeavor. Correctly done, a switch will let the Arduino Uno be turned on from outside of the plane. The alternate solution of having a delay built into the program was a limitation that always slowed the schedule of the flight test. A switch might also save a few SD cards as the Arduino would be able to record just the amount of data that one needs. Several of the SD cards that were used got corrupted during testing, possibly because of the amount of times that the card is accessed during each flight test.

While the damping system was adequate it can be improved. A better system would be to put a damping/spring material on the top and the bottom of the sensor package instead of just the bottom. This would yield more reliable results as it would compensate for the difference in tensile and compressive stiffness in the material.

A lot was learned about the challenges one faces when designing airplanes. The challenge does not stop when the initial design is finished. Testing is critical and figuring out how to reliably test design parameters can be as difficult as it was to come up with the initial design.

References

[1] D. Bobkoff, "Why every flight you take is obsessively monitored", *Business Insider*, 2016. [Online]. Available: http://www.businessinsider.com/airplanes-and-big-data-sensors-2015-6. [Accessed: 18- Mar- 2016].

[2] "Airfoil Tools", *Airfoiltools.com*, 2016. [Online]. Available: http://airfoiltools.com/. [Accessed: 15- Sep- 2015].

[3] Kroo, "Aircraft Design, Synthesis and Analysis", *Adg.stanford.edu*, 2016. [Online]. Available: http://adg.stanford.edu/aa241/AircraftDesign.html. [Accessed: 15- Mar- 2015].

[4] A. Brezina and S. Thomas, "Measurement of Static and Dynamic Performance of Small Electric Propulsion Systems", Wright State University, Dayton, OH.

[5] "Samsung S7", *Samsung Electronics America*, 2016. [Online]. Available: http://www.samsung.com/us/mobile/cell-phones/SM-G930VZDAVZW. [Accessed: 10- May- 2016].

[6] "eCalc - the most reliable RC Calculator on the Web", *Ecalc.ch*, 2016. [Online]. Available: http://www.ecalc.ch/. [Accessed: 07- Nov- 2015].

[7] Freescale Semiconductor Inc., "Datasheet: MPXV7002 Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated", 2015.

[8] ST microelectronics, "Data Sheet: MEMS motion sensor: three-axis digital output gyroscope", 2013.

[9] ST Microelectronics, "Data Sheet: Ultra compact high performance e-compass 3D accelerometer and 3D magnetometer module", 2011.

[10] MATLAB and Statistics Toolbox Release 2015b, The MathWorks, Inc., Natick, Massachusetts, United States.

Appendix

Airplane Design Code

```
%Created by Jacob Scheer
%e-mail jacobscheer2008@hotmail.com

%This code is used for calculating all of the aerodynamic parameters of
% the M-plane for the 2016 DBF competition
%This is a very generic code. It can easily be remolded for any type of
% airplane.

clc
clear
clf
close all


%% Waitbar parameters
countwait=0;
counterwait=0;
%hwait=waitbar(counterwait,'MAKING AN AIRPLANE...');
TOTALWAIT=4617;
%% Define base parameters
rho=1.225;
g=9.81;
ReC=68459;
mu=1.846*10^(-5); %dynamic viscosity of air

length_to_nosecone=.127;
e=0.95;
V_cruise=20;

C=340.29;
M=V_cruise/C;
Thrust = 1.5; % thrust of motor in kg

m=2; %estimate need to update, will probably increase
%% Tail parameters
C_ht=0.6;  %volumetric tail coefficient horizontal
C_vt=0.09; %volumetric tail coefficient vertical
L_ht=0.9144; %length of boom from centers of lift
L_vt=L_ht;
% L_tail=0.1331; %length of each component of the V-tail
% Depth_tail=0.1664; %depth of the tail
```

```
%% Drag Parameters
%fuslage
S_wetf=1.25; %Wetted surface area of fuselage
Lf=1.524; %length of fuselage
Df=0.2143; %4 inch tube

%Landing gear
Cds=0.2; %Assume turbulent flow and elliptical rod for strut
Ss=[8.71/2*10^-4 8.71/2*10^-4 7.10*10^-5]; %main landing gear followed by
rear (frontal surface area)
dg=[0.0381 0.0381 0.0312]; %diameter of the wheel
wg=[0.003175 0.003175 0.0074];  %width of the wheels
Cdlg=0.30; % is 0.15 if there is a fairing around wheel
for i=1:length(dg)
    Slg(i)=dg(i)*wg(i);  %frontal surface areas of the wheels
end

% struts for wing

Lstrut=1;
Cdstrutu=.3;
Dstrut=0.00635;
Astrut=Lstrut*Dstrut;

%% Stability Parameters
Volume_fus=0.0398; %fuselage volume (need to check this number)

W_fus=0.2143; %Width of fuselage (need to double check)
D_fus=W_fus; %circular cross section
Z_wf=D_fus/2;%wing mounted on top
x_cg=0;
x_acw=0;




%% Get data from excel file about airfoil
filename='Airfoilsdata-M.xlsx';sheet=[1 2 3 4 5 6]; %location of excel file
%data for Re=200k
%As a side note. Putting it in basic mode makes it run much faster 2
%seconds vs 30 seconds.

promptstart = 'COMPARE AIRFOILS? 1 = YES; 0 = NO \n';
fprintf('\n')
answerstart=input(promptstart);

if answerstart == 1
    startfor=1;
    endfor=6;

elseif answerstart == 0
    promptchoose = 'WHICH AIRFOIL? (1410; 2412; 2408; 23012; 23112; 24112
63(2)-215 <-- type 63215) \n';
```

```
    fprintf('\n')
    answerchoose=input(promptchoose);



    if answerchoose == 1410
        startfor=1;
        endfor=1;
    elseif answerchoose == 2412
        startfor=2;
        endfor=2;
        % elseif answerchoose == 2408
        %     startfor=7;
        %     endfor=7;  Really low range of AoA ~10
    elseif answerchoose == 23012
        startfor=3;
        endfor =3 ;
    elseif answerchoose ==23113
        startfor=4;
        endfor=4;
    elseif answerchoose ==24112
        startfor=5;
        endfor=5;
    elseif answerchoose == 63215
        startfor=6;
        endfor=6;
    else
        fprintf('INPUT INVALID AIRFOIL NUMBER\n');
    end
end

for i=startfor:endfor
    if answerstart == 1

        alpha_200k(:,i)=xlsread(filename,sheet(i),'A45:A103','basic'); %angle
of attack vector
        C_L_200k_inf(:,i)=xlsread(filename, sheet(i), 'B45:B103','basic');
%infinite wing Cl
        C_D_200k_inf(:,i)=xlsread(filename, sheet(i), 'C45:C103','basic');
%infinite wing CD induced
        C_D_p_200k_inf(:,i)=xlsread(filename, sheet(i), 'D45:D103','basic');
%infinite wing CD parasitic
        CM_200k_inf(:,i)=xlsread(filename, sheet(i), 'E45:E103','basic');
%pitching moment 2d wing
        Top_Xtr_200k(:,i)=xlsread(filename, sheet(i), 'F45:F103','basic');
%transition to turbulence top surface
        Bot_Xtr_200k(:,i)=xlsread(filename, sheet(i), 'G45:G103','basic');
%transition to turbulence bottom surface

        %Best line approx


        ZOOMalpha(:,i)=alpha_200k(18:38,i);
        ZOOMC_l(:,i)=C_L_200k_inf(18:38,i);
```

```
        coeffs = polyfit(ZOOMalpha(:,i), ZOOMC_l(:,i), 1);

        % Get fitted values
        lsline_alpha = linspace(min(ZOOMalpha(:,i)), max(ZOOMalpha(:,i)),
132);
        lsline_Cl = polyval(coeffs, lsline_alpha);
        a_0(i) = (lsline_Cl(5)-lsline_Cl(1))/(lsline_alpha(5)-
lsline_alpha(1)); %slope of best fit line
        Yintercept_2D(i)=lsline_Cl(5)-(lsline_alpha(5)*a_0(i)); %Y-intercept
of best fit line for infinite wing
        Xintercept(i)=interp1(lsline_Cl,lsline_alpha,0); %X-intercept of best
fit line


    elseif answerstart == 0
        alpha_200k=xlsread(filename,sheet(i),'A45:A103','basic'); %angle of
attack vector
        C_L_200k_inf=xlsread(filename, sheet(i), 'B45:B103','basic');
%infinite wing Cl
        C_D_200k_inf=xlsread(filename, sheet(i), 'C45:C103','basic');
%infinite wing CD induced
        C_D_p_200k_inf=xlsread(filename, sheet(i), 'D45:D103','basic');
%infinite wing CD parasitic
        CM_200k_inf=xlsread(filename, sheet(i), 'E45:E103','basic');
%pitching moment 2d wing
        Top_Xtr_200k=xlsread(filename, sheet(i), 'F45:F103','basic');
%transition to turbulence top surface
        Bot_Xtr_200k=xlsread(filename, sheet(i), 'G45:G103','basic');
%transition to turbulence bottom surface

        %Best line approx


        ZOOMalpha=alpha_200k(1:58);
        ZOOMC_l=C_L_200k_inf(1:58);




        coeffs = polyfit(ZOOMalpha, ZOOMC_l, 1);

        % Get fitted values
        lsline_alpha = linspace(min(ZOOMalpha), max(ZOOMalpha), 132);
        lsline_Cl = polyval(coeffs, lsline_alpha);
        a_0 = (lsline_Cl(5)-lsline_Cl(1))/(lsline_alpha(5)-lsline_alpha(1));
%slope of best fit line
        Yintercept_2D=lsline_Cl(5)-(lsline_alpha(5)*a_0); %Y-intercept of
best fit line for infinite wing
        Xintercept=interp1(lsline_Cl,lsline_alpha,0); %X-intercept of best
fit line
    end
```

```
end

if answerstart == 1
    for j=startfor:endfor
        for i=1:length(C_D_200k_inf(:,j)) %calculate cl/cd for infinte wing
            CLCD_inf(i,j)=C_L_200k_inf(i,j)/C_D_200k_inf(i,j);
        end
    end

elseif answerstart == 0
    alphaend=10;
    alpha=[-3.25:0.25:alphaend];
    AoAindex=find(alpha_200k==0);
    %find the cruising C_L
    C_Lcruise=C_L_200k_inf(AoAindex);
    SA=2*m*g/(rho*V_cruise^2*C_Lcruise);
    SA=SA*1.4; %factor of safety of 1.4
    %define the chord length
%     c=11.5*2.54/100;%11.5in to m
%     b=SA/c;
%     AR=b/c;

AR = 7;
c = 10.5*2.54/100;
b = 27*2*2.54/100;

    %now define the angle of incidence

    %% finite wing correction

    n=length(alpha);

    % CL correction
    a=a_0/(1+57.3*a_0/(pi*e*AR)); % find the correct slope, 57.3 is to
convert between 1/degrees and 1/radians
    Y_intercept3D=-Xintercept*a; %The zero lift angle of attack is a constant

    np=length(alpha); %define the resolution of the new alpha vector
    for i=1:length(alpha)
        C_L(i)=a*(alpha(i)-Xintercept);
    end

    [Minimum,AoAindex]=min(abs(C_L-C_Lcruise));
    AngleIncidence=alpha(AoAindex);

    startfinite=find(alpha_200k==alpha(1));
    endfinite=find(alpha_200k==alpha(end));

    %normalize the vectors to match the prior defined range
    CDalpha=C_D_200k_inf(startfinite:endfinite);
    CMalpha=CM_200k_inf(startfinite:endfinite);

    for i=1:length(alpha) %calcuate finite cd
        C_D(i)=CDalpha(i)+C_L(i)^2/(pi*e*AR);
```

```
end

%% TAKEOFF SPEED
indexalphatakeoff=find(alpha==9);
C_L_takeoff=C_L(indexalphatakeoff);
V_takeoff=sqrt((2*m*g)/(rho*C_L_takeoff*SA));
%% Tail calculations

S_ht=C_ht*c*SA/L_ht;
S_vt=C_vt*b*SA/L_vt;

% for initial design
%   AR_tail=AR*.8;
%   c_tailh=sqrt(S_ht/AR_tail);
%   b_tailh=S_ht/c_tailh;
%   c_tailv=c_tailh; %set the chord lengths to be the same
%   b_tailv=S_vt/c_tailv;
%   AR_tailv=b_tailv/c_tailv;
%   Sv=S_ht+S_vt;



c_tailh = 0.143;
c_tailv = c_tailh;
b_tailh = 0.5255;
b_tailv = 0.3622;

AR_tailv=b_tailv/c_tailv;
Sv=S_ht+S_vt;
AR_tail = 3.76;

%% Drag of Various components
%fuselage
C_Dcruisew=C_D(AoAindex); %wing drag at cruise
Re=rho*V_cruise*Lf/mu;
Cf=0.455/((log10(Re))^2.58);
Fld=1+60/((Lf/Df)^3)+0.0025*(Lf/Df);
Fm=1-0.08*M^1.45;
Cdof=Cf*Fld*Fm*S_wetf/SA;

%landing gear
Cdos=0;
Cdow=0;
for i=1:length(Ss)
    Cdos=Cdos+Cds*Ss(i)/SA;%struts
    Cdow=Cdow+Cdlg*Slg(i)/SA; %wheels
end

%empinage
Cdtail=0.005;
Cdtail=Cdtail*Sv/SA;

%wing struts
Cdstrut=Cdstrutu*Astrut/SA;
```

```
Cdelse=Cdtail+Cdow+Cdos+Cdof+Cdstrut; %Cd of items other than the wing
%separate the drag of various components
CD_0=C_D;
F_0=0.5*V_takeoff^2*SA*rho*CD_0;
CD_1=C_D+Cdof;
F_1=.5*V_takeoff^2*SA*rho*CD_1;
CD_2=CD_1+Cdtail;
F_2=0.5*V_takeoff^2*SA*rho*CD_2;
CD_3=CD_2+Cdow+Cdos;
F_3=0.5*V_takeoff^2*SA*rho*CD_3;

C_D=C_D+Cdelse;

CLCD=zeros(length(alpha),1);
CLCD_inf=zeros(length(C_D_200k_inf),1);
%calculate cl/cd for finite wing
for i=1:length(alpha)
    CLCD(i)=C_L(i)/C_D(i);
end


for i=1:length(C_D_200k_inf) %calculate cl/cd for infinte wing
    CLCD_inf(i)=C_L_200k_inf(i)/C_D_200k_inf(i);
end


for i=1:length(alpha)
    Thrust_cruise(i)=rho*SA*.5*C_D(i)*V_cruise^2;
end



for i=1:length(alpha)
    Thrust_takeoff(i)=rho*SA*.5*C_D(i)*V_takeoff^2;
end
AoA_takeoffi=find(alpha==9);
T_takeoff=Thrust_takeoff(AoA_takeoffi); %Drag at 9 degrees
L_D=CLCD(AoAindex);



%% Turning radius
m = [2.2 2.4];

maxCL=max(C_L); %factor of safety of 1.4
for i = 1:2
theta=acos((2*m(i)*g)/(maxCL*rho*V_cruise^2*SA));
Rmin(i)=(m(i)/SA)/(0.5*rho*maxCL*sin(theta));
theta_d=theta*180/pi;
accell_g(i)=V_cruise^2/(Rmin(i)*g);
end
```

```
C_Lm=.5;
accell_gs=[1.1:0.1:2.5];
C_LM=zeros(length(accell_gs),1);
% find CL for different g loadings
for i=1:length(accell_gs)
    for j=1:200
        R(i)=V_cruise^2/(accell_gs(i)*g);
        theta=acos(2*m*g/(C_Lm*rho*V_cruise^2*SA));
        C_Lm=(m/SA)/(.5*rho*R(i)*sin(theta));

    end
    C_LM(i)=C_Lm;
    Loading(i) = C_LM(i)*.5*V_cruise^2*SA*rho;
end
[minimum, Loadingi] = min(abs(Loading-2.5*9.8*2.2));

R_chosen = R(Loadingi);
Loading_chosen = Loading(Loadingi);
accel_gchosen = accell_gs(Loadingi);
%close(hwait)




%% Tail lift calculations
a_0_tail=2*pi/57.3; %conversion between /rad to /deg

a_tail=a_0_tail/(1+57.3*a_0_tail/(pi*e*AR_tail));
%research shows that a low aspect ratio plate stalls at about 20 degrees
%research/energies. But let's plot it to 30 degrees so that we have the
%data
alpha_tail=[0:0.01:30];
alpha_tailw=[0:2:20];
x_intercept_tail=0; %flat plate
C_L_tailw=a_tail*alpha_tailw; %done for wind tunnel calcs
C_L_tail=a_tail*alpha_tail;
Force_tail=zeros(length(alpha_tail),2);
V=[V_takeoff,V_cruise];

a_tailv=a_0_tail/(1+57.3*a_0_tail/(pi*e*AR_tailv));

for i=1:length(V)
    for j=1:length(alpha_tail)
        Force_tail(j,i)=.5*C_L_tail(j)*V(i)^2*S_ht*rho;

    end
end
```

```
    %% Lateral (YAW) Static Stability
    Xac=L_ht/b; %Assume tha the cg is at quarter chord of wing
    lamda=0; %Wing sweep is zero
    lat_dir=0.724+(3.06*S_vt/SA)/(1+cos(lamda))-0.4*Z_wf/D_fus+0.009*AR;
%Lateral directional derivative for angles of attack and effectiveness
    C_Ls=[0.25 .9]; %Range of Cl for the wing (cruise and takeoff).
    C_nbw=C_Ls.^2.*(1/(4*pi*AR)-
(tan(lamda)/(pi*AR*(AR+4*cos(AR))))*(cos(lamda)-AR/2-
AR^2/(8*cos(lamda))+(6*Xac*sin(lamda))/AR)); %Wing contribution
    C_nbfus=-1.3*Volume_fus*D_fus/(SA*b*W_fus); %Fuselage contribution
    C_nbv=a_tailv*57.3*lat_dir*C_vt; % Vertical tail contribution. 57.3 to
convert from /deg to /rad.
    C_nb=C_nbw+C_nbfus+C_nbv;


    %% LONGITUDINAL STATIC STABILITY
    eta_h=0.9;
    x_ach=L_ht/c;
    downwash=1-a*57.3/(pi*e*AR);
    Kf=0.02; %from big book page 498 based on the position of the wing
    c_mafus=Kf*Df^2*Lf/(c*SA);
    C_ma=a*(x_cg-x_acw)+c_mafus-eta_h*(S_ht/SA)*a_tail*downwash*(x_ach-x_cg);
    Staticmargin=C_ma/-a*100;

    Re_cruise=rho*V_cruise*c/mu;


    %% TAKEOFF ANALYSIS
    D = [alpha',C_D',C_L'];
    t0 = 0;
    tf = [2 5];
    for i=1:30
        for j=1:2
    [t,Z] = ode45(@TakeoffSim,[t0,tf(j)],[ 0 0 0 0],[],D);
    H(j) = Z(end,3);
        end
        Th=tf(1)+(V_takeoff-H(1))/(H(2)-H(1))*(tf(2)-tf(1)); %interpolate
another guess
    tf(1)=(tf(1)+Th)/2; %make it into two guesses to get another
interpolation
    tf(2)=(tf(2)+Th)/2;
    Delta = H(2)-V_takeoff;
    end

    Takeoffd = [t,Z];
    Saveprompt = 'SAVE DATA? 1 = Yes | 0 = No';
    Saveanswer = input(Saveprompt);
    if Saveanswer == 1
    xlswrite('Takeoffsim.xlsx',Takeoffd,1);
    end
    IC = Z(end,:);
    m_real = [2.2 2.4];
    for i = 1:2
    V_cruise_calc(i) = sqrt(2*m_real(i)*g/(rho*SA*C_Lcruise));
```

```
    end

end
%% plot everything
MARKER={'-k','--k',':k','-.k','-k','--k',':k'};

if answerstart == 1
    figure(20)
    for i=startfor:endfor
        plot(alpha_200k(:,i),C_L_200k_inf(:,i),MARKER{i})
        hold on
        grid on
        xlabel('Angle of Attack (deg)')
        ylabel('Coefficient of Lift')
        title('C_L vs AoA for Multiple Airfoils')



    end
    legend('1410','2412','23012','23112','24112','63(2)-215')
    figure (21)
    for i=startfor:endfor
        plot(alpha_200k(:,i),CLCD_inf(:,i),MARKER{i})
        hold on
        grid on
        xlabel('Angle of Attack (deg)')
        ylabel('Lift/Drag')
        title('C_l/C_D for Multiple Airfoils')

    end
    legend('1410','2412','23012','23112','24112','63(2)-215')
elseif answerstart == 0



    figure(1)
    hold on
    %plot(alpha_200k,C_L_200k_inf)%plot alpha vs CL

    plot(alpha, C_L,'k') %plot the Cl for various aspect ratios

    ylabel('CL')
    xlabel('Alpha (deg)')
    grid on
    %legend('2D','Finite')
    title('CL vs Angle of Attack')

    figure(2)
    %plot(C_D_200k_inf,C_L_200k_inf) %plot CD vs CL
    hold on

    plot(C_D,C_L,'k')

    %legend('2D','Finite')
```

```
xlabel('CD')
ylabel('CL')
grid on
title('CL vs CD')

figure(3)
%plot(alpha,CDalpha) %plot alpha vs CD
hold on

plot(alpha,C_D,'k')

%legend('2D','Finite')
ylabel('CD')
xlabel('Alpha (deg)')
grid on
title('CD vs Angle of Attack')

figure(4) %plot CL vs CM
%plot(C_L_200k_inf,CM_200k_inf)
hold on

plot(C_L,CMalpha,'k')

%legend('2D','Finite')
ylabel('Cm')
xlabel('Cl')
grid on
title('Cm vs CL')

figure(5) %plot CLCD
%plot(alpha_200k,CLCD_inf)
hold on

plot(alpha, CLCD,'k')

%legend('2D','Finite')
ylabel('Cl/Cd')
xlabel('Angle of Attack (deg)')
grid on
title('Cl/Cd vs Alpha')




figure(7)
plot(accell_gs,C_LM,'k')
grid on
ylabel('Required CL')
xlabel('Loading (g)')
title('Required Lift for Turns')

figure(8)
```

```
plot(alpha,Thrust_cruise,'k')


grid on
ylabel('Thrust (N)')
xlabel('Alpha (deg)')
title('Required Thrust for Angle of Attack During Cruise')


figure(9)

plot(alpha,Thrust_takeoff,'k')

grid on
ylabel('Thrust (N)')
xlabel('Alpha (deg)')
title('Required Thrust for Angle of Attack During Takeoff')


figure(13)

plot(C_L,CLCD,'k')


xlabel('Coefficient of Lift')
ylabel('Lift over Drag')
title('L/D vs CL')
grid on

figure(14)
subplot(2,1,1)
for i=1:length(V)
    plot(alpha_tail,Force_tail(:,i),MARKER{i})
    hold on
end
grid on
legend('Takeoff','Cruise')
ylabel('Force (N)')
xlabel('Angle of Attack (deg)')
title('Force vs Alpha')
subplot(2,1,2)
plot(alpha_tail,C_L_tail,'k')
grid on
ylabel('Coefficient of Lift')
xlabel('Angle of Attack (deg)')
title('Cl vs Alpha')

% plot takeoff sim
figure(15)

plot3(t,Z(:,2),Z(:,1)) %plot time vs x and y location
grid on

figure(16)
```

```
    plot(t,Z(:,3))
    grid on


     %% Print All Results
     clc
     fprintf('\n');
     fprintf('-----------------------------------------------------------------
\n');
     fprintf('REYNOLDS NUMBER:                  %.3f \n',Re_cruise);
     fprintf('CRUISE SPEED (W/ SAFETY):         %.3f m/s\n',V_cruise);
     fprintf('TAKEOFF SPEED:                    %.3f m/s\n',V_takeoff);
     fprintf('ANGLE OF INCIDENCE:               %.3f deg\n',AngleIncidence);
     fprintf('L/D:                              %.3f \n',L_D);
     fprintf('MINIMUM TURNING RADIUS:           %.3f m %.3f m
\n',Rmin(1),Rmin(2));
     fprintf('CHOSEN TURNING RADIUS:            %.3f m at %.3f
g\n',R_chosen,accel_gchosen);
     fprintf('DRAG AT TAKEOFF:                  %.3f N \n',T_takeoff);
     fprintf('YAW COEFFICIENT DERIVATIVE:       %.3f   \n',C_nb(1));
     fprintf('STATIC MARGIN (PITCH):            %.3f   \n',Staticmargin);
     fprintf('CALCULATED REAL CRUISE SPEED:     %.3f m/s \n %.3f m/s
\n',V_cruise_calc(1),V_cruise_calc(2));
     fprintf('-----------------------------------------------------------------
\n');
     fprintf('\n-AEROMAN\n')
end
```

Technical Analysis Code

```
function Accel = TakeoffSim(t,Z,D)
rho = 1.225;
SA = .39;
m = 2.4;
g = 9.81;
M = m*g;
Crr = 0.002;
Alpha = D(:,1);
Cd = D(:,2);
Cl = D(:,3);
V = sqrt(Z(3)^2+Z(4)^2);
T = 1.2;
Thrust = T*g;
Theta = 1.5;
[Minimum, Theta_i] = min(abs(Alpha-Theta)); %find index of angle of attack
for takeoff
CD = Cd(Theta_i);
CL = Cl(Theta_i);

Lift = 0.5*CL*V^2*SA*rho;
if Lift < M
    Normal = M-Lift;
elseif Lift >= M
    Normal = 0;
end
Friction = Crr*Normal;

Drag = 0.5*CD*V^2*SA*rho;
Int_y = Lift/m-g;
Int_yp=sqrt((Int_y*abs(Int_y)+Int_y^2)/2);
Accel = zeros(4,1);
Accel(1) = Z(3);
Accel(2) = Z(4);
Accel(3) = (Thrust-Drag-Friction)/m;
Accel(4) = Int_yp;
```

IMU Analysis Code

```
clear
clc
clf
close all

Startprompt = 'Which Flight? (4-30 (1) or 5-6 (2))\n';
answer = input(Startprompt);
if answer == 1

filename = 'M-plane_4-30-16.xlsx';

elseif answer == 3
    filename='Flight_4-1-16.xlsx';
elseif answer == 2
    filename = 'M-plane 5-6-16.xlsx';
end


sheet=1;

if answer == 1
Data = xlsread(filename,sheet,'A2:K25000','basic');
elseif answer == 2
    Data = xlsread(filename,sheet,'A11000:K72000','basic');
elseif answer == 3
    Data = xlsread(filename,sheet,'A2:K76760','basic');
end
Time=Data(:,1)/1000;

for i = 2:4
    Data(:,i)=Data(:,i)*57.3;
end
%% Find Start and End of Flight
flag = 0;
i=1;
while flag == 0
    if Data(i,11) <= 590 %640 for first flight
        flag = 0;
    elseif Data(i,11) > 590
        starttime = i;
        flag = 1;
    end
    i=i+1;
end
if answer == 1
[Minimum,endtime]=min(abs(Time-100-Time(starttime)));
elseif answer == 2
  [Minimum,endtime]=min(abs(Time-420-Time(starttime)));
elseif answer == 3
    [Minimum,endtime]=min(abs(Time-330-Time(starttime)));
end
if answer == 1
```

```
endtime=endtime-1;
end
Snip=Data(starttime:endtime,:);

fprintf('GATHERED DATA\n');
%% Fast Fourier Transform
L=endtime-starttime;
time_length=Time(endtime)-Time(starttime);
Sampl_freq=L/time_length;
Period=1/Sampl_freq;

Y=fft(Data(starttime:endtime,7));
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);


freq = Sampl_freq*(0:(L/2))/L;
plot(freq,P1)



%% filter
[b,a] = butter(6,.15);
[d,c] = butter(6,.05);
[f,e]=butter(6,0.15);
[h,g]=butter(3,[0.01 0.05], 'stop');

figure(20)
freqz(f,e)
n=endtime-starttime+1;
Gyrob=zeros(n,3);
Accel=zeros(n,3);
Mag=zeros(n,3);


Gyrob(:,3)=filter(b,a,Snip(:,4));
Accel_med=filter(f,e,Snip(:,7));
Accel(:,3)=Accel_med; %filter(h,g,Accel_med);


Mag(:,3)=filter(b,a,Snip(:,10));
for i=1:2
    Gyrob(:,i)=filter(b,a,Snip(:,i+1));
    Accel(:,i)=filter(f,e,Snip(:,i+4));
    Mag(:,i)=filter(b,a,Snip(:,i+7));
end
Pitot=filter(d,c,Data(:,11));
Pitot=Pitot(starttime:endtime);
PitotData = xlsread('Pitot_Calibration2.xlsx',1,'A2:F12','basic');
P = polyfit(PitotData(:,1),PitotData(:,4),4);

Airspeed = polyval(P,Pitot);
Pitot_err = [560 560+0.26];
```

```
Pitoterror = polyval(P,Pitot_err);
Pitot_toterr = Pitoterror(2)-Pitoterror(1);

Bestfitx  = [530:0.1:600];
Bestfit = polyval(P,Bestfitx);
figure(22)
plot(Bestfitx,Bestfit)
grid on

fprintf('FILTERED DATA \n');

%% Turning analysis
%choose 43-60 seconds for flight 1
%choose 115 - 133 seconds for flight 2

Time_a = Time(starttime:endtime);
Time_a = Time_a-Time(starttime);




if answer == 1
    Turnstarti = 43;
    Turnendi = 60;

elseif answer == 2
    turnprompt2 = 'Which turn? [(1)118, (2)136, (3)294?)\n';
    Turnstartip = input(turnprompt2);
    if Turnstartip == 1
        Turnstarti = 122;
        Turnendi = 136;
    elseif Turnstartip == 2
        Turnstarti = 136;
        Turnendi = 153;
    elseif Turnstartip == 3
        Turnstarti = 294;
        Turnendi = 310;
    end
    Endflicki = [321 331];
    Startflicki = [314 322];
elseif answer == 3
    Takeoffstarti = 42;
    Takeoffendi = 53;
    Turnstarti = 138;
    Turnendi = 151;


end


[minimum , Startturn] = min(abs(Time_a-Turnstarti));
[minimum , Endturn] = min(abs(Time_a-Turnendi));

if answer == 1
```

```
    Takeoffstarti = 30;
    Takeoffendi = 39;
elseif answer == 2
    Takeoffstarti = 42;
    Takeoffendi = 49;
end


[minimum , Starttakeoff] = min(abs(Time_a-Takeoffstarti));
[minimum , Endtakeoff] = min(abs(Time_a-Takeoffendi));


%remove the bias
for i=1:3
    Gyromean(i) = mean(Gyrob(:,i));

    Gyro(:,i) = Gyrob(:,i)-Gyromean(i);

end

Gyro_turn = Gyro(Startturn:Endturn,:);
Time_t = Time_a(Startturn:Endturn);
Accelturn = Accel(Startturn:Endturn,:);
Pitch = zeros(length(Time_t),1);
Roll = Pitch;
Yaw = Pitch;
Accelrot_f(1,:) = Accelturn(1,:);


for i=1:length(Time_t)
    acceltot(i) = sqrt(dot(Accelturn(i,:),Accelturn(i,:)));
end

figure(100)
plot(Time_t,acceltot)
hold on
plot(Time_t,Accelturn(:,3))
legend('Total Accel','Z-direction')
for i=1:(length(Time_t)-1)

Pitch(i+1) = (Time_t(i+1)-Time_t(i))*(Gyro_turn(i,1))+Pitch(i);
Roll(i+1) = (Time_t(i+1)-Time_t(i))*(Gyro_turn(i,2))+Roll(i);
Yaw(i+1) = (Time_t(i+1)-Time_t(i))*(Gyro_turn(i,3))+Yaw(i);
Euler(i,:) = [Pitch(i+1) Roll(i+1) Yaw(i+1)];
Gyro_turni(i,:) = Rotation(Euler(i,:),Gyro_turn(i+1,:));
Gyro_turn(i+1,:) = Gyro_turni(i,:);
Euler_accel = [Pitch(i+1) Roll(i+1) 0];
Accelrot_f(i+1,:) = Rotation(Euler_accel,Accelturn(i+1,:));

end


%% Takeoff Analysis
% choose 30-37 seconds for flight 1
```

```
% choose 42-49 seconds for flight 2


Gyro_takeoff = Gyrob(Starttakeoff:Endtakeoff,:);

Gyromean(2) = mean(Gyrob(:,2));
Gyro_takeoff(:,2) = Gyrob(Starttakeoff:Endtakeoff,2)-Gyromean(2);




Time_takeoff = Time_a(Starttakeoff:Endtakeoff);



Acceltakeoff = Accel(Starttakeoff:Endtakeoff,:);
Pitch_takeoff = zeros(length(Time_takeoff),1);
Roll_takeoff = Pitch_takeoff;
Yaw_takeoff = Pitch_takeoff;
Accelrot_takeoff(1,:) = Acceltakeoff(1,:);



for i=1:(length(Time_takeoff)-1)

Pitch_takeoff(i+1) = (Time_takeoff(i+1)-
Time_takeoff(i))*(Gyro_takeoff(i,1))+Pitch_takeoff(i);
Roll_takeoff(i+1) = (Time_takeoff(i+1)-
Time_takeoff(i))*(Gyro_takeoff(i,2))+Roll_takeoff(i);
Yaw_takeoff(i+1) = (Time_takeoff(i+1)-
Time_takeoff(i))*(Gyro_takeoff(i,3))+Yaw_takeoff(i);
Euler_takeoff(i,:) = [Pitch_takeoff(i+1) Roll_takeoff(i+1) Yaw_takeoff(i+1)];
Gyro_takeoffi(i,:) = Rotation(Euler_takeoff(i,:),Gyro_takeoff(i+1,:));
Gyro_takeoff(i+1,:) = Gyro_takeoffi(i,:);
Euler_acceltakeoff = [Pitch_takeoff(i+1) Roll_takeoff(i+1) 0];
Accelrot_takeoff(i+1,:) = Rotation(Euler_acceltakeoff,Acceltakeoff(i+1,:));

end




[Max_takeoff, Time_max_takeoffi] = max(Accelrot_takeoff(:,2));
Time_max_takeoff = Time_takeoff(Time_max_takeoffi);


%% Flick Analysis
if answer == 2
for i = 1:2
[minimum , Startflick(i)] = min(abs(Time_a-Startflicki(i)));
[minimum , Endflick(i)] = min(abs(Time_a-Endflicki(i)));
Endflick(i) = 2*round(Endflick(i)/2);
end
```

```
%flick 1
Gyro_flick1 = Gyro(Startflick(1):Endflick(1),:);
Time_f1 = Time_a(Startflick(1):Endflick(1));



Pitch_flick1 = cumtrapz(Time_f1,Gyro(Startflick(1):Endflick(1),1));
Roll_flick1 = cumtrapz(Time_f1,Gyro(Startflick(1):Endflick(1),2));
Yaw_flick1 = cumtrapz(Time_f1,Gyro(Startflick(1):Endflick(1),3));
Accel_flick1 = Accel(Startflick(1):Endflick(1),:);
Euler_flick1 = [Pitch_flick1 Roll_flick1 Yaw_flick1];

for i=1:length(Pitch_flick1)
    Accelrot_flick1(i,:) = Rotation(Euler_flick1(i,:),Accel_flick1(i,:));
end



%flick2


Gyro_flick2 = Gyro(Startflick(2):Endflick(2),:);
Time_f2 = Time_a(Startflick(2):Endflick(2));




Pitch_flick2 = cumtrapz(Time_f2,Gyro(Startflick(2):Endflick(2),1));
Roll_flick2 = cumtrapz(Time_f2,Gyro(Startflick(2):Endflick(2),2));
Yaw_flick2 = cumtrapz(Time_f2,Gyro(Startflick(2):Endflick(2),3));
Accel_flick2 = Accel(Startflick(2):Endflick(2),:);
Euler_flick2 = [Pitch_flick2 Roll_flick2 Yaw_flick2];

for i=1:length(Pitch_flick2)
    Accelrot_flick2(i,:) = Rotation(Euler_flick2(i,:),Accel_flick2(i,:));
end



end
%% plotting

figure(2)
for i=1:3
    plot(Time_a,Accel(:,i))
    hold on

end
grid on
legend('x','y','z')
title('accelerometer')
```

```
figure(3)

for i=1:3
    subplot(3,1,i);
    plot(Time_a,Gyrob(:,i));
    if i==1
        title('Gyro X')
        grid on
    elseif i==2
        title('Gyro Y')
        grid on
    elseif i==3
        title('Gyro Z')
        grid on
    end

end
```

```
figure(4)
plot(Time_a,Pitot);
grid on
xlabel('Time(s)');
ylabel('Pitot Voltage')
```

```
figure(5)
subplot(3,1,1)
plot(Time_a(Startturn:Endturn),Roll)
ylabel('Deg')
xlabel('Time(s)')
title('Roll vs Time During Turn')
grid on
subplot(3,1,2)
plot(Time_a(Startturn:Endturn),Yaw)
grid on
ylabel('Deg')
xlabel('Time (s)')
title('Yaw vs Time During Turn')
subplot(3,1,3)
plot(Time_a(Startturn:Endturn),Pitch)
ylabel('Deg')
xlabel('Time (s)')
title('Pitch vs Time During Turn')
grid on
```

```
figure(6)

subplot(3,1,1)
plot(Time_t,Accelrot_f(:,1))
```

```
xlabel('Time(s)')
ylabel('Accel X (m/s^2')
title('X Acceleration vs Time')
grid on

subplot(3,1,2)
plot(Time_t,Accelrot_f(:,2))
xlabel('Time(s)')
ylabel('Accel Y (m/s^2')
title('Y Acceleration vs Time')
grid on

subplot(3,1,3)
plot(Time_t,Accelrot_f(:,3))
xlabel('Time(s)')
ylabel('Accel Z (m/s^2')
title('Z Acceleration vs Time')
grid on


Turningd = [Time_t Accelrot_f(:,1) Accelrot_f(:,2) Accelrot_f(:,3)
Airspeed(Startturn:Endturn)];
figure(7)

plot(Time_t,Airspeed(Startturn:Endturn))
grid on
xlabel('Time (s)')
ylabel('Airspeed (m/s)')
title('Airspeed During Turn')


figure(8)

title('Takeoff Analysis')
subplot(3,1,1)
plot(Time_takeoff,Roll_takeoff)
ylabel('Deg')
xlabel('Time(s)')
title('Roll vs Time During Takeoff')
grid on
subplot(3,1,2)
plot(Time_takeoff,Yaw_takeoff)
grid on
ylabel('Deg')
xlabel('Time (s)')
title('Yaw vs Time During Takeoff')
subplot(3,1,3)
plot(Time_takeoff,Pitch_takeoff)
ylabel('Deg')
xlabel('Time (s)')
title('Pitch vs Time During Takeoff')
grid on


figure(9)
```

```
title('Takeoff Analysis')
subplot(3,1,1)
plot(Time_takeoff,Accelrot_takeoff(:,1))
xlabel('Time(s)')
ylabel('Accel X (m/s^2)')
title('X Acceleration vs Time')
grid on

subplot(3,1,2)
plot(Time_takeoff,Accelrot_takeoff(:,2))
xlabel('Time(s)')
ylabel('Accel Y (m/s^2)')
title('Y Acceleration vs Time')
grid on

subplot(3,1,3)
plot(Time_takeoff,Accelrot_takeoff(:,3))
xlabel('Time(s)')
ylabel('Accel Z (m/s^2)')
title('Z Acceleration vs Time')
grid on


figure(10)

plot(Time_takeoff,Airspeed(Starttakeoff:Endtakeoff))
grid on
xlabel('Time (s)')
ylabel('Airspeed (m/s)')
title('Airspeed During Takeoff')

Takeoffd = [Time_takeoff Airspeed(Starttakeoff:Endtakeoff)
Accelrot_takeoff(:,1) Accelrot_takeoff(:,2) Accelrot_takeoff(:,3)];
figure(11)
plot(Time_a,Airspeed)
xlabel('Time (s)');
ylabel('Airspeed (m/s)');
title('Airspeed vs Time')
grid on

if answer == 2
figure(12)

plot(Time_f1,Gyro_flick1(:,1))
hold on
plot(Time_f1,Gyro(Startflick(1):Endflick(1),1))
grid on
ylabel('Pitch (deg/s')
xlabel('Time (s)')
title('Pitch Rate vs Time During Flick Test')
legend('After Rotation','Before Rotation')

figure(13)

plot(Time_f1,Pitch_flick1(:,1))
```

```
ylabel('Pitch (deg)')
xlabel('Time (s)')
title('Pitch vs Time During Flick Test')
Flick1d = [Time_f1 Gyro(Startflick(1):Endflick(1),1) Pitch_flick1];

figure(14)

plot(Time_f2,Gyro_flick2(:,1))
hold on
plot(Time_f2,Gyro(Startflick(2):Endflick(2),1))
grid on
ylabel('Pitch (deg/s')
xlabel('Time (s)')
title('Pitch Rate vs Time During Flick Test 2')

figure(15)

plot(Time_f2,Pitch_flick2(:,1))
ylabel('Pitch (deg)')
xlabel('Time (s)')
title('Pitch vs Time During Flick Test 2')
Flick2d = [Time_f2 Gyro_flick2 Pitch_flick2];
end

%% PRINT RESULTS
fprintf('MAX TAKEOFF ACCELERATION: %.3f m/s^2 at %.3f
s\n',Max_takeoff,Time_max_takeoff);
fprintf('PITOT TUBE ERROR: %.3f m/s \n',Pitot_toterr);

%% SAVE RESULTS TO EXCEL

Saveprompt = 'SAVE DATA? 1 = yes | 0 = no\n';
Saveanswer = input(Saveprompt);
if Saveanswer == 1
    if answer == 2
        xlswrite('IMU_Results2.xlsx',Flick2d,1);
        xlswrite('IMU_Results2.xlsx',Takeoffd,2);
        xlswrite('IMU_Results2.xlsx',Turningd,3);
        xlswrite('IMU_Results2.xlsx',Flick1d,4);
    end
    if answer == 1
        xlswrite('IMU_Results1.xlsx',Takeoffd,1);
        xlswrite('IMU_Results1.xlsx',Turningd,2);
    end
    if answer == 3
        xlswrite('IMU_Results1.xlsx',Turningd,8);
        xlswrite('IMU_Results1.xlsx',Takeoffd,9);
    end
end
```

Pitot Tube Calibration Code

```
clear
clc
clf
close all
promptstart = 'GATHER DATA? 1 = YES | 0 = NO \n'; %ask if data should be
gathered
answerstart = input(promptstart);

if answerstart == 1


for i=1:12
    if i >=1 && i <=10
        filenames{i}=sprintf('flight0%d.xlsx',i-1);
    else
        filenames{i}=sprintf('flight%d.xlsx',i-1);
    end
end

%change filename order
filename{1} = filenames{1};
for i=2:7
    filename{i} = filenames{i+5};
end
for i = 8:12
    filename{i} = filenames{i-6};
end

Pitotdata=zeros(6000,1);
startpitot=1;
endpitot=300;
for i=1:12
    startpitot=(i-1)*300+1;
    endpitot=i*300;

Pitotdata(startpitot:endpitot)=xlsread(filename{i},1,'K701:K1000','basic');
end
filenamewind={'0.xlsx','10.xlsx','15.xlsx','20.xlsx','25.xlsx','30.xlsx','35.
xlsx','40.xlsx','45.xlsx','50.xlsx','55.xlsx','60.xlsx'};
Windtunneldata=zeros(11*12,1);
for i=1:12
    startwind=(i-1)*11+1;
    endwind=i*11;

Windtunneldata(startwind:endwind)=xlsread(filenamewind{i},1,'C2:C12','basic')
;
end

Windtunneldata = Windtunneldata*0.44704; %convert mph to m/s
%% STATISTICS ANALYSIS

percentilepitot = tinv(.975,299); %student t for pitot
```

```
percentilewind = tinv(.975,11); %student t for wind tunnel
for i=1:12
    startpitot = (i-1)*300+1;
    endpitot = i*300;
    Ptmean(i) = mean(Pitotdata(startpitot:endpitot));
    Ptstd(i) = std(Pitotdata(startpitot:endpitot))/sqrt(300); %standard
deviation of the means of the pitot tube data
    startwind = (i-1)*11+1;
    endwind = i*11;
    Wnmean(i) = mean(Windtunneldata(startwind:endwind));
    Wnstd(i) = std(Windtunneldata(startwind:endwind))/sqrt(11); %standard
deviation of the means of the wind tunnel data
end

%calculate precision error

uPt = Ptstd*percentilepitot;
uWn = Wnstd*percentilewind;

Exportdata = [Ptmean;Ptstd;uPt;Wnmean;Wnstd;uWn].';
xlswrite('Pitot_Calibration2.xlsx',Exportdata,'A2:F12');



elseif answerstart == 0
    Indata2 = xlsread('Pitot_Calibration2.xlsx',1,'A2:F12','basic');
    Indata1 = xlsread('Pitot_Calibration.xlsx',1,'A2:F12','basic');
    Ptmean2 = Indata2(:,1);
    Wnmean2 = Indata2(:,4);
    Ptmean1 = Indata1(:,1);
    Wnmean1 = Indata1(:,4);
    scatter(Ptmean2,Wnmean2);
    hold on
    scatter(Ptmean1,Wnmean1);
    grid on
    ylabel(' Airspeed (m/s)')
    xlabel('Pitot Voltage Units')
    title('Airspeed vs Pitot Voltage')
    legend('With Wing','Clean Air')

end
fprintf('DONE\n');
```

Arduino Code

```
const int buttonPin = 2;
#include <Wire.h>
//#include <SPI.h>
#include <SD.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_BMP085_U.h>
#include <Adafruit_L3GD20_U.h>
#include <Adafruit_10DOF.h>
#include <MemoryFree.h>

/* Assign a unique ID to the sensors */
Adafruit_10DOF                dof   = Adafruit_10DOF();
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_L3GD20_Unified       gyro  = Adafruit_L3GD20_Unified(20);
Adafruit_BMP085_Unified       bmp   = Adafruit_BMP085_Unified(18001);

/* Update this with the correct SLP for accurate altitude measurements */
//float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;

int chipSelect = 4; // select the CS port
File dataFile;
unsigned int time;
int A_in;
unsigned long TimeStart;
char filename[] = "FLIGHT00.csv"; // set the default filename

/**************************************************************************/
/*!
    @brief  Initialises all the sensors used by this example
*/
/**************************************************************************/
void initSensors()
{
  if (!accel.begin())
  {
```

```
  /* There was a problem detecting the LSM303 ... check your connections */
  //Serial.println(F("Ooops, no LSM303 detected ... Check your wiring!"));
  while (1);
 }
 if (!mag.begin())
 {
  /* There was a problem detecting the LSM303 ... check your connections */
  //Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
  while (1);
 }
 if (!bmp.begin())
 {
  /* There was a problem detecting the BMP180 ... check your connections */
  //Serial.println("Ooops, no BMP180 detected ... Check your wiring!");
  while (1);
 }

 gyro.enableAutoRange(true);
 if (!gyro.begin())
 {
  //Serial.println("There was a problem with the gyro you dipshit");
  while (1);
 }
 TimeStart = millis();
}

/**************************************************************************/
/*!

*/
/**************************************************************************/
void setup(void)
{
// This is for the button //
//  pinMode(buttonPin, INPUT);
//  pinMode(9, OUTPUT);
//  pinMode(A3, INPUT);
//  while (digitalRead(buttonPin) == LOW) {
//    delay(1000);
//  };
//delay(1500000);
  Serial.begin(9600);
```

```
 /* Initialise the sensors */
 initSensors();
 /*
  while (!Serial) {
  ;
  }
 */

 if (!SD.begin(chipSelect)) {
  //Serial.println("Card failed, or not present");
  // don't do anything more:
  return;
 }
 //Serial.println("card initialized.");

 for (byte i = 1; i <= 99; i++)
 {
  if (SD.exists(filename))
  {
   //filename exists so we need to change the filename
   filename[6] = i / 10 + '0';
   filename[7] = i % 10 + '0';
  } else {
   break; // the filename doesnt exist so we can move on
  }
 }

}



/*****************************************************************************/
/*!
   @brief  Constantly check the roll/pitch/heading/altitude/temperature
*/
/*****************************************************************************/
void loop(void)
{

  dataFile = SD.open(filename, FILE_WRITE);
```

```
sensors_event_t accel_event;
sensors_event_t mag_event;
sensors_event_t bmp_event;
sensors_vec_t   orientation;
sensors_event_t gyro_event;

String dataString;

//while(1)
for (int i = 0; i < 10; ++i)
{
  dataString = "";
  dataString += millis() - TimeStart;
  dataString += ";";

  //Gets x, y, and z data from gyro
  gyro.getEvent(&gyro_event);
  dataString += gyro_event.gyro.x;
  dataString += ";";
  dataString += gyro_event.gyro.y;
  dataString += ";";
  dataString += gyro_event.gyro.z;
  dataString += ";";

  //Gets x, y, and z data from accelerometer
  accel.getEvent(&accel_event);
  dataString += accel_event.acceleration.x;
  dataString += ";";
  dataString += accel_event.acceleration.y;
  dataString += ";";
  dataString += accel_event.acceleration.z;
  dataString += ";";

  //Gets x, y, and z data from magnetometer
  mag.getEvent(&mag_event);
  dataString += mag_event.magnetic.x;
  dataString += ";";
  dataString += mag_event.magnetic.y;
  dataString += ";";
  dataString += mag_event.magnetic.z;
  dataString += ";";
```

```
dataString += analogRead(A3);
dataString += ";";



// if the file is available, write to it:
if (dataFile) {
  dataFile.println(dataString);
  //digitalWrite(9, HIGH);
  //dataFile.flush();
  //Serial.println(dataString);

}

// if the file isn't open, pop up an error:
else {
  //Serial.println("error opening datalogs.txt");
  //Serial.println(dataString);
  //digitalWrite(9, LOW); // turn off LED

  //New addition
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");  // Try to reconnect the SD card
    // don't do anything more:
    return;
  }
  for (byte i = 1; i <= 99; i++) //the file isn't open so we make a new one
  {
    if (SD.exists(filename))
    {
      //filename exists so we need to create a new file
      filename[6] = i / 10 + '0';
      filename[7] = i % 10 + '0';
    } else {
      break; // the filename doesnt exist so we can move on
    }
  }
  break; //the file isn't open so exit the for loop

  // end new addition
```

```
    }

  }
  if (dataFile)
  {
    dataFile.close();
  }
}
```