EXPLORATION OF A LOW-COST AUTOPILOT SYSTEM FOR USE IN ACADEME

A Thesis By

Bryan J. Kissack

Bachelor of Science, Wichita State University, 2010

Submitted to the Department of Aerospace Engineering
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

July 2012

EXPLORATION OF A LOW-COST AUTOPILOT SYSTEM FOR USE IN ACADEME

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science, with a major in Aerospace Engineering.

_____
L. Scott Miller, Committee Chair

_____
John Watkins, Committee Member

_____
Animesh Chakravarthy, Committee Member

DEDICATION

To my parents and my wife for their loving support

ACKNOWLEDGEMENTS

I would like to thank my advisor, Scott Miller, for his advice and support. I also extend my gratitude to my committee members John Watkins and Animesh Chakravarthy. A special thanks to my support team on the project—Jimmy Tennant, Kevin Hagen, and Jonathan Mowrey.

ABSTRACT

With the world's growing use of autonomous unmanned aerial systems (UASs), there is also a growing need for higher education to teach students how to design and implement the autopilot systems that many of these UASs depend on to perform their designed missions. Given the inherent cost and complexity of these systems, it has been difficult in recent years to provide students the hands-on experience that is crucial to understanding how these autopilot systems work. Another stumbling block to implementing this type of education has been the proprietary nature of autopilots, which restricts the ability to modify/enhance the autopilot. The good news is that autopilot-related components continue to become lighter and cheaper, which has created the development of open software/hardware platforms. Arduino is one such microcontroller that has come to the forefront as a leader in the open software/hardware autopilot system market. The goal of this study was to determine whether or not an Arduino-based autopilot system would be a viable candidate for implementation into higher education at a design level. A series of flight tests were performed to discover the strengths and weaknesses of this product in order to help determine how easy or difficult it would be to integrate it into undergraduate studies. Results from the flight tests show that this autopilot system is fairly robust and has a wide range of functionality. Through these tests, it has been concluded that the Arduino-based ArduPilot-Mega microcontroller would be a worthwhile educational tool and is an inexpensive alternative to proprietary autopilot systems.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

Chapter Page

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF ABBREVIATIONS / NOMENCLATURE

AIAA	American Institute of Aeronautics and Astronautics
APM	ArduPilot-Mega
AR	Aspect Ratio
AVL	Athena Vortex Lattice
BEC	Battery Eliminator Circuit
CFD	Computational Fluid Dynamics
DIP	Dual In-Line Package
DOF	Degrees of Freedom
ESC	Electronic Speed Controller
FBW	Fly-By-Wire
GPS	Global Positioning System
GUI	Graphical User Interface
HIL	Hardware-in-the-Loop
ID	Inner Diameter
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
LiPo	Lithium-Ion Polymer
MIT	Massachusetts Institute of Technology
NiMH	Nickel–Metal Hydride
OD	Outer Diameter
PID	Proportional-Integral-Derivative
PPM	Pulse-Position Modulation
PWM	Pulse-Width Modulation
RC	Remote Control
S&C	Stability and Controls
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle

CHAPTER 1

INTRODUCTION

As technology continues to progress, one of the most fascinating and quickly growing fields in aviation is that of unmanned aerial systems (UASs). Like any branch of education, as the field evolves, so must the way the field is taught. It is essential for academic institutions to continue to improve their educational system, keeping it as relevant to real-world applications as possible; the caveat to this is that some subjects in education are harder to incorporate into a learning environment than others. Common factors causing this difficulty are the time it would take to teach the subject, the expenditure required on behalf of the teaching institution or the student, and the complex nature of some subjects.

As the author of this thesis, I have had the unique opportunity to be both a student and an educator at the same academic institution. This has allowed me to experience education from both sides and has given me an interesting perspective on the current state of the educational system, especially when dealing with actively controlling an aircraft through computation methods. In the recent past, it has been fairly difficult for academic institutions to provide hands-on training for most forms of autopilot systems. This is mainly due to the expensive nature and complexity of such systems. Some opportunities exist at the graduate level, but most undergraduates will never get the opportunity to work with the hardware to make an autopilot system. Students are able to learn about autopilots on a theoretical basis through a controls class, which is essential to learning active controls, but is this enough? Other aerospace fields offer opportunities for hands-on experience for students, so should autopilots offer this as well? In order to better answer these questions, it is beneficial to take a look at the history of UASs to really understand the scope of this field.

1.1 Brief History of Unmanned Aerial Systems

Unmanned aerial vehicles (UAVs) have been a dream ever since humans could first fly. One of the initial documentations of this idea was in 1915 when Nikola Tesla introduced the idea of unmanned flight in his dissertation in which he described an armed pilotless-aircraft designed to defend the United States [1]. As far back as World War I, remote-controlled airplanes have been used to try and gain a tactical advantage on the enemy. The Hewitt-Sperry Automatic Airplane was a project undertaken by the United States during World War I, the goal of which was to develop a pilotless aircraft that was essentially a flying bomb. This would have been equivalent to a 1910 cruise missile. During World War II, the Germans developed the V-1, a pulse-jet-powered UAV that again

resembled a cruise missile. During the 1960s, UAVs started to come into their prime because they were being developed not as missiles, but as target drones and remotely-piloted combat vehicles. An example of this was the United States' AQM-34 Ryan Firebee aircraft, which was launched and controlled from a DC-130 for covert surveillance missions. The Firebee program continued through the 1970s sparking many other countries to develop their own UAVs [2].

With the close of the 20th century came another progression in UAV capability. The technology boom that began in the 1980s spurred new development in UAVs to a more sophisticated level. UAVs were now capable of taking off and landing, performing missions with the help of sophisticated sensors and stability/control augmentation systems. The most widely known of these UAVs is the MQ-1 Predator. The Predator, whose maiden flight was in 1994, was capable of being piloted by a ground crew at the point of takeoff [2]. By the 2000s, the Predator was able to be controlled from a ground station that could be nearly anywhere in the world [2].

The focus now has shifted to fully autonomous vehicles such as the Global Hawk, which is a surveillance vehicle developed my Northrop Grumman that can takeoff, fly a mission, and land without any human intervention [2]. This vehicle is capable of reconnaissance anywhere in the world, and without a crew, it is lighter and can fly longer missions than if it were manned. The next step in autonomous vehicles sounds more like science fiction, but it is quickly becoming a reality. Northrop Grumman is just one company honing in on creating unmanned combat aircraft such as the X-47B, which had its maiden flight on February 4, 2011. Imagine dogfights between fully autonomous aircraft, in which the winner was determined solely by the engineering behind the aircraft and the autopilot system.

With the turn of the 21st century, there has also been a shift in terminology, and the UAV acronym has become somewhat obsolete. How can a remote-controlled airplane that a child would play with and a sophisticated aircraft such as the Global Hawk both be considered UAVs? The Department of Defense has made a position of renaming many of their unmanned aerial vehicles as unmanned aircraft systems (UASs). The emphasis here is on the word "system," indicating that there is much more going on than just an aircraft capable of being unmanned.

1.2 Educational Aspect

With such a long history of the military utilizing UAVs and with autonomous UASs having been around for nearly 30 years now, why have autopilot systems not been better incorporated into aerospace undergraduate studies? Quite simply, the cost of these systems has been far too high, and the size of these systems has been far too

large to reasonably incorporate into an educational institutional where the intention is to put these systems on actual aircraft that is either bought or built with limited academic funds. Another problem has been access to an autopilot system that is not protected due to proprietary measures. Using an autopilot system that has restrictions on changes would hinder the learning ability of the systems.

In recent years, there has been a new mile marker in UASs. While the history of UAVs and UASs has been strongly rooted in military applications, a new frontier is emerging. As technology continues to improve, the dream of smaller/lighter sensors, processors, and communication devices has become a reality. Many of these components have become much cheaper as well, opening up the use of these systems for commercial and recreational applications.

The only obstacle left in order for educational institutions to have an optimal system for teaching autopilots is autopilot software/hardware that does not have accompanying stringent proprietary protection. This, too, looks to be a problem that is being eliminated. A byproduct of these new smaller, lighter, and cheaper components is that it has become much easier for the average person to "get into the autopilot game." This combined with the emergence of social networking has created an environment where individuals can work on their own autopilot systems, while comparing their results with other individuals who are doing the same thing. This creates an environment where individuals can "feed off each other", learning from one another and working together to better their individual products. This is much like a learning environment you would see at a university, only without the classroom. Entire online communities whose sole purpose is to advance the development of autopilot systems in recreational vehicles such as remote-controlled airplanes have been popping up and growing rapidly. Many of these communities have based their autopilot systems around open software and hardware. The advantage of this is that instead of everyone independently developing their own autopilot systems, a community of developers can work together and share ideas to advance the system much more rapidly than an individual or small group could. Another advantage of community development is that the overall program tends to be simpler, since a large number of people need to be able to understand it. The overall product that is produced ends up being simple, clean, and very effective.

1.3 Introduction to Arduino

One such open-source environment is called Arduino. The Arduino project was started in 2005 and has been growing rapidly. This group of open software and hardware microcontrollers was built with the intention of

providing people of all fields with a tool to interact with their intended environment. A microcontroller is essentially a small processing unit similar to a computer, but it generally has only one dedicated task. In Arduino's case, this series of microcontrollers has been built to be able to interact with many different external sensors and actuators. Interestingly, Arduino was not started with the intention of creating autopilot systems specifically. It was based on a much broader idea of creating a simple tool for interacting with the environment. For this reason, the Arduino language was designed to be straight forward, with a shorter learning curve than many other programming languages because it was designed for people of non-technical background to use as well. This, along with its relatively cheap components, has made it one of the most popular open-source microcontrollers in the market today. A more in-depth view of Arduino and more specifically the ArduPilot-Mega (APM) board will be discussed later.

With such a product available, it begs the question: "Can this be implemented into a learning environment, and more specifically a design learning environment?" The following investigation and the core of this report will be to determine whether or not Arduino is a suitable tool for giving students hands-on learning experience with an autopilot system, the ultimate goal of which is implementation into an aircraft. In order to do this, a study must be done to find the strengths and weaknesses of the product.

Initially, there are possible stumbling blocks to incorporating such a product into a classroom. First, since this is not currently standard at colleges or universities, changes would need to be made to the curriculum to make this addition. Also, can a product that is so cheap and open-source be a consistent platform for the use of teaching? With this in mind, great care must be taken to ensure that Arduino is suitable for a traditional leaning environment, and a comprehensive test of the autopilot system must be done to ensure that the system is adaptable, robust, and, most importantly, reliable.

1.4 Arduino Autopilot System Validation Steps

The following objectives have been identified to try and accomplish the goal of validating the Arduino autopilot system:

- Create a fully autonomous Arduino autopilot system that will be implemented into a test aircraft.
- Put the autopilot through a rigorous flight test screening to find weaknesses and strengths.
- Summarize all findings, and relate this information back to the feasibility of incorporating the autopilot system into a learning environment.

- Provide some tutorial information of how to implement this system academically, and describe a process of how to accomplish autonomous flight.

At the end of this exploration, a recommendation will be given as to the validity of implementing a program, based on the Arduino software and hardware environment, which will teach the undergraduate aerospace student about autopilot systems. A higher-level objective is to provide feedback on the Arduino autopilot system to assist faculty and administration in making the decision about whether or not to incorporate this program into their curriculum.

CHAPTER 2

ARDUINO AS AN AUTOPILOT

The Arduino Project was started in 2005 by a group of engineers led by Professor Massimo Banzi in Ivrea, Italy. Originally designed for Banzi's students at the Interaction Design Institute Ivrea, Arduino has become one of the most popular open-source microcontrollers to date. According to the Institute of Electrical and Electronics Engineers (IEEE), as of October 2011, more than 250,000 Arduino boards had been sold worldwide, not including the clones because it is open-source [3].

"Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments [4]." What makes Arduino so interesting is the open-source aspect. Open-source could be described as a philosophy—that a person or group's product is given freely to the public for them to redistribute and even redefine the product however they see fit. More simply put, it allows anyone to use a product and tailor make it for what their agenda is, as opposed to most commercial products, which the main emphasis is on profiting off the product itself and quality control. A more in-depth look at the difference between open-source and proprietary products will be considered in the following sections.

Arduino has two sides: hardware (physical microcontroller) and software (programming language used to program the microcontroller). With an open hardware platform, it is possible for anyone to read the schematics and build an Arduino board. Since the average person does not have the tools available to build such a microcontroller, one is able to buy Arduino microcontrollers from a variety of places and for a relatively low cost. Several different official Arduino boards range from the tiny Arduino Nano to the much larger, much more powerful Arduino Mega 2560. Each board is specially designed to offer different amenities that will allow users to accomplish a particular task more easily. From the number of input/output pins available to the microcontroller having a connector that will allow you to plug it into the wall, each Arduino board is different, but all of them operate fairly similarly. The board that most satisfies our needs to build an autopilot system is the Arduino Mega 2560.

The software side of Arduino is attractive when looking to integrate into a learning environment. The Arduino was built on the premises of one question: "How do you teach students to create electronics—fast [2]?" When Banzi, a software architect, joined the Interaction Design Institute Irvea as an associate professor, his job was to promote new ways of interactive design. The microcontroller that was commonplace at that time was limited in

processing power, expensive, and not able to run on all operating systems. From these needs came Arduino, based on a programming language called Processing (a simple language that even inexperienced programmers could learn fairly quickly), which was built by a colleague of Banzi's from the Massachusetts Institute of Technology. Along with a friendly integrated development environment (IDE) based on a platform called Wiring, Banzi's group created a product that is able to be used and modified by students who are not immersed in computer programming, for a price that a student can afford. Having its roots in education, Arduino appears to be the perfect candidate for developing an educational process for teaching students about autopilot systems.

2.1 Main Autopilot Hardware

As discussed earlier, there are several different Arduino boards from which to choose. This thesis will focus on the ArduPilot-Mega 2560 board, due to its abundance of input/output pins and also because it is one of the more powerful microcontrollers, with 256 KB of flash memory (Atmel AVR 8-bit at 16 MHz). This may not seem like much memory compared to the specifications for a modern laptop, but for performing the single task of driving an autopilot system, it is plenty. The ArduPilot-Mega is not an official Arduino board but rather a "derivative" of the Arduino Mega 2560 and built on behalf of DIY Drones, the largest amateur unmanned aerial vehicle (UAV) community on the web, by 3-D Robotics (the manufacturing arm of DIY Drones). As stated previously, Arduino hardware is open-source, so DIY Drones has taken the original Arduino Mega 2560 and molded it to better suit their needs of creating a microcontroller that is easy to set up as an autopilot system. Figure 2.1 shows a visual representation of the ArduPilot-Mega 2560, which hereafter will be referred to simply as APM.

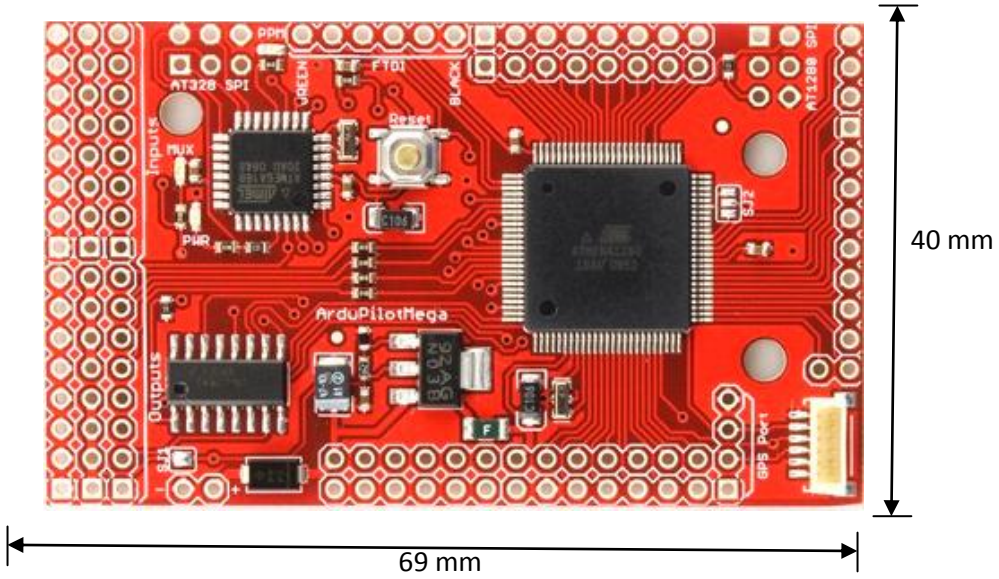


Figure 2.1. APM microprocessor board with dimensions [5].

A nice feature of APM is the RC/servo rail, which easily allows the user to plug in up to eight servos. This is one example of how this board has been outfitted to integrate more easily into a remote control (RC) system over the official Arduino system, where the user would have to install wiring in order to get power to all of the servos. This feature gives APM more of a plug-and-play feel. Other convenient features include a reset button and indicator lights, which are convenient during flight testing.

It is preferred to power the board with 5 volts, although through testing, it has been found that it can be powered off of a four-cell NiMH battery, which has a nominal voltage rating of 4.8 volts. The board is capable of handling 500 ma of current, after which the on-board's resettable fuse will trip. The microcontroller will normally be powered from the RC/servo rail, but it can also be powered directly, if there is a need to isolate the power to APM from the RC system.

Equally important to the autopilot system is the ArduPilot-Mega inertial measurement unit (IMU) shield, which mates with the APM microprocessor and can be seen in Figure 2.2. The inertial measurement unit will be referred to as either the shield or the IMU. This board is equipped with standard sensors needed to create an autopilot system. Most notably on this board are three-axis gyros and accelerometers, giving a full six degrees of freedom (DOF) system, a 16 MB data logging flash memory chip, a micro USB port for easy interfacing with a computer, a pressure/temperature sensor for absolute altitude sensing, and plenty of extra ports to add other digital and analog sensors. After the APM microprocessor and the IMU are assembled, the final product, which can be seen in Figure 2.3, is compact and easily mountable on an RC aircraft

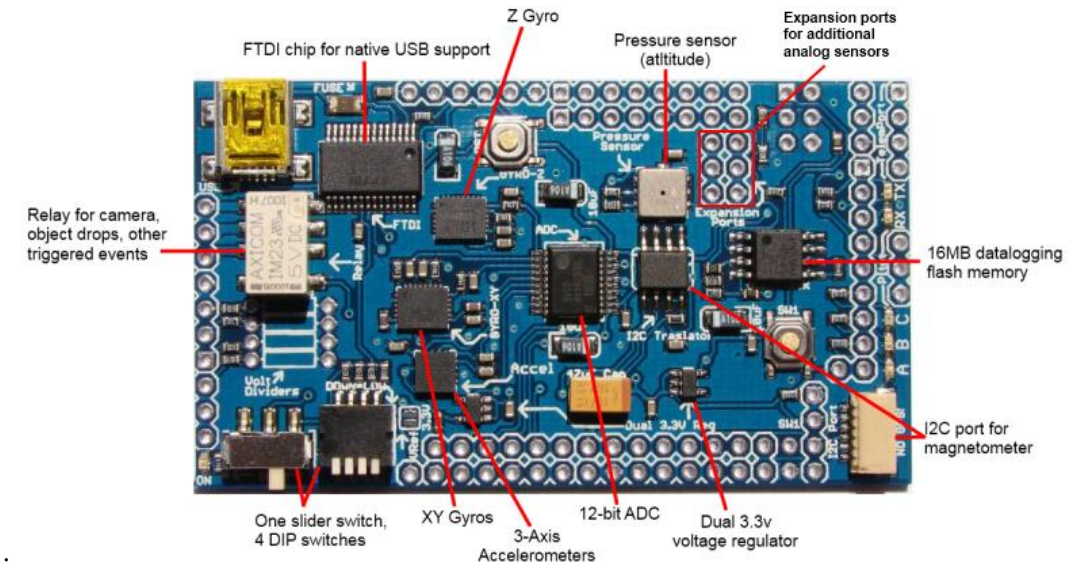


Figure 2.2. IMU board that mates with main microprocessor board [6].

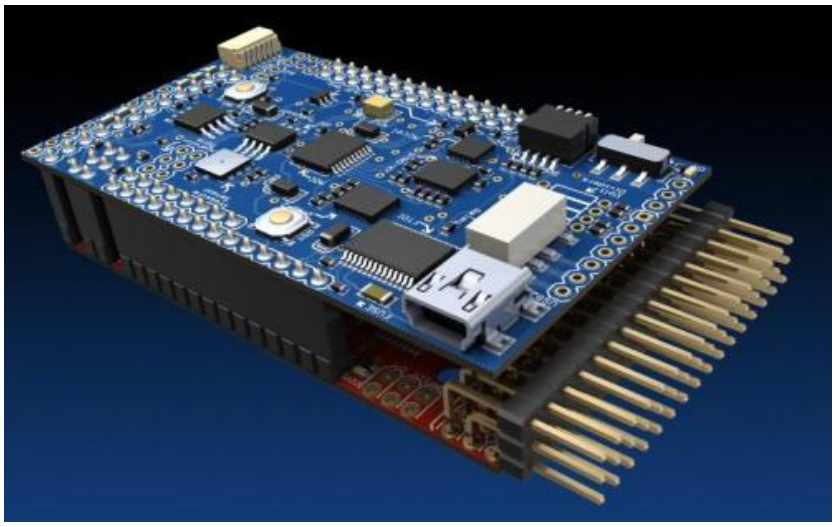


Figure 2.3. Main autopilot system assembled [7].

2.2 Additional Options

Additional options can be added to the autopilot in order to create a more robust system and aid in the learning process. These components will be discussed in the following sections.

2.2.1 Global Positioning System

Sold standard with the APM kit from DIY Drones and an essential part of the autopilot system, the global position system (GPS) allows the autopilot to understand where it is in space and is essential for mission scripting. The APM comes equipped with a six-pin EM406 female connector that allows the user to plug the GPS directly into the board. The standard DIY Drones GPS module used is the MediaTek MT3329 GPS 10Hz .

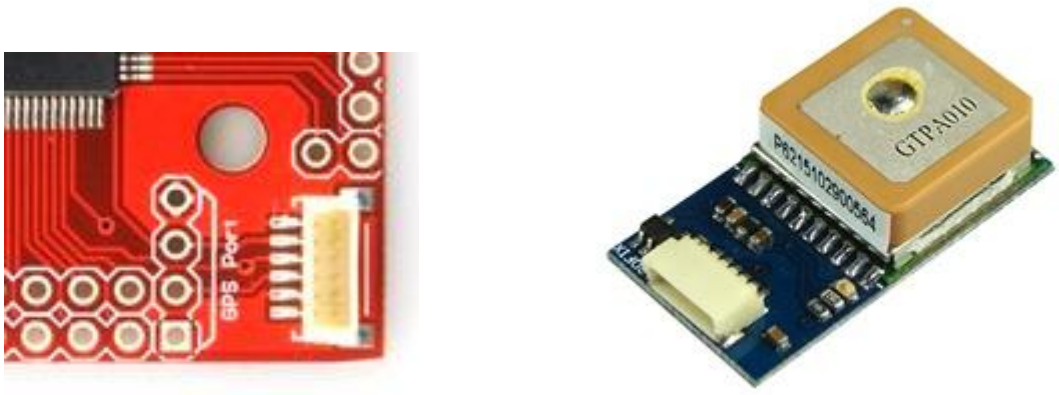


Figure 2.4. Six-pin EM406 female connector on APM board (left) and Media Tek MT3329 GPS module (right) [5, 8].

2.2.2 Airspeed Sensor

The airspeed sensor is a combination of a pitot tube set and a differential pressure sensor. With this airspeed sensor, the autopilot is able to fly by airspeed and not by groundspeed (which is received from the GPS). This provides higher mission flight fidelity, especially in windy conditions. Minimal soldering is needed to connect the sensor to the IMU shield. The standard DIY Drones differential pressure transducer is the MPXV7002.

Figure 2.5. Airspeed sensor components [9].

2.2.3 Voltage/Current Sensor

While flying, it is beneficial to collect data pertaining to the battery performance, such as voltage and current. A few different options with APM do just that. The shield has a built-in voltage sensor that, with a small amount of soldering, will give voltage information. This is a good option, but a better option is to add an external current sensor, which enables the user to measure voltage, current, and cumulative current consumption in milli-amp hours. An example of such a current sensor and how it can be connected to the shield can be seen in Figure 2.6.

Figure 2.6. AttoPilot current sensor (left) and input pins (right) [10, 11].

2.2.4 Wireless Telemetry

The wireless telemetry accessory allows the user to communicate with the autopilot while it is being flown. There are many advantages to being able to communicate with the airplane in flight, some of the most important being logging data directly to a computer, monitoring performance while in flight, giving commands to the

autopilot, and changing missions while in flight. The standard DIY Drones modems used for wireless telemetry are the XBee-PRO 900 extended range modules. The system requires two interface boards: XtreamBee board and XtreamBee USB adapter. The modems operate in the 900 MHz frequency band. Outside of the United States and Canada, 2.4 GHz frequency may have to be used, depending on local laws.

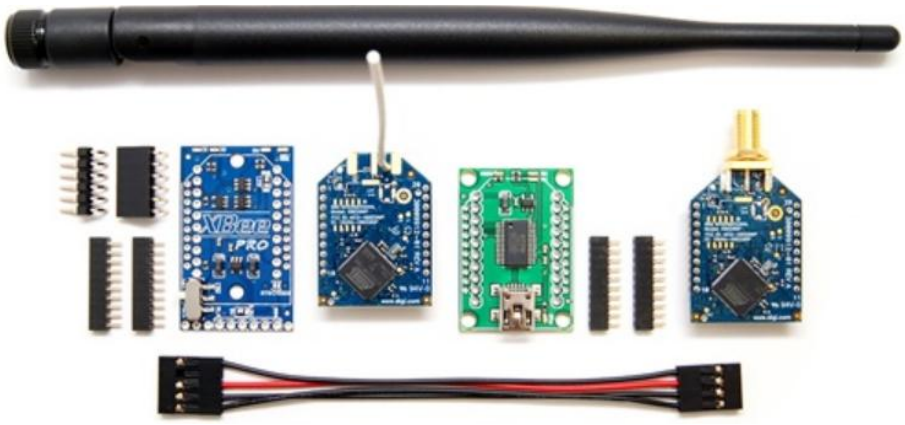


Figure 2.7. XBee wireless telemetry kit sold by DIY Drones [12].

2.2.5 Magnetometer

In layman's terms, a magnetometer is a sophisticated compass, and putting one on an autopilot system gives the aircraft's current orientation not just the moving direction of the airframe. This is an important distinction especially if the aircraft is dealing with crosswinds. Using only the GPS, the autopilot will interpret the plane's moving direction as the plane's facing direction. For example, in an extreme case, if the aircraft is heading into a stiff wind and is actually moving backwards, then the GPS will interpret the flying direction a full 180 degrees in the wrong direction. The magnetometer solves this problem and also takes the reliance off the GPS for fixing sensor drift. The standard DIY Drones magnetometer is the Honeywell HMC5883, which is shown in Figure 2.8.

Figure 2.8. Honeywell HMC5883 magnetometer [13].

Additional sensors can be added, such as an ultrasonic range finder, in order to perform flares on landing. For the purpose of this thesis, the configuration that has been listed will be sufficient for testing an autopilot system capable of completing an entire mission autonomously. More importantly, it is the writer's opinion that this configuration will provide the greatest learning utility, while remaining simple enough and not overwhelming potential students. The total cost of the system adds up to just under $500, as can be seen in Table 2.1.

TABLE 2.1

PRICE BREAKDOWN OF AUTOPILOT SYSTEM

APM + Shield + GPS	$250.00
Telemetry Kit	$150.00
Magnetometer	$44.90
Air Speed Kit	$24.95
Current Sensor	$19.95
Total	**$489.80**

Note: Price in U.S. currency as of January 31, 2012.

CHAPTER 3

DIY DRONES

Knowing who is developing APM will give a better understanding of how the autopilot can be implemented into aerospace academia. DIY Drones is the backbone of what makes the Arduino-based APM special. Based out of San Diego, California, DIY Drones describes itself as "the home for everything about amateur Unmanned Aerial Vehicles (UAVs)" [14]. While DIY Drones supports a number of autopilots and UAV control systems, its main focus is on APM. It boasts that APM is the world's first universal autopilot. It should be noted that APM is able to pilot not just fixed-wing airplanes but also helicopters, multi-motor helicopters (multicopters), ground vehicles, and boats. Airships and submersibles may well be on the agenda for DIY Drones as well. The fact that all of these functions are capable with a single set of hardware is remarkable, but since the intent of this thesis is relative to incorporation into an aerospace environment, the main focus will be on fixed-wing aircraft. It should be noted that while helicopters and multicopters are also aerospace related, it is the author's opinion that they are drastically different from fixed-wing aircraft and should be researched in depth on their own.

Much can be learned about DIY Drones by examining its founder. Chris Anderson is the Editor-in-Chief of *Wired Magazine*, cofounder of GeekDad.com and BookTour.com, and also founder of 3D Robotics. DIY Drones is a not-for-profit company, but its manufacturing arm, 3D Robotics, is a multimillion dollar company. ". . . what Steve Jobs and Steve Wozniak did with the Apple 2, of taking away computing from the governments and putting it in the hands of everybody, we're hoping to do with aerial robotics . . ." (Chris Anderson) [15].

DIY Drones has utilized the rapid popularization of social networking to grow its product. This, along with the clean, simple language of Arduino is what sets DIY Drones apart. Putting out a product for the public to use for free, while still trying to profit off of it is not the typical approach used by most companies. What DIY Drones has assumed is that the social aspect of the product will be the driver and that the product will grow in value as more people use it. By setting the price point of the APM so low, it has encouraged thousands of people to join the community. These people, once tied to the community, have a self interest to see the autopilot grow and become more powerful. With thousands of people looking at the source code every day, DIY Drones has created a snowballing effect in which the autopilot develops more quickly than a typical commercial or military autopilot system would, given the same initial investment. Thanks to this strategy, DIY Drones has rapidly become the leader in open-source autopilots. How many people are involved in DIY Drones? As of February 2012, DIY Drones has

approximately 22,000 DIY members and averages a new member every 43 minutes [15]. Other statistics from this site can be found in Figure 3.1.

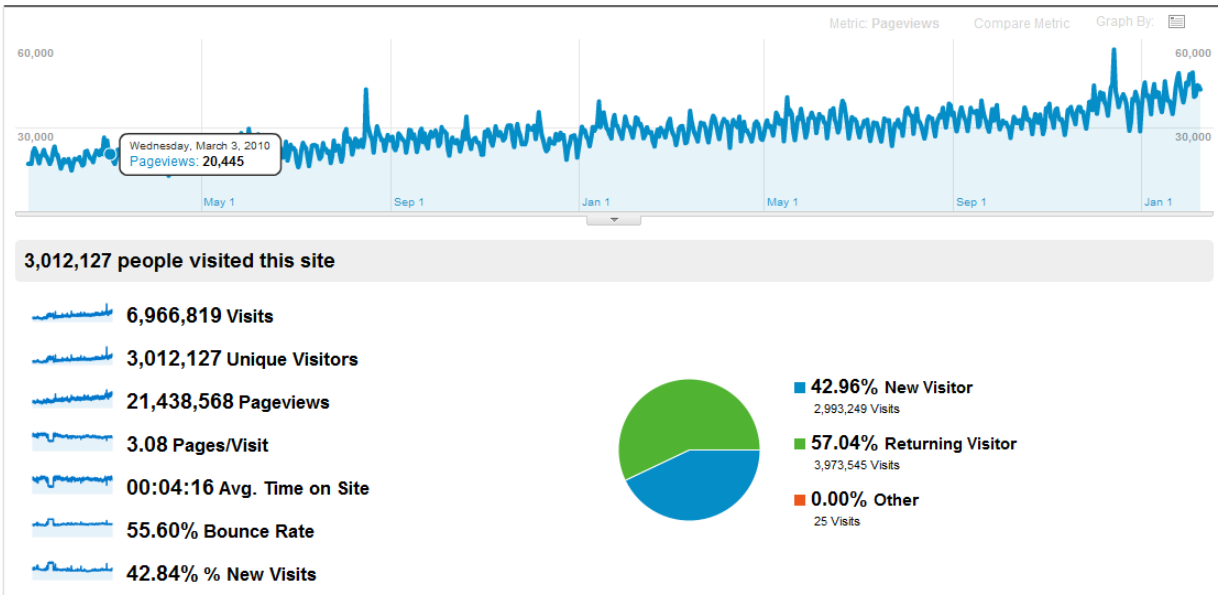


Figure 3.1. Statistical data of DIY Drones website visits as of February 2012 [16].

In trying to incorporate APM into an educational environment, it is important to look at both the positive and negative aspects. While there are many benefits to open-source products, there are some fundamental drawbacks that should not be overlooked. First, open-source projects have difficulty keeping their product's quality from being deluded. With hundreds of people writing and changing code every day, it is difficult to ensure that all changes to the code are beneficial. To mitigate this, DIY Drones has developed a core group of about 25 developers (Dev Team) who are, in essence, quality control. These members have proven to be knowledgeable engineers and are directly responsible for updates to the official DIY Drones core code. This is important for educational purposes. If students are writing code to add to their autopilot system that will execute the specific tasks they need performed, then they should be working with a core code base that has as few flaws in it as possible. If the core code is flawed, then when students encounter problems, they will not know if it is their own code that is to blame or if the error is in the core code. This would most likely lead to students losing faith in the autopilot system and may lead to the demise of teaching with an autopilot system.

Second, most commercial applications undergo rigorous testing before they are released to the public to ensure that all of the bugs are worked out. This is a large expense to the company and a luxury that an open-source project cannot afford. The DIY Drones Dev Team does their best to prevent code with bugs from being released to

the public, but it is an inevitable part of open-source projects. One way to avoid this may be to require students to use a standard version of the core code—one that has been thoroughly tested by a member in the department and has proven to be stable.

Third, there is no "customer support" for open-source projects. When a person has invested in a commercial product, certain expectations are tied to that purchase. If these expectations are not met, then the customer is well within their right to ask for support from the entity from which the product was purchased. This is why most companies have some form of customer support. However, when a product is received for free, there is a lower expectation for such customer support. While DIY Drones' manufacturing arm, 3D Robotics, does provide customer support for the hardware it sells, its software side has none. Instead, the site depends on the DIY Drones community to provide support to each other. In the student's perspective, if there is a problem with the autopilot that the student cannot seem to fix, then that student must rely on the DIY Drones community to help resolve the problem. This actually resembles what happens in most students' classes; if there is a particular course problem that they cannot figure out, then usually they will turn to a friend in the class to help them solve it.

Now that a firm understanding of where the APM autopilot system comes from has been established, the autopilot is examined more closely to see how it works and how well it performs. A full series of simulations and flight testing was run on the autopilot to see whether or not this product is a viable candidate to incorporate into an undergraduate learning environment.

CHAPTER 4

TESTING PROCEDURE

When trying to decide the best way to test the APM autopilot system, a full range of options was assessed as to how to validate whether or not this system would work in an aerospace educational environment. A great deal of thought went into what programs to use to validate the autopilot, what aerodynamics model-building processes to use, what wind tunnel data would be pertinent, what flight dynamics engine would be best suited, and what optimization methods could be used. After much indecision, it was decided to go back and look at the mission of this thesis . . . to find a suitable autopilot system to teach undergraduate aerospace students how to design and build autonomous aircraft. What better way to do this than to design and build a prototype capable of flying fully autonomous missions while using APM, all the while keeping detailed steps of how to accomplish this task. Afterwards a full review of all the data collected was done in order to decide how much benefit students would get from designing with this product.

After reviewing past studies on the APM autopilot, it was decided to go with flight testing rather than theoretical analysis to provide a unique view into the autopilot's capabilities. Detailed information on the theory behind the APM autopilot code has been published, but most flight test information available is scattered and incomplete. Learning from past experience with designing and building remote-controlled airplanes, there is a large leap from theory to the real world, and integrating RC equipment into an aircraft mixed with unpredictable flight conditions can spell disaster.

In order to validate the APM autopilot, an aircraft initially must be designed to act as a test-bed for the autopilot. At first it was thought that it might be beneficial to use a pre-built commercial RC model as the test aircraft, but since the autopilot will eventually be installed on student-designed and -built airplanes, what better way to test the autopilot than on a self-designed and -built aircraft. This will also enable us to tailor the aircraft to best fit flight-testing conditions.

The next step is to run some simulations on the aircraft. This seems contradictory to what was decided earlier, but there is a need to simulate the aircraft and autopilot system before flight testing in order to resolve any problems before the airplane is in the air. The purpose of the simulations in this project is that there will not be any answers derived from the simulations themselves. Instead, the simulations will be used to support the flight testing, which is where the results of the thesis will be derived. Not to run simulation flights first would be foolish and

possibly even dangerous. The makers of APM have developed a way to easily test the autopilot system in a flight simulator, such as X-Plane or Flight Gear, with full hardware-in-the-loop. This means that the airplane can be flown with a transmitter, receiver, and APM all interacting with the flight simulator, giving a scenario that is as close to real flight as possible.

Next, flight testing will begin. During flight testing, a large array of tests will be done to evaluate nearly every aspect of the autopilot system. At this point, most of the data will be collected and examined to determine the fidelity of the APM autopilot. Five main areas will be focused on during these tests:

- **Gain Tuning -** Simulations will be done first to try to determine the initial gains. This will provide a good starting point. Flight tests will then be conducted in order to fine tune the gains more precisely for the actual aircraft. This will be an iterative process, going from simulation to flight test and back again. Results from the flight test may even be used to review the simulation model and, if need be, make changes so that it better represents the actual aircraft. This process should allow convergence to a simulation model and a flight test model that have identical aerodynamic and propulsion characteristics, and near-identical tuned gains.
- **Mission Scripting -** Once tuned, different mission capabilities will be tested. Every aspect will be tested including automatic takeoff, hitting waypoints, automatic landing, etc.
- **Hardware Performance -** Performance will be monitored on how well the instruments communicating with the autopilot are working. Components such as airspeed sensor, current sensor, telemetry, altimeter, GPS, gyros, accelerometers, magnetometer, etc., play a vital role in the unmanned aerial system. All of them are vital to the fidelity of the autopilot, and without some of them, it is impossible to achieve autonomous flight.
- **Ability to Handle Changing Flight Conditions –** During flight testing, there is a likelihood of changing parameters and flight conditions, both planned and unplanned. A good example of this would be a center of gravity shift during flight due to a payload drop, or a shift in wind direction while on approach. Comments will be made on what parameter changes the APM can handle and how well.
- **Data Acquisition -** Possibly the most important part of the autopilot system as a learning tool is that the acquisition of flight data be consistent. It will be determined what flight data can be recorded and to what degree of accuracy. Most likely this will be taken care of while testing the other four parameters.

Once the five parameters are complete, a comprehensive review of the APM autopilot system will be done in terms of incorporating it into aerospace academe. Strengths and weaknesses of the autopilot will be reiterated, and suggestions for implementation into an aerospace learning environment will be made.

CHAPTER 5

FLIGHT TEST AIRCRAFT DETAILS

As stated previously, the reasoning behind a home-designed and -built test airplane is to make an effort to best match the process that a student would go through in designing and building an aircraft to fly. This will provide a custom-built aircraft, specifically designed for flight testing, avoiding fitting electronics into a confined space. Having easy and quick access to all of the electronic components will ultimately save time and give better results. When designing the aircraft, a few parameters were set out in order to better define the conceptual design:

- **Reliability** - Reliability of the aircraft is paramount. If/when there is a problem during flight testing, it is important to know that the problem lies in the autopilot system and not in the airplane. Otherwise, considerable time can be spent solving problems in the autopilot systems that are not actually there.
- **Simplicity** – This could be considered an aspect of reliability but should be emphasized on its own. A simple aircraft will make the entire process of flight testing easier.
- **Flexibility** - It will be beneficial for the aircraft to be able to cover a wide range of performance parameters, giving an overall clearer picture of the autopilot system's capability. For example, the airplane should have a wide range of speed in order to test the autopilot at multiple points in the speed realm.
- **Durability** - As with any flight test vehicle, durability is key. The aircraft should be able to handle multiple flight tests and should have the same flight characteristics on the last day of testing as on the first day.
- **Accessibility** – The aircraft's layout should be such that electronics can easily be accessed in order to make changes to the autopilot system/RC system.

After a selection procedure, Figure 5.1 shows the design that best meets the requirements listed above. It is based on an aircraft that was built for a water-drop competition the previous year and provided to be agile and structurally sound. Using a design similar to a proven aircraft already built should increase the reliability. With a few adjustments to better incorporate the autopilot system, this design should meet all of the requirements stated above. A more in-depth look into the airplane follows.

The main goal of providing details of the flight test aircraft is to allow the reader to clearly view the entire flight-testing process of the autopilot system. The most important information for this purpose is the stability/control specifications of the airplane, but the aerodynamic characteristics, structure, and propulsion system will be discussed as well. These sections will not be as in-depth as the stability/control section, however. For example, the focus of

structures will be on how main components of the aircraft are laid out, and not be on dealing with bending moments and shear stresses. Hopefully, this approach will give the reader plenty of knowledge about the flight test vehicle, without getting bogged down in unnecessary technical details.

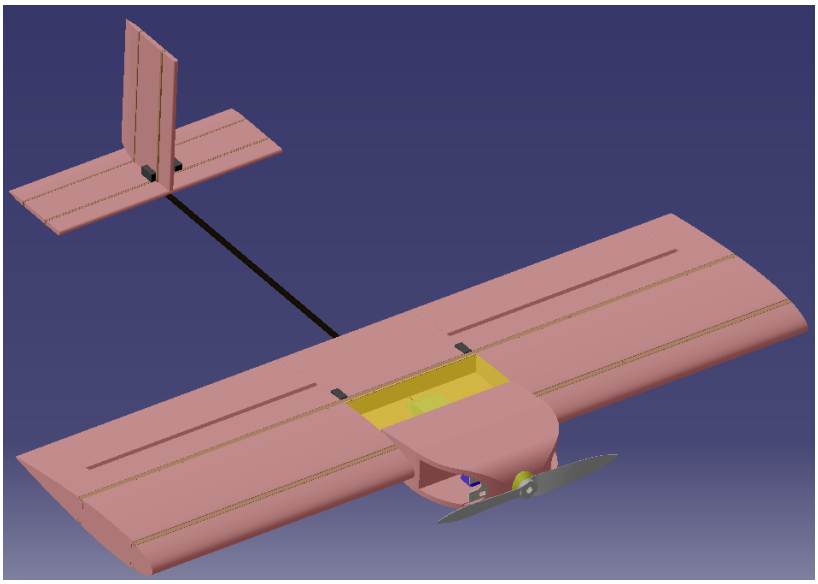


Figure 5.1. Catia V5 drawing of concept selected for flight test vehicle [17].

5.1 Structures

Given that durability was one of the key parameters discussed earlier, the overall structure of the aircraft was designed with that in mind. For this reason, the plane could be considered "over-built" structurally. The airplane's wing and tail are made up of balsa wood and foam, giving a composite structure in the nontraditional sense. This construction has proven to provide a strong, light structure. In an attempt to place the microprocessor and IMU board at the center of gravity in order to minimize vibration noise, a two-spar configuration instead of one was used. Extra care was also taken to ensure that the APM was able to be mounted level in both the pitch and roll axes. Figure 5.2 shows a side view of the center section of the wing with the two-spar configuration and the pocket in the middle for placement of the APM, GPS, receiver, pressure transducer, and XBee modem.

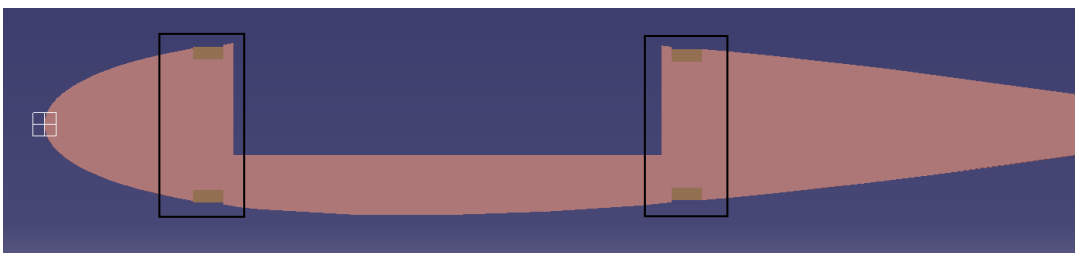


Figure 5.2. Side view of center section of wing [17].

The aircraft was initially designed for hand launch with the ability to add landing gear fairly easily. Not having landing gear will reduce the total weight of the aircraft, and the ability to hand launch will provide more opportunities and places to flight test.

Longitudinally, the airplane is supported by a composite tube running from the motor mount to the tail. This is a quick and easy way to transfer loads throughout the entire aircraft and also minimize the wetted area (area of aircraft contacted by external flow). The diameter of the composite tube (0.3 inch OD and 0.244 inch ID) was chosen based upon handling the torsion produced by the vertical tail in gusty conditions.

Other major components include the front fairing, the outer section wing, and the tail. The motor mount is supported by a front fairing, which is also the housing for the electronic speed controller (ESC), battery, current sensor, and external battery eliminator circuit (BEC). The top of the fairing is removable, allowing easy access to electronic components. The outer section of the wing is hollowed out to reduce weight, with three foam ribs to support the thin foam skin around the cavity. The balsa spar caps extend out to the wing tips. The tail is also made with a combination of foam and balsa wood. All of these major components can be seen in Figures 5.3 and 5.4.

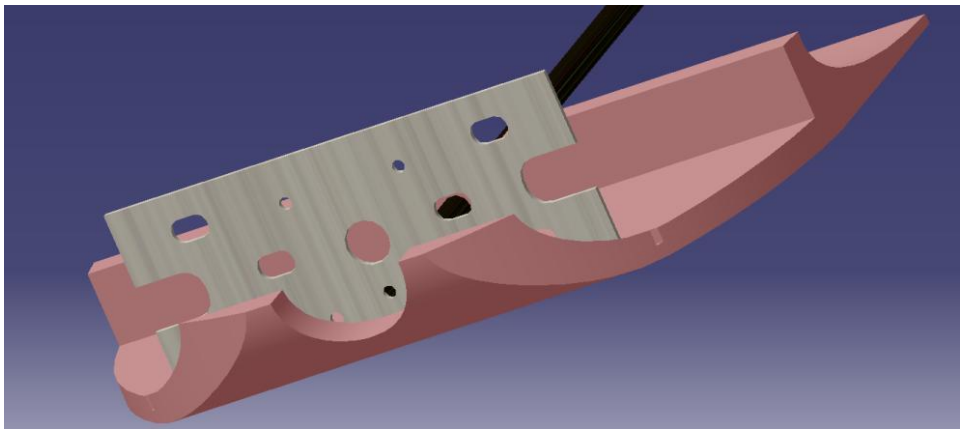


Figure 5.3. Isometric view of front fairing and motor mount [17].

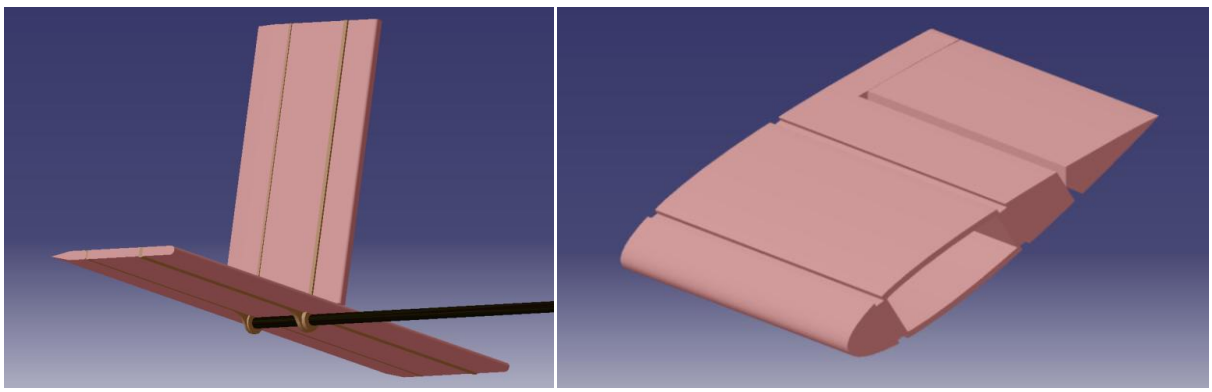


Figure 5.4. Isometric view of tail (left) and outer wing section (right) [17].

5.2 Aerodynamics

Aerodynamically, the aircraft was designed with a wide range of flight tests in mind, preferably able to consistently achieve trimmed flight airspeeds ranging from 9 to 25 m/s. Thanks to the minimalistic nature of the airplane's structure, a low wetted area provides reasonable parasitic drag to be able to meet these speed requirements. Another desirable characteristic is for the aircraft to have gentle stall characteristics. For this reason, a fairly low aspect ratio (AR) of 3.33 was used for the wing. Keeping with the simplicity and reliability approach, no sweep or taper was built into the wing.

The most interesting aerodynamic design feature of the airplane is the NACA 0015 airfoil that was used for the wing. Although cambered airfoils are usually employed when designing an aircraft, using a symmetric airfoil allows the aircraft to have nearly identical flight characteristics when flying upside down as when right side up. This was done in hopes of being able to have autonomous flights while inverted without having to drastically change any trim values or control gains.

5.3 Stability/Control

A majority of the thought processes behind designing this aircraft was spent on stability and controls (S&C). In order to test a large scope of what the autopilot is able to handle, the aircraft needs to be flexible, giving an array of flight characteristics with just a few simple changes. All of these flight characteristics were computed using Mark Drela and Harold Youngren's Athena Vortex Lattice (AVL) program [17]. This software package is a computational program that solves for aerodynamic characteristics linearly using a vortex lattice method. Figure 5.5 is a geometrical view of the aircraft that was programmed into AVL.

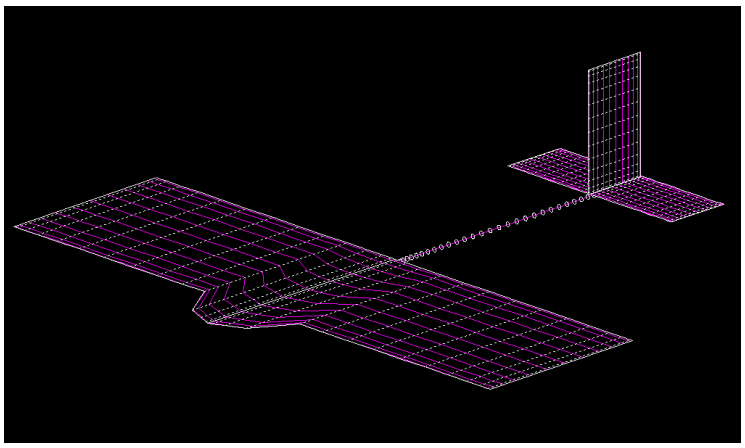


Figure 5.5. AVL geometrical model of flight test aircraft [18].

For initial testing, it is important that the aircraft be fairly easy to control, and therefore a good amount of static margin (~15%) was used. This will make it easy for a human RC pilot to control the aircraft as well as the autopilot. Once the autopilot has proven capable of handling a high static margin case, the center of gravity will be incrementally moved back in order to see how smaller static margins affect control gains and how the autopilot's mission capability is affected. Currently, there is no standard way in the autopilot's core code to measure the angle of attack, so flying an unstable aircraft is not an option. The angle of attack could be incorporated through the use of a vane. It could also be calculated from the z-axis accelerometer, given that the lift-curve-slope of the aircraft is known. For this project, only passively stable aircraft will be tested.

The aircraft was designed to easily move the center of gravity to perfectly set the static margin desired. This was accomplished by designing the front fairing larger than necessary in order to shift the battery pack toward the tail, as shown in Figure 5.6. This movement allows for a variation of the static margin from 15% to 5%.

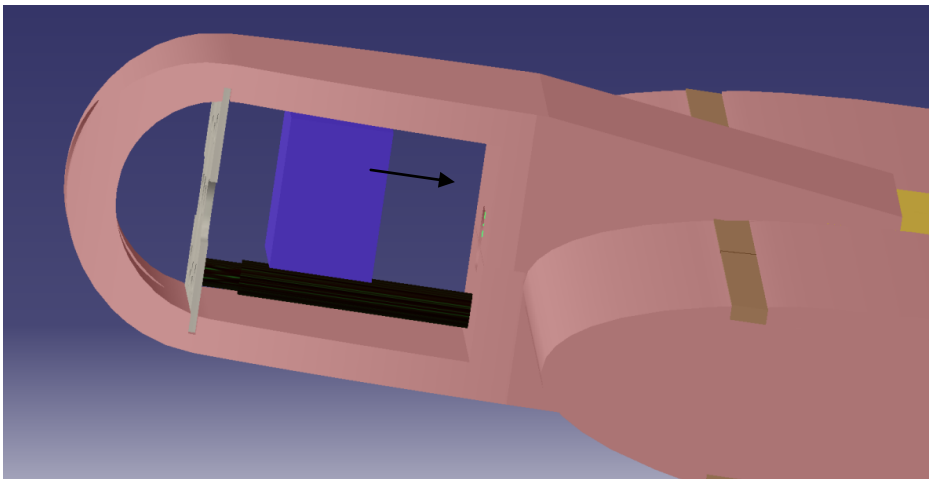


Figure 5.6. Illustration of how battery pack can be moved to change center of gravity [17].

Laterally, the aircraft was designed with no dihedral, thus giving a near neutrally stable aircraft in roll. There is still a small amount of roll stability due to the vertical tail being placed above the center of gravity, but this was necessary in order to land the aircraft without landing gear. The reason for reducing the roll stability was to test if/how well the autopilot could stabilize the aircraft laterally, given poor passive stability characteristics. Not having dihedral also made for a simpler structure.

Mass moments of inertia were taken from Catia and placed into AVL in order to analyze the dynamic modes of the aircraft [16, 17]. Since the aircraft is symmetric, only the Ixx, Iyy, and Izz mass moments of inertia

were entered. See Table 5.1. These moments will also be advantageous when building the simulation model of the aircraft to better match roll, pitch, and yaw rates.

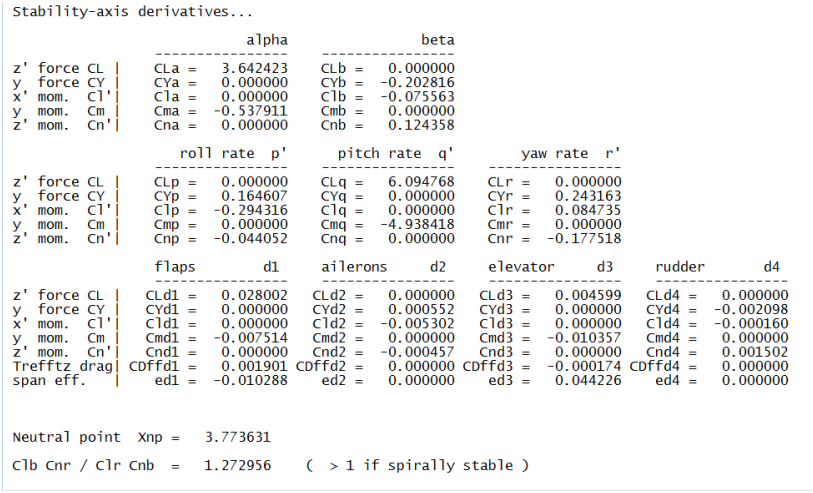
TABLE 5.1

FLIGHT TEST AIRCRAFT MASS MOMENTS OF INERTIA

	Ixx	**Iyy**	**Izz**
kg x m^2	0.147	0.235	0.383
lb x in^2	502.8	804.5	1307.4

Note: The mass moments of inertia are given in two different units and are not different values. The first units were taken out of Catia, and the second units are more user-friendly [17].

Figure 5.7 shows the stability derivatives for the plane taken from AVL [18]. These stability derivatives are for a run case of trimmed flight at 5 degrees angle of attack, with a static margin of 15 percent, and no sideslip or bank angle.



```
Stability-axis derivatives...

                              alpha                beta
                         ----------------    ----------------
z' force CL |    CLa =   3.642423    CLb =   0.000000
y  force CY |    CYa =   0.000000    CYb =  -0.202816
x' mom.  Cl'|    Cla =   0.000000    Clb =  -0.075563
y  mom.  Cm |    Cma =  -0.537911    Cmb =   0.000000
z' mom.  Cn'|    Cna =   0.000000    Cnb =   0.124358

                          roll rate  p'      pitch rate  q'       yaw rate  r'
                         ----------------    ----------------    ----------------
z' force CL |    CLp =   0.000000    CLq =   6.094768    CLr =   0.000000
y  force CY |    CYp =   0.164607    CYq =   0.000000    CYr =   0.243163
x' mom.  Cl'|    Clp =  -0.294316    Clq =   0.000000    Clr =   0.084735
y  mom.  Cm |    Cmp =   0.000000    Cmq =  -4.938418    Cmr =   0.000000
z' mom.  Cn'|    Cnp =  -0.044052    Cnq =   0.000000    Cnr =  -0.177518

                  flaps        d1     ailerons     d2     elevator     d3     rudder       d4
                ----------------    ----------------    ----------------    ----------------
z' force CL |   CLd1 =   0.028002   CLd2 =   0.000000   CLd3 =   0.004599   CLd4 =   0.000000
y  force CY |   CYd1 =   0.000000   CYd2 =   0.000552   CYd3 =   0.000000   CYd4 =  -0.002098
x' mom.  Cl'|   Cld1 =   0.000000   Cld2 =  -0.005302   Cld3 =   0.000000   Cld4 =  -0.000160
y  mom.  Cm |   Cmd1 =  -0.007514   Cmd2 =   0.000000   Cmd3 =  -0.010357   Cmd4 =   0.000000
z' mom.  Cn'|   Cnd1 =   0.000000   Cnd2 =  -0.000457   Cnd3 =   0.000000   Cnd4 =   0.001502
Trefftz drag| CDffd1 =   0.001901 CDffd2 =   0.000000 CDffd3 =  -0.000174 CDffd4 =   0.000000
span eff.   |    ed1 =  -0.010288    ed2 =   0.000000    ed3 =   0.044226    ed4 =   0.000000


Neutral point  Xnp =   3.773631

Clb Cnr / Clr Cnb  =   1.272956    (  > 1 if spirally stable )
```

Figure 5.7 Stability derivatives taken from AVL [18].

5.4 Propulsion

The only requirements for the propulsion system, one of the easier parts of the aircraft to design, was capability of hitting 25 m/s and enough thrust at takeoff to be hand launched. To save time and effort, the propulsion system was not analyzed in depth during the design process. Instead, an existing system that had been used on the same water drop aircraft from the previous year was utilized, with the exception of replacing the nickel-metal hydride battery with a lithium-ion polymer (LiPo) battery. Switching to a LiPo battery will provide the ability to conduct longer flight tests. An external battery eliminator circuit was added to relieve the electronic speed

controller's battery eliminator circuit (BEC) from this duty (the electronic speed controller's BEC was getting very hot, so in order to protect the ESC, an external BEC was used). A folding propeller was used to help with belly landings. All propulsion components can be seen in Figures 5.8 through 5.10, including all RC components used.

Figure 5.8. DX8 Spectrum transmitter (left), Spectrum AR8000 receiver (center), and Castle Creations 10-amp-peak external BEC (right).

5.9. RimFire .10 motor (left), E-flite 3200 mAh LiPo battery (center), and Futaba S3114 micro servo (right).

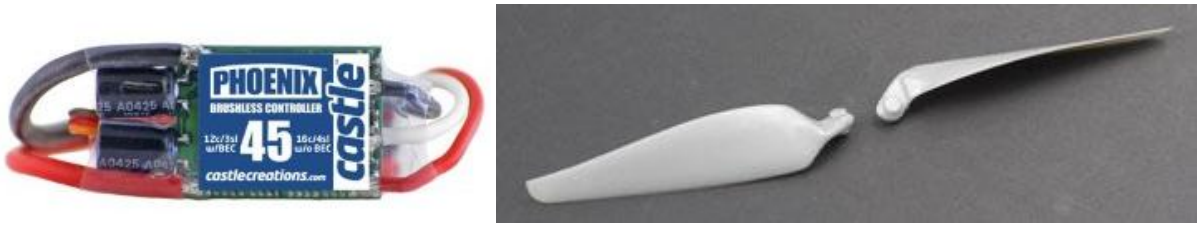


Figure 5.10. Phoenix 45 amp electric speed controller (left) and APC 11x8 folding propeller (right).

The final product can be seen in Figure 5.11. A list of specifications for the vehicle are listed in Tables 5.2 to 5.4. The final product that was developed meets every one of the predescribed requirements, while at the same time being very light in weight (1.91 lbs. ready to fly with all autopilot equipment). The next step in the autopilot testing

phase is to develop a method of simulating flights so that the autopilot can be tested before releasing the vehicle into the air. This method of simulation will be covered in the next section.

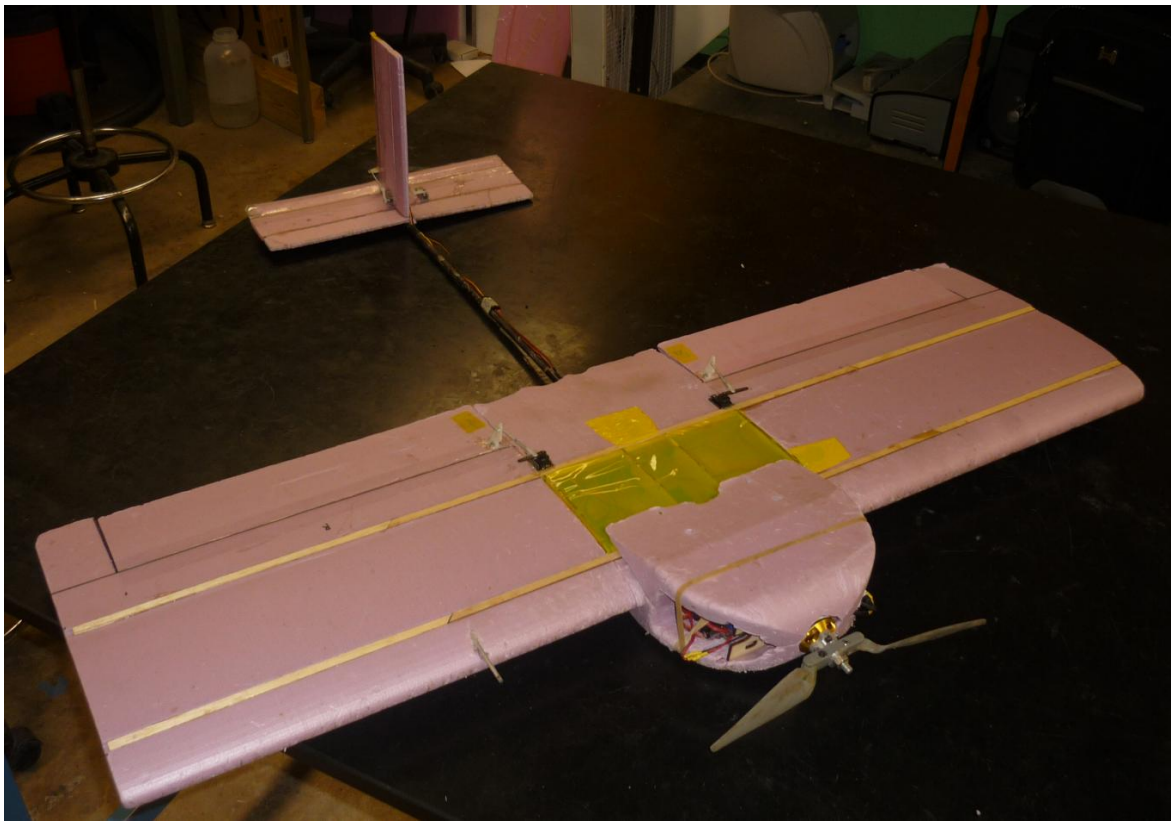


Figure 5.11 Actual flight test aircraft after over 50 flights and landings.

TABLE 5.2

AIRCRAFT PRIMARY DIMENSIONS

Weight (lbs)	1.91
Wing Span (in)	40
Chord (in)	12
Aspect Ratio	3.33
Longitudinal Length (in)	36
Airfoil	NACA 0015

TABLE 5.3

HORIZONTAL TAIL DIMENSIONS

Span (in)	13.76
Chord (in)	4.375
Aspect Ratio	3.15
Airfoil	Flat Plate

TABLE 5.4

VERTICAL TAIL DIMENSIONS

Span (in)	7.78
Chord (in)	4.4
Aspect Ratio	1.77
Airfoil	Flat Plate

CHAPTER 6

SIMULATION MODEL

Simulation is very useful when developing an autopilot system. It allows the developer to perform tests computationally, thus minimizing the inherent risk brought about by an actual flight testing. While there is still a large jump from simulation to actual flight testing, simulating flights will help in obtaining initial gain values. It will also help in testing different mission commands such as takeoff, mission planning, and landing. This will allow the user to eliminate problems and better understand how the actual aircraft will perform during flight testing. For this reason, every flight test performed during this project will be simulated before it is attempted in the air.

There are several different ways to accomplish such a simulation. To reiterate, the goal of the simulations is to most closely replicate a series of real-life flights in simulation format. Several different simulation methods will be discussed in the following sections, because others reading this thesis may want to follow up on a different approach, and the ultimate method that will be most advantageous for this project will be selected. Five essential parts are utilized for the autopilot simulation process:

- The executable autopilot code.
- An aerodynamic model of the aircraft.
- A flight dynamics modeling application.
- A data acquisition utility.
- A visual representation of the aircraft in an environment similar to the actual flying environment.

6.1 Simulation Options

A few different ways can be used to employ the APM autopilot code in a simulation format. Much of the dynamic system modeling done by college students is built in a block-diagram environment, such as MATLAB's Simulink [19] or a free equivalent such as ScicosLab [20]. In using such a program, the main proportional-integral-derivative (PID) gain control loops may be easy to build, but some of the smaller details that are in the autopilot code may be difficult to match perfectly. For this reason, any self-built version of the code is not as attractive an option for what this project is trying to accomplish.

There are two different ways to run the near-identical core autopilot code that will be running on the actual aircraft during flight testing. The first is to run a software version of the autopilot code. DIY Drones has provided a build-up of the autopilot code that uses a C++ compiler. The second is to run the code directly from the APM

hardware, which is known as hardware-in-the-loop (HIL) simulation. Either option works, but this project will rely on HIL simulation. It should be noted that if the HIL option of simulation were to be used in an undergraduate course, it would be essential for every student that is working on the autopilot system to have their own actual APM board, while the software option would allow students to share boards for flight testing and use the software for simulation, which could be a much cheaper option. In any case, for this project, since the board was already acquired, the HIL process will be used.

Now that the method of emulating the autopilot system has been chosen, the aerodynamic model and flight dynamics application needs to be selected. Again, there are several different ways that an accurate aerodynamic model of the aircraft can be formed. Hand calculations are the most basic but, done correctly, can yield good results. The next level option would be to use a linear software package such as AVL. A high-level option would be to run a computational fluid dynamics (CFD) model or perform wind tunnel testing. Values from the aerodynamic model can then be plugged into a program like Flight Gear [21]. Flight Gear is an open-source flight simulator that uses JSBSim [22], an open-source dynamics modeling application, to give both the look and feel that is preferred for the simulation that will be performed. When integrated with the autopilot HIL, it gives a very nice simulation platform, and all of the software is free.

Another option to Flight Gear and the one that will be used on this project is X-Plane [23], a commercially licensed flight simulator that uses blade element theory to model the aerodynamics of an aircraft. X-Plane will be used on this project instead of Flight Gear for a few reasons. One, it is available at the university at which this research was conducted. Two, it combines aerodynamic modeling, flight dynamic modeling, and visual representation all into one application. Three, the undergraduate students and the author of this thesis are more familiar with X-Plane than Flight Gear. If other academic institutions are unable or unwilling to attain X-Plane licenses, then Flight Gear will most likely be the preferred option.

The final component needed for a solid simulation platform is a data acquisition system and a method for integrating the autopilot into X-Plane. DIY Drones has created Mission Planner, a software package for accomplishing both of these tasks [24]. Mission Planner takes simulated flight data from X-Plane and feeds it into the APM. In turn, the APM uses the Mission Planner to communicate back the desired control to X-Plane. APM and Mission Planner also have the ability to communicate with the actual RC equipment while performing the simulation. This means that the actual toggle switches and control-stick movements of the radio control transmitter

can be fully simulated as well. A visual of the flow that accomplishes this simulation can be seen in Figure 6.1. Mission Planner has more features, such as mission scripting as well as being a ground control station, which will be covered in following sections.

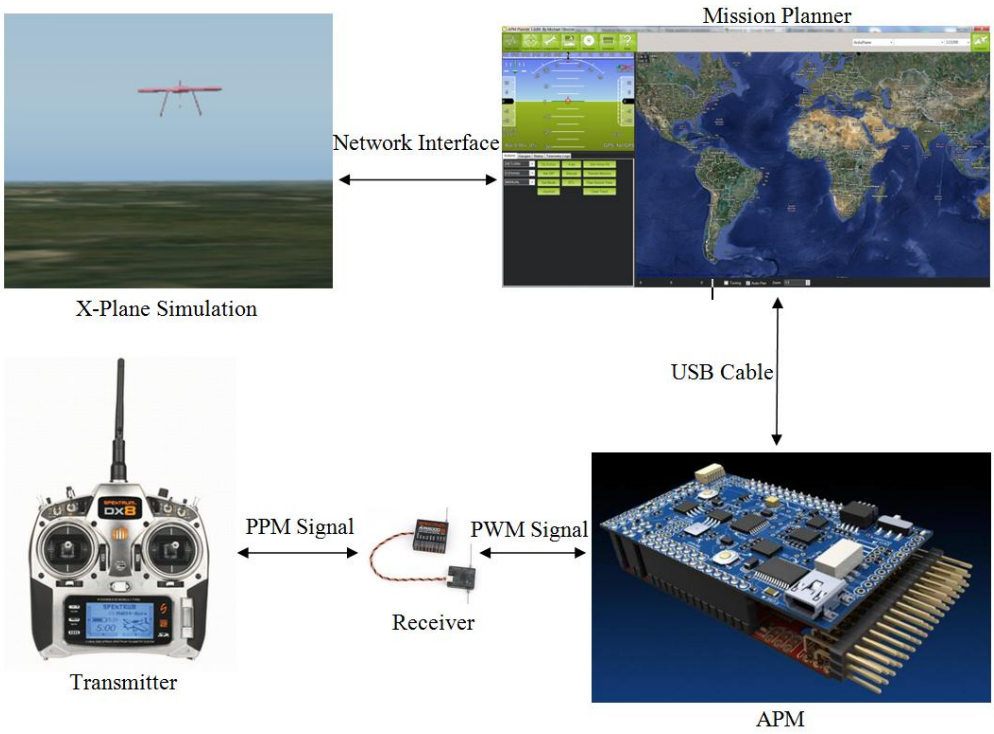


Figure 6.1. Diagram of how X-Plane, Mission Planner, APM, receiver, and transmitter communicate with each other.

6.2 X-Plane Model

Now that X-Plane has been chosen to perform the task of simulation, an accurate model must be built in Plane Maker as well as an accurate airfoil in Airfoil Maker [23]. These two programs come standard with X-Plane and have user manuals provided. A majority of the airplane information entered into Plane Maker is fairly straight forward. In the author's experience, there are three areas where students tend to have problems in X-Plane: over-predicting the propulsion performance, defining arbitrary control geometry and deflections, and using the default mass moments of inertia, rather than finding the actual values on their own. Since an accurate aircraft model is essential to a good simulation and since these three subjects are particularly important for determining initial PID gains, the thought process behind avoiding these frequent mistakes will be discussed.

First, when dealing with X-Plane, many students' main propulsion emphasis is on matching rpm's and thrust values of the propulsion system instead of focusing on matching flight speeds. Plane Maker has several different rpm settings: red line rpm, design rpm of the propeller, design speed of the aircraft, etc. It also has the

ability to give the actual pitch and cord of the propeller that will be used. All of these factors will greatly impact the thrust of the aircraft in X-Plane. However, the emphasis should be on matching the takeoff speed, cruise speed, and maximum speed of the actual aircraft in order to tune PID gains. Putting the emphasis on matching the expected speeds and not focusing on entering perfect rpm values will both simplify the X-Plane design and make it more relevant to what the student is trying to accomplish with the simulation.

Second, many students do not give much attention to providing the accurate control surface deflections, but instead enter arbitrary values into the X-Plane model. Students will often put in a default of ±15 or 20 degrees for the elevator, rudder, and aileron deflections, and 30 degrees for the flap deflection. For some aircraft, these may be the correct settings, but for most aircraft, these default settings are inaccurate. Ordinarily, if a human pilot was flying the aircraft, then this would not be a problem since s/he can adjust how much "stick throw" to use while in flight, or s/he can flip on dual rates through the transmitter. This is equivalent to the pilot changing the control gains while the aircraft is in flight. By changing the gains while in flight, however, any gains that were tuned during simulation will now be inaccurate, because the control surface deflections of the actual aircraft are different than the ones simulated. Engaging the autopilot after the control surface deflections have been changed could cause poor performance or even result in a crash. For this reason, extra care should be given to defining the expected control surface deflection angles in the simulation model.

The last potential issue is with regard to the mass moments of inertia of the aircraft in Plane Maker. Plane Maker does a descent job of estimating what the moments of inertia are (the program actually uses radii of gyration, which can be directly computed from the mass moments of inertia) for most traditional aircraft. These are estimations, however, and should not be presumed to be accurate. This is especially true for more unique aircraft configurations because Plane Maker may have a more difficult time accurately predicting mass moment of inertias for them. Extra time should be spent in finding the aircraft's precise mass moments of inertia. Failing to do this will give the simulation model different pitch, roll, and yaw rates than the actual aircraft, making the gain values found during simulation inaccurate.

Airfoil Maker is a program that allows the user to define the performance of a lifting surface's airfoil. The lift, drag, and pitching moment of the lifting surface with respect to the angle of attack can all be defined here. For this project, Java Foil [25] (a free program that performs airfoil analysis) was used to obtain the 2D airfoil performance of a NACA 0015 airfoil, which was then entered into Airfoil Maker. Once the 2D airfoil is entered,

Plane Maker will help determine the 3D airfoil performance by the user entering the aspect ratio (AR) of the wing as well as the efficiency factor (e). It should not be assumed that Airfoil Maker is perfect and therefore should be double-checked with hand calculations. This step of creating the airfoil is critical or else the simulation will not be accurate enough to obtain good gain values.

If all of these factors are taken into consideration before flight testing, assuming the other more simple characteristics are done correctly, then the success rate of the autopilot being able to pilot the aircraft in the air the first or second flight will drastically increase. The X-Plane simulation model can be seen in Figure 6.2. Now that a suitable simulation model has been created, initial simulations and flight tests can begin.

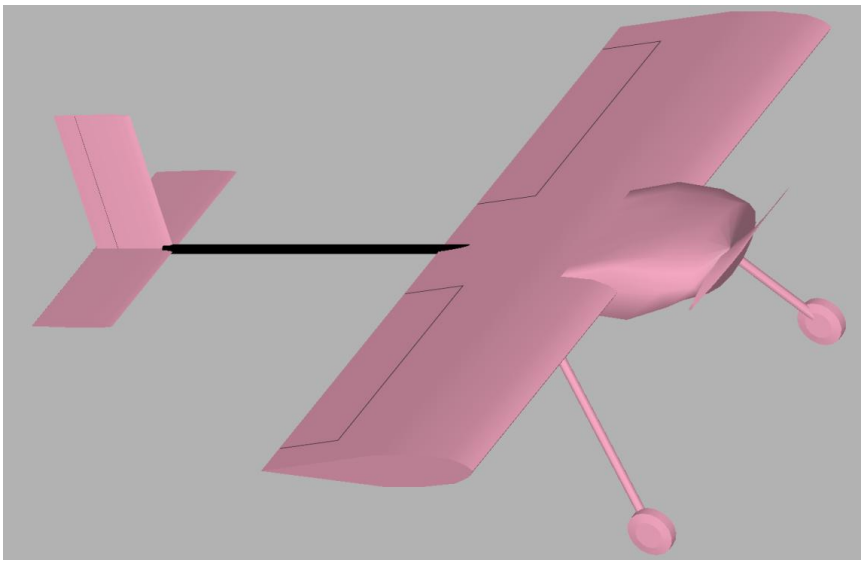


Figure 6.2. Simulation model produced in Plane Maker [22].

CHAPTER 7

MISSION PLANNER

Before starting simulations it would be useful to give a brief description of the Mission Planner utility because it will be relied on heavily throughout the simulation and flight testing process. There should also be a brief discussion of the version of the APM code that will be used. ArduPlane version 2.4 is the code used for a majority of this project. As with any software package, there are often revisions that are released with potential problem fixes and new features, but one of the pitfalls of open software projects is that there is not as much testing and validating done on the code before it is released, compared to proprietary software, due to a smaller budget. For this reason, it was decided to stick with one version of the code throughout the project.

The version of Mission Planner that will be used is Mission Planner 1.0.84. One of the main functions of Mission Planner is to act as a ground station while the aircraft is in flight. Once the XBee modems have been connected to each other, this ground station is able to give a considerable amount of information about the aircraft and the autopilot system while in flight. Figure 7.1 shows a visual of the main ground station, with some of the main features pointed out. This application is also used to see streaming data from the autopilot in both graphical and numerical form, which can be very helpful when tuning the aircraft, as will be explored later. There is also the ability to use "point-and-click" mission control, where the user can control the aircraft's position in the air in real time from a computer on the ground.

Figure 7.1. Screen shot of Mission Planner, which will act as a ground station during flight testing [26].

Another function of Mission Planner is to assist in scripting flight missions. Every part of mission planning, including waypoints, altitude, speed, ascent and descent rate, takeoff, landing, etc., can be controlled here. After the mission is planned, it can be pre-loaded onto the APM before the flight, or it can be loaded onto the aircraft while in flight, given telemetry connection.

The configuration feature in Mission Planner is exactly what it sounds like. Mission Planner provides a graphical user interface (GUI) for setting several parameters, including gains, trim settings, failsafe options, and others. Different configurations can be saved to files called param files, which are simple text format files. Figure 7.2 shows a visual of the configuration GUI.

Figure 7.2. Configuration menu in Mission Planner.

The next two tabs shown in Figure 7.2 are the simulation and firmware tabs. The simulation tab is where the user is able to control the link between the APM microcontroller and the X-Plane or another flight simulator. The firmware tab is where the latest version of the code can be uploaded to the microprocessor board. This can also be done with the Arduino sketch application. Programs built in Arduino are often referred to as sketches. It is beneficial to become familiar with loading the ArduPlane code from an Arduino sketch, because the firmware load from the Mission Planner will only allow an upload of the latest version of the core code.

The last function of the Mission Planner is the terminal. The terminal is simply a serial port communication tool that allows communication with the microcontroller. In the terminal, the user is able to run several tests to check

different sensors, browse and download logs from the onboard data logger, and view the current setting of the autopilot. A similar terminal in Arduino can connect to the autopilot system, but the mission planner terminal is easier to understand for first-time users.

CHAPTER 8

INITIAL GAIN TUNING AND PARAMETER SETTING THROUGH SIMULATION

Now that a suitable simulation model has been established and Mission Planner has been explained, the next step is to begin simulating, in order to better define the appropriate gain settings. In order to do this, objectives must be met:

- Retrieve an initial set of gains in order to control the aircraft.
- Define any other parameters necessary for flight.
- Understand and set failsafe functions.

Chapters 8 and 9 were written with the intention of providing an outline of how to create an autonomous aircraft using the ArduPilot-Mega autopilot and the simulation described earlier. It will provide a step-by-step process that was used during this project to achieve consistent autonomous flight.

8.1 Tuning Gains

8.1.1 Servo PID Gain Tuning

To begin, all of the different servo gains as well as the approach to take when tuning them will be discussed. The PID gains for the pitch, roll, and yaw servos are the starting blocks for controllability, followed by the PID gains for the navigational roll, pitch, and altitude. Figure 8.1 is a simplified version of a PID controller block diagram.

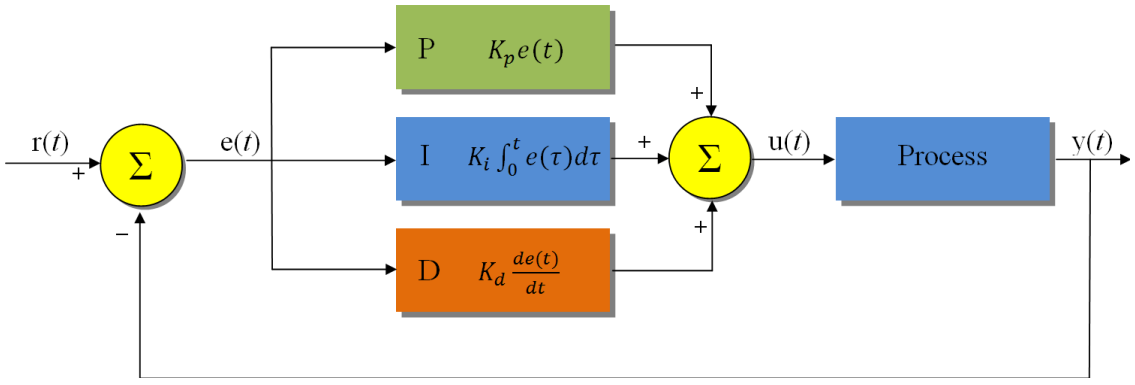


Figure 8.1. Diagram of PID control loop.

In equation form, the PID control loop can be expressed as

$$u(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{d}{dt} e(t) \quad (8.1)$$

$$e(t) = r(t) - y(t) \quad (8.2)$$

A very brief PID gain theory will be given here [27]. The P gain stands for proportional and is the simplest and most important gain of the controller. This gain will be directly multiplied by the current error and, if tuned correctly, may be all that is needed to make the aircraft controllable. As the proportional term is increased, the movement towards the setpoint will accelerate at the expense of an overshoot of the setpoint. The I term stands for integral. This term is multiplied by the integral of the error, which means that it is focused on the sum of the past error. Some controllers with only a proportional gain will produce a steady-state error, but the integral gain is able to eliminate this as well as accelerate the movement towards the setpoint. However, with this acceleration, an overshoot past the setpoint will most likely occur. The D term stands for derivative. This gain is multiplied by the slope of the error over time, which means that it is the opposite of the integral gain in that it is concerned with where the error is going and not where it has been. Increasing the derivative gain will reduce the amount of overshoot produced by the integral term, however, at the cost of slowing the progress toward the setpoint.

As stated previously, the PID gains for pitch, roll, and yaw are the most central part of the controller and the building blocks of controlling the aircraft using the autopilot. The easiest method, and best learning tool, for tuning these gains is to manually tune the gains in the hardware-in-the-loop X-Plane simulation. Optimization techniques could be used as well, but many undergraduate students may not have been introduced to these concepts. Manual tuning will provide an excellent visualization of the different movements and oscillations experienced by the aircraft as different gains are changed. It should be noted that when tuning, it is important that only one variable be changed at a time in order to minimize mistakes and understand exactly what each gain is doing.

The process of tuning is fairly simple and very effective. The same process can be done in each axis; therefore, only the roll gain tuning will be discussed. The default gain values that come on the autopilot can be seen in Table 8.1. These values are a good starting point, and more than likely, the aircraft will be controllable with these default values, but optimizing the gains will give better mission capability.

TABLE 8.1

DEFAULT SERVO PID GAIN SETS

	Servo Roll PID	**Servo Pitch PID**	**Servo Yaw PID**
P	0.4	0.6	0
I	0	0	0
D	0	0	0

Beginning the tuning process, after setting up the HIL simulation, the aircraft was manually taken off and placed in fly-by-wire-A (FBW-A) mode. A list of modes and their functions can be found in the appendix. The throttle was then set to achieve a flight speed of 12.5 m/s, the aircraft's predetermined cruise speed for testing. Moving the transmitter's aileron stick full right produced a rolling moment to the right of the maximum bank angle (the default of which is 45 degrees). Letting go of the stick caused the airplane to roll back into its trimmed flight condition. This is similar to how a car's steering wheel will move back to center after a turn. The gains were manipulated one at a time starting with the proportional gain. Figures 8.2 and 8.3 show a visual of the motion.

Figure 8.2. Aircraft in trimmed flight in X-Plane [22].

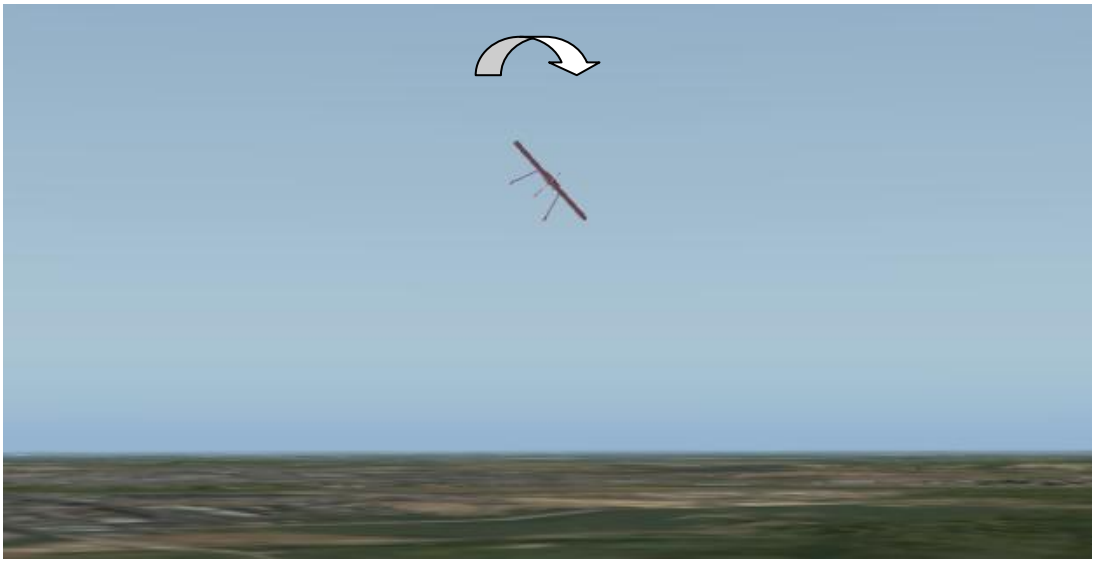


Figure 8.3. Aircraft executing a 45-degree bank in X-Plane [22].

Under the flight data tab in Mission Planner, a graphing utility allows the user to view several flight characteristics while flying, simulating, or playing-back recorded missions. Using this graphing utility, the response characteristics of the aircraft will be monitored to see how each gain change is affecting the autopilot's response. It should be noted that the graphing utility does have a slight flaw. The frequency at which the navigational command is graphed is lower than the frequency of the actual aircraft's response. This gives the appearance that the commanded state is behind the actual movement of the aircraft, which is impossible. Because most students will be using this utility, it will be the graphing utility of choice for this project, despite the flaw.

Performing the 45-degree roll maneuver described earlier gave the response that can be seen in Figure 8.4. As shown, this response yields a few issues. First, there appears to be a steady-state error where the aircraft's roll never reaches the commanded roll. Second, the rise time is about one second. From previous tests performed by a RC pilot, a typical roll rate delivered by the pilot during a semi-aggressive maneuver is about 90 degrees per second. Because the performance of the pilot is meant to be duplicated, the rise time delivered by the autopilot needs to be quicker. In order to fix the slow rise time, the proportional gain was increased to 0.6, as shown in Figure 8.5.

Figure 8.4. Servo roll gain set of P = 0.4, I = 0, D = 0 performing roll of 45°.

Figure 8.5. Servo roll gain set of P = 0.6, I = 0, D = 0 performing roll of 45°.

As can be seen in Figure 8.5, the rise time has now gone to about 0.7 seconds, but there is an overshoot. Also, the problem of steady-state error still exists. The next step is to turn on the integral value, the results of which can be seen in Figures 8.6 and 8.7.

Figure 8.6. Servo roll gain set of P = 0.6, I = 0.1, D = 0 performing roll of 45°.

Figure 8.7. Servo roll gain set of P = 0.6, I = 0.1, D = 0 showing removal of steady-state error.

As can be seen from these graphs, the addition of the integral term has done two things: decreased the rise time further and removed the steady-state error. However, now there is a large overshoot, and the error correction is sluggish. These issues will be handled one at a time. Looking at Figure 8.8, as expected, raising the integral value increases the overshoot, but the error is canceled out much more quickly.

Figure 8.8.Servo roll gain set of P = 0.6, I = 0.2, D = 0 performing roll of 45°.

Next the derivative gain value will be manipulated in order to decrease the overshoot. Turning on the derivative term significantly reduced the overshoot as well as increased the rise time, as shown in Figure 8.9. Also, it has affected how quickly the steady-state error is canceled out.

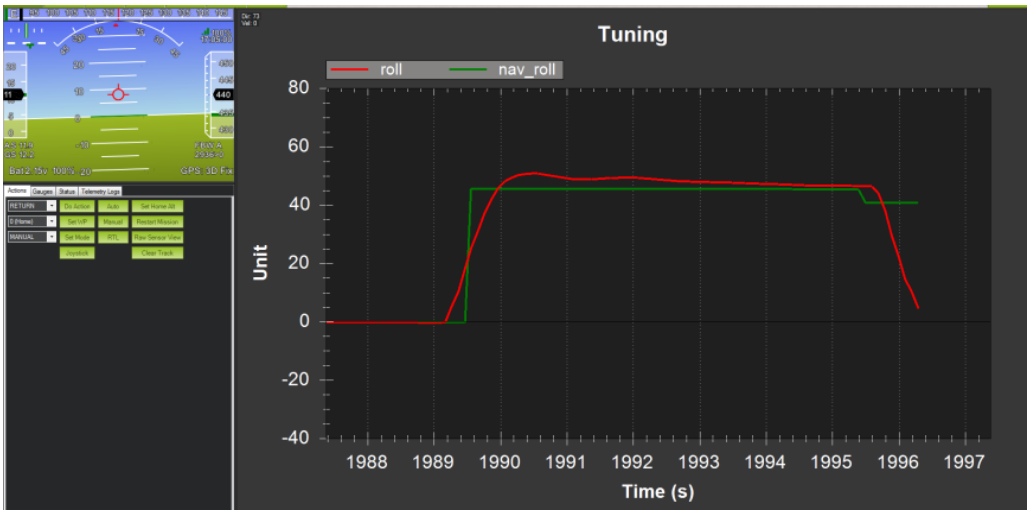


Figure 8.9. Servo roll gain set of P = 0.6, I = 0.2, D = 0.1 performing roll of 45°.

Now that a clear understanding of how all of the gains will affect the roll, they will be adjusted until an optimal performance is found. As with any optimization, this is a give-and-take process, and the gains will be adjusted for those performance characteristics that are most important at the sacrifice of other performances. For this aircraft, agility is important, since many of the missions will be in confined spaces. For this reason, the aircraft's maximum roll will also be increased from 45 to 55 degrees. Figure 8.10 shows the optimized gain selection for the aircraft with respect to the simulation.

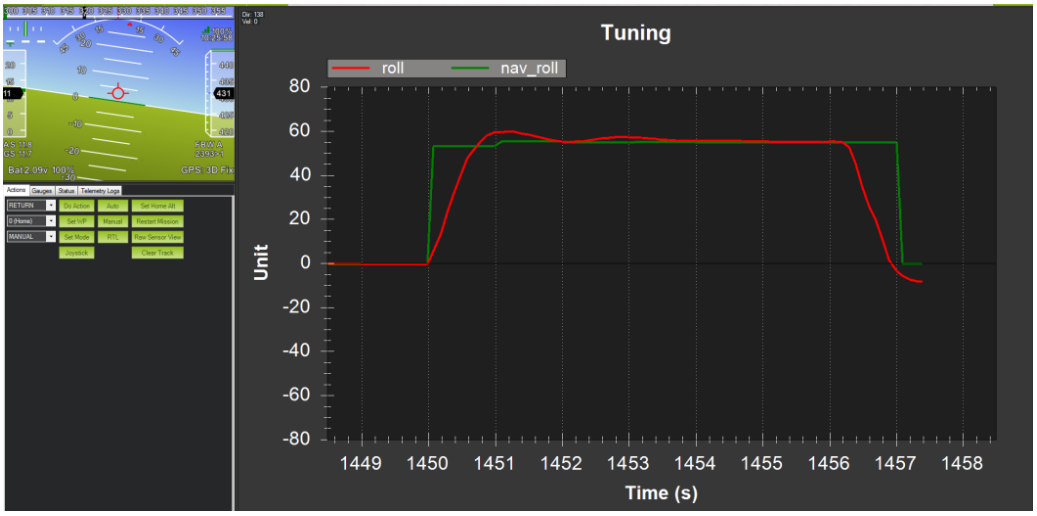


Figure 8.10. Servo roll gain set P = 0.7, I = 0.4, D = 0 performing roll of 55°.

The end result is an aircraft with about a 0.6 second rise time to a 55 degrees roll, with minimal overshoot, and quick error correction, at the expense of some small oscillation. This same process is repeated for the pitch axis. For now, the servo yaw PID gain set will be kept at zero, because the yaw can be controlled by the rudder mix gain that will be described in Section 8.1.6. Table 8.2 displays the values for the pitch roll and yaw PID servo gains. The next step is to see how well the airplane performs in an actual flight test with these gain settings, but first the other gains and parameters will be discussed.

TABLE 8.2

OPTIMIZED SERVO PID GAIN SETS

	Servo Roll PID	**Servo Pitch PID**	**Servo Yaw PID**
P	0.7	0.7	0
I	0.4	0.25	0
D	0.12	0.05	0

8.1.2 Pitch Compensation Gain

The pitch compensation gain is very important to maintaining altitude through turns. The goal of this parameter is to pitch the plane up relative to how much roll is induced in a turn. The default gain entered for this is 0.2, but a simulation will help to better tune this gain. Putting the autopilot in FBW-A mode, the airplane was rolled to about 45 degrees and left there, causing a loitering motion. Keeping an eye on the altitude and the pitch angle, the gain was adjusted until the aircraft was neither loosing nor gaining altitude. The pitch compensation gain that was found to maintain altitude for this aircraft was 0.175. This gain can be very useful, especially in loiter missions. In very tight turns where high degrees of roll angles are used, the pitch compensation gain will not be able to keep the aircraft from losing altitude, but most likely these high roll angles will not be held for very long. One other thing to note about the pitch compensation gain is that it will tend to produce some oscillation in the pitch axis when coming back to center, as can be seen by the blue line in Figure 8.11. If this is unacceptable for the mission being performed, it is best to lower the gain and accept the altitude loss.

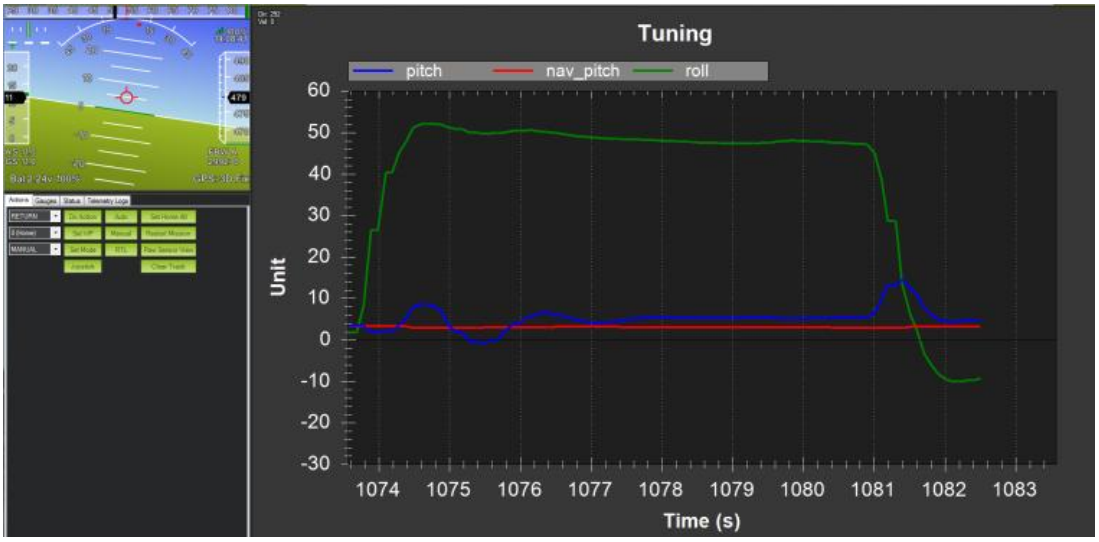


Figure 8.11. Graph of pitch reacting to changes in roll.

8.1.3 Navigational PID Gain Tuning

The two navigational PID gain parameters that will be examined are the navigational roll and the airspeed-to-pitch parameters. The third, which is the altitude-to-pitch parameter, is only used if the airspeed sensor is absent. In order to tune these gains, the autopilot function must be employed, which requires scripting a mission. The first mission that will be simulated is a four-corner box mission, as shown in Figure 8.12. All waypoints are at the same altitude, and the distance between waypoints is 140 meters. The navigational roll gains will be tuned first. Figures 8.13 and 8.14 show the navigational roll commanded by the autopilot and the response of the aircraft, respectively.

Figure 8.12. Autopilot completing box misssion with default navigational roll values.

Figure 8.13. Turn at waypoint 3 with navigational roll gain set of P = 0.7, I = 0, D = 0.02.

Figure 8.14. Turn at waypoint 3 with a navigational roll gain set of P = 1, I = 0, and D = 0.02.

As can be seen in Figure 8.12, the default gain does a good job of keeping the aircraft on course. There is a slight overshoot of the path after the turn, but this is mainly due to crosstrack error, which will be explained in a later section. A comparison of Figures 8.13 and 8.14 shows that turning up the proportional gain of the navigational roll increases the aggressiveness of the maneuver. The navigational roll integral and derivative gains work similarly to that of the servo PID gains. For the initial test flights conducted for this project, the default navigational roll gains should give adequate maneuverability. These gains will be tuned according to the mission thereafter.

TABLE 8.3

NAVIGATIONAL ROLL PID GAIN SET

P	0.7
I	0
D	0.02

The second gain set to tune is the airspeed-to-pitch parameter. This parameter corrects for airspeed by using pitch. Its goal is to always keep the aircraft at cruise speed. Note: As with most autopilots, altitude is controlled by speed. This means that if there is a desire to gain altitude, the autopilot will tell the aircraft to speed up, and vice versa for descending. If the airspeed increases, the airspeed-to-pitch parameter will pitch the aircraft up in an attempt to slow the plane down. The overall effect that is achieved is the aircraft speeding up and pitching up in order to gain altitude. The opposite of this scenario, during a command to descend, the aircraft will slow down and pitch down. In order to simulate this tuning sequence, the same mission as above will be used.

As can be seen in Figure 8.15, provided the altitude commanded is not wildly different than the altitude at which the aircraft is currently, the pitch will adjust until an airspeed of 12.5 m/s has been reached. If the altitude commanded is much larger or smaller than the current altitude, then the altitude parameter will be weighted much more heavily than the airspeed, speeding the aircraft up to gain altitude, or slowing it down to decrease altitude.

Figure 8.15. Airspeed and pitch angle at default airspeed-to-pitch gain set of P = 0.7, I = 0, D = 0.

The easiest way to test the pitch response is to do the mission above and allow the autopilot to get into a comfortable state (airspeed = desired cruise airspeed, altitude =altitude commanded). With the default gains, there

was found to be some steady-state error, where the airspeed never settled to the desired cruise airspeed, so an integral value of 0.1 was added. Noticing that after every turn the aircraft took too long to settle back down to a comfortable state, the proportional gain was turned up to 1.5. This change increased the settling time but also created higher amplitude oscillation. To dampen the oscillations, a D value of .1 was added. This process of tuning was repeated until optimal values were found, as shown in Table 8.4.

TABLE 8.4

NAVIGATION PITCH-TO-AIRSPEED PID GAIN SET

P	1.65
I	0.1
D	0.075

8.1.4 Energy/Altitude PID Gain Tuning

This set of gains determines how much effort the throttle puts into controlling the altitude of the aircraft. In order to perform a simulation for tuning this set of gains, a mission similar to the one used earlier will be performed. The one difference in this mission is that waypoints 1 and 3 will be set to an altitude 10 meters above waypoints 2 and 4. This will cause the aircraft to ascend and descend between waypoints, and the throttle to do more work.

As can be seen from Figure 8.16, the autopilot will vary the throttle in order to attain the desired altitude. Increasing the proportional gain will make the altitude more precise, but it will also increase the fluctuation of the throttle. The easiest way to show this is by turning the P gain up and listening to the sound of the motor. If the gain is too high, the motor will have a pulsing sound. This could cause unnecessary ware on the motor and is an inefficient use of the battery. For these reasons, the proportional gain value will be turned down from the default. Simulation showed some steady-state error from the desired altitude, so the integral gain value was turned on. From Figure 8.17 it can be seen that even a small addition of the integral gain value causes oscillation. Even smaller amounts of integral gain still caused over-revving during flight, so it was decided to keep the value zero and live with the steady-state error. Since there is virtually no overshoot or oscillation, the derivative gain value was kept at zero as well. The optimized gains can be found in Table 8.5.

Figure 8.16. Throttle output while climbing from waypoint 2 to 3 with default gain set of P = 0.5, I = 0, D = 0.

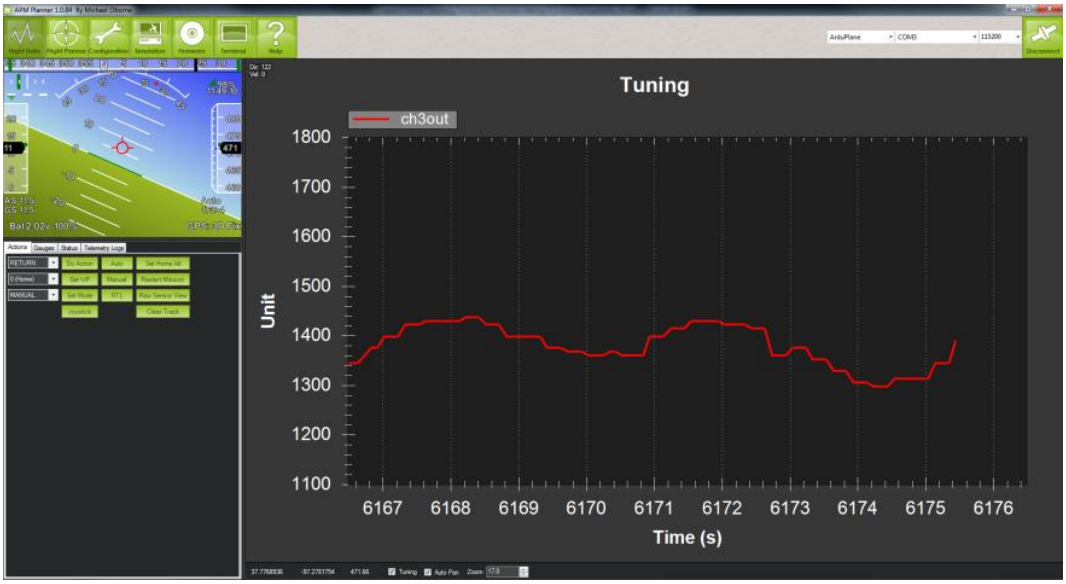


Figure 8.17. Throttle output while climbing from waypoint 2 to 3 with gain set of P = 0.5, I = 0.1, D = 0.

TABLE 8.5

ENERGY/ALTITUDE PID GAIN SET

P	0.4
I	0
D	0

8.1.5 Pitch-to-Throttle Compensation

Pitch-to-throttle feed-forward gain is a parameter that will increase the throttle when the aircraft is pitching downward and will decrease throttle when the aircraft is pitching upwards. When testing this gain, it appeared to be directly fighting with the airspeed-to-pitch navigational gains, causing an oscillatory reaction in pitch. For this reason, this gain was kept at zero for this aircraft.

8.1.6 Rudder Mix

The rudder mix gain mixes the rudder with aileron, giving a coordinated turn. The default is a rudder mix of 0.5. Changing this parameter will determine how aggressive the rudder will be when a roll is commanded. The higher the rudder mix, the more aggressive the rudder. In Figures 8.18 and 8.19, channel 4 corresponds to rudder output, and channel 1 corresponds to aileron output.

In comparing these two figures, turning up the rudder mix gain causes channel 4 (rudder) to produce a higher magnitude output. These graphs also show that a rudder mix of 0.5 corresponds to the ailerons and rudder outputting the same magnitude, while a rudder mix of 1 causes the rudder to produce an output that is twice that of the ailerons. The flight test vehicle does not have a very large rudder, and therefore most of the turn is achieved through the ailerons. For this reason, the rudder mix gain will be left at its default value of 0.5 until flight testing.

Figure 8.18. Rudder mix of 0.5 with arbitrary roll induced.

Figure 8.19. Rudder mix of 1 arbitrary roll induced.

8.1.7 Xtrack and Entry Angle

Two very different approaches are used to deal with the Xtrack gain. This gain, when turned up, causes the aircraft to follow the line in between two waypoints. For some missions, it may be important to follow a specific path. For example, if flying in between light poles or trees, it is probably a good idea to have the Xtrack gain turned up. On the other hand, if flying a mission where unneeded motion is detrimental, an Xtrack of zero may be the most beneficial. Figures 8.20, 8.21 and 8.22 show different Xtrack settings.

The entry angle is the roll angle that the autopilot will not exceed in order to get the aircraft onto the flight path (yellow line). Lowering the entry angle from 30 to 10 degrees for this mission has removed most of the snaking action that occurred at the higher entry angles. Eliminating the Xtrack altogether creates a more circular path and makes the turns at the waypoints less dramatic. A constant Xtrack value and entry angle will not be chosen at this point. Instead, it will vary with the mission that is being flown. Now that all of the gains have been argued, several other parameters must be discussed, some of which are just as important if not more so than the gains.

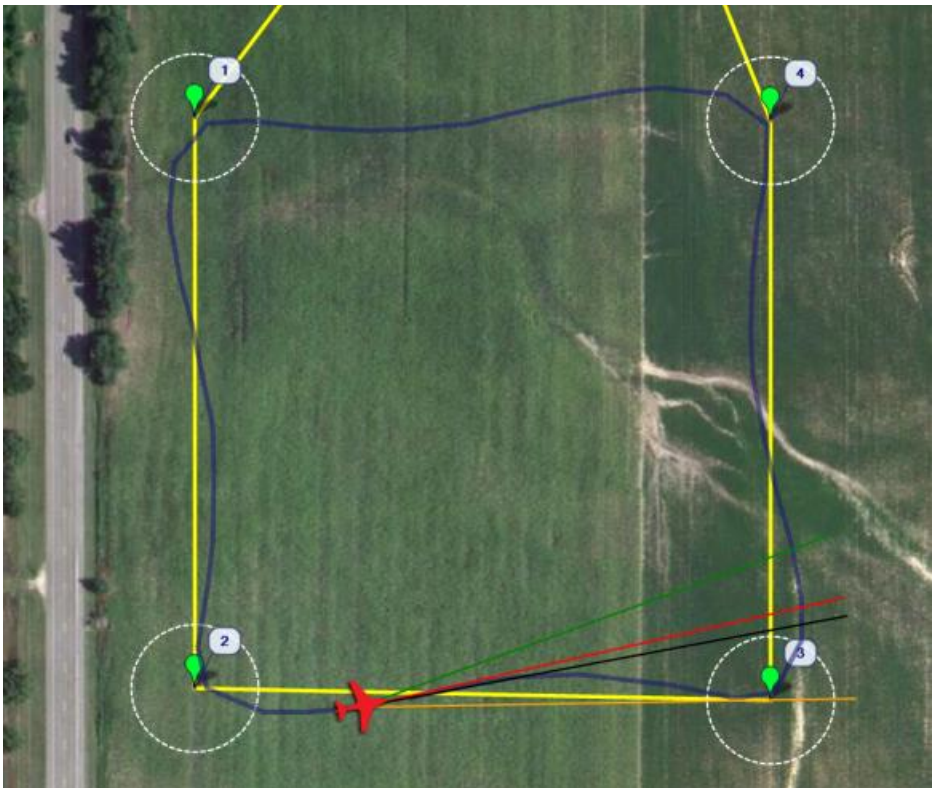


Figure 8.20. Box mission performed with Xtrack of 100 and entry angle of 30°.

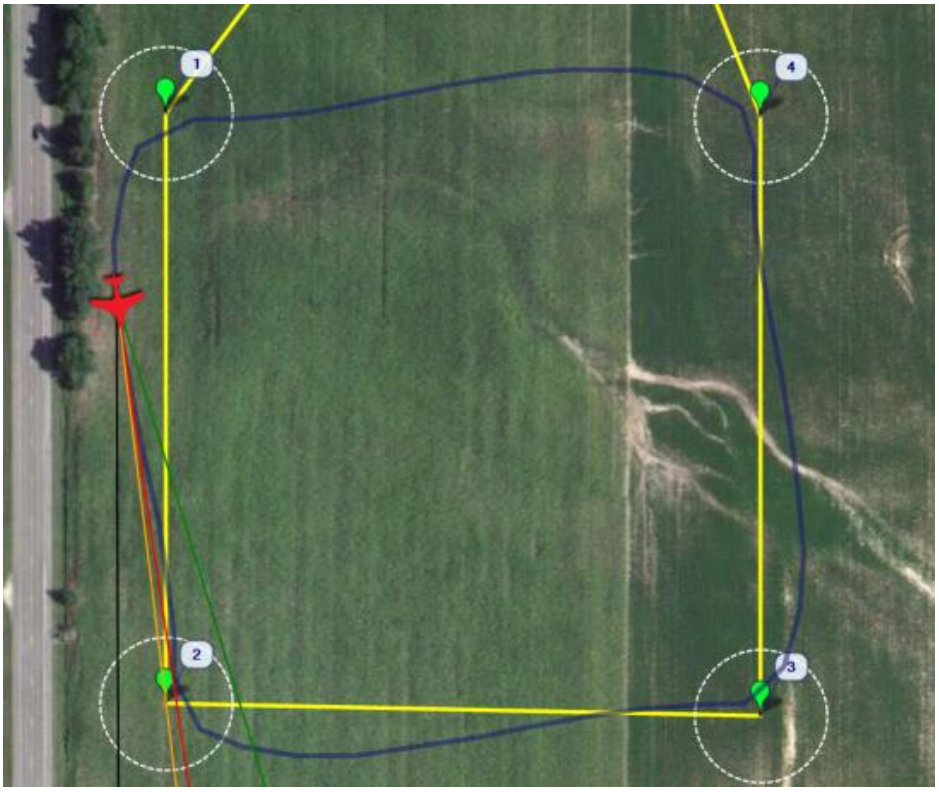


Figure 8.21. Box mission performed with Xtrack of 100 and entry angle of 10°.

Figure 8.22. Box mission performed with Xtrack of 0.

8.2 Other Parameters

8.2.1 Throttle Parameters

Throttle parameters allow the user to decide the limits of the aircraft's propulsion system while being auto-piloted. The throttle cruise value is used as a reference point in order for the autopilot to better understand what throttle to give while completing a mission. The flight test aircraft will use a 30 percent throttle cruise value. The throttle minimum value is the lowest throttle setting that will be applied while in autopilot mode, assuming that a landing has not occurred, in which case the autopilot will sense the lack of airspeed and groundspeed and will cut the throttle. The flight test vehicle has a good power-off glide slope, so the minimum throttle setting will be set to zero. The throttle maximum value is the highest setting that will be allowed while in autopilot. This is good for limiting over-powered aircraft. The flight test vehicle is slightly over powered, so the maximum throttle will be limited to 70 percent.

8.2.2 Navigation Angles

Briefly discussed during the servo PID gain tuning section, the navigational angles are the maximum and minimum angles that the aircraft can pitch or roll when in FBW-A, FBW-B, or autopilot. Already having changed the maximum roll angle from 45 to 55 degrees, the pith angles will be left to their defaults for this flight test aircraft.

8.2.3 Airspeed Parameters

The three airspeed parameters are cruise, FBW min, and FBW max. The cruise parameter is used when in autopilot mode and is the speed at which the autopilot tries to maintain while completing a mission. The FBW min and max are used when in FBW-B mode. While in this mode, the lowest throttle setting on the transmitter will correspond to the FBW min value, while the highest throttle setting will correspond to the FBW max value. Table 8.6 provides values for the airspeed parameters.

TABLE 8.6

AIRSPEED PARAMETERS

Function	Speed m/s
Cruise	12.5
FBW_min	8.0
FBW_max	20.0

8.2.4 Altitude Hold Return to Launch

The return-to-launch function is a handy mode that has been added to the autopilot system. When this mode is activated, the autopilot will fly to home and loiter until commanded otherwise. Home is set when the autopilot initializes on the ground, although it can be set manually as well. The altitude hold return-to-launch parameter is the altitude that the user specifies the autopilot to keep while the aircraft is in return-to-launch mode. The way the autopilot works is when the return-to-launch mode is activated, the aircraft will head towards home and come to the altitude specified. It will take the most direct path to home; therefore, if the aircraft is being flown where there are obstructions, such as trees or power lines, the altitude should be set fairly high. For most flight test missions, the altitude hold return-to-launch parameter will be set to 50 meters.

8.2.5 Altitude Mix

This fairly straight-forward parameter allows the user to decide the way in which they would like to measure altitude. The two options are through the GPS or through the on-board barometric pressure sensor. Using a value of 1 will specify using pressure, while a value of 0 will tell the autopilot to use the GPS. The altitude can also be mixed. A simple example of this is if the GPS is reading 100 meters and the pressure sensor is reading 110 meters, then the autopilot will be fed an altitude of 105 meters, given an altitude mix of 0.5. The following test flights will use an altitude mix of 1, relying solely on barometric pressure.

8.2.6 Battery Monitor

A few different options will allow the user to monitor the performance of the battery. Option one is to use the built-in voltage sensor that comes standard on the APM IMU board. The easiest way to do this is to connect the balance plug that comes standard on most LiPo batteries to the voltage sensor pins on the APM shield. This method could be a problem, because many student flying competitions such as AIAA's Design/Build/Fly currently do not allow students to use LiPo batteries for safety reasons. Also, this option does not provide the current draw, and therefore cannot provide how much battery has been used during the flight. For this reason, option two, which is the addition of a current sensor, will be used. Using this option will provide both the ability to use nickel-metal hydride or nickel-cadmium batteries and to give the current and mill-amp hour consumption. The current sensor used on the test vehicle was outfitted with deans plugs so that it is easily removable.

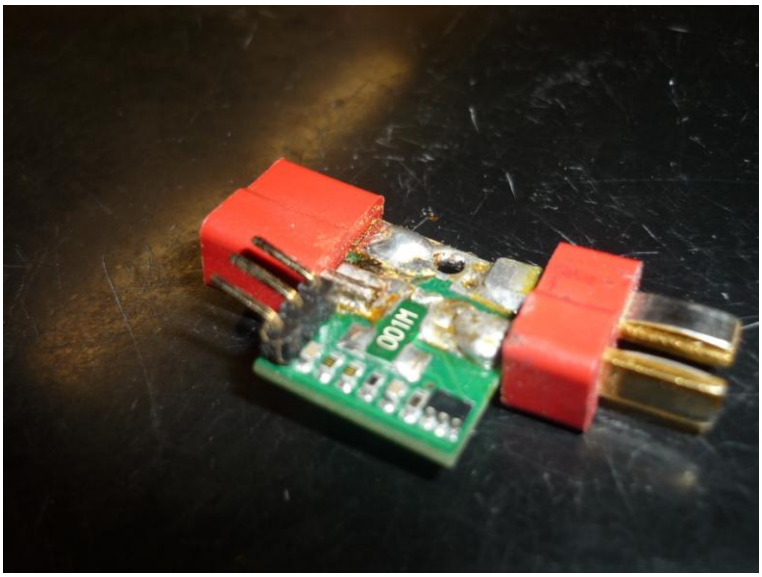


Figure 8.23. Current sensor with deans plugs soldered on.

8.2.7 Battery Capacity

This parameter is the capacity of the battery being used in mill-amp hours. For the most part, a value of 3200 will be used, which corresponds to the main battery that will be used for flight testing.

8.2.8 Compass Declination

The makers of Mission Planner have made it easy to determine and enter the magnetic declination of the aircraft's current location. The reason behind this correction is that in most places in the world, true north is not the same as magnetic north. Since true north is what is needed, compensation must be added to the magnetic north

reading that the magnetometer produces. This can easily be done through an option in mission planner. The declination of Wichita, Kansas, is positive 4° 12' East so a declination of 4.12 will be used.

8.2.9 Elevon Mixing

The elevon mixing channel allows the user to mix the elevator and the aileron, as would be necessary on certain configurations, such as a flying wing. Since the flight test aircraft does not need this mixing, this aspect of the autopilot system will not be tested.

8.2.10 Flap Percent and Speed

There are no flaps on the flight test aircraft. ArduPlane 2.24 does not support flaperons, which is a combination of flaps and ailerons. For this reason, these parameters will be ignored.

8.2.11 Flight Mode Channels

The flight mode channels can be set using the Mission Planner utility. Table 8.7 shows a list of flight modes that will be set in the transmitter during flight testing.

TABLE 8.7

FLIGHT MODES

Mode 1	FBW-B
Mode 2	FBW-A
Mode 3	Stabilize
Mode 4	Return-to-Launch
Mode 5	Autopilot
Mode 6	Manual

8.2.12 Ground Absolute Pressure and Ground Temperature

Both the ground absolute pressure and the ground temperature are automatically calibrated when the aircraft initializes. For this reason, these values do not need to be entered by hand.

8.2.13 Inverted Flight Channel

The inverted flight channel is a channel declared by the user, that when a pulse-width modulation (PWM) value larger than 1750 is inputted from the receiver, the airplane will flip upside down and fly inverted. Since the flight test vehicle has a symmetric airfoil, this functionality should work very well.

8.2.14 Serial Baud Rate

The serial baud rate is the rate at which information will be transmitted over the Xbee modems. This can be turned up or down, but for our purposes, this rate will be left at 57,600.

8.2.15 Sonar Enable

This functionality is not available in the ArduPlane 2.24 version of the code. In future versions of the code, sonar may be used to help the aircraft flare while landing. Currently, this is not a core capability and therefore will not be tested.

8.2.16 Switch Enable

The APM board comes standard with dual in-line package (DIP) switches that allow the user to reverse the servos by flipping tiny switches on the board. The elevator will need to be reversed on the flight test vehicle, so the switch enable will be turned on.

8.3 Failsafe Options

Another important part of the autopilot system is the failsafe options. Great care should be given to these parameters. If they are set up correctly, they could save the aircraft from a crash. If they are set up incorrectly, they could actually induce a crash.

8.3.1 Throttle Failsafe

The throttle failsafe is an option that allows the user to recover the aircraft in the event of a loss of transmitter signal. In normal RC aircraft, the receiver that loses the signal from the transmitter usually ends up in a catastrophic crash. APM, however, has the ability to save the aircraft in such an event. When the throttle failsafe is enabled, if the throttle drops below the throttle failsafe value, the failsafe function is activated. The best way to implement this failsafe is to adjust the throttle to its lowest setting on the transmitter and then bind it to the receiver. A typical low PWM value would be around 900. Once the receiver has been bound, the throttle is adjusted back to its normal values. If the transmitter is turned off while in flight or simulation, the receiver will default back to the values at which it was bound. In this example, if the throttle failsafe is turned on, and the throttle failsafe value in APM is 950, when the APM sees the default value of 900, the failsafe will activate. Users can customize the failsafe action in order to better suit their needs. The following parameters determine how the autopilot acts when the failsafe function is activated.

8.3.2 Failsafe Long/ Short Action

Once the failsafe is activated, it can do two things. In any auto mode, the aircraft will continue its mission as if nothing had happen, unless the failsafe long action or the failsafe short action is turned on. Turning on the failsafe long action tells the autopilot that if the transmitter signal is not reestablished in 20 seconds, then it should return to launch. The failsafe short action does the same thing, except it will only wait 1.5 seconds. If the autopilot is in manual mode, FBW-A/B, stabilize, or any other non-fully autonomous mode, the aircraft will begin to loiter when transmitter signal is lost. If the failsafe long and short actions are turned off, the airplane will continue to loiter until the receiver reestablishes a connection or until it runs out of battery. If turned on, the failsafe long and short actions will work in the same way in bringing the aircraft back home, if the signal is not reestablish in the allotted amount of time. For this flight test aircraft, the failsafe short action will be utilized, under the assumption that since the transmitter and receiver combo being used has such a long range, at no point during the flight should the receiver loose signal unless something is wrong.

8.3.3 Failsafe Ground Control Station

This is another failsafe option that allows the user to force the autopilot to return to launch if the ground control station's heartbeat is lost for 20 seconds. This failsafe will not be used during flight testing, although a test will be done to see if it works.

CHAPTER 9

FLIGHT TESTING (VALIDATION)

Now that a full range of simulations has been performed to help tune the aircraft, flight testing can begin. Before starting, it is important to note the anticipated differences between flight testing and simulation. First, the simulation realm consists of perfect flight conditions with perfect sensors, which is not the case in real flight testing. Granted, wind and turbulence can be simulated in X-Plane, but this is not a perfect representation of real flight conditions. Second, X-Plane is a linear-based software and does not account for factors like separation of flow over surfaces. These issues will most likely lead to the need for further tuning during flight testing, but the simulation results are "in the ball park." Aside from this, it should still be possible to obtain good data that will give a decent view of how the autopilot performs. It should also be noted that many of these flight tests were simulated with different wind conditions. It is important to ensure that the limits of the mission with respect to wind are known before the flight test. Special effort will be made in the flight test results to emphasize the weather conditions at the time of the flight.

9.1 Servo PID Test

The most important gains will be tested first. Without an accurate servo PID gain set, the autopilot will not be able to control the aircraft. The testing/tuning of the servo PID gains was done with the same process as the simulation. On the day of testing, there was a 10 mph wind out of the south, so for all the tests, the aircraft was flown into the wind and kept at an airspeed of around 12.5 m/s. The aircraft was taken off in manual mode and then placed in FBW-A. The first axis tuned was the roll axis. A full right aileron input was given, causing the aircraft to produce a 55-degree bank angle. The stick was then released, allowing the autopilot to level out the aircraft. Figure 9.1 shows the response of the roll to the stick input. A few things must be noted from this graph. First, compared to Figure 8.10, which is the equivalent test performed during simulation, the red line is not as smooth as during the simulation, due to the non-uniform wind in which the aircraft is being flown, which is to be expected. Second, the rise time is about 1 second, compared to 0.6 second in the simulation. Notice that there is a rounding effect of the roll as it reaches the 55-degree bank angle and back to level, which did not appear in the simulation. Also, hardly any overshoot occurs when traveling to the new desired state.

The aircraft's roll performance is not as optimized as in the simulations, but it appears to be quite capable of performing autonomous missions. The only change made to the roll PID gain set was tuning the proportional gain

up from 0.7 to 0.9. This provided slightly more aggressive maneuvers without adding undesirable overshoot or oscillations. Figure 9.2 shows the roll response with the new gain, this time with a left-hand turn.

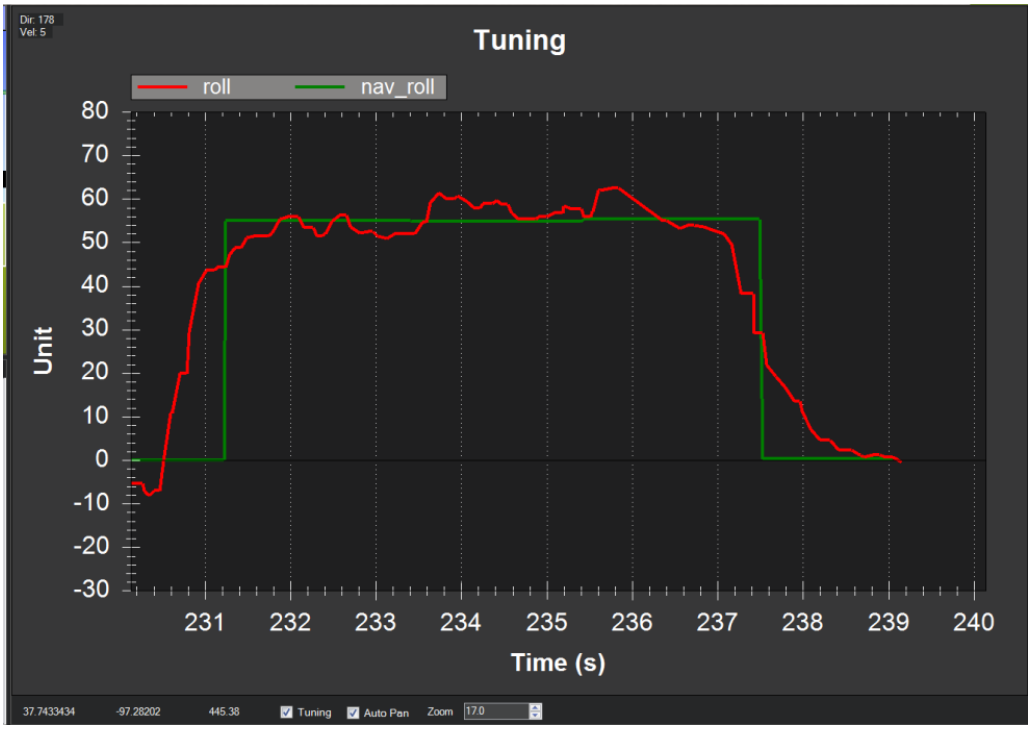


Figure 9.1. Servo roll gain set of P = 0.7, I = 0.4, D = 0.12 performing roll of 55° during flight testing.

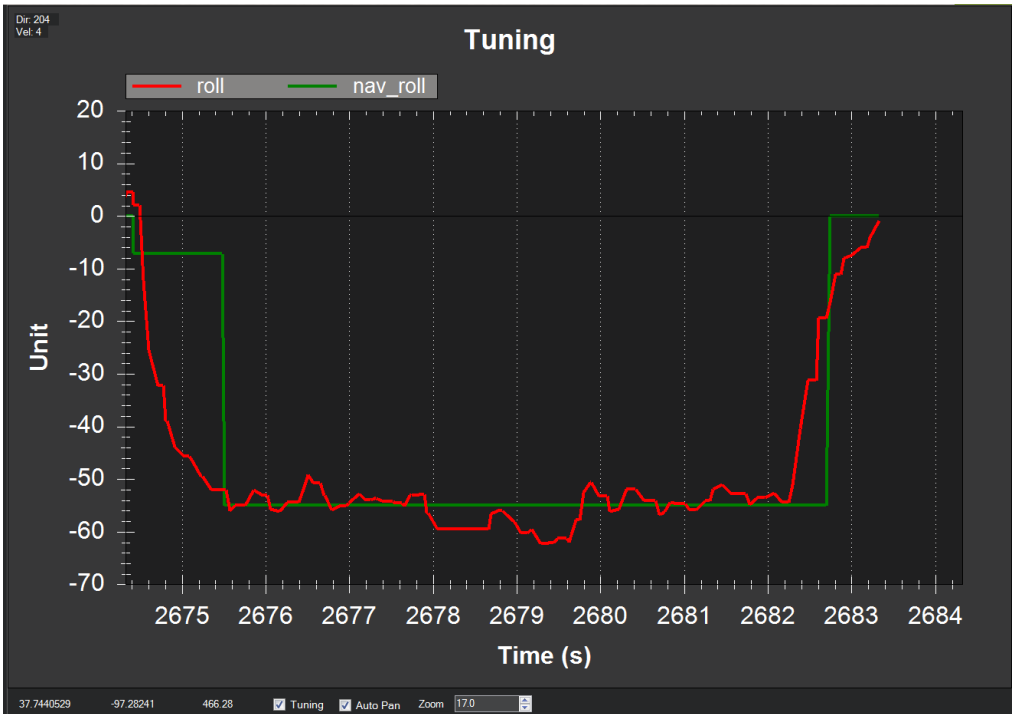


Figure 9.2. Servo roll gain set of P = 0.9, I = 0.4, D = 0.12 performing 55° roll during flight testing.

The same process was carried out for the pitch axis. An input of -25 degrees, which is the maximum negative pitch angle allowable, was given by the pilot and then released. Figure 9.3 shows the aircraft's reaction.

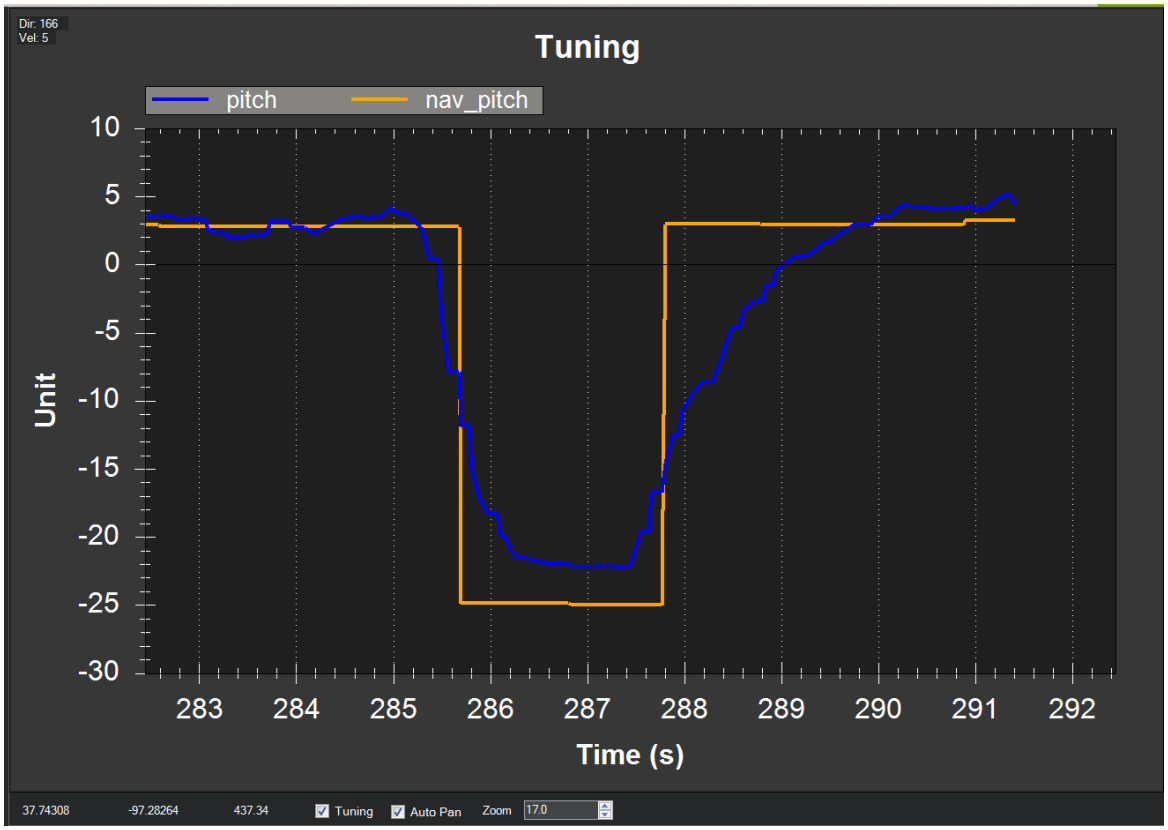


Figure 9.3. Servo pitch gain set of P = 0.7, I = 0.25, D = 0.05 performing pitch of -25° during flight testing.

Rise time, or in this case decline time, is about one second. The pitch never truly hits the desired -25 degrees, but rather settles around -22. This is due to the elevator setting for trim flight being about 3 degrees, which subtracted from 25, gives the difference that is shown. This is similar to what was seen in the simulation. The rise time is dreadfully slow at more than 2 seconds. There does not seem to be any over shoot either. Both of these characteristics lend themselves to achieving flight improvements by tuning up the proportional pitch servo gain.

By turning up the pitch proportional gain from 0.7 to 1.2, as shown in Figure 9.4, the aircraft gives pitch characteristics similar to what was observed in the simulations. Both the roll and pitch tests show a trend of the real aircraft performing slower maneuvers than what was seen in the simulation. This is most likely due to the aircraft's control surfaces being not as efficient as the simulation. This would make sense because the simulation is a linear process and not able to predict factors like separation.

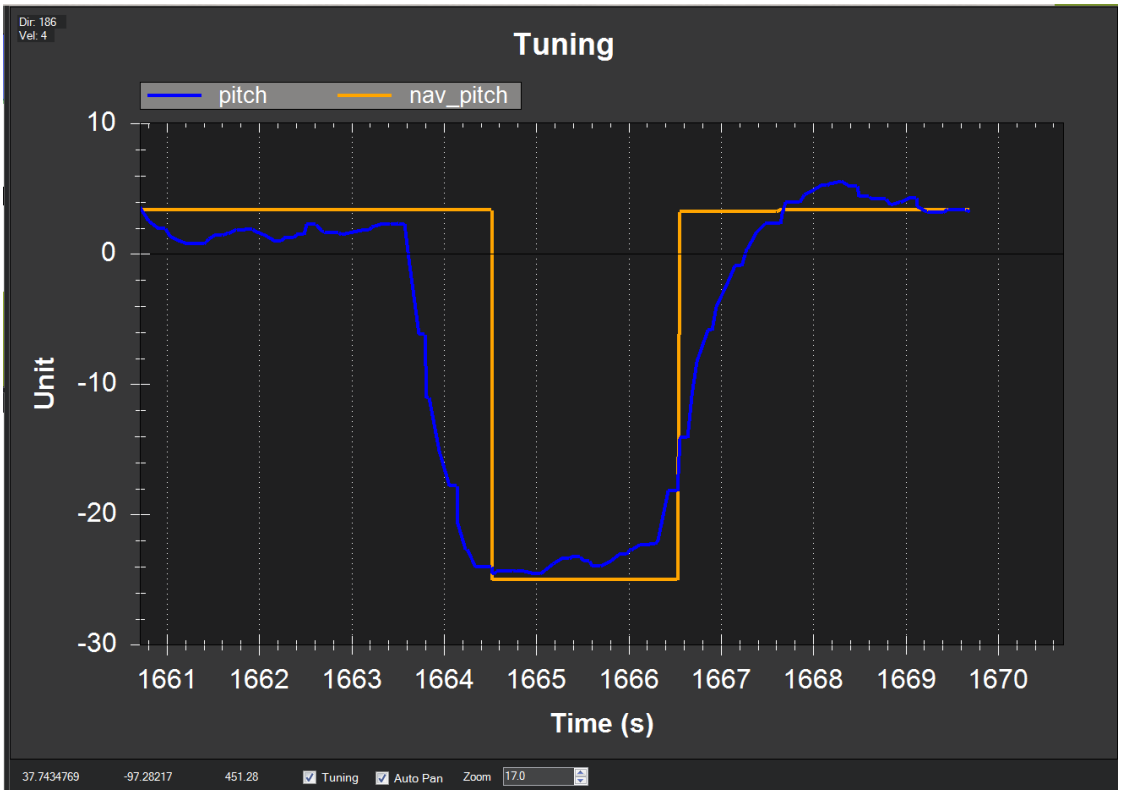


Figure 9.4. Servo pitch gain set of P = 1.2, I = 0.15, D = 0.05 performing pitch of-25° during flight testing.

The yaw is more difficult to analyze than the roll or pitch. The only real role it will play in the autopilot is if the rudder mix parameter is turned up. The actual PID gains that are being used for yaw are zero. Coincidentally, it is not as significant to the flight mission success as the other two parameters. For this reason, the yaw was judged by viewing it from the ground, instead of analyzing graphs, to see how affective it was at turning the aircraft. The rudder, with its current throws, seemed to do a good job of turning the aircraft. The pilot was able to keep the aircraft flying into the wind by using only the rudder. The next step will be to test different rudder mixes to see what impact on the turn the rudder will have while performing a coordinated turn with the ailerons.

9.2 Loiter Test

The next step is to view the performance of the navigational PID gains. This is done through a series of autopilot missions, the first of which will be a simple loiter mission. This test was performed with a 10 mph wind from the south. Figure 9.5 shows an over-head view of the flight path, and Figures 9.6 and 9.7 show some of the flight characteristics.

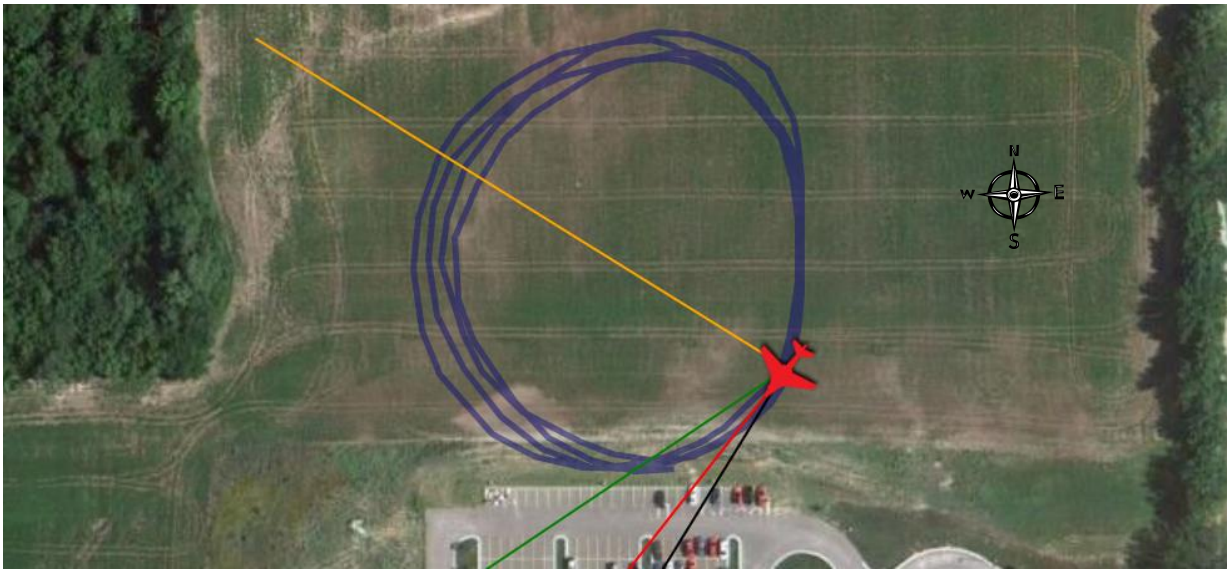


Figure 9.5. Ground track of loiter mission with radius of 45 meters.

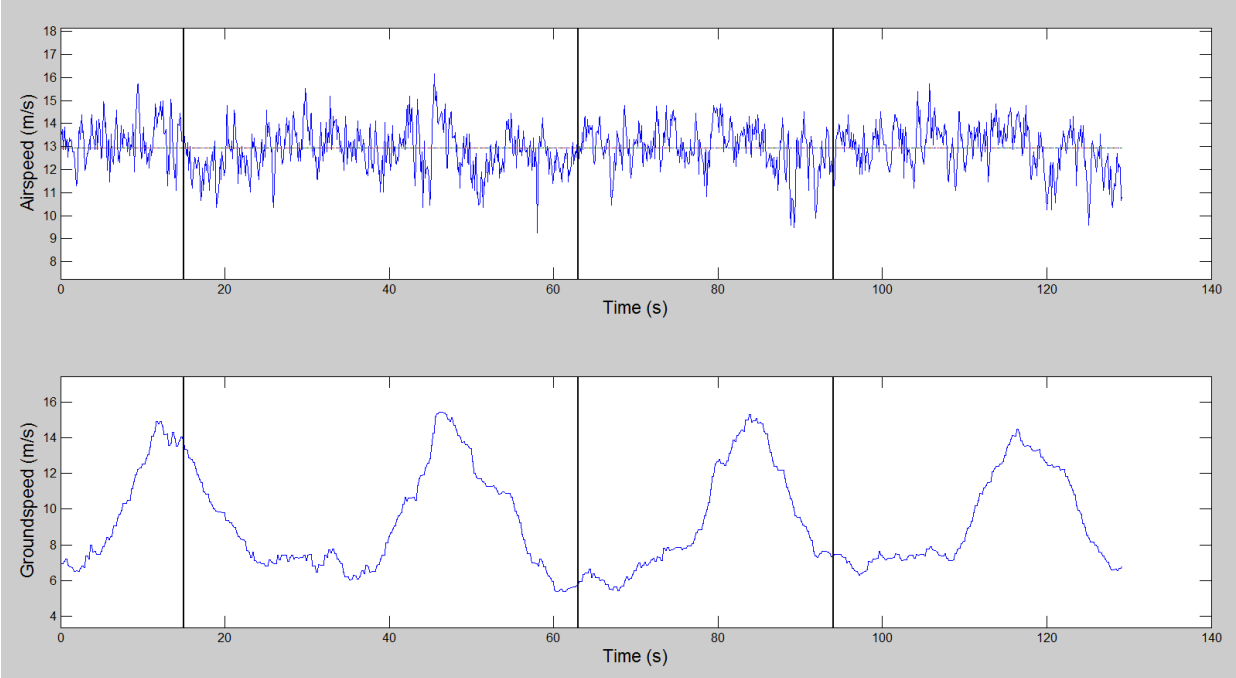


Figure 9.6. Airspeed and groundspeed vs. time of loiter mission with radius of 45 meters.

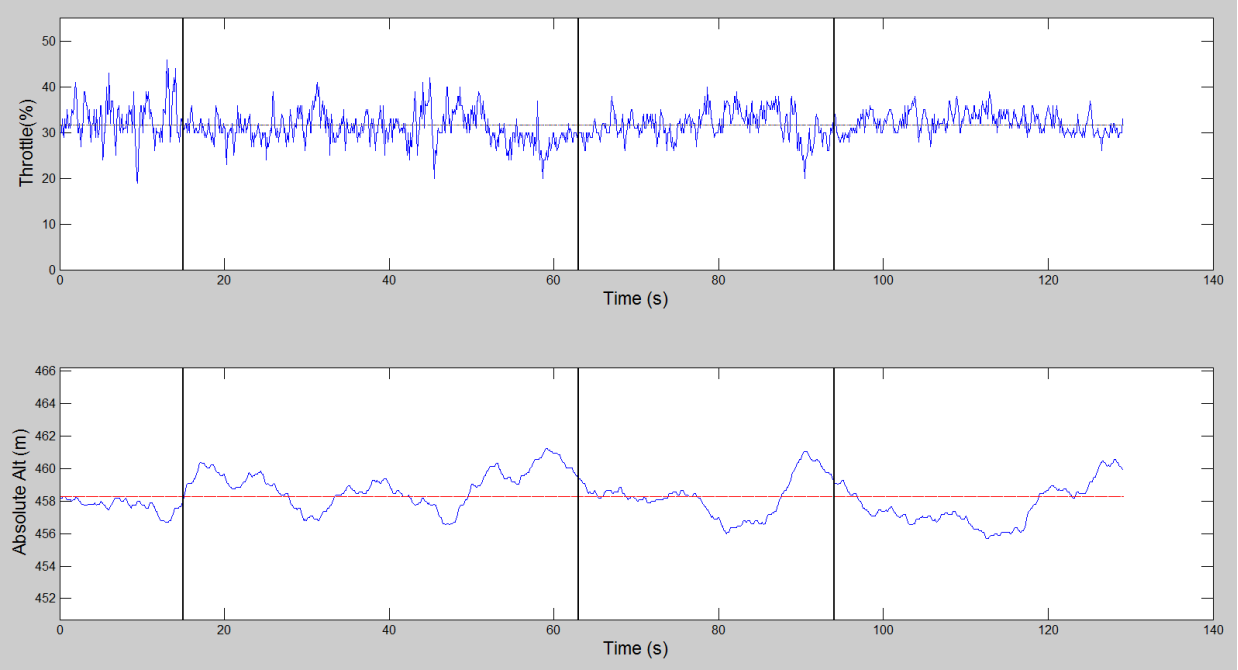


Figure 9.7. Throttle and altitude vs. time of loiter mission with radius of 45 meters.

From Figure 9.5 it can be seen what affect the wind has on the flight pattern. Going into the wind the path is very consistent, but flying downwind or crosswind, the flight pattern gets less consistent. Overall though, consistency of the flight path was pretty impressive.

Figure 9.6 shows the airspeed and groundspeed during the flight. It can be seen from the peaks and valleys of the groundspeed that this data was collected over four laps. The aircraft's autopilot proved the ability to control airspeed very well, given the windy conditions. The airspeed consistently stayed within ±2 m/s. The mean airspeed from this data set is 12.93 m/s, which is 0.43 m/s faster than the cruise speed desired. Equations 9.1and 9.2 were used to find the standard deviation of airspeed which was 0.9845 m/s.

$$s = \left(\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \right)^{\frac{1}{2}} \tag{9.1}$$

where

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{9.2}$$

During the loiter mission, it was noticed that the motor was revving considerably. In order to fix this, the energy/altitude proportional gain was turned down until the revving disappeared. On both Figures 9.6 and 9.7, the vertical lines represent these parameter changes. At the first parameter change, the first vertical line, the energy/altitude proportional gain, was turned down from 0.4 to 0.3. The second parameter change, the second vertical line, was turned down from 0.3 to 0.25, and the third was turned down to its final value of 0.2. Following these parameter changes, there seems to be no discernible difference in airspeed or groundspeed. It is difficult to obtain a definitive answer from the altitude graph on what impact that turning down this gain has on the altitude due to the gusting conditions. It appears, however, that the altitude enters a slower oscillatory mode as the gain is turned down. The main change is in throttle, where it can clearly be seen that magnitude of the fluctuations decreases significantly. The standard deviation of the throttle input at an energy/altitude proportional gain of 0.4 is 4.13%, while the standard deviation for the throttle at a gain of 0.2 is 2.25%.

After a few laps the autopilot was commanded to loiter 20 meters above its current location. This caused the aircraft to accelerate and pitch up until it reached the new desired altitude. Figures 9.8 and 9.9 show this process. The green dotted line represents where the command is given to increase altitude up to 480 meters absolute altitude. The autopilot quickly sped the aircraft up to 20 m/s and pitched up the aircraft. The overall climb took about 18 seconds. It is possible in Mission Planner to command desired assent and descent rates.

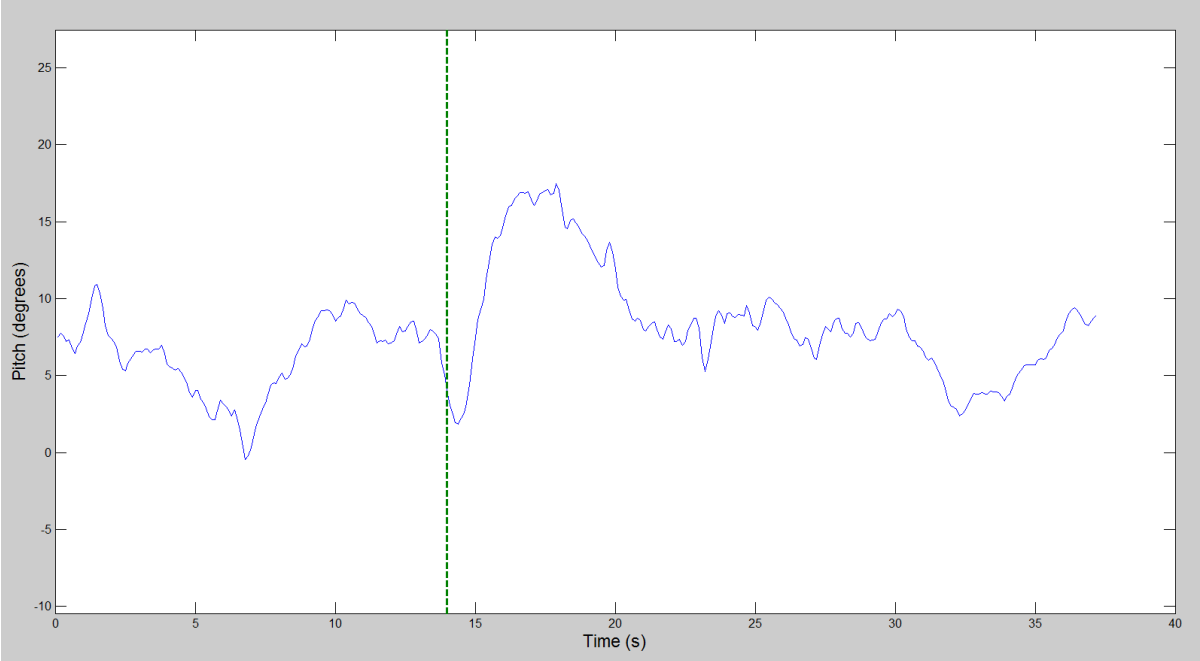


Figure 9.8. Pitch vs. time for climb of 20 meters to new loiter altitude.

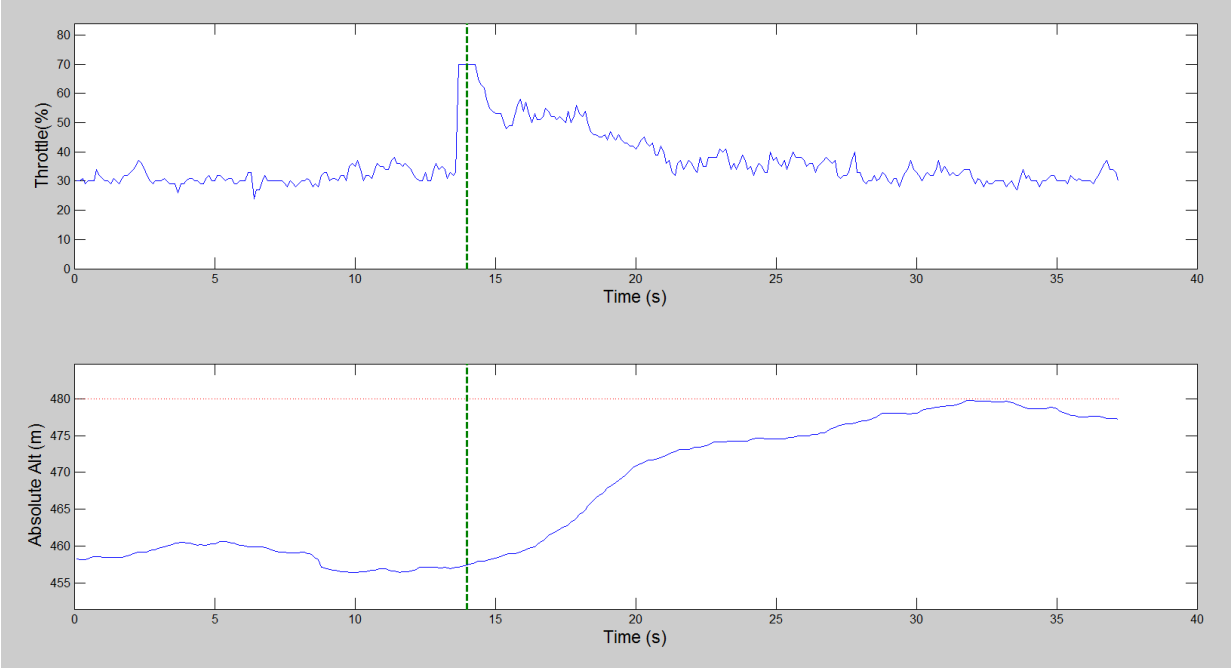


Figure 9.9. Throttle and altitude vs. time for climb of 20 meters to new loiter altitude.

Through this loiter mission, it has been shown that the autopilot is capable of maintaining both altitude and airspeed during a mission without significant variation from the desired values. The throttle predicted during simulations was nearly perfect, only 1% higher than anticipated. The main difference between the simulation and the flight test seems to be the revving motor, which was compensated for by turning the energy/altitude proportional gain parameter down from 0.4 to 0.2. This mission also proved that the aircraft both speeds up and pitches up to gain altitude, thanks to the navigational pitch-to-airspeed PID gain set. Overall, this mission has proven that the autopilot is capable of piloting the aircraft autonomously.

9.3 560 Meter Box Test

The next test flight mission to perform is the 560 meter box mission. This is identical to the mission run during simulations, with the box flight path consisting of four waypoints 140 meters away from each other. These tests were performed with a 10 mph wind from the north-northeast. The first mission was performed with an Xtrack gain of zero. This means that the autopilot is only concerned with hitting the waypoint and will take the shortest distance to get there. Figure 9.10 shows the flight pattern of the aircraft with no Xtrack gain.

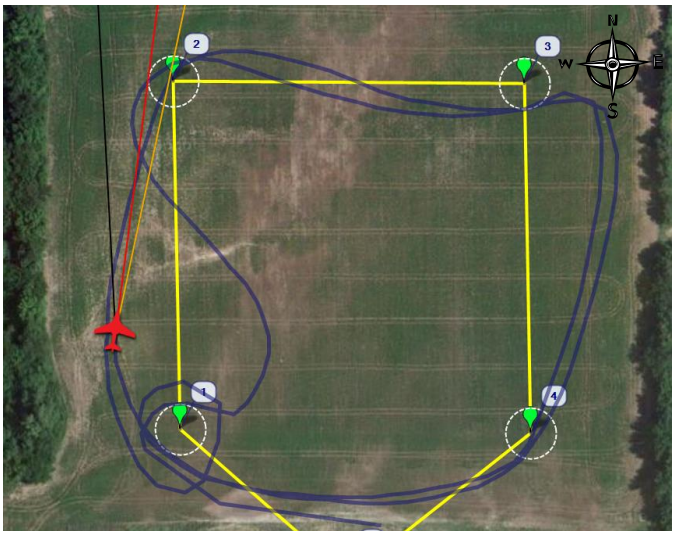


Figure 9.10. Ground track of two laps around 560 meter box mission with Xtrack = 0.

As can be seen from the Figure 9.10, the aircraft is blown off course due to the wind. Without the Xtrack turned on, the autopilot has no desire to get back on the designed flight path. Instead it will stay pointed at the waypoint, often missing it and having to go back and get it. The same flight test was then performed with the Xtrack gain turned to its default value of 100 and the entry angle set to a value of 30 degrees. Recall that this is the angle at which the aircraft will not exceed when attempting to get back on the flight path. Figure 9.11 shows the new flight path with the Xtrack gain turned on.

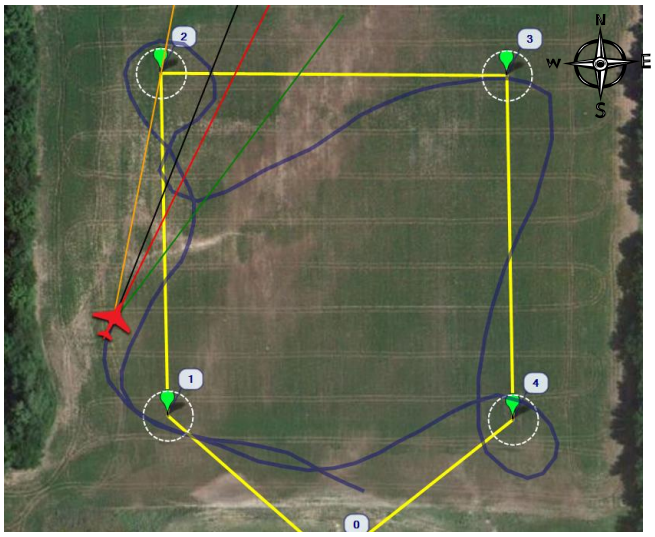


Figure 9.11. Ground track of one lap around 560 meter box mission with Xtrack = 100 and entry angle = 30.

The aircraft is now trying to stay on track, as demonstrated by the s-shape of the ground track between waypoints in Figure 9.11, but it is still missing waypoints due to the wind. During simulation it was decided to go with the default values for the navigational roll PID gains because they were easily handling when hitting these

waypoints. For a windy scenario, however, these navigational roll gains need to be turned up to help the aircraft consistently hit the waypoints. Through a series of flight tests, it was decided to change the navigational proportional gain from 0.7 to 1.3, turn the rudder mix from 0.5 to 1, and turn the Xtrack gain from 100 to 75. The result can be seen in Figure 9.12. Considerable snaking still occurs between the waypoints, but the aircraft is now consistently hitting the waypoints in spite of the wind. Noticing that the airplane was losing altitude during the turns, the pitch compensation gain was also turned up from 0.175 to 0.25. Now that the autopilot has proven to be capable of consistently hitting the waypoints, the automatic landing and takeoff will be tested.

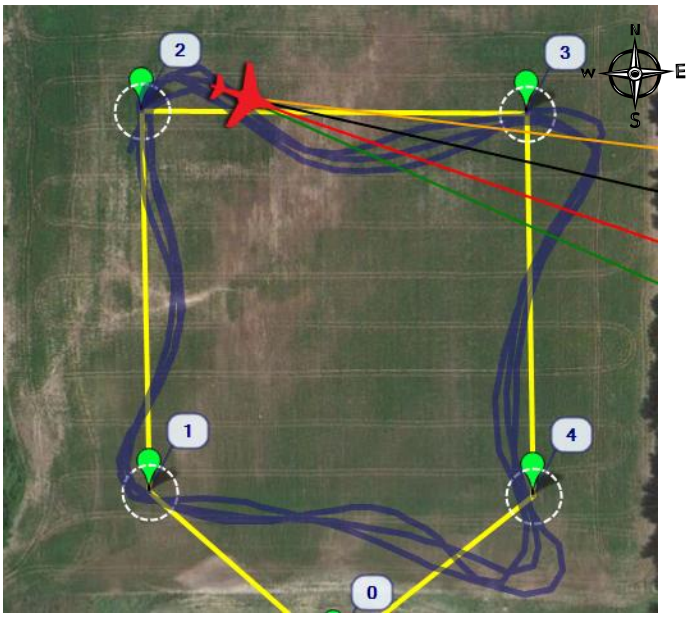


Figure 9.12. Ground track of three laps around 560 meter box mission with new improved gains.

9.4 Automatic Takeoff

Several automatic takeoff tests were performed to better understand the capabilities of the autopilot while performing this function. On the day of the test, the wind was 10 to 15 mph. Each automatic takeoff was performed into the wind. Since the aircraft was hand launched, an attempt was made to release the aircraft the same way each time. The aircraft was released with a slight push, with as little moment induced on the aircraft as possible. The starting takeoff angle was set to 15 degrees and then incrementally increased by 10 degrees for the following tests. The takeoff concluded once the aircraft hit 50 meters altitude above ground level.

APM performs an automatic takeoff by throttling up to the maximum throttle allowable and maintaining a minimum angle of attack. Both the minimum angle of attack and the maximum throttle are entered by the user. Remember that the test aircraft's maximum throttle value was set to 70%. The user must also input an altitude,

which once reached, the autopilot will consider the takeoff complete and move onto the next task. The heading of the aircraft on takeoff is established by reading the magnetometer when the aircraft's groundspeed exceeds 3 m/s, as measured by the GPS. The automatic takeoff command will not adjust to try to hit the next waypoint; it will only maintain its initial heading. Figures 9.13 and 9.14 show performance parameters of the first takeoff test at a minimum angle of 15 degrees for the first 4 seconds of the takeoff.

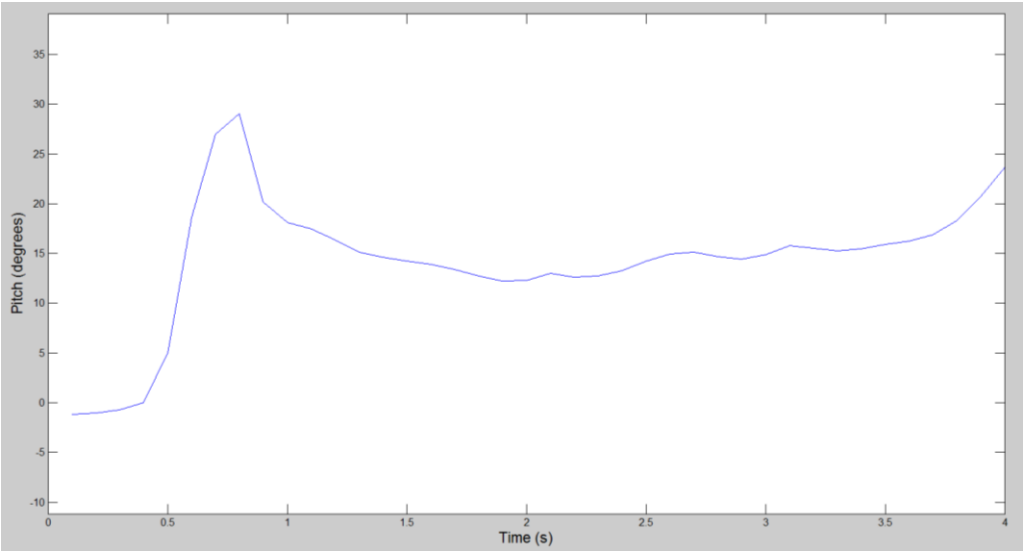


Figure 9.13. Pitch vs. time of 15° minimum pitch auto takeoff.

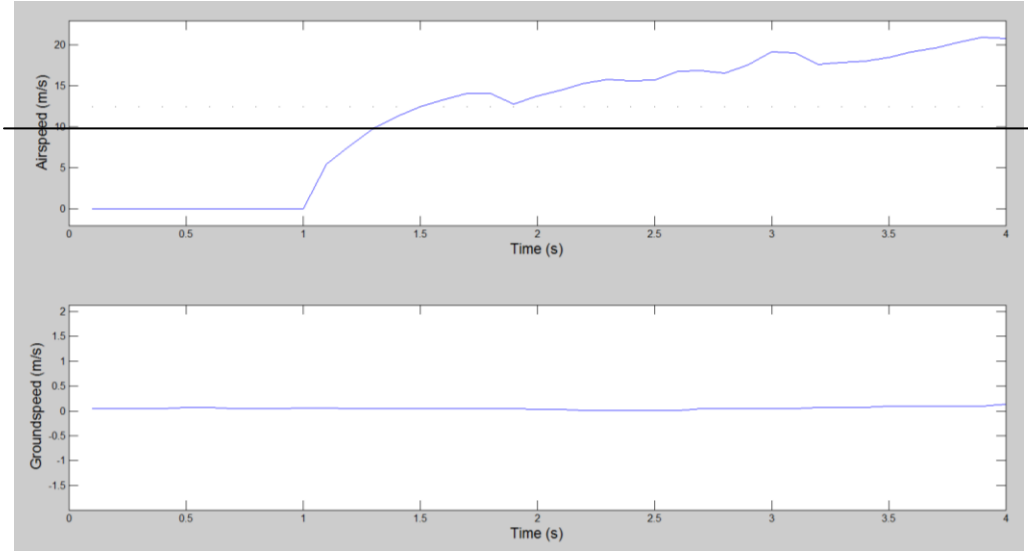


Figure 9.14. Airspeed and groundspeed vs. time of 15° minimum pitch auto takeoff.

A few features can be noticed from these two graphs. First, the pitch jumps up to 30 degrees and then settles back down to around 15 degrees, which was commanded as the minimum pitch angle. At about 2 seconds, the aircraft starts to pitch back up, due to the airspeed-to-pitch PID gain set. Notice that also around 2 seconds, the

aircraft reaches airspeed of 12.5, which is the desired cruise speed. As the aircraft speeds up past this desired speed, the airspeed-to-pitch gain rotates the aircraft up to try and bring the speed back down to 12.5 m/s. This is a desirable characteristic at takeoff. Undesirable, however, is that the groundspeed is still reading zero. There is actually about a 4-second delay between when the aircraft is launched and when the groundspeed reading is activated. This is a problem because the autopilot does not lock onto a heading until it reads that the groundspeed has reached 3 m/s. Four seconds is a long time for the aircraft to be able to drift until the heading is locked. For this reason, it is recommended that there be plenty of room to the left and the right of the aircraft when attempting an auto takeoff.

This same test was repeated for minimum takeoff angles of 25, 35, 45, 55, and 65 degrees. At these higher pitch angles, the airspeed-to-pitch parameter had little effect, and the aircraft mostly stayed at the minimum pitch angles after the initial pitch spike (which decreased as the minimum pitch angle increased). There was a consistent 4-second delay between the aircraft being launched and the GPS indicating a change in groundspeed. Forty-five degrees was found to be the preferred takeoff angle with a quick climb rate of around 5 m/s. The limit of the aircraft was found to be 65 degrees, where the aircraft was unable to reach altitude in a timely manner.

An interesting event occurs at higher minimum pitch angles. Figures 9.15 and 9.16 show an auto takeoff at a minimum pitch angle of 55 degrees. Once the aircraft hits the desired altitude, a tremendous shift in the autopilot logic occurs and the aircraft pitches down very quickly. For the takeoff shown in Figure 9.15, a pitch rate of -80 degrees per second was experienced. This may be unacceptable structurally for some aircraft, and these high-angle takeoffs should be avoided.

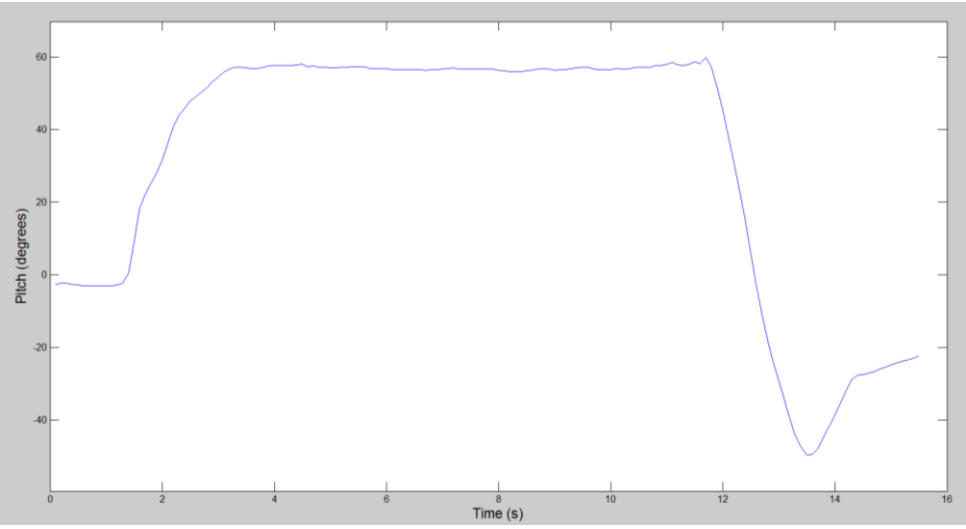


Figure 9.15 Pitch vs. time of 55° minimum pitch auto takeoff.

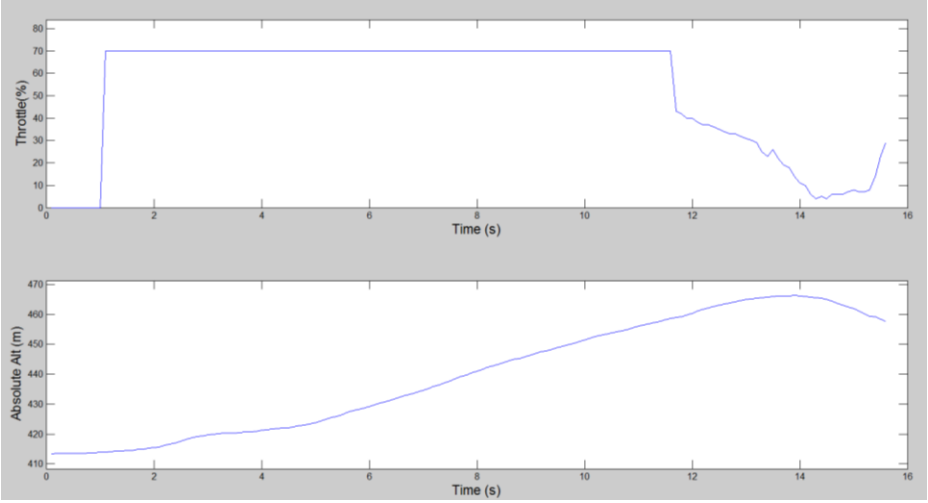


Figure 9.16.Throttle and altitude vs. time of 55° minimum pitch auto takeoff.

APM has proven capability of completing an automatic hand-launched takeoff. Further flights with landing gear were also successful. As a note, on takeoff, the aircraft will attempt to pitch up immediately to the minimum angle declared by the user. Many aircraft need to attain some speed before rotating for lift off, so a different approach to takeoff may need to be exercised.

9.5 Automatic Landing

The last and most difficult part of autonomous flight, the automatic landing tests, has been left to be performed at the end, given the inherent risk of performing such tests. For this series of tests, a 1550 mAh battery was used instead of the 3200 mAh battery dropping the aircrafts weight by approximately 12%. This will help in lowering the stall speed so the aircraft can be landed slowly. On the day of testing, there was a 5 to 10 mph wind out of the west. Figure 9.17 shows the mission layout, and Table 9.1 gives the mission profile.

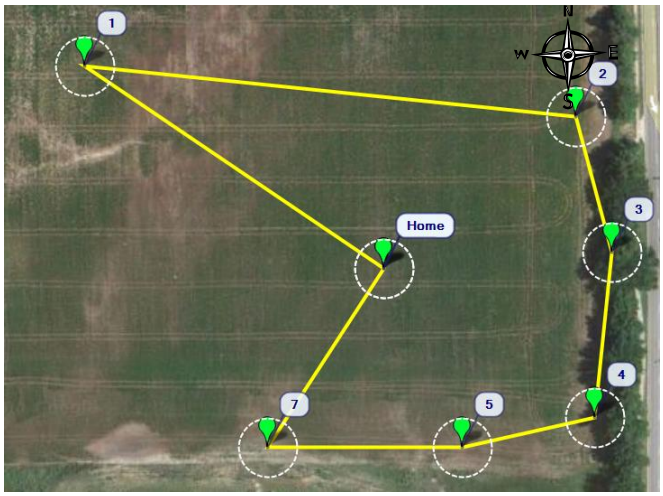


Figure 9.17 Landing mission ground course.

TABLE 9.1

AUTOMATIC LANDING MISSION PROFILE

Action #	Command	Altitude(m)	Speed (m/s)
1	Waypoint	40	12.5
2	Waypoint	40	12.5
3	Waypoint	40	12.5
4	Waypoint	30	12.5
5	Waypoint	20	12.5
6	Change Speed	N/A	9.0
7	Land	0	9.0

Note: Altitude is in reference to the ground and is not absolute altitude.

In order to better set up the landing approach, several low-speed passes were done in order to hone in what the placement of the waypoints should be, and the corresponding altitudes to get a consistent landing. It was learned during simulations that the airspeed would have to be brought down in order to land flat. Since the airspeed-to-pitch gain set has been tuned fairly aggressively, the aircraft preferred to pitch down to lose altitude. This works well for flying to waypoints but is a hazard when trying to land. To compensate for this, the cruise speed after waypoint 5 was reduced to 9 m/s. This caused the aircraft to pitch up slightly and to slow down rather than nose down. The overall affect was an artificial flaring of the aircraft upon landing.

With the speed change and the approach set up correctly several landings were attempted. All landed very smoothly, but it became obvious that something needed to be changed to obtain a more accurate landing. Depending on the wind conditions, one landing would be 35 meters short of the target, and then the next would be 30 meters long. Turning up the energy/altitude proportional gain from 0.2 to 0.3 seemed to help the aircraft maintain the correct altitude while on approach, at the expense of some motor revving. Figure 9.18 shows a typical landing, with the final approach in red. Figure 9.19 shows a cluster of four landings performed to get a sense of their accuracy.

All four landings were within a 20-meter radius of each other, three which landed in a 10-meter radius. During the fourth, the outlier in this set, the wind picked up on approach. Most likely a good pilot would be able to land the aircraft more accurately, but overall these auto-piloted landings are acceptable and have proven that the autopilot is capable of performing a fully autonomous flight consisting of takeoff, hitting waypoints, and landing.

Figure 9.18 Visual of approach and landing in Google Earth [28].

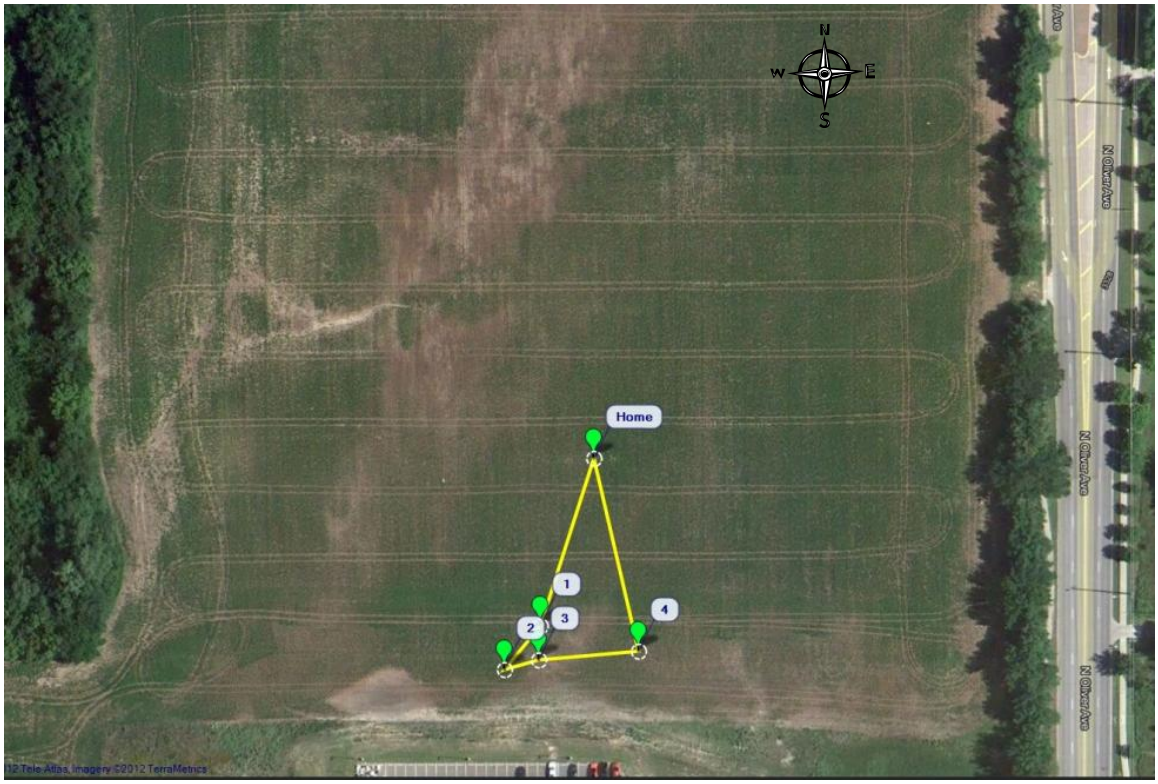


Figure 9.19 Grouping of four landing attempts.

CHAPTER 10

SUMMARY

After a full range of flight testing, it was determined that the ArduPilot-Mega microcontroller is able to anchor a very robust autopilot system that is fully capable of piloting autonomous flights on RC aircraft. The autopilot system seems to have all of the attributes necessary for implementation into higher education. Through this project, a process was developed to do the following: design and build an aircraft, run flight simulations on a model representation of the aircraft for the purpose of tuning gains, and run flight tests on the aircraft to tune the gains further. The end product will ultimately allow students to complete fully autonomous missions with their aircraft. The process described would fit perfectly into a design-oriented class aimed at teaching students how to design and build remote-controlled unmanned aerial systems. By going through this process on their own, students may get a better understanding of how autopilot systems work, thus providing them with a more well-rounded education on the subject.

As more people join the open-source autopilot community, the ArduPilot-Mega microcontroller will only continue to get cheaper, lighter, smaller, faster, and more robust, as it already has proven to do during the writing of this thesis. More features will continue to be added to the autopilot system, giving students even more creative opportunities and fueling innovation. Further research will entail the addition of new sensors and actuators to aid in performing new missions and increase the spectrum of education. The addition of flexure sensors to wing joints could provide stress information for structural consideration. The addition of range finders could aid in landing flare control and obstacle avoidance maneuvers. The addition of programming for aircraft to "talk to each other" would provide opportunities for aircraft to fly in formation or perform coordinated tasks. The open-source nature of this autopilot system truly makes the sky the limit.

REFERENCES

REFERENCES

[1] U.S. Army UAS Center of Excellence, "U.S. Army Unmanned Aircraft Systems Roadmap 2010-2035," 2010.

[2] Aguirre, Lauren, "Time Line of UAV's," *NOVA* by PBS, November 2002, http://www.pbs.org/wgbh/nova/spiesfly/uavs.html [accessed 2/13/12].

[3] Kushner, David, "The Making of Arduino," IEEE Spectrum, October 2011, http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino/0 [accessed 1/20/12].

[4] Arduino, http://www.arduino.cc/ [accessed 12/24/11].

[5] Munoz, Jordi, "The ArduPilot Mega Hardware," DIY Drones Repository, http://code.google.com/p/ardupilot-mega/wiki/Hardware [accessed 4/2/12].

[6] "The ArduPilot Mega IMU Shield Hardware," DIY Drones Repository, http://code.google.com/p/ardupilot-mega/wiki/IMUHardware [accessed 4/2/12].

[7] "The ArduPilot Mega autopilot" DIY Drones Repository, http://code.google.com/p/ardupilot-mega/wiki/Introduction [accessed 4/2/12]

[8] DIY Drones Store, https://store.diydrones.com/MediaTek_MT3329_GPS_ 10Hz_Adapter_Basic_p/mt3329-02.htm [accessed 4/2/12].

[9] "Using an Airspeed Sensor," DIY Drones Repository, http://code.google.com/p/ardupilot-mega/wiki/Airspeed [accessed 9/15/12].

[10] "AttoPilot Voltage and Current Sense Breakout - 90A," Sparkfun Electronics, http://www.sparkfun.com/products/9028 [accessed 4/2/12].

[11] "Measuring your battery voltage and current consumption with APM," DIY Drones Repository, http://code.google.com/p/ardupilot-mega/wiki/Voltage [accessed 4/2/12].

[12] "Xbee Telemetry Kit," DIY Drones Store, https://store.diydrones.com/Xbee_Telemetry_kit_p/kt-telemetry-xbee.htm
[accessed 2/13/2012].

[13] "HMC5883L - Triple Axis Magnetometer" DIY Drones Store, https://store.diydrones.com/HMC5883L_Triple_Axis_Magnetometer_p/br-hmc5883-01.htm [accessed 4/2/12].

[14] Anderson, Chris, DIY Drones, http://diydrones.com/ [accessed 3/1/12].

[15] "Wired's Anderson on 3D Robotics Drone Sales," Bloomberg, December 2011, http://www.bloomberg.com/video/82642032/ [accessed 2/13/2012].

[16] "DIY Drones at 22,000 members!," DIY Drones, http://diydrones.com/profiles/blogs/diy-drones-at-22-000-members [accessed 4/2/12].

[17] Catia (Version 5 Revision 21) [Software], (2010), Dassault Systèmes.

[18] Drela, Mark , Youngren, Harold AVL (Version 3.27) [Software], 1988, [retrieved from http://web.mit.edu/drela/Public/web/avl/].

REFERENCES (continued)

[19] MATLAB (Revision 2009b) [Software], (2009), MathWorks.

[20] ScicosLab (Version 4.4.1) [Software], 2011, [retrieved from http://www.scicoslab.org/].

[21] FlightGear (Version 2.6.0.1) [Software], 2012, [retrieved from http://www.flightgear.org/download/].

[22] JSBSim (Version 1 Release Candidate 2) [Software], 2009, [retrieved from http://sourceforge.net/projects/jsbsim/files/].

[23] X-Plane (Version 9.7) [Software], 2011, Laminar Research, [retrieved from http://www.x-plane.com/downloads/landing/].

[24] Oborne,Michael, Mission Planner (Version 8.24) [Software], [Retrieved from http://code.google.com/p/ardupilot-mega/downloads/list].

[25] Hepperle ,Martin Java Foil [Software], 2008, [retrieved from http://www.mh-aerotools.de/airfoils/javafoil.htm]

[26] "Using the Mission Planner Ground Station," DIY Drones Repository, http://code.google.com/p/ardupilot-mega/wiki/MPGCS [accessed 4/2/12]

[27] Visioli, Antonio, "Basics of PID Control," in *Practical PID Control*, Springer-Verlag, London, 2006, pp. 1-18.

[28] Google Earth (Version 6.1) [Software], 2011, [retrieved from http://www.google.com/earth/download/ge/].

APPENDIX

APPENDIX

LIST OF FLIGHT MODES OF CORE APM AUTOPILOT FIRMWARE

Manual – While in this mode, the pilot has complete control of the aircraft. The autopilot system has no influence on the control of the aircraft.

Stability Mode – This is the mode that every other auto-piloted mode uses, incorporating the main servo PID gain controller. The pilot still has overall say of what the aircraft does, but the controller will always try to bring the aircraft back to zero pitch and zero roll. For the pilot, the controls may feel "mushy," since the controller will want any movement away from center in pitch and roll to come back to center. If the pilot lets go of the stick, the plane should go back to center. The controller has no influence over the throttle.

Fly-by-Wire-A (FBW-A) – This mode is similar to stability mode, except the controller has more say over navigational control. Throwing the aileron stick full right will cause the aircraft to continuously roll to the maximum roll limit inputted by the user, where normally the aircraft would do barrel rolls. Flying in this mode has been described as similar to driving a car. This is a very good mode to test the servo PID gains. The throttle is controlled manually

Fly-by Wire-B (FBW-B) –This mode is similar to FBW-A, except the throttle is controlled by the controller. The airspeed sensor must be enabled, or the mode will revert back to FBW-A. There is no direct control over the elevator in this mode. Instead, if an elevator-down control is given by the pilot, the logic of the controller will understand the command as wanting to lower altitude and therefore lower the throttle. This flight mode is not recommended for testing.

Autopilot – In this mode, the controller has complete control over the aircraft. The aircraft will use a pre-programmed mission to fly to waypoints. The pilot is still able to give suggestions to the aircraft, which is equivalent to nudging the aircraft in the right direction, but the autopilot still has the overall say.

Return-to-Launch – Similar to autopilot mode, instead of following a course of GPS waypoints, the controller will tell the aircraft to return to the home GPS waypoint and loiter until given further instruction.

Loiter – Similar to autopilot mode, instead of navigating through a course of waypoints, the aircraft will loiter around a specific GPS point until given another command.