Worcester Polytechnic Institute

A Major Qualifying Project

AURORA
Autonomous Unpowered Recovery of Radiosonde Aircraft

Submitted By:

Richard Eberheim, Robotics Engineering

Nicholas Hassan, Robotics Engineering and Electrical & Computer Engineering

Joshua O'Connor, Mechanical Engineering

Advised By:

Kenneth Stafford, Professor

Robotics Engineering, Mechanical Engineering

Fred Looft, Professor

Electrical Engineering

Abstract

This project developed an autonomous radiosonde glider that actively steers itself from the apex of its flight to safe recovery locations on the ground. This enables easy and reliable recovery, reducing costs and offering new capabilities to atmospheric researchers. The glider integrates the essential weather sensors used on current radiosondes with those needed for autonomous flight in a durable, easy to manufacture airframe capable of multiple data gathering flights with minimal repairs between each flight.

Acknowledgments

This project was made possible through the support, guidance, and assistance of the staff and students of Worcester Polytechnic Institute. We would like to thank Professors Ken Stafford and Fred Looft for advising the project. We would also like to thank Keshuai Xu for assisting us with soldering, Bill Jones from Intermet Systems for his help on the radiosonde design, and Raymond Ranellone for loaning us a regulator for our helium tank.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Every year, the U.S. National Weather Service (NWS) launches over 70,000 weather balloons to study the atmosphere. These balloons provide the most cost effective way of recording important atmospheric data up to the very edge of Earth's atmosphere. Each balloon carries an instrumentation package called a radiosonde.

Radiosondes weigh between 250 and 500 grams and transmit data such as GPS location (for winds aloft), temperature, humidity, and pressure back to a ground station for the duration of the flight. Radiosondes used by the National Weather Service are the most common models; they ascend by balloon at a rate of 1000 feet per minute and transmit the data they collect using 300 milliwatt or less transmitters on the 400 MHz meteorological band. Once the balloon bursts, the radiosonde falls back to earth with a small parachute designed to prevent it from hitting the ground hard enough to harm people or property. [1]

Unfortunately, of the 70,000+ weather balloon launches per year, less than 20% of the radiosondes are recovered. There is no reason the radiosondes cannot be reused mechanically or electrically, the issue lies in the lack of a cohesive recovery effort. The only means for radiosonde recovery is to be found by a passer-by who sends it back to the NWS using the mailer included with every unit. At a cost of roughly $290 per unit, the annual cost of lost radiosondes is over $16.2 million.

This project aims to solve this problem by incorporating the radiosonde into a small UAV capable of autonomously selecting and flying to a safe location after disconnecting from the balloon at apogee. This will greatly increase the recovery rate of radiosondes and will reduce the pollution that results from their use.

Radiosondes collect data for computer-based weather prediction models, forecasting, weather and climate research, air pollution models, and verifying satellite data. [1] A radiosonde-glider capable of flying a preprogrammed path with substantial cross range capability will open up new research opportunities in areas such as extreme weather. A reusable non-powered flying radiosonde that can be launched a safe distance from hazardous weather would be an invaluable tool for scientists wishing to study those phenomena without risking expensive equipment or lives. [2]

The radiosonde we intend to develop is a technical challenge in every aspect that will draw on all of our experiences at WPI. It will involve the design, implementation, and testing of both mechanical and electrical systems, as well as the writing and debugging of a substantial amount of software.

2. Background

2.1. Introduction

Radiosondes have been used for years as an effective, inexpensive and simple tool to obtain accurate data about conditions in the lower to upper atmosphere. The technology and process for using these radiosondes has changed very little since the very first units, except for the recent addition of lightweight GPS modules. Thus, the radiosonde industry presents a market ready for changes and improvements.

2.2. Current Radiosonde Technology and Operations

There are currently two types of radiosondes in use by the National Weather Service (NWS): The LMS-6 by Lockheed Martin and the RS92-NGP by Vaisala. [3] There is another class of sensor, known as the dropsonde, which is not lifted by balloons, but is instead dropped from aircraft in order to study large weather phenomena like hurricanes. Most radiosondes have a mass between 250g and 450g and are used to measure temperature, air pressure, relative humidity, wind speed, and direction.

Each flight can last over 2 hours, reach a maximum height of 35,000 meters, and cover more than 320 km. The balloons are inflated with helium or hydrogen to about 1.5 meters in diameter at launch and climb at nearly 5 meters per second, growing up to 8 meters in diameter before bursting.

Each radiosonde model is capable of three distinct functions. 1) They can be tracked either by radar or GPS in order to establish wind speed and altitude. 2) They can transmit data from their sensors until balloon burst. 3) They will not cause damage to people or property; meaning they will land at a safe velocity and are made of materials that can be consumed by a jet engine in the rare event an aircraft strikes the radiosonde.

Radiosondes are lifted by a weather balloon filled with either helium or hydrogen. The radiosonde is suspended about five meters below the balloon and remains attached until the balloon expands to about four times its initial diameter and bursts. [3] At burst the radiosonde is considered destroyed and tracking data is collected during descent until contact is lost. [4] [5] As the radiosonde falls back to Earth, a small parachute and what remains of the balloon slow its descent. The descent of the radiosonde is completely uncontrolled and no concerted effort is made to recover it. The only chance for recovery is being found by a passerby. From there the radiosonde is either returned to the National Weather Service via mail or thrown away at the finder's discretion. [6]

2.3. Sounding Data

The data returned from a radiosonde flight is generally formatted into what is known as a Skew-T diagram:

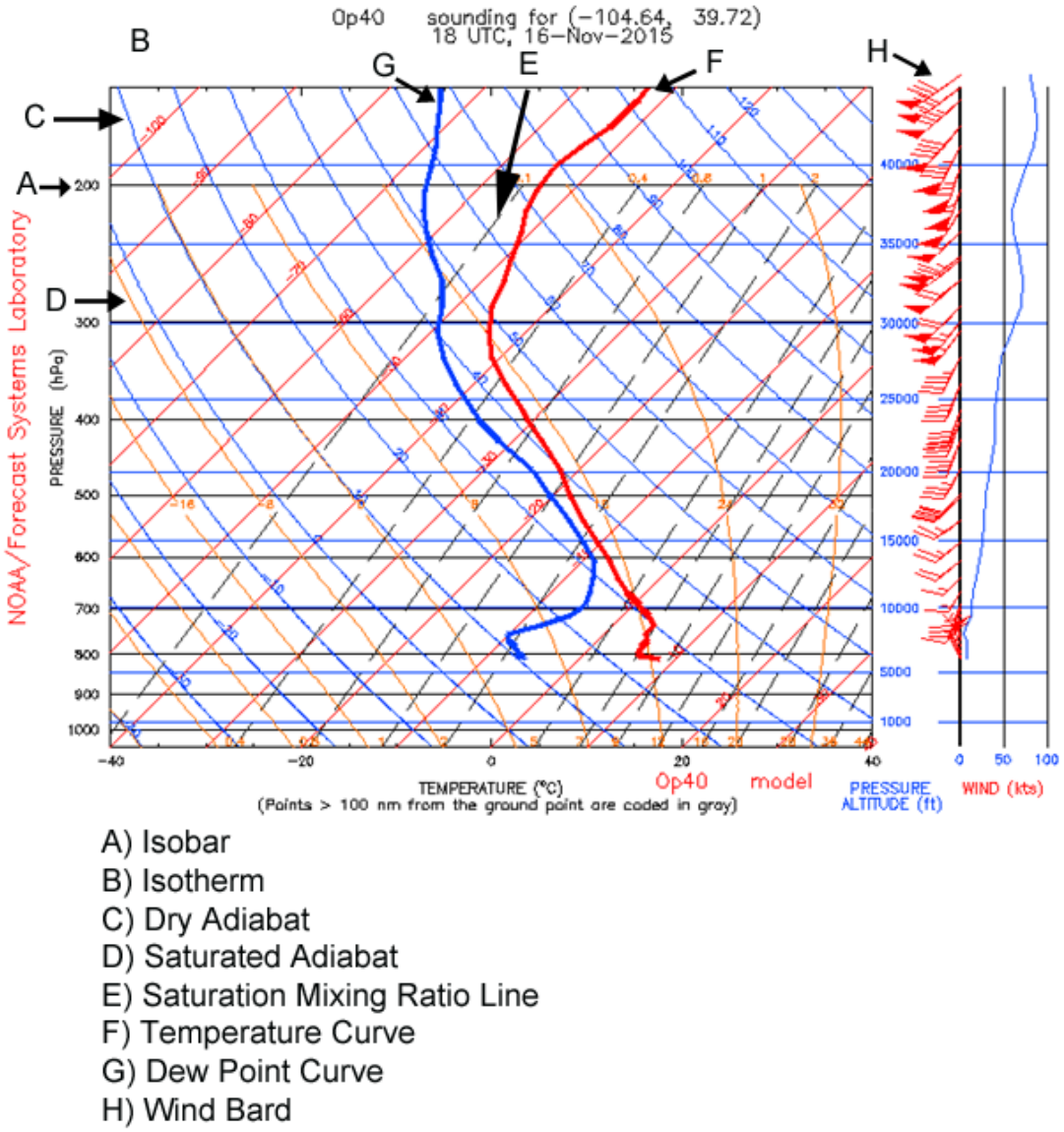

Figure 1: Example Skew-T Diagram [7]

These graphs use radiosonde data to show the temperature, dewpoint, and wind speed and direction at a range of pressures starting from ground level. The horizontal black lines are called *Isobars*, and represent lines of equal pressure. Spacing between *Isobars* increases vertically due to the logarithmic representation. The straight red lines which slope from the lower left to upper right are called *Isotherms*, and they represent lines of equal temperature.

The slightly curved blue lines which slope from the upper left to lower right are *Dry Adiabats*. These lines indicate the rate of temperature change in a dry air parcel rising or descending adiabatically. The slightly curved, orange lines sloping from upper left to lower right

are *Saturated Adiabats*, and indicate the rate of temperature change in a saturated air parcel as it rises pseudo-adiabatically.

The dashed black lines sloping from lower left to upper right are known as *Saturation Mixing Ratio Lines*, and represent lines of equal mixing ratios of saturated air. The bold red curve is known as the *Temperature Curve*, and is a plot of the temperature measurements. The bold blue curve is the *Dew point Curve*, and is a plot of the dew point measurements taken by the radiosonde. On the right of the diagram is a section showing the wind speed and direction measurements, represented with *Wind barbs*.

The information presented in these diagrams tells much about the atmosphere in a given location. A quick look at the slopes of the temperature and dew point curves can give one a sense of the stability of the atmosphere at various altitudes. This data can be used for anything from aiding weather predictions to showing glider pilots when and where to fly. [8]

2.4. Current Recovery Efforts

Current radiosonde recovery efforts do not employ any active system. Every radiosonde is launched with a prepaid mailer that can be used to mail it for free in the US back to the NWS for reconditioning. Whether a member of the public stumbles upon the radiosondes and then returns it is left up to pure chance.

In an interview of a professional in the radiosonde industry, the team was told of a NASA project that was supposed to be researching the possibility of having radiosondes land directly at post offices; however later research to find said project did not return any results. [4] The concept did, however, provide a starting point for defining the "safe landing zones" that the project's radiosonde would be aiming for.

2.5. Acquisition of Background Information

To gain a better understanding of what requirements and constraints will be relevant to this project; stakeholders such as the NWS and Intermet (A manufacturer of radiosondes) were contacted to learn more information about these programs.

Bill Jones of Intermet Systems was interviewed on November 10th, 2015 about the radiosondes that Intermet Systems makes and some of the engineering challenges associated with radiosondes.

2.6. The Atmosphere

The purpose of any radiosonde is to collect up-to-date information on the conditions present in the atmosphere. This data is fed into forecast and climate models, as well as standard models of the atmosphere; such as the 1976 U.S. Standard Atmosphere. The U.S. Standard Atmosphere ranges from -5000 meters to 100000 meters relative to sea level. It contains data on temperature, pressure, air density, viscosity of the air, and other altitude-related properties that are not directly connected to the air itself. The atmosphere is also divided into layers known as the: Troposphere, from ground level up to 11,000 meters above sea level; Stratosphere, 11,000

meters to 51,000 meters; Mesosphere, 51,000 meters to 71,000 meters; and Thermosphere, 71,000 meters and up. [9]

Nearly all water vapor and weather phenomena is found in the troposphere. The troposphere is the most turbulent and is greatly affected by oceans and terrain. The stratosphere has stronger winds and contains the jetstream. It is much drier, less dense, and colder than the troposphere and unaffected by oceans and terrain. Radiosondes probe these two regions the most because the balloons that carry them burst around 35,000 meters, or about halfway through the stratosphere.

2.7. Conditions to Expect

Flight conditions change with altitude. Aircraft flying in the lower troposphere are susceptible to terrain hazards, weather, bird strikes, thermals, and other phenomena. It is however easier to fly at slower speeds at these lower altitudes due to the thicker air. Aircraft do not have to be insulated or pressurized since conditions are fairly close to what is encountered on the ground. This changes around 5,100 meters, when the air becomes too thin for humans to breathe without special equipment.

As an aircraft climbs higher into the atmosphere, the temperature and pressure both drop. Between the upper troposphere and lower stratosphere, icing on aircraft can become a serious problem. Aircraft that fly in this region must be fitted with anti-icing equipment such as heaters, airbags, or special chemical coatings. The lower stratosphere is where most commercial air travel flies. Here the atmosphere is more uniform and much less chaotic than the troposphere. High altitude winds are dominant here, reaching speeds as high as 110m/s. [3]

The conditions that a radiosonde is expected to travel through and take data from are diverse and challenging. For a project at the scale of an MQP, every condition cannot be designed and tested for with the resources available. In order to narrow the scope of the project, specific criteria were selected that would result in a recoverable radiosonde that could perform all of the basic functions of current radiosondes.

3. Project Goals and Requirements

3.1. Goals for Project Success

The following criteria were set for measuring the success of this project:

- Develop a control board for vehicle navigation with similar data recording capabilities to current radiosondes
- Develop software capable of choosing a safe landing location from a pre-programmed list while avoiding hazards
- Develop an unpowered aircraft which can cover at least 7000 meters aerial range for every 1000 meter of vertical drop
- Must land in a condition such that it can be returned to flight without unscheduled repairs or maintenance.
- Use materials safe for consumption by a jet engine
- Test the full scale glider in small scale drop from a height of at least 1000 feet
- Unit cost under 200% of the cost of current, semi-recoverable radiosondes
- Cost per launch should be no more than the cost for current radiosondes
- Develop a CONOPS of how our radiosonde can be used

3.2. Develop Flight Controller with Radiosonde Abilities

For this project, a custom Flight Controller PCB must be designed to incorporate the tools necessary for autonomous flight control with those necessary for radiosonde functionality. Commercially available flight control boards (such as the Pixhawk) could be used, but these would lack the weather sensors and radios required, meaning additional hardware would still be necessary. In order to make the lightest, most capable, and most application-specific device, a custom Flight Controller must be created.

3.3. Develop Software Capable of Selecting Landing Site

Software written for this project must enable the radiosonde-glider to record and transmit data during ascent. It must also be capable of computing a flight path to the most reachable landing site from a pre-loaded list. Each landing site entry will contain information about the location and altitude of the landing site. Additionally, it will contain information about the heading that the craft should follow during the landing. Criteria for choosing the best landing site must include the wind data recorded during ascent. The software must make the best effort to be as redundant as possible in case of multiple system failures, as there is no human fallback option.

3.4. Develop Unpowered Aircraft Capable of 7:1 Glide Ratio

The airframe needs to rugged, inexpensive, and simple to manufacture. The airframe must be able to survive a wide range of temperatures, pressures, and flight regimes.

3.5. Able to Land and Return to Flight with Little Maintenance

A major goal for this project is to create a radiosonde-glider which is able to be readily reused. For this to happen, it must be able to land in such a way that it does do significant damage to itself. Though a small amount of regular maintenance would be acceptable to ensure it is still in flying order, any damage upon landing which prevents it from flying again would be unacceptable.

3.6. Use Materials Safe for Jet Engine Consumption

In the interest of safety, one of our requirements for this project is for all materials of the radiosonde and balloon to be able to safely pass through a jet engine without harming the aircraft. Materials such as foam, fiberglass, and small electronics are all safe in this regard, while those such as large pieces of metal would need to be avoided.

3.7. Test Glider from 1000 Feet

In order to sufficiently test all systems of the radiosonde-glider, a goal was created to perform a 300 meter tethered balloon flight. This test would involve setting up the radiosonde-glider on a balloon just as if a full-scale flight, and having it release from the balloon at an altitude of 300 meters and land with 3 meters of a chosen approach vector. This test is meant to prove all features of deployment, data collection and transmission, and flight parameters.

3.8. Unit Cost Under 200% of Current Radiosondes

In order to ensure the radiosonde-glider would be able to be marketed in the real world, an objective was set to keep the total cost of the radiosonde (excluding balloons and helium) under 200% of the cost of current radiosondes, or $580. This number was chosen because this glider ought to be able to perform at least two flights without failure.

3.9. Cost per Launch Equivalent to Current Radiosondes

The cost of launching our radiosonde-glider should be no more than the cost of launching current radiosondes. To achieve this, the radiosonde-glider should be able launch on the same type of weather balloons used by modern radiosondes, using approximately the same amount of helium/hydrogen per launch.

3.10. CONOPS

A concept of operations (CONONPS) is a document which describes a number of important characteristics of the system, what it can and can't do, the stakeholders, its cost, and how it works. It is meant to give an overview of the system from the point of view of a user, and to communicate concisely the system characteristic to all stakeholders.

4. Flight Controller Implementation

4.1. Introduction

Creating a custom Flight Controller for this radiosonde-glider was vital to the success of this project. This section covers the selection of major components and the PCB design process of the Flight Controller and modules. This section also covers the testing and debugging of hardware as well as the coding and testing of software.

4.2. Flight Controller Requirements

For the Flight Controller to meet all criteria set by this project, it must be able to:

- Operate as a radiosonde during ascent
- Control the airframe during descent
- Transmit GPS location from landing site

Behaving as a radiosonde will enable this project to be a useful tool for collecting weather data. Meeting this goal will require a number of sensors and a method of transmitting data from those sensors to the ground. Most radiosondes include the following: a temperature sensor, a humidity sensor, a barometric pressure sensor, and a Global Positioning System (GPS) sensor for measuring winds aloft. These will all need to be incorporated into the Flight Controller to enable proper weather data acquisition. Transmitting this data to the ground will involve the use of a radio able to transmit a signal powerful enough to be received several hundred miles away. A MicroSD card reader may also be added to this Flight Controller, as this would enable the device to record weather data during ascent, which may then be used for calculating flight paths during descent.

The ability for this Flight Controller to control our airframe during descent is vital. Implementing this will require a number of sensors and several servos outputs. Included in these sensors should be an accelerometer, gyroscope, magnetometer, and GPS. These sensors will give the airframe full knowledge of its orientation, rates, and location. In addition, the MicroSD card mentioned above can also be used for storing the locations of landing sites for the Flight Controller to select from.

Knowing the landing location of this radiosonde-glider is essential to the operator's ability to recover it. This will be accomplished by transmitting a message from the ground with the devices GPS location. Such a radio will likely need to transmit to either a satellite or cellular network to ensure reliable delivery of the message, as failure to receive the location of the radiosonde-glider could easily result in the loss of the device.

4.3. Overall Design Considerations

Several decisions in the overall design of the Flight Controller were made at the start of the project. First, the decision was made to split the code between two microcontrollers. One microcontroller (referred to as the Navigation Computer) would collect and transmit weather data during ascent, and perform navigation calculations during descent. The other (referred to as the Autopilot Computer) would have the sole responsibility of performing real-time flight control

during descent. This system of two microcontrollers eliminates the potential for catastrophic latency in the flight control loop caused by GPS data acquisition and long calculations necessary for navigation. Such latency would cause the aircraft's control surfaces to essentially lock-up whenever the microcontroller computes a new flight path. Ideally, this latency would be no more than 20ms, as the servo PWM signal runs at a speed of 50Hz. The inclusion of a GPS was the main reason for this decision, as they require large amounts of processor time for reading and parsing data sent by the GPS.

Second, the decision was made to use Arduino-compatible Atmel microcontrollers. This decision would allow the use of Arduino's user-friendly take on embedded C/C++ to program the Flight Controller. The Arduino project includes a library called *Wiring*, which provides many common I/O procedures, greatly simplifying the embedded coding process. It also allows for the same piece of code to be run on a multitude of microcontrollers, and is backed by an enormous, helpful community of developers. Using this would save valuable time and effort in the programming process.

Third, it was decided that the flight controller should use 3.3V logic wherever possible. This decision was made based on the fact the most modern-day digital sensors operate on 3.3V. This design choice would eliminate the need for multiple voltage regulators and a multitude of logic level converters.

Fourth, the decision was made to use components with very low operating temperatures. Given that this Flight Controller is meant to operate at up to 30,000 meters, the ambient temperatures experienced will go as low as -57°C. [10] Research has shown the vast majority of electronic components are given operating temperatures no lower than -40°C, so this was chosen as the minimum operating temperature for all components (with the exception of those only meant to operate at lower altitudes, such as those on the GSM module). This was deemed acceptable given the thinness of the atmosphere at altitude and the insulating nature of the airframe.

Finally, it was decided that the PCB would be ordered from OSH Park. OSH Park is a U.S.-based, community PCB manufacturer with a quick turn-around time and small minimum order of 3 PCBs. This would require the PCB to be designed to their specifications, which simply limit the minimum trace widths and drill sizes. [11]

4.4. Selecting PCB Design Software

At the start of this project, the decision was made to design the PCB for the Flight Controller in Fritzing. Fritzing is a free, open source, user-friendly tool for virtual bread-boarding and PCB design. [12] After experimenting with the software and attempting the first design, it was found that Fritzing, while great for small DIY projects, is too slow and limiting for the purposes of this project.

Research was then put into finding more suitable software for PCB design. DipTrace was tried at first. [13] This software was quickly decided against due to its lack of a user-friendly interface, difficult-to-use part editor, and sharp learning curve. EAGLE CAD was then

tested. This software was found to have an intuitive interface, good part editor, and acceptable auto-routing, so it was chosen for chosen for this project. It was also chosen for SparkFun's vast library of EAGLE designs available to the public, and for OSH Park's support of EAGLE files. [14] [11] The light (free) version of this software is limited to board areas of 100 x 80 mm and two signal layers, though these were determined to not be an issue given the size limitations set by the airframe. [15]

4.5. Selecting Major Components

4.5.1. Microcontrollers

The microcontroller selected for this project was initially the ATmega328P. This was chosen for a number of reasons: It's familiarity to the team members and its Arduino compatibility. This is the same 8-bit microcontroller used on the Arduino Uno, which the team members had plenty of experience working with. The ATmega328P was found to be perfectly acceptable as a microcontroller for the Autopilot Computer given the number of I/Os necessary. The Navigation Computer was quickly changed from an ATmega328P when the need arose for multiple UART ports (necessary for GPS and GSM). The ATmega2560V was chosen for this role. This 8-bit microcontroller is the lower-voltage variant of the Arduino Mega's microcontroller.

Microcontroller	ATmega328P	ATmega2560V
Number of I/Os	23	86
Number of UARTs	1	4
Flash Memory	32 KBytes	256 KBytes

Table 1: Comparison of Microcontrollers [16] [17]

In order for these microcontrollers to communicate, Universal Asynchronous Receiver/Transmitter (UART) would be used. This was chosen for its availability, speed, and reliability.

Initially, the microcontrollers selected were to be through-hole mounted for ease of soldering and replacement. After deciding to use the ATmega2560V, which is an SMD-only chip, the SMD variant of the ATmega328P was chosen as well. Though this would be more difficult to solder, it would also take up less space on the Flight Controller.

4.4.2. Inertial Measurement Unit (IMU)

An IMU is a device which measures and reports a body's acceleration, angular rate, and (sometimes) magnetic heading. This device is critical to an aircraft's ability to fly autonomously, as it allows the aircraft to know its orientation and rates. Initially, the IMU was to be split amongst three separate sensor packages; one each for the accelerometer, gyroscope, and magnetometer, allowing them to be separately researched and specified.

Accelerometers that were researched include the ADXL345 and LIS331. The ADXL345 was rejected for its significant 0g offset due to temperature in the Z-axis (would be approximately -0.2925g at -40°C). [18] The LIS331 was a reasonable alternative, as it lacks this

extreme 0g offset. [19] The gyroscope initially researched was the mostly-temperature-stable ITG-3200. [20] The magnetometer researched was the MAG3110, which was found to have an unacceptable amount of zero-flux offset due to temperature change. [21]

Further research into sensors led to the discovery of the LSM9DS1, a 9 degree-of-freedom IMU. This sensor includes a 3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer, and temperature sensor. What is so intriguing about this sensor is the magnetometer's "set / reset pulse" system. This automatic operation performed before each data acquisition degausses the sensor and ensures alignment of the magnetic dipoles, effectively creating a magnetometer with a zero-gauss level independent of temperature. Unfortunately, the datasheet for the LSM9DS1 is lacking in information on the temperature sensitivity of accelerometer and gyro. [22]

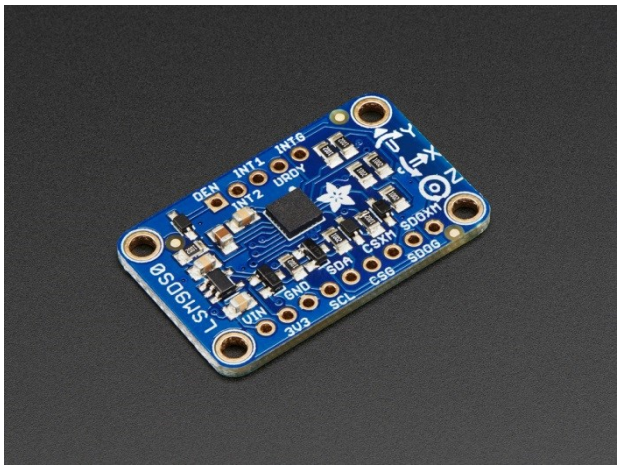

Figure 2: Adafruit LSM9DS0 Breakout Board [23]

Luckily, the datasheet for the LSM9DS0 contains all information necessary to make a decision. It contains the same temperature stable magnetometer as the LSM9DS1. In addition, the accelerometer and gyroscope show very little offset due to temperature, and what little there is can easily be compensated for using the built-in temperature sensor. The sensor works down to -40°C, can measure acceleration up to +/- 16g, and can measure rotational velocity up to +/- 2000dps. [24] The only downside found is the difficulty in soldering such a small device. Luckily, Adafruit sells a breakout board for this sensor, allowing it to easily be installed on the Flight Controller, so this IMU was selected for this project. [23]

4.5.3. GPS Unit

The GP-20U7 was initially chosen due to its low power consumption and very low cost (about $16). This GPS was later decided against due to its altitude limitation of 18,000 meters. [25] Research into this issue showed that all but a select few GPS units have this drawback. GPS units which take advantage of a uBlox chipset are an exception to this.

Figure 3: GP-735 GPS [26]

The GP-735, found on SparkFun, uses a uBlox 7th generation chipset and is able to measure altitudes up to 50,000 meters. Though more expensive than the GP-20U7, this unit is incredibly small and powerful, so it was chosen as the GPS for this project. [26]

4.5.4. Weather Sensors

Most radiosondes contain a barometer, temperature sensor, humidity sensor, and GPS. For the Flight Controller to behave as a fully-functional radiosonde, it would have to include these sensors. The temperature sensor was found on SparkFun as the DS18B20. This sensor is inexpensive and able to measure down to -55°C. It uses a unique 1-wire interface, is able to operate at the required 3.3V, and is a contained in a though-hole package which can easily be setup for off-board mounting. [27] The humidity sensor was also found on SparkFun. The HTU21D is a low-cost humidity sensor which communicates via I^2C. A breakout board for this sensor was found on SparkFun, allowing it to easily be mounted off-board. [28] The barometer took a significant amount of research to find. Most barometers, such as the BMP180, are only able to measure pressure up to about 10,000 meters. [29] Eventually, the MS5803-01BA was discovered. It is able to measure up to altitudes of just over 30,000 meters, communicates via I^2C, operates at 3.3V, and is easily SMD soldered to a PCB. [30]

4.5.5. Voltage Regulator

The voltage regulator for the flight controller had to meet several criteria. First, it had to output 3.3V and at least 200mA (calculated from maximum current draw of microcontrollers, sensors, and other 3.3V components). Second, it had to be highly efficient. Dropping from 6V+ to 3.3V using a linear regulator would be inefficient and produce a significant amount of waste heat. The solution is to use a step-down voltage regulator. Third, the stepping regulator had to be low-noise. Searching for regulators with these characteristics led to the discovery of the Traco Power TSR regulator series. These regulators are an all-in-one SIP package with built in filtering and short-circuit protection. The two regulators considered were the TSR 0.5-2433 (outputs 3.3V at 500mA) and the TSR 1-2433 (outputs 3.3V at 1A). The TSR 1-2433 was chosen because it has superior built-in filtering, weighs slightly less than the TSR 0.5-2433, and only costs a few dollars more. [31] [32] This would also increase the amount of current available to be drawn by any devices added later in the project.

4.5.6. Sensor/Servo Connectors

Figure 4: Right-angle 3-pin JST Connectors [33]

During flight operations, the Flight Controller may be subject to a rather significant amount of force. Any such force has the potential to dislodge or unplug anything loose, so selecting a robust connector type for the Flight Controller's external hardware was critical. JST connectors were chosen for this purpose. They use small plastic tabs to lock in place, requiring far more force to be unplugged than would be experienced during flight (and likely even during a crash). The mating ends are easily crimped and assembled, and they can handle 2A continuous. [33]

4.5.7. GSM Module: GSM Unit

The Iridium Communications Network was originally considered as a means of transmitting data. This network of satellites allows for global phone and data connectivity with essentially no dead-zones. The Iridium module would transmit weather data during ascent and location data during and after descent. The Iridium 9603 is the world's smallest commercially available two-way satellite data transceiver, and it would have served this purpose well. The reason it was not chosen for this project essentially came down to money. The unit itself costs roughly $185 and data plans for the unit cost $1.25 per kilobyte. Over the course of a single flight, this could easily add up to over $400 in data charges. [34] Replacing this system with something less costly would require two systems to be created: a GSM Module for transmitting location data after landing, and a Radio Module for transmitting weather data.

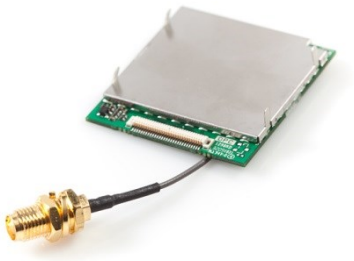

Figure 5: SM5100B GSM Unit [35]

The GSM unit selected was the SM5100B. This unit has a minimum storage temperature of -40°C (the operating temperature is less important since this will not be operating at high altitudes), and is able to transmit to the GSM850 band used in the United States.

Communication to the unit is done through a single UART port. [35] SparkFun once sold a GSM Arduino Shield which took advantage of this unit. EAGLE files for the shield are still available on their website and would help in designing the GSM Module PCB.

4.5.8. GSM Module: Voltage Regulator

The GSM unit chosen required a voltage of 3.6V with a current of 2A. Originally, research was done to find a step-down regulator with these specifications. Traco Power does not offer one with this output voltage, and those found on Digikey would require significant external hardware to function. Given that this module is only meant to run for a short time after landing, it was determined that the losses from using a linear regulator over a stepping regulator would be acceptable. The LT1085IM-3.6, a linear regulator with an output of 3.6V at 5A, was determined to be suitable for powering the GSM Module. [36]

4.5.9. GSM Module: Cellular Network/SIM Card

The cellular network which the GSM Module operates on is vital to our ability to receive an SMS message sent by the aircraft after landing. The cellular network had to meet two criteria: have a wide area of coverage across the United States (this would include major carriers such as Verizon, AT&T, T-Mobile, etc.), and offer an inexpensive, no-contract service plan. T-Mobile offers a 6-month unlimited SIM card for about $80. This was deemed too expensive for the purposes of this project. AT&T (which has arguably better coverage than T-Mobile) offers contract-free service through their GoPhone plans. This includes a plan which charges $2 per day on the days used for unlimited calls and text messages, an ideal plan for our project. Verizon's no-contract plans all involve a monthly payment, so the decision was made to use AT&T's GoPhone plan.

4.5.10. Radio Module: Transmitter

Significant research went into finding the best way to transmit data to the ground from up to 30,000 meters. One idea was to make use of the radio from an expired (but unused) radiosonde. This was quickly decided against given the proprietary nature of the radio and the requirement for a very expensive ground station to receive data. Further research led to the possibility of sending data with a basic "Morse-code" transmitter by turning a transmitters output on and off. This was decided against due to the extremely slow attainable data-rate and the requirement for a high-gain ground station.

Figure 6: Radiometrix HX1 300mW 144.39MHz Transmitter [37]

Eventually, a project called the Trackuino was discovered. This project is an open-source APRS tracker based on the Arduino platform, and is designed primarily to track high altitude balloons. The most intriguing part of this project is how it transmits data to the ground. Using a Radiometrix HX1 300mW 144.39MHz transmitter, it sends data to the Automatic Packet Retrieval System (APRS). The APRS is comprised of thousands of amateur radio operators around the globe. Data received by an APRS operator is found via the internet on aprs.fi. Using this system removes the need for any sort of ground station other than a laptop with internet access. Given these features, APRS was chosen as the means for data retrieval. [38]

4.5.11. Radio Module: Voltage Regulator

The Radiometrix HX1transmitter requires a 5V input at 140mA and uses 5V logic. To power this, a stepping regulator would be used. Stepping regulators are more efficient than linear regulators, and can both decrease and increase voltage. The TL2575-05I is a stepping regulator which outputs 5V at 1A from an input of 4.75V to 40V. Though this regulator requires external filters, this was acceptable given its ability to operate with a low input voltage. [39]

4.5.12. Batteries

The batteries for the Flight Controller had to meet several criteria. They had to have a very low operating temperature, very high energy density, and a high output current. The need for high energy density immediately spawned research into lithium batteries. Rechargeable lithium batteries tend to have poor performance in the cold. The lowest operating temperature found was with $LiFePO_4$ batteries, which have a minimum operating temperature of -4°C. This is far from the required minimum of -40°C. Further research led to the Energizer Ultimate Lithium AA battery. These use a lithium/iron disulfide chemistry and have a minimum operating temperature of -40°C. Though they are not rechargeable, they are incredibly light weight for their energy capacity, able to store just over 3000mAh in a 15 gram package. Four of these batteries would give a usable voltage of 6V. [40]

To verify the ability of these batteries to power the Flight Controller for an entire flight, the following calculation was done to estimate the total power consumption. This assumes a 4 hour flight (3 hours ascent, 1 hour descent), and excludes GSM transmission, as it does not take into account power consumption after landing.

Item	**Estimated Power Consumption**
ATmega2560V	~7mA for 4 hours = 28mAh
ATmega328P	~3mA for 4 hours = 12mAh
LEDs	Assume they are on 25% of the time: ~3mA * 4 hours = 12mAh
9DOF IMU	Up to 6.45mA * 4hours = 25.8 mAh
GPS	37mA * 4 hours = 148mAh
MicroSD	100mA for writes (5sec period for estimated 20ms) * 3 hours = 1.2mAh
Barometer	12.5uA * 4 hours = 0.05mAh
Temperature Sensor	1mA * 4 hours = 4mAh
Humidity Sensor	0.5mA * 4 hours = 2mAh
Airspeed Sensor	2mA * 4 hours = 8mAh
Voltage Regulator	~88% efficient (from datasheet) and 241.05mAh (above) = 32.9mAh

Servos	Estimated 250mA each & 50% duty cycle: 250mA * 1hour = 250mAh
Radio Module Tx	140mA at 20% duty cycle * 3 hours = 84mAh
Radio Module VReg	~77% efficient (from datasheet) and 84mAh (above) = 25.1mAh
GSM Module VReg	5mA * 4 hours = 20 mAh
Total	**653.05 mAh**

Table 2: Flight Controller Power Consumption

With a total power consumption of 653.05 mAh, this would leave the majority of the 3000mAh capacity of the batteries available for periodic SMS messages after landing, and provides a good safety buffer if there are additional losses not accounted for.

4.6. Designing the Flight Controller PCB

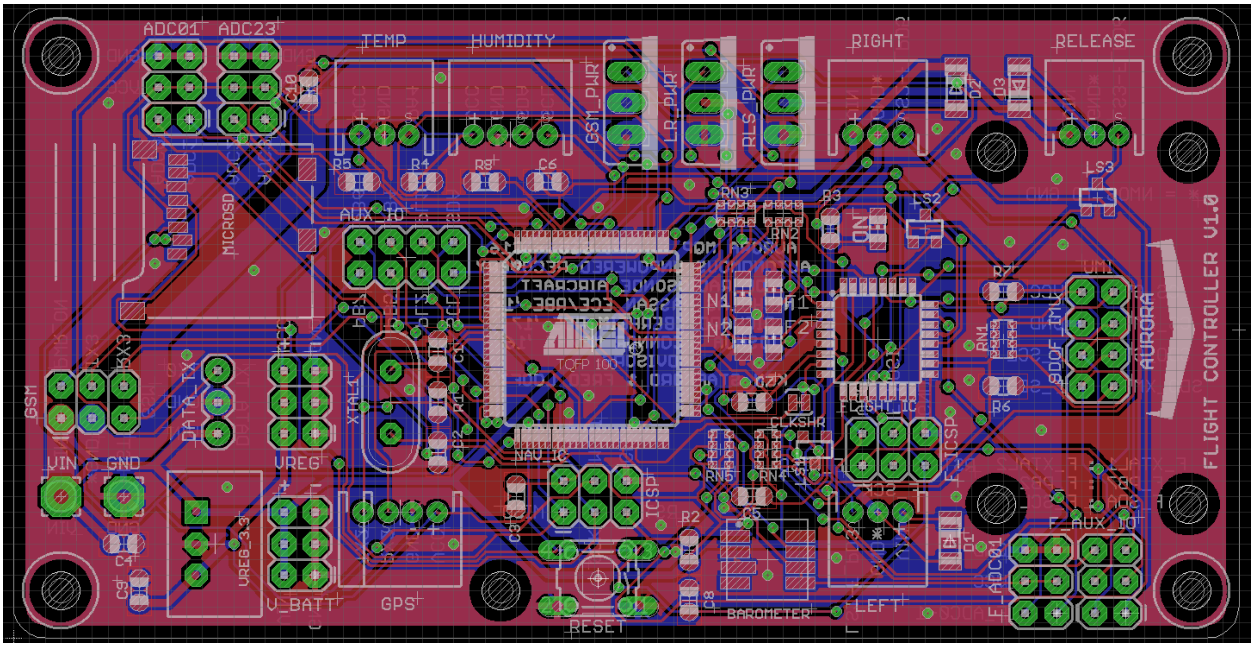

Figure 7: EAGLE Board View of Flight Controller

4.6.1. Single Crystal for Two Microcontrollers

The first step in designing this PCB (after adding the microcontrollers) was to add an external crystal oscillator. ATmega microcontrollers have an internal oscillator, but these are very inaccurate and vary in frequency based on a number of factors, including temperature. An external ceramic or crystal oscillator is used to achieve a steady, reliable clock frequency. For this project, ceramic oscillators would not work due to their sensitivity to colder temperatures, so a quartz crystal oscillator was selected.

In order to save on components and ensure the two microcontroller clocks are synchronous, a way had to be found to link a single oscillator to two microcontrollers. To solve this, the oscillator circuit was attached to XTAL1 and XTAL2 of the ATmega2560V, just as it normally would be. The CLKO pin (PE7) of the ATmega2560V will output same clock frequency as is being received from the oscillator. This pin was connected to the XTAL1 (clock input) pin of the ATmega328P. Changing a simple fuse setting in the ATmega2560V would enable this clock output and allow both microcontrollers to run off the same oscillator. [16]

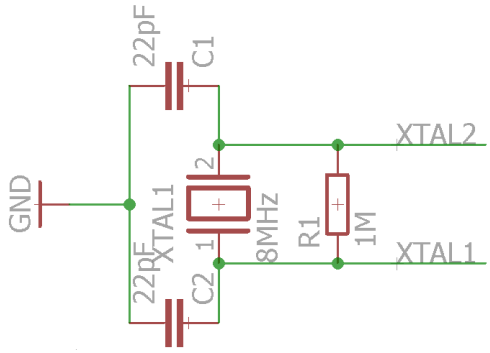

Figure 8: EAGLE Schematic View of Crystal Oscillator Circuit

The circuit for the crystal oscillator is shown above. An 8MHz oscillator was chosen because the microcontrollers are running at 3.3V. This circuit was taken from the Arduino Mega EAGLE schematic and verified on page 41 of the ATmega2560 datasheet. [17]

4.6.2. ICSP Headers for Programming

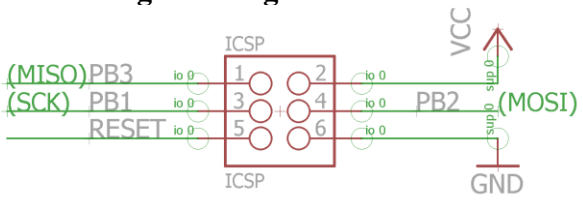

Figure 9: EAGLE Schematic View of ICSP Headers

In order to program the microcontrollers, headers were added for In-Circuit Serial Programming (ICSP). These pins are used to upload a program via an AVR programmer. The Atmel STK500 board is used as an AVR programmer for this project. The ICSP header includes ports for VIN, GND, MISO, MOSI, SCK, and Reset.

4.6.3. Reset Circuit

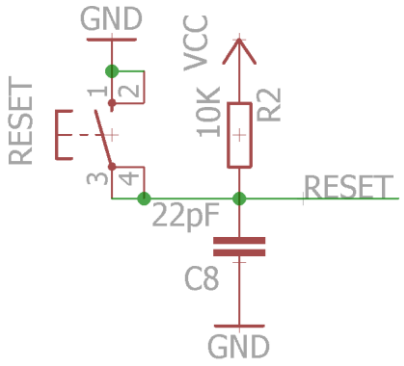

Figure 10: EAGLE Schematic View of Reset Circuit

A proper reset circuit is important to any PCB containing a microcontroller. The ATmega2560V and ATmega328P will reset if they're reset pin is pulled low, and will operate while it is pulled high. The circuit above is connected to the reset pins of both microcontrollers, so that one button will reset both. The pin is normally pulled high through a 10kΩ resistor, and is pulled to ground when the button is pressed. A 22pF capacitor bridges the reset pin to ground to help filter any noise on the line.

4.6.4. Voltage Regulator

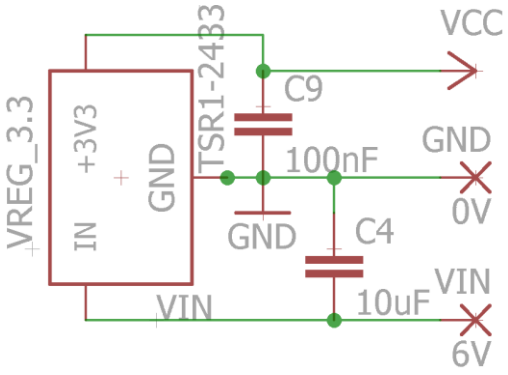

Figure 11: EAGLE Schematic View of Voltage Regulator Circuit

The TSR 1-2433 step-down regulator has three pins: IN, GND, and +3V3. IN is connected to the positive lead of the battery, GND is connected to common ground, and +3V3 is connected to the voltage input of the +3.3V device(s). While this regulator does not require any external components to function, two external capacitors were added for additional filtering. As suggested in the listing for this regulator on Adafruit, a 10uF capacitor was added across the input (IN to GND) for additional stability. A 100nF capacitor was added across the output for additional filtering. [31]

4.6.5. Indicator LEDs

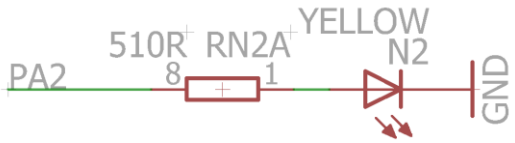

Figure 12: EAGLE Schematic View of LED Circuit

Light Emitting Diodes (LEDs) can be incredibly useful with microcontrollers for indicating software events. Four yellow LEDs were added to the Flight Controller for this purpose (two for each microcontroller), and one green LED was added to indicate that the board is receiving power. Sourcing the LEDs involved using Digikey's search tool to find ones with a proper forward voltage and a low operating current. The yellow LTST-C170YKT and green LTST-C170GKT were selecting using this method. They each have a forward voltage drop of 2.1V and a nominal operating current of 10mA. In the interest of saving energy, a 510 ohm resistor was added in series with each LED, limiting the current each LED can draw to about 2.35mA. [41] [42]

4.6.6. Inertial Measurement Unit

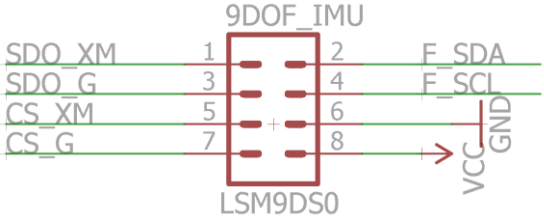

Figure 13: EAGLE Schematic View of 9DOF IMU Headers

The pins of the LSM9DS0 breakout board needed for the Flight Controller are VIN (soldered to 3V3), GND, SCL, SDA, CSG, CSXM, SDOG, and SDXM. A header for these pins

is shown in the image above. VIN connects to VCC on the Flight Controller and GND connects to common ground. CSG, CSXM, SDOG, and SDXM are all pulled high through 10kΩ pullup resistors. This hard-sets I^2C mode, sets the gyroscope's I^2C address to 0x6B, and sets the accelerometer's and magnetometer's I^2C address to 0x1D. SDA and SCL are also pulled high through 10kΩ pullup resistors. These pins are connected the Autopilot Computer's SDA pin (PC4) and SCL pin (PC5) to enable I^2C communication. [23]

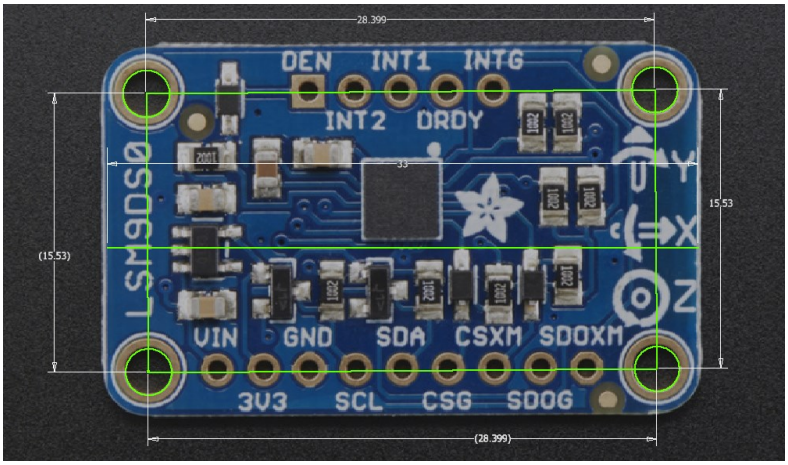

Figure 14: Using Autodesk Inventor to Measure Hole Dimensions

In order to mount the IMU to the PCB, mounting holes were required. Unfortunately, Adafruit supplies no information on the size and placement of mounting holes. They do, however, supply the outer dimensions of the board. Using this information, an image of the breakout board, and Autodesk Inventor, measurements of the holes were taken. An image of this is shown above. Mounting the IMU to the PCB would be done using nylon nuts and bolts, so as to avoid getting any ferrous metals too close to the magnetometer.

4.6.7. GPS and Weather Sensors

Connecting the GP-735 GPS requires four ports: GND, VCC, TXA, and RXA. VCC takes an input of 3.1V to 5.5V to power the unit. TXA is the serial data output and RXA is the serial data input. At the time of designing the PCB, the exact GPS to be used was unknown, but it was known that it would need VCC, GND, TX, and RX, so these were the ports designated for the GPS. This was done with the knowledge that, if any other ports were necessary for the GPS, the auxiliary I/O ports could be used. TX was connected to RXD2 (PH0) and RX was connected to TXD2 (PH1) on the Navigation Computer. Once powered on, the GP-735 returns standard NMEA sentences at a frequency of 1Hz. [26]

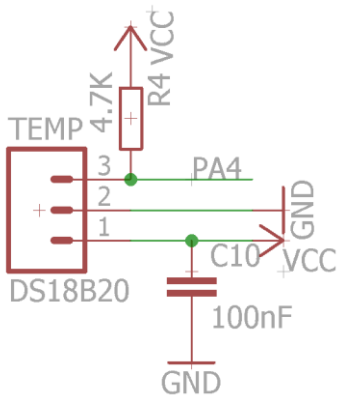

Figure 15: EAGLE Schematic View of Temperature Sensor Port

The DS18B20 temperature sensor has three pins: V_{DD}, GND, and DQ. Power is received on V_{DD} and GND, and data is sent from DQ. Following a tutorial on Bildr, a 4.7kΩ pullup resistor was placed from DQ to V_{DD}. A 100nF capacitor was added from V_{DD} to GND to filter the power line. [27]

The HTU21D breakout board has 4 ports: VCC, GND, SDA, and SCL. SDA and SCL are connected to the SDA port (PD1) and SCL port (PD0) of the Navigation Computer for I2C communication. Since this breakout board already includes a filter capacitor, no additional hardware was necessary. [28]

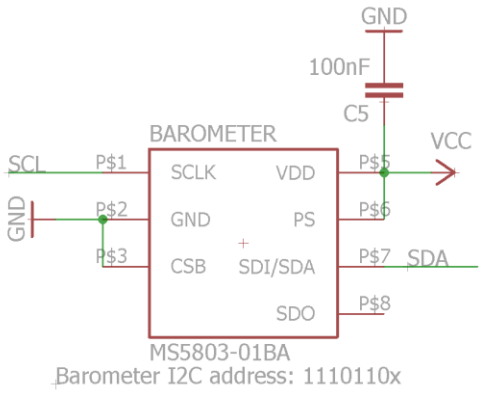

Figure 16: EAGLE Schematic View of Barometer Circuit

The MS5803-01BA barometric pressure sensor runs on 3.3V and is able to be setup for either I^2C or SPI communication. To configure this sensor for I^2C, SCLK was connected to SCL (PD0), SDI/SDA was connected to SDA (PD1), CSB was pulled low by jumping it to ground (in I^2C mode, CSB represents the LSB of the I^2C address, so pulling it low set the address to 0x77), and PS was pulled high by connecting it to VDD (hard-setting I^2C mode). No EAGLE part file for this sensor could be found online, so one was made using EAGLE's part creator and the recommended pad layout on page 15 of the datasheet. [30]

4.6.8. MicroSD Card

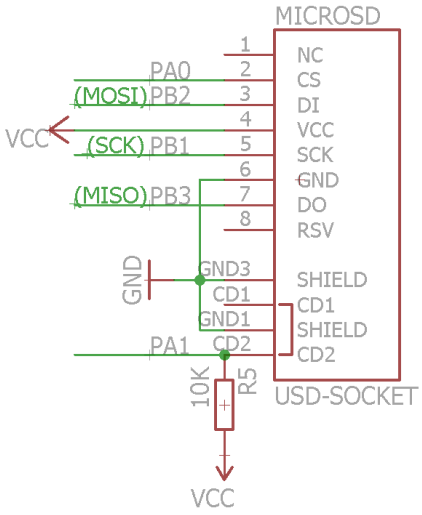

Figure 17: EAGLE Schematic View of MicroSD Card Slot

MicroSD cards run on 3.3V and use SPI as one method for communication with their host, which in this case is the Navigation Computer. To setup SPI communications, CS (chip select) is connected to PA0, DI (data input) is connected to MOSI (PB2), DO (data output) is connected to MISO (PB3), and SCK is connected to SCK (PB1). CD, a mechanical card detect feature, is connected to PA1, and the metal shield of the card reader is connected to GND. The EAGLE part file for this microSD card slot was taken from the SparkFun MicroSD breakout board. [43]

4.6.9. Communication between Microcontrollers via UART

Setting up communication between the Navigation Computer and the Autopilot Computer was simple. TXD1 (PD3) of the Navigation Computer was connected to RXD (PD0) of the Autopilot Computer, and RXD1 (PD2) of the Navigation Computer was connected to TXD (PD1) of the Autopilot Computer. These connections allow the two microcontrollers to communicate via UART.

4.6.10. Servos with Logic Level Converters and MOSFETs for Power Control

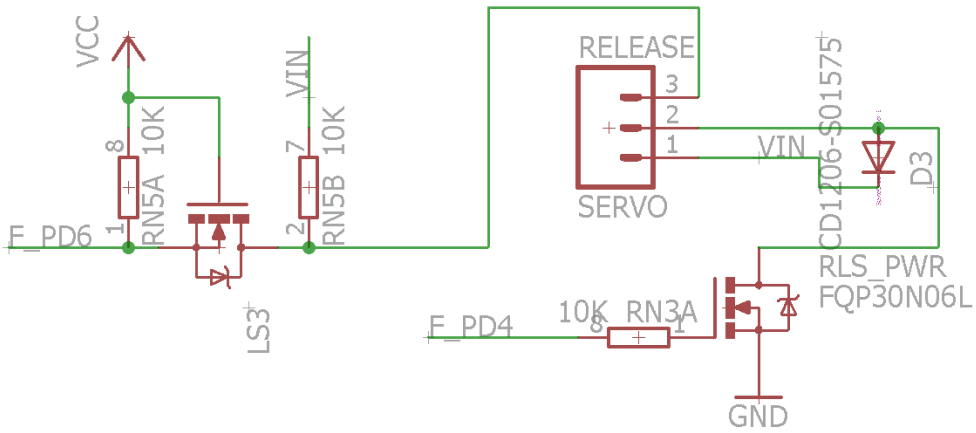

Figure 18: EAGLE Schematic View of Servo Port and Circuit

In order to fly the airframe, the Autopilot Computer must control several servos: two servos for the control surfaces (left and right), and one servo for the release mechanism. The servos are powered by battery voltage (6V) and controlled with a 50Hz PWM signal. The left servo takes a signal from PD3, the right servo from PD5, and the release servo from PD6.

In the interest of protecting the Flight Controller from any foreseeable problems, diodes were added across the VCC and GND lines of each servo. These would protect the board from any back EMF the servos may produce.

Due to concerns that a 6V servo may not operate with 3.3V logic, a logic level shifter circuit was added to each servo's signal line. Using the circuit shown above, a small N-channel MOSFET takes in a signal from the 3.3V output of the microcontroller and outputs a signal at battery level voltage. This circuit was taken from the EAGLE file for SparkFun's Bi-directional Logic Level Converter. [44]

In order to save power, MOSFETs were added to the power line of the servos. This allows the Autopilot Computer to cut power to the servos when they are not needed. While the power drawn by servos in idle may not be overly significant, it would certainly add up over the course of a several hour flight. One MOSFET was added for controlling power to the left and right servo, and another was added for the release servo, which only needs power for a short period of time. At the time of designing the PCB, the exact MOSFETs to be used were unknown; it was only known that plenty were available on-hand and that they used a TO220BV package. When the input from the Flight Controller is low, the MOSFET does not allow current to flow from the GND pin to common ground. When the input is high, current is able to flow freely. This circuit effectively behaves as a solid-state switch.

4.6.11. GSM Module Ports

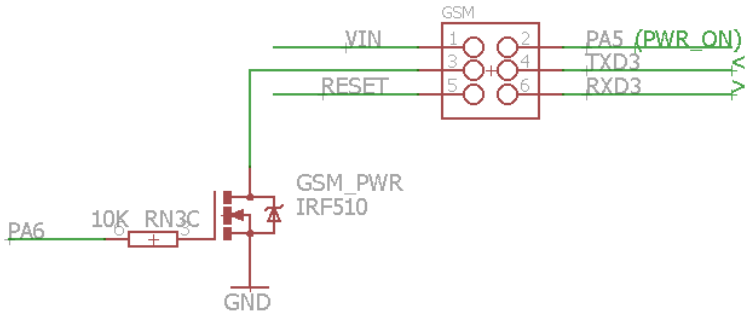

Figure 19: EAGLE Schematic View of GSM Module Headers and Circuit

The GSM Module is used after landing to send periodic SMS messages with location information from the GPS. Six ports were left open for the GSM Module: VIN, GND, RST, PWR_ON, TX3, and RX3. VIN and GND supply power to the module directly from the batteries. RST is a reset signal port that will allow the module to be reset at the same time as the microcontrollers. PWR_ON is used to send a power on or power off command to the module. TX3 (connected to TXD3, or PJ1) and RX3 (connected to RXD3, or PJ0) are used for communicating between the GSM Module and the Navigation Computer. Similar to the servos, a MOSFET was placed on the power line to the GSM Module in order to cut power to the module's linear regulator.

4.6.12. Radio Module Ports

At the time of designing this PCB, it was unclear how the Radio Module would operate. What was known was that it would require power directly from the batteries, and a port for data to be sent over. The ports left open for the Radio Module were VIN and GND for battery power and TX0 (connected to TXD0, or PE1) for data. This was done with the knowledge that if the Radio Module required more or different I/O ports, the auxiliary I/O ports could be used.

4.6.13. Auxiliary I/O and Analog Ports

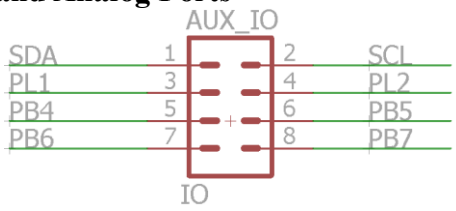

Figure 20: EAGLE Schematic View of Auxiliary I/O Headers

In order to ensure the Flight Controller would be able to use any additional hardware, a number of auxiliary ports were left open on the PCB. The Navigation Computer was given eight auxiliary I/O ports: SCL, SDA, PB4, PB5, PB6, PB7, PL1, and PL2. SCL and SDA allow for additional I^2C devices to be added. PB4, PB5, PB6, and PB7 are PWM and interrupt capable ports. PL1 is a general I/O which also can be used as an In Capture Pin, and PL2 is a general I/O which can be used as a timer output.

Four ADC ports from the Navigation Computer were added to the PCB. ADC0 (PF0) through ADC3 (PF3) made available along with ports for VCC and GND. These allow analog sensors and hardware to be read by the 10-bit ADCs of the ATmega2560V. [17]

The Autopilot Computer was given six auxiliary I/O ports: SDA, SCL, PB2, PD7, XTAL1, and XTAL2. SDA and SCL were added to allow for additional I^2C devices. PB2 is a general I/O which is PWM capable, and PD7 is simply a general I/O. XTAL1 and XTAL 2 are used for attaching an external oscillator to the microcontroller. These were added in case the Navigation Computer were for some reason unable to send a clock signal to the Autopilot Computer.

Two ADC ports from the Autopilot Computer were also made available. Ports for ADC0 and ADC1 can be seen above along with ports for VCC and GND. Just as with the Navigation Computer, these allow for analog sensors and hardware to be read by the 10-bit ADCs of the ATmega328P. [16]

In addition to all the auxiliary I/O discussed above, plenty of ports were made available for power. Three ports were added for regulated 3.3V power and three were added for unregulated battery power.

4.6.14. Mounting Holes

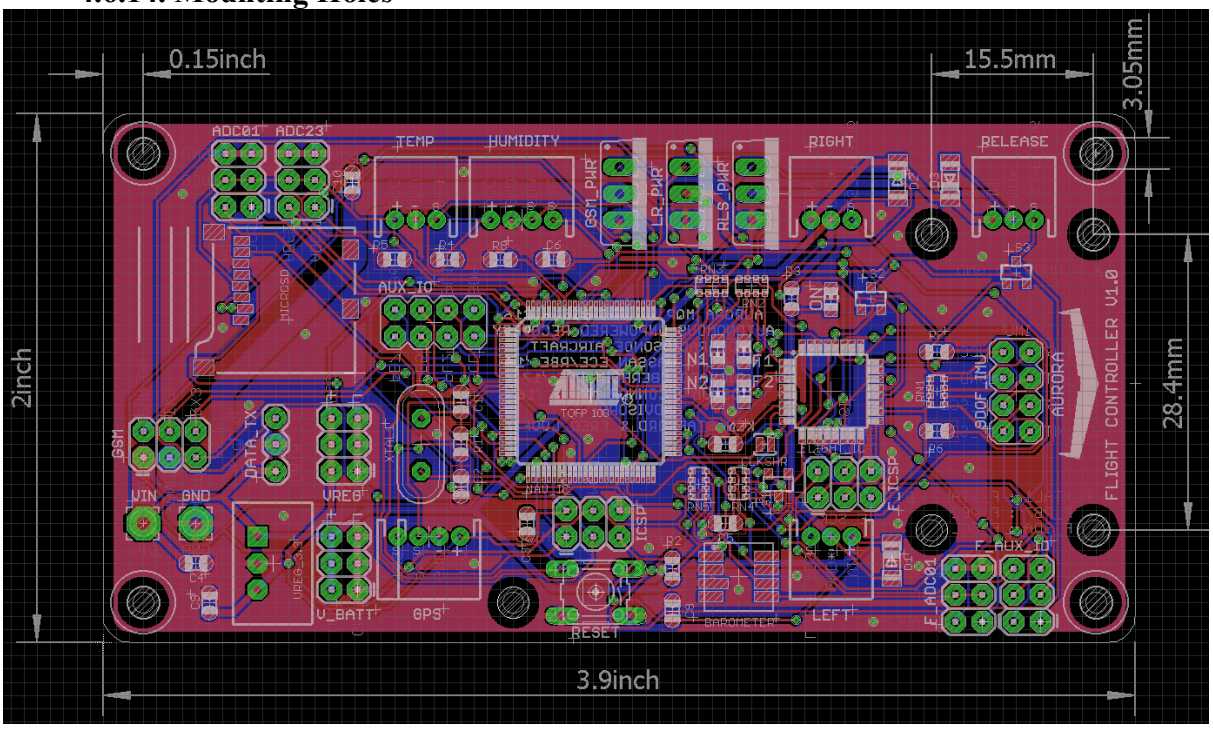

Figure 21: EAGLE Board View of Flight Controller with PCB Dimensions

Mounting holes for this PCB needed to be added for two reasons: to allow the other modules to be attached and to securely attach the Flight Controller to the airframe. The mounting holes are sized for M3 hardware and were placed at all four corners of the PCB. An additional mounting hole for the GSM Module was place on the left side of the PCB near the GPS port, giving the GSM Module three points of contact with the Flight Controller.

4.6.15. Routing Traces

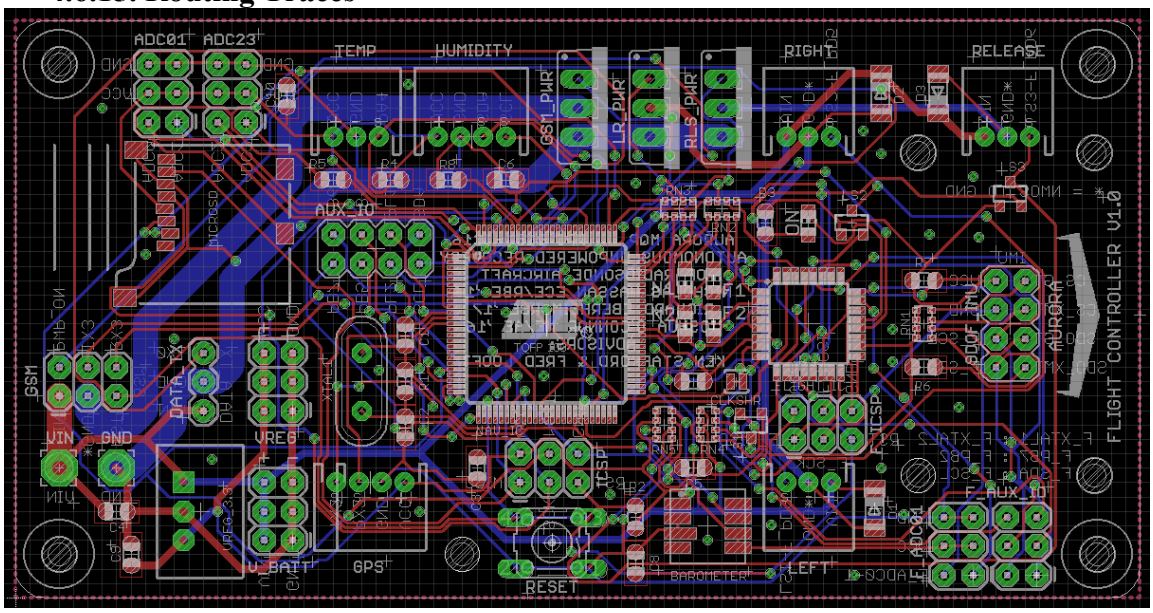

Figure 22: EAGLE Board View of Flight Controller PCB without Ground-Fill

Routing traces for this PCB was done within OSH Park's constraints. OSH Park PCBs use one ounce copper sides, and they're 2 layer boards have minimum specifications of 6 mil traces with 6 mil spacing and 13 mil drills with 7 mil annular rings. [11] In accordance with this, it was decided that all traces would be at least 8 mil wide and all vias would have a 13 mil drill. This would allow 377mA of current through the traces and the 1.837A through the vias.

For this PCB, not all traces were to be the same width, so auto-routing couldn't be used for everything. Doing some traces manually also allowed for noise-sensitive lines to be placed properly. The first traces routed were for the crystal oscillator and the clock-output line. These traces were made as short and direct as possible. Next, 24 mil traces were added as the power lines to the servos and the two power ports, allowing them to draw up to 835mA of continuous current. The traces for GSM power were added next. The GSM Module is able to draw up to 2A of current for short periods. For this purpose, 86 mil traces were used which can handle 2.107A of continuous current. These trace and via sizes were calculated using an online trace width calculator and via diameter calculator. [45] The remainder of the traces were auto-routed and ground-fill was added to the PCB.

4.6.16. Silk Screen

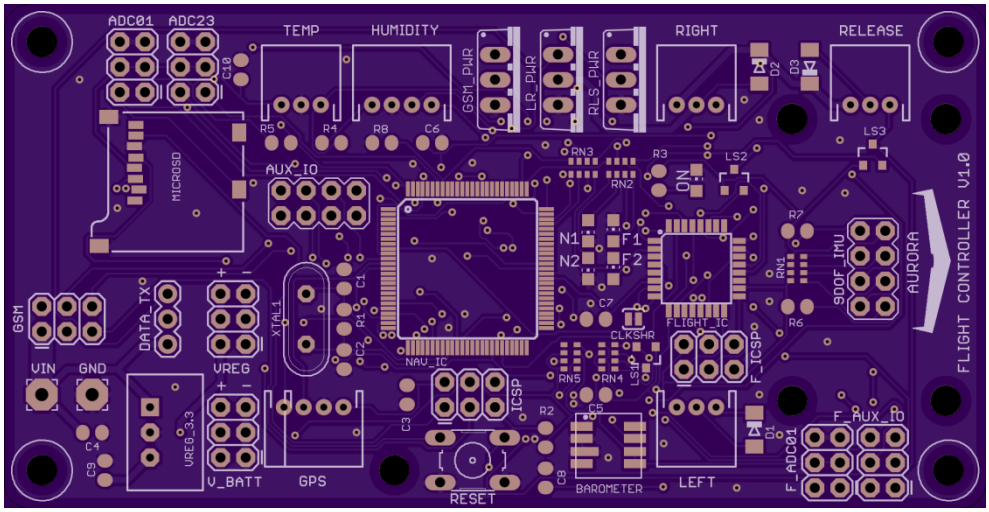

Figure 23: Front Side of OSH Park Render of Flight Controller

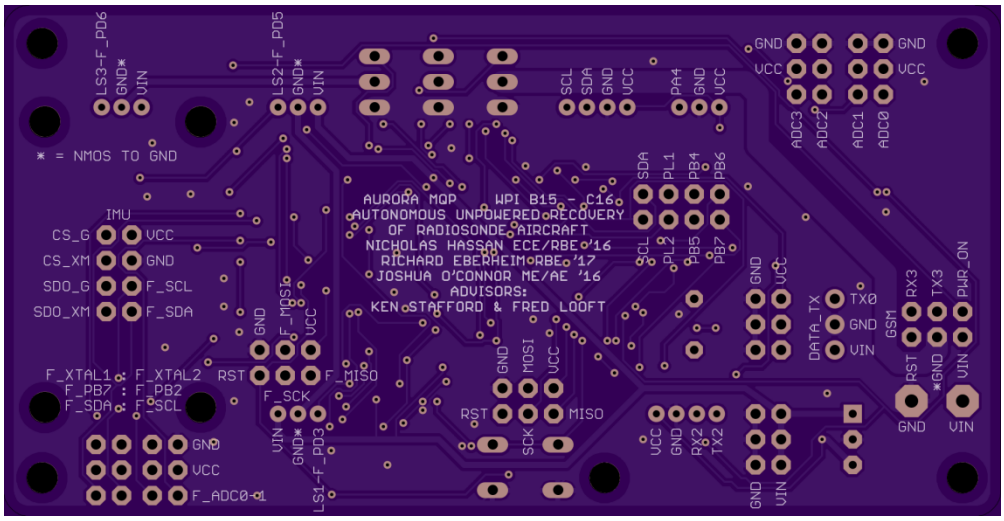

Figure 24: Back Side of OSH Park Render of Flight Controller

A proper silk screen can be incredibly helpful in assembling and using a PCB. On the Flight Controller PCB's top layer, the silk screen includes labels for every component, group of ports, and device connector. The top layer also includes a label with the board name and version number, as well as an image of a flying wing glider. The bottom layer has labels for every individual ports, along with project information. This includes the project name, school, terms, group members, and advisors.

4.7. Designing the GSM Module PCB

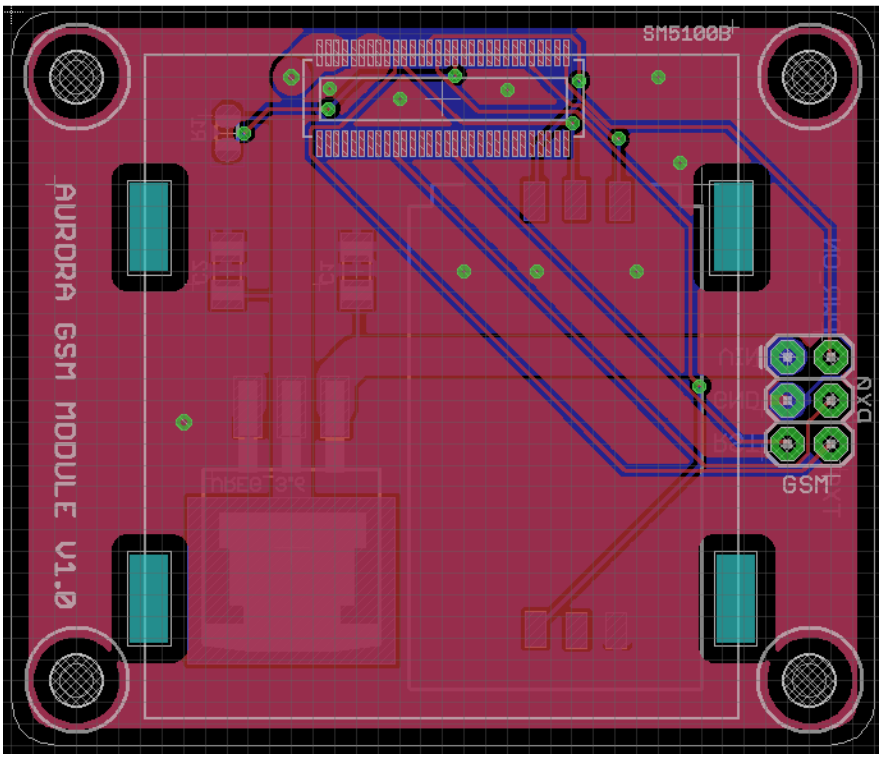

Figure 25: EAGLE Board View of GSM Module

4.7.1. GSM Ports

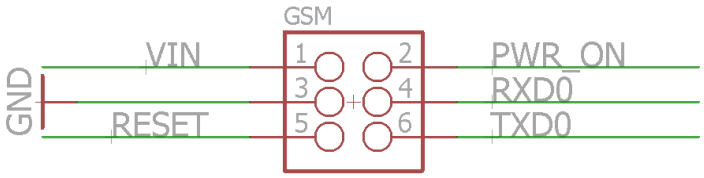

Figure 26: EAGLE Schematic View of GSM Headers

The GSM Module has six ports as seen above. Battery power is taken into VIN and is sent it directly to the linear regulator. RST goes to the reset pin of the GSM unit (pin 34), TX0 goes to TXD0 of the GSM unit (pin 19), RX0 goes to RXD0 (pin 20), and PWR_ON is connected to POWER_ON (pin 59). Communication with the Flight Controller is done with TX0 and RX0 using UART. The GSM Module can optionally be powered on and off using the PWR_ON input. The EAGLE layout for the SM5100B GSM Unit was taken from a SparkFun GSM Shield made for Arduino, and this layout includes rectangular holes in the PCB for the GSM unit's metal tabs.

4.7.2. SIM Card

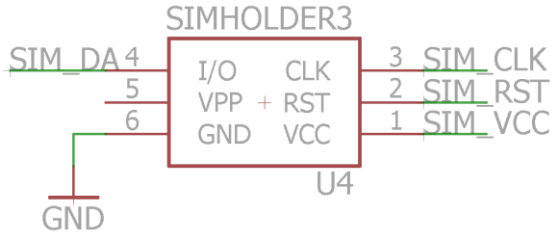

Figure 27: EAGLE Schematic View of SIM Card Holder

The SIM card on the GSM Module is what allows the cellular network to identify the device and enable operation. The six ports of the SD card reader are connected to their appropriate matching port on the GSM unit. The EAGLE part file for this SIM card holder was taken from SparkFun. [46]

4.7.3. Voltage Regulator

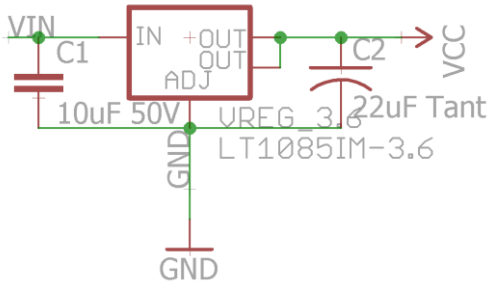

Figure 28: EAGLE Schematic View of GSM Module Voltage Regulator

The LT1085IM-3.6 has three pins: V_{IN}, V_{OUT}, and GND. V_{IN} takes an input of 4.75V to 15V, GND is connected to common ground, and V_{OUT} outputs 3.6V. Two additional capacitors are required to properly setup this linear regulator, as seen in the image above. A 10uF capacitor bridges V_{IN} to GND for noise reduction, and a 22uF tantalum capacitor bridges V_{OUT} to GND for stability. [36]

4.7.4. Mounting Holes

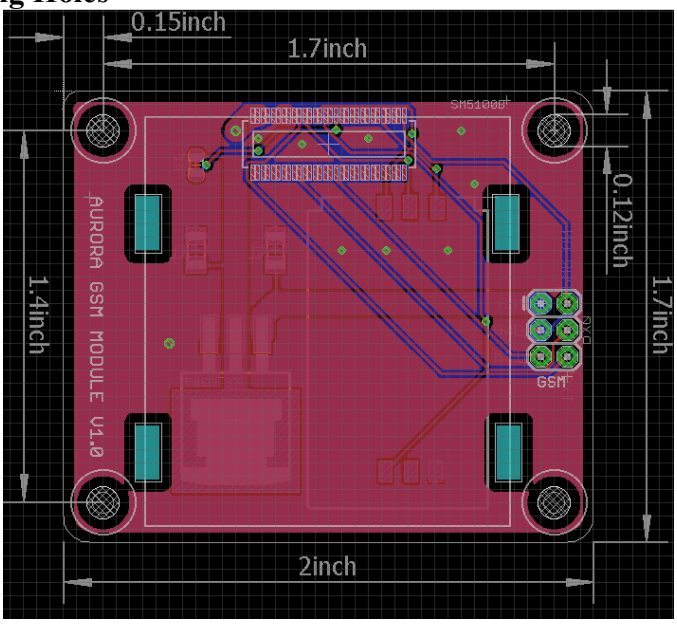

Figure 29: EAGLE Board View of GSM Module with PCB Dimensions

In order to mount the GSM Module to the Flight Controller (or anywhere else it may be situated), four mounting holes were added to the PCB, one in each corner. Just as with the Flight Controller, these holes were sized to fit M3 hardware.

4.7.5. Routing Traces

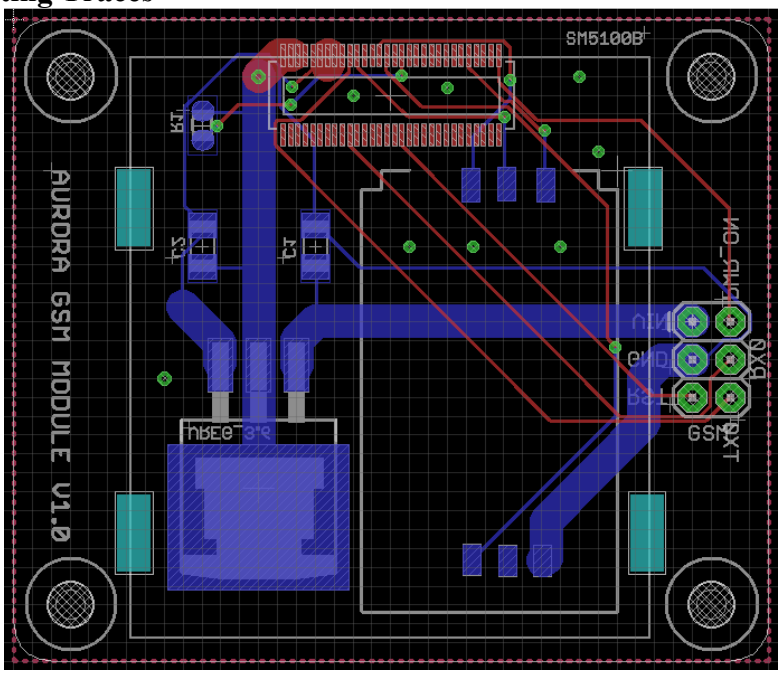

Figure 30: EAGLE Board View of GSM Module without Ground-Fill

In the interest of saving space and reducing the cost of the PCB, the GSM unit was placed on the top of the PCB while all the other components were placed on the bottom. Keeping the PCB small also allows it to be mounted without interfering with the MOSFETs on the Flight Controller. The first traces added were for the linear regulator. These traces were given a width

of 86 mil, allowing them to handle up to 2.107A of current. The vias for this trace were given a width of 20 mil, allowing them to handle up to 2.563A of current. The remaining traces were auto-routed with widths of 8 mil and via drill widths of 13 mil. Just as with the Flight Controller, ground-fill was added to the remainder of the PCB.

4.7.6. Silk Screen

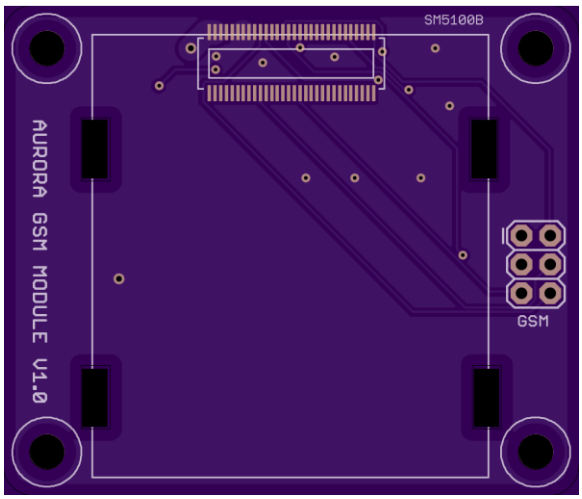

Figure 31: Front Side of OSH Park Render of GSM Module

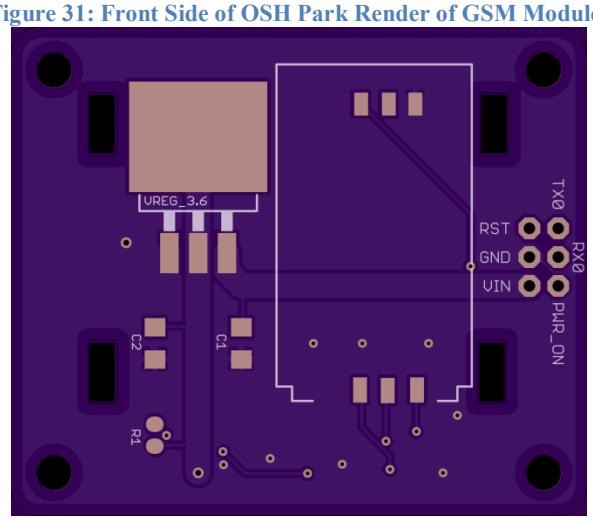

Figure 32: Back Side of OSH Park Render of GSM Module

As with the Flight Controller, the silk screen for the GSM Module includes labels for all components and ports, as well as a label with the board name and version number.

4.8. Designing the Radio Module PCB

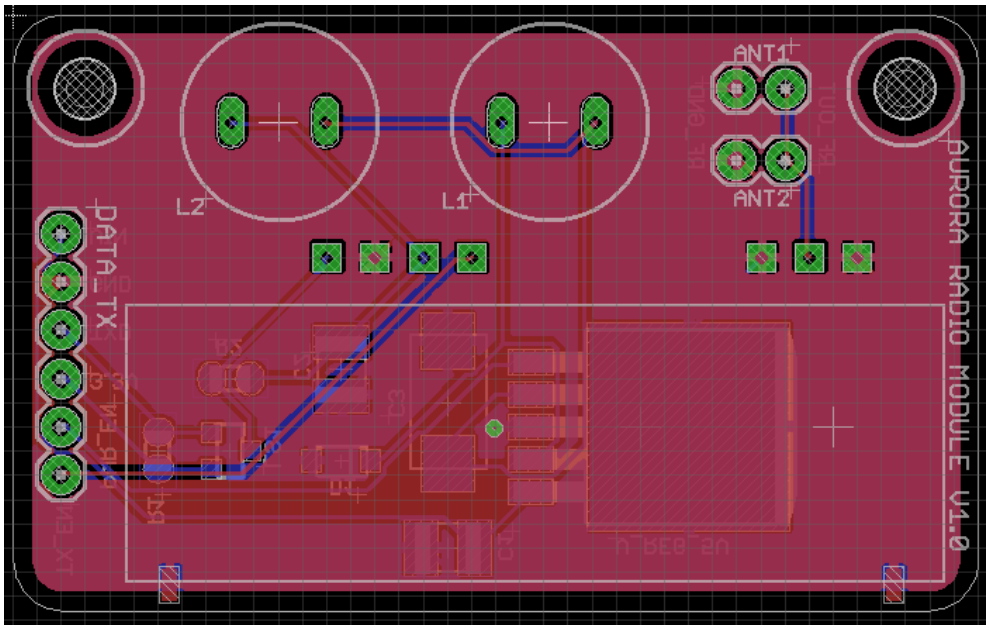

Figure 33: EAGLE Board View of Radio Module

4.8.1. Radio Ports

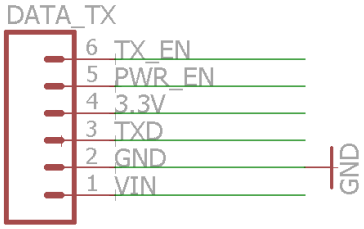

Figure 34: EAGLE Schematic View of Radio Module Headers

The Radio Module has six ports: VIN, GND, TXD, 3.3V, PWR_EN and TX_EN. Battery power is fed into VIN for the voltage regulator, TXD is the transmitter's data input from the Flight Controller, 3.3V is used for logic level shifting, PWR_EN is the voltage regulator's enable port, and TX_EN is the transmitter's enable port.

4.8.2. Radio Transmitter and Logic Level Converter

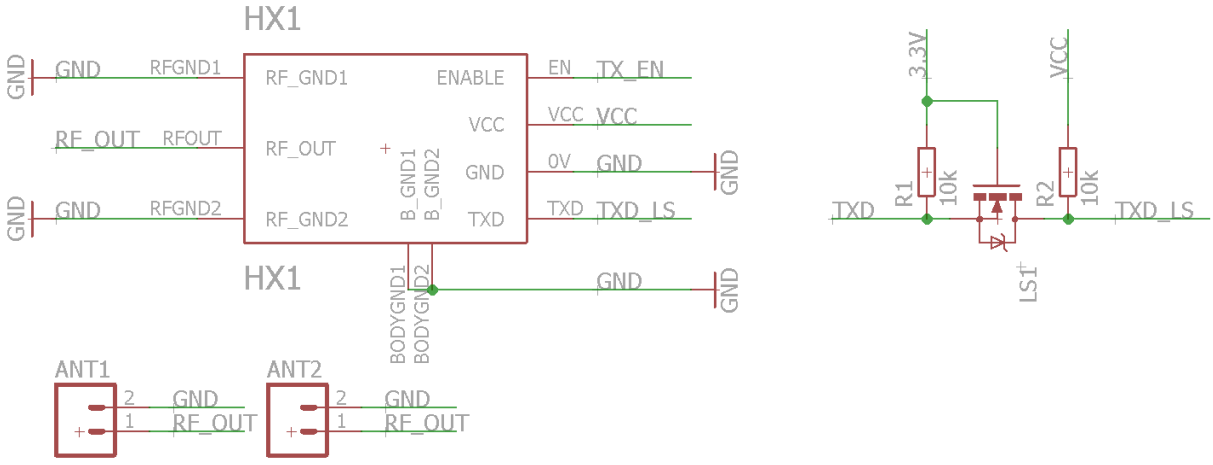

Figure 35: EAGLE Schematic View of HX1 Transmitter Circuit

The Radiometrix HX1 transmitter is powered by the 5V regulator. The signal to be transmitted is received on TXD. Enabling the transmitter is done by pulling ENABLE high. The EAGLE part file for this transmitter was taken from the Trackuino EAGLE file listed on their GitHub. [47] RF_OUT and RF_GND were each given two ports for an antenna to be securely attached. Since this transmitter takes in 5V logic, a logic level converter was added. Just as with the servos, a small circuit with an N-Channel MOSFET takes in 3.3V logic and outputs 5V logic to the transmitter.

4.8.3. Voltage Regulator

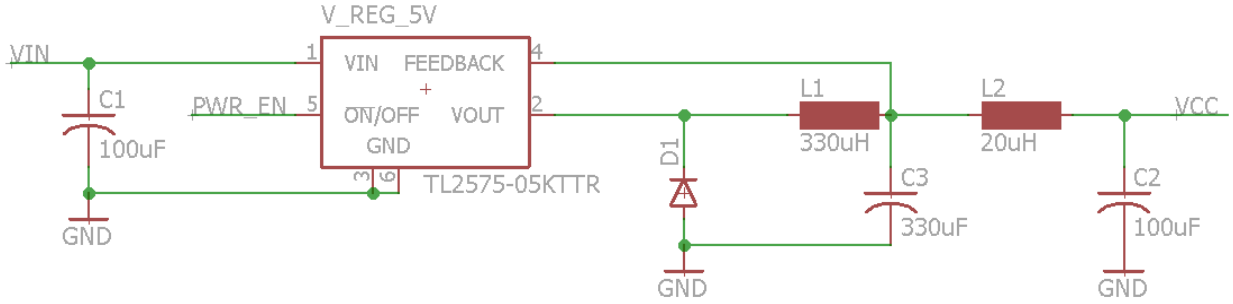

Figure 36: EAGLE Schematic View of Radio Module Voltage Regulator

The TL2575-05I switching voltage regulator has five pins: V_{IN}, V_{OUT}, GND, FEEDBACK, and ON/OFF. Battery voltage is received on V_{IN} and the regulated voltage is accessed through V_{OUT}. FEEDBACK is the voltage feedback pin, and ON/OFF is the manual shutdown pin. A significant amount of external hardware was needed to properly setup this regulator. The circuit shown above was taken from page 13 of this regulator's datasheet. This circuit include multiple LC filters in order to filter as much noise as possible. [39]

4.8.4. Mounting Holes

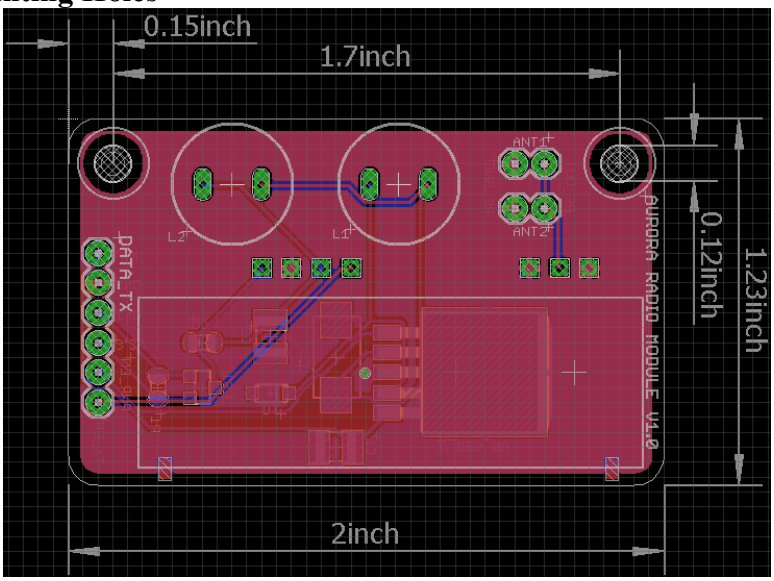

Figure 37: EAGLE Board View of Radio Module with PCB Dimensions

In order to mount the Radio Module to the Flight Controller or GSM Module (or wherever it best fits), two mounting holes were added to this PCB. As with the other two boards, these mounting holes were designed to work with M3 hardware.

4.8.5. Routing Traces

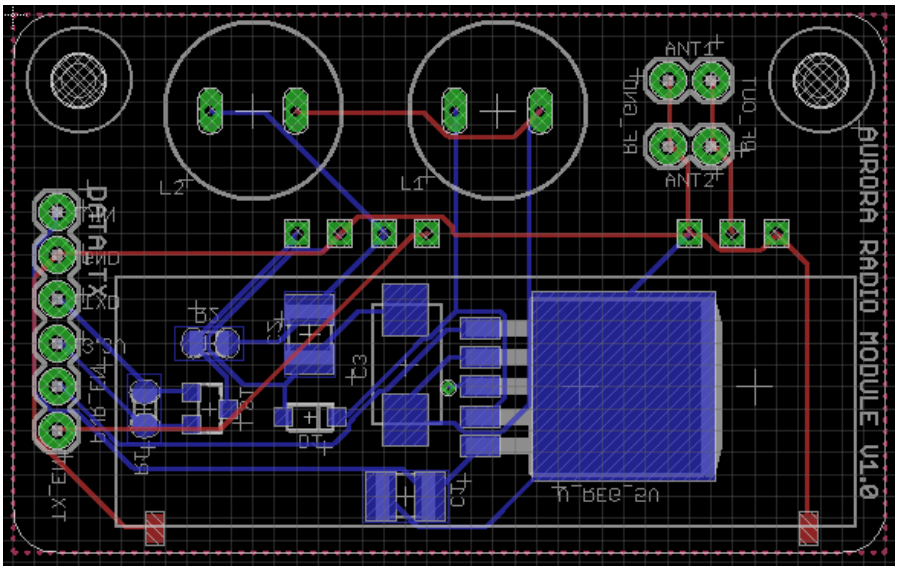

Figure 38: Eagle Board View of Radio Module without Ground-Fill

In the interest of saving space and PCB cost, the HX1 transmitter was mounted on the opposite side from the other components. The HX1 transmitter is the device with the highest current consumption. It draws up to 140mA during transmission, meaning the standard 8 mil traces and 13 mil vias can handle everything on this PCB. The traces were auto-routed and ground fill was added.

4.8.6. Silk Screen

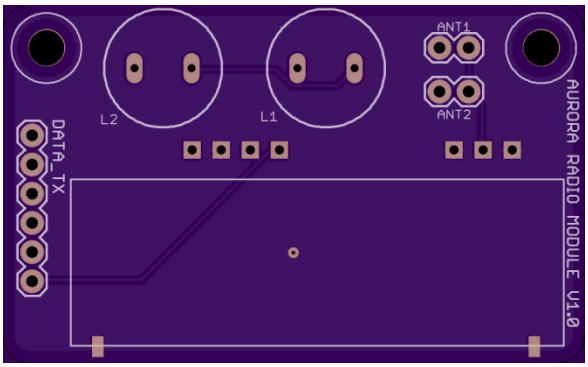

Figure 39: Front Side of OSH Park Render of Radio Module

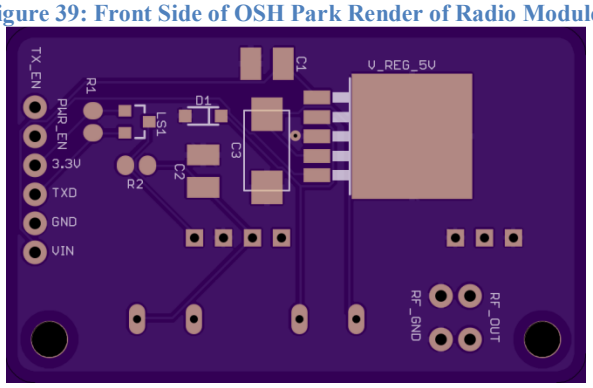

Figure 40: Back Side of OSH Park Render of Radio Module

As with the other two PCBs, the silk screen for the Radio Module contains all of the component and port names, as well as the board's name and version number.

4.9. Sourcing Components

Sourcing components to create this Flight Controller was a challenge in its own right. Most of the more general components (such as capacitors, resistors, LEDs, etc.) were ordered from Digikey. Digikey has an excellent tool for searching for components by specifications, which was incredibly helpful. The majority of sensors and specialized components (such as the GSM unit) were ordered from SparkFun. All other components were sourced separately out of necessity. Other companies used include Mouser, Adafruit, Linear Technologies, Lemos International, and Amazon. As previously stated, the PCBs themselves were ordered from and manufactured by OSH Park. The cost of individual components is shown in the bill of materials in Appendix A.

4.10. Weighing the Components and Soldering the PCBs

4.10.1. Weighing the Components

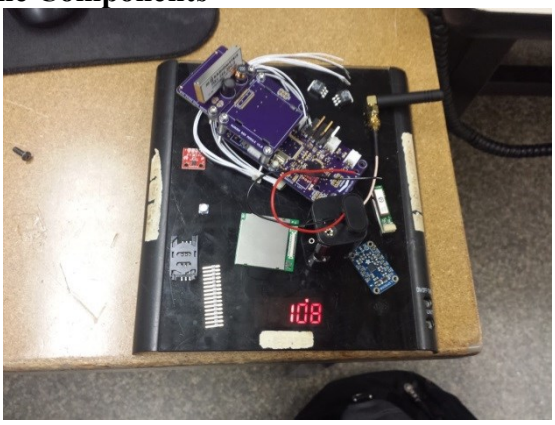

Figure 41: Weighing the Flight Controller Components

In order to verify the mass of the components would not cause the glider to exceed its specified weight limit, they were weighed as shown above. Placed on the scale, they came in at 108 grams. This excludes the weight of the servos and batteries.

4.10.2. Soldering the PCBs

Several methods of SMD soldering were researched prior to attempting to solder the boards. One option was to use a solder reflow oven after squeegeeing solder paste onto the PCB with a template. Using a reflow oven would have been ideal, but no such equipment exists on campus. Another option was to apply solder paste to the copper pads of the PCBs. Videos on the internet showed this making for a clean solder joint. Experiments with this method, however, found the result to be messy, with a significant number of loose solder balls left on the pads. The final method examined is called drag soldering. This method is used on SMD components with small, close pads (such as those on the ATmega2560V and ATmega328P). It involves accumulating a small amount of solder on the tip of a soldering iron and dragging the solder

across all the pads on one side of the chip. The solder wicks to the pads and generally requires little cleanup.

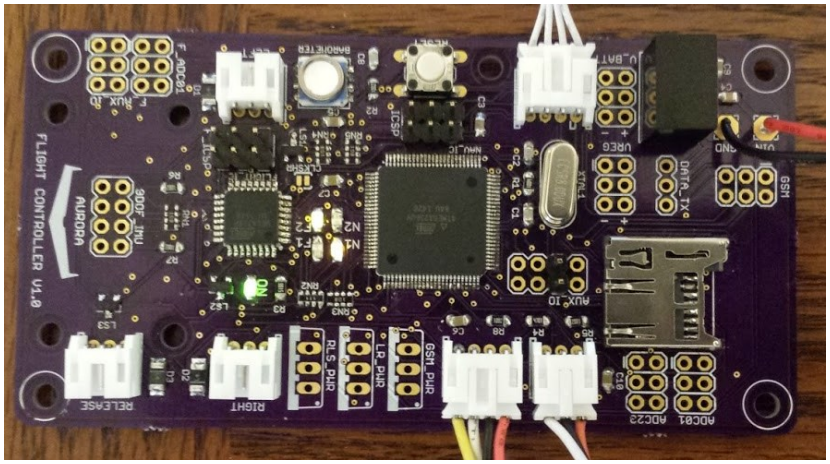

Figure 42: A soldered Flight Controller

No one in this project group had any experience soldering SMD components, and it was certainly daunting to have to solder something so small when there are so few chances to get it right. Luckily for us, a friend of ours is has experience SMD soldering. Keshuai Xu, a fellow WPI student, graciously offered his assistance. Using a knife-tip soldering iron, plenty of flux, some solder wick, and a microscope, he was able to solder our PCBs. He used the drag solder method on the microcontroller chips and used the standard soldering method for the other SMD components.

4.11. Low Level Coding and Debugging

4.11.1. Setting the Microcontroller Fuses

ATmega microcontrollers have what are called “Fuse Settings”. These are chip settings that determine a number of factors about how it functions. These settings include clock speed and source, brown-out detection level, enabling serial program downloading, enabling the clock output pin, and a number of others. There are three fuse bytes: Low, High, and Extended. Each byte controls different settings, each setting based on the hex value last written to the fuse. Fuse settings can be changed using the Atmel STK500 AVR Programmer with Atmel Studio. [48]

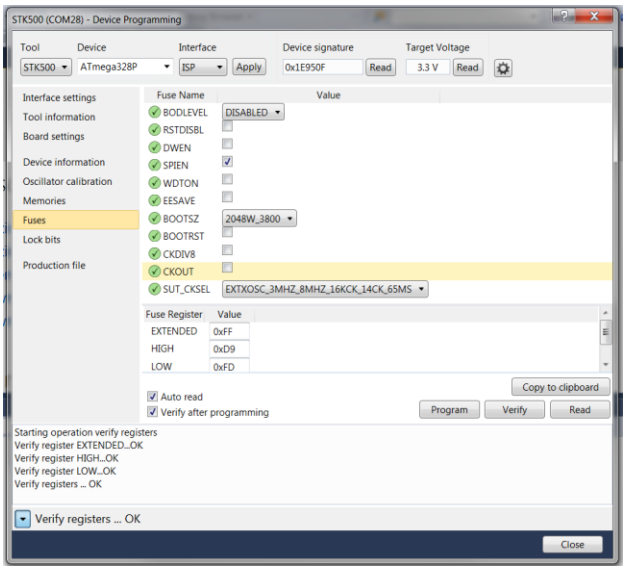

Figure 43: Fuse Settings in Atmel Studio

The fuse settings used for the ATmega328P are 0xFD for LOW, 0xD for HIGH, and 0xFF for EXTENDED. These fuse settings disable brown out detection, enable SPI programming, and set the clock to use an external source with a frequency between 3MHz and 8MHz.

The ATmega2560V fuse settings are 0xBD for LOW, 0x99 for HIGH, and 0xFD for EXTENDED. This sets the brown out detection to 2.7V, enables SPI programming, turns on the clock output pin to the ATmega328P, and sets the clock to use an external oscillator with a frequency from 3MHz to 8MHz.

A problem was encountered on the first Flight Controller board when attempting to program fuses. The ATmega2560V was not responding to the STK500. It could not find a device signature or write fuses. After a day of debugging, it was determined that the microcontroller's fuses were set incorrectly from the factory. The default factory fuses set SPI programming to be enabled, allowing the use of ICSP headers to program the device through SPI. If this fuse setting were somehow left off after manufacturing, the chip would be rendered useless. The second flight controller had no issues with fuse programming.

4.11.2. Custom Arduino Boards File

In order for the Flight Controller to be programmed with Arduino IDE, a custom boards.txt file was created. The boards file contains information on all the Arduino devices it is able to program. This includes device name, upload protocol, maximum program size, clock speed, fuse settings, etc. Two board entries were added to this file: One for the Navigation Computer and one for the Autopilot Computer.

```
##############################################################

auroraFlight.name=AURORA MQP Flight Computer

auroraFlight.upload.tool=avrdude
auroraFlight.upload.protocol=arduino
auroraFlight.upload.maximum_size=32256
auroraFlight.upload.maximum_data_size=2048
auroraFlight.upload.speed=57600

auroraFlight.bootloader.tool=avrdude
auroraFlight.bootloader.low_fuses=0xFD
auroraFlight.bootloader.high_fuses=0xD9
auroraFlight.bootloader.extended_fuses=0xFF
auroraFlight.bootloader.unlock_bits=0x3F
auroraFlight.bootloader.lock_bits=0x0F
auroraFlight.bootloader.file=optiboot/optiboot_atmega328.hex

auroraFlight.build.mcu=atmega328p
auroraFlight.build.f_cpu=8000000L
auroraFlight.build.core=arduino
auroraFlight.build.variant=standard
```

Figure 44: Boards.txt Entry for AURORA Autopilot Computer

```
##############################################################

auroraNav.name=AURORA MQP Nav Computer

auroraNav.upload.tool=avrdude
auroraNav.upload.protocol=wiring
auroraNav.upload.maximum_size=253952
auroraNav.upload.maximum_data_size=8192
auroraNav.upload.speed=57600

auroraNav.bootloader.tool=avrdude
auroraNav.bootloader.low_fuses=0xBD
auroraNav.bootloader.high_fuses=0x99
auroraNav.bootloader.extended_fuses=0xFD
auroraNav.bootloader.unlock_bits=0x3F
auroraNav.bootloader.lock_bits=0x0F
auroraNav.bootloader.file=stk500v2/stk500boot_v2_mega2560.hex

auroraNav.build.mcu=atmega2560
auroraNav.build.f_cpu=8000000L
auroraNav.build.core=arduino
auroraNav.build.variant=mega

auroraNav.build.board=AVR_MEGA2560
```

Figure 45: Boards.txt Entry of AURORA Navigation Computer

This resulted in the two microcontrollers being added to Arduino IDE with the names "AURORA MQP Nav Computer" and "AURORA MQP Flight Computer" (Flight Computer was later renamed to Autopilot Computer for clarity).

4.11.3. Making the LEDs Blink

In order to make the LEDs work, a simple function was written for each microcontroller called *setLED()*. This function takes in an Arduino pin number (which were defined so that *LED1* and *LED2* could be used instead) and a boolean of HIGH or LOW to set the LED state. In order to test the working order of the microcontrollers, a simple piece of code was written for each to have the LEDs blink back and forth once every second. This code was first uploaded before the proper fuse settings were programmed, having the microcontrollers use their internal oscillators. It was very clear that the microcontrollers were not operating on the same frequency (the internal oscillators are quite inaccurate). With the proper fuse settings, the LEDs on each microcontroller blinked perfectly in sync with one another.

4.11.4. Software Serial for Debugging

Proper debugging of the Flight Controller would require a serial output which can be read by a computer. Since all UART lines of each microcontroller are in use, a library called Software Serial was used. This library can take any two pins on an Arduino and make them behave as if they are UART pins. In order to set this up, the library *SoftwareSerial.h* was included, and the following line of code was placed before the setup function:

```
SoftwareSerial debugSerial(12, 13); //RX, TX
```

A 90-degree header pin was soldered on PB6 (Arduino Mega pin 12) and PB7 (Arduino Mega pin 13) of the Navigation Computer's auxiliary I/O ports. In order to use this software serial port, the output would have to be transferred to an Arduino, which would relay it to the host computer to be displayed. A simple piece of code was written to do this, completing this setup and allowing for proper debugging.

4.11.5. Coding and Testing the GPS

The GP-735 GPS, when powered on, returns what are called NMEA output messages. These messages contain information on date, time, latitude, longitude, course, speed, number of satellites, etc. The first step in testing the GPS was to attach it to an Arduino Mega and read the raw data output. Being powered by 5V from the Arduino, the GPS had no issues turning on and transmitting proper position, date, and time information.

A library called TinyGPS++ is used to make the task of parsing data extremely simple. During the main loop, a function called *smartDelay()* is called. This function runs for a specified number of milliseconds, and in that time uses *gps.encode()* to capture data from the GPS serial stream.

```
static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (Serial2.available())
      gps.encode(Serial2.read());
  } while (millis() - start < ms);
}
```

This gathers data for the TinyGPS++ library which then parses it. The data is accessed through commands such as *gps.location.lat()*. A function *updateGPSvalues()* is then called which takes all these values and stores them in global variables. No major issues were encountered setting up the GPS. [49]

4.11.6. Coding and Testing the Weather Sensors

The first sensor tested was the DS18B20 temperature sensor. This sensor was set up with the help of a Bildr tutorial linked to from the sensor's SparkFun page. The library *OneWire.h* is included to make uses of this sensor. The value *TEMP* is defined as the temperature sensors pin, which is Arduino Mega pin 26. The function *getTemp()* was then created to return a value in degrees Celsius from the sensor. The code for this function was taken from the Bildr tutorial, and worked without any need for modification. [50]

The next sensor tested was the HTU21D humidity sensor. A SparkFun library called *SparkFunHTU21D.h* is included to use this sensor. A function called *getHumidity()* was written, and returns a floating point relative humidity value after compensating for temperature. The humidity sensor worked perfectly and required no debugging. [51]

The MS580301BA01-00 barometer was tested next. Just as with the humidity sensor, a SparkFun library exists for this sensor. The library *SparkFun_MS5803_I2C.h* is included to make use of this sensor. [52] A simple function called *getBarPressure* was written to return the

barometric pressure with a specified ADC accuracy of 4096. The value returned was found to not be in mBar, but actually in tenths of mBar, so it is multiplied by 10. Aside from that, this sensor required no debugging. A problem did occur, however, when one of the barometers was somehow damaged and started returning very odd values (on the order of 0.6 atmospheres at ground level). The sensitive gel coating of the sensor is believed to have somehow been pierced. The damaged sensor was desoldered and replaced.

Figure 46: Protective Cover for Barometer

In order to ensure this wouldn't happen again, a 3D printable protective cover was designed and printed. The cover sits snuggly on top of the sensor and has a very small hole to let in air.

4.11.7. Coding and Testing the IMU

The IMU was first tested by attaching it to an Arduino Mega. In order to use the LSM9DS0 IMU, a SparkFun library called *SFE_LSM9DS0.h* was included with the code for the Autopilot Computer. [53] An object was made for the sensor which puts it into I2C mode and a function called *startDOF()* was written to setup the IMU. A function was made to access data from each axis of each sensor. An example of is shown below:

```
double getAccelX(void)
{
  dof.readAccel();
  return dof.calcAccel(dof.ax);
}
```

This code worked without any issues at first, but a problem arose when powering off and on the board. The IMU started returning data showing a constant +/-1.5g in all axes of the accelerometer. A forum discussion on adafruit.com revealed that the problem had to do with parasitic capacitance left in the breakout board after shutdown. [54] Any voltage being applied to the accelerometer would cause it to not reset properly and enter this state of error. To fix this problem, a 10kΩ resistor was placed across 3.3V and GND in order to drain any capacitance left over after shutdown. This solved the problem. The IMU started working again and would not go into its state of error upon being reset. However, it was found that power cycling the IMU very quickly would still cause it to enter this state. No way was found to remedy this other than to be careful when plugging in the Flight Controller.

4.11.8. Coding and Testing the Servos

In order to make servos function in Arduino, a library called *servo.h* is included. To control the servos, a simple function was written called *setServo()*, which takes in the servo name and desired position. This code worked without issue, and the logic level converters worked flawlessly.

To set up the MOSFETs for controlling power flow, a function called *setMOSFET()* was written. This takes in the MOSFET pin and desired state. A problem arose when attempting to

use the MOSFETs available on-hand to control power flow to the servos. The MOSFETs were IRF520s, which cannot operate with a gate-source voltage of 3.3V applied to them, as the data sheet shows they need at least 4V. To fix this, the FQP30N06L is used instead of the IRF520. These have a minimum gate-source voltage of 2V, and are able to supply more than 10A when given 3.3V.

4.11.9. Debugging MicroSD

The MicroSD card reader on the Flight Controller uses an SPI interface. The first test performed was with the Arduino SD example program called *CardInfo*. This program returns information on the SD card including card type, formatting, size, and a list of all files on the card. Initial tests with this code showed no response from the SD card. With the card inserted and the proper Chip Select pin set, it would return an error saying "Initialization failed". Using a multimeter, continuity was verified to all the pins of the MicroSD card reader, so this was not the source of the problem. Using an oscilloscope, the output from MOSI and SCK were verified to be clean, but nothing was being received from MISO. The Chip Select pin was also verified to be operational.

In order to verify that the source of the problem was not the software, a MicroSD card reader breakout was connected to an Arduino Mega. The same code ran with this setup without any issues. The MicroSD card reader breakout was then connected to the Flight Controller via a logic level converter. It did not work in this setup, so at this point the source of the issue seemed to be the microcontroller.

The first Flight Controller was then given a replacement ATmega2560V (as mentioned earlier, the first one came with bad fuse settings, rendering the chip useless). The *CardInfo* program was loaded onto this Flight Controller and it worked perfectly. In the interest of finding the source of the problem on the other board, the MicroSD card reader breakout was connected to the newly working Flight Controller. The breakout did not work in this setup, invalidating the test that was done earlier on the first Flight Controller. As such, the problem is believed to be the Micro SD card reader itself.

4.11.10. Coding and Testing the GSM Module

In order to avoid damage to the GSM unit, the linear regulator's output was first probed before connecting the GSM unit. The linear regulator showed a clean output of 3.57V, so the GSM unit was attached. Testing was first done through an Arduino Mega via a logic level converter. With the AT&T SIM card installed and the GSM Module powered on, serial data started to stream to the computer. A series of string were received, ending with the string "+SIND: 8", meaning the module has been rejected from the network. Attempts to diagnose the problem were initially unsuccessful, as it was not responding to commands.

Attempts to communicate with the GSM Module were at first done with Arduino IDE serial and with Terminal Com Port Dev Tool, both of which resulted in no response. Using PuTTY, however, resulted in a response. Sending the generic "AT" command resulted in a return of "OK". The GSM Module was then given the command "AT+SBAND=7", forcing the module to use the North American 850MHz cellular band. Upon resetting, the board returned all the same strings, except "+SIND: 8" was replaced with "+SIND: 11", meaning the module is registered on the cellular network. To test the connection, it was given a command to call a cell phone, which worked without issue. [55]

The next step was to enable the ability to send text messages. A library had to be included called *SerialGSM.h* and an object called *cell* was made for it. This library, unfortunately, forces the use of Software Serial, though this didn't seem to be an issue. A

function called *sendGSMdata()* was then written to send SMS messages with information on the device's location. This worked without need for much debugging. The GSM Module was able to send GPS latitude and longitude.

4.11.11. Coding and Testing GSM Module Power Sequence

Once attached to the Flight Controller, an issue with the power MOSFET arose. The batteries running the Flight Controller at the time had a total voltage of about 5.3V. The linear regulator on the GSM Module requires a minimum of 4.75V to operate. With a 0.7V drop across the power MOSFET, this dropped the GSM Module's voltage input to 4.6V. The result was the GSM Module would power on and begin returning strings, but would reset itself whenever it attempted to draw power for communicating with the cellular network. To fix this, the MOSFET was removed and a jumper was left in its place. The GSM Module has a pin for powering on and off the GSM unit itself manually. This would mean the linear regulator is always powered on, but the losses would be minimal if almost no current is being drawn. With the GSM unit powered off, the GSM Module draws about 5mA of current. This loss is an acceptable trade off to significantly reduce the risk of the GSM Module not operating at the end of a flight.

Two functions were written to power on and off the GSM Module: *GSMstartup()* and *GSMshutdown()*. *GSMstartup()* begins with a command to write high to the power-on pin, activating the GSM unit. Normally it would be fine to wait a set amount of time and then continue with transmitting SMS messages. However, it was found that when powering on, the GSM Module would occasionally return "+SIND: 7", meaning it entered an emergency-call-only mode. [55] It would also occasionally fail to find the network several times before connecting. Due to these issues, it would be foolish to simply set a delay and hope it connected properly. In order to solve this, the serial streams coming from the GSM Module are analyzed until the proper strings have all been received. If it enters emergency-call-only mode, the module is power cycled.

The *GSMshutdown()* function writes low to the power-on line for three seconds, writes high to it for two seconds, and then writes low to it. On a cell phone, this power-on signal would be attached to the power button on the side of the phone. This function essentially presses and holds the "power button" until the "phone" turns off.

4.11.12. Coding and Testing the Radio Module

To begin coding the Radio Module, the source code for the Trackuino firmware was downloaded and examined. This source code contains a large number of files and includes many features which are not needed for this project. In order to modify this code for our purposes, all features were removed from it except for what is essential to transmit APRS data. What remained were the following files:

File(s)	Purpose
afsk.cpp, afsk_avr.cpp, & afsk_avr.h	Creates an AFSK signal from AX.25 input data and outputs it on a PWM capable pin to the radio.
aprs.cpp & aprs.h	Reads in sensor data and formats it for APRS transmission. Sends formatted data to AX.25 modem code.
ax25.cpp & ax25.h	AX.25 is a data link layer protocol designed for use by amateur radio operators. This code modifies the input APRS data for transmission and sends it to the AFSK code.
config.h	Contains various settings including call sign, APRS comment, modem configuration, and debugging.
pin_avr.cpp & pin.h	Contains code for *pin_write()* function to replace *digitalWrite()*. Necessary since *digitalWrite()* interrupts timer2, which is used for transmission.
radio.cpp, radio.h, radio_hx1.cpp, & radio_hx1.h	Contains code for powering on and off the HX1 transmitter.

Table 3: Descriptions of Files used for APRS Transmissions

As explained above, APRS data is sent via the following process: Data is read from the GPS and other sensors. This data is formatted into an APRS packet such as the one shown below:

KC1BQU-11>APRS,WIDE2-1:/021709zh3822.20N/07254.38EO073/014/A=015370/T=-2107/H=-1667/P=6730/V=218/S=34 AURORA

Item	Description
KC1BQU	User's callsign
-11	Signal source = balloon
>APRS	Destination callsign
WIDE2-1	Digipeating path
021709z	Zulu date/time (17:09 on the 2nd day of the month)
h3822.20N	Latitude
07254.38E	Longitude
O	Device is a balloon
073/014	Course: 073 degrees, Speed: 014 knots
A=015370	Altitude: 15,370 feet
T=-2107 … S=34	Various sensor readings
AURORA	APRS comment string

Table 4: Breakdown of APRS Packet Components

This formatted data is sent to the AX.25 modem code where it is further formatted and added into FIFOs for transmission. This information is then given to the AFSK code where it is sent to the HX1 transmitter via a PWM capable pin.

Before testing could be started, an antenna had to be made for the Radio Module. Following instructions on page 6 of the HX1 transmitter's datasheet, a quarter-wave whip antenna was created with a length of about 493mm. [56] About 5mm of this was trimmed off to account for the length of the trace leading to the antenna holes on the PCB. The antenna itself is

made of solid-core copper wire, and was attached securely by passing the wire though both antenna output holes.

Testing was first done on an Arduino Uno, as the Trackuino is designed to use this type of microcontroller board. The Radio Module was powered by an external battery pack for this test. TXD was connected to pin 3 and TX_EN was connected to pin 4. The APRS code itself was then modified to only output callsign information and a lengthy APRS comment. Using an amateur radio, the signal was made audible. What was heard sounded similar to a dial-up modem and lasted about half a second. This confirmed the Radio Module was in working order.

Since the Radio Module is required to work with an ATmega2560V (as opposed to the ATmega328P used by the Arduino Uno), the module was connected to an Arduino Mega. The PWM pin was changed from 3 to 10 and the TX_EN pin was changed from 4 to 11. Pin 10 was chosen to allow the use of the OC2B timer. The Trackuino firmware is already set up to support this timer with pin 11 on the Arduino Uno, so making this work simply involved redefining the pin numbers [47]. Once again, an amateur radio was used to verify that the Radio Module was working.

Sending actual data involved the creation of a number of functions. A function was written to format APRS data for each of the following: Date/time, latitude, longitude, course, speed, altitude, temperature, humidity, pressure, battery voltage, and seconds (APRS date/time format excludes this). A function called *transmitRadioData()* was then written which takes all formatted APRS data and sends it to a modified *aprs_send()* function. To avoid interrupting Timer2, all instances of *digitalWrite()* were replaced with *pin_write()*. The Radio Module was then attached to the Flight Controller with TXD attached to PB4 (pin 10 on Arduino Mega) and TX_EN attached to PB5 (pin 11 on Arduino Mega). The output of this setup was tested and verified via serial. The APRS radio data transmission takes about 1 second.

4.11.13. Adding an Airspeed Sensor

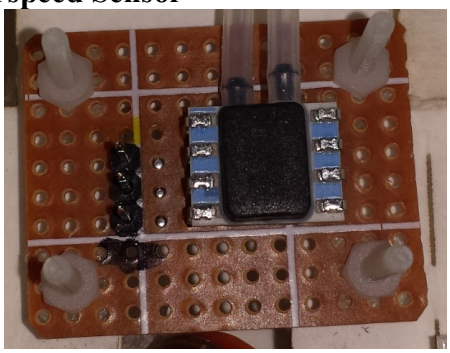

Figure 47: Airspeed Sensor on Perfboard

During glider testing, pitch control was found to not be sufficient for flying the aircraft. An airspeed reading was needed, and the sensor chosen for this task was the SSCDRRN025MDAA3 differential pressure sensor. This sensor was chosen for its exceptional temperature stability and wide range of measurement (about 4.5 to 90 m/s with a pitot tube). This is an analog 3.3V sensor with the following pins: VIN, VOUT, and GND. In order to use this sensor, a header was placed on the ADC0 port for the Autopilot Computer. VIN is given 3.3V, GND is connected to common ground, and VOUT is connected to ADC0. [57] Since the

sensor couldn't be mounted directly to the Flight Controller, it was soldered to a piece of perfboard which had mounting holes drilled in it.

4.11.14. Fixing the Power Connection

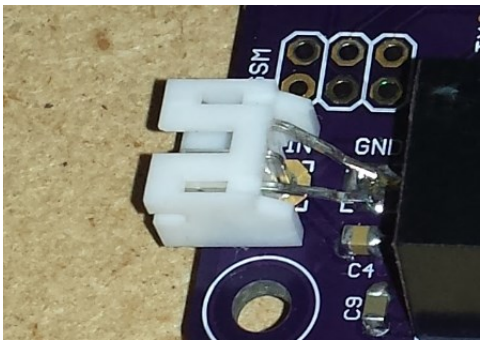

Figure 48: JST Connector Attached to Power Terminals

It was found that simply soldering the wires of the power connector directly to the PCB was not mechanically sound. They would break off after being moved back and forth for a short period of time. To solve this, a spare JST connector was soldered as shown in the image above.

4.11.15. Reset Button Replacement

An interesting problem arose when testing the glider. Upon hitting the ground, it appeared that the Flight Controller was resetting itself. The immediate suspicion was that the force of the impact was somehow causing the reset button to actuate itself. In order to ensure this wasn't the case, the button was removed and a jumper was put in its place. With this in place, a reset could only happen if a piece of metal bridged the two pins of the jumper, eliminating the button as a potential source for this problem.

4.12. Assembling the Modules and Attaching to Airframe

4.12.1. Assembling the Modules

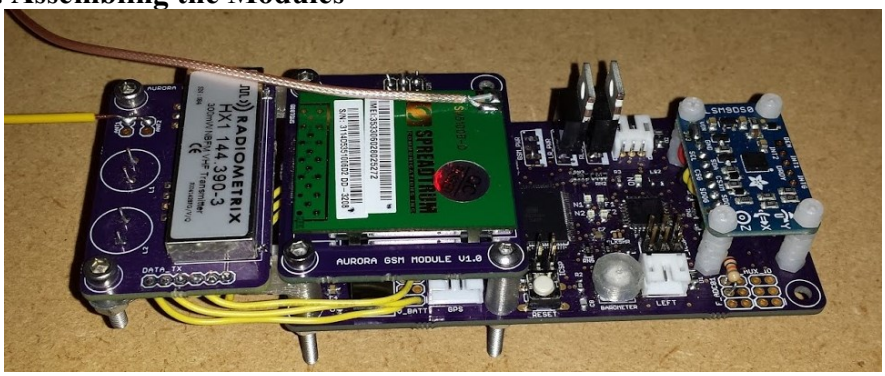

Figure 49: Fully Assembled Flight Controller with Modules

In order to properly assemble the Flight Controller and modules, several spacers were required. The spacers were designed for 3D printing using Autodesk Inventor. Originally, it was thought that the Radio Module would be mounted on top of the GSM Module. This was changed in accordance with the shape of the airframe. With the Radio Module mounted in front of the GSM Module, the spacers included arms to reach the mounting holes of the Radio Module. The IMU was mounted with #2 nylon nuts and bolts. Six nuts were used as spacers on each screw, with two lock-nutted on the underside of the board.

4.12.2. Attaching the Flight Controller to the Airframe

Attaching the Flight Controller to the airframe required several more 3D printed parts to be made. Small spacers were created to lift the Flight Controller about 5mm from the surface of the airframe, leaving plenty of room for the pins and nylon nuts on the underside of the board. Large washers were then made for the underside of the mounting surface. Zip ties were used to attach the board so that that it would not be damaged in the event of a crash, as the zip ties would simply shear before damaging the PCB (as they did on several occasions during testing). The airspeed sensor board was attached in a similar manner.

4.13. Breadboard Flight Controller for Initial Test Flights

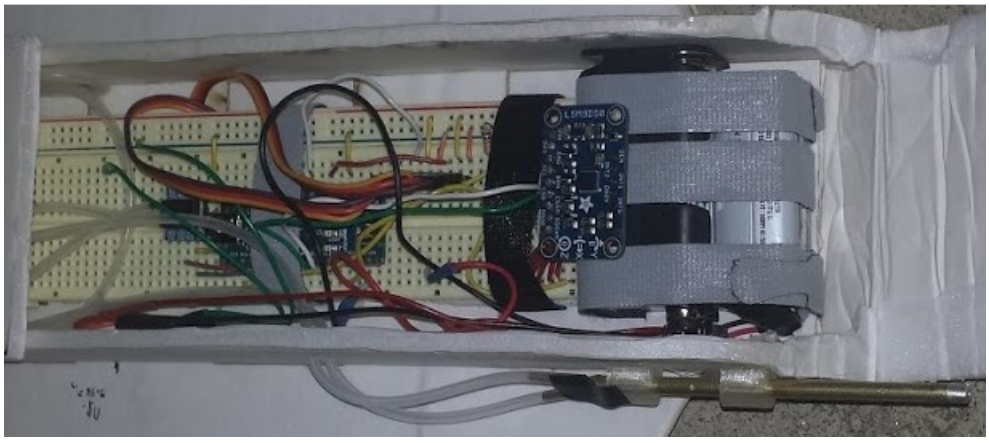

Figure 50: Breadboard Flight Controller Installed in Airframe

For the initial flight tests, a breadboard Flight Controller was used instead of the actual PCB Flight Controller. Many crashes were expected in the early stages of flight testing, so this setup was used to avoid damaging the Flight Controller. The breadboard contained the same IMU used in the actual Flight Controller, an Arduino Nano (which uses an ATmega328P), the airspeed sensor, and servo outputs.

4.14. Changing ATmega328P to Cortex-M4

A point came during software testing where the team realized the ATmega328P was simply not up to the task of running the ever-expanding autopilot code. The code had become so large, it had reached the limit of what the microcontroller could store. To remedy this issue, a Teensy was incorporated externally into the Flight Controller. The Teensy is a USB-based microcontroller development system which makes use of the Cortex-M4 microcontroller. It runs at a 72MHz, far faster than the ATmega328Ps 8MHz. Additionally, it has 262Kbytes of flash memory as opposed to the ATmega328P's 32Kbytes. [58]

A perfboard was setup to allow the Teensy to act as a drop-in replacement for the ATmega328P. This was done by making use of the Flight Controller's auxiliary I/O ports, and by setting up the Teensy to use the same ports for servos and sensors as the ATmega328P.

The addition of this hardware caused several changes to how the Flight Controller attaches to the airframe. The perfboard is attached to the Flight Controller using two nylon

screw and nuts. Those boards together are then attached to the airframe with 8 zip ties, using two large spacers in front and two small spacers in rear.

4.15. Results

As demonstrated in the testing described earlier, the Flight Controller was able to meet all requirements set for it. Using the Radio Module, weather sensors, and GPS, it is able to transmit weather data to the APRS network for easy retrieval, allowing the device to behave as a radiosonde. It is also able to control an aircraft using data from its 9DOF IMU and airspeed sensor. This data is used in controlling three separate servo outputs. Finally, it is able to transmit the device's GPS landing location via SMS using the GSM Module.

In addition to these three objectives, the Flight Controller was also kept well within the price range specified in section 3.1. As can be seen in the Bill of Materials (Appendix A), a fully-functional Flight Controller costs $405.98, and a full AURORA radiosonde-glider costs $425.81. These numbers are well under the specified $580 limit, and would be further reduced with revisions to the Flight Controller and with mass production.

4.16. Future Design Considerations

As with any printed circuit board, the first design was not perfect. Though everything worked in the end, design changes would certainly be made in version 2.0. The very first change would be the replacement of both microcontrollers with Cortex-M4s. These microcontrollers are far faster and more powerful than those originally chosen for this project. A proper power connector would also be added. One such as a two-wire JST would be much more resilient than wires simply soldered to the PCB. The connection between modules would also be changed. Instead of simply solder solid-core wire to interconnect them, actual connectors would be used.

The IMU would also see a few changes. The orientation of the IMU pins would be changed to allow easier setup, and the orientation of the IMU itself would change (it was accidentally set to be mounted backwards). The MOSFETs for controller power to the servos and GSM Module would be better sized for the amount of current actually required to pass through them. This would save on weight and component cost. The linear regulator for the GSM Module would also be changed. One with a far lower dropout voltage would allow it to operate with a lower battery voltage. Finally, the mounting holes on the Radio Module would be repositioned to the opposite side of the board, eliminating the need for the spacers to have arms extending out to the holes.

5. Airframe Development

5.1. Introduction

The airframe is the physical package that will give the radiosonde the ability to fly while providing protection to the electronics inside. This section lays out how the airframe was developed over the course of the project from initial concepts to the fully realized glider.

5.2. Overall Design Considerations

The airframe had several simple design considerations that are difficult to bring together. First, it had to be lightweight to fit within the limits of balloons already in use by the National Weather Service; approximately 450 grams. Second, it had to be cheap enough to offset the cost of the control board and stay within the same price range of current radiosondes. Third, the airframe had to be able to glide far enough to reach a safe landing location. For this a glide ratio of 7:1 was chosen after looking at the glide ratios of several unpowered aircraft, as well as the average distance a weather balloon drifts in a typical flight.. Fourth, the airframe had to survive landings without needing excessive, unscheduled maintenance to keep the airframe reusable. Finally, the airframe had to be made of materials that were typical of radiosondes and that could be consumed by a jet engine without damaging the engine in the rare event that the glider were to encounter another aircraft.

5.3. Preliminary Concepts

When the project began, the concept of a rigid airframe based off the Me-163 ‘Komet’ was already being considered. This design was first considered for the project because it is a high speed glider shape that was thought to be easy to produce. As the project commenced, other unpowered airframes were considered as well. The four major design types that were considered are detailed below.

5.3.1. The Rogallo Wing

Figure 51: A Rogallo Glider in Flight [59]

The rogallo wing was the leading contender for a non-rigid airframe design. It is common in ultralight aircraft and has been investigated as an alternative to parachutes by NASA. Rogallo wings are a type of flexible airfoil that consists of two self-inflating, partial-conic sections whose application varies from kites, parachutes, gliders, and powered aircraft.

The shape of the conical sections, specifically the ratio of the conical sections' heights and bases, determines the optimal flight regime of a rogallo wing. Wide, shallow cones are best for low-speed flight while long, narrow wings are faster, even able to reach supersonic speeds of Mach 2 or 3. [60] Because the wings are built to allow flexing and bending, they have springy dynamics and are less susceptible to turbulence than rigid wings, resulting in a gentler flight. The un-stiffened tailing edge also allows the wings to twist and provide directional stability without a tail.

NASA experimented with using rogallo gliders in the Gemini and Apollo instead of parachutes. The testing went as far as full-sized test articles in the Gemini project. Parachutes were ultimately used because they were simpler and were easier to implement with the space program's schedule. [61]

One of the biggest advantages of the rogallo for the project is that it can be folded up and spring loaded to deploy upon release from the balloon. The materials are also extremely lightweight; test gliders used for this project were made from plastic sheeting only a few thousandths of an inch thick.

The biggest drawback of the rogallo wing is that in a nose-down orientation, the wing does not generate lift. This can be combatted by adding some reflex curve to the tailing edge to induce an up-pitching moment at the cost of stability. This was one of the main reasons it was not ultimately chosen for the glider; the wing would ideally be starting out in a nose-down configuration from the weather balloon. There was also concern that in the near vacuum of

30,000 meters, there would not be enough air to inflate the wing upon release, which would have resulted in an uncontrolled descent.

5.3.2. The Square Parachute

Figure 52: A Navy Paratrooper Demonstrating a Square Parachute [62]

The square parachute, a type of ram-air parachute, was another glider design that was brought to the attention of the project group by an advisor. These steerable parachutes are commonly used by skydivers and paratroopers and use a lightweight, inflatable airfoil to provide lift. Some ultralight aircraft use this type of wing to provide lift as well. These parachutes are steered by guide cables attached to the corners that increase drag on one side when pulled to induce a turn. Stability is provided by suspending the payload well underneath the parachute to provide a pendulum effect.

The biggest advantage of the parachute is its weight; made up of thin sheets or fabric and string, the parachute would weigh only a few grams at the scale of the project's glider. The design of the parachute would also allow for the use of the same packaging that radiosondes already use; it would only have to be enlarged slightly to accommodate the extra sensors, servos, and batteries. Overall, a square parachute radiosonde would look very similar to current systems.

The biggest disadvantage to the parachute is the same as the rogallo wing: at 30,000 meters it cannot be ensured that the parachute will catch enough air to inflate. The problem is compounded in the parachute design by the fact that it does not contain a rigid framework to at least hold it open to catch thicker air as it falls into the atmosphere like the rogallo wing can. There were also concerns that the parachute strings could become tangled in the remains of the weather balloon after it burst, preventing any opening of the parachute.

Current radiosondes do use simple, non-steerable parachutes that slow their descent, but descent information of radiosondes is limited to rates, making estimates of when a parachute may begin to provide useful steering difficult to make. These parachutes are also low-speed wings with a glide ratio around 4:1. Even if the desired glide ratio for the project could be achieved, the forward speed of parachutes is relatively low and could be easily counteracted by winds aloft. [63]

5.3.3. Ember 2 RC Plane

Figure 53: The Ember 2 RC Plane

Looking into rigid airframes, one particular concept examined was the Ember 2 RC plane. This plane weighs less than 30 grams and is very agile with an impressive glide ratio. The Ember 2 also uses two control surfaces and a high dihedral wing to maneuver. Prior flight experience with this plane before the project led the group to agree that it is a fun and surprisingly capable airframe. [64]

The biggest advantage of the Ember 2 is its simplicity. With only a rudder and elevator, controlling this airframe would be easy. The Ember 2 is also great at recovering from dives at

any speed, eliminating concerns of gaining control after the balloon bursts. This plane also exhibits an ability to stay in control after its battery power dropped too low to run the propeller.

The Ember 2's greatest drawback is its flight speed. It is meant to be an indoor glider and, even when under power, cannot move forward in any significant wind. Like the parachute, this would make returning to a reasonable landing site all but impossible with a headwind. While the Ember 2 is well within the weight limitations of the project, it is simply too lightweight and fragile. Control surface will wear out and tear in the wind, and the battery is far too small to last for the hours it would need to in a full flight. Scaling it up to accommodate the needs of the project was an option, however it was decided that this would require more materials than the other rigid option, the flying wing.

5.3.4. Flying Wing/ Tailless Delta

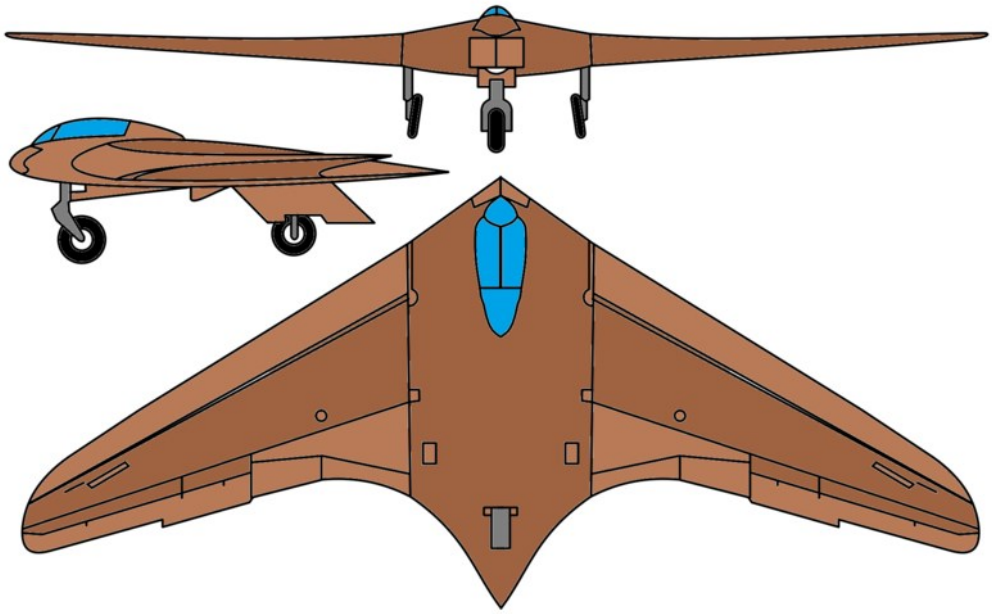

Figure 54: Schematic of h0 229 WWII Flying Wing [65]

The flying wing design, initially taken from the Me-163 concept the project began with, later evolved to look at aircraft such as the Ho 229 and lippisch delta wings. Flying wings are challenging aircraft which often share characteristics of tailless delta winged aircraft.

A tailless delta wing cannot handle high wing loading and requires a larger area to lift the same weight as equivalent swept wings. Efficient airfoils are also unstable on tailless delta wings; so less efficient, yet stable airfoils such as symmetrical airfoils or airfoils with reflex camber are used. Another option is to lightly twist the outer leading edges down, which reduces the incidence of the wingtip and improves characteristics of the wing in a stall as well as at supersonic cruise.

The advantages of a flying wing for the project are the speeds the wing can accommodate and the lift is can produce. Like the rogallo wing, it can fly in any flight regime but would not have the inflation issue the rogallo would encounter. Having discrete control surfaces instead of flexing the wing was another advantage over the rogallo and parachute that would make controlling the glider easier from a programming standpoint. A computer simulation also showed

that a flying wing design provides all the speed and maneuverability that would be required from the glider.

The disadvantages of the flying wing are mostly stability related. As mentioned before, they cannot handle high wing loading and need larger areas to not break apart. There is also little inherent yaw stability in a flying wing. Larger aircraft like the B-2 Spirit solve this by using split control surfaces that can deflect both up and down at the same time. The issue was solved on the project glider by using a design with proportionally large winglets that act as vertical stabilizers. Flying wings also have a tendency to 'porpoise', or dive when stalled, pick up speed, then lift their nose until they stall again and repeat the process. It was decided to move forward with this design and rely on the flight computer to compensate for porpoising.

5.4. Initial Tests

5.4.1. Virtual Testing

The program RealFlight 7.5 was used to test different airframes and get a feel for their performance, as well as get solid numbers on the expected performance of any particular airframe. RealFlight simulates RC aircraft with realistic physics and controls and is used to train RC pilots on their aircraft before going out to fly the real plane. While it could simulate most rigid airframes, options for flexible airframes like the rogallo wing or parachute could not be found. Use of this simulator was continued because it was the cheapest available to the project group, needing only a $60 upgrade from version 4.5 already installed on one of the computers.

In these tests, the 'Slinger' flying wing, a 24-inch (.6 meters) wingspan powered aircraft that was the closest to the Me-163 and Ho 229 flying wings, was found to have the desired characteristics for the project's glider. The file's characteristics were modified to reduce its size by 1/2 (12-inch, .3 meters wingspan), remove its motor, and determine that the weight was correct for the project: under 250 g. While flying the model manually in the simulator, it was determined that it could reach speeds as high as 115 mph in a dive and glide ratios approaching 20:1. The simulator's file revealed that it was using a Sipkill 1.7-10B airfoil and was made of foam.

5.4.2. Other Initial Tests

Since the rogallo wing could not be simulated in RealFlight, a model was made out of spare parts to test in the real world. Using old welding rod and polypropylene liner, several models were constructed that were used to gauge the feasibility of the rogallo wing.

Figure 55: Initial Testing of Rogallo Glider

The first model worked well and achieved a 9 meter flight when thrown from 1.8 meter (5:1 glide ratio). The construction was uneven and it had a tendency to fly to one side. Solder had been used to attach welding rod frame pieces, which did not work, and the plastic liner was unevenly cut. In efforts to fix these problems the model was destroyed.

The next model was better constructed and more symmetrical. Notches were used in the frame with hot glue to hold it together. While it did fly straighter than the first model, after flying about 3-6 meters, it would nose dive out of control. To improve the angle of attack, a deadweight was attached underneath the wing since the resources to make an active steering system were not yet available to the project. This weight worked to improve the angle of attack, but made launching the wing more difficult and unstable. The concept was soon abandoned after testing of the physical flying wing design began.

5.5. Designing the First Rigid Airframe (Pink Foam)

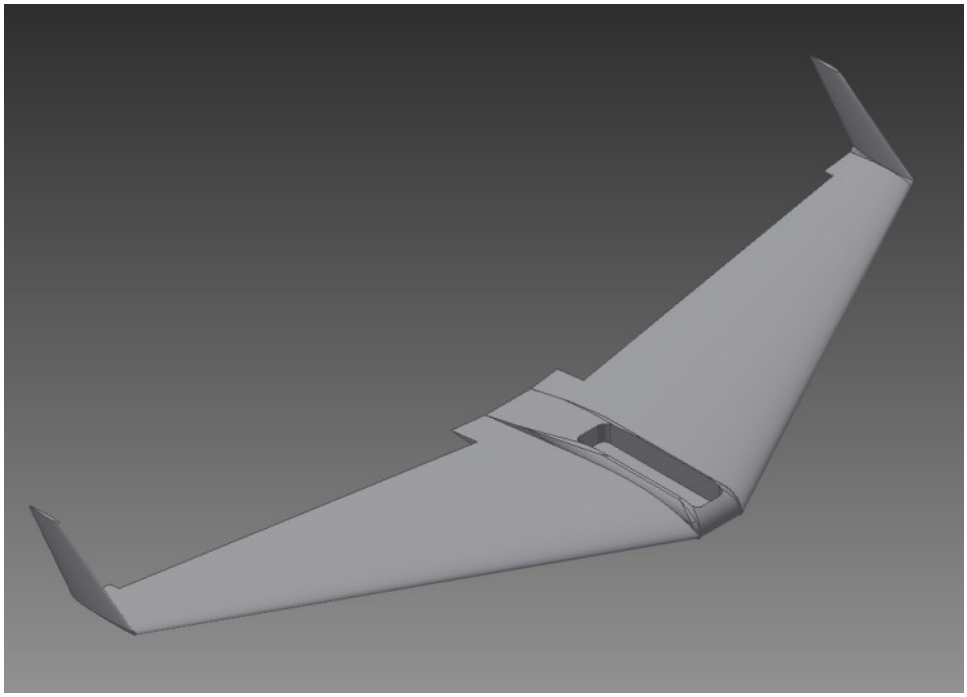

Figure 56: CAD Model of Pink Foam Prototype Glider

5.5.1. Selecting Materials

To validate the simulation, the decided was made to use a foam model like the 'Slinger' aircraft. Pink foam insulation was selected as the material for its light weight, rigidity, and ability to be shaped by different manufacturing techniques. It was also decided that CNC machining would be the fastest and most repeatable way to achieve the complex airfoils needed. Solid models were created of the wings as well as a fuselage big enough to hold RC electronics for testing.

5.5.2. Manufacturing Difficulties

Initial plans called for the pink foam to be cut using a CNC mill for accuracy and repeatability. Those in charge of the CNC mills advised against the plan on account of the glider's size and a new method was researched.

After observing the foam cutting techniques of other aerospace MQPs, templates were produced of the airfoils that could be laser cut from plywood and used to guide a hot wire cutter through the foam. Despite the best efforts of the team, the foam proved very difficult to cut at the project's scale. The heated wire could not be kept taut and moving both ends of the wire at different speeds through the foam was impossible to do by hand. The resulting parts were too thin, too flimsy, and often had uneven surfaces from where the hot wire melted them too far.

The differences between these results and the more successful results of the other MQP teams were scale and shape. The other MQPs were larger, having wingspans three to four times larger than this glider's. Their wings also did not taper like the glider's and had the same chord length at the root and tip of the wing, allowing them to move both ends of the hot wire at the same speed through the foam.

5.6. Designing the Second Rigid Airframe (Small Foam Board)

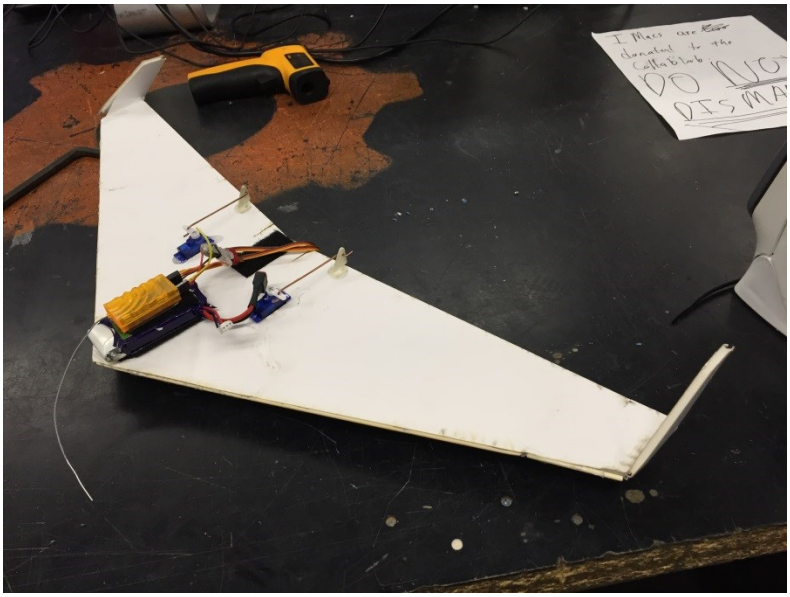

Figure 57: Small Foam Board Glider with RC Equipment

5.6.1. Selecting Design and Materials

With the complete failure of the pink-foam design, the project was left with little time before a deadline to have a flying model before the end of B-term. Various other materials and manufacturing methods were brainstormed that could be used within the span of a couple days. It was clear that the project would have to stick to foam; it was the only material light and strong enough that the project had access to. The manufacturing options were narrowed down to two: Injection molding and laser cutting.

Injection molding of expanding foam is how most foam RC aircraft are manufactured. It would allow the retention of the same design yet allow the addition of supportive wing spars. The main drawback of this method is the time required, as a negative mold would have to be manufactured using materials the project could not access in time, such as aluminum. The idea of using laser cut plywood segments stacked up to create the mold was toyed with, but it was determined that generating that many profiles would be difficult and would require even more finishing work to sand them smooth.

The final option was to use foam board (polystyrene sandwiched between two layers of craft paper) in a laser cutter to create a flat version of the glider. RealFlight was brought up again and the 'Slinger' model's airfoil was changed from the Sipkill 1.7-10B to a 4% flat plate with bullnose edges. To the surprise of the group, an even greater glide ratio than the first version was seen in the new simulation. It was determined that the tradeoff was in the stall characteristics, which were less controlled than the Sipkill airfoil's stall. This was the solution the project continued with.

5.6.2. Manufacturing

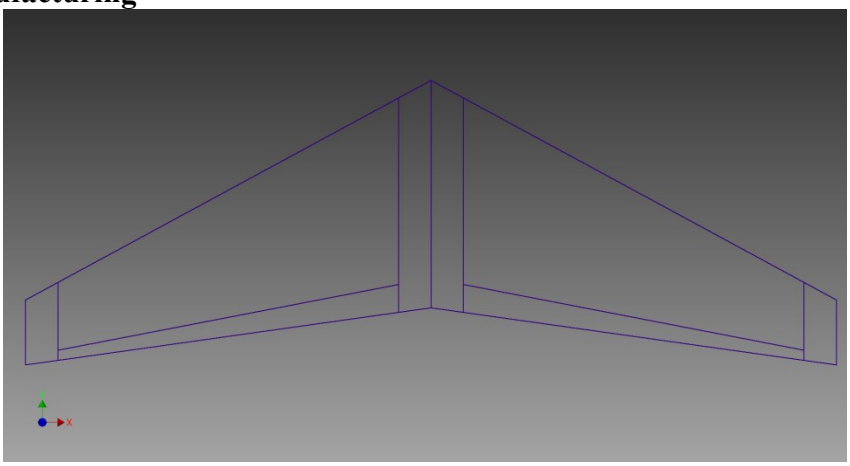

Figure 58: CAD Drawing For First Laser Cut Glider

Creating the drawings for laser cutting proved far easier than generating the airfoil in any CAD program. This model was referred to as the 'half-scale' model because it has half the wingspan of the original 'Slinger' model. Laser cutting the foam board was a quick process, and the actual operation only took a minute or two to create a full set of parts for this airframe. Since the laser cutter can vary its output, control surfaces can be cut directly into the wing without cutting all the way through the board, and mounts for servo motors could be included as well.

Laser cutting does have one major drawback with the foam board: it melts the foam back from the cut edge of the paper. This gives a concave edge that is not suitable for the leading or trailing edges of a wing. Since the simulation used bullnose edges, bamboo skewers were mounted to the leading edges with hot glue to provide the desired leading edge and increase stiffness.

5.6.3. Testing.

To test this airframe, traditional RC controls were used to manually pilot the glider, since none of the autonomous controls were ready. The first rounds of testing were done in Alden Hall, where glide ratios of about 2:1 were seen. The center of mass was moved around to find the optimum position, however batteries were used that were far heavier than were expected to be using later on, throwing off these estimates. Throwing tests were also conducted from the hill behind Morgan Hall overlooking the football field, as well as inside Harrington Auditorium.

5.6.4. Results

The glide ratio never reached the targeted 7:1. Many control issues were seen which were later determined to be caused by the airspeed: the model could not get above its stall speed. It was also found that with hard landings, the nose of the glider was being destroyed. Adding a pink foam skid helped, but only if the glider managed to land belly first, which it could not do with the weight balance it had. Finally, in one test the glider collided with a chair in Alden hall. The collision broke the bamboo leading edge and damaged the wing behind it.

5.7. Designing the Third Rigid Airframe (Large Foam Board)

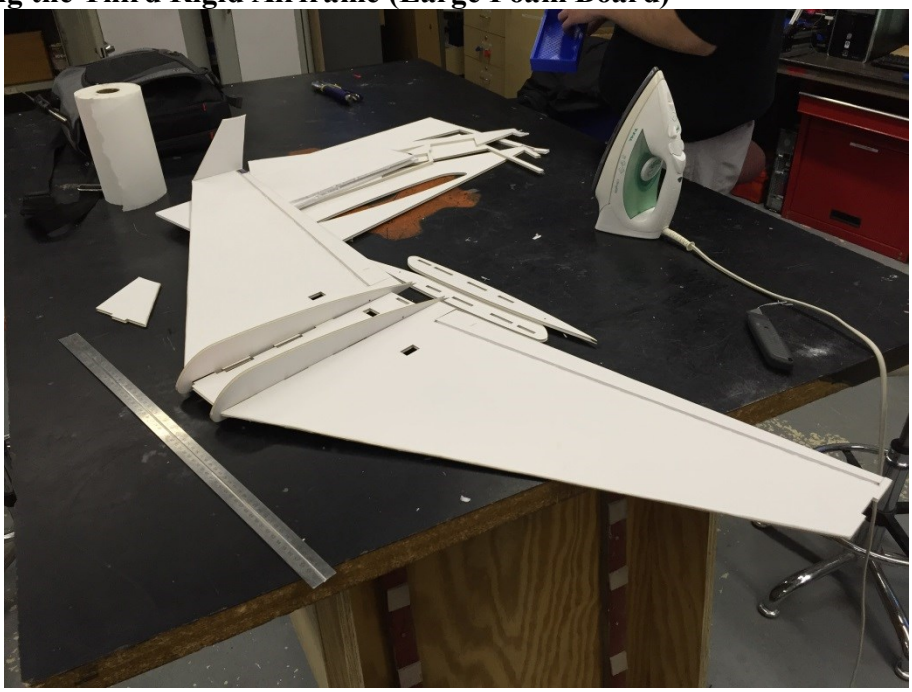

Figure 59: Large Foam Board Glider Under Construction

5.7.1. Selecting Design and Materials

For the next design overhaul, the wingspan was doubled to see if less wing loading would give a better glide ratio while still staying under the project's ultimate weight limit of 450g. This design is referred to as the full-sized model because it is the same scale as the 'Slinger' model with a span just over 1.2 meters. Due to the size limits of the Washburn laser cutter, the full-sized model had to be broken into three parts: two wings with a bulkhead in between them that would carry the electronics. Two fuselage walls were made that had an arbitrary aerodynamic shape to keep the fuselage streamlined and reduce drag from the electronics. The wings and bulkhead slot into the fuselage walls, and bamboo skewers were pushed into the foam as wing spars. To cover the fuselage, transparency sheets were found that were flexible and easy to cut.

5.7.2. Manufacturing

Several manufacturing processes were changed for this larger model. Since there were not enough bamboo skewers to both cover the leading edges and act as wing spars, a method of ironing over the paper edges was used to produce a bullnose edge. This worked brilliantly, stiffening not only the leading edge but any other aerodynamic edge that was needed without adding weight with glue or bamboo. Ironing the edges also added strength to the foam board and was easily scaled to any edge on the glider.

Pressing the bamboo skewers into the foam provided a strong friction fit that still allowed the wings to be pulled off for transportation and servicing. Four skewers were used along the length of the glider to provide support.

To cover the fuselage, transparencies were used that could easily be bent around the curve of the fuselage walls. The transparency covers had tabs on both sides so they fit over the walls and scotch tape was used to hold them down. It was discovered that scotch tape loses adhesiveness in the cold and the covers often had to be re-taped after every flight.

5.7.3. Testing

By this point in the project, the weather was not favorable for repeated testing outdoors, so many of the tests were conducted in Harrington Auditorium, and a few on the hill behind Morgan Hall when the weather permitted. These tests were the same as before, throwing the glider to see how well it controlled and how far it could fly. Drop tests were also conducted to show that it could recover from a nose-down orientation with RC control. Once the first set of autonomous code was ready for test, it was tested with this model.

5.7.4. Results

This glider performed much better than the half scale glider. Glide ratios of around 4-5:1 were seen when RC receiver issues did not prevent it from working. Since these tests were still operating at or around the stall speed of the airframe, it had a tendency to crash. Crashes would damage the nose of the fuselage but usually left the wings unharmed; however they would separate if a crash was particularly hard. On occasion, the glider ran out of space in Harrington to fly, and would hit the back wall.

One phenomenon that became apparent in these tests was a flapping of the wings when the glider maneuvered in any way. These oscillations induced drag and reduced the glide ratio, and eventually led to the destruction of this airframe. In one test with the autonomous controller (an Arduino Mega hooked up to a breadboard that contained the necessary sensors), the glider was thrown off of the hill behind Morgan Hall. The oscillations grew so severe that the bamboo skewers snapped and the wings folded in flight.

5.8. Developing the Fourth Airframe

Figure 60: The Fourth Airframe

5.8.1. Revising Design

Considering Alternative Designs

After the wings folded in the test flight, the team began reviewing the possible causes for the failure. Slow-motion video was recorded during the test; together with the remains of the glider, the joint between the wings and body of the glider was identified as a stress concentration point and steps were taken to reduce the stresses there. Reducing the size of the wings to about a 1 meter span to reduce the moment around the joint, or even going back to a .6 meter span to remove the joint entirely were considered as well. Ultimately, these designs were decided against because they would not give the 7:1 glide ratio goal set for this project.

Changes Made to Existing Design

Another step was to change the length of the wings' inserts. In the old model, these tabs were only 5 mm, the same as the thickness of the foam board used in the fuselage walls. This made the moment of the wings flexing centered on the same axis as the joint between the fuselage walls and bulkhead. To solve this, the tabs' lengths were increased to 20 mm so that they extended into the bulkhead.

The fuselage cover was also changed. The transparencies were abandoned and replaced by foam board after it was found that bending some scrap material left over from the laser cutting process would suffice. Hot gluing the excess foam board to the fuselage wall proved

difficult, but resulted in a stronger fuselage body and provided more of a crumple zone for protecting the electronics from crashes.

While the final control board was being assembled, a switch away from the Arduino Uno test board was made because it was discovered its setup was too heavy for the glider (the overall system weighed in at over 500g) The old Arduino was replaced by a spare Arduino Nano. To further reduce weight, the alkaline batteries used in testing were replaced with lighter lithium batteries.

Center of Mass Revision

Closer attention was given to the dynamics of the glider itself after some particularly bad crashes which lacked any control. Equations were found for calculating the aerodynamic center (neutral point) of flying wings [66]. Since the center of gravity (CG) was supposed to be ahead of this point, it was assumed that the first calculated neutral point made sense given that the current CG was already a few centimeters ahead of it.

More crashes followed and an even closer look was taken at the CG, going as far as to look at the original RealFlight file to figure out why the glider was not working. The group found that the CG for the model was much farther back than that of the real world glider. It was then calculated that the CG should be further back, 6 inches (15 cm) from the leading edge of the wing at the root. Since this was behind the calculated neutral point, another look was taken of that calculation, and it was discovered that it was only accounting for half of the overall wingspan. Fixing the error brought the neutral point behind the CG as it should be.

After fixing these errors, a new problem arose getting the actual CG to line up with the desired CG. Since the batteries were the heaviest components, they were moved from the nose to behind the desired CG position. Ideally, the project group wanted to place them under the bulkhead and wings to give the glider more stability, but this would require making the fuselage walls bigger to accommodate them. Instead, the battery pack was switched from a square arrangement to a flat one and placed behind the second wing tab. Flight testing saw immediate, significant improvement with the balanced CG.

Finally, the connection between the fuselage walls, bulkhead, and wings was simplified by combining the four slots in the fuselage wall into one long slot. This gave the bulkhead more area to connect to the fuselage walls to and allowed the wing spars to be moved to be easily moved to more optimum points along the wings.

5.8.2. Revising Selected Materials

As more replacement parts and airframes were being cut, it became clear that the $6 Elmer's foam boards that were being used would be too expensive. Foam boards were then purchased from the dollar store for $1 as a brand called "Readi-board". The only difference between the two brands is the type of paper the polystyrene foam is sandwiched between. The craft paper on the Readi-board turned out to be lighter than the paper used on the Elmer's boards and saved a significant amount of weight at the cost of rigidity.

After more crashes and being unable to remove the flapping behavior with code, the decision was made to stiffen the airframe further by removing the bamboo spars and replacing them with carbon fiber. The carbon fiber had ten times the stiffness as the bamboo and had the desired effect; however further testing showed that as the airframe became more damaged, particularly the bulkhead, the flapping behavior could return at high speeds.

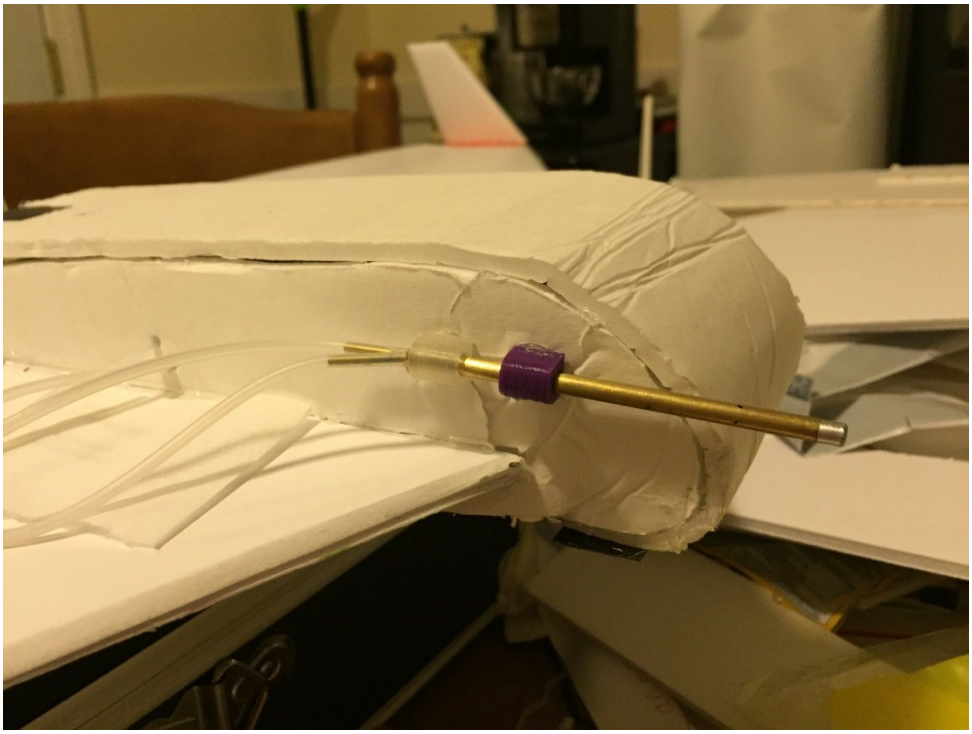

Figure 61: The Pitot Tube Mounted to the Airframe

An airspeed sensor was added that required a pitot tube be mounted to the outside of the glider. To receive accurate airspeed data, the pitot tube must be placed facing forward ahead of every other part of the airframe so that airflow around the body does not affect the sensor. 3D printed mounts were designed and printed out of PLA and were hot glued to the starboard fuselage wall above the wing. Holes for the sensor's tubes were punched into the wall manually.

5.8.3. Revising Manufacturing Process

As more airframes were rebuilt and assembled, the manufacturing process became more streamlined. The amount of hot glue used to attach the fuselage cover was reduced to allow the cover to bend open more and give better access to the electronics. Ironing the joint between the wings and winglets was also stopped because ironing the joint seemed to make it weaker; this decision also saved time. Lengths of carbon fiber were cut from 4 foot, 3/32" round stock (1.2 meter, 2.3 mm).

Until this point, the bamboo wing spars had simply been pushed into the foam with brute force. The bamboo skewers were pointed on one end and another point was cut into the other end to make the spars easier to push in. Since the bamboo wasn't always straight, it could curve towards the front or rear of the wing, or even towards the top or bottom and tear the paper. The carbon fiber did not have pointed ends, and cutting them would have been extremely difficult with the tools available; therefore a new method was developed.

A hot air gun was used to heat up one end of a steel welding rod with a diameter slightly smaller than the carbon fiber rod. The hot end of the welding rod was then pushed into the foam and used to melt it. This method required less effort and time than pushing the spars into the wing and usually created straighter paths for the spars to follow. Carbon fiber can then be inserted into the wings carefully, although pushing a bamboo skewer in first can expand the hole if the steel rod did not melt enough and prevent the carbon fiber from tearing at the foam.

To mount the control board, zip ties were used along with 3D printed PLA washers and spacers to spread the force more evenly over the bulkhead. The zip ties were used to facilitate quick removal and attachment to new bulkheads as the old ones were damaged in crashes.

5.8.4. The Release Mechanism

Figure 62: 3-Ring Release Mechanism used in Skydiving [67]

Development of the release mechanism took about a week. From research done into parachutes, a 3-ring release mechanism used by parachutists was found that was thought to be a reliable release mechanism for the glider. A set of 3D printed rings were made and sewn into a spare scrap of fabric to test the mechanism at the scale of the glider (most 3 ring releases operate at the weight of a full grown human). While the mechanism worked on occasion, it could not release consistently and had a tendency to catch on itself.

Figure 63: Final Release Mechanism

Further research was done looking at RC glider release mechanisms and it was found that a common solution was to glue tubes to the side of a servo with a short control rod passing through. A glider's tether was tied to the control rod and when the servo was actuated, the tether slid off and released. It was thought that putting the entire weight of our glider on one servo in the rear would be a bad idea, so instead of tubes glued to the servo body a PLA mount was

designed and 3D printed to be glued to the bulkhead next to the release servo. Testing showed this setup released successfully and reliably.

5.8.5. Testing

At the previous test sites, there was not enough altitude to gain airspeed and properly test the capabilities of our glider. At the suggestion of the project advisors, the Gateway parking garage was chosen as a new test site, given its height of just over 18 meters. For these tests, the glider was thrown either towards the field to the southwest or into the parking lot to the north. To avoid hitting cars or people in the parking lot, testing was typically conducted at night or after business hours, although some exceptions to these times were possible with favorable winds.

At this point the final control board was assembled and allowed for SD card logging of flights, giving us more data to analyze after each flight. These results are covered in section 6.

5.8.6. Results

The iterative tests off the Gateway garage provided much insight into the ruggedness of the design in an urban environment and during worst case scenario landings. In the first two tests off the 18 meter structure, there was no control and the glider impacted nose first onto pavement. While disheartening, these tests did prove that the airframe could survive at least one impact of this nature and still fly again, and even after the second crash it proved that it could protect the electronics inside from major damage. The exceptions were the then still nose-mounted batteries, which suffered the brunt of the impact and had to be replaced. Moving the batteries to the rear fixed the control problem and made impacts less traumatic to the airframe.

After the CG and aerodynamic center problem was fixed, the best glide ratio yet was achieved of 8.8:1, meeting the target of 7:1. There was still significant ‘porpoising’ in this particular flight, so an even better glide ratio is still feasible. Later testing focused more on the navigation qualities of the glider and no attempts were specifically made to demonstrate the 7:1 glide ratio.

More flight tests that ended in diving crashes from software bugs revealed that the new fuselage design with the single slot was more easily deformed or broken than the older version with multiple slots. After using up all of the single slot versions of the glider, it was switched back to the multiple slot configuration.

5.9. Final Test Revisions

The final airframe underwent a few minor revisions and testing before the final drop test. The biggest revision was the method of cutting the wings, winglets, and ultimately fuselage walls. Due to an accident in the Washburn laser cutter that was handled poorly, the foam board was banned from being cut with the laser. Therefore a plywood template was cut on the Washburn laser cutter to facilitate hand cutting of the wings. When time ran out to use the laser cutter between the first and second attempted balloon drop tests, the last cut fuselage wall was used as a template to hand cut more from the remaining foam board.

At about the same time, it was decided that the servo motors should be moved closer to the control surfaces. This was done in order to shorten the control rods, making them stiffer and prevent them from bending when excess force was applied to the control surfaces.

The first balloon test flight ended in disaster when the glider was buffeted against the balloon’s tether by the wind. As a result, the left wing was torn off and the fuselage cover came open. While the wing was recovered, the 24 cm length of carbon fiber was lost, along with an Xbee module being used to gather real time data. After the test, several small adjustments were made to the design to ensure the next test would work. The wing tabs were cut slightly wider and

the fuselage holes were cut slightly smaller for a tighter friction fit. The wings were then glued in place to ensure they would not come apart. A ¼ inch zip tie was inserted through the fuselage to anchor it closed, instead of relying on electrical tape as previous versions had. Larger 3D printed washers were made to be used with the larger zip tie.

Finally, as cosmetic changes, the left wing tip was colored orange and the right wing colored blue. Traditional red and green navigation colors would have been used but those colors were not available.

5.10. Final Airframe Results and Analysis

Figure 64: Final Glider Ready for Balloon Testing

The glider developed in this project underwent several major and countless minor revisions before the final test flight. At the end of the project, the glider proved it could meet or exceed all but one of the initial criteria. The final dimensions and results are outlined below. Further details and equations used can be found in the appendix.

Wing Root Chord	34.1cm
Wing Tip Chord	9.7cm
Wingspan (root to tip)	56cm
Wingspan (tip to tip)	1.2m
Wing Sweep	28.4º
Wing Dihedral	0º
Winglet Root Chord	9.7cm
Winglet Tip Chord	3cm
Winglet Span	9.7cm
Winglet Sweep	50.7º
Winglet Dihedral	90º
Fuselage Width	7.5cm
Fuselage Height	6.7cm
Fuselage Length	41.5cm
Center of Pressure Location (Distance From Leading Edge Of Wing Root)	21.4cm
Center of Gravity Location (Distance From Leading Edge of Wing Root)	15cm
Ideal Terminal Velocity	98.875 m/s
Mass	367g (450g max configuration)
Glide Ratio	8.8:1

Table 5: Airframe Characteristics

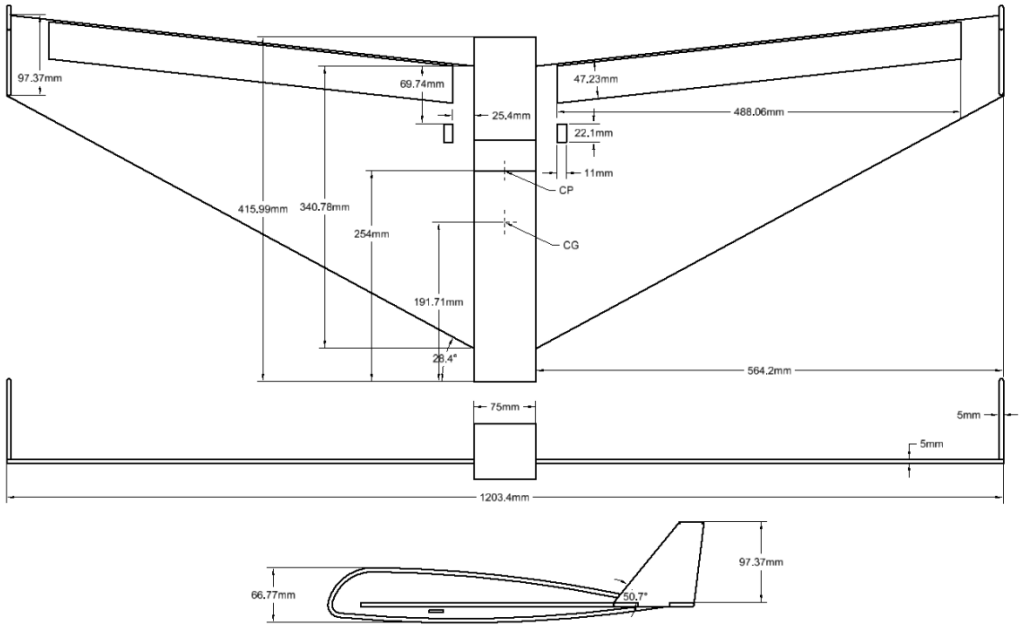

Figure 65: Schematic View of the Final Glider

5.10.2. Mass

The total final mass of the glider came to 367g with an all Readi-board frame, and was at the limit of 450g when the wings had to be switched back to Elmer's foam board due to time constraints. This mass includes the extra electronics that were added later in the project for data gathering as well as ballast applied to the nose in the final glider to offset the heavier wings.

5.10.3. Airframe Properties

The fuselage's shape is an arbitrary shape that streamlines the electronics compartment. In future development of the project this shape can be better optimized to reduce drag and accommodate more or less space. It may even be switched out for a known airfoil shape whose lift and drag coefficients have already been determined.

Using the best estimates for overall drag on the glider, the ideal terminal velocity of the glider at sea level in a dive is 98.875 m/s (221.177 mph). This number takes into account drag induced by lift, the shape of the glider, and friction with the air. It is called ideal because the glider will begin to flutter at speeds in excess of 22 m/s (50mph). The intended flight velocity for testing was set to 8.9 m/s (20 mph), which was just above the 6 m/s (14 mph) stall speed predicted by the RealFlight 7.5 simulations.

With the carbon fiber spars, the glider can be expected to handle bending forces of up to 235 N, assuming all of the force is transferred from the wings to the spars. This would be equivalent of suspending the glider by its wings and applying 24 kg of mass to the center. This is also a 10-fold increase from bamboo, which could only handle 28N, or an equivalent mass of 2.9 kg, despite using more spars.

5.10.4. Control Surface Modeling

The control surfaces were modelled as flat plates in a fluid jet to calculate the forces that should be expected on them. The servo motors being used are also good up to approximately 40m/s (90 mph), where they begin to stall out at high deflections. At ideal terminal velocity, the servos will only be able to deflect a maximum of 8.145º. With the forces and moments calculated, as well as the moments of inertia and some relatively obscure coefficients of lift and drag, the roll rate could also be determined. The constant was found to be approximately .05839 1/m, which can be multiplied by the deflection of the control surfaces in degrees and velocity of the glider in m/s to find the roll rate in deg/s.

5.10.5. Cost

The cost of the final glider comes in at around $20. This cost is broken down in the next section, but includes the frame materials and control servos. The use of foam board for most of the airframe is what makes the glider so cheap and easy to produce. Carbon fiber spars are the most expensive component, more so than all the other frame materials combined. The carbon fiber is also one of the most reusable components along with the servos, so the cost is justified when the spars can be reused after the rest of the glider is destroyed.

5.10.6. Glide Ratio

The glide ratio of 7:1 was achieved in a test flight that happened at night when significant porpoising was involved at the end. Early calculations based on integrals of the airspeed showed this test was only 6.666:1, however reviewing the video later on and using trigonometry, the actual glide ratio was determined to be as high as 8.8:1. It is reasonable to assume that other test flights achieved similar or better glide ratios as well; however the positioning of the camera in those tests as well as the quality of nighttime filming means they cannot be verified.

5.10.7. Survivability

The survivability of the airframe was thoroughly tested. It can survive a dive from 18 meters with some damage, yet can be made ready to fly again with only a hot glue gun and extra zip ties to replace any that may have sheared. Repeated impacts will of course require the airframe to be replaced eventually, but the low cost and guarantee that it will fly again if recovered is one of the most important and impressive features of the glider.

5.10.8. Materials

Finally, the materials used are similar to those used on current radiosonde devices. Foams and light-duty plastics are the most common frame material on current devices, so it made complete sense to construct the glider out of similar materials. Ultimately, whether the glider can be consumed by a jet engine without damaging the engine could not be tested within the scope of the project.

6. Software Development

6.1. Introduction

The software for the AURORA project can be divided into three main parts, radiosonde code, navigation code, and autopilot code. The radiosonde code is responsible for recording, transmitting, and logging weather data. The navigation code performs the calculations required to determine a landing site and the flight path to it. The autopilot code is responsible for the direct control of the aircraft. The radiosonde and navigation code both run on the navigation computer, while the autopilot code runs on the autopilot computer. Both the radiosonde code and navigation code were fairly straightforward to code, even though they were complex and required significant research to develop. The testing of the radiosonde and navigation code revealed the need for only minor changes and corrections. This was not the case with the autopilot. The autopilot code changed completely between early versions and the final version. Even up into the latest versions of the code the foundational elements would need significant alterations to provide the desired effects.

6.2. Flight Mode State Machine

Both the navigation computer and the autopilot computer are governed at the highest level by a state machine. This state machine handles the requirement for each microcontroller to have multiple modes of operation to correspond with different stages of flight. Some stages of flight require completely different processes to be running, and a state machine makes this process relatively easy. The following is a table of the various modes in roughly chronological order along with a short description of what the navigation and autopilot computers do during that mode.

Mode Name	• Navigation Computer Jobs	• Autopilot Computer Jobs
Preflight	• Acquire satellites • Test sensors	• Calibrate gyros • Test sensors and servos
End Preflight	• Nothing	• Shutdown servos
Ascent	• Acquire weather data • Log & transmit weather data	• Check for release conditions (freefall and high airspeed)
Release	• Transmit release message • Begin landing site evaluations	• Activate servos • Disconnect from balloon
Flight Heading	• Transmit occasional updates • Update nav. info regularly • Send nav. Info to autopilot	• Fly safely • Follow heading given by navigation computer

Flight Spiral	• Transmit occasional updates • Update nav. info regularly • Send nav. Info to autopilot	• Fly in gentle spiral
Flight Land	• Transmit APRS & GSM updates in case of crash • Update nav. info regularly • Send nav. Info to autopilot	• Fly slowly along heading given by navigation computer
Postflight	• Send regular homing signals and messages	• Maximum power conservation (servos off)
Flight Level*	• Recover from issue	• Fly wings level at a safe airspeed • Recover
*this mode is available for emergencies where heading following cannot be trusted for whatever reason		

Table 6: Software Operationl Modes

6.3. Radiosonde Code

The primary goal of the radiosonde code is to record and transmit the weather data obtained from the sensors. The functions required to accomplish this goal are listed below:

- Read and interpret data from the onboard weather sensors (Temperature, pressure, humidity)
- Read and decode NMEA sentences from the GPS module
- Transmit data via the APRS module
- Record weather data onto the SD card

The radiosonde code is fairly simple and uses a number of canned functions to read and transmit the data. The code is set up to transmit data every 5 seconds to avoid saturating the APRS network with packets. For convenience the data is written to the SD card at the same time. In the time between the transmissions, the controller is waiting for GPS strings to be received via UART. As soon as that data arrives, it is pulled into the Arduino TinyGPS+ library so that it can be parsed. This library handles the complex task of parsing out the numerical data from a long a complex string as well as providing that data in various units for use by the rest of the code. The TinyGPS+ library is well tested and proven. TinyGPS+ stores decoded data in variables that can be accessed by external code. These variables are automatically updated as fresh data is received.

6.4. Navigation Code

The primary goal of the navigation code is to perform the data, memory, and math intensive functions involved in the return segment of the flight. These functions are listed below:

- Read and decode NMEA sentences from the GPS module

- Perform Great Circle Navigation calculations to determine range and course to the landing site
- Determine best landing site based on range and weather data acquired during ascent
- Determine which flight mode the autopilot to use based on the range to the landing site
- Send frequent updates to the Autopilot on the current GPS heading, the bearing to the landing site, and the current GPS ground speed

Based on the fact that the GPS module operates at 1Hz, and all of these calculations utilize GPS data, the update rate is also set to 1Hz. This provides the optimal balance between update rate and reading GPS NMEA sentences from UART. The NMEA sentences are decoded with the Arduino TinyGPS+ library.

The navigation code uses GPS and preloaded data exclusively to perform the calculations necessary to find the course to the landing site. These calculations are all categorized as Great Circle Navigation calculations. Great Circle Navigation relies on Great Circles, which are a means of finding the shortest path between any two points on the surface of a sphere. A Great Circle is defined as the circular "slice" of a sphere intersected by a plane and defined by two points on the surface of the sphere and the center of the sphere. The shorter arc segment along the circle between the two points is the shortest path.

Great Circles also provide a method of calculating the heading between two points. This calculation; however, is somewhat more complex. Because the path between the points is an arc, it does not follow a single heading along the entire path. The calculation used here is for the initial heading, which if followed will get from point A to point B, but not in the shortest path. The software overcomes this because the starting point is always the current location of the vehicle. By repeatedly performing the initial heading calculation during the flight, the software is able to keep up with the continuously changing heading associated with following a Great Circle arc. The two calculations for Great Circle Distance and Heading, provide nearly all of the information needed to navigate to the selected destination. In addition to providing the information used during navigation to the landing site, these calculations are also used to determine the best landing site.

Prior to the start of the flight, all of the landing sites are loaded onto the SD card in a specified format which includes the latitude and longitude in decimal degrees, the heading for the landing site, and the ground altitude of the landing site. This data is loaded into memory at the start of the flight. It is not until the balloon reached apogee that the landing data is next used.

Immediately after release from the balloon the navigation code begins the process of determining the best landing site. The best landing site is determined based on two parameters. The first, and most important, is the distance between the current location and the possible destination; and the second is the wind direction and speed recorded during ascent. The distance aspect to this calculation is basic logic: the closer a particular landing site is to the current location, the better the landing site. The wind based calculations are complex and operate on a

case-by-case basis. The decision tree below details the process for how weather data is used to determine the best landing site.

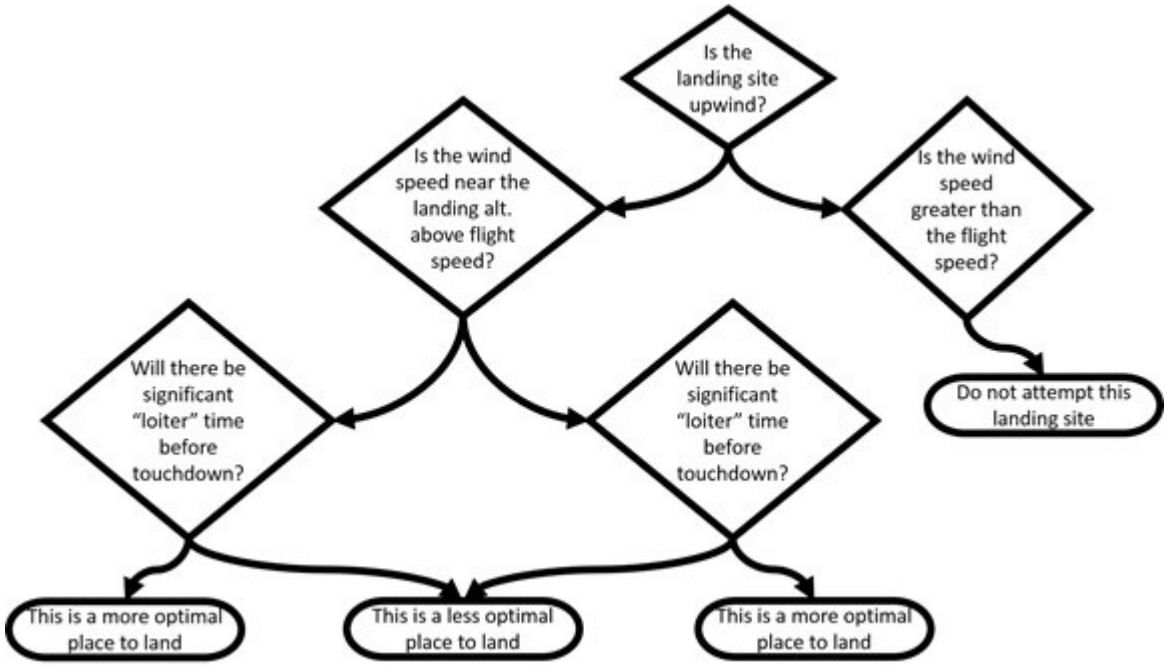

Figure 66: Decision tree for determining quality of landing site based on weather

Once the navigation code has selected a landing site it communicates the information to the autopilot. The information is composed of operational mode information and GPS data. Operational Mode: For the most part, it is the navigation computer that decides and sets which operational mode the autopilot is in. This communication is handled efficiently through the UART connection between the microcontrollers, allowing the autopilot computer to respond quickly to changing flight path requirements. The updates are sent every time there is a change to the mode as well as every few seconds during the flight as a failsafe in case the autopilot computer undergoes an unexpected reset during flight.

GPS data: GPS data is sent to the autopilot computer once every second. The one second interval is the optimal delivery interval given the 1 Hz update rate. Each update includes the current heading obtained directly from the GPS, the current heading to the landing site, and the current GPS ground speed. All of this information is used by the autopilot code to accurately determine its attitude and flight path to the landing site.

6.5. Autopilot Code

The development, testing and recoding of the autopilot code was the most intensive of the three code parts. The general functions of the autopilot are:

- Drive actuation of control surfaces from sensor data
- Use of IMU and GPS data to fly vehicle while following a given heading

The implementation level of these two functions, including the strict requirements for speed, RAM, and code space would require replacement of the original processor, an 8MHz

ATmega 328p. By replacing the original processor with a 72MHz, 32-bit Cortex-M4, the autopilot code was able to perform the functions and run at an acceptable speed.

6.5.1. Autopilot Functional Bounding Considerations

The autopilot should update its calculations as frequently as possible, but it is unnecessary to update faster than 50Hz This consideration is important given the "real-time" operation required to fly an aircraft. Faster updates mean faster responses, this leads to better damping of external forces on the vehicle and better prevention of undesirable flight conditions such as stalls or spins. This also leads to much smoother flight which in turn results in less loading on the airframe and a more efficient flight. Overall the craft behaves more reliably and safely if the updates are performed at a faster rate. The increasing advantage of the faster calculations continues to improve almost indefinitely in theory, but the design of the servos almost completely halt any improvements from update speed at frequencies faster than 50Hz. Almost all stock RC vehicle servos use a 50Hz PWM frequency as the control signal. A diagram of the signal is shown below.

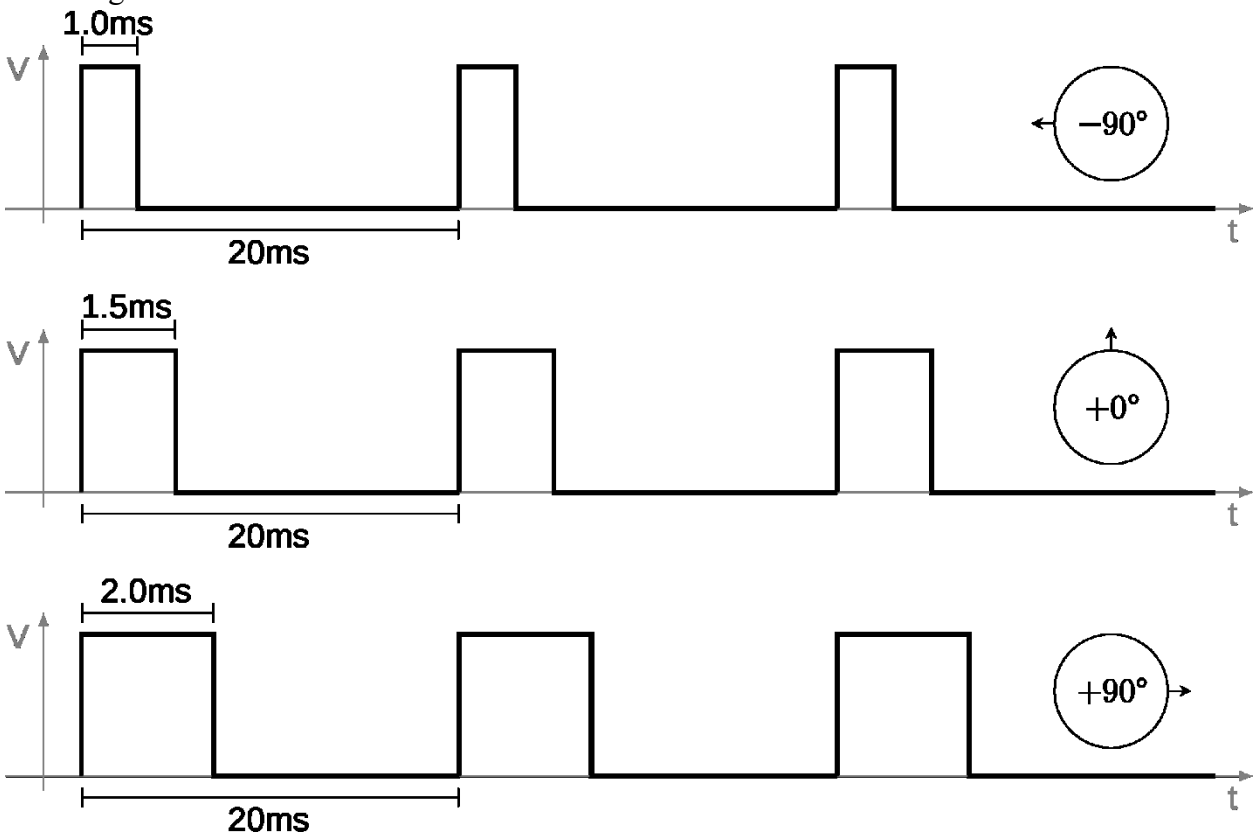

Figure 67: Diagram showing timing of servo PWM [68]

Designing new servos capable of faster operation was well beyond the scope of this project; and as such, this was a hardware limitation that had to be worked with. All of these factors played into the consideration that faster is better, with the caveat that 50Hz would be the upper limit on useful frequency of updates.

The software should not cause or maintain any oscillating motions. Any externally induced oscillations should be damped by the software. This is important for a number of reasons.

1) Amplified oscillation: any situation where an oscillating motion is amplified by the control software is extremely dangerous. This can rapidly progress to the point where the flight is no longer controllable, eventually leading to the overstress and destruction of the airframe. This case is easily preventable by using a PID controller with an initially conservative P value that is then fine tuned based on data from flights. "Reasonable" values are easily determined with calculations based on basic understanding of the system being controlled. For this project "reasonable" values involved finding the proportional relationship between a reasonable range of control surface deflection (+- 10 degrees) and the expected range of the input (for example, airspeed of +-5 mph). This example suggests a P coefficient of two since the desired output range is twice the size of the input range.

2) Sustained oscillations: Though not as immediately harmful as amplifying oscillations, sustained oscillations can, over a longer period of time, cause the exact same problems if they are not corrected. Sustained oscillations can be more difficult to detect with just visual observation of a flight, but shows up well on a collected flight data graph. It is much easier to see this data on a graph

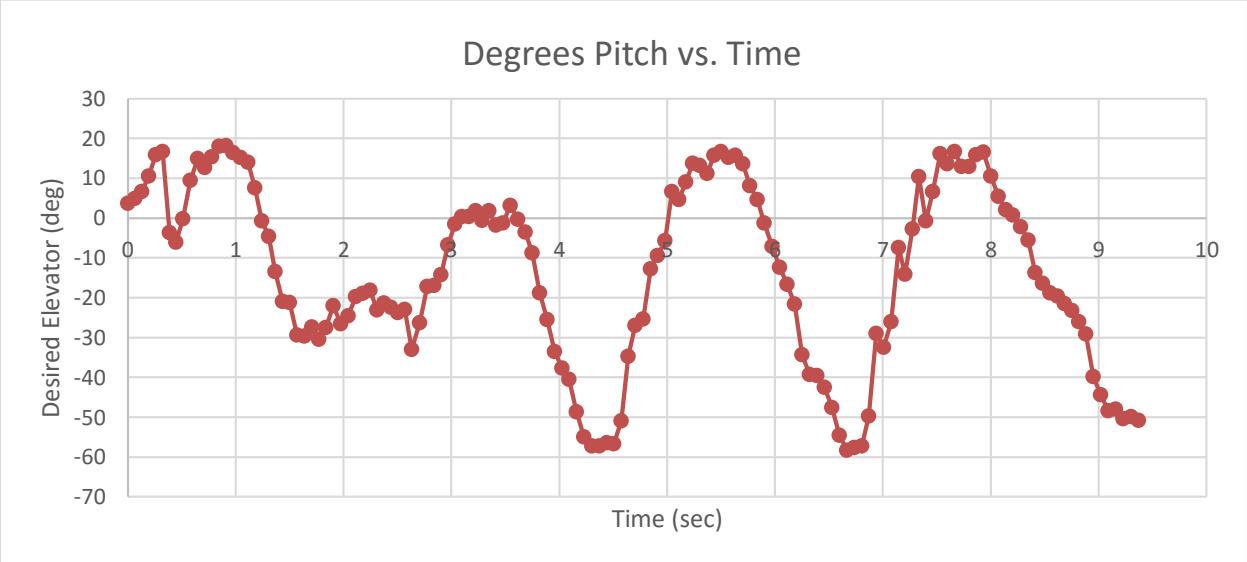

Figure 68: Graph showing sustained pitch oscillations

This graph shows an example of a sustained oscillation on one particular AURORA test flight. There is a strong oscillation on the pitch axis. Additional tuning of the PID gains as well as adding a separate damping PID controller (discussed in detail later) ultimately fixed this problem.

3) Externally induced oscillations: The glider systems must be able to compensate in the event that some external force is able to overcome the stability of the control software. Long term oscillations are a serious problem for any aircraft, causing major stress fatigue on the airframe, and forcing the control software to work overtime, sometimes to no avail. Ensuring

that any oscillations that do start are quickly damped is crucial to ensuring a successful flight. The best solution to preventing oscillations is to have a means to "break-up" any oscillations as soon as they start. One common implementation of this is to use a second control algorithm that works against and out of phase with the primary control algorithm. This causes a non-periodic shift in the output signal preventing any oscillations from forming.

The software should do everything possible to maintain a stable flight condition and act to avoid potentially unrecoverable flight conditions. This is especially true for flying wings. There are many flight conditions, steep spirals, stalls, spins, etc, that are remarkably complex and even skilled human pilots can struggle to recover from these situations. This issue is further compounded by the fact that a flying wing is inherently more difficult to manage in these situations and can be far more difficult to recover. Although it would be convenient to have autopilot code that works with every possible situation and can recover from all of these, this is far beyond the scope of this project. Instead, this autopilot is designed to be very stable and leave wide margins around the flight regimes that can lead to unrecoverable situations. This prevents any accidental entry into unrecoverable flight conditions. Of course, it is possible that environmental factors, such as strong wind gusts could force the craft into one of these conditions; but with proper tuning and decent update frequency, the craft should be able to react in time to avoid the vast majority of these sorts of situations.

6.5.2. Autopilot Code Development Process

The first iterations of the autopilot code were lightweight, only implementing the basic functionality required for straight and level flight. As development progressed the bare flight code would be complemented by the additional code needed to integrate the autopilot code with the other AURORA systems.

The first few attempts at controlled autonomous flight were mostly hampered by a lack of rigidity in the wing spars. These flights did little to validate the efficacy of the autopilot code as the airframe nearly shook itself apart in the process. Once the flutter issue was resolved, code testing began.

6.5.3. A Note about IMU Orientation

Based on the extensive research into autopilot control algorithms**,** there an overall consensus about which axis should be in which direction; however, the space and mounting constraints in the AURORA design dictated that the axes had to be aligned differently from that convention. The orientation used is**:**

- Positive X axis points "left"
- Positive Y axis points "backward"
- Positive Z axis points "down"
- Positive Pitch rotation (about X-Axis) is nose down
- Positive Roll rotation (about Y-Axis) is left wing down
- Positive Yaw rotation (about Z-axis) is top-down counter-clockwise

All examples given below will use this setup unless otherwise stated.

6.5.4. Basic Control Algorithms: Servo Mixing

Conventional, “tailed” aircraft have separate control surfaces for pitch and roll: the elevators and ailerons, respectively. This separation of functions makes for a simple realization of control algorithms that determine control values for pitch and roll separately. With a flying wing this is not possible because the functions of ailerons and elevators are combined into a surface called “elevons”. Fortunately, the process for combining these two separate controls is simple; they are added together. For example, consider a conventional aircraft where the elevator is deflected 10 degrees up, the right aileron 5 degrees down, and the left aileron 5 degrees up. If this same control surface configuration were applied to a craft with elevons, the right elevon would be deflected 5 degrees up (10-5=5) and the left elevon would be deflected 15 degrees up (10+5=15).

Limitations of elevons: The only problem with elevons comes when the control surfaces are near their maximum deflection. Because the two virtual control surfaces are summed together, there is a possibility for a much greater total deflection than the system was designed for. In the example above, assume that the all control surfaces (on both crafts) are limited to +-12 degrees of deflection. On the conventional aircraft, this is no problem, both control surfaces are within the limits. However, on the craft with elevons, there is a problem because the left elevon is requested to go to 15 degrees of deflection.

In practice, this is less of a problem and more of a point of consideration. Solutions to this problem are simple and straightforward. For AURORA the chosen solution was to limit the final outputs to avoid the worst case scenario of a servo crashing, but the primary defense against going too far is being conscientious of the expected output ranges from the various control loops. This section of code persisted with minimal modifications from the initial tests until the final design.

The final axis of control is yaw. AURORA has only passive yaw control, thus yaw is only discussed as a property of the vehicle, and not a direct axis of control. In this paper, references to “yaw” refer to the relative movement about the vertical axis, while “heading” refers to the absolute position and movement about the vertical axis. For instance, if the vehicle yaws to the left, the heading value will decrease.

An important note on the servo mixing topic: On the physical glider the roles of the ailerons and the elevator are combined onto a single set of control surfaces called elevons. However, in this paper the ailerons and elevator are discussed separately with the implicit understanding that they are combined together on the physical aircraft. This is done to clarify the discussion about pitch and roll control.

6.5.5. The First Autopilot: Accelerometers Only

This first version of the code was simple, using accelerometer data exclusively to calculate the pitch and roll values of the vehicle. An accelerometer measures acceleration. The accelerometer used in AURORA has three separate accelerometers, each measuring acceleration for a single axis. These sensors are mounted orthogonally so that together they provide the acceleration data for all three spatial axes. When the accelerometers are at rest on Earth, they can measure the acceleration due to Earth's gravity. Because Earth's gravity is always straight

down, it can be used as a reference for determining the orientation of the craft. In the simplest form this involves taking the arctangent of the acceleration axis in the plane of rotation divided by the vector sum of the other two acceleration axes. This is the most fundamental method of using accelerometers to determine orientation. All methods for determining orientation from accelerometers utilize this equation in some form.

The pitch and roll values calculated from this data were then fed into PID controllers. The PID controllers for this project used Brett Beauregard's PID library [69]. This was chosen because of its proven reliability, excellent design, and easy implementation. Most importantly, it made implementing numerous PID controllers as easy as implementing the first one. For this first autopilot only two PID controllers were needed. The first controlled the elevator position based on pitch, and the second controlled the aileron position based on roll.

The pitch PID loop took in the pitch value calculated by the method described above, and was given a fixed set point, something negative, usually around -10 degrees, and output an elevator deflection based on the deviation from the set point. The roll PID performed the same function, only with roll as the input, the set point set to zero, and aileron deflection as the output. This setup was never intended to be the final configuration due to sensor noise issues that will be discussed later, but would be the fastest way to test autonomous flight. The first few flights were erratic and disappointing. It was impossible to determine if the autopilot worked due to the intense oscillations caused by the flexible bamboo wing spars. Continued tests were attempted, but to no avail. Eventually the oscillations became so intense that one of the wings tore off in flight. This stopped all future flight testing until a more rigid spar could be built or purchased. Despite this, static testing of the software revealed spasmodic control surfaces were a serious concern. Even the slightest accelerations and vibrations would cause the control surfaces to jump around wildly. This was most clearly demonstrated when the glider was allowed to settle down perfectly still on a table, and then gently touched. Even the slightest tap would cause violent actuations of the control surfaces. This led to the development of the next major revision of the autopilot.

6.5.6. Autopilot Version 2: Accelerometers, Gyros, and Kalman Filters

The two main sensor types used for determining the attitude of small UAVs such as AURORA are accelerometers and gyros. Prior experience with these sensors and their behavior was demonstrated clearly in the initial autopilot. From the beginning, there was an understanding that the pure accelerometer code would not work well due to the nature of the sensors themselves. Leaving the accelerometer data out is not realistic either because it is absolute. Without absolute data it would be impossible to use only gyros and account for drift, making any flight more than a few minutes long impossible. Moreover, the lack of an absolute value for attitude would require the craft to be oriented exactly the same at every startup or have the orientation input manually. These factors make leaving out the accelerometer altogether a prohibitively difficult task. It is possible that the accelerometer could be used alone by using some sort or rolling average or weighted rolling average, but this will reduce response time and is it far from the best solution, especially because it does not take advantage of the gyros at all. The table below shows the practical differences, advantages, and disadvantages for gyros and

accelerometers. For rows with a yes or no response, the yes response is an advantage, no is a disadvantage.

A Comparison between Accelerometers and Gyros		
Parameter	Accelerometers	Gyros
Data Provided	Acceleration	Rate of Rotation
Conversion to Roll/Pitch	Arctangent of orthogonal axes	Integration of each axis
Is absolute?	Yes	No
Short term stable?	No	Yes
Long term stable/Has Drift?	Yes	No
Unaffected by outside forces?	No	Yes

Table 7: A Comparison between Accelerometers and Gyros

There are many different ways of using this information to improve the autopilot. Based on the data in the table above it easy to see that this is an excellent opportunity for sensor fusion; the advantages of the accelerometer perfectly compliment the disadvantages of the gyros and vice versa. Research on how to combine the data from these two sensors resulted in two viable methods. The first and simpler method, is to use a complementary filter, which requires that the angular measurement for both sensors be calculated and fed into a weighted average between the two values.

The second and more complex option is to use a Kalman Filter. A Kalman Filter has elements of many different filters such as the weighted rolling average, weighted average on the current values and many more. An Arduino library was found that uses a Kalman filter to synthesize gyro and accelerometer data to get an accurate and steady pitch and roll angle [70]. This was quickly implemented in the autopilot code. Static test results proved the Kalman Filter worked extremely well keeping stable roll and pitch values that did not accumulate drift. Unfortunately, the flight tests did not seem to show significant improvement as was expected. The roll certainly seemed improved, but the glide ratio was still atrocious, and the glider fell more than it flew.

6.5.7. Airspeed based Control and SD Data Logging

The use of an airspeed sensor (using a pitot tube and differential pressure sensor) was discussed, but had been avoided due to the high cost of differential pressure sensors as well as the difficulty in sourcing one that could meet all of the specifications for this project. However, with every single flight test up this point attaining an atrocious glide ratio of only 2-3:1 and with no way to know the speed of the craft up to this point; there was a growing concern on the team that the craft was never attaining flight speed, and could never do so due to the control algorithm

locking the aircraft to a specific pitch. Therefore the decision was made to purchase a differential pressure sensor and a pitot tube. Code was prepared so that the sensor could be tested and installed immediately on arrival.

Figure 69: Labeled picture of Pitot Tube

The design of the Pitot tube includes both static ports and a pitot port. The pitot port is at the front and exposes the sensor to the stagnation pressure of the air, that is, the pressure applied by moving air as it is brought to a stop against the diaphragm in the sensor. The other side of the sensor is attached to the static ports on the Pitot tube. These are holes along the side of the Pitot tube designed to measure the static pressure of the air, which is the atmospheric pressure. The difference in these two pressures, known as the dynamic pressure, can be used in Bernoulli's formula to find the speed of the air going into the Pitot tube. This airspeed can then be used as the only means for elevator control on the aircraft. To implement this new means for control, the PID controller that originally used pitch as the input and an angle as the set point now used airspeed as the input and a specific airspeed as the set point. In the code the airspeed is always in miles per hour (mph) because the project team can most easily relate to mph as a unit. Any raw data and graphs in this paper will have the airspeed in mph, but for continuity with the rest of the paper text speed references will be in m/s.

Based on the early tests in RealFlight G7.5 the best glide speed was determined to be around 8 m/s and the stall speed is around 6.7 m/s. For the early tests where the capabilities of the airspeed control were unknown the threshold of only 1.3 m/s above stall speed was deemed too small. To give a better margin of 3.3 m/s, a speed of 10 m/s was chosen for the set point.

Another upgrade was made prior to continuing test flights. This was the addition of an SD card slot for data logging. The SD card slot on the board is connected to the navigation computer, leaving the autopilot with no built in logging option. With all of the problems being encountered with the autopilot, it was realized that the ability to log data from the autopilot would be very valuable for testing purposes. The setup of the SD card proved very time consuming due to a number of small faults that were very difficult to debug. The Arduino SD card library also caused trouble. The SD library is so large and used so much RAM that it could not be used on the 328p alongside the autopilot code. Research turned up an alternative library that could only access FAT16 SD cards, 3GB or smaller, which was no problem other than that obtaining an SD card that small was actually a bit of a challenge at first. The code to log data to the SD card was rewritten to use the smaller library, and this solved the space and RAM usage problems. The SD card would prove troublesome time and again throughout the project, but the data it provided was extremely valuable. Almost every single improvement to AURORA that

occurred after this point in development relied on data logged to the SD card. Future changes to the SD setup were not done for performance enhancement, but merely to ensure continued functionality, and as such they are documented in the paper separately. The SD card did undergo significant changes when the switch to the flight controller board was made and when the Teensy was installed. The specific values logged changed frequently as the importance of certain parameters became more or less relevant to the current flight testing. Once the flight testing was completed, the SD card would no longer be needed for the autopilot computer, and it would be disabled with a #define and then the physical card and holder could be removed.

6.5.8. Continued Glide Ratio Problems

Despite all of the software changes to try and keep the glider from plummeting to the ground after every launch, problems persisted that prevented any flights from making any reasonable progress. Even after changing the PID coefficients numerous times, it seemed as though there was no way to get and decent glide ratio out of the glider. With no explanation for the current failures, much less a solution for them, the RealFlight simulator was consulted. The goal of this was to determine what, if any, differences existed between the real craft and the simulated one. The simulated model had demonstrated excellent flight performance which was not at all close to the performance being seen of the actual model. A couple of key differences between the models were noted: First, the CG was much farther back on the simulation model; second, the control surface deflection in the simulator was far less than on the real model. The CG discrepancy is discussed in great detail in the Airframe development section, but in short, the CG was determined to be completely wrong on the actual craft. The vastly smaller control surface deflection on the simulator model opened up the question of how the deflection amount will affect glide ratio. The simulator made it easy to perform a series of tests at different deflections to see how that affected glide ratio. The simulator model was set up using a yaw stabilizer to ensure a straight flight path, and winds were turned off. The elevator deflection was controlled using the trim tab so that the deflection would remain perfectly constant. Data was gathered in 31 tests at differing elevator deflections. The deflections ranged from 2.4 to 27.9 degrees of upward deflection. The graphs below show the results of those tests.

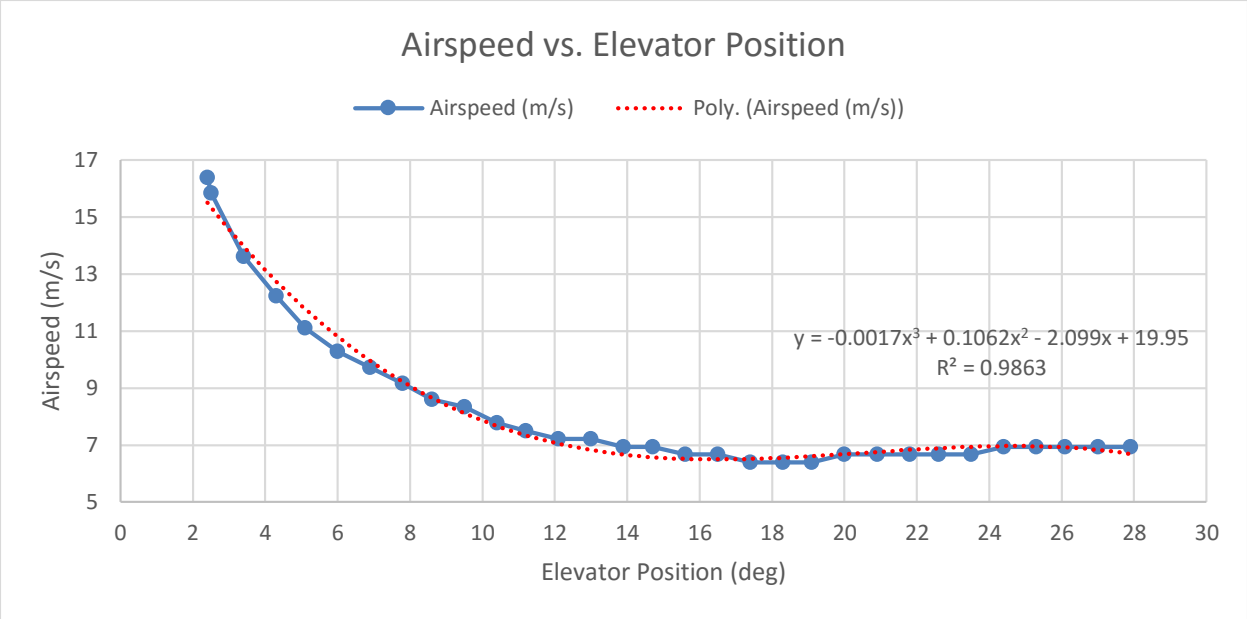

Figure 70: Graph of Airspeed vs. Elevator Position from RealFlight

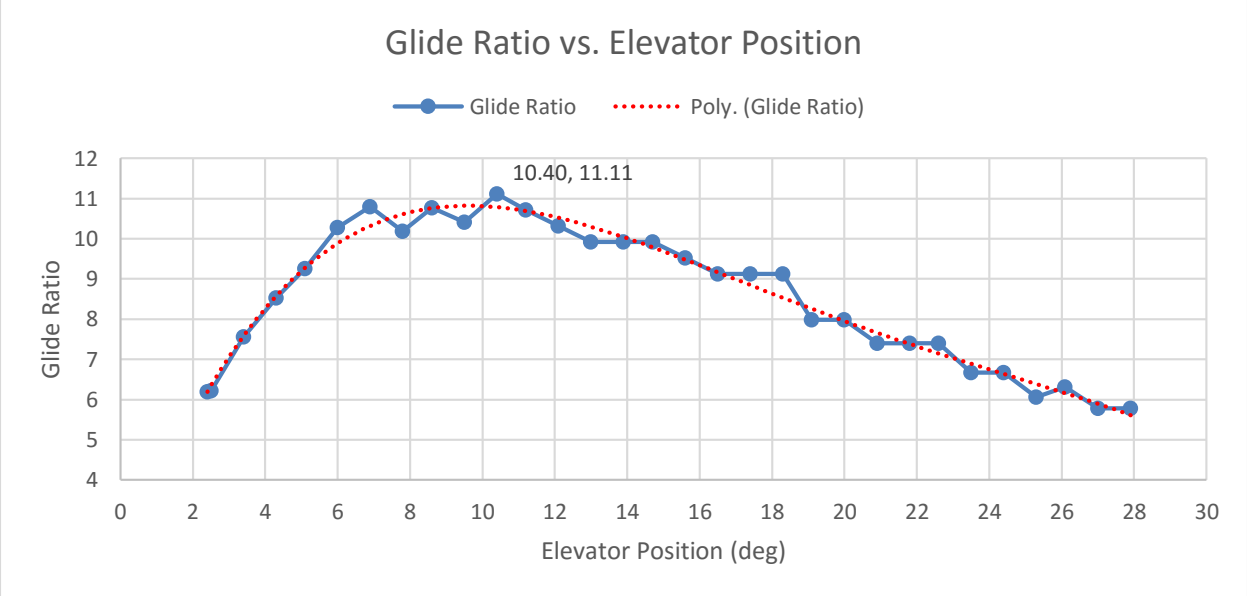

Figure 71: Graph of Glide Ratio vs. Elevator Position from Realflight

This data clearly showed that deflections greater than roughly 10 degrees cause an increasing drop in glide ratio. AURORA was configured so that it could easily deflect the elevators and ailerons to their full mechanical deflection of nearly 60 degrees. Deflections in the range of 30-40 degrees were normal occurrences. This data demonstrated that the high deflections being used in the autopilot were too extreme and needed to be severely toned down. This led to a concern that if the controls were tuned down significantly, the craft could not respond quickly enough in a very steep dive and recover. To alleviate this concern multiple control laws were implemented. When the craft is flying normally, the controls are very light, but if the pitch value drops below 30 degrees, the control authority escalates and increases the

range of the controls. If the pitch value drops beyond 60 degrees, a third control law is activated which provides the maximum amount of control range.

6.5.9. Obtaining Clean, Fast Airspeed Acceleration with Min-Max Filtering

The original, and most important, purpose of the airspeed acceleration data is to provide a means to get the aircraft out of a steep dive as quickly as possible, and to do this without relying on pitch data. The second role the airspeed acceleration data plays is too damp out any porpoising that develops as a result of the delay in the airspeed PID controller. Unfortunately the long-time-interval derivative combined with multiple averaging steps results in a time delay that causes the airspeed damping PID to amplify porpoising instead of attenuating it. This effect appears exaggerated when serving as the means to recover from a dive because the averaging suppresses the sudden increase in acceleration for some time, reducing the magnitude and speed of the control response. Based on the flight data obtained from the last few tests, the graphs showed that for a slope direction change of the airspeed (up → down or down → up), the current filters required at worst between .25 and .3 seconds before the calculated airspeed acceleration would show a sign change(+ → - or - → +), and since this is a gradual change it takes still longer before there is enough of an output to have any effect on the control surface position. It was also possible to come up with a rough idea of what the maximum transient response time should be for seeing a sign change in the acceleration after the slope direction change of the airspeed. Based on visual analysis of how the airspeed responds to control inputs and how the porpoising develops, it was determined that the maximum transient response time should be less than .1 seconds.

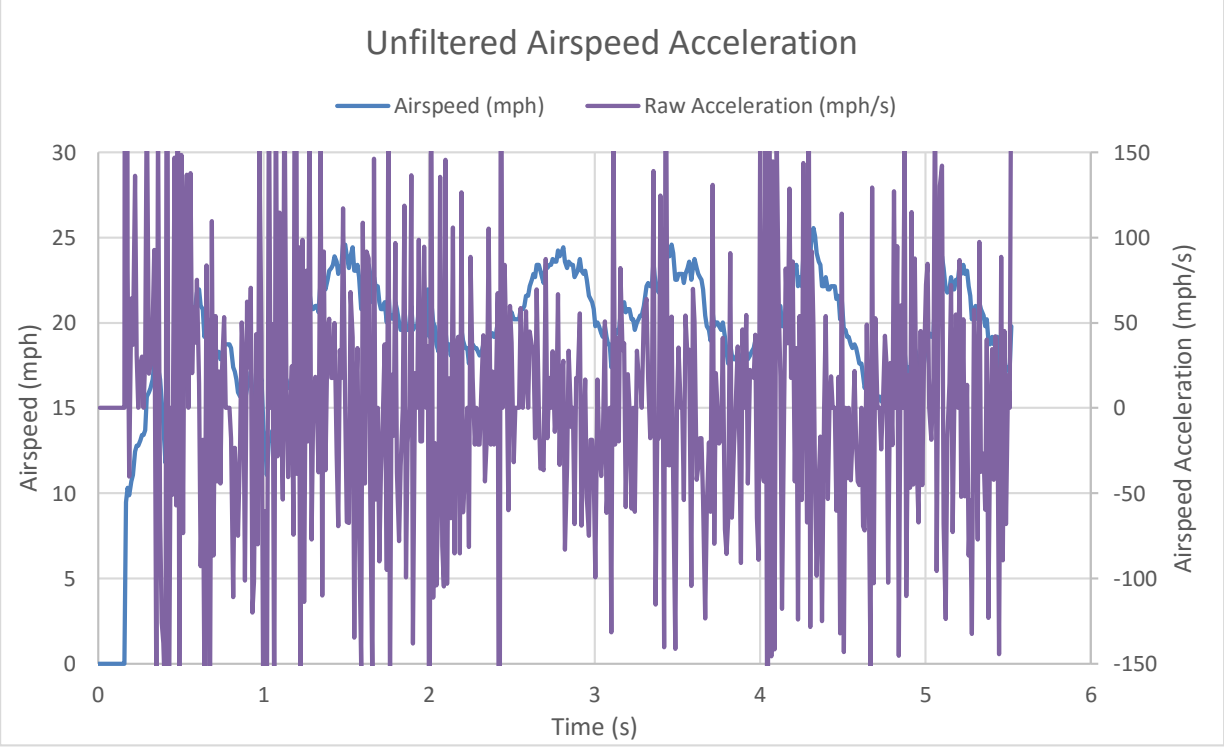

Figure 72: Graph of Raw Airspeed Acceleration without Filtering

The original filtering method for the airspeed acceleration using multiple stages of averaging produced a clean data stream which contained reasonably accurate acceleration data, but that averaging introduced significant time delays in the response loop. A new filtering solution is needed that can take in very noisy data, filter it, and do so without compromising the transient response time. The development of the new filter was completed using the data from test flights using airspeed acceleration, Excel, and MATLAB. This way the new filters can be compared to the old both in terms of filter quality and transient response time. The development of the new filter began where the old left off: with averages. The old averaging method used limited rolling averages. The new averages were also rolling averages, but they were not time limited. The averages did not store past values, each time a new value became available; it was averaged with the old average. A single rolling average made a substantial improvement over the raw data, but it alone was not enough. The single rolling average was useful for finding out how averaging at different stages of a calculation affects the output. Testing the average at different stages of the calculation showed minimal effect on the output. The averaging can be performed at any point, and it will produce a nearly identical result. From here, the simple rolling average was modified to allow for weighting. Weighted rolling averages look almost identical to simple rolling averages, but when a new average is taken, each component is given a fractional weight such that all the weights add up to one. This technique was found to allow for better control of the output filtering, but once again, no matter how the weights were tuned, getting out clean data with a good time response proved mutually exclusive. The next attempt combined multiple averages together in various ways to filter the most while performing the least amount of averaging. After hundreds of fruitless attempts to find a workable combination, the conclusion was made that the fundamental characteristics of an averaging filter make them unsuitable for this application. For an averaging filter to be able to filter this data effectively, the new data point must be so significantly diluted that it cannot provide a large enough pull to move the entire average when a transient change occurs. This means that once the averaging was sufficient enough to produce a useable signal, the transient response time was .25 seconds, 2.5 times what had been determined as acceptable.

To gain a wider knowledge of the filter options available, significant research was done on the types of filters available that can perform the filtering needed for AURORA. Nothing groundbreaking was discovered, and so it was decided that the best option may be to look at the Kalman filter. The Kalman filter was used to great success on this project for the purpose of combining gyro and accelerometer data to completely attenuate jitteriness in the control surfaces very early on in the project. Additionally, research showed that the Kalman filter was highly praised by many for its ability to clean up terribly messy data with ease. Unfortunately, the Kalman filter library used for the accelerometers and gyros was written very specifically for that purpose, and could not be used as is. Searches for generic Arduino Kalman filter examples turned up no useful results, leaving a custom Kalman filter as the only remaining option. At the time, none of the members of the project team felt comfortable enough with Kalman filters to write the software for one. Several internet tutorials on the workings of Kalman filters were

found and worked through until at least a minimum working knowledge of Kalman filters could be obtained. One tutorial by Greg Czerniak [71] in particular had some Python sample code for a one dimensional Kalman filter. Elements of this code were used as a model for the autopilot code written in Arduino and the testing code written in MATLAB. As a result of this study valuable information about the function and use of Kalman filters was garnered. Particularly prudent to the problem on hand was the information that a Kalman filter is, at its core, a weighted rolling average. This was a definite concern given the prior experience using averages. Despite this, the Kalman filter's ability to dynamically adjust the weighting of the average based on the estimate of the current state could make a huge difference in the output quality. Testing the Kalman filter required tuning the various coefficients until the graphs were showing that they had been fully optimized. Having no experience with this tuning process on Kalman filters meant that for quite a while there was no significant progress made because the coefficient assignments were made based on guesses. Eventually decent starting point numbers were found and the tuning process began. A substantial amount of time went into tuning, including a couple of attempts to find entirely different values that would give a better result. The results showed that the Kalman filter could perform a better filtering job than the averages, but it could not give a much better transient time response. At very best, the Kalman filter was giving a transient time response in the area of .17 seconds.

The testing up to this point had ruled out the two most common methods for filtering. To try and resolve this issue, the decision was made to put serious analytical thought into how the average filter behaves, and how to isolate the desirable behaviors. The average filter was chosen because trying to do this for the Kalman filter would be far more difficult considering the mathematical complexity of the Kalman filter. After deep analysis and considerable scratch work, a concept for how to perform initial data filtering was developed, using a system which should have a much better transient response than the averaging filter. This concept was given the name Progressive Min-Max filter. The logic of this filter is as follows. The reason for the high amount of dilution of the average filter is a result of the input data swinging aggressively from positive to negative values. However, there is some regularity to this swinging because the Kalman filter works reasonably well even with this swinging. This means that the process noise must be at least close to Gaussian in nature and therefore by looking at the true value + maximum noise or the true value + minimum noise will yield a line identical to, but offset from the true value. By finding a recent maximum for each point of data, it should be possible to see this effect.

This is accomplished by using the maximum values from a rolling list of the last ten values. This data is less noisy because the maximum calculation is in a sense averaging on the data by only keeping the maximum value out of the last ten. This will be far more responsive than the normal averaging filter in circumstances when the airspeed goes from decelerating to accelerating. As soon as the airspeed begins to increase, the maximum value will always be the newest value. This can also be applied to the minimums as well, but the response will be immediate when the slope of the airspeed switches to a negative slope. The process noise being

Gaussian means that the amount of error above the actual value should be similar to the error below. By finding the median between these two values, an approximation of the actual value is obtained. This value ought to be more stable because the noise is effectively filtered out by using the maximum and minimum values of the noise and cancelling them out. Thus any noise that occurs between the maximum and the minimum will have no effect on the value

In working on this method of filtering, another realization was had. It would be possible by slightly modifying the current algorithm to get a value proportional to the time based derivative, without using time. Because the max value line follows the highest values, and the min line follows the lowest, the difference between those lines can be used to find a value that is proportional to the time based derivative. This is the result when the slope is negative, the minimum value will tend to follow the value at the very front of the value range, while the maximum will follow the last value. The opposite is true of a positive slope. As the slope increases, the difference between the values at the front and the back of the range will get larger, corresponding to the higher rate of change. Instead of taking the derivative, the difference between the rolling maximum and rolling minimum can be taken to determine a value proportional to the acceleration, and the value returned is a well filtered value for airspeed acceleration. The only issue with this method is that the difference between the max and the min will always be positive, so there must be a calculation to determine the correct sign of the airspeed acceleration. This is done by combining the first method with the second. Because the Progressive Min-Max filter is now applied to the airspeed directly, the median value will be a filtered value for airspeed. The sign for the acceleration is found by getting the sign of the change in airspeed over each time interval. The sign of this change is applied to the difference between the max and min, and this value is used for the airspeed acceleration. The filtering is not quite as good as with the original averaging filter, but it is still much better than the raw data shown above in Figure 72, and furthermore in test this degree of filtering has worked exceedingly well. A comparison between the raw time derivative airspeed acceleration and the Progressive Min-Max filtered value is shown below.

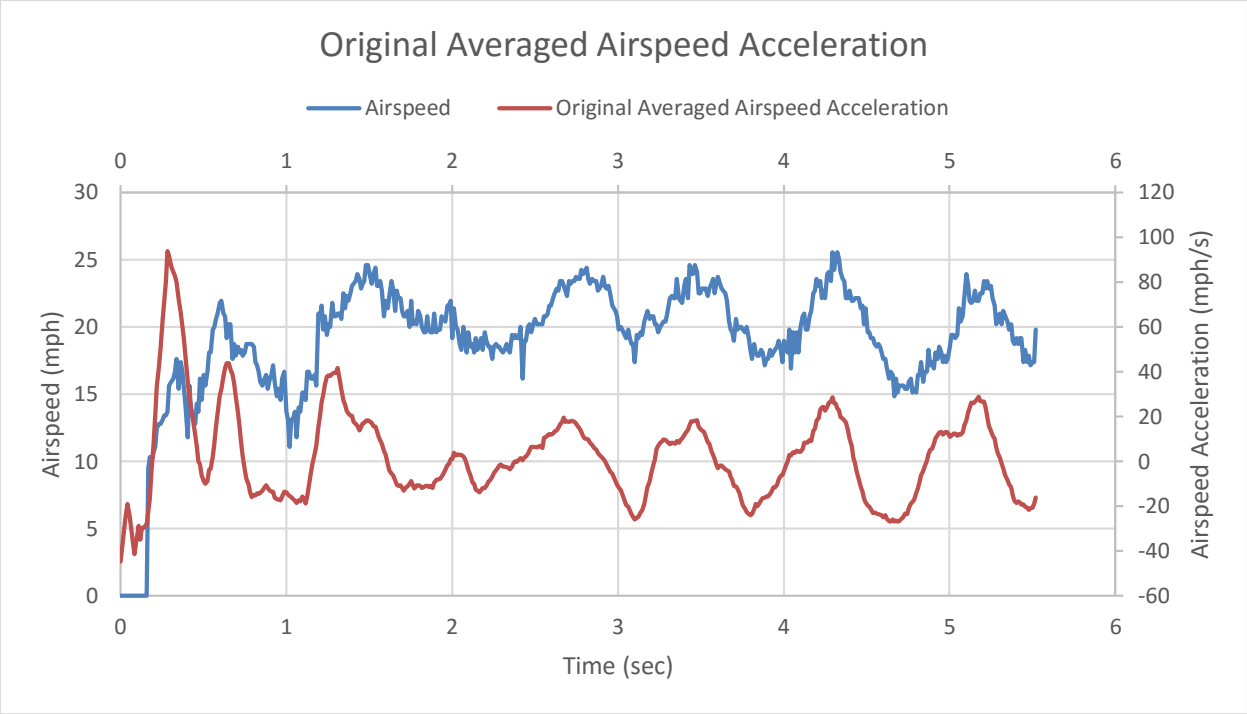

Figure 73: Graph of Original Averaged Airspeed Acceleration

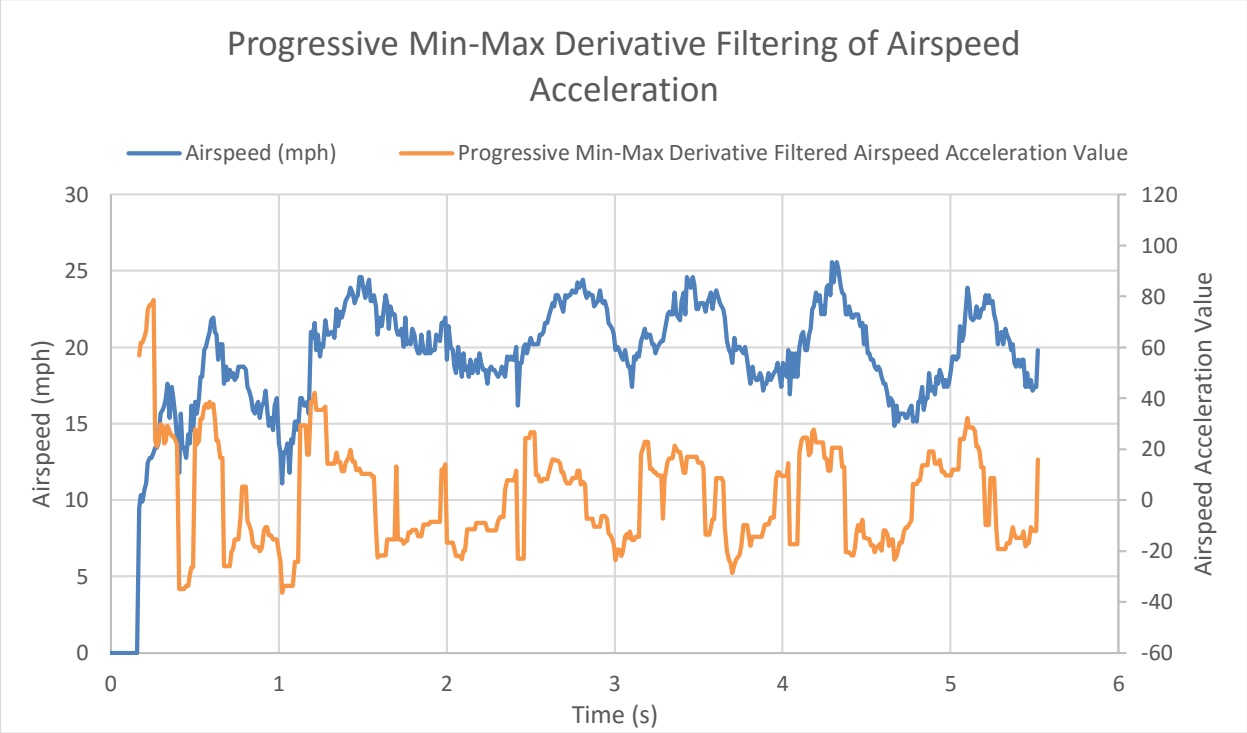

Figure 74: Graph of Progressive Min-Max Filtering of Airspeed Acceleration

Although this work took nearly three full days of trial and error math along with deep analysis of the flight data to complete, the rewards were well worth it. Testing showed that the

glider could easily and quickly recover from steep dives, and the porpoising that had plagued flights from the very start was eliminated.

6.5.10. Issues with Inter-Microcontroller Communications

Very early on in the project it was decided to use UART for the communication between the Navigation Computer and the Autopilot Computer. UART was chosen because it is relatively high speed, easy to use, full duplex, bidirectional, and does not slave one device to the other. Minimal effort was initially placed in researching alternatives because it was felt that this option was by far the best. This assumption was quickly proven to be false as numerous problems arose leading to tens of hours of unexpected debugging. Many of the problems and their solutions were bizarre and have yet to be explained even after consulting both peers and professors. The problems included: character transmission issues, initialization issues, receive buffer overflow issues, and broken receive buffers. Given the initial high expectations for the easy implementation of UART; the numerous hours spent debugging the countless issues made UART communications one of the most frustrating parts of the software development process. One of the most important lessons learned during the process was the need to read from the buffer frequently. Despite the buffer being able to hold 64 characters, it seemed that if the buffer was not read out multiple times a second to a software buffer, the data would get corrupted, and sometimes blocked that UART port from receiving any more data. Specifically, most uses of the standard Arduino method “delay(long msec)” would cause a failure of the UART port. Eventually the UART code was refined to work correctly, and after it functioned near flawlessly and met all of the requirements for data transmission, reliability, and not interrupting or blocking the flow of the rest of the program.

6.5.11. Update to Directional Cosine Matrix

The prior flight tests were mostly complete failures. It appeared that the failure mode had shifted again, indicating that the problem addressed by the prior fixes was well taken, but unfortunately another new failure mode had developed. This time, the failures were able to slip past the airspeed, airspeed acceleration, pitch, pitch damping, roll, roll damping, and heading control algorithms. As with the vast majority of the previous flight failures, this one resulted from an undefined flight condition. The aircraft entered into a very steep spiral and the flight data completely misrepresented the actual condition of the craft, leading the autopilot to believe the craft was flying poorly, but still within reason. The entry into the steep spiral was made possible because the craft began a turn after launch, as well as entering into a dive. The autopilot commanded dive was compounded by the rolling because any airplane will have a tendency to nose down when roll is increased. This compounding created a very rapid pitching down. The rapid pitching down resulted in a situation where the roll and pitch values quickly became unreliable due to the increasing downward acceleration. In addition, strong winds on the nights of these tests likely helped throw the craft into the steep angle of roll. Once the roll angle approached 90 degrees, the chance for pitch and roll recovery was completely lost and the system was left to rely on the airspeed controls to step in as the craft rapidly accelerated and eventually reached unsafe speeds. This was not able to happen due to the steep spiral. The steep spiral resulted in heavy accelerations and induced drag, slowing the dive to the ground. This

prevented the forward acceleration needed to trigger the airspeed emergency overrides; the descent was at a nearly constant speed of roughly 20 mph. And although this lack of downward acceleration should have caused the accelerometers to provide more accurate pitch information, the high G turn produced other accelerations that, when used to calculate pitch angle, resulted in registering a gentle dive.

Given that this sort of problem had happened multiple times before, and had almost always required the development of a complex solution to address only that particular unique condition, the decision was made to not come up with yet another "band-aid" solution, but to try and find either a mechanical alteration to improve stability, or a programmatic way to completely fix the problem. The first step to finding a solution was to better understand the flight dynamics at work that resulted in such an aggressive flipping action. To try and understand some of the flight dynamics at work, a recent graduate of WPI with excellent understanding of Aerospace Engineering was consulted. After less than a hour of conversation, a simple conclusion was reached: If the craft could not be made to accurately measure pitch and roll, there would be no guarantee that the craft could avoid leaving stable flight.

This would require a complete change of the way that the craft calculates pitch and roll. The current pitch and roll calculations were based off of the Kalman filters which were initially added in one of the very early versions of the autopilot, back when the issue of calculating pitch and roll was first investigated. Most of the results of various internet searches reference Kalman filters, and complementary filters. For many of these results, especially those referencing complementary filters, the applications in discussion were ground based robots or quadrotors that would never have to deal with many of the situations AURORA has to deal with on a regular basis. The only piece of open source software, known by the project team, to deal with fixed wing UAVs was the Ardupilot project. The source code for the Ardupilot had already been obtained for the purpose of study some time ago, but the code was so difficult to comprehend, the endeavor had been dropped. While attempted numerous searches with different queries in the hopes of turning up something slightly different, the most important software discovery of the entire project was made.

In one search a project called the "ArduIMU" was uncovered. The ArduIMU was developed by the team at DIYDrones, the same who developed the Ardupilot. The ArduIMU is not in itself an autopilot, but it does perform much of the math on an ATmega 328p to get values such as pitch and roll. Although it appears that the project had died in 2014, all of their source code still existed and was easily available online. Fortunately, the source code for the ArduIMU is far shorter and less complicated than the source code for Ardupilot. It was immediately obvious that they were using a completely novel method for calculating pitch and roll. A brief search through the Google code site for the ArduIMU turned up a "Theory" Wiki page which linked to a number of papers that describe the exact method they were using to calculate the attitude of the aircraft [72]. This method is called the "Directional Cosine Matrix". It is likely this method was not discovered earlier due to the fact that it is somewhat new and not used nearly as often as other methods. As with the Kalman filter discussed earlier the Directional

Cosine Matrix is a very complicated mathematical work and will not be discussed in mathematical detail in this paper. Instead a brief discussion of why it works and how it improves upon the Kalman filter for navigational functions will be discussed here. The Directional Cosine Matrix improves immensely on the Kalman filter by being far less indiscriminate with how it uses data from the sensors. While the Kalman filter can be slightly tuned to prefer one sensor over another, the Directional Cosine Matrix is capable of specifically picking out the best features of each sensor and avoiding their pitfalls. The Directional Cosine Matrix starts with the gyro data, which it uses as the primary source of directional information. As was discussed earlier, the gyros need to be corrected by something else to ensure they do not drift too much. To correct the gyros, the Directional Cosine Matrix uses the accelerometer data. However, to avoid correcting the gyros with false accelerometer data, the accelerometers are corrected as well using GPS and airspeed to eliminate a few external sources of accelerations. The gyros are not corrected directly by the accelerometer; however, the acceleration data and gyro data is fed into a PI controller which is then used to correct the gyro values. In this way, the I value in the PI controller is able to "learn" the drift bias of the gyro over time, making the system increasingly accurate during the flight [72]. The effects of this more careful use of sensor data are remarkable. The glider behaved completely differently, and all of its former bad behaviors were seen to completely disappear. The Directional Cosine Matrix method of determining the attitude of the aircraft finally solved a problem that had been plaguing AURORA from almost the very beginning.

6.6. Final Software Design and Results

Though the radiosonde and navigation code did not require much development, they did receive a large amount of testing. All of the functionality of that software was verified to work long before the autopilot code was ready. Although the autopilot took much longer to be fully developed, it was fully tested, and is ready for more full scale testing. Every aspect of the software has been verified to work, and the modes are able to switch between one another correctly.

6.6.1. Final Autopilot Design Overview

The final design for the autopilot was by far the best and most reliable design up to that point. It was the first one that had serious potential to not suffer any flight control errors. This was primarily due to the addition of the Directional Cosine Matrix. The controls for each of the three axes are described below.

- Roll: Roll control is derived from two PID controllers. The first one is used to determine the aileron position based on a desired value coming from the heading control PID. The second PID uses the roll gyro to damp the motion in roll. The value for the actual amount of roll comes from the Directional Cosine Matrix using the gyros and accelerometer.
- Pitch: Pitch control is derived from two PID controllers. The first one is used to control the pitch from the set airspeed. The second PID controller uses the

airspeed acceleration data to damp change in airspeed by adjusting pitch. The airspeed acceleration data is derived from the airspeed data after being fed through the Progressive Min-Max Derivative filter. Additionally, the pitch control is adjusted by the roll of the aircraft to ensure the nose doesn't drop too much when turning.

- Yaw/Heading: This craft does not have active yaw control, therefore the Heading PID is used to direct the roll controls so that a heading can be turned to and followed. The heading data is derived from the gyros and GPS heading after those are fed through the Directional Cosine Matrix.

6.6.2 Final Navigation Code Design Overview

The navigation code was in a fully tested and complete state for over a month before it had its first real flight test. The hours spent simulating the code, and then running mock data to it had certainly paid off because it worked on the very first attempt. The navigation code demonstrated that it was able to read in a list of landing sites, parse GPS data, and the use that data and the landing site info to select the closest landing site. It then uses that information to calculate a heading to the landing site and transmits that information along with the GPS heading and speed to the autopilot. The navigation code was also able to effectively switch between modes. Though never tested in flight, ground tests with fake GPS data showed the navigation code switching into spiral descent mode, then landing mode, and even back out again if the situation arose. The navigation code was able to meet all of the requirements set for it and passed tests that verified that functionality.

6.6.3 Final Radiosonde Code Design Overview

The radiosonde code was by far the shortest and simplest code of the three, but it is still crucial, because it is the core reason for the rest of the project to exist. This code was tested and verified very early on in the project to send weather data via the APRS network. This test confirmed the efficacy of the radiosonde code. Other tests using the SD card confirmed that it can be written to and read from.

6.6.4 Software Conclusions

The section of code which required the most effort, the autopilot, was initially thought to be fairly straightforward, and that it would be the navigation code that would be the most difficult. Obviously this was not the case, and it was the autopilot that would require up until the last possible moment to debug and recode. A tremendous amount of research and analysis went into creating the final editions of the AURORA software, and it paid off because in the end, all of the code did work.

7. Conclusion

The goal of this project was to develop a radiosonde-glider capable of autonomously navigating to a safe recovery location for reuse. Over the course of three short academic terms, the project went from concept to reality, challenging the team's skills at every step along the way.

Almost every criteria that determine the success of the project were met or exceeded in a design that is competitive with modern solutions. The Flight Controller was developed and is more capable than any radiosonde already on the market. It was designed from the ground up to operate in the conditions expected at the edge of space. It has proven time and again that it can fly AURORA.

The software written to run on the Flight Controller has demonstrated time and again that it can both fly AURORA as well as select a landing site from a preloaded list and initiate the process of navigating to it.

An airframe was developed which was proven to be able to land in a condition not requiring extensive repairs or maintenance. Although the glider never had a flight which covered the full distance of 304.8 x 2133.6 meters, the glider did demonstrate that it could sustain a glide ratio of 8.8:1 for greater than 10 seconds at a time. This clearly surpasses the glide ratio for a 304.8 x 2133.6 meters flight which is only 7:1. The glider was made of materials very similar to those found on current radiosondes and, at 367 grams, remained under the set mass limit of 500 grams. This configuration will be easily consumed by a jet engine should such a collision occur. The glider demonstrated the ability to reliably separate from the balloon and was tested thoroughly for ruggedness and for the ability to fly smoothly.

The entire unit cost of $425.81 is well under the limit of $580, making this device a viable contender with other non-recoverable systems. Additionally the need for only one balloon for a launch and no special equipment means that the launch cost will be the same as current radiosondes. Project AURORA was also characterized in a brief in a CONOPS developed at the start.

The project did fall short of the more ambitious test that were laid out at the beginning, a drop test from at least 1000ft. This test could not be completed due to time constraints and a lack of adequate space and resources. In the end, there was nothing the 1000ft test would prove that was not proven in the final 200ft tests that were conducted. Moving forward, the glider should be tested at higher altitudes and eventually be put through the full 100,000ft flight.

The project has great potential to change the way weather data is collected in the future. A unique and often overlooked problem is addressed by an elegant and affordable solution. Higher recover and reuse rates will save millions of dollars every year and allow for more data to be collected with more specialized sensors. This radiosonde-glider is a proof of concept which only needs a little more development to become a fully realized tool in the Earth science community.

Works Cited

[1] "NOAA National Weather Service Radiosonde Observations," [Online]. Available: http://www.us.nws.noaa.gov/factsheet.htm.

[2] B. Jones, Interviewee, *Intermet Systems Employee.* [Interview]. 9 November 2015.

[3] "NOAA National Weather Service Radiosonde Observations," [Online]. Available: http://www.ua.nws.noaa.gov/factsheet.htm . [Accessed 15 November 2015].

[4] B. Jones, Interviewee, *Intermet Employee.* [Interview]. 9 November 2.15.

[5] N. P. S. R. B. G. B. M. P. B. V. K. M. M. Venkat Ratnam, "Assessment of GPS Radiosonde Descent Data," 2014. [Online]. Available: http://www.atmos-meas-tech.net/7/1011/2014/amt-7-1011-2014.pdf.

[6] "Radiosonde Information," 3 November 2009. [Online]. Available: http://www.nws.noaa.gov/ops2/ua/radiosonde/.

[7] "Plot sounding from RAP/RUC Analyses/Forecasts," [Online]. Available: http://www-frd.fsl.noaa.gov/mab/soundings. [Accessed 16 Novmeber 2015].

[8] "How to Read a Skew-T," [Online]. Available: http://www.atmos.millersville.edu/~lead/SkewT_HowTo.html . [Accessed 16 November 2015].

[9] "U.S. Standard Atmosphere," [Online]. Available: http://www.engineeringtoolbox.com/standard-atmosphere-d_604.html.. [Accessed 16 November 2015].

[10] "U.S. Standard Atmosphere," [Online]. Available: http://www.engineeringtoolbox.com/standard-atmosphere-d_604.html. [Accessed October 2015].

[11] "OSH Park," [Online]. Available: http://www.oshpark.com. [Accessed November 2015].

[12] "Fritzing," [Online]. Available: http://fritzing.org. [Accessed November 2015].

[13] "DipTrace," [Online]. Available: http://http://www.diptrace.com/. [Accessed November 2015].

[14] "SparkFun," [Online]. Available: https://www.sparkfun.com/. [Accessed December 2015].

[15] "CadSoft EAGLE Freeware," [Online]. Available: http://www.cadsoftusa.com/download-eagle/freeware/. [Accessed December 2015].

[16] Atmel, "ATmega48A/PA/88A/PA/168A/PA/328/P," November 2015. [Online]. Available: http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf.

[17] Atmel, "Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V," February 2014. [Online]. Available: http://www.atmel.com/images/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf.

[18] Analog Devices, "Data Sheet ADXL345," 2015. [Online]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf.

[19] ST, "LIS331HH," October 2009. [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Accelerometer/LIS331HH.pdf.

[20] InvenSense, "ITG-3200 Product Specification Revision 1.4," 30 March 2010. [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf.

[21] freescale Xtrinsic, "Xtrinsic MAG3110 Three-Axis, Digital," February 2013. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Magneto/MAG3110_v9.2.pdf.

[22] ST, "LSM9DS1," March 2015. [Online]. Available: http://www2.st.com/content/ccc/resource/technical/document/datasheet/1e/3f/2a/d6/25/eb/48/46/DM00103319.pdf/files/DM00103319.pdf/jcr:content/translations/en.DM00103319.pdf.

[23] Adafruit, "Adafruit 9-DOF," [Online]. Available: https://www.adafruit.com/products/2021. [Accessed December 2015].

[24] ST, "LSM9DS0," August 2013. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/LSM9DS0.pdf.

[25] ADH Technologies Co. Ltd., "GP-20U7," [Online]. Available: https://cdn.sparkfun.com/datasheets/GPS/GP-20U7.pdf.

[26] ADH Technologies Co. Ltd., "Data Sheet / GP-735," [Online]. Available: https://cdn.sparkfun.com/datasheets/GPS/GP-735T-150203.pdf.

[27] maxim integrated, "DS18B20 Programmable Resolution 1-Wire Digital Thermometer," January 2015. [Online]. Available: http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

[28] measurement specialties, "HTU21D(F) Sensor," October 2013. [Online]. Available: http://cdn.sparkfun.com/datasheets/BreakoutBoards/HTU21D.pdf.

[29] Bosch, "BMP180 Digital Pressure Sensor," 5 April 2013. [Online]. Available: http://cdn.sparkfun.com/datasheets/Sensors/Pressure/BMP180.pdf.

[30] measurement specialties, "MS5803-01BA Miniature Variometer Module," 05 July 2011. [Online]. Available: http://www.amsys-sensor.eu/sheets/amsys.de.ms5803_01b.pdf.

[31] Traco Power, "TSR-1 Series," May 2012. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/tsr1.pdf.

[32] Traco Power, "TSR 0.5 Series," 13 January 2015. [Online]. Available: http://www.tracopower.com/products/tsr05.pdf.

[33] JST, "PH Connector," [Online]. Available: http://www.jst-mfg.com/product/pdf/eng/ePH.pdf.

[34] Iridium, "Iridium 9603," [Online]. Available: http://www.rock7mobile.com/downloads/Iridium9603_Brochure.pdf.

[35] Send True, "SM5100B-D GSM/GPRS Module," [Online]. Available:

http://cdn.sparkfun.com/datasheets/Cellular/SM5100B-D_HW_spec_V1.0.0.pdf.

[36] Linear Technology, "LT1083/LT1084/LT1085 Fixed," [Online]. Available: http://cds.linear.com/docs/en/datasheet/1083ffe.pdf.

[37] "Radiometrix HX1," [Online]. Available: http://www.radiometrix.com/content/hx1.

[38] "Trackuino," [Online]. Available: http://www.trackuino.org/.

[39] Texas Intruments, "TL2575, TL2575HV 1-A Simple Step-Down Switching Voltage Regulators," November 2014. [Online]. Available: http://www.ti.com/lit/ds/slvs638c/slvs638c.pdf.

[40] Energizer, "Energizer L91 Ultimate Lithium," [Online]. Available: http://data.energizer.com/PDFs/l91.pdf.

[41] LITE-ON, "LTST-C170GKT," [Online]. Available: http://media.digikey.com/pdf/Data%20Sheets/Lite-On%20PDFs/LTST-C170GKT.pdf.

[42] LITE-ON, "LTST-C170YKT," [Online]. Available: http://media.digikey.com/pdf/Data%20Sheets/Lite-On%20PDFs/LTST-C170YKT.pdf.

[43] SparkFun, "SparkFun microSD Transflash Breakout," [Online]. Available: https://www.sparkfun.com/products/544.

[44] SparkFun, "SparkFun Logic Level Converter - Bi-Directional," [Online]. Available: https://www.sparkfun.com/products/12009.

[45] Advanced Circuits, "Trace Width Website Calculator," [Online]. Available: http://www.4pcb.com/trace-width-calculator.html.

[46] SparkFun, "SIM Socket," [Online]. Available: https://www.sparkfun.com/products/548.

[47] Trackuino, "Trackuino Github," [Online]. Available: https://github.com/trackuino/trackuino.

[48] Atmel, "Atmel Studio 7," [Online]. Available: http://www.atmel.com/Microsite/atmel-studio/.

[49] "mikalhart/TinyGSPlus," [Online]. Available: https://github.com/mikalhart/TinyGPSPlus.

[50] "One Wire Digital Temperature. DS18B20 + Arduino," [Online]. Available: http://bildr.org/2011/07/ds18b20-arduino/.

[51] SparkFun, "SparkFun Humidity and Temperature Sensor Breakout - HTU21D," [Online]. Available: https://www.sparkfun.com/products/12064.

[52] SparkFun, "SparkFun Pressure Sensor Breakout - MS5803-14BA," [Online]. Available: https://www.sparkfun.com/products/12909.

[53] SparkFun, "SparkFun 9 Degrees of Freedom IMU Breakout - LSM9DS0," [Online]. Available: https://www.sparkfun.com/products/retired/12636.

[54] "9DOF LSM9DS0 Breakout Accelerometer Problems," [Online]. Available: https://forums.adafruit.com/viewtopic.php?f=19&t=66571.

[55] LinkSprite, "GSM/GPRS Module User Manual," September 2008. [Online]. Available:

https://www.sparkfun.com/datasheets/Cellular%20Modules/CEL-09533-User%27s%20Manual.pdf.

[56] Radiometrix, "VHF Narrow Band FM 300mW Transmitter," 27 July 2012. [Online]. Available: http://radiometrix.com/files/additional/hx1.pdf.

[57] Honeywell, "TruStability Board Mount Pressure Sensors," August 2014. [Online]. Available: http://www.mouser.com/ds/2/187/honeywell-sensing-trustability-ssc-series-standard-740340.pdf.

[58] "Teensy USB Development Board," [Online]. Available: https://www.pjrc.com/teensy/.

[59] "Paresev 1-B in Tow Flight," 8 April 2009. [Online]. Available: Paresev 1-B in Tow Flight. (2009, April 8). Retrieved from https://commons.wikimedia.org/wiki/File:Paresev_1-B_in_Tow_Flight_-_GPN-2000-000212.jpg.

[60] "Rogallo's Flexible Wing," [Online]. Available: http://history.nasa.gov/SP-4308/ch11.htm#382.

[61] Smithsonian National Air and Space Museum, "Wing, Rogallo Paraglider, Gemini," [Online]. Available: http://airandspace.si.edu/collections/artifact.cfm?object=nasm_A19710831000.

[62] "Parachute," 30 August 2004. [Online]. Available: https://en.wikipedia.org/wiki/Parachute#/media/File:USN_parachute_demo_team_at_Minot_AFB.jpg.

[63] "INTRUDER SYSTEM RAM-AIR TACTICAL PARACHUTE SYSTEMS," [Online]. Available: http://www.airborne-sys.com/pages/view/intruder-system.

[64] "Parkzone Ember 2 RTF Micro Airplane | Horizon Hobby," [Online]. Available: http://www.horizonhobby.com/product/airplanes/airplanes-14501--1/ready-to-fly/ember-2-rtf-pkz3400. [Accessed 26 April 2016].

[65] "Horton Ho 229," 9 August 2009. [Online]. Available: https://commons.wikimedia.org/wiki/File:Horten_Ho_229.png.

[66] M. Hepperle, "Basic Design of Flying Wing Models," 12 January 2002. [Online]. Available: http://www.mh-aerotools.de/airfoils/flywing1.htm.

[67] "3ring," 27 January 2006. [Online]. Available: https://commons.wikimedia.org/wiki/File:3ring.png.

[68] S. Tauner, "File:Servo Diagram.svg," 24 December 2013. [Online]. Available: https://commons.wikimedia.org/wiki/File:Servo_Diagram.svg. [Accessed 26 April 2016].

[69] B. Beauregard, "Improving the Beginner's PID - Introduction," 15 April 2011. [Online]. Available: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/. [Accessed 27 April 2016].

[70] K. S. Lauszus, "A practical approach to Kalman filter and how to implement it," 10 September 2012. [Online]. Available: http://blog.tkjelectronics.dk/2012/09/a-practical-

approach-to-kalman-filter-and-how-to-implement-it/. [Accessed 27 April 2016].

[71] G. Czerniak, "Greg Czerniak's Website Kalman Filter Tutorial," [Online]. Available: http://greg.czerniak.info/guides/kalman1/. [Accessed 27 April 2016].

[72] W. Premerlani and P. Bizard, "Ardu-IMU Theory.wiki," 17 May 2009. [Online]. Available: gentlenav.googlecode.com/files/DCMDraft2.pdf. [Accessed 27 April 2016].

[73] "StratoStarTV How to prepare a weather baloon for launch," 19 July 2012. [Online]. Available: https://www.youtube.com/watch?v=6_06Q_eWta8.

[74] "StratoStarTV How to fill a weather balloon," 19 July 2012. [Online]. Available: https://www.youtube.com/watch?v=5Z23L4QIgtQ.

[75] "StratoStarTV How to calculate lift for a weather balloon," 19 July 2012. [Online]. Available: https://www.youtube.com/watch?v=NxwlNnfKMHs.

Appendix A: Bill of Materials

The following is a bill of materials of all components necessary to build an AURORA radiosonde-glide:

Device	Value	Package	Part #	Quantity	Unit Price	Total Price
AURORA Flight Controller:						
Capacitor	100nF	C0805		6	0.036	0.216
Capacitor	10uF	C0805		1	0.2	0.2
Capacitor	22pF	C0805		3	0.048	0.144
Resistor	10k	R0805		5	0.0152	0.076
Resistor	1M	R0805		1	0.021	0.021
Resistor	4.7k	R0805		1	0.021	0.021
Resistor	510R	R0805		1	0.021	0.021
Resistor Array	10k	CAT16		4	0.068	0.272
Resistor Array	510R	CAT16		1	0.036	0.036
Diode		D1206		3	0.394	1.182
LED Green		CHIP-LED0805		1	0.29	0.29
LED Yellow		CHIP-LED0805		4	0.253	1.012
Header Pins				17	0.00783	0.133
N MOSFET (logic shift)	50V 200mA	SOT23-3	BSS138	3	0.277	0.831
N Power MOSFET		TO220BV	FQP30N06L	2	0.95	1.9
Crystal	8MHz	QS		1	0.7	0.7
ATMEGA2560V-8AU		TQFP100		1	16.55	16.55
ATMEGA328P-TQFP		TQFP32-08		1	3.58	3.58
9DOF IMU	Breakout		LSM9DS0	1	24.95	24.95
GPS		JST4	GP-735	1	39.95	39.95
Temp Sensor		JST 3	DS18B20	1	4.25	4.25
Humidity Sensor	Breakout	JST 4	HTU21D	1	14.95	14.95
Barometer			MS5803-01BA	1	18.98	18.98
microSD Socket		MICROSD_1:1		1	3.95	3.95
Step down regulator	3.3V 1A	TSR-1	TSR1-2433	1	9.56	9.56
JST 3 Header		right angle		5	0.169	0.845
JST 3 Housing				5	0.08	0.4
JST 4 Header		right angle		2	0.22	0.44
JST 4 Housing				2	0.1	0.2
JST Crimps				23	0.0376	0.8648
JST 6 Combo		right angle		1	1.5	1.5
Servo				3	1.0995	3.2985
4AA flat holder				1	3.09	3.09

PCB				1	13	13
Energizer Ultimate Lithium AA				4	1.59	6.36
Airspeed Sensor				1	54.21	54.21
Pitot Tube				1	12.38	12.38
Teensy (Cortex M4)				1	19.95	19.95
AURORA GSM module:						
GSM module			SM5100B	1	39.95	39.95
GSM module connector				1	3.74	3.74
GSM module antenna				1	7.95	7.95
Linear VREG	3.6V	D2PACK/A	LT1085IM-3.6	1	5.5	5.5
SIM Holder				1	0.95	0.95
Capacitor	10uF 25V	C1206		1	0.24	0.24
Capacitor Tantalum	22uF Tant	C1206		1	0.77	0.77
Resistor	10k	R0805		1	0.0152	0.0152
SIM Card			AT&T 6006A	1	5.56	5.56
PCB				1	5.67	5.67
SIM Card Plan				1	10.63	10.63
AURORA Radio module						
HX1-144.390-3				1	47.95	47.95
Buck Converter	5V 1A	TO-263-5	TL2575-05IKTTR	1	2.28	2.28
Diode			1N5819HW	1	0.44	0.44
Capacitor Tantalum	100uF 10V	C1210		2	1.62	3.24
Capacitor Tantalum	330uF 10V	C2917		1	1.84	1.84
Inductor	20uH			1	1.15	1.15
Inductor	330H			1	0.72	0.72
PCB				1	4.1	4.1
Flight Controller Assembly						
Nylon Nuts				38	0.0563	2.1394
Nylon Bolts				6	0.0527	0.3162
Small Zip ties				8	0.0428	0.3424
3D Printed Spacers Short				2	0.01	0.02
3D Printed Spacers Tall				2	0.01	0.02
3D Printed Washers				4	0.01375	0.055
3D Printed Barometer Cover				1	0.01	0.01
3D Printer GSM Spacer				1	0.01	0.01
3D Printer Spacer w/ Arms				2	0.03	0.06

Original Flight Controller Total						386.03
Flight Controller + Teensy Total						405.98
Airframe						
Foam Board (Readi-Board)				2	1	2
SG90 Servo				3	1.8	5.4
Nylon Control Horn				2	0.545	1.09
Control Rod				1	1.09	1.09
Carbon Fiber Rod				1	10.25	10.25
AURORA Radiosonde-Glider Total						**425.81**

Table 8: Bill of Materials

Appendix B: PCB Schematics

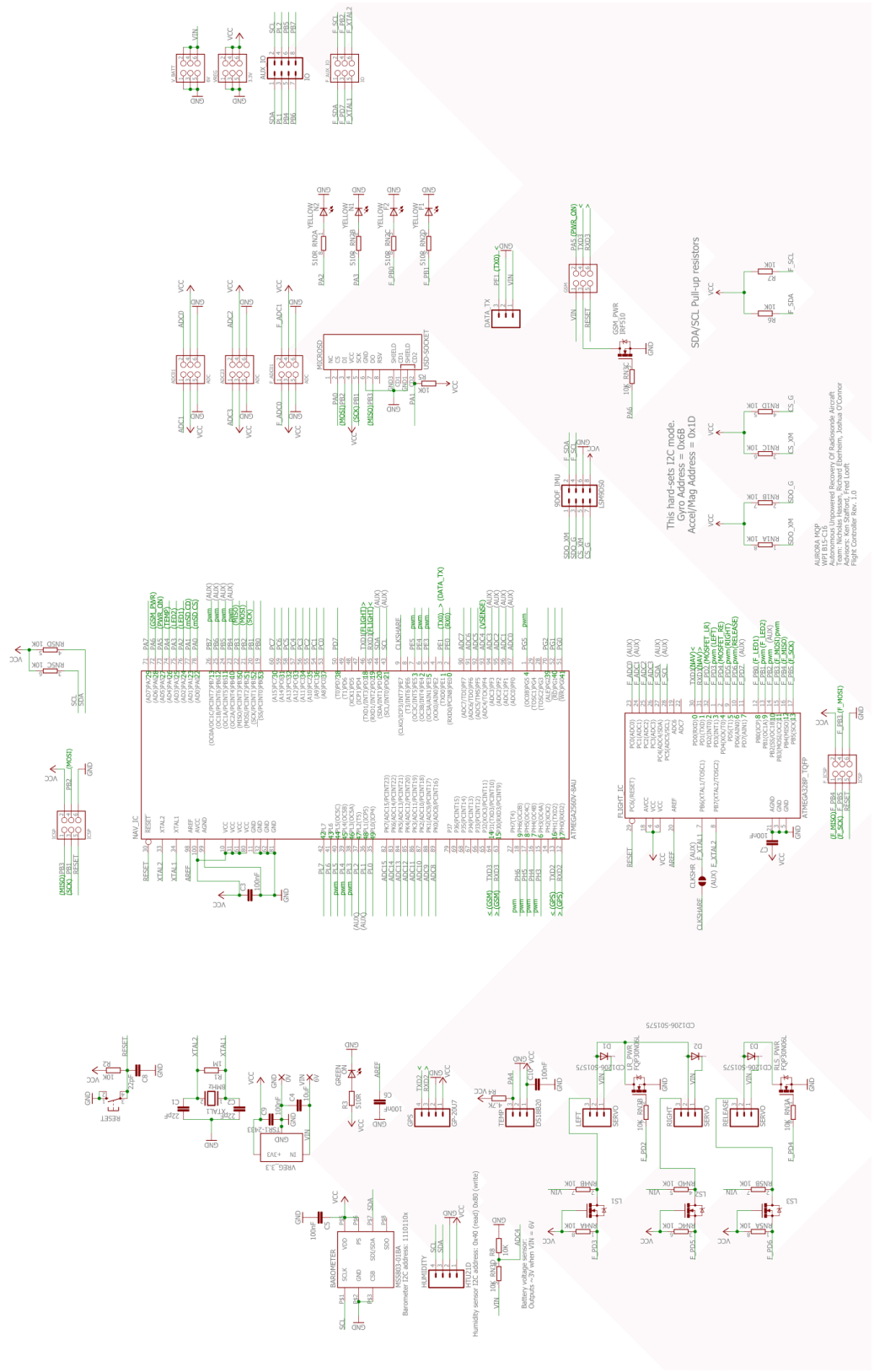

Figure 75: Flight Controller PCB Schematic

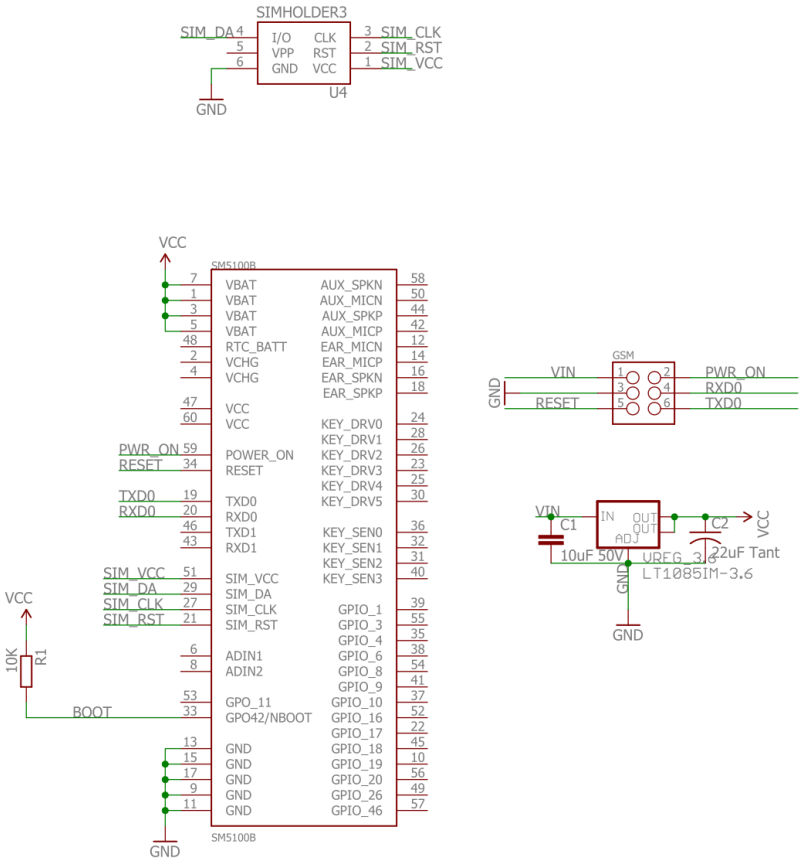

Figure 76: GSM Module PCB Schematic

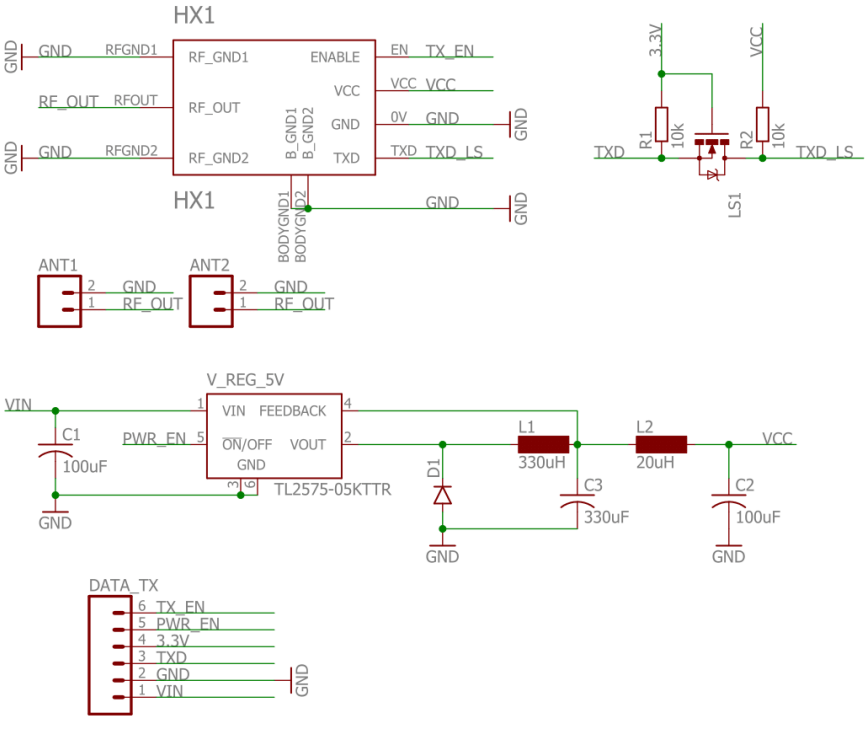

Figure 77: Radio Module PCB Schematic

Appendix C: Procedure for Building the Airframe

Introduction

This section provides the step-by-tep procedure developed to build the airframe from scratch. With experience and the recommended tools, total assembly time is approximately two hours. If starting from scratch without tools like a laser cutter, the assembly time is closer to six hours, which is still plenty of time to prepare the entire glider between the launch times of National Weather Service balloons, every twelve hours.

Materials/cost

Item	Cost per Unit	Quantity	Total Cost
Readi-Board Polystyrene Foam Presentation Board .508m x .762m (20" x 30")	$1.00	2	$2.00
SG90 Servo motors	$1.80	3	$5.40
Nylon Control horns (set of 2)	$1.09	1	$1.09
Control rods	$1.09	1	$1.09
2.38mm(3/32 in) carbon fiber rod (1.2m)	$10.25	1	$10.25
Total			$19.83

Table 9: Airframe Materials and Cost

Tools

The tools needed to build the glider are listed below. Required tools are necessary to build the glider in its most rough form, whereas optional tools allow manufacture to be finer and assembly to proceed faster.

Required Tools

- Utility knife
- Hot glue gun with hot glue
- Straight edge/1 meter ruler
- Iron
- Electrical tape
- Pen/pencil

Optional Tools

- Hobby knife
- Laser Cutter
- Heat gun

- Vice
- Dremel with cutoff wheel
- 1/16 in steel welding rods
- ⅛ in drill bit
- Flat-head screwdriver
- Isopropyl alcohol

6.4 Steps

1. **Acquire materials**

   The materials and tools needed to build the glider are detailed in the proceeding sections, most can be found at a local dollar or hobby store. Servo motors and carbon fiber rods can be found in hobby stores but are usually available from online retailers for cheaper.

2. **Cut foam board to fit laser cutter**

   An optional step, one 20” x 30” board can accommodate either two wings, two copies of every other part together, or one wing with a copy of one fuselage and winglet. The laser cutter available to the project in Washburn shops could only hold materials that are 24” x 20”, meaning that the boards had to be trimmed down with a straightedge and utility knife to fit. The excess material should not be discarded as it can be cut into the covers for the fuselage later on.

3. **Cut parts from foam board**

   Use either the CAD files with a laser cutter or plywood templates to lay out the patterns of the parts onto the board with a pen. Laser cutter settings should account for the hinge of the control surfaces in the wings. All parts except the wings have symmetry and can be cut in any orientation. The wings control surface hinge cut will be facing down in the final airframe, so wings should be flipped with each cut to ensure that there are equal number of left and right wings.

4. **Finish wings**

   After all parts are cut, begin heating the iron. As it heats up, the control surfaces of the wings should be bent back to ensure all the foam has been cut. With the control surface folded back, the utility knife is used to cut an angled edge along the control surface so that it can deflect in both directions. Care should be taken that the knife does not cut the paper hinge. If the paper is accidently cut, a thin layer of hot glue on the top side of the hinge can be applied to reinforce it.

   Use the iron once it is heated to round the leading and trailing edges of the wings and winglets into a bullnose shape. Slowly sliding the iron at an angle along the edges of the wing while applying light pressure is the best method to ensure even heating and avoid uneven edges. The edges of the winglets can be applied all at once to the iron.

   The servos should now be mounted into the slots already cut into the wings. They are hot glued in place. The control horns are placed with screws that are pushed through guide holes included on the template. These control horns are then tightened in place with the optional screwdriver or any tool that can tighten flat head screws. Connecting the servos to the control horns with the control rods completes the wings.

   One optional but highly recommended step to do now is to attach the fuselage walls to the bulkhead and tack them in place with some hot glue on the underside. This

will make attaching the wings easier later on if the fuselage walls are not free to move around while the spars are being inserted.

5. Cut carbon fiber spars to length

The carbon fiber rods are the most reusable part of the glider and only need to be cut if none are already available to use. Three spars are needed at lengths of 24 cm, 30.5 cm, and 61cm. Mark the lengths and use a non-toothed blade to cut the rods. Ideally a diamond cutoff wheel should be used, but if none are available the utility knife will suffice.

If a heat gun and welding rod are not available, both ends of the carbon fiber rod should be cut to a point so that they can be pressed into the wings with as little tearing of the foam as possible.

6. Insert spars into wings

If a heat gun and welding rod are available, begin heating one end of the rod. While it is heating, mark the locations that the spars will go into on the wings and bulkhead. These locations are the center of both tabs on the wing and the center of the aft-most tab of the bulkhead. Make sure all marks line up between the wings and bulkhead before boring out the foam.

When the rod is heated (usually some bluing can be observed on the rod), insert it into the foam perpendicular to the edge at the marks that were just made. The rod should slide in easily, melting the foam back as it goes. If it does not, it needs to be heated more. Press the rod as far as the carbon fiber rod will need to be pressed in, marks can be made on the surface of the wing or rod to help with this step. Pull the rod out quickly to prevent excess melting at the end of the hole. Reheating the rod between boring into the foam makes the task easier and helps clean melted foam off the end of the rod.

After all the holes have been bored out, insert the carbon fiber rods through the bulkhead. After they are lined up, apply the fuselage walls if they have not been tacked on already followed by one wing. The wing should have the hinge cut facing down and care should be taken to avoid bending the wing, tearing the foam, or pushing the carbon fiber through the wing's surface. After the first wing is properly inserted, the second wing can be put on the other side, again with care taken to avoid damage.

7. Cut and bend fuselage covers

The easiest way to make the fuselage cover is to take the excess foam board cut before cutting the actual glider parts. These scrap sections are usually 50.8 cm x 15.25 cm, and an accurate cut will give two equally-sized covers. Each cover must be 75mm wide, and can easily be cut with the utility knife and straightedge.

After the covers are cut, take one cover and begin curving one end of it by hand. Test fitting the curve against the fuselage walls is recommended. The curve should follow the nose around the top to the bottom, ending about 6-7 mm from the tip of the nose.

Optionally, go to step 8 at this point and install the electronics. The covers are designed to allow easy access for maintenance however installation of the electronics is usually easier before the covers have been installed.

Once the curve is formed, it should be glued down to the edges of the fuselage walls from the bottom end to approximately the same position back from the nose at the top edge. The cover should be held in place until the glue hardens to ensure it is secure. At this point the back edge can be cut to size so that the top cover now bends around the nose and cover the entire top of the fuselage to the tailing edge. The top cover should

now be cut again about at the point that is above the halfway point of the battery pack. This ~15 cm section is where the access port for the release mechanism is cut. It should be cut so that it overlays the release servo when attached to the fuselage. A simple square port will suffice, like the control surfaces on the wings one side should be a hinge cut that allows the port cover to open outwards. This hinge can be reinforced with electrical tape applied while it is open. Once the port is cut, the cover section can then be glued to the top rear of the fuselage along its entire length.

After the top cover is complete, the bottom cover can be sized by placing it against the front edge of the top cover and trimming it at the tailing edge. Using a pen or hobby knife, the location of the release servo should be marked out on the bottom cover and then cut out; this will allow the servo to fit into the bottom cover. A hinge should then be cut about 1.3cm forward of the servo motor's hole that allows the majority of the cover to open outwards and give access to the underside of the bulkhead. With the cover closed, the rear section should be glued in place.

8. Install electrical components

The electronics are installed with zip ties. eight ⅛ in zip ties are used, four for each corner of the board and four used to tie the other ends of the other four. 3D printed PLA washers are used on the underside to spread the load on the zip tie's heads and prevent them from pulling through the foam bulkhead. PLA spacers are used between the bulkhead and the electronics board itself, and then the extra zip tie heads are applied and tightened.

The servo wires are threaded under the bottom cover and between the bulkhead and wing slots. They are connected into their respective ports on the electronics board. The battery pack is installed using two Velcro straps that thread through the aft-most wing slot and slots cut into the bulkhead. The batteries should be perpendicular to the fuselage to prevent them from shifting as the glider decelerates in landing and causing a power loss and reset.

The release servo is inserted into the slot already cut out for it in the bulkhead and glued into place. This servo will have to bear the entire weight of the glider in ascent, and it is recommended that it be glued on all sides on the top and bottom of the bulkhead if possible. The PLA release mechanism is glued down next to the servo. The mechanism uses excess control rod from the control surfaces to hold the string since it is strong and rigid.

The GPS receiver is hot glued over the center (30.5cm) wing spar between the board and battery pack. The hot glue should be applied to the bulkhead and allowed to cool slightly before installing the receiver to prevent any heat damage to the component.

Using the optional drill bit, two holes should be punched through the fuselage wall for the pitot tube's air tubes. These holes should be large enough that they do not constrict the tubes but small enough to hold them in place at the airspeed sensor's ports. Finally, a slot should be cut in the underside of a fuselage wall that the reset button can be inserted through. Like the servo motors' wires, the button's wires should be threaded between the bulkhead and wing slots. The button is then hot glued into place.

9. Final Gluing and Fastening

After the electronics and covers are installed, the final gluing can be done. The winglets can be glued to the wings at any time in after they are both ironed, however now is the

best time to glue them in place since the glider is no longer being flipped and turned repeatedly.

Tabs can be cut from scrap foam board that can be glued to the bottom of each cover that will help them stay down when closed. These tabs should extend 6-12 mm from under the edge of the covers. Applying lengths of electrical tape to each latch allows for easy opening and closing before launch. Two lengths are recommended; one across the stationary part of the cover and one hanging over the tab on the opening part of the cover. This prevents the electrical tape from tearing the paper off the foam board when the cover is opened.

The wings can either be left as is or glued along the root to secure them in place. In testing it was found that leaving the wings unglued expended energy in a crash as they separated without damage. In tethered situations, this ended up hampering the glider in the wind and allowed the wings to be torn off before flight. Gluing the wings before flight is recommended.

Right before flight, one large (¼ in) zip tie should be inserted through the top cover, bulkhead, and bottom cover with two large PLA washers on either end before being tied off with the head of a second large zip tie. This zip tie will keep the covers closed tightly during flight.

10. Balancing

After assembly, the CG may be too far back depending on the electronics' configuration. The CG should be approximately 15 cm back *from the leading edge of the wing at the root.* A deviation of 2 cm forward or aft of this point is acceptable but ill advised. If ballast must be applied, it should be applied to the underside of the nose to use the least amount of weight and prevent the ballast from colliding with the control board if it ever comes loose. AAA batteries work very well for this application.

Appendix D: Preparing the Weather Balloon

Materials

-800g weather balloon
-Payload lines
-Zip ties
-Electrical tape
-PVC pipe (approx. 2 inch length, diameter varies)
-Flush cutters

Procedure

Taking a length of payload line, tie off both ends with figure-8 climbing knots so that each end had 5-8 cm loop. Find the center of the line and tie off another figure-8 knot on that end. The final length should be at least 61 cm long with two loops on one end and one loop on the other.

Pull the neck of the balloon out of the plastic packaging while avoiding touching the thin part of the balloon. Depending on the neck of the balloon, which varies by manufacturer and size of the balloon, a 5 cm PVC insert is made from pipe and inserted into the neck about 7.5-10 cm. If cut from a pipe the edges should be smoothed to prevent damage to the balloon.

A zip tie is threaded through one loop of the payload line, around the balloon neck at the insert, and through the second loop before coming around to be zipped off tight. After it is secure, three more zip ties are added for redundancy. The excess ties are trimmed down and then wrapped in electrical tape to prevent sharp edges from puncturing the balloon. The balloon is now ready to be filled. [73]

At the launch site; the tank holding the lifting gas, usually helium or hydrogen, is secured to prevent it from moving and damaging the regulator. Safety lines are tied between the tank and the two loops already zip tied to the balloon. A tarp or blanket is laid out over the ground or table (depending on launch location) to prevent debris from damaging the balloon.

The balloon is removed from the packaging and laid out over the tarp so that any wind at the launch site is moving from the bottom to the top of the balloon. The hose from the regulator is inserted through the PVC insert and a zip tie is tightened around the neck under the PVS insert so that helium does not escape during filling.

When it is time to start filling the balloon, it must be filled slowly at first to allow the latex to unwind without knotting or tearing. After the balloon is opened up, the filling speed can be increased. As the balloon begins to rise, it should be held up by the neck to keep it off the ground and collide with anything sharp. [74]

A spring scale can be used to measure the lift of the balloon. The scale can be tied in line with one of the safety lines. Neutral lift is the point at which the lift of the balloon equals the weight of the payload. It is recommended that the balloon be filled to have at least 1.5x the payload weight before being released to ensure a safe launch. Having too little lift will result in a longer ascent time and the balloon will not burst for a long time, allowing it to drift much farther than intended.

To estimate the required amount of helium, it can be estimated that 1 cubic foot (28.3 L) of helium equals about 1 ounce (.278 N) of lift. The required helium should be able to lift the payload as well as the weight of the balloon and lines. [75]

For the Aurora glider, the glider is attached by threading the payload line into the release mechanism while the glider is in preflight mode. The servo should latch into place and secure the balloon to the glider once the glider enters ascent mode. To launch the balloon, remove the safety line and walk the balloon up by the payload line until holding the payload itself. From there simply walk with the wind and let go of the balloon away from any trees or tall buildings.

Appendix E: Airframe Component Schematics

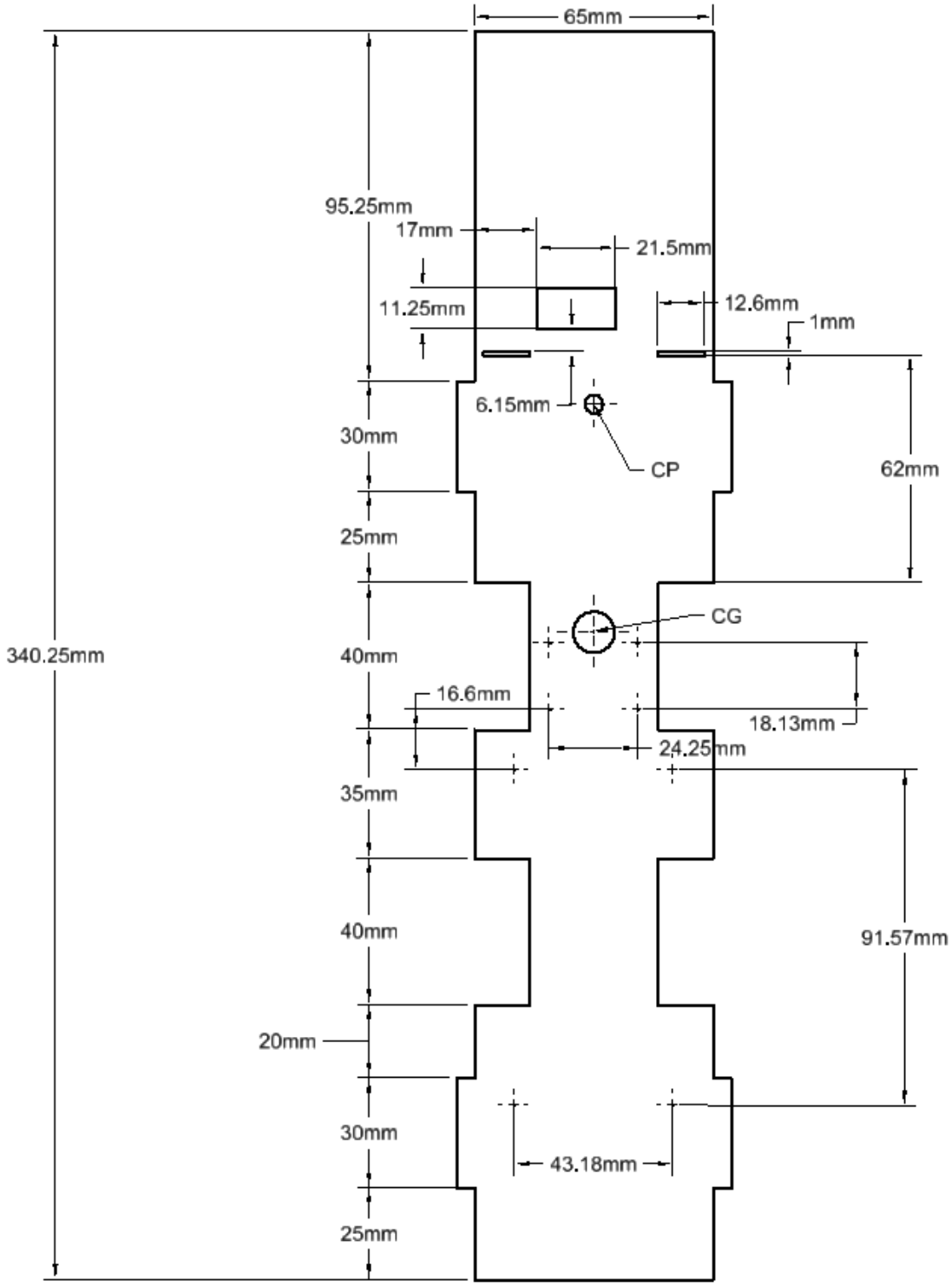

Figure 78: Airframe Bulkhead

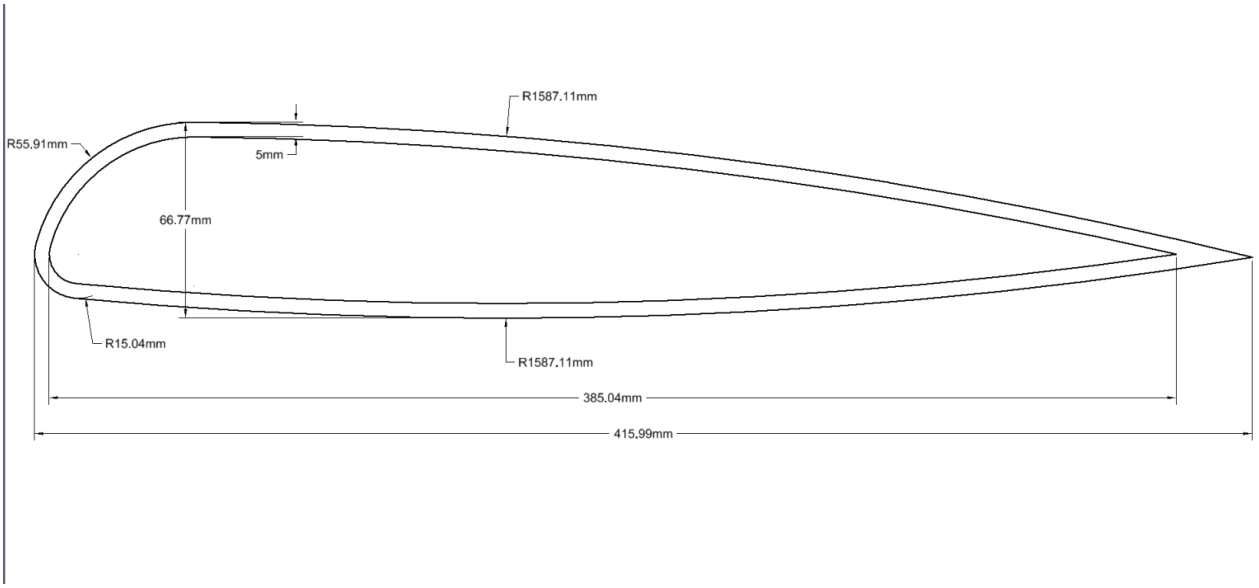

Figure 79: Airframe Fuselage Covers

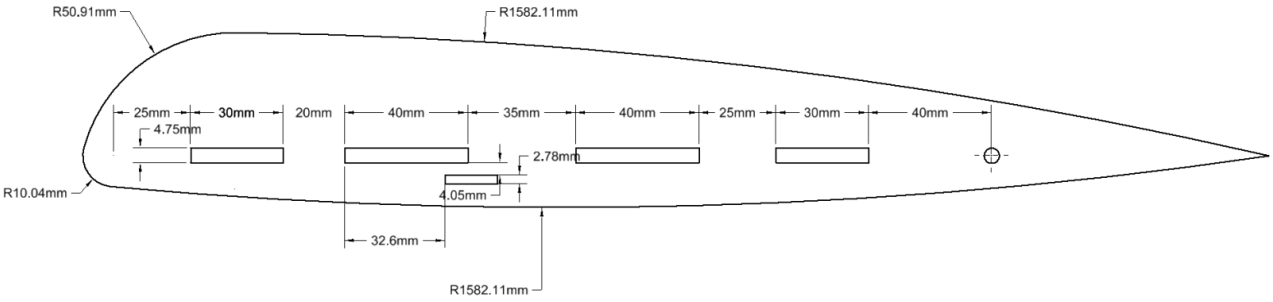

Figure 80: Airframe Fuselage Wall

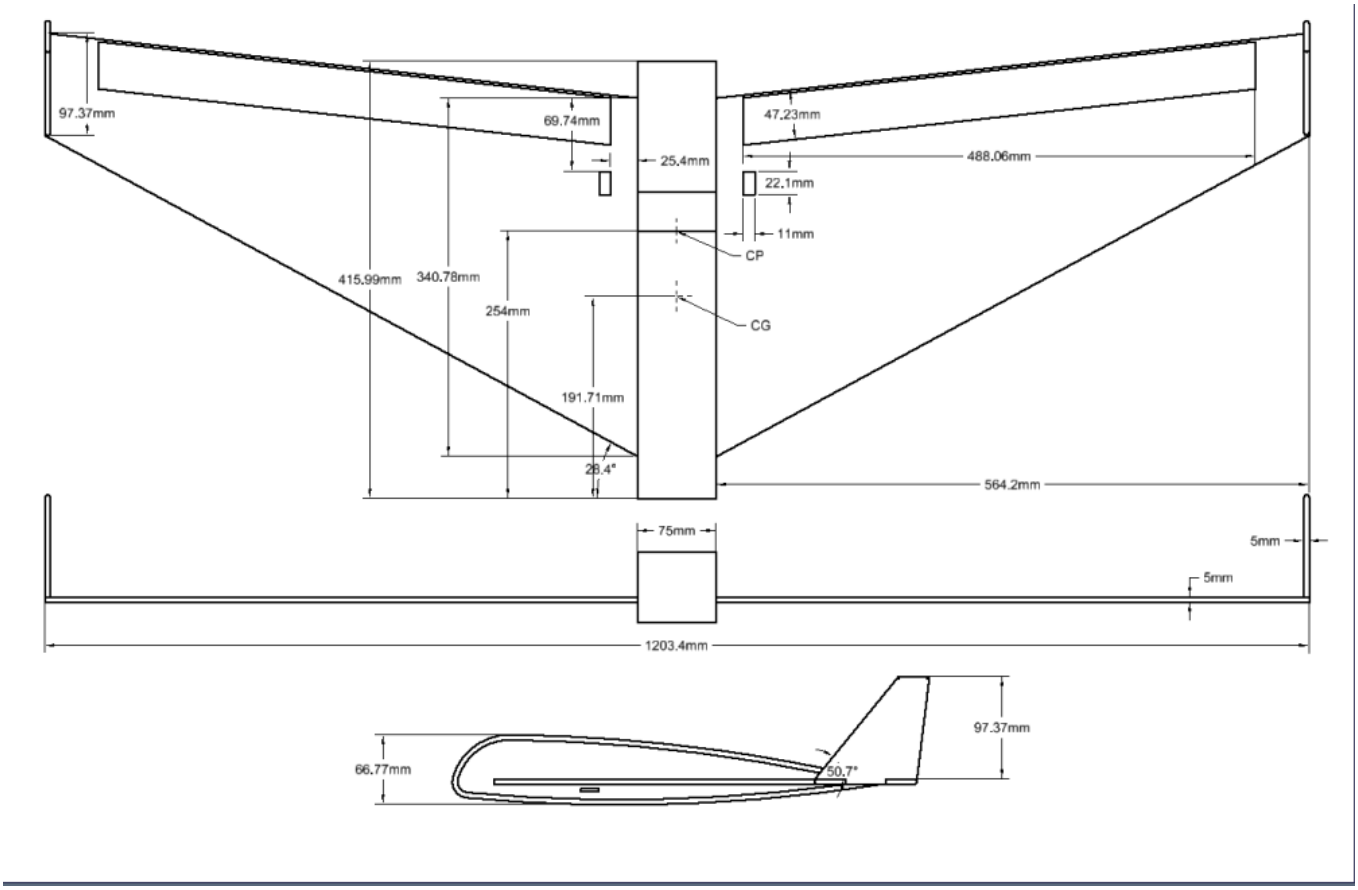

Figure 81: Schematic View of the Final Glider

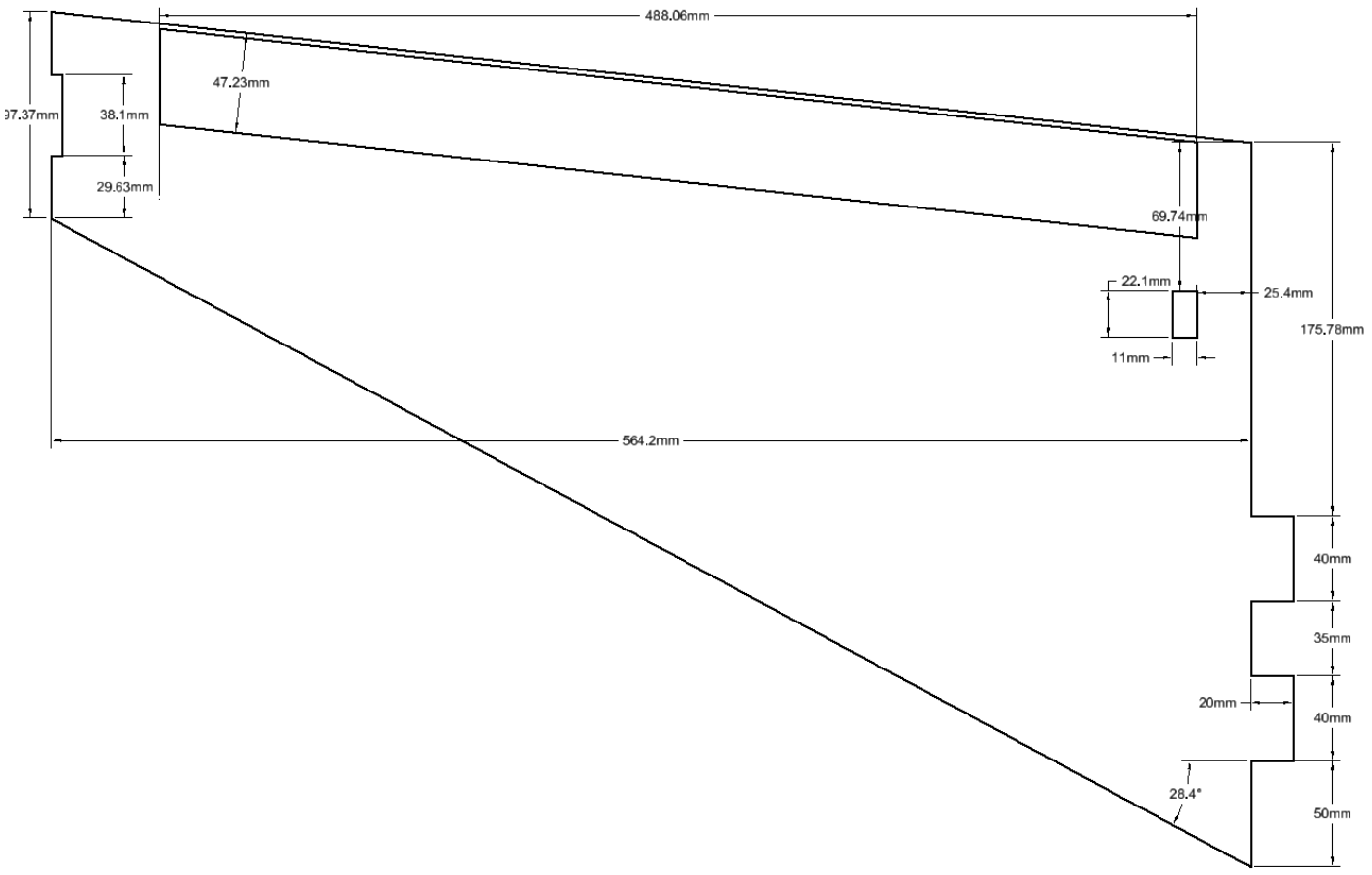

Figure 82: Airframe Wing

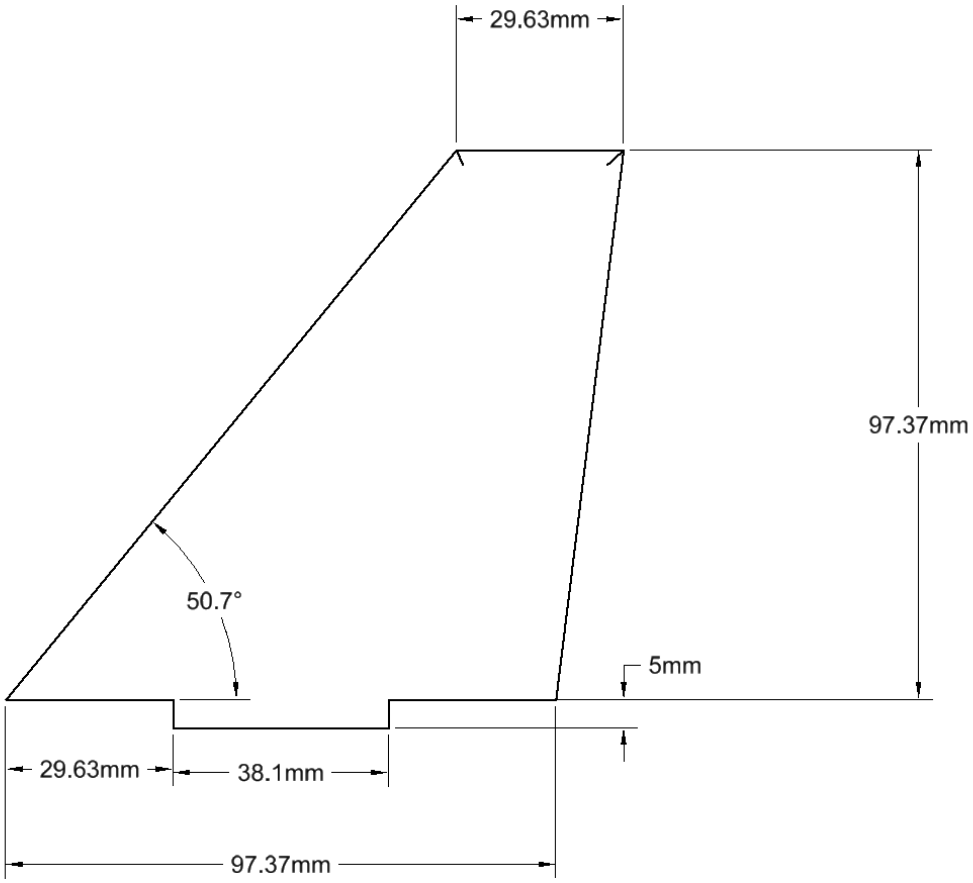

Figure 83: Airframe Winglet