Flight Data Logger

Winston James, Brian Lichtman, and
Shaun Mosley, John (Tony) Torres

Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — This paper describes a senior design project which investigates a means to verify the accuracy of the United States Air Force Stability and Control Digital DATCOM software for L-3 Communications. L-3 Communications has expressed the ability to predict the performance of an aircraft, using the DATCOM software set, based on data about the geometry of an aircraft, called the geometry sample. From this sample, L-3 Communications is able to create a simulation model of that particular aircraft for tasks such as the creating of combat training scenarios and the training of pilots for flight. Currently, the software has only formally been tested on large aircrafts. However, L-3 Communications would like to know the viability of the DATCOM software set on small UAVs. Therefore, in this paper, we describe our design of a method to assess the validity of the DATCOM software set.**

Index Terms — Autonomous drones, flight coefficients, remote control airplane, autopilot

I. Introduction

L-3 Communications is a defense contractor which designs and supplies simulation and training equipment to many of the United States' military divisions. Currently, L-3 Communications uses DATCOM to retrieve an aircraft's coefficients of flight based on its geometry. They then use these outputs to simulate test flights in software such as FlightGear Flight Simulator. However, L-3 Communications as well as many other leading defense contractors face a common conundrum, how do they know their simulator is providing feedback equivalent to that of the actual aircraft in flight. L-3 Communications has informed us that the currently accepted method is to hire a trained pilot to fly the simulator to determine if it is performing as expected. This method tends to inefficient and not cost effective. While many times the software does prove accurate, it has never thoroughly been tested on small aircrafts, specifically, Unmanned Aerial Vehicles (UAVs). Therefore, we devised a method to obtain real-time flight data about the performance of a UAV in flight. Using this information, we compared the results of the flight performance data with the predicted performance data based on their geometry technique.

The motivation for this project starts with the sponsor, who wishes to produce a viable, and accurate, innovative means of predicting the performance of an aircraft in flight without putting the aircraft in flight. This would eliminate the use of resources such as flight time, plane fuel, and providing a low-risk testing environment that ensures the safety of the plane, the public and the pilot.

The secondary motivation for this project comes in the form of technical expertise from the research and design team comprised of 3 computer engineering students (Brian, Tony, and Shaun) and an electrical engineering student (Winston). Winston will specialize in the hardware portion of this project, Brian will pursue the interfacing of the hardware and software components, Tony will develop the applications that will program the hardware, and Shaun will dedicate himself to the programming integrated circuits on the hardware

Our goal is to produce a low-cost, portable, lightweight, & accurate product that is general enough to be capable of being installed in various aircrafts (military, commercial, even R/C models). Our product will be responsible to measure various parameters that determine the flight characteristics of that aircraft such as thrust, lift, and accelerations about various axes. The in-flight parameters, in addition to the geometry parameters crucial to this project are listed later in a large table.

Our objective is therefore to populate a printed circuit board (PCB) with sensors, digital signal processing (DSP) and micro-controlling capabilities, GPS, and provide software application that will program that PCB to be able to provide mixed flight (manual and automatic) control, relayed flight data (altitude, pressure, force, various accelerations, etc.), flight data logger must be versatile, collect as much live data as is possible. This leads to better comparisons later sponsored aircraft must be small unmanned aerial vehicle (UAV) but big as is feasible under budget, extract from the excel spreadsheets L-3 Communications provided, the physical parameters that the hardware must measure in real-time, and equivalently software must be prepared to process.

The manual flight will be possible via the R/C transmitter given the plane is in the range of it. Automatic flight will require a flight profile of the path that will be followed via GPS. Piccolo, the R/C autopilot L-3 Communications proposed, is too expensive and not used. The autopilot chosen was the Ardupilot because it is free open source software. For the flight data a ground receiver on a laptop or other viewing display with recording capabilities will be incorporated and an on-board memory (in the case that the plane is out of range of the ground station)

L-3 Communications has decided to sponsor our Senior Design group in order to complete this task. L-3 Communications funded our group in order for us to obtain the resources necessary in order obtain the flight characteristics needed in order to perform the comparison. In the following sections, we will describe our project. Section II describes the project's requirements. Section III entails the coefficients to be calculated and the methods to calculate them. Section IV describes the hardware used by our project, the microcontroller software design and implementation, and the PBC integration. Section V describes the software system used for the aerodynamic coefficient calculations. Section VI discusses other external components used by our system. Lastly, Section VII concludes our project.

II. Project Requirements

Our sponsor decided that the requirements and specifications were best left to our choosing. After brainstorming idea after idea, we created a set of strict requirements to test our design. The list of requirements as follows:

- Ability to record data at speeds of up to 30 Hz
- PCB board dimensions should be no bigger than 6" x 5" x 1"
- Aircraft should be able to fly for at least 20 minutes while recording data

- Test vehicle must have a wingspan larger than 3 feet
- Must not have more than ½lb weight offset to one side
- Cost must be under $1500.00
- Must be complete and fully functional by December 15, 2011

As described before, we were tasked to design a versatile system which would measure live flight data of a small unmanned aircraft. We were asked to collect as much in flight data as possible would be compared to the outputs of the simulation software used by our sponsor, L-3 Communications. As such, L-3 provided us with several excel spreadsheets which provide the inputs as well as the outputs of their software collection. From this we have formulated the list of data we will attempt to collect. Nonetheless, the list of values obtained from these spreadsheets is too large to list in this document; therefore they are just listed in our final paper.

As for how high the plane must fly, how long, and how far, L-3 Communications has not decided on these specifications because they want to do a cost analysis of the types of airplanes and sensors that are available as these materials can range vastly in price. L-3 Communications said that after we collect a list of prices and specifications of different models of aircraft and sensors they will make a decision as to how much they are willing to spend.

III. Coefficient Calculations

In our project we had to measure 9 coefficients: lift, drag, side force, pitch, roll, yaw, angle of attack (AoA), angle of sideslip (AoS), and dynamic pressure (q). The lift force is calculated by finding the total force on both the top and bottom of the plane. It is then divided by the area that the force sensors take up and multiplied by the total area of the top and bottom of the plane, respectively. These two forces are then subtracted, bottom minus top, and the total lift force is found as seen in (1).

$$F_l = \frac{\sum F_{botSensor}}{A_{botSensors}} * A_{botSerface} - \frac{\sum F_{topSensor}}{A_{topSensors}} * A_{topSerface} \quad (1)$$

The plane experiences forces from air flow on it. The difference in pressures over the wing cause the lift force. The wind alone on the plane causes the sideslip force. Finally the drag force is caused by the plane pushing through the air. The force coefficients are the basic three dimensional forces on the plane. Figure 2 below shows the various forces acting in the different dimensions.

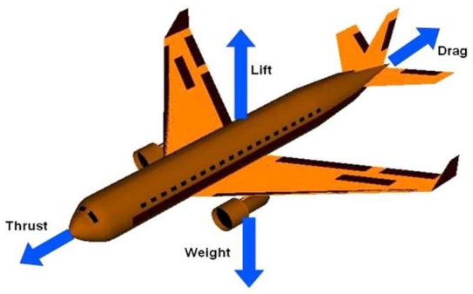

Figure 2 - Forces on the Plane

The side force and drag force is found in a similar way just using the sensors on the side of the plan. These forces are calculated after adjusting for the angle of the sensors on the plane and the angle of attack. As noted in Figure 3, the angle of attack is the direction of the wind relative to the horizontal of the plane.

The pitch, roll, and yaw are all found using a gyroscope. The moments of these, however, are found again with the force sensors. Again the air flow causes forces on the plane and these forces cause moments. The roll is the moment around the x axis, the pitch is the moment around the y axis, and lastly the yaw moment is the moment is around the z axis. Figure 1 below further illustrates this notion.

The moment coefficients are the basic three dimensional moments on the plane. Knowing ahead of time how far each

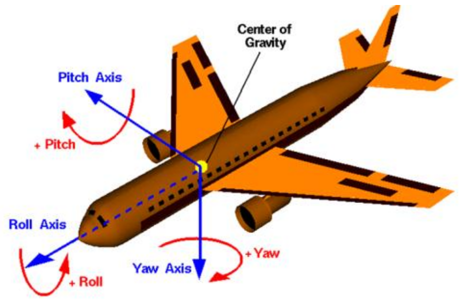

Figure 1 - Moments About the Plane
(Reproduced with permission from NASA Glenn)

sensor is from the center of gravity it was just simple matter of using moment equations to find the moments on the plane. The equation is given below. Again the Forces are calculated after the physical angle and planes actual angle are taken into account.

$$M = \sum_{n=1}^{max} F_n * d_n \quad (2)$$

AoA and AoS are both measured with angles sensors mounted on the boom of our plane. There are no calculations needed after the value comes in.

Dynamic pressure is found by finding the difference in pressure from the front of the boom and the side. Again this value comes in as the number needed and no calculations are needed.

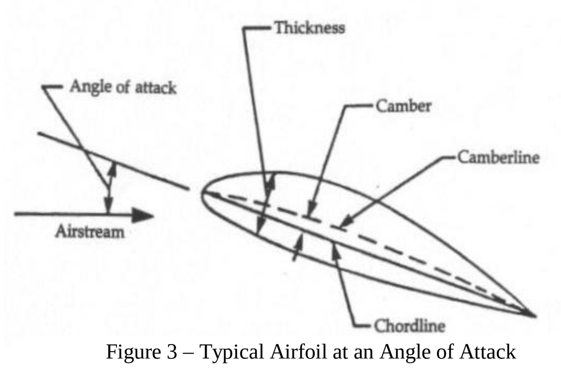

Figure 3 – Typical Airfoil at an Angle of Attack
(Reproduced with permission from NASA Glenn)

IV. Hardware

A. Sensory Selection

The hardware is a data-logger, so the PCB is outfitted with various sensors to measure variables that pertain to DATCOM, these are primarily forces and torques coefficients. A coefficient is dependent on a quantity called dynamic pressure which is dependent on true airspeed and air density. Therefore the goal is to measure forces and torques with respect ultimately to true airspeed since for our conditions; air density shouldn't change very much. Space constraints (RC plane fuselage) called for as many measurements as possible in as small amount of space possible, and thus we desired multitasking parts wherever possible. For this reason we chose a 3D accelerometer (AIS326DQTR) for drag and lift, and gyroscope (L3G4200DTR) for angular velocity, as oppose to two or three 1D accelerometers and gyroscopes. The accelerometer also is low powered at a mere 2.2mW.Since we needed to judge air density, and that depends more or less on humidity, temperature and altitude, we have a Analog Devices temperature (AD7814ARMZ),a Honeywell humidity (HIH-5031-001) sensor, and Bosch barometric pressure sensor (BMP085). In order to measure true airspeed, we employed the pitot-tube technique by using a Freescale differential pressure sensor (MPXV7002DP) with tubes that will be mounted on a boom; one tube facing into the wind for total pressure, and the other on the side of the boom for static pressure from the air flowing perpendicular to its opening. On this boom will be a pair of angle sensors (SV01A103AEA01B00) with wind vanes attached to sense the Angle of Attack and Angle of Sideslip of the aircraft, which are require parameters in DATCOM. The pair of angle sensors is basically potentiometers except that they are tinier to not impede air flow past the boom. Thin membrane force sensors were chosen as the method to record a discrete force distribution on the surface of the aircraft without modifying the airflow over the flight surface. The justification of these sensors is as follows. The force sensors (FSR Design Kit) had to be thin for their functional transparency in wind-flow, and they were the cheapest solution, at $100, for our budget as opposed to $5000 systems [i] that measured a continuous force distribution, and also they were lightweight as opposed to bulkier discrete force sensors, and could measure the forces likely experienced on a flight surface. The accelerometer and gyroscope met space requirements with their 3D axis measurement capability, and they were each sensitive enough for our application with the accelerometer sensing a change as small as 9mm/s^2 /Lsb per axis and gyroscope sensing down to 9mps/Lsb per axis . Influencing our decision was also cost, and thus free parts were very desirable in order to stay under our budget if we needed parts for breaking, testing, and final assembly. Thus we also chose the gyroscope because free samples were available and it was one of the newest ICs from ST Microelectronics. The gyroscope also had the capability to give a temperature reading, thus providing a level of redundancy in estimating ultimately air density, or if we chose to discard our temperature sensor to save more space and MCU pinout. Lastly the gyroscope was a low power device consuming at max about 19mW, which was affordable relative to our power budget of 8.3W. Less power draw meant more flight time for us. Our 10-bit temperature sensor was also sampled (free), it was more accurate and resolute than the 8-bit gyroscope temperature measurement. It also supported SPI, and eliminated the potential problem of a reading being affected by the heat generated from the gyro if it was heating up measuring angles. The differential pressure sensor was among the cheapest found, it was low-powered (50mW), and available for sampling. Being an analog part, it helped eased the load of the digital traffic into the MCU, and was sensitive to 1mV/Pa. It was chosen strongly based on the fact that it is sold commercially in a pitot-tube setup from DIYDrones.com[ii] to measure true airspeed (TAS) which is exactly what we aim to measure. Even more, it was sold for RC drones, one of which we bought! In order to track altitude we're using popular altitude sensor found both on our autopilot hardware, and also as a standalone unit on Sparkfun.com [iii]. The barometric pressure was chosen for its low power (30uW), despite its moderate accuracy of ±8ft in judging atmospheric pressure. We can tolerate a moderate accuracy since the humidity and temperature sensor are already zeroing in on air density, and the barometric pressure sensor is simply helping this measurement, so there is already some redundancy involved. The humidity sensor is also low-powered at 600uW, which is desirable because the instrument is surviving off a Li-Po RC battery which can go empty fairly quickly in flight. We desired to use the ADC of the MCU to have a more balance use of analog and digital MCU features. They are analog for ADC utilization and are low-powered at 5mW.Our communication preference was SPI over I2C, which both the sensors supported. One of the main reasons is that SPI is generally faster than I2C though; I2C can simplify a design like ours that has one master device (MCU) and many slaves (sensors); by putting all the slaves on one main bus. We chose higher speeds in order to approximate a continuous stream of data from every sensor in as short amount of time as possible. We do though have a asymmetric mix of SPI and I2C devices; the I2C barometric sensor, and the SPI gyro and accelerometer, temperature sensor, and micro SD card.

B. Sensory Data Acquisition

With all of the flight data that we are obtaining, we utilize an array of sensors to obtain various data to solve for variables that we need. Since there are over thirty-five sensors installed into the Flight Data Logger, each sensor will be discussed in brief detail. The list of sensory is as follows: force sensor, angle sensor, differential pressure sensor, barometric pressure sensor, humidity sensor, temperature sensor, gyroscope, and accelerometer.

The force sensor is a very critical sensor in the entire design because it is used to calculate more than just wind force. Before any actual flight tests, preliminary tests had to be done to determine the plane's moment of inertia and the force sensors helped ease this process. Having 32 force sensors spread all over the airplane's surface allows for us to get somewhat accurate measurements to determine average force on the surface of several areas over the vehicle. The outputs from the sensors are analog voltages and are converted into 12 bit resolution by the MCU's analog to digital converter.

Another analog sensor is the angle sensor and its use requires additional hardware. The angle sensor is connected to a wind vane which will move from the force of the wind. Figure 4 illustrates what the instrumentation resembles.

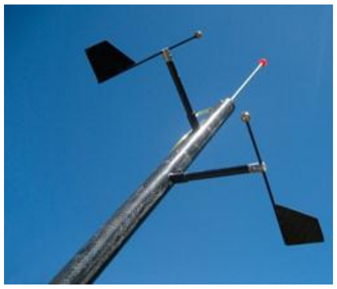

Figure 4 - Angle of Attack Probe
(Reproduced with permission from DIYDrones)

As the wind vanes rotate, the angle sensors will help us determine the direction of the wind flow relative to the horizontal and vertical axes of the vehicle. These values will then be used to calculate the angle of attack and angle of sideslip.

Airspeed is also a variable involved in calculating the angle of attack and sideslip. In order to obtain that data, we use a differential pressure sensor which will compare the pressure from the front of the plane to that of the pressure on the side. With the value this derives, we are able to calculate airspeed and then derive angles of attack and sideslip. This too is an analog sensor.

In addition to the differential pressure sensor, we installed the barometric pressure sensor. Although they both grab data concerning its surrounding pressure, they both serve very different purposes. As explained earlier, the differential pressure sensor is used to determine airspeed, while the barometric sensor is used to calculate the estimated altitude. This device communicates with the MCU via the digital inter-integrated circuit protocol.

The next sensor is another analog device – the humidity sensor. This IC is used to simply obtain the surrounding humidity. Once this information is recorded it is used in our computer based software to calculate other flight data.

Next is our temperature sensor. This sensor communicates via 4-wire SPI to the microcontroller and has a resolution of the temperature to 0.25˚ C. Again, the values obtained from this sensor are plugged into several other equations to validate other findings.

Lastly are the gyroscope and accelerometer. Both are presented simultaneously because even though they are two individual parts, they work together to calculate the plane's movement. The gyroscope is used to measure the rotary movements of the plane in any of the three dimensions – particularly referred to in aero terms as yaw, pitch, and roll. Since some data from the gyroscope can be a little bit shaky, the accelerometer is used to validate that the plane actually was in motion in the specified axis.

Each sensor operates independently of one another. Hence any failure of one sensor will not affect the operation of any other in the system. All integrated circuits produce various data types with a broad range of outputs – for example, the force sensors put out an analog signal representative of the force on the device, while the temperature sensor produces digital, results with a user specified resolution.

As required, our system is set to run at a rate of 60 Hz; so every second the MCU will have communicated with each sensor at least 60 times. Since all of the sensors run independently, there is no concern of sensor failure slowing down system run time. After data has been extrapolated, it must be converted into some understandable value.

Although each sensor may produce various outputs, all data is converted, with its respective units, into an IEEE 754 single precision floating point value. After the data has been polled and the calculations have been complete, the float is then transferred over to the microSD card for storage. Then the microcontroller goes onto the next sensor to repeat the process.

C. Microcontroller Storage System Software Design

As has been discussed, this system will be recorded various flight characteristics at a very rapid rate. Therefore, the system we implemented must be able to handle a continuous influx of data as well as have a method to store it. The microcontroller we used, the dsPIC33EP512MU810, has the ability to read large amounts of data from both analog and digital I/O devices at relatively fast rates. The microcontroller has 52 KBs of RAM which can be used to hold values temporarily. Nonetheless, 52 KBs is not a lot of space. It is highly likely we would have run out of memory well before we stop taking samples from our sensors if we only used the built in RAM. Therefore, we decided to use an external memory device to store our data. When choosing a device, we first calculated an approximation of how much memory we approximated to be needed to store all of the data we intend to collect over a set period of time.

In order to come up with an approximation of how much memory was to be required to store all the data, we made the assumptions that our flight time would run about 20 minutes. Therefore, for safety purposes, we doubled this value. This ensured that we had more than ample space for storing our data. We also made the assumption that our test will run for 40 minutes. The first value we calculated was approximately how many measurements would be taken over the course of the flight. We assumed we would perform n measurements cycles a second. Using this, we then approximated the number of measurements that will need to be taken in terms of n.

$$Flight\ time\ (sec) \times n\ \frac{measurement\ cycles}{second} \qquad (3)$$

$$= Number\ of\ measurements\ cycles$$

Since we already had chosen a value of 40 minutes as the length of time we would be taking measurements for, we were able to plug this value in and remove some of the variables in the equation.

$$40\ \text{min} \times 60\ \frac{sec}{min} \times n\ \frac{measurements}{sec} \qquad (4)$$

$$= 2400n\ measurement\ cycles$$

Now we had a good idea as to how many measurement cycles we would need to perform. Using this value of 2400n, we can saw a linear increase in memory usage occurred as we increase the amount of times we poll data from the sensors on the aircraft. Therefore, choosing a value for n was left until after we determined how much memory each measurement will require.

Next, we chose to follow the IEEE 754 single precision floating point standard to store our data obtained by our microcontroller. According to Goldberg, the IEEE 754 single precision floating standard requires the use of a 32 bit word of data to store one value. [1] Using this knowledge, we calculated how many bytes of data would be required to store each measurement.

$$\frac{32\ bits}{IEEE\ single\ persion\ floating\ point\ value} \times \frac{1\ byte}{8\ bits} \times \frac{1\ IEEE\ single\ persion\ floating\ point\ value}{measurement} = \frac{4\ bytes}{measurement} \quad (5)$$

As described earlier, we used 36 sensors. From this we deduced that 36 floating point values were to be stored each measurement cycle performed by the microcontroller. Using this information we calculated it would take 152 bytes of memory for each measurement cycle. Using the values obtained thus far, we calculated the anticipated memory usage of the system per measurement cycle.

$$\frac{4\ bytes}{measurement} \times \frac{38\ measurements}{measurement\ cycle} = \frac{152\ bytes}{measurement\ cycle} \quad (6)$$

Based on the values we had obtained thus far, we form a linear relationship between the number or measurement cycles we perform per second and the amount bytes which would be required to store the values of the measurements taken over a period of 40 minutes. This relationship can be seen below.

$$2400m\ measurement\ cycles \times \frac{152\ bytes}{measurement\ cycle} = 364{,}800n\ bytes \quad (7)$$

Using this relationship, we chose a frequency at which we would want to obtain data from our sensors. According to the specification of the dsPIC33EP512MU810, 1.1 million samples can be taken per second from the ADC I/O. However, although this may be true, not all of our I/O devices were capable of providing data at such a rate. Using our slowest part, the three axis digital gyroscope, as a ceiling, the maximum read rate we could obtain was 800 samples per second. However, this many samples per second was overkill as far as our needs were concerned. In addition, it would cost approximately 291,840,000 bytes of data to store at this rate. This was approximately 280 megabytes of data. This is quite a lot of space for just raw data. Therefore, we decided to limit our frequency of collection well below the maximum that is allowable by the dsPIC33EP512MU810.

We decided to take samples at a rate of 30 Hz. At this rate, we had more than ample amounts of data to work with. Additionally, we decreased our memory usage by about 96% which brought us down to a memory cost of about 10,944,000 bytes which is approximately 10.5 megabytes. This was a much more reasonable amount of data to work with.

Although 10.5 MBs doesn't seem like a lot of data in today's computing world, when you consider the fact that the dsPIC33EP512MU810 has only 52 KBs of RAM, you can see that the microprocessor would not have been able to handle more than a few measurement cycles before its RAM becomes full. Additionally, this number had the increase or decrease drastically should our sponsor have decide that the design needed to be changed and therefore sensors needed to be added. Additionally, we were not even taking into account the amount of RAM that will need to be used just to run the program which will poll the data from the sensors before it is even stored.

Using this information, we then decided had to decide on a means to store the data. L-3 Communication's only requirement was that the data be simple to retrieve. There were a few options as to how to transfer the observed flight data obtained by the sensors to a computer for later use by our sponsor. Since it had already been determined that it will be impossible to store all the data on the processor's internal memory, the idea of simply using an onboard USB connection to read from the microcontrollers memory had to be dismissed. Memory could have been added in order to utilize USB functionality; however, we found other options that seemed to provide more desirable results.

The three options we had to choose from included the use of a radio frequency tag reader collect data that would be read via a computer, live transfer of data via radio frequency, or storage using a using a micro Secure Digital card (microSD card). We decided against the radio frequency tag because a secondary medium would still be needed to store the data before it could be collected. We also decided against live transfer as it was possible that data loss if there was too much interference in the area the test was run.

Therefore, we went with our last option, using a micro Secure Digital card (microSD card). The microSD card was determined to be the best fit because it is an easily attainable piece of hardware and available in a variety of data sizes as well as physical sizes and offers a variety of read/write speed capabilities. Plus, it is removable and most computers offer an SD card reader as a standard which can be used with a microSD to SD card converter. Or in the event that the computer lacks that hardware, microSD-to-USB adapters are readily available at a reasonable price. This solution answered our needs of being able to handle data storage and being able to get the data to the user.

Connection of the microSD card to the microcontroller was done using the SPI. A microSD card can be connected using the pin layout shown in Figure 5.

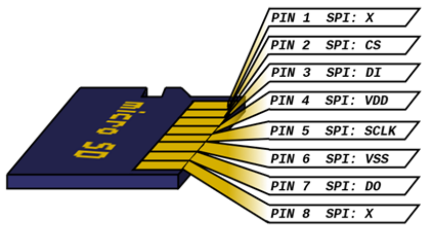

Figure 5- SPI Pin layout of a microSD card
(Reproduced with permission from Amanda Bogeman)

With the microSD card interfaced to our microcontroller, we used Microchip Technologies' File System I/O Library write our data to it. All data is stored in a binary format on the microSD card using the IEEE single precision floating point standard just described. Every time a value is read from a sensor, it was written to a storage buffer created by the library. The appropriate function from the library was then called to flush the buffer to the microSD card. This process was repeated every sensor in the system until the completion of the test.

D. PCB Design

Our 1st 2-layer PCB revision had power, ground, and signal traces mixed with no planes, and looked very confusing, so we simplified our design by designing a 2nd 4-layer revision with 2 planes for power and ground and 2 signal planes, per the advice of the Sr. PCB engineer of ZTEC instruments. The 2nd revision was cleaner in design, with a power planes filtered and less traces. Starting the PCB design without integrating manufacturing constraints into design rules led to fabrication holds.

Thus the final design had to be tweaked to fit the manufacturing capabilities of Advanced Circuits our fabricator. DFM rule checking constantly rechecked our updated gerber files until the PCB could be manufactured. We changed silkscreen text fonts, trace widths, readjusted footprint soldermask clearances, and fixed spacing violations. The "show stoppers" to manufacturing were eliminated, and we proceeded to order our PCB. Electrical impedance checks were done manually with a multimeter to validate fabrication accuracy, and the power nets were isolated, so the assembly process could start. Budget constraints forced a hand-assembly, but assembly soon revealed a list of other problems. A major problem was that +3.3V and ground were shorting after the PCB was almost completely assembled. Initial guesses suspected a connection was taking place via a part that shorted its terminals when off, but advice from the Principal Design Engineer at ZTEC Instruments quickly dismissed the idea. Visible shorts were found on our accelerometer, whose solderpaste bubbled after reflow in the senior design lab, forming shorts on some power nets. After cleaning and eventual removal, the short lingered. Finally after referencing a working breadboard circuit, the humidity sensor was the culprit with a bad footprint; shorting a power pin with a PCB ground pad. The GND pad was insulated with kapton tape then covered with copper tape and then the part resoldered.

Other checks showed that the barometric pressure sensor had a similar problem. It had all power but no ground trace, so one power trace had to be cut and the ground pin wired to ground. The SD card connector also didn't fit perfectly and the footprint pinout belonged to an SD card, not the desired microSD pinout. So 30 and 28 AWG wires fixed that sequence problem, but yet another surfaced. The SPI modules in the datasheet for SPI ICs didn't specify explicitly that pull-up resistors were needed on some lines, and so the PCB was designed without them. Breadboard experiments showed that they were necessary for SPI to work, thus 10K resistors were added to the gyroscope, accelerometer, and SD card's SPI interface connections and pulled-up to +3.3V. Also, ignorance of MOSI and MISO SPI lines caused us to have MISI and MOSO connections. Fortunately for the gyroscope, mislabeling the SDI and SDO lines prior to fabrication fixed the MOSO and MISI problem. Thus two wrongs did make a right for the gyroscope lines. The accelerometer had the problem of its MOSI line going to a input line and it's MISO going to a I/O line on the MCU. We planned a 4-wire SPI operation, but since the problem it was decided best to operate the device in 3-wire mode, allowing the slave's SDO and SDI lines to short internally and utilize the SDO line as it now becomes an I/O pin that's on a reprogrammable I/O pin on the master MCU.

This solution provided us an alternative to scraping the SDI and SDO traces that were less than 0.01" thick and less than 0.02" apart, and swapping them. Lastly, because high assembly costs were not known during the design phase due to our ignorance of the common manufacturing procedures of buying reels or other packaging that contains more parts than that required for a single build, and laser-cut stencils etc., we replaced small components like 0402 resistors and capacitors, in favor of larger 1210 and 1206 types in anticipation of hand soldering all parts to the board, which was the only thing we could afford. In all the problems we found solutions, and because of that, experience was had in soldering, making breakout boards, and being innovative. A 3D representation of the board is shown in Figure 6 below.

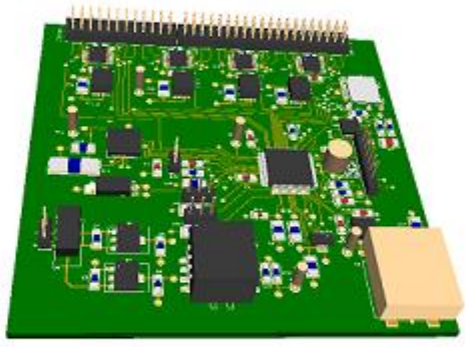

Figure 6 - 3D representation of manufactured PCB Board (Generated using design software)

Thus we've learned to get parts as early on and test them on custom breakouts on perfboards or bread boards, then transfer circuits known to work to our PCB, rather than simply trust datasheets circuits, which can lack details about things readers are assumed to know; like SPI connections. For this reason of building before testing, we built a 4x4 sq.in PCB rather than a rectangular board. Though we fit our PCB into the RC plane, initially we didn't have the plane to take measurements to constrain the PCB dimensions by, and thus made guesses from scale pictures of the plane, since our sponsor purchased the plane after we ordered the PCB. We fixed other problems like a badly designed on-off switch, needing a SPST instead of a SPDT switch. A toggle switch from RadioShack fixed the problem. We also built a custom RJ-11 cable for the PC interface because we had the problem of reversing the RJ-11 pinout on the PCB for ICD3. However the PCB powered on without smoking and the PC successfully communicated & programmed the MCU via Microchip's ICD3.

V. Software System

The software application takes the data from the sensor array and outputs a ".csv" file. The software is needed to interpret the data gathered by the array. With the data that is calculated by the software the client will has a readable sheet of information. The sheet can then be used in comparisons. This is to allow the client to compare the data collected with the data that comes from there simulator.

The software is independent of the hardware so that there is more portability of the system. Rather than doing the calculations on broad and wasting memory and processor time the data is stored and then transported to a PC via SD card. With the on board processor free more data collection points can be made and therefor more accurate results can be produced. From there the program will use aerospace formulas to find the values needed for the client to compare.

The software takes the raw data from the file and parses it. These data points are put into structs. Each data point, which represents one collection of all the sensors, or one time unit, is in its own struct. Then the program runs through loops to find the values needed. Once done there is about as many different values as there are data points. These values are then averaged so it produces the results required by the client. Quality of the produce was tested by making a small sample and calculated by hand.

Our software is a very simple interface that requires very little user input. We wanted to keep it as simple as possible because the application does not require an extensive user interface and so we could focus more energy into other aspects of the software. The interface has three buttons and two text fields and is displayed below in Figure 7.

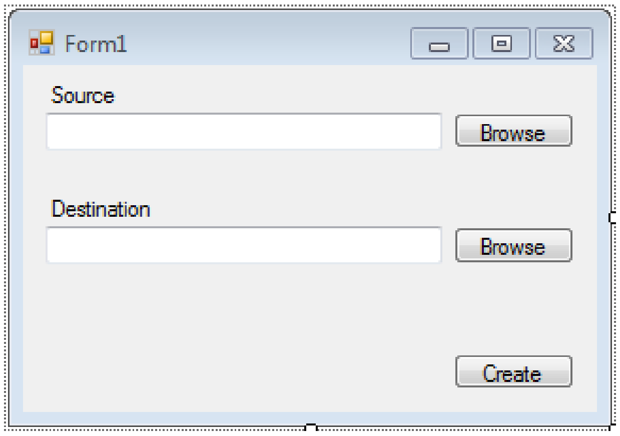

Figure 7 - Application Interface

Two buttons are browse buttons that help specify the source and destination of the files and the last one to run the math and create the output file. The text fields specify the source and destination files. When the “create” button is clicked the meat of the program runs. First it does checks to make sure the user put in the correct file data. There is one function that loads and parses the data from the file. Then there are a few functions that find the derivatives of the data that was found. To find the derivative at a point we must find the weighted average of the two slopes on either side. For example if we have point 1, 2, and 3 then the slope of 2 will be the weighted average of the slop from 1 to 2 and from 2 to 3. The equation is derived below. As it turns out the weighted average of the two slopes around point 2 is the same as the slope from point 1 to 3.

$$m_2 = \frac{(x_2 - x_1) * \left(\frac{y_2 - y_1}{x_2 - x_1}\right) + (x_3 - x_2)\left(\frac{y_3 - y_2}{x_3 - x_2}\right)}{(x_2 - x_1) + (x_3 - x_2)} \quad (8)$$

$$m_2 = \frac{y_2 - y_1 + y_3 - y_2}{x_2 - x_1 + x_3 - x_2} \quad (9)$$

$$m_2 = \frac{y_3 - y_1}{x_3 - x_1} \quad (10)$$

A series of math functions that find the results that are output. Finally there is a function that creates the “.csv” file from the results found. This condenses the potentially thousands of data points to just a few averaged values that our client wants.

Two data structures are used both are Linked List. Linked List is used for saving on space and overhead. Also the data is only accessed sequentially and is of unknown size. The first Linked List holds the initial data that is found from the parsed data. The second holds the data that is calculated after it is put through the functions that calculate it.

After thorough design and planning, we decided to implement a plan to approach and solve our software design. The life cycle chosen was the waterfall model as shown below in Figure 8.

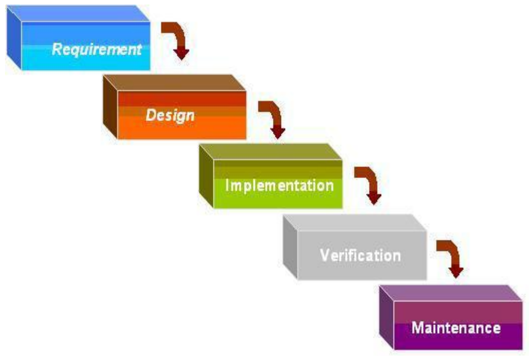

Figure 8 - Software Design Waterfall Model

Only one person worked on the software so it was simple to use this model and the benefits of this model outweighed the other approaches available.

VI. External Components

Although our project is focused around the extrapolation of data, we need a plane to install of our sensors on. In addition, L3 request that we use an autopilot to create repeatable flight patterns for our test. Both of these parts are considered external components because they are not vital for our sensors to operate, but an airplane is required for the sensors to record valid data.

The airplane we decided on for testing is a model replica of the MQ-9 Reaper, commonly known as the military’s Predator drone. The Reaper has a wingspan of close to 8 feet and will be the closest simulation to an actual small fixed-wing aircraft. The most important factors in deciding on this plane was the wingspan and its likeness to actual planes that are used in various military applications.

Seeing as this plane is very large and a scale model of the actual MQ-9, the plane manufacturers allowed the users ample room and ability to implement features uncommon to other remote controlled planes. The most interesting of the extra available features, the plane came with a rotary shell beneath the cockpit that had space for a rotating camera to be installed. Unfortunately, our project didn't supply adequate resources for us to successfully implement this feature. In addition to the camera, we had ample space available to route our many strands of wire through the plane and install the ArduPilot.

Although the model plane had plenty of space available at the start, we eventually utilized almost the entirety of the free space in the plane. This was a huge problem at first because all of the external sensors (force sensors and angle sensors) were already installed, but our board was unable to fit into the plane. After a few modifications to the layout of our system and the plane's body, we were able to force our finalized PCB into the top half of the hull of the plane. In a few instances the current location is ideal since the lengths of some wires are made slightly shorter and we avoided cutting into the fuselage of the plane. On the other hand, since our movement is being recorded at the head of the plane it may not be as accurate had we recorded the data closer to the plane's center of gravity. But it is important to note that the benefits of the approach taken outweighs the option of making a huge cut into the plane's fuselage and risking flight of the plane itself. To go off of the point of accuracy, we also had to place a good bit of focus into creating valid accurate test cases. Since we are trying to obtain data that is highly accurate, repeated test cases are very vital to being able to validate recorded data – the autopilot helps us achieve that goal.

We are using the open-source ArduPilot to handle our autopilot movements. In order to adhere to FCC regulations, we have set our autopilot to run in two modes – manual and waypoint loop. Manual mode allows for us to control the plane via the remote control. Whereas the waypoint mode makes the autopilot take control of the plane and follows a set of GPS waypoints to a set destination. Our approach is to set in a small round track of waypoints and repeat the circuit three times for the autopilot to follow. Our reasoning to use two different flight modes on the autopilot is so that in the rare event that the autopilot goes haywire, we are able to commandeer the plane back and allow a human to control the plane to safety.

Again these devices do not directly affect our system functioning properly, but they allow us to test our prototype in actual flight environments.

VII. Conclusion

After reaching out into a different area of aerospace studies, we feel that are project will be of help to L3 to study and confirm findings for their future research of small fixed-wing study. Although there may be other devices similar to ours that record plane movement, ours is separated from the others because we are analyzing more aspects of the flight characteristics. Our experiences with this were challenging and interesting – we had to study various aerospace terms in addition to applying our Electrical and Computer Engineering knowledge. Lastly, we would like to acknowledge the following individuals for offering their time and knowledge to build our prototype: L3 Communications for sponsoring, Merrill Lay for project guidance, Rebekah Reams and Zhihau Qu for plane tips, and finally, Michael Bosse and Alan Beauchamp for microcontroller assistance. In addition the authors of this article would like to thank the professors and faculty – Avelino Gonzalez, Zhihau Qu, and Chung Yong Chan – for serving as a part of our review committee.

Meet The Engineers

Winston James, an Electrical Engineering major. He designed and manufactured the PCB board for the Flight Data Logger. He hopes to do more PCB designing in the future especially for digital signal processing and wireless communications applications.

Brian Lichtman is a Computer Engineering major. After graduation, Brian looks to become a full time Software Developer for L3 Communications. His responsibility was to interface the MCU with the SD card.

Shaun Mosley, a Computer Engineering major. Shaun led the interfacing of the sensors and the microcontroller, in addition to studying the plane and autopilot. He is currently pursuing opportunities to work as a Software Developer for several companies in Atlanta, Georgia.

John (Tony) Torres, another Computer Engineering major. He headed the software application development for this project. His current plan is to work at Cognizant Technology Solutions in Tampa Bay, Florida as a Java developer upon graduation.

References

[1] **Davis, Leroy.** MultiMedia Card Pinout. *interfacebus.com.* [Online] March 3, 2011. [Cited: July 20, 2011.] http://www.interfacebus.com/Multi_Media_Card_Pinout_MMC.html.

---

[i] http://www.sensortechcorp.com/pressure_maps.php#

[ii] http://store.diydrones.com/Kit_MPXV7002DP_p/kt-mpxv7002dp-01.htm

[iii] http://store.diydrones.com/Kit_MPXV7002DP_p/kt-mpxv7002dp-01.htm