Unmanned Aerial Vehicle

Figure 1: 4-Channel Electric UAV RQ-1 Predator 63" Radio Remote Controlled RC Spy Plane ARF [12]

Paul Agyiri & Tristan Lawson
E90 Final Report, 8 April 2008
Advisors: Professor Erik Cheever & Professor E. Carr Everbach

Table of Contents

Abstract

This report summarizes the work accomplished by Paul Agyiri and Tristan Lawson on an Autonomous Unmanned-Aerial Vehicle. The motivation behind this project stemmed from Tristan Lawson's experience with flying gliders while studying abroad in Australia. Also, Paul Agyiri's background in Control Theory, Electronic Circuit Applications, Fundamentals of Digital Systems and other engineering courses provided a sufficient knowledge base to design and develop an Autonomous Unmanned-Aerial-Vehicle. The main idea was to design and construct an autonomous flight control system for a scaled model unmanned aerial vehicle. This new system replaced the conventional R/C transmitter and receiver used in flying model airplanes. The system was essentially an autopilot that flew the airplane on a preset course.

The project established a sturdy radio frequency data downlink between a ground station and the avionics onboard the aircraft.

The microcontroller on the ground station powered the potentiometers on the transmitter via digital to analog converters which in turn controlled the control surfaces of the airplane.

This system was used alongside the typical R/C manual control system. A switch was designed and built to disengage the aircraft from manual control to autonomous and vice versa when deemed necessary.

System Overview

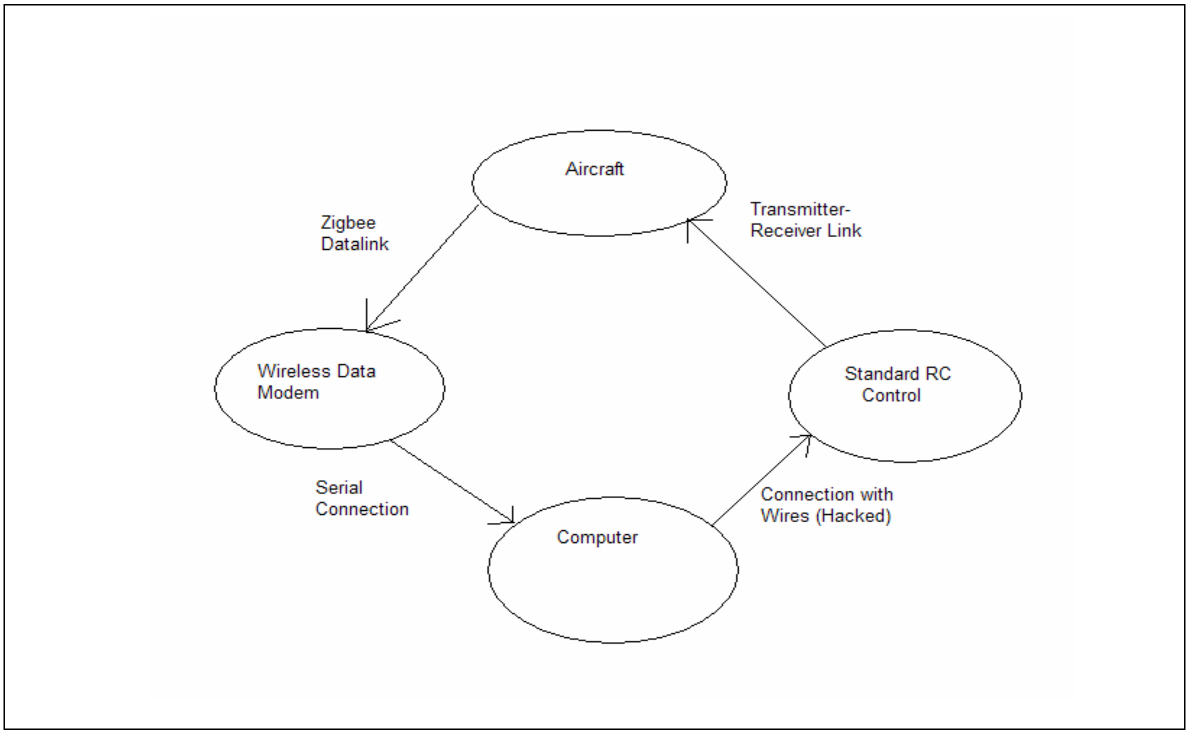


Figure 2: Overview of Communications System

Introduction

Unmanned aerial vehicles are becoming highly utilized by search and rescue teams, the military and other branches of law enforcement. This is a result of relatively low human and collateral costs, long mission endurance and increased terrain accessibility. Autonomous aircraft technology is still a highly researched field, and while relatively simple systems already exist such as in commercial aircraft autopilots, the capability of the system to adjust to certain changes such as in dynamic path planning remains quite difficult to build.

The inspiration for the design of our Autonomous Unmanned-Aerial-Vehicle is the open source Paparrazi UAV system. The system is quite complex, using powerful computers and advanced Kalman filters to obtain ideal flight control. The UAV system comprises two main components: a ground station component which provides telemetry feedback for the operator and allows for control of the aircraft, and a flight system component onboard the vehicle.

Due to time constraints, limited financial resources, and the unnecessary complexity of systems such as that of the Paparrazi, it was decided to restrict the focus of our project to designing a simple autonomous control system for a scaled model RC airplane by transmitter hacking. To enable easy switching between the computer and manual control of the airplane, the system implemented literally worked alongside the standard manual control transmitter which came as a standard accessory with the model airplane kit.

The following report discusses the assembly of the aircraft, the setup of the telemetry downlink as well as the ground station, the design of the control system and test flights of our aircraft. Finally, a conclusion and suggestions for possible future work that can be done to improve the performance and design of the current model are presented at the end of the paper.

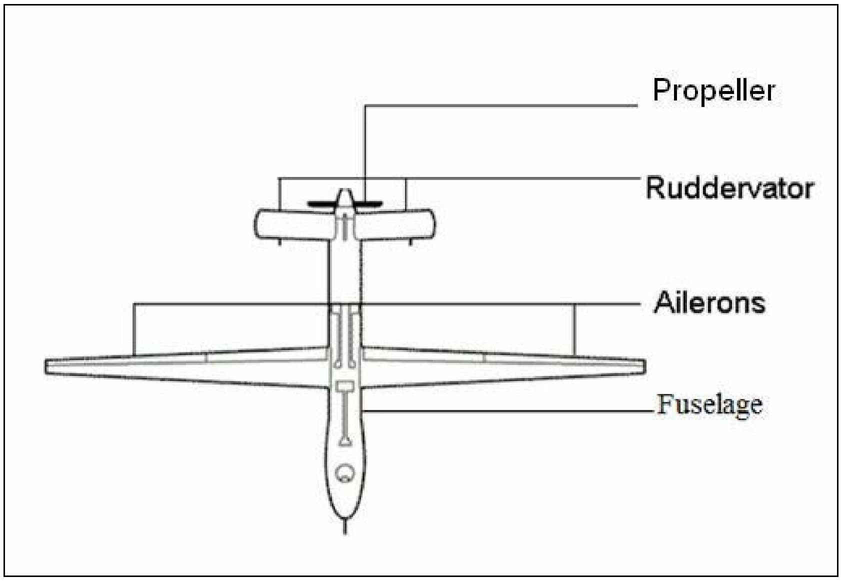


Figure 3: Illustration of Parts of the Aircraft [11]

UAV Design

Model Aircraft

The first thought that came to mind was to find a scaled model UAV RC aircraft since a project on UAVs was being pursued. In deciding which model aircraft to base our project on, various designs, cost, sizes, fuselage material and main power/propulsion source were taken into consideration. Options included having airplanes built out of Styrofoam or Balsa, or having electric motors versus gas motors. Only later in the project were we informed that an aircraft such as the Ready-Built Senior Telemaster, shown in the figure below, would have been ideal for our intentions because of its high payload capacity.

Figure 4: Ready-Built Senior Master

However, the decision to work with the 4-Channel Electric UAV RQ-1 Predator 63" Radio Remote Controlled RC Spy Plane, a picture of which is shown on first page, had already been made. This aircraft was chosen mainly because of limited financial resources. In addition, it was capable of handing the payload designed for our project.

The specifications that came with the aircraft as well as the specification that needed to be met are listed below:

- Wing Span – 63in/1600mm
- Wing Area – 372 sq in/24 sq dm
- Flying Weight – 1.75 lb / 800g
- Fuselage Length – 36 in/920 mm
- CG – 50 to 55mm from rear of leading edge of aircraft
- Radio Required – 4 channels, 4 servos
- Out runner Brushless Motor 150 ~ 200W

Aircraft Components Overview and Assembly

The airplane kit arrived with the parts shown in the figure below:

Figure 5: Parts of the Aircraft that came with the Kit [13]

Structural considerations in aircraft construction

The model aircraft used as our flight platform had a body which was entirely composed of balsa wood and heat shrink film. Consequently the aircraft was structurally fragile while maintaining a light weight which was important for efficient flight.

In selecting the airplane kit, it was noticed that a few hobbyists experienced wing failure with their models. This phenomenon occurred mainly due to high wing loading which resulted in the entire wing breaking off from the fuselage. To increase structural strength of the wing-fuselage connection, a 1 foot long hollow carbon fiber rod was run through 4 inches of one wing, continuing through the fuselage and ending inside the next wing. Noting the failures that some pilots experienced, it was decided to add an extra measure of structural integrity by inserting a single 1 foot long, 1/8 inch thick metal push rod inside the hollow carbon fiber rod. This metal pushrod had less tensile strength than the carbon fiber rod but nonetheless served to bolster the strength provided by the carbon rod. Since the diameter of the carbon rod was 3 times as great as that of the metal rod, epoxy glue was poured into the carbon rod to fill the spacing and secure the metal rod inside the carbon. This setup performed as expected several times until the last flight of the aircraft.

Epoxy glue was used significantly in assembling the airplane. Besides the assembly of the ailerons and rudder-vator control surfaces, epoxy glue was used wherever secure, firm connections had to be made. Again, in the interest of having as strong an airplane as possible, the epoxy was applied to the various interfaces very liberally, most times extending its application up to two centimeters away from the interface of two surfaces. One important point to note in using the epoxy is that when seeking a firm bond, it is essential to ensure that the two surfaces are completely bare. In other words, only bare wood should be exposed. Any residual film or thick paint may weaken the desired bond.

It was also noted that despite the large amount of epoxy on the airplane, there was no significant increase in weight. Other components such as the lithium polymer battery contributed greatly to the overall flying weight, making the weight of the epoxy relatively negligible.

<u>Nose landing gear replacement</u>

Before our final, fatal flight of the airplane, the nose landing gear had been replaced with a customized skid since the plane was being landed in 2-3 inch high grass. The skid was made by bending 2 leftover push rods in a C shape and connecting it to the lower fuselage with the convex edge pointing forwards. Epoxy was used to attach the skid to the fuselage with a 1x1 inch section of thin breadboard along with a similarly sized strip of paper. The paper separated the fuselage from the breadboard/skid component and was designed with the objective of the skid making a clean separation from the fuselage in a hard landing. With the common wheel/metal strut setup, the plane landed very abruptly as the wheels caught in the grass. This hard landing induced large forces on the airframe via the landing struts which could potentially crack and weaken the fuselage especially because of the way that the wheel struts were connected to the fuselage.

The Center of Gravity (CG) was one of the most important specifications of the aircraft that had to be met. As such, after the aircraft was assembled, a test was carried out to ensure that it had been met. This was done by balancing the aircraft on the tip of its wings. If the fuselage tilted forward within a range of approximately 0 to 30 degrees (nose heavy), the CG requirement had been met. However, if the fuselage tilted backwards to any degree (tail heavy), the position of the aircraft's payload had to be adjusted accordingly until the aircraft was no longer tail-heavy.

Choosing A Propulsion System – Airframe and Environment

To a large extent, the choice of the airframe affected our choice of motor. The only means of powering the Predator UAV model was by electric motors. In retrospect, an electric powered aircraft would have been sought for this project in any case. Though leaving a very tiny footprint, the emissions from a gas powered motor would have directly contributed to air pollution. Also, balsa, the material from which the airframe was made comes from a renewable resource unlike Styrofoam which is a hydrocarbon byproduct.

Electrical System Overview

The aircraft had 2 separate electrical systems; One for the sensors together with the wireless module downlink and the other for the airplane's manual control and propulsion systems.

Specifications: Electrical System and Kit Components

The table below lists the additional components that were bought for the airplane's manual control and propulsion systems:

Quantity	Item	Function
4	5G TowerPro SG50 Micro RC Servo	Directly controls push rods to manipulate control surfaces.
1	45A Electronic Brushless Speed Controller [ESC]	Converts throttle signal from radio system to a form that can power the motor.
1	11.1V 3200mAh 15C "Exceed RC Fusion Power Series" High Performance Lithium Polymer Battery	Powers the motor, receiver and servos
1	2-3cell charger Switching Adapter [DK-C1000-S]	Powers the charger for the LIPO battery
1	2-3cell charger Balance charger [DK-C1000]	Balance charges the cells in the LIPO battery
1	20 A maximum draw Brushless Outrunner Motor [E8-S-10, (480 size)]	Powers the propeller
1	8x6 APC-E Electric Airplane Propeller	Provides thrust to aircraft

Table 1: Additional Components Bought For Construction Model Aircraft

The aircraft's main power/propulsion system consisted of the following:
One 11.1 V 3200 maH LiPO battery (Nitroplanes)
One 1000kv 20A brushless outrunner motor (Common Sense RC)
One 45A Electronic Brushless Speed Controller (Common Sense RC)

In choosing components for an RC airplane, and it is assumed any airplane in general, careful attention has to be paid to the flying weight of the aircraft, typical flying speeds required, size/ power consumption of the motor, propeller size, speed controller ratings and battery voltage ranges.

Most often, the weight of the aircraft along with the desired speed and propeller size, determine the choices for the remaining components. The usual trend is that the heavier and quite possibly the bigger the airplane, the larger the propeller needed. Also, the faster and longer one wishes to fly, the larger the voltage and the milliamp hour rating the battery needs to have respectively

Our component search began by considering the size of the airplane and the size of the propeller. A 9 inch long propeller was chosen in order to get maximum propulsion while avoiding propeller strikes on landing and takeoff. Speed was not a big issue but in the event that heavy avionics were used, more speed would be needed to stay in level flight. This additional speed could be provided by an electronic speed controller and motor with higher current ratings. Because of the aircraft's light weight and large wingspan, this particular RC model also functions as a glider. That is to say that it can be flown without the use of the engine for extended periods of time as long as it has sufficient altitude and can ride any waves of hot, rising air. With all of this information, an electric motor which draws a continuous current of 24 Amps in normal operation was purchased.

The next step was choosing an electronic speed controller (ESC). The ESC converts the throttle signals to voltage outputs which drive the motor. ESCs have a continuous current rating for normal operation as well as a slightly higher maximum burst current. This burst current may be used in a climb where more power is needed. While it is theoretically okay to use an ESC with a current rating of that of the motor, it is usually safer to choose an ESC with a current rating which is higher than that of the motor. One can never tell when there may be an unexpected extended surge in motion by the propeller which could cause the motor to draw more power.

Usually in such a case, the motor will burn up, sometimes closely followed by the ESC. If the ratings of the two devices are very close to each other, the very high currents passing through the ESC may cause it to overheat and malfunction, sometimes permanently. These chains of events will render almost the entire power system useless and it unfortunately took place in our case. After destroying 1 motor and 2 ESCs, a working propulsion system was developed. A 45 Amp ESC (with 55 Amp burst current capacity) was chosen. It gave us a 20 Amp margin of safety with respect to our motor. To ensure that none of the electrical components burnt out from overheating, the cowling of the aircraft was removed and the ESC was tied externally to the tail of the aircraft to enable continuous circulation of air around the ESC, thereby allowing it to self-cool. The ESC was not programmed. Hence, it kept its default settings since they were suitable for our needs.

Finally, the battery was chosen. It was an 11.1V lithium polymer based battery which was good for its high capacity and longevity. The ESC would have worked with a specific range of battery voltages depending on the battery type. The more important consideration made in choosing the battery was the milliamp-hour rating. The higher this value, the longer the battery would have lasted before dying. However, a tradeoff had to be made since the physical size and weight of the battery increases as the milliamp-hour rating increases. A higher than normal rating for the battery was chosen at 3200 maH but its weight and size turned out to work perfectly in terms of fitting the battery in the nose compartment and regulating the balance of the aircraft about its center of gravity.

Note: Extra components had to be bought such as thick wire to carry large currents from the battery to the ESC, servo y-connectors for the ailerons, and heat shrink tubing to enclose live wires. This list is not exhaustive.

System Design

Choosing a means of autonomous control

Our initial plans for autonomous control were based on current UAV technologies in use by organizations such as NASA and other high end open source UAV groups. The core of the flight system was to be located on the airplane itself while the ground station received telemetry and updated flights paths via a wireless uplink. This idea was eventually discarded since a system where one could switch between manual and computer control when deemed necessary was desired. The main technological gist of these commercial systems was that the 'manual' and autonomous capabilities were built into a single entity and placed on board the aircraft.

Final Solution – Ground-Based Flight Computer

Having a ground based flight control system was the best option for our project. With this system, one could easily switch between autonomous and manual control at the literal flip of a switch and have no need to worry about restarting any one of the communications systems. There was always an uninterrupted flow of data between the ground station and airplane thus guaranteeing 'full' control at all times.

It was ultimately was decided to execute autonomous flight by using sensors to wirelessly obtain data about the orientation and position of the aircraft whilst in the air. In addition, the RC transmitter would be hacked so that data could be collected about the positions of the control sticks on the transmitter that corresponded to the motions of the aircraft. These sets of data would then be used to design a controller for autonomous flight of the aircraft which would be loaded onto a computer to enable the aircraft fly on its own. A switch which would also enable one to take manual control of the aircraft via the RC transmitter would be incorporated.

The computer chosen for the UAV system was the 16F873 PIC microcontroller. The PIC, in the form of a 28-pin dual inline package, was chosen for its relative simplicity in both its programming and physical handling. The PIC was also readily available in the labs in Hicks and had the additional advantage of being very cheap. The PIC was responsible for things ranging from collecting and processing data to actively controlling external devices. Both the aircraft's avionics system and the ground station had a PIC as their core, the roles of which will be explained in more detail later.

Avionics

Main Avionics Components

Sensors

The avionics sensors included:

1) A 3 axis accelerometer for measuring tilt.

2) A pressure sensor to measure altitude.

3) A digital compass for heading information.

Communications

The airborne hardware included the following communications devices:

1) Zigbee wireless module for flight data downlink.

2) Traditional RC Receiver for manual control.

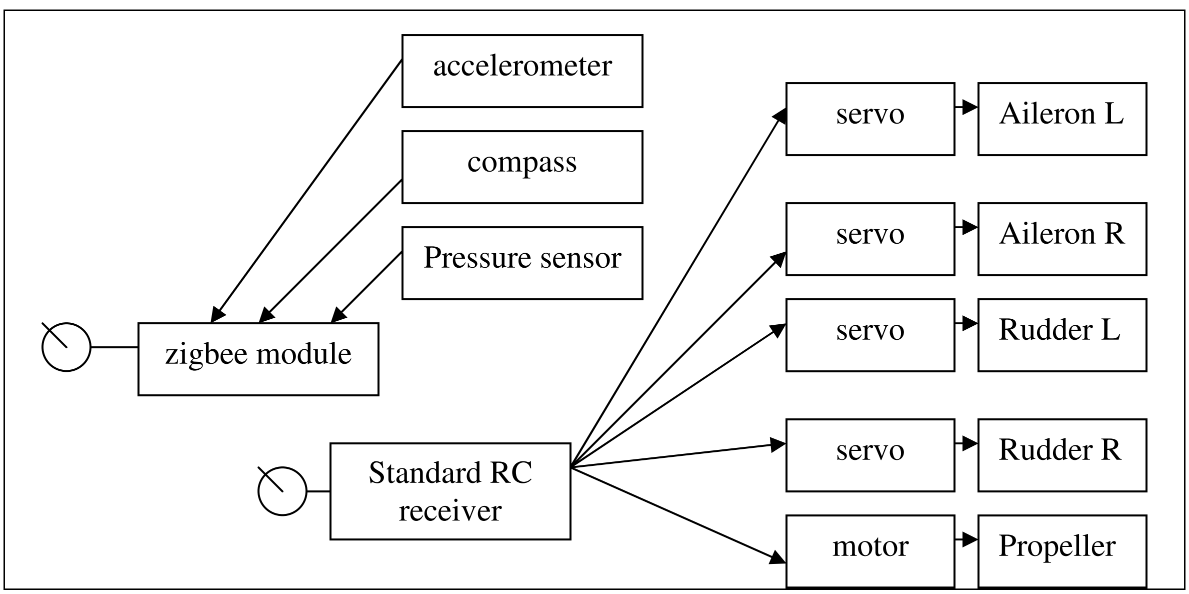


Figure 6: Outline of Path From RF Link to Control Surfaces Onboard Aircraft

Avionics Sensor Options

The sensors that were acquired were chosen with the intent of replicating the sense of sight that humans possess and require in order to fly aircraft safely. In order to detect the position and spatial orientation of the airplane, the following sensors were obtained:

Honeywell HMC6352: 2-Axis Digital Compass

This device provided us with a good estimate of heading. An initial issue with this device stemmed from the fact that it was 2 axes in nature and hence would have potentially given erroneous values if it were tilted off the horizontal to some extent.

VTI Technologies SCP1000 Pressure Sensor

This absolute barometric pressure sensor gave a rough value for the altitude of the airplane. It was not perfect since although in ideal conditions it can only resolve a vertical difference of 10cm of air. When in flight, the sensor would have been exposed to incident wind if not properly enclosed thus affecting the static air pressure value for a certain altitude. It was decided not to convert the raw pressures to an altitude since atmospheric pressure varies with ones location on the earth. For purposes of autonomous flight in this project, we would have used a short range of pressures to simulate a desired altitude.

VTI Technologies SCA3000 3 - Axis Accelerometer

The accelerometer measured acceleration with respect to the direction of acceleration due to gravity. An algorithm was written to convert the raw accelerator output values to mimic those seen on an artificial horizon instrument of any aircraft. The accuracy was within one degree. The following table summarizes the usable outputs of the accelerometer.

Pitch value		Roll value	
Pitch up	0 to -90	Roll to starboard	0 to +90
Level flight	0	Level flight	0
Pitch down	0 to +90	Roll to port	0 to -90

Table 2: Pitch and roll values for the airplane computed from customized algorithm

Note: These values are based on the orientation of the avionics board relative to the airplane. In our project, the avionics board was placed along the length of the fuselage with the Zigbee module end placed towards the nose of the aircraft while the PIC end was to the rear.

While the accelerometer performed as expected when it was in a static position, there was a potential problem when the device was in motion. In a banking turn, there is the possibility of a large centripetal force being exerted on the aircraft and in turn the accelerometer. This can override the typical gravitational acceleration exerted by the Earth and affect the acceleration measurements by superposition of unwanted centripetal accelerations. Further inspection of the data sheet showed that our accelerometer in fact had a potential use in inertial navigation systems since it had a +/- 2g range of measurement. If the range were 1g or less then the device would not have been able to compensate for centripetal accelerations in banking turns. Hence the greater the range of the accelerometer in g's, the more complex accelerations could be exerted on it without a decrease in accuracy for uses such as inclination measurements. So far, there have not been any problems with this phenomenon.

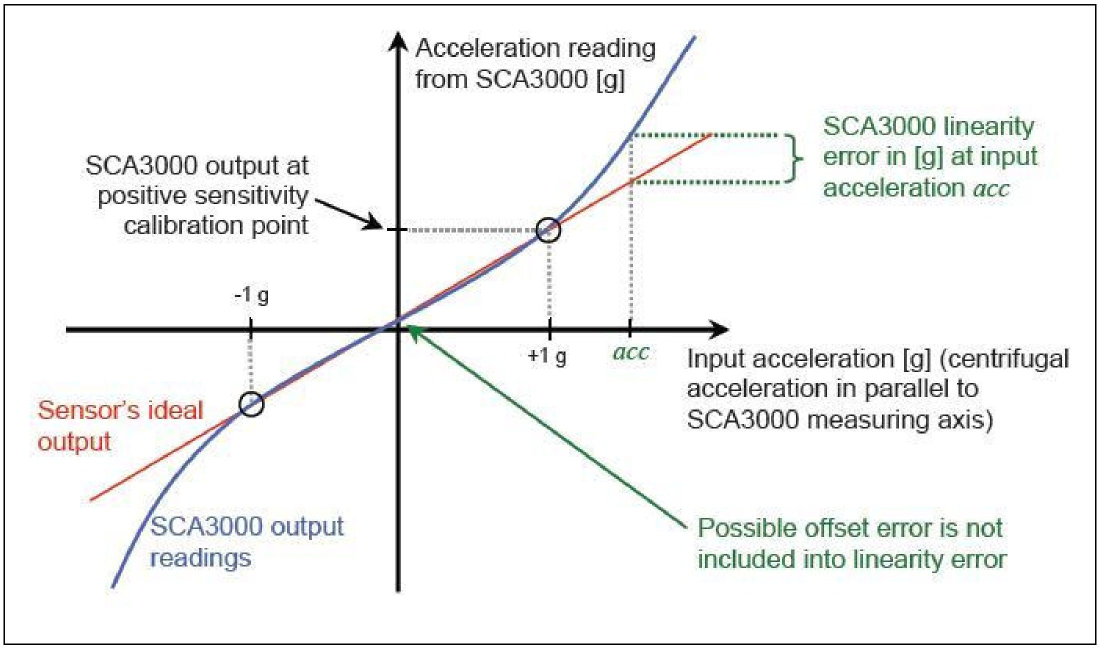


Figure 7: Graph showing usable range for accelerometer. Note our accelerometer had a range of +/- 2g. Illustration was taken from SCA3000 data sheet.

Choosing Sensors based on their Bus Protocol

During the early stages of the project, the sensors for the avionics were chosen mainly based on price, and availability of a digital output. The reason for the choice of digital sensors was that extra analog to digital conversions were not necessary in the PIC. In terms of digital sensors, there were two available formats: those meant to work with the Serial Peripheral Interface bus (SPI) and those which used the Inter-Integrated Circuit (I^2C) bus. The SPI is full duplex unlike I^2C and is conceptually simpler. However, a problem arose in setting up the SPI in the preprocessor statements in the C code. SPI has various modes of operation depending on the peripheral, and strict adherence to any other specifications for the slave device must take place. Consequently, the setup of the SPI is somewhat tricky despite the fact that the executable code was significantly simpler and required less statements than that found in I^2C. Furthermore, since most of the initially acquired sensors were I^2C devices and proper functioning of these devices was attained, it I^2C was the common format chosen for all the devices and code used in this project. That marked the end of any use of SPI devices.

The I^2C Bus Protocol

I^2C or Inter-Integrated Circuit is a multi-master serial bus which was developed by Philips in the 1980s. It governs the means by which data is transferred between the microcontroller and the various sensors or peripherals. Physically the bus consists of two bidirectional data lines; one being for serial data (SDA) and the other a serial clock line (SCL). These lines are open drain and are pulled up to Vdd by resistors.

In software, the data transfer process starts when the PIC issues a start condition. The PIC then sends an address down the SDA line corresponding to only one device or slave on the bus. This address also indicates whether the operation is going to be a read or write. When the particular slave receives the address it responds by sending an acknowledge bit back to the PIC. In our code, the portions for checking for and sending acknowledge bits were eliminated as it was deemed relatively inconsequential. In most cases, acknowledge processing statements were substituted with short delay commands on the order of 1-10 ms. After this took place, the PIC sent whatever data was necessary to the slave device. Each data transfer was 8 bits long, most significant bit first. Usually, when one needed to read data from a slave, a restart condition was issued followed by an address transmission which also embedded data indicating a read was about to take place. A read statement would then be issued by the PIC and it would read the data which the slave was sending. At the end of the process, the PIC sends a stop condition and the bus becomes free again for use by some other slave if present.

Avionics Computer

The PIC onboard the aircraft was solely responsible for collecting data from the accelerometer, pressure sensor and digital compass, combining and packaging that data, and porting it to the Zigbee Pro® module for transmission to the ground station.

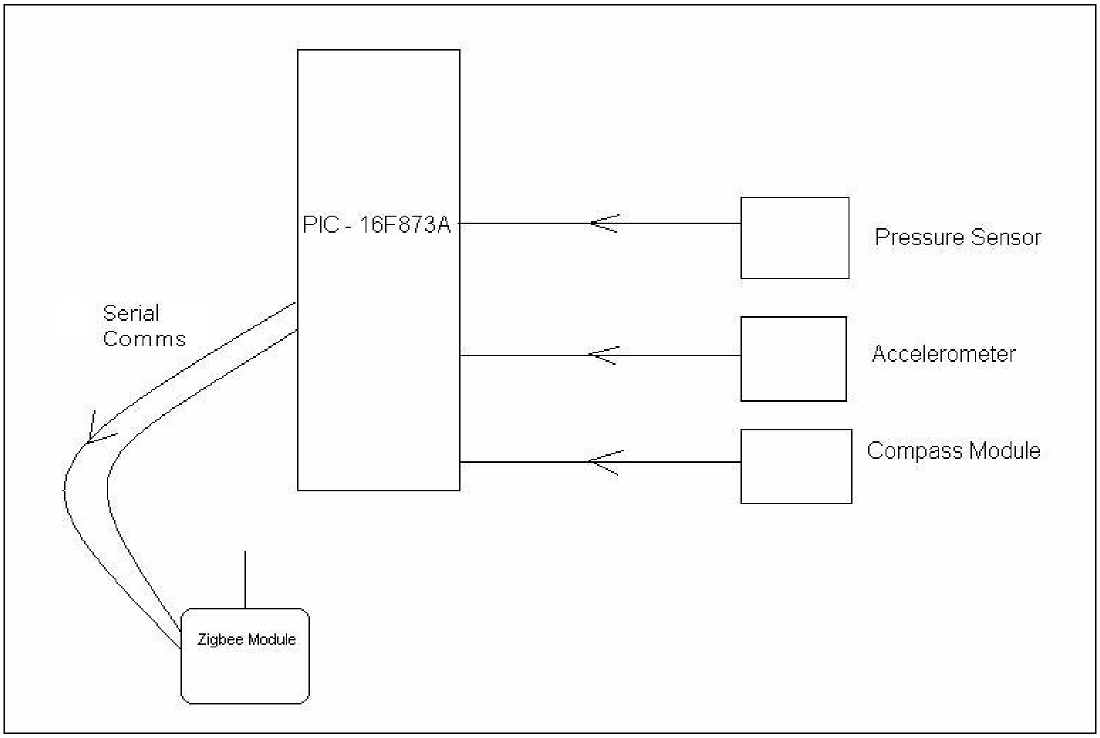


Figure 8: Outline of Data Transfer in Avionics

Communication between each sensor and the PIC was performed using the I^2C protocol. In the code, there was at least one function which pertained to each sensor whose purpose was to get data from the said sensors. In the main function, all the measured data were concatenated using tabs and ‘printed’, thereby sending the data automatically out of the RS-232 transmit pin of the PIC. The data was combined in such a way that when it arrived at the ground station it was already filtered into columns, paving the way for easy manipulation in programs such as Matlab®.

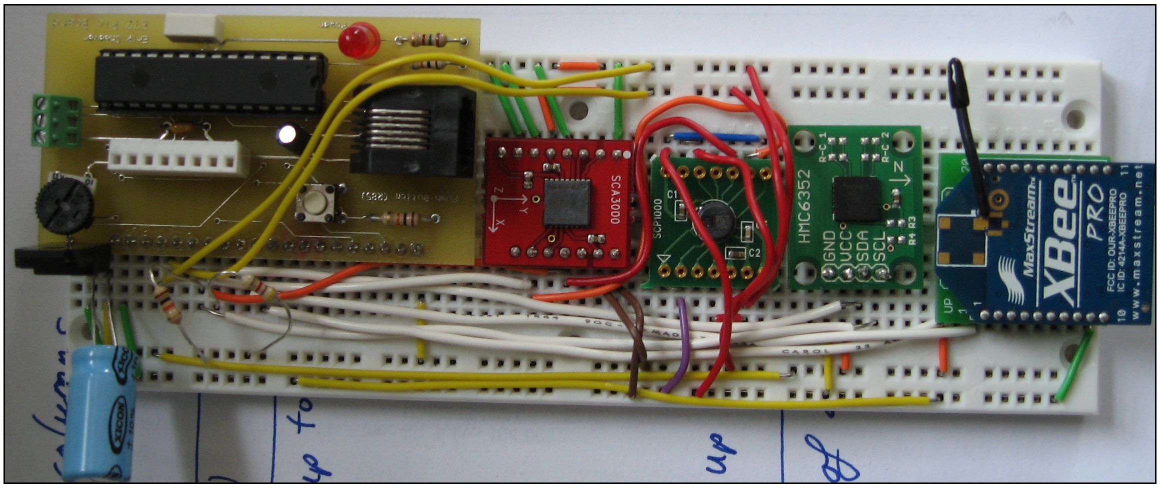


Figure 9: Prototype Avionics Board Showing The PIC in the Upper Left Hand Side

PCB Design for Avionics

A PCB was designed in Simulink to hold the avionics circuitry. This idea was introduced so as to reduce the overall weight of the payload on the aircraft; The breadboard on which the avionics circuitry was initially built was much heavier than the PCB. Below is a schematic of the circuitry in Simulink. The digital compass, pressure sensor and accelerometer did not exist in any of the libraries in Simulink. As such, they were designed in Ultiboard. After the schematic was completed in Simulink, it was exported to Ultiboard where the components and wiring were arranged optimally. An image of the optimum wiring of the schematic is shown in Figure 10. Despite checking the components on the PCB schematic several times and confirming the accuracy of the design, when the devices were soldered onto it, some of the sensors got destroyed. Again, after careful inspection of the PCB and verification that the connections were right, it was decided to go back to using the breadboard for the avionics circuitry.

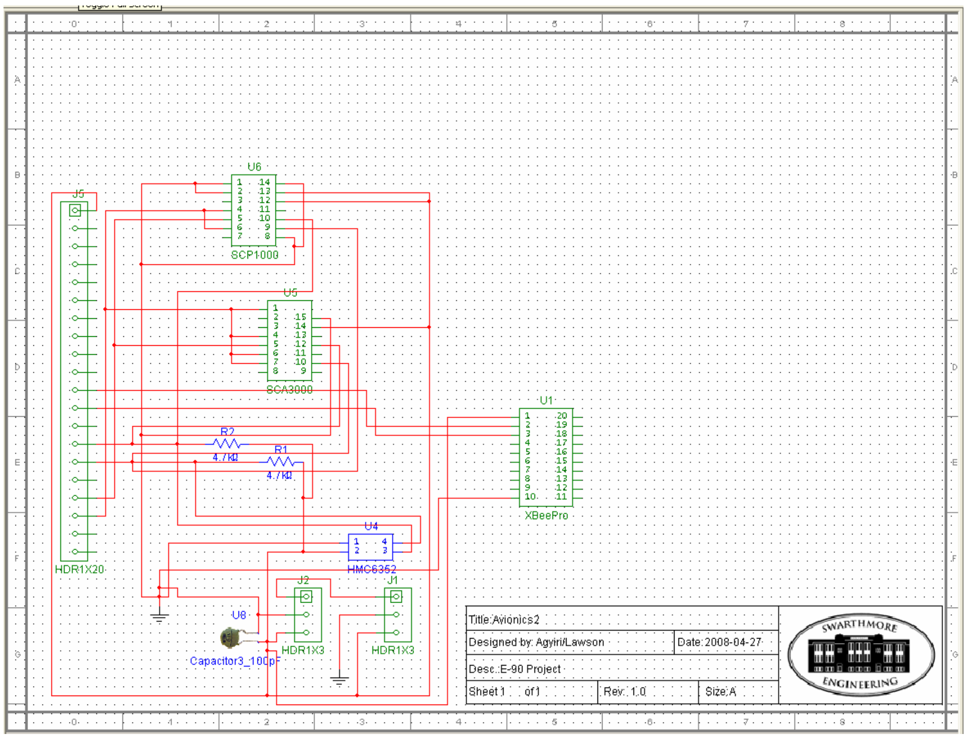


Figure 10: Multisim Layout of Avionics

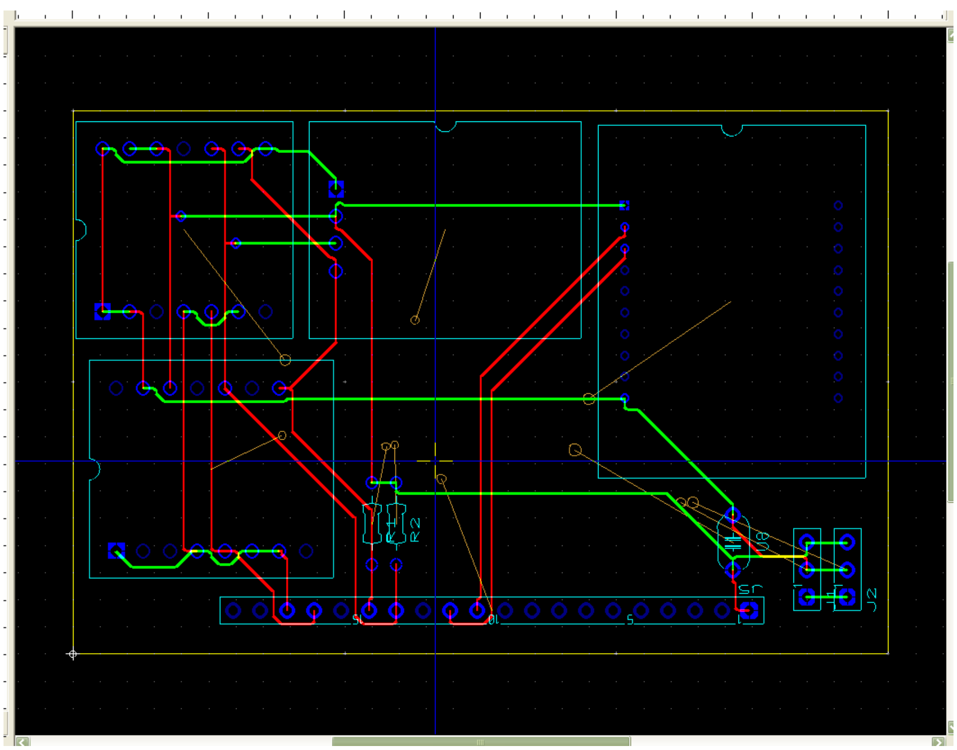


Figure 11: Ultiboard Layout of Avionics

Choosing a radio system

The radio system chosen was the Futaba 2.4Ghz 6EX. The first consideration was that of the operating frequency. Since the thought of controlling the airplane by modifying signals in software had initially crossed our minds, it was agreed upon that this platform would work well alongside the 2.4Ghz Zigbee data downlink (and possibly uplink). The initial plan was to have the main flight computer on board the airplane and as such a sturdy data uplink for updating waypoints and so forth would have been required.

In that case, the manual transmitter would have been off while the flight computer was flying the airplane thus preventing any interference that could have occurred between the Zigbee and the Futaba systems. This was a mistake. Ideally one would range test a radio system before use to ensure that the transmitter can control the airplane. However, with the airplane hundreds of feet away in the air, one would have needed a means of switching between manual and autonomous control where the switch was on board the airplane. Furthermore, assuming that such a switch could have been devised, there was no perfect means of ensuring that manual control of the airplane could have been regained after switching away from the flight computer.

In the end, with our grounded flight computer setup, the Zigbee and Futaba communications systems were both operating at the same time. After some thorough tests, it was noticed that there was no radio interference between the two devices so everything worked well in that regard.

Other minor considerations included the number of channels in the Futaba system. Our airplane required 4 channels corresponding to all the control surfaces and the motor. A 6 channel system was purchased such that in the event that an extra feature such as a movable landing gear was needed, then it could be quickly added to the aircraft.

Choosing a wireless communication system

The wireless communication system used for transmitting data from the avionics onboard the aircraft to the ground computer was Zigbee via the XBee Pro Module, a picture of which is shown below:

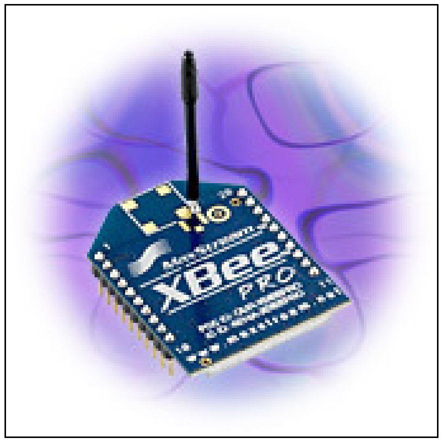


Figure 12: Xbee Pro Module by Maxstream [10]

The connection of the module to the computer on the ground was via rs232.

This Pro module was chosen as opposed to the regular XBee module because it had a longer operation range (1 mile) whereas the regular XBee had an operation range of 100m. As such, the aircraft was able to cover much greater distances than it would have had the regular XBee module been chosen instead.

Ground Station

General Overview

The main roles of the ground computer are to collect the serial stream of data coming from the airplane, forward the respective portions to various control loops and output the results to a digital to analog converter (DAC). Like the sensors on the airplane, the DAC was also an I^2C device.

Digital to Analog Converters (DACs) Used

Four 8-bit Maxim MAX517 I^2C digital to analog converters (DACs) were used to convert PI controller outputs with 8 bits of resolution to analog voltages. 8 bits were adequate for our purpose since the range of voltages across the potentiometers in the transmitter ranged from approximately +1V to +4 V. The 8 bit resolution grants 256 unique possible analog voltages and this is more than what was needed for the 3V range. I^2C was chosen as the data transmission protocol primarily because of our competence with its operation. Data transfer speeds were not much an issue since in normal (low speed) mode, the baud rate is 100 kbps. Furthermore, transfer of data from the input registers to the output registers of the DACs occurred in approximately 5µs. This is essentially the speed at which the desired voltage would be output after an update. The DACs can be configured for one of the two output voltage ranges, from 0V to +2V or from 0V to +5V. These configurations depend on a voltage reference which could be either external or internal. The internal reference voltage is +4V. The maximum output voltage for the DACs was +5V. The output voltage is given by

$$V_{OUT} = (N \times V_{REF}) / 256$$

where V_{REF} in our case is an external voltage reference equal to +5V, N is the input code calculated by the PI or P controller algorithm, and V_{OUT} is the analog output voltage supplied to the transmitter. For control of the aircraft's control surfaces and throttle, an output voltage range from 0 to +5V was used but later limited that range to +1 to +4V to mimic the range of voltages in the transmitter potentiometers in normal operation. One could only use a maximum of 4 of the MAX517 DACs on a single I2C Bus due to the limited number of address bits per DAC. Each DAC has an address comprising 2 bits and this gives a total of 2x2 = 4 possible addresses. The address of each DAC was chosen by connecting the AD0 and AD1 pins to either ground or Vdd depending on the address one desired to assign. In the I^2C code, each DAC address had a component which corresponded to the pin connections one previously made. Vdd represents a binary one while ground represents a zero. Despite the limited addressing, the setup fit our needs perfectly since we ideally wanted control of 3 control surfaces and the throttle giving a total of four devices.

Final Solution – Ground-Based Computer Design

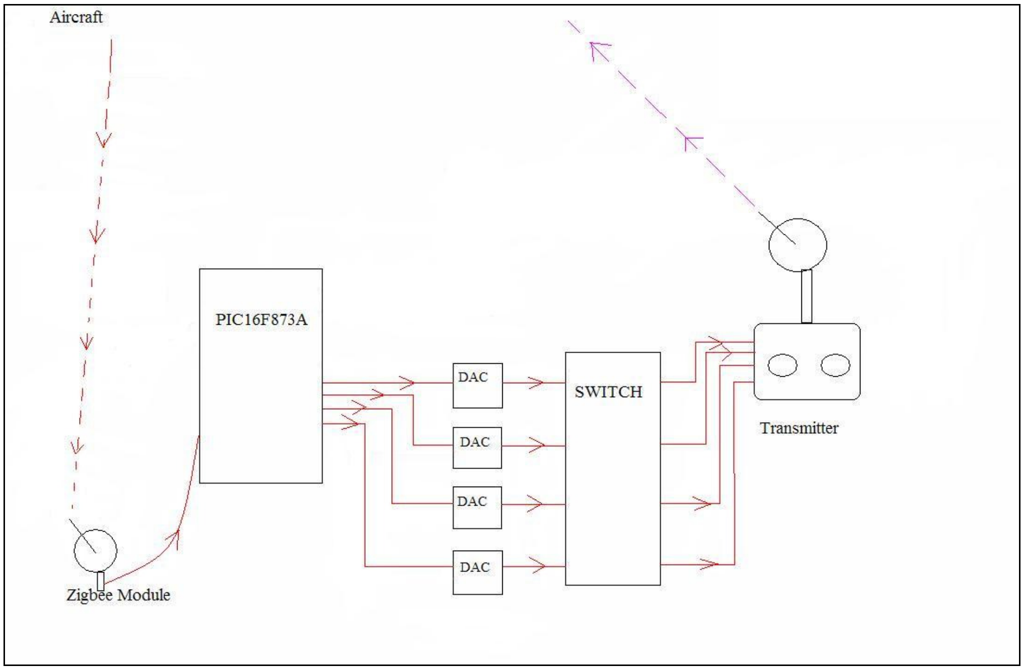


Figure 13: Layout of Ground Station

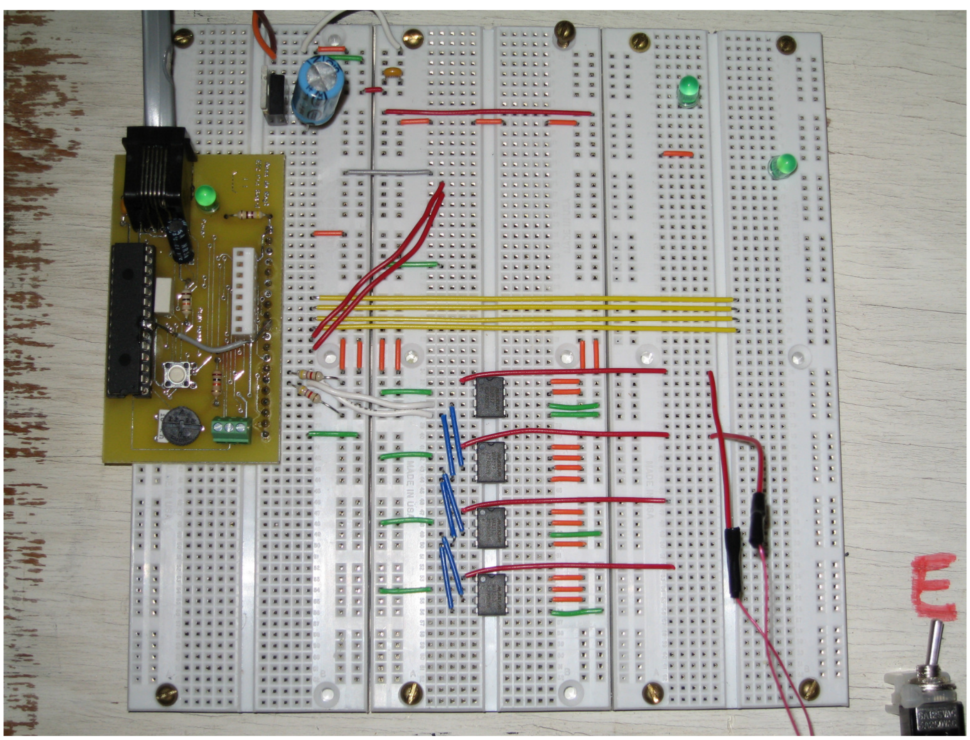


Figure 14: Ground Station Hardware

The diagram above shows a picture of the ground station. The autopilot program was loaded onto the PIC. Depending on the spatial orientation of the aircraft, the autopilot program sent appropriate signals to the transmitter via digital to analog converters and a master switch. The internal transmitter connections were modified so that voltages coming from the digital to analog converter could have simulated inputs that would have normally come from the potentiometers inside the transmitter (which perform the actual control of the aircraft).

A switch was devised so that one could easily change between manual control of the airplane via the control sticks and computer control via the ground station.

Inside the Transmitter

In order to hack the transmitter potentiometers, the wire leading from the potentiometer into the system board was cut. This wire was extended out of the transmitter to a switch. There were 4 switches in total: three for the three control surfaces on the airplane plus one for the throttle. Each switch had two inputs and one output. The two inputs comprised the wire coming from the potentiometer and the wire coming from the digital to analog converter. The output consisted of the wire leading into the transmitter's system board. It was found to be absolutely essential to have a common electrical ground in the ground station system and so an additional wire was added to link the ground on the system board of the transmitter to the ground station breadboard ground rail. The two electrical grounds initially were a result of different power systems for the transmitter and the breadboard

Note: The jumpers on the transmitter system board to which the wires are attached are labeled ST1 through ST4. These labels correspond to the 4 joystick inputs. When coding the controller in the flight computer, care has to be taken so as not to cause the DACs to exceed the average maximum and minimum voltage outputs from the potentiometers.

Device	V_{low} / V	V_{mid}/V	V_{high}/V	Notes
Elevator	0.94	2.41	3.91	V_{low} = full down stick
Aileron	0.94	2.57	4.11	V_{low} = full right stick
Rudder	0.88	2.51	4.16	V_{low} = full right stick
Throttle	0.92	n/a	3.98	V_{low} = full throttle*

Table 3: Experimentally found Voltage Ranges in Transmitter Potentiometers

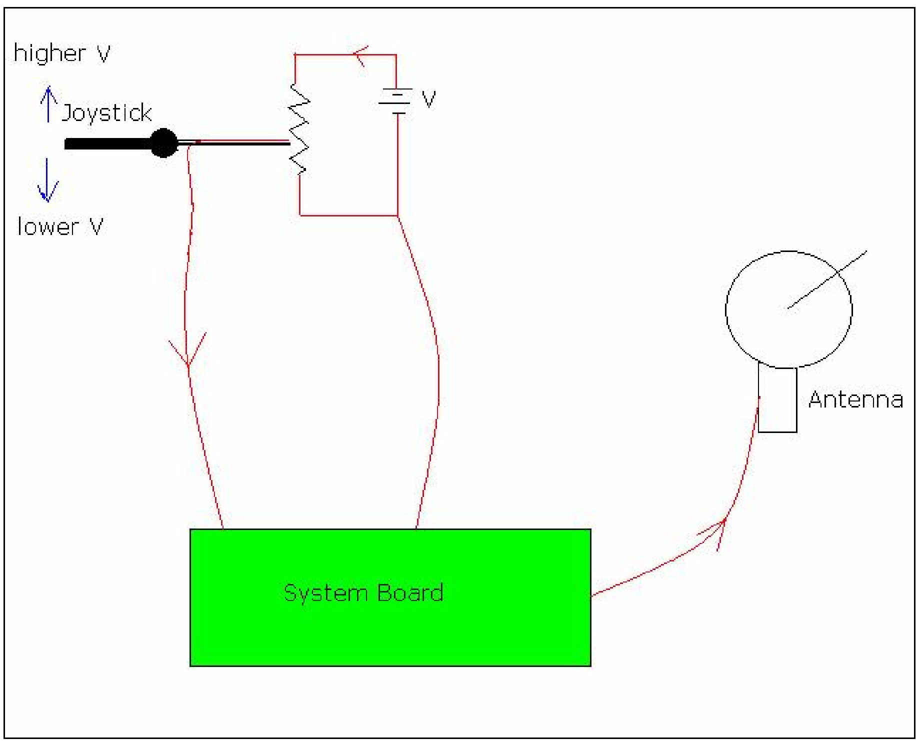


Figure 15: Illustration of the Electrical Connections Involving a Single Potentiometer

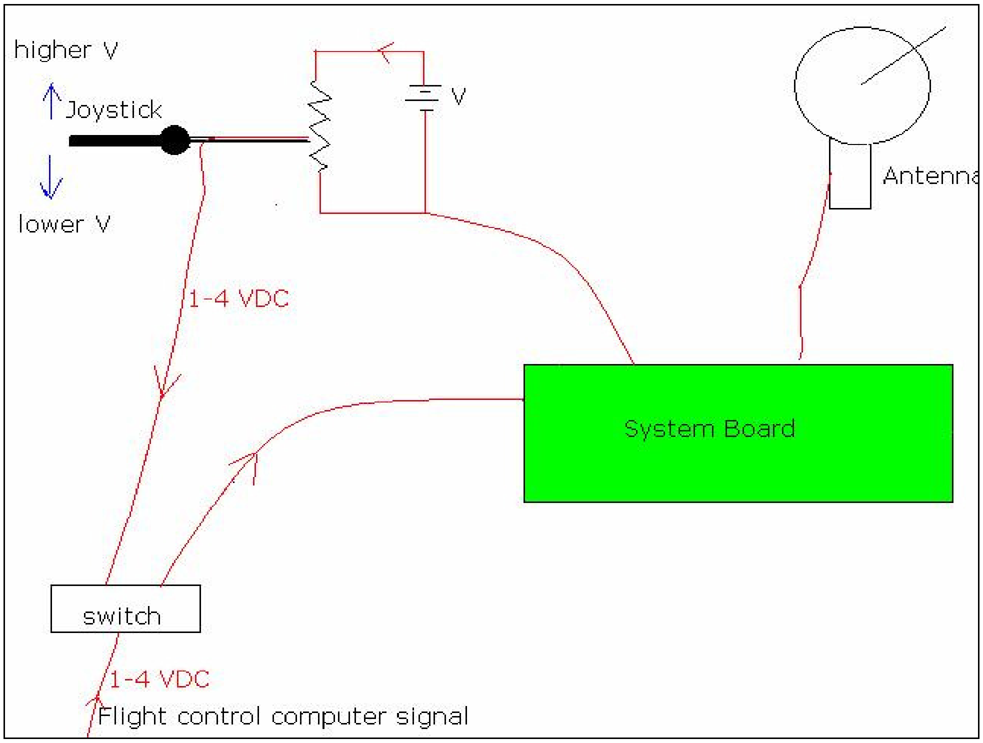


Figure 16: Illustration of The Electrical Connections Involving a Single Hacked Potentiometer

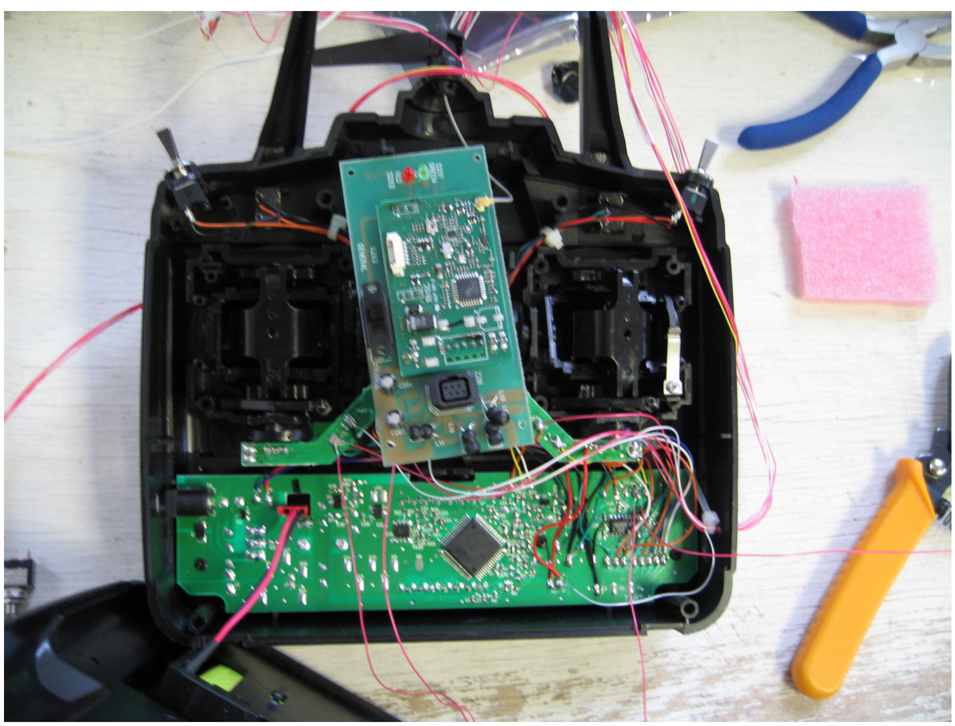


Figure 17: Hacked Transmitter

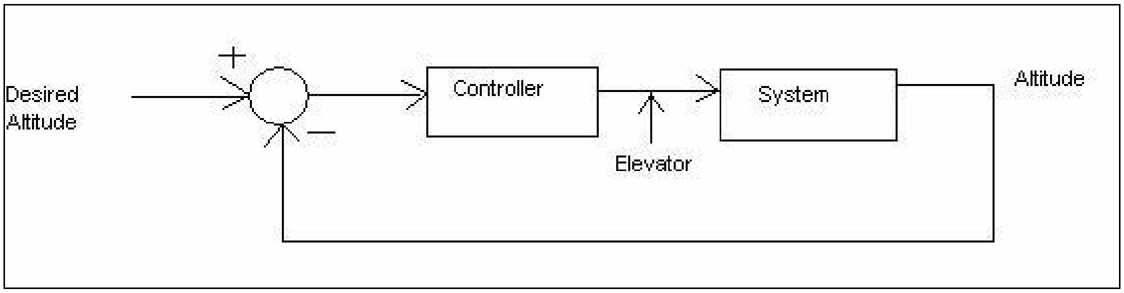
Proportional Controller

A proportional controller was chosen in project for reasons of simplicity especially with respect to coding. It was also capable of carrying out the function desired. Because of limited time we decided to implement only one proportional controller to control the elevators and hence the altitude of the aircraft. The aircraft crashed during a data acquisition flight session. The crash created enough damage to the plane such that it was impossible to rebuild; a new aircraft had to be bought and assembled but time constraints did not permit us to do so. As such, the data needed to design the controller could not be obtained.

Assuming the crash had not occurred, the data from both the transmitter voltage readings and the avionics would have been used to develop the controller. The data from the transmitter would come from the hacked system. For example, in designing a controller for the elevators and thus the altitude of the aircraft, the voltage readings from the potentiometer controlling the elevator as well as the corresponding data collected from the pressure sensor (a measure of altitude) onboard the aircraft would be used to design a controller. This procedure is described below using this data collected from another source.

Soren (our pilot), created a file from a flight simulator/training software package called ClearView R/C. There are several types of aircraft that can be flown with this software. The trainer with aileron control that had similar characteristics as our UAV was chosen. The durations and aircraft responses were used to create a file that was similar to data collected by our UAV. The file contained a 90 second full clockwise pattern i.e. a rectangular path. The data collected is shown in Appendix D.

The negative feedback system used to model the proportional controller is shown below:



The basic operation of the controller was as follows: Assume that the aircraft initially had an altitude of zero while the desired altitude was relatively high. The actual altitude would have been measured by the pressure sensor while the desired altitude would have been fixed in the autopilot code. As a result of the negative feedback, the input to the controller, the difference between the desired altitude and the actual altitude, would have been large. The proportional controller thus would have input a proportionally large value (voltage) into the system (elevators) resulting in an increase in the actual altitude of the aircraft. This would have caused the difference between the desired altitude and actual altitude to decrease. Thus, the proportional controller would have input a relatively smaller value (voltage) into the system (elevators). This process would have continued until the difference between the desired and actual altitude was zero. At that point, the aircraft would have reached the desired altitude.

Note: The throttle would have had to be monitored during such a climb in order to prevent the aircraft from stalling.

In order to determine the order of the system, a graph of elevator and altitude versus time was plotted. The graph is shown below:

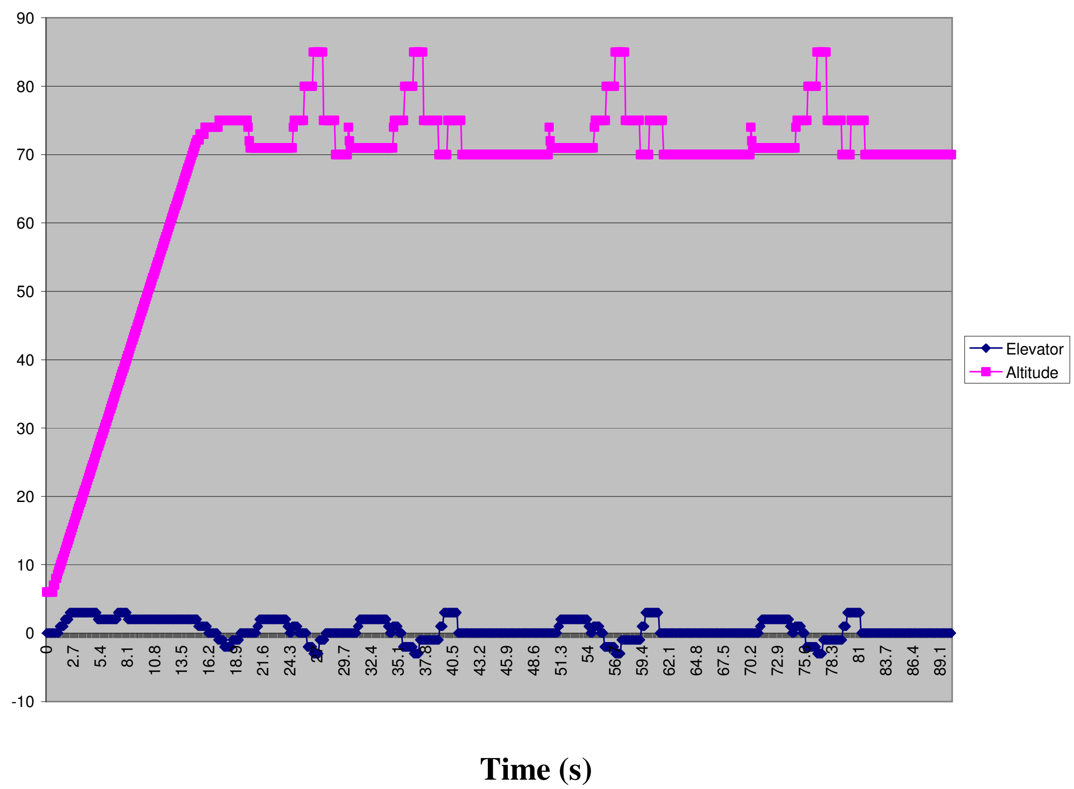


Figure 18: Plot of Altitude and Elevator vs Time

The shape of the altitude graph resembled that of a second order under-damped response. Thus, the transfer function, H, used for the system was of the form:

$$H = \frac{K}{s^2 + 2\zeta\omega_n s + \omega_n^{\,2}}.$$

The lsim(system, input(s), time vector) function in MATLAB was used to help us determine the values of the constants: ζ and ω_n; K was set to 1. lsim(system, input(s), time vector) simulates a linear system. The code used in determining the values these values is shown in Appendix D. lsim(system, input(s)) works by taking three kinds of arguments, the system, input(s) and time vector. One can change the values of the input(s) and MATLAB will produce the time response of the system.

The algorithm designed varied ζ and ω_n over a range of N = 1001 values each. Since the system being modeled was that of a second order, this meant that the lsim(system, input(s)) calculations would take $O(N^2)$ time; Each lsim(system, input(s)) calculation took approximately 1 to 2 seconds. This meant that in our situation, the program would have taken approximately 1,002,001 to 2,004,002 seconds to complete.

The simulation took a very long time to run and did not produce any values after 3 hours of running. As such, no values for the constants, ζ and ω_n were obtained.

Performance

Testing and Results

Ground testing

Avionics board

The avionics board supplied data successfully which was intercepted by the receiving zigbee module. The sensors were programmed to output a count value, the roll and pitch values of the airplane, the aircraft's heading, the external barometric pressure, and the temperature in that order per data string.

Ground station

The PIC could successfully control the control surfaces on the airplane as well as the throttle by linking the PIC to the transmitter via DACs. A looping test program is included in the appendix. A PI controller was already written in C and required the inclusion of the proportional and integral constants to work. These constants would have been derived from analysis of actual flight data

Note: Due to time constraints, the serial flight data collection and parsing code was not successfully tested but a prototype involving interrupts is included in the appendix. This *would have been the final component necessary to result in closed loop for autonomous flight control.*

Flight testing

In order to fly any experimental aircraft, one should have sufficient flight experience especially with aircraft similar to the one being tested. Furthermore, attention should be made to FAA rules pertaining to UAVs. Examples of restrictions include flying only at sanctioned airfields and not having the airplane fly above 400ft above ground level (A.G.L.). Consequently a membership with the Academy for Model Aeronautics had to be obtained and a pilot at a RC airplane club had to be found to fly our airplane. It took some time to find a good club nearby but eventually contact was made with the Delaware County Radio Control R/C Model Airplane club. Mr Soren Spring, a member of the club, flew our airplane.

Launching

The aircraft could not take off from the ground because there was no suitable runway available. The runways that were in close proximity of the college were either not smooth enough or were lawns. Since the aircraft had very tiny wheels, it required a smoothed paved surface from which to take-off and none could be found. As such, the plane was hand-launched by our pilot Soren Spring. Hand launching was very dangerous for an aircraft such as ours since its propeller was situated at the rear of the aircraft. Another difficult maneuver involved having Soren hold the transmitter and hand launch simultaneously. He nonetheless executed perfect launches because of years of experience. The picture below shows him hand-launching the aircraft.

Figure 19: Pilot Hand Launching Aircraft

Test Flight Phases
Phase 1) Airworthiness test – No payload

This test involved flying the aircraft without any payload. It was carried out to ensure that all the parts of the aircraft had been securely fastened and aligned. Another goal of this flight was to ensure that the electrical components purchased for this aircraft would keep it in the air with no problems. During an ascent of the aircraft in the third flight of this phase, its motor burnt out. This was not due to the ratings of the speed controller and the motor used on the plane; the continuous current rating of the speed controller was greater than that of the motor so the speed control was able to handle the power requirement of the motor. A possible reason for this burnout is that the cowling of the plane did not provide a circular/continuous flow of air over the speed controller, which is a device that heats up very quickly to high temperatures. The speed controller also had a high temperature cutoff so that if it overheated it would have shutdown in an attempt to protect itself from damage. Part of the motor was also covered by the cowling. The build-up of heat in the cowling caused the coils to overheat and burn. Another possibility for this failure is the speed controller and motor initially selected were cheap quality. To solve this problem, the cowling was removed, and a better quality speed controller and motor were bought. The new speed controller had a continuous rating of 40A and the motor had a 25A rating. These specifications ensured the electronic speed controller could meet the power requirements of the motor.

Phase 2) Payload check flight test

This test involved using a dummy payload, a metal bar in this case, which had a weight approximately equal to that of the actual avionics breadboard. This test flight was successful.

Phase 3) Initial data collection flight

The purpose of this flight was to collect voltage readings from the RC transmitter as well as sensor readings from the avionics circuitry whilst the plane was in flight. During the test flight, the wire coming out of the RC transmitter and connecting to the common ground on the ground station became disconnected. This wire removal was confirmed by the transmitter data which was collected and indicated unusual zeroes in at least 2 control surfaces. The pilot ultimately lost control of the aircraft within 10-15 seconds. Due to erratic, uncontrolled behavior of the aircraft and the relatively heavy avionics, the starboard wing snapped due to high wing loading and the airplane quickly nosedived and crashed into the ground. Approximately 30 seconds of data were collected from launch until crash but due to the untimed and unexpected nature of the aircraft maneuvers, this data was not used in the controller design.

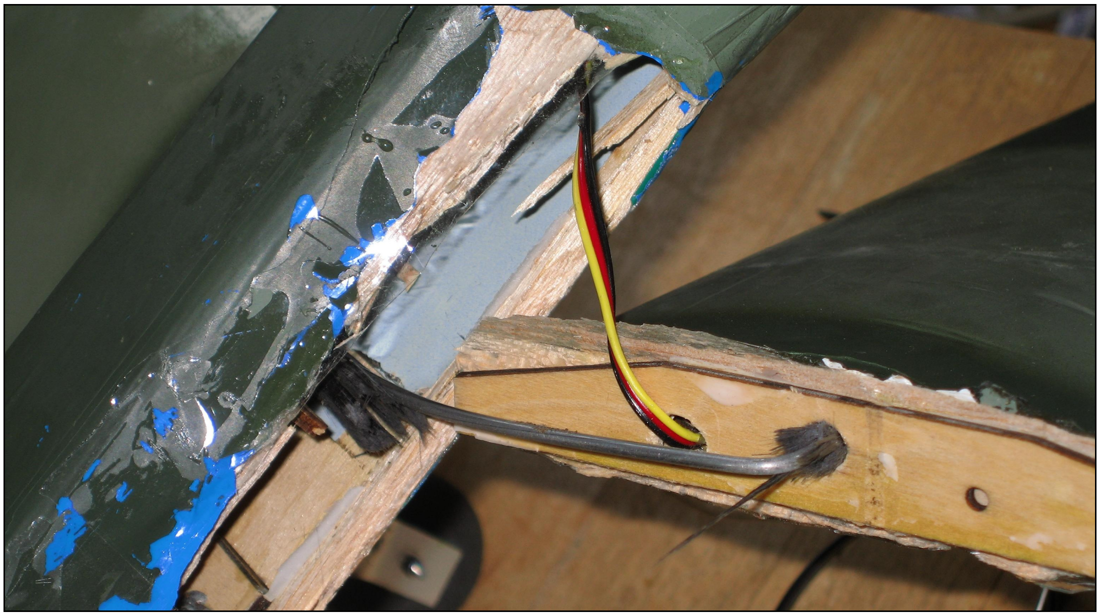


Figure 20: Wing Broken From Crash Showing Shattered Black Carbon Fiber Rod Surrounding the Bent Metal Pushrod (Wing Loading In-Flight Structural Failure).

Phase 4) Autonomous flight check

This flight phase was not carried out because of loss of the aircraft during phase three.

Coding – some lessons learned

As previously implied, substantial time was dedicated to the SPI bus protocol as a means of communicating with sensors. After learning of the impracticality of the varying setups for SPI it was decided to go with I^2C. In the end, it all comes down to a question of taste and comfort with a particular standard. For some, SPI may be easy while for others it may not. A few of the smaller, yet important findings over the coding of the systems included having the right amount of delays in the I^2C procedures. Pushing some procedures too quickly may return incorrect data from a device, if any data returns at all, while large delays in the I^2C process may cause the PIC to hang due to inactivity. One interesting circumstance which arose was the newer 10 bit addressing needed for the accelerometer. Typically slave devices have 7 bit address onto which another final read/write bit was added before transmission. However, the many devices on the market along with the limited number of slave address catered by 7 bits warranted a larger address. In this case, the device address was sent in two transmissions of one byte (eight bits) each. With the 10 bit system, the device address, which is given by the manufacturer, is spliced into another (mostly default) binary number for a total of 16 bits. The 10 bit system seemed confusing at first but was eventually very easy to understand after careful observation of how it generically works.

The PIC-C compiler from Custom Computer Services (CCS) was used throughout in order to program the PIC. During the debugging process, extensive use of the 'monitor' window was used. A specially modified 'USE RS-232 with DEBUGGER' directive statement was used to route any printed data from what would have been the normal serial port to the 'monitor' window within the PIC-C program instead. This precluded the problem of seeing the data via the normal serial port via the typical wired interface since there was no wired PC serial connector on our PIC board/prototyping board. In sum the 'monitor' made it easy to see any data whenever a print statement was called.

We should note, that at the end of the project, we had written off two accelerometers, one pressure sensor, and a stack of DACs mainly because of programming issues. The first set of DACs as well as the pressure sensor was SPI controlled and so they did not work. The accelerometer data from the first two accelerometers was very erratic and the devices themselves required substantial 'setup code' before calling normal I^2C functions to read data. This setup code was responsible for settings such as enabling high data update rates but this tweaking involved a lot of tampering with control registers. If one was not careful, permanent damage may have resulted rendering the accelerometer useless.

Other small problems involving coding were simple as formatting print statements correctly or defining number types appropriately. At one stage of testing, it was noticed that some numbers were being truncated. It was simply an issue of changing the variable in question from an 'int' to an 'int32', the latter of which can represent a much larger range of numbers. Any other minor problems or hints are including in comments in the code in the appendix. Those comments, together with this report, should make it easy for anyone wishing to make significant use of I^2C devices in a project at a later date.

Future Work

A more robust control system (both physically and in software) would have to be built. On two of the three flight phases, manual control failure was potentially a result of broken connections between the transmitter and the ground station. Longer, stronger and more flexible wires for this purpose, especially if the airplane is being hand launched by the pilot, are recommended.

Furthermore, experimentation with other aircraft which may yield more stable flight platforms is encouraged. More work would also have to be done on building a more robust software based autonomous control system. Use of more complex controllers such as the PID may result in a better autonomous platform.

During this project we also considered including an imaging system on board the airplane. This imaging system may be created as a major component of a future e90 project or be simply bought from a commercial manufacturer for quick installation on the airplane. Another idea included construction of a parachute recovery system for emergencies such as loss of transmitter signal.

Acknowledgements

We would like to thank our advisers Professor Erik Cheever and Professor E.Carr Everbach for their advice and support throughout the course of this project. Our gratitude also extends to Mr. Edmond Jaoudi for his help and resourcefulness when it came down to finding suitable hardware solutions for our circuits. A big thank you also goes to Mr. Soren Spring for the many hours he spent helping us set up and fly the plane. We would also like to show our appreciation to Mr. Brian Pasternack, chair of the Delaware County RC airplane club for his initial help in finding Mr. Spring and for his tips on our aircraft systems. Finally, we would like to thank Kofi Anguah and Scott Taylor for their assistance in designing the control system for the aircraft.

References

[1] B.Park, S. Realov, Solar *Powered Wireless Sensor Network*, E90 Final Project, Swarthmore College. May 4, 2006.
http://engin/academics/courses/e90/2005_6/E90Reports/BP_SR_Final.pdf

[2] NXP Semiconductor, *I^2C Bus Specification*, January, 2000.
The I^2C- Bus Specification Version, 2.1. NXP Semiconductor. January, 2000.
www.nxp.com/acrobat_download/literature/9398/39340011.pdf

[3] T. Mukherji, Aircraft Autopilot Design, ME 125 System Dynamics, Duke University. December, 2004
www.duke.edu/~tkm8/**Aircraft**%20**Autopilot**%20**Design**.doc

[4] Simple Aerodynamics Of The V-Tail
http://www.embedded.com/columns/technicalinsights/175801127

[5] The basics of control system design: Part 2 - Tuning a Proportional Controller
http://www.embedded.com/design/207402278

[6] C. Hernandez-Rosales, R. Femat-Flores; G. Quiroz-Compean, Make a PI controller on an 8-bit micro
http://www.embedded.com/columns/technicalinsights/175801127

[7] Using the I2C Bus
http://www.robot-electronics.co.uk/htm/using_the_i2c_bus.htm

[8] Electric RC Airplanes Advisor
http://www.rc-airplane-advisor.com/electric-rc-airplanes.html

[9] Aircraft configurations
http://selair.selkirk.bc.ca/ProfessionalPilot/aerodynamics/controls/configurations.htm

[10] Zigbee Pro Module
http://www.digi.com/products/wireless/point-multipoint/xbee-pro-series1-module.jsp

[11] US Navy Predator UAV Illustration
https://wrc.navair-rdte.navy.mil/warfighter_enc/aircraft/UAVs/images/pred2.jpg

[12] Picture of RC Predator model airplane
http://www.nitroplanes.com/4eluavrqprra.html

[13] Parts of the Aircraft
https://s.hostingprod.com/@www.toysonics.com/ssl/catalog/product_info.php?products_id=897&osCsid=2ef01ae73930d2465c3a6e05ac4c4a

APPENDIX A: *Data Sheets*

PIC Microcontroller
http://ww1.microchip.com/downloads/en/DeviceDoc/30292c.pdf

XBee-PRO™ ZigBee OEM Module
http://www.maxstream.net/products/xbee/xbee-pro-oem-rf-module-zigbee.php

Honeywell HMC6352 2 - Axis Digital Compass
http://www.sparkfun.com/datasheets/Components/HMC6352.pdf

VTI Technologies SCP1000 Pressure Sensor
http://www.vti.fi/en/products-solutions/products/pressure-sensors/scp1000-pressure-sensor/

VTI Technologies SCA3000 3 - Axis Accelerometer
http://www.vti.fi/en/products-solutions/products/accelerometers/sca3000-accelerometers/

2-Wire Serial 8-Bit DACs with Rail-to-Rail Outputs
http://datasheets.maxim-ic.com/en/ds/MAX517-MAX519.pdf

APPENDIX B: *Kits*

RC Predator Model Kit
http://www.nitroplanes.com/4eluavrqprra.html

APPENDIX C: *PIC Code – avionics, ground station –voltage reader, PI controller, flight plan code*

```
/*tristan lawson, paul agyiri.  e90 2008, avionics computer */
//data every 0.5 seconds'
// counter is for usability in recognizing when a manoeuvre takes place
// 04/11/2008

#include <16F873A.H>
#device ICD=TRUE
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, parity=N, bits=8)
//#use rs232(DEBUGGER)
#use i2c(Master, sda=PIN_C4, scl=PIN_C3, FORCE_HW)

// define registers/actions for registers on various sensors
#define accelslaveregread 0xF3
#define accelslaveregwrite1 0xF2
#define accelslaveregwrite2 0xF1
#define accelreadselect 0x17
#define xlowreg 0x04
#define xhighreg 0x05
#define ylowreg 0x06
#define yhighreg 0x07
#define zlowreg 0x08
#define zhighreg 0x09
#define nack 0x00
#define spcslaveregread 0x23
#define spcslaveregwrite 0x22
#define pressurerega 0x7F
#define pressureregb 0x80
#define temperaturereg 0x81
#define init 0x07
#define highres 0x0A
#define operation 0x03
#define ack 0x01
#define compassslavewrite 0x42
#define compassslaveread 0x43
#define compassdatareg 0x41

signed int32 xaxis, yaxis;
int32 heading, pressure, temperature, counter;
int32 number = 0;
```

```
int32 read_count(void)
 {
 number = number + 1;

 return number;
 }


 //------READ ALL THE VALUES FROM ACCELEROMETER RING BUFFER---------

  signed int32 read_accelerometer1(void)
   {
   signed int32 x_low, x_high ;

      i2c_start();
       i2c_write(accelslaveregwrite1);
       delay_ms(10);
       i2c_write(accelslaveregwrite2);
       delay_ms(10);
       i2c_write(accelreadselect);
       delay_ms(10);
       i2c_write(xhighreg);

       delay_ms(10);

       i2c_start();
       i2c_write(accelslaveregread);
       delay_ms(10);
       x_high = i2c_read(1);
        delay_ms(10);
       x_low = i2c_read(0);
       delay_ms(10);
      i2c_stop();

//-------------------------------------format x axis = roll angles-----------------------------------
// changes accelerometer output to number from -90 to 90 to indicate roll

      x_high = x_high % 255;
      if(x_high > 224)
      {
      x_high = x_high - 255;
      }
      x_high = x_high*3;
//---------------------------------------------------end-----------------------------------------------
```

```
  return(x_high);
}


//------------------------------------------------------------------------------------------------------
  signed int32 read_accelerometer2(void)
   {
   signed int32 y_low, y_high;


      i2c_start();
       i2c_write(accelslaveregwrite1);
       delay_ms(10);
       i2c_write(accelslaveregwrite2);
       delay_ms(10);
       i2c_write(accelreadselect);
       delay_ms(10);
       i2c_write(yhighreg);

       delay_ms(10);

       i2c_start();
       i2c_write(accelslaveregread);
       delay_ms(10);
       y_high = i2c_read(1);
        delay_ms(10);
       y_low = i2c_read(0);
        delay_ms(10);
      i2c_stop();

//---------------------------------format y axis = pitch angles-------------------------------------
// changes accelerometer output to number from -90 to 90 to indicate pitch

      y_high = y_high % 255;
      if(y_high > 224)
      {
      y_high = y_high - 255;
      }
      y_high = y_high*3;
//--------------------------------------------------end-----------------------------------------------

  return(y_high);
}
```

```
//---------------------------------------------compass--------------------------------------------

// Read the compass heading.
int32 read_heading(void)
  {
    int32 compasslsb,compassmsb;

    //start sequence and send slave adress byte
    i2c_start();
    i2c_write(compassslavewrite); // send slave address with write data
    i2c_write(compassdatareg); // data register
    i2c_stop();

    delay_ms(10); // actual delay is 6

    //inititate sensor read
    i2c_start();
    i2c_write(compassslaveread); // slave address with read data
    compassmsb = i2c_read();
    compasslsb = i2c_read(nack);
    i2c_stop();

    return((int32)compasslsb | ((int32)compassmsb << 8))/10;
  }

//---------------------------------------------end compass------------------------------------------


//--------------------------------------------pressure sensor-----------------------------------------

initialize_pressure_sensor(void)
{
    i2c_start();
    i2c_write(spcslaveregwrite);
    delay_ms(25);
    i2c_write(operation);
    delay_ms(50);
    i2c_write(highres);
    delay_ms(25);
    i2c_stop();
    delay_ms(50);
}


int32 read_temperature(void)
{
```

```
    int32 tempbits1, tempbits2;
    output_low(pin_C0); // connects to digital gnd or vdd on device
    output_high(pin_C1); // connects to digital gnd or vdd on device

     //GET temp DATA

    //start sequence and send slave adress byte
    i2c_start();
    i2c_write(spcslaveregwrite); // send slave address with write command
    delay_ms(25); // fake receiving the 'ACK' aka check for a '0' on SDA
    i2c_write(temperaturereg); // send data register address
    delay_ms(25); // fake receiving the 'ACK' aka check for a '0' on SDA

    //inititate sensor read
    i2c_start();
    i2c_write(spcslaveregread); // slave address with read command
    tempbits1 = i2c_read(); // read msb2
   // one clock cycle fake 'ACK' goes here, aka check for a '0' on SDA
    tempbits2 = i2c_read(nack); //  read lsb with 'NACK' condition
    i2c_stop();

return ((int32)tempbits2 | ((int32)tempbits1 << 8)) /20;

}

// Read the pressure sensor (19 bits total).
int32 read_pressure(void)
  {
    int32 pressurelsb,pressuremsb,pressuremsb2;
    output_low(pin_C0);
    output_high(pin_C1);

     //GET MSB DATA

    //start sequence and send slave adress byte
    i2c_start();
    i2c_write(spcslaveregwrite); // send slave address with write command
    delay_ms(25); // fake receiving the 'ACK' aka check for a '0' on SDA
    i2c_write(pressurerega); // send data register address
    delay_ms(25); // fake receiving the 'ACK' aka check for a '0' on SDA

    //inititate sensor read
    i2c_start();
    i2c_write(spcslaveregread); // slave address with read command
    pressuremsb2 = i2c_read(nack); // read msb2
    i2c_stop();
```

```
    // GET LSB DATA

     //start sequence and send slave adress byte
     i2c_start();
     i2c_write(spcslaveregwrite); // send slave address with write command
     delay_ms(25); // fake receiving the 'ACK' aka check for a '0' on SDA
     i2c_write(pressureregb); // send data register address
     delay_ms(25); // fake receiving the 'ACK' aka check for a '0' on SDA

     //inititate sensor read
     i2c_start();
     i2c_write(spcslaveregread); // slave address with read command
     pressuremsb = i2c_read(); // read msb
     pressurelsb = i2c_read(nack); //  read lsb with 'NACK' condition
     i2c_stop();

     return((int32)pressurelsb | ((int32)pressuremsb << 8) | ((int32)pressuremsb2 << 16))
/4;
   }

//--------------------------------end pressure/temperature sensor----------------------------------


//-------------------------------------MAIN FUNCTION------------------------------------------

void main()
{
  while(1)
      {
        output_low(pin_C0);
        output_high(pin_C1);

    //init functions to read sensors
        counter = read_count();
        xaxis = read_accelerometer1();
        yaxis = read_accelerometer2();
        heading = read_heading();
        initialize_pressure_sensor();
        temperature = read_temperature();
        pressure = read_pressure();

     //print all data to serial in 'column' format - easy for collection in matlab
      printf("%lu\t%ld\t%ld\t%lu\t%lu\t%lu\n\r",counter , xaxis, yaxis, heading, pressure,
temperature);
        delay_ms(25);
```

```
    }
}

//---------------------------------------END OF PROGRAM-----------------------------------------
```

```
/*tristan lawson, paul agyiri.  e90 2008, straight and level flight */
// fuses defs--> high speed oscillator, no watchdog timer, no code protect, no low voltage
programming
// use delay must precede use rs232 directive
// flight controller
// 04/14/2008

#include <16F873A.H>
#device ICD=TRUE
#fuses XT, NOWDT, NOPROTECT, NOLVP, DEBUG
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, parity=N, bits=8)
#use rs232(DEBUGGER)
#use i2c(Master, sda=PIN_C4, scl=PIN_C3, FORCE_HW)

#define dacconfig 0x00    // 00000000  this also represents the default settings
#define dacaddress 0x58   // 01011xx0 where xx = 00 since we will tie AD0 and Ad1 to
ground

signed int32 altitude, direction, aileron_pi, elevator_pi, pi_aileron_out, pi_elevator_out;
signed int32 pi_altitude_decision, pi_direction_decision
,altitude1,direction1,aileron_voltage, elevator_voltage;
float Kp_aileron; // insert values after system id
float Ki_aileron; // insert values after system id
float Kp_elevator; // insert values after system id
float Ki_elevator; // insert values after system id
float altitude_setpoint; // insert value we want
float direction_setpoint; // insert value we want
float Tp_aileron, Ti_aileron, Tp_elevator, Ti_elevator, temp_aileron_float,
temp_elevator_float; //proportional and integral term
float elevator_error_sum = 0, aileron_error_sum = 0, altitude_error, direction_error; //
sum of errors, for integral term

/*

read_altitude()
  {
    signed int sensor_altitude;


    return(sensor_altitude);
  }



read_heading()
  {
```

```
  signed int sensor_heading;



  return(sensor_heading);
  }

*/

//------------------------------PITCH/ALTITUDE CONTROLLER-----------------------------

pi_controller_pitch(signed int32 altitude1)
  {
    altitude = altitude1;

    // calculate the raw error
    altitude_error = altitude_setpoint - altitude;

    // calculate the proportional term
    Tp_elevator = -Kp_elevator * altitude_error;

    // calculate the integral term
    elevator_error_sum = elevator_error_sum + altitude_error;
    temp_elevator_float = elevator_error_sum;
    Ti_elevator = Ki_elevator * temp_elevator_float;

     // calculate the desired power
    elevator_voltage = Tp_elevator + Ti_elevator;

    // set the correct power
    if (elevator_voltage < 0)
    elevator_voltage = 0;
    else if (elevator_voltage > 255)
    elevator_voltage = 255;
    else
    pi_elevator_out = elevator_voltage;
    return(pi_elevator_out); // this could be pwm duty, etc
}

elevator_dac(signed int32 elevator_value)
{
    elevator_value = elevator_value;

    i2c_start();
    i2c_write(dacaddress); // send dac/slave address with write command
    delay_us(1); // fake receiving the 'ACK'
```

```
    i2c_write(dacconfig); // send dac config data
    delay_us(1); // fake receiving the 'ACK'
    i2c_write(elevator_value); // send value to dac for d/a conversion
    i2c_stop();
}

//------------------------------END PITCH/ALTITUDE CONTROLLER------------------------



//--------------------------------ROLL/DIRECTION CONTROLLER---------------------------

signed int32 pi_controller_roll(direction1)
  {

    // calculate the raw error
    direction_error = direction_setpoint - direction;

    // calculate the proportional term
    Tp_aileron = -Kp_aileron * direction_error;

    // calculate the integral term
    aileron_error_sum = aileron_error_sum + direction_error;
    temp_aileron_float = aileron_error_sum;
    Ti_aileron = Ki_aileron * temp_aileron_float;

     // calculate the desired power
    aileron_voltage = Tp_aileron + Ti_aileron;

    // set the correct power
    if (aileron_voltage < 0)
    aileron_voltage = 0;
    else if (aileron_voltage > 255)
    aileron_voltage = 255;
    else
    pi_aileron_out = aileron_voltage;
    return(pi_aileron_out); // this is for DAC
}


aileron_dac(signed int32 aileron_value)
{
    aileron_value = aileron_value;

    i2c_start();
    i2c_write(dacaddress); // send dac/slave address with write command
```

```
      delay_us(1); // fake receiving the 'ACK'
      i2c_write(dacconfig); // send dac config data
      delay_us(1); // fake receiving the 'ACK' A
      i2c_write(aileron_value); // send value to dac for d/a conversion
      i2c_stop();
}

//--------------------------END ROLL/DIRECTION CONTROLLER--------------------------



//-------------------------------------MAIN PROGRAM------------------------------------------

void main()
{
   while(1)
       {
         //altitude = read_altitude();
        // direction = read_heading();
         pi_altitude_decision = pi_controller_pitch(altitude);
         elevator_dac(elevator_pi);
         pi_direction_decision = pi_controller_roll(direction);
         aileron_dac(aileron_pi);
         delay_ms(100);
       }
}

//--------------------------------------END OF PROGRAM-------------------------------------
```

```
/*tristan lawson, paul agyiri.  e90 2008, ground code */
// detects the changes in voltage coming from transmitter potentiometers corresponding to
// changes in control stick position. Required to develop controller.
// Adapted from Erik Cheever's E72 class assignments
// 04/18/2008

#include <16F873A.H>
#device ICD=TRUE
#device ADC=8 // 8 bit resolution is what we want
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)
//#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, parity=N, bits=8)
#use rs232(DEBUGGER)


signed int32 read_elevator_tx, read_aileron_tx, elevator_tx_value, aileron_tx_value,
number = 0, counter;
signed int32 read_rudder_tx, read_throttle_tx, rudder_tx_value, throttle_tx_value;

signed int32 read_count(void)
 {
  number = number + 1;
  return number;
 }

signed int32 read_elevator_function(void)
 {
   //Set up adc port to read from channel 2 (pin A2)
  set_adc_channel(2);
  delay_ms(1);
  read_elevator_tx = read_adc();

  return (read_elevator_tx);
 }


signed int32 read_aileron_function(void)
 {
    //Set up adc port to read from channel 1 (pin A1)
  set_adc_channel(1);
  delay_ms(1);
  read_aileron_tx = read_adc();

  return (read_aileron_tx);
 }
```

```
signed int32 read_rudder_function(void)
 {
    //Set up adc port to read from channel 0 (pin A0)
  set_adc_channel(0);
  delay_ms(1);
  read_rudder_tx = read_adc();

  return (read_rudder_tx);
 }

 signed int32 read_throttle_function(void)
 {
    //Set up adc port to read from channel 4 (pin A5)
  set_adc_channel(4);
  delay_ms(1);
  read_throttle_tx = read_adc();

  return (read_throttle_tx);
 }

void main()
  {
  setup_adc(ADC_CLOCK_INTERNAL);
  setup_adc_ports(ALL_ANALOG);
  setup_spi(FALSE);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);

  //Main loop
 while (1)
   {
     counter = read_count();
     elevator_tx_value = read_elevator_function();
     aileron_tx_value = read_aileron_function();
     rudder_tx_value = read_rudder_function();
     throttle_tx_value = read_throttle_function();
     printf("%ld\t%ld\t%ld\t%ld\t%ld\n\r",counter, elevator_tx_value, aileron_tx_value,
rudder_tx_value, throttle_tx_value);
   }
}
```

```
/*tristan lawson, paul agyiri.  e90 2008, ground code */
// code to read incoming serial data from airplane and parse
// NOTE: this code is untested and incomplete
// 05/02/2008

#include <16F873A.H>
#device ICD=TRUE
#include <stdlib.h>
#fuses XT, NOWDT, NOPROTECT, NOLVP, DEBUG
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, parity=N, bits=8,
stream=FLIGHT_DATA, ERRORS)

#define FALSE = 0
#define TRUE = !FALSE

// array 50 and i = 49
int full_line,i;
char value;
char Serial_Array[60];


#INT_RDA
void serial_isr()
 {

        i =0;
        #use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, parity=N, bits=8,
stream=FLIGHT_DATA, ERRORS)
        Serial_Array[i] = getc(FLIGHT_DATA);
        i++;

        if (Serial_Array[i-1] == '\r')
          {
                              full_line = TRUE;
                                   i = 0;
                             }

             delay_ms(1);
             #use rs232(DEBUGGER, stream=FLIGHT_DATA1)
             printf("%s\n\r",Serial_Array);

 }
```

```
void main()
{
  clear_interrupt(INT_RDA);
  enable_interrupts(int_rda);
  enable_interrupts(GLOBAL);

  while(1)
  {
  serial_isr();
  // parsing function goes here

  }

}
```

```
/*tristan lawson, paul agyiri.  e90 2008, avionics computer */
//MAX517 DAC CODE
// min 500 ms delay between update required - insert this delay into dac function
//in ground computer code
// to have multiple dacs, change dacaddress and corresponding voltage inputs into them
// we switched the adressing with the rudder and throttle DACs due to confusion.
// 04/27/2008

#include <16F873A.H>
#device ICD=TRUE
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)
//#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, parity=N, bits=8)
#use rs232(DEBUGGER)
#use i2c(Master, sda=PIN_C4, scl=PIN_C3, FORCE_HW)

#define dacconfig 0x00    // 00000000
#define elevatordacaddress 0x58   // 01011xx0 where xx = 00, we will tie AD0 and AD1
to Vss
#define ailerondacaddress 0x5E   // 01011xx0 where xx = 11, we will tie AD0 and AD1
to Vdd
#define rudderdacaddress 0x5C   // 01011010 where xx = 01, we will tie AD0 to Vss and
AD1 to Vss
#define throttledacaddress 0x5A   // 01011100 where xx = 10, we will tie AD0 and AD1
to Vdd


//-------------------------------------------elevator DAC------------------------------------------

int32 elevator_dac(int32 elevator_value)
{
    elevator_value = elevator_value;

    i2c_start();
    i2c_write(elevatordacaddress); // send dac/slave address with write command
    delay_ms(1); // fake receiving the 'ACK'
    i2c_write(dacconfig); // send dac config data
    delay_ms(1); // fake receiving the 'ACK' A
    i2c_write(elevator_value); // send value to dac for d/a conversion
    i2c_stop();
    delay_ms(100);
}

//-------------------------------------------end elevator DAC-------------------------------------
-
```

```
//---------------------------------------------aileron DAC------------------------------------------

int32 aileron_dac(int32 aileron_value)
{
    aileron_value = aileron_value;

    i2c_start();
    i2c_write(ailerondacaddress); // send dac/slave address with write command
    delay_ms(1); // fake receiving the 'ACK'
    i2c_write(dacconfig); // send dac config data
    delay_ms(1); // fake receiving the 'ACK' A
    i2c_write(aileron_value); // send value to dac for d/a conversion
    i2c_stop();
    delay_ms(100);
}

//-----------------------------------------end aileron DAC----------------------------------------



//---------------------------------------------rudder DAC------------------------------------------
int32 rudder_dac(int32 rudder_value)
{
    rudder_value = rudder_value;

    i2c_start();
    i2c_write(rudderdacaddress); // send dac/slave address with write command
    delay_ms(1); // fake receiving the 'ACK'
    i2c_write(dacconfig); // send dac config data
    delay_ms(1); // fake receiving the 'ACK' A
    i2c_write(rudder_value); // send value to dac for d/a conversion
    i2c_stop();
    delay_ms(100);
}

//--------------------------------------------end rudder DAC-------------------------------------


//---------------------------------------------throttle DAC-----------------------------------------

int32 throttle_dac(int32 throttle_value)
{
    throttle_value = throttle_value;
```

```
    i2c_start();
    i2c_write(throttledacaddress); // send dac/slave address with write command
    delay_ms(1); // fake receiving the 'ACK'
    i2c_write(dacconfig); // send dac config data
    delay_ms(1); // fake receiving the 'ACK' A
    i2c_write(throttle_value); // send value to dac for d/a conversion
    i2c_stop();
    delay_ms(100);
}

//----------------------------------------end throttle DAC -----------------------------------------


//----------------------------------------MAIN FUNCTION----------------------------------------

void main()


  while(1)
     {
        elevator_dac(70);
        delay_ms(1000);
        elevator_dac(215);
         delay_ms(1000);
        elevator_dac(70);
         delay_ms(1000);
        elevator_dac(210);
         delay_ms(1000);
        elevator_dac(140);
        delay_ms(1000);


        aileron_dac(70);
        delay_ms(1000);
        aileron_dac(215);
         delay_ms(1000);
        aileron_dac(70);
         delay_ms(1000);
        aileron_dac(210);
         delay_ms(1000);
        aileron_dac(140);
        delay_ms(1000);
```

```
        // be careful, the throttle servo may be backwards

        throttle_dac(100);
         delay_ms(500);
        throttle_dac(120);
         delay_ms(500);
        throttle_dac(160);
         delay_ms(500);
        throttle_dac(200);
         delay_ms(500);
        throttle_dac(150);
         delay_ms(500);
        throttle_dac(100);
         delay_ms(500);
        throttle_dac(100);
         delay_ms(500);
        throttle_dac(80);
         delay_ms(500);
        throttle_dac(80);
         delay_ms(500);
        throttle_dac(100);
         delay_ms(500);
        throttle_dac(130);
         delay_ms(500);



        //---

        rudder_dac(70);
        delay_ms(1000);
        rudder_dac(215);
         delay_ms(1000);
        rudder_dac(70);
         delay_ms(1000);
        rudder_dac(210);
         delay_ms(1000);
        rudder_dac(140);
        delay_ms(1000);
    }
}

//--------------------------------------END OF PROGRAM---------------------------------------
```

Appendix D: *Data Used in Designing Control System, Code used in Determining System Transfer Function*

Event	Time	Throttle	Elevator	Aileron	Rudder	Altitude	Attitude Nose Relative Horizon	Roll RT Wing Relative Horizon	Heading 0-360
	(Sec)	0 = 0%	-10 = Stick Forward (Dive)	-10 = Stick Left	-10 = Stick Left				
		10 = 100%	10 = Stick Back (Climb)	10 = Stick Right	10 = Stick Right	(ft)	(deg)	(deg)	(Deg)
	Time	Throttle	Elevator	Aileron	Rudder	Altitude	Attitude	Roll	Heading
Take-Off	0	0	0	0	0	6	0	0	0
	0.1	0	0	0	0	6	0	0	0
	0.2	5	0	0	0	6	0	0	0
	0.3	10	0	0	0	6	0	0	0
	0.4	10	0	0	0	6	0	0	0
	0.5	10	0	0	0	6	0	0	0
	0.6	10	0	0	0	6	0	0	0
	0.7	10	0	0	0	7	0	0	0
	0.8	10	0	0	0	7	0	0	0
	0.9	10	0	0	0	8	0	0	0
	1	10	0	0	0	8	0	0	0
	1.1	10	0	0	0	9	0	0	0
	1.2	10	0	0	0	9	5	0	0
	1.3	10	1	0	0	10	5	0	0
	1.4	10	1	0	0	10	5	0	0
	1.5	10	1	0	0	10	5	0	0
	1.6	10	1	0	0	11	5	0	0
	1.7	10	1	0	0	11	5	0	0
	1.8	10	2	0	0	12	5	0	0
	1.9	10	2	0	0	12	5	0	0
	2	10	2	0	0	13	5	0	0
	2.1	10	2	0	0	13	5	0	0
	2.2	10	2	0	0	14	5	0	0
	2.3	10	3	0	0	14	10	0	0
	2.4	10	3	0	0	15	10	0	0
	2.5	10	3	0	0	15	10	0	0
	2.6	10	3	0	0	15	10	0	0
	2.7	10	3	0	0	16	10	0	0
	2.8	10	3	0	0	16	10	0	0
	2.9	10	3	0	0	17	10	0	0
	3	10	3	0	0	17	10	0	0
	3.1	10	3	0	0	18	10	0	0
	3.2	10	3	0	0	18	10	0	0

3.3	10	3	0	0	19	10	0	0
3.4	10	3	0	0	19	10	0	0
3.5	10	3	0	0	20	10	0	0
3.6	10	3	0	0	20	10	0	0
3.7	10	3	0	0	20	10	0	0
3.8	10	3	0	0	21	10	0	0
3.9	10	3	0	0	21	10	0	0
4	10	3	0	0	22	10	0	0
4.1	10	3	0	0	22	10	0	0
4.2	10	3	0	0	23	10	0	0
4.3	10	3	0	0	23	10	0	0
4.4	10	3	0	0	24	10	0	0
4.5	10	3	0	0	24	10	0	0
4.6	10	3	0	0	25	10	0	0
4.7	10	3	0	0	25	10	0	0
4.8	10	3	0	0	26	10	0	0
4.9	10	3	0	0	26	10	0	0
5	10	3	0	0	26	10	0	0
5.1	10	2	0	0	27	10	0	0
5.2	10	2	0	0	27	10	0	0
5.3	10	2	0	0	28	10	0	0
5.4	10	2	0	0	28	10	0	0
5.5	10	2	0	0	29	10	0	0
5.6	10	2	0	0	29	10	0	0
5.7	10	2	0	0	30	15	0	0
5.8	10	2	0	0	30	15	0	0
5.9	10	2	0	0	31	15	0	0
6	10	2	0	0	31	15	0	0
6.1	10	2	0	0	31	15	0	0
6.2	10	2	0	0	32	20	0	0
6.3	10	2	0	0	32	20	0	0
6.4	10	2	0	0	33	20	5	0
6.5	10	2	0	0	33	20	5	0
6.6	10	2	0	0	34	20	5	0
6.7	10	2	0	0	34	20	5	0
6.8	10	2	0	0	35	20	5	0
6.9	10	2	0	0	35	20	5	0
7	10	2	0	0	36	20	5	0
7.1	10	3	0	0	36	20	10	0
7.2	10	3	0	0	36	20	10	0
7.3	10	3	0	0	37	20	10	0
7.4	10	3	0	0	37	20	10	0
7.5	10	3	0	0	38	20	10	0
7.6	10	3	0	0	38	20	10	0
7.7	10	3	0	0	39	20	10	0
7.8	10	3	0	0	39	30	10	0
7.9	10	3	0	0	40	30	20	0
8	10	3	2	0	40	30	20	0
8.1	10	2	2	0	41	30	15	0
8.2	10	2	2	0	41	30	10	0
8.3	10	2	2	0	42	30	10	0

8.4	10	2	-4	0	42	30	-5	0
8.5	10	2	-4	0	42	30	-5	0
8.6	10	2	-4	0	43	30	-5	0
8.7	10	2	-2	0	43	30	-5	0
8.8	10	2	-2	0	44	30	0	0
8.9	10	2	0	0	44	30	0	0
9	10	2	0	0	45	30	0	0
9.1	10	2	0	0	45	30	0	0
9.2	10	2	0	0	46	30	0	0
9.3	10	2	0	0	46	30	0	0
9.4	10	2	0	0	47	30	0	0
9.5	10	2	0	0	47	30	0	0
9.6	10	2	0	0	47	30	0	0
9.7	10	2	0	0	48	30	0	0
9.8	10	2	0	0	48	30	0	0
9.9	10	2	0	0	49	30	0	0
10	10	2	0	0	49	30	0	0
10.1	10	2	0	0	50	30	0	0
10.2	10	2	0	0	50	30	0	0
10.3	10	2	0	0	51	30	0	0
10.4	10	2	0	0	51	30	0	0
10.5	10	2	0	0	52	30	0	0
10.6	10	2	0	0	52	30	0	0
10.7	10	2	0	0	52	30	0	0
10.8	10	2	0	0	53	30	0	0
10.9	10	2	0	0	53	30	0	0
11	10	2	0	0	54	30	0	0
11.1	10	2	0	0	54	30	0	0
11.2	10	2	0	0	55	30	0	0
11.3	10	2	0	0	55	30	0	0
11.4	10	2	0	0	56	30	0	0
11.5	10	2	0	0	56	30	0	0
11.6	10	2	0	0	57	30	0	0
11.7	10	2	0	0	57	30	0	0
11.8	10	2	0	0	58	30	0	0
11.9	10	2	0	0	58	30	0	0
12	10	2	0	0	58	30	0	0
12.1	10	2	0	0	59	30	0	0
12.2	10	2	0	0	59	30	0	0
12.3	10	2	0	0	60	30	0	0
12.4	10	2	0	0	60	30	0	0
12.5	10	2	0	0	61	30	0	0
12.6	10	2	0	0	61	30	0	0
12.7	10	2	0	0	62	30	0	0
12.8	10	2	0	0	62	30	0	0
12.9	10	2	0	0	63	30	0	0
13	10	2	0	0	63	30	0	0
13.1	10	2	0	0	63	30	0	0
13.2	10	2	0	0	64	30	0	0
13.3	10	2	0	0	64	30	0	0
13.4	10	2	0	0	65	30	0	0

	13.5	10	2	0	0	65	30	0	0
	13.6	10	2	0	0	66	30	0	0
	13.7	10	2	0	0	66	30	0	0
	13.8	10	2	0	0	67	30	0	0
	13.9	10	2	0	0	67	30	0	0
	14	10	2	0	0	68	30	0	0
	14.1	10	2	0	0	68	30	0	0
	14.2	10	2	0	0	68	30	0	0
	14.3	10	2	0	0	69	30	0	0
	14.4	10	2	0	0	69	30	0	0
	14.5	10	2	0	0	70	30	0	0
	14.6	10	2	0	0	70	30	0	0
	14.7	10	2	0	0	71	30	0	0
	14.8	10	2	0	0	71	30	0	0
	14.9	10	2	0	0	72	30	0	0
	15	10	2	0	0	72	30	0	0
Return to Level	15.1	10	1	0		72	20	0	0
flight	15.2	10	1	0		72	20	0	0
	15.3	10	1	0		73	20	0	0
	15.4	10	1	0		73	20	0	0
	15.5	10	1	0		73	20	5	0
	15.6	10	1	0		73	20	5	0
	15.7	10	1	0		73	20	5	0
	15.8	10	1	0		74	20	5	0
	15.9	10	1	0		74	10	5	0
	16	10	1	0		74	10	10	0
	16.1	10	0	0		74	10	10	0
	16.2	10	0	0		74	10	10	0
	16.3	10	0	0		74	10	20	0
	16.4	10	0	0		74	10	20	0
	16.5	10	0	0		74	10	20	0
	16.6	10	0	0		74	10	20	0
	16.7	10	0	2		74	10	20	0
	16.8	10	0	2		74	5	15	0
	16.9	10	0	2		74	5	10	0
	17	10	0	2		74	5	10	0
	17.1	10	-1	-4		74	5	-5	0
	17.2	10	-1	-4		75	5	-5	0
	17.3	10	-1	-4		75	5	-5	0
	17.4	10	-1	-2		75	5	-5	0
	17.5	10	-1	-2		75	5	0	0
	17.6	10	-1	0		75	5	0	0
	17.7	10	-2	0		75	5	0	0
	17.8	10	-2	0		75	0	0	0
	17.9	10	-2	0		75	0	0	0
	18	10	-2	0		75	0	0	0
	18.1	10	-2	0		75	0	0	0
	18.2	10	-2	0		75	0	0	0
	18.3	10	-2	0		75	0	0	0
	18.4	10	-2	0		75	0	0	0

	18.5	10	-1	0	75	0	0	0
	18.6	10	-1	0	75	0	0	0
	18.7	10	-1	0	75	0	0	0
	18.8	10	-1	0	75	0	0	0
	18.9	10	-1	0	75	0	0	0
	19	10	-1	0	75	0	0	0
	19.1	10	-1	0	75	0	0	0
	19.2	10	-1	0	75	0	0	0
	19.3	10	0	0	75	0	0	0
	19.4	10	0	0	75	0	0	0
	19.5	10	0	0	75	0	0	0
	19.6	10	0	0	75	0	0	0
	19.7	7	0	0	75	0	0	0
	19.8	7	0	0	75	0	0	0
	19.9	7	0	0	75	0	0	0
	20	7	0	0	75	0	0	0
90 degree	20.1	7	0	2	74	0	30	0
Right Turn	20.2	7	0	3	72	0	40	0
	20.3	7	0	4	71	0	60	0
	20.4	7	0	5	71	0	70	0
	20.5	7	0	5	71	0	80	0
	20.6	7	0	4	71	0	80	0
	20.7	7	0	3	71	0	80	0
	20.8	7	0	3	71	0	80	0
	20.9	7	0	2	71	0	80	10
	21	7	1	0	71	0	80	20
	21.1	7	1	0	71	0	80	22
	21.2	7	2	0	71	0	80	24
	21.3	7	2	0	71	0	80	26
	21.4	7	2	0	71	0	80	28
	21.5	7	2	0	71	0	80	30
	21.6	7	2	0	71	0	80	32
	21.7	7	2	0	71	0	80	34
	21.8	7	2	0	71	0	80	36
	21.9	7	2	0	71	0	80	38
	22	7	2	0	71	0	80	40
	22.1	7	2	0	71	0	80	42
	22.2	7	2	0	71	0	80	44
	22.3	7	2	0	71	0	80	46
	22.4	7	2	0	71	0	80	48
	22.5	7	2	0	71	0	80	50
	22.6	7	2	0	71	0	80	52
	22.7	7	2	0	71	0	80	54
	22.8	7	2	0	71	0	80	56
	22.9	7	2	0	71	0	80	58
	23	7	2	0	71	0	80	60
	23.1	7	2	0	71	0	80	62
	23.2	7	2	0	71	0	80	64
	23.3	7	2	0	71	0	80	66
	23.4	7	2	0	71	0	80	68

	23.5	7	2	0	71	0	80	70
	23.6	7	2	0	71	0	80	72
	23.7	7	2	0	71	0	80	74
	23.8	7	2	0	71	0	80	76
	23.9	7	2	-2	71	0	80	78
	24	7	1	-4	71	0	40	80
	24.1	7	1	-6	71	0	30	82
	24.2	7	0	-1	71	0	20	84
	24.3	7	0	0	71	0	10	86
	24.4	7	0	0	71	0	5	88
	24.5	7	1	0	71	5	0	90
	24.6	7	1	0	74	5	0	90
	24.7	7	1	0	75	5	0	90
	24.8	7	1	0	75	5	0	90
	24.9	7	1	0	75	0	0	90
	25	7	1	0	75	0	0	90
Return to Level	25.1	7	0	0	75	0	0	90
flight	25.2	7	0	0	75	0	0	90
	25.3	7	0	0	75	0	0	90
	25.4	7	0	0	75	0	0	90
	25.5	7	0	0	75	5	0	90
	25.6	7	0	0	75	5	0	90
	25.7	7	0	0	80	5	0	90
	25.8	7	0	0	80	5	0	90
	25.9	7	0	0	80	0	0	90
	26	7	-2	0	80	0	0	90
	26.1	7	-2	0	80	0	0	90
	26.2	7	-2	0	80	0	0	90
	26.3	7	-2	0	80	-5	0	90
	26.4	7	-2	0	80	-5	0	90
	26.5	7	-2	0	80	-5	0	90
	26.6	7	-3	0	85	-5	0	90
	26.7	7	-3	0	85	-5	0	90
	26.8	7	-3	0	85	-5	0	90
	26.9	7	-3	0	85	-5	0	90
	27	7	-3	0	85	-5	0	90
	27.1	7	-3	0	85	-5	0	90
	27.2	7	-1	0	85	-5	0	90
	27.3	7	-1	0	85	-10	0	90
	27.4	7	-1	0	85	-10	0	90
	27.5	7	-1	0	85	-10	0	90
	27.6	7	-1	0	75	-10	0	90
	27.7	7	-1	0	75	0	0	90
	27.8	7	0	0	75	0	0	90
	27.9	7	0	0	75	0	0	90
	28	7	0	0	75	0	0	90
	28.1	7	0	0	75	0	0	90
	28.2	7	0	0	75	0	0	90
	28.3	7	0	0	75	0	0	90
	28.4	7	0	0	75	0	0	90

	28.5	7	0	0	75	0	0	90
	28.6	7	0	0	75	0	0	90
	28.7	7	0	0	75	0	0	90
	28.8	7	0	0	70	0	0	90
	28.9	7	0	0	70	0	0	90
	29	7	0	0	70	0	0	90
	29.1	7	0	0	70	0	0	90
	29.2	7	0	0	70	0	0	90
	29.3	7	0	0	70	0	0	90
	29.4	7	0	0	70	0	0	90
	29.5	7	0	0	70	0	0	90
	29.6	7	0	0	70	0	0	90
	29.7	7	0	0	70	0	0	90
	29.8	7	0	0	70	0	0	90
	29.9	7	0	0	70	0	0	90
90 degree	30	7	0	0	70	0	0	90
Right Turn	30.1	7	0	2	74	0	30	90
	30.2	7	0	3	72	0	40	90
	30.3	7	0	4	71	0	60	90
	30.4	7	0	5	71	0	70	90
	30.5	7	0	5	71	0	80	90
	30.6	7	0	4	71	0	80	90
	30.7	7	0	3	71	0	80	90
	30.8	7	0	3	71	0	80	90
	30.9	7	0	2	71	0	80	100
	31	7	1	0	71	0	80	120
	31.1	7	1	0	71	0	80	122
	31.2	7	2	0	71	0	80	124
	31.3	7	2	0	71	0	80	126
	31.4	7	2	0	71	0	80	128
	31.5	7	2	0	71	0	80	130
	31.6	7	2	0	71	0	80	132
	31.7	7	2	0	71	0	80	134
	31.8	7	2	0	71	0	80	136
	31.9	7	2	0	71	0	80	138
	32	7	2	0	71	0	80	140
	32.1	7	2	0	71	0	80	142
	32.2	7	2	0	71	0	80	144
	32.3	7	2	0	71	0	80	146
	32.4	7	2	0	71	0	80	148
	32.5	7	2	0	71	0	80	150
	32.6	7	2	0	71	0	80	152
	32.7	7	2	0	71	0	80	154
	32.8	7	2	0	71	0	80	156
	32.9	7	2	0	71	0	80	158
	33	7	2	0	71	0	80	160
	33.1	7	2	0	71	0	80	162
	33.2	7	2	0	71	0	80	164
	33.3	7	2	0	71	0	80	166
	33.4	7	2	0	71	0	80	168

	33.5	7	2	0	71	0	80	170
	33.6	7	2	0	71	0	80	172
	33.7	7	2	0	71	0	80	174
	33.8	7	2	0	71	0	80	176
	33.9	7	2	-2	71	0	80	178
	34	7	1	-4	71	0	40	180
	34.1	7	1	-6	71	0	30	180
	34.2	7	0	-1	71	0	20	180
	34.3	7	0	0	71	0	10	180
	34.4	7	0	0	71	0	5	180
	34.5	7	1	0	71	5	0	180
	34.6	7	1	0	74	5	0	180
	34.7	7	1	0	75	5	0	180
	34.8	7	1	0	75	5	0	180
	34.9	7	1	0	75	0	0	180
	35	7	1	0	75	0	0	180
Return to Level	35.1	7	0	0	75	0	0	180
flight	35.2	7	0	0	75	0	0	180
Downwind leg	35.3	7	0	0	75	0	0	180
	35.4	7	0	0	75	0	0	180
	35.5	7	-2	0	75	5	0	180
	35.6	7	-2	0	75	5	0	180
	35.7	7	-2	0	80	5	0	180
	35.8	7	-2	0	80	5	0	180
	35.9	7	-2	0	80	0	0	180
	36	7	-2	0	80	0	0	180
	36.1	7	-2	0	80	0	0	180
	36.2	7	-2	0	80	0	0	180
	36.3	7	-2	0	80	-5	0	180
	36.4	7	-2	0	80	-5	0	180
	36.5	7	-2	0	80	-5	0	180
	36.6	7	-3	0	85	-5	0	180
	36.7	7	-3	0	85	-5	0	180
	36.8	7	-3	0	85	-5	0	180
	36.9	7	-3	0	85	-5	0	180
	37	7	-3	0	85	-5	0	180
	37.1	7	-3	0	85	-5	0	180
	37.2	7	-1	0	85	-5	0	180
	37.3	7	-1	0	85	-10	0	180
	37.4	7	-1	0	85	-10	0	180
	37.5	7	-1	0	85	-10	0	180
	37.6	7	-1	0	75	-10	0	180
	37.7	7	-1	0	75	-10	0	180
	37.8	7	-1	0	75	-10	0	180
	37.9	7	-1	0	75	-10	0	180
	38	7	-1	0	75	-10	0	180
	38.1	7	-1	0	75	-5	0	180
	38.2	7	-1	0	75	-5	0	180
	38.3	7	-1	0	75	-5	0	180

38.4	7	-1	0	75	-5	0	180
38.5	7	-1	0	75	-5	0	180
38.6	7	-1	0	75	-5	0	180
38.7	7	-1	0	75	-5	0	180
38.8	7	-1	0	75	-5	0	180
38.9	7	-1	0	75	0	0	180
39	7	-1	0	75	0	0	180
39.1	7	-1	0	70	0	0	180
39.2	7	1	0	70	0	0	180
39.3	7	1	0	70	5	0	180
39.4	7	1	0	70	5	0	180
39.5	7	1	0	70	5	0	180
39.6	7	3	0	70	5	0	180
39.7	7	3	0	70	5	0	180
39.8	7	3	0	70	10	0	180
39.9	7	3	0	70	10	0	180
40	7	3	0	75	10	0	180
40.1	7	3	0	75	10	0	180
40.2	7	3	0	75	10	0	180
40.3	7	3	0	75	10	0	180
40.4	7	3	0	75	10	0	180
40.5	7	3	0	75	10	0	180
40.6	7	3	0	75	10	0	180
40.7	7	3	0	75	10	0	180
40.8	7	3	0	75	10	0	180
40.9	7	3	0	75	10	0	180
41	7	0	0	75	10	0	180
41.1	7	0	0	75	10	0	180
41.2	7	0	0	75	0	0	180
41.3	7	0	0	75	0	0	180
41.4	7	0	0	70	0	0	180
41.5	7	0	0	70	0	0	180
41.6	7	0	0	70	0	0	180
41.7	7	0	0	70	0	0	180
41.8	7	0	0	70	0	0	180
41.9	7	0	0	70	0	0	180
42	7	0	0	70	0	0	180
42.1	7	0	0	70	0	0	180
42.2	7	0	0	70	0	0	180
42.3	7	0	0	70	0	0	180
42.4	7	0	0	70	0	0	180
42.5	7	0	0	70	0	0	180
42.6	7	0	0	70	0	0	180
42.7	7	0	0	70	0	0	180
42.8	7	0	0	70	0	0	180
42.9	7	0	0	70	0	0	180
43	7	0	0	70	0	0	180
43.1	7	0	0	70	0	0	180
43.2	7	0	0	70	0	0	180
43.3	7	0	0	70	0	0	180
43.4	7	0	0	70	0	0	180

43.5	7	0	0	70	0	0	180
43.6	7	0	0	70	0	0	180
43.7	7	0	0	70	0	0	180
43.8	7	0	0	70	0	0	180
43.9	7	0	0	70	0	0	180
44	7	0	0	70	0	0	180
44.1	7	0	0	70	0	0	180
44.2	7	0	0	70	0	0	180
44.3	7	0	0	70	0	0	180
44.4	7	0	0	70	0	0	180
44.5	7	0	0	70	0	0	180
44.6	7	0	0	70	0	0	180
44.7	7	0	0	70	0	0	180
44.8	7	0	0	70	0	0	180
44.9	7	0	0	70	0	0	180
45	7	0	0	70	0	0	180
45.1	7	0	0	70	0	0	180
45.2	7	0	0	70	0	0	180
45.3	7	0	0	70	0	0	180
45.4	7	0	0	70	0	0	180
45.5	7	0	0	70	0	0	180
45.6	7	0	0	70	0	0	180
45.7	7	0	0	70	0	0	180
45.8	7	0	0	70	0	0	180
45.9	7	0	0	70	0	0	180
46	7	0	0	70	0	0	180
46.1	7	0	0	70	0	0	180
46.2	7	0	0	70	0	0	180
46.3	7	0	0	70	0	0	180
46.4	7	0	0	70	0	0	180
46.5	7	0	0	70	0	0	180
46.6	7	0	0	70	0	0	180
46.7	7	0	0	70	0	0	180
46.8	7	0	0	70	0	0	180
46.9	7	0	0	70	0	0	180
47	7	0	0	70	0	0	180
47.1	7	0	0	70	0	0	180
47.2	7	0	0	70	0	0	180
47.3	7	0	0	70	0	0	180
47.4	7	0	0	70	0	0	180
47.5	7	0	0	70	0	0	180
47.6	7	0	0	70	0	0	180
47.7	7	0	0	70	0	0	180
47.8	7	0	0	70	0	0	180
47.9	7	0	0	70	0	0	180
48	7	0	0	70	0	0	180
48.1	7	0	0	70	0	0	180
48.2	7	0	0	70	0	0	180
48.3	7	0	0	70	0	0	180
48.4	7	0	0	70	0	0	180
48.5	7	0	0	70	0	0	180

	48.6	7	0	0	70	0	0	180
	48.7	7	0	0	70	0	0	180
	48.8	7	0	0	70	0	0	180
	48.9	7	0	0	70	0	0	180
	49	7	0	0	70	0	0	180
	49.1	7	0	0	70	0	0	180
	49.2	7	0	0	70	0	0	180
	49.3	7	0	0	70	0	0	180
	49.4	7	0	0	70	0	0	180
	49.5	7	0	0	70	0	0	180
	49.6	7	0	0	70	0	0	180
	49.7	7	0	0	70	0	0	180
	49.8	7	0	0	70	0	0	180
	49.9	7	0	0	70	0	0	180
	50	7	0	0	70	0	0	180
90 degree	50.1	7	0	2	74	0	30	180
Right Turn	50.2	7	0	3	72	0	40	180
	50.3	7	0	4	71	0	60	180
	50.4	7	0	5	71	0	70	180
	50.5	7	0	5	71	0	80	180
	50.6	7	0	4	71	0	80	180
	50.7	7	0	3	71	0	80	180
	50.8	7	0	3	71	0	80	180
	50.9	7	0	2	71	0	80	190
	51	7	1	0	71	0	80	200
	51.1	7	1	0	71	0	80	202
	51.2	7	2	0	71	0	80	204
	51.3	7	2	0	71	0	80	206
	51.4	7	2	0	71	0	80	208
	51.5	7	2	0	71	0	80	210
	51.6	7	2	0	71	0	80	212
	51.7	7	2	0	71	0	80	214
	51.8	7	2	0	71	0	80	216
	51.9	7	2	0	71	0	80	218
	52	7	2	0	71	0	80	220
	52.1	7	2	0	71	0	80	222
	52.2	7	2	0	71	0	80	224
	52.3	7	2	0	71	0	80	226
	52.4	7	2	0	71	0	80	228
	52.5	7	2	0	71	0	80	230
	52.6	7	2	0	71	0	80	232
	52.7	7	2	0	71	0	80	234
	52.8	7	2	0	71	0	80	236
	52.9	7	2	0	71	0	80	238
	53	7	2	0	71	0	80	240
	53.1	7	2	0	71	0	80	242
	53.2	7	2	0	71	0	80	244
	53.3	7	2	0	71	0	80	246
	53.4	7	2	0	71	0	80	248
	53.5	7	2	0	71	0	80	250

	53.6	7	2	0	71	0	80	252
	53.7	7	2	0	71	0	80	254
	53.8	7	2	0	71	0	80	256
	53.9	7	2	-2	71	0	80	258
	54	7	1	-4	71	0	40	260
	54.1	7	1	-6	71	0	30	262
	54.2	7	0	-1	71	0	20	264
	54.3	7	0	0	71	0	10	266
	54.4	7	0	0	71	0	5	268
	54.5	7	1	0	71	5	0	270
	54.6	7	1	0	74	5	0	270
	54.7	7	1	0	75	5	0	270
	54.8	7	1	0	75	5	0	270
	54.9	7	1	0	75	0	0	270
	55	7	1	0	75	0	0	270
Return to Level	55.1	7	1	0	75	0	0	270
flight	55.2	7	0	0	75	0	0	270
	55.3	7	0	0	75	0	0	270
	55.4	7	0	0	75	0	0	270
	55.5	7	0	0	75	0	0	270
	55.6	7	-2	0	75	5	0	270
	55.7	7	-2	0	75	5	0	270
	55.8	7	-2	0	80	5	0	270
	55.9	7	-2	0	80	5	0	270
	56	7	-2	0	80	0	0	270
	56.1	7	-2	0	80	0	0	270
	56.2	7	-2	0	80	0	0	270
	56.3	7	-2	0	80	0	0	270
	56.4	7	-2	0	80	-5	0	270
	56.5	7	-2	0	80	-5	0	270
	56.6	7	-2	0	80	-5	0	270
	56.7	7	-3	0	85	-5	0	270
	56.8	7	-3	0	85	-5	0	270
	56.9	7	-3	0	85	-5	0	270
	57	7	-3	0	85	-5	0	270
	57.1	7	-3	0	85	-5	0	270
	57.2	7	-3	0	85	-5	0	270
	57.3	7	-1	0	85	-5	0	270
	57.4	7	-1	0	85	-10	0	270
	57.5	7	-1	0	85	-10	0	270
	57.6	7	-1	0	85	-10	0	270
	57.7	7	-1	0	75	-10	0	270
	57.8	7	-1	0	75	-10	0	270
	57.9	7	-1	0	75	-10	0	270
	58	7	-1	0	75	-10	0	270
	58.1	7	-1	0	75	-10	0	270
	58.2	7	-1	0	75	-5	0	270
	58.3	7	-1	0	75	-5	0	270
	58.4	7	-1	0	75	-5	0	270
	58.5	7	-1	0	75	-5	0	270

58.6	7	-1	0	75	-5	0	270
58.7	7	-1	0	75	-5	0	270
58.8	7	-1	0	75	-5	0	270
58.9	7	-1	0	75	-5	0	270
59	7	-1	0	75	0	0	270
59.1	7	-1	0	75	0	0	270
59.2	7	-1	0	70	0	0	270
59.3	7	1	0	70	0	0	270
59.4	7	1	0	70	5	0	270
59.5	7	1	0	70	5	0	270
59.6	7	1	0	70	5	0	270
59.7	7	3	0	70	5	0	270
59.8	7	3	0	70	5	0	270
59.9	7	3	0	70	10	0	270
60	7	3	0	70	10	0	270
60.1	7	3	0	75	10	0	270
60.2	7	3	0	75	10	0	270
60.3	7	3	0	75	10	0	270
60.4	7	3	0	75	10	0	270
60.5	7	3	0	75	10	0	270
60.6	7	3	0	75	10	0	270
60.7	7	3	0	75	10	0	270
60.8	7	3	0	75	10	0	270
60.9	7	3	0	75	10	0	270
61	7	3	0	75	10	0	270
61.1	7	0	0	75	10	0	270
61.2	7	0	0	75	10	0	270
61.3	7	0	0	75	0	0	270
61.4	7	0	0	75	0	0	270
61.5	7	0	0	70	0	0	270
61.6	7	0	0	70	0	0	270
61.7	7	0	0	70	0	0	270
61.8	7	0	0	70	0	0	270
61.9	7	0	0	70	0	0	270
62	7	0	0	70	0	0	270
62.1	7	0	0	70	0	0	270
62.2	7	0	0	70	0	0	270
62.3	7	0	0	70	0	0	270
62.4	7	0	0	70	0	0	270
62.5	7	0	0	70	0	0	270
62.6	7	0	0	70	0	0	270
62.7	7	0	0	70	0	0	270
62.8	7	0	0	70	0	0	270
62.9	7	0	0	70	0	0	270
63	7	0	0	70	0	0	270
63.1	7	0	0	70	0	0	270
63.2	7	0	0	70	0	0	270
63.3	7	0	0	70	0	0	270
63.4	7	0	0	70	0	0	270
63.5	7	0	0	70	0	0	270
63.6	7	0	0	70	0	0	270

63.7	7	0	0	70	0	0	270
63.8	7	0	0	70	0	0	270
63.9	7	0	0	70	0	0	270
64	7	0	0	70	0	0	270
64.1	7	0	0	70	0	0	270
64.2	7	0	0	70	0	0	270
64.3	7	0	0	70	0	0	270
64.4	7	0	0	70	0	0	270
64.5	7	0	0	70	0	0	270
64.6	7	0	0	70	0	0	270
64.7	7	0	0	70	0	0	270
64.8	7	0	0	70	0	0	270
64.9	7	0	0	70	0	0	270
65	7	0	0	70	0	0	270
65.1	7	0	0	70	0	0	270
65.2	7	0	0	70	0	0	270
65.3	7	0	0	70	0	0	270
65.4	7	0	0	70	0	0	270
65.5	7	0	0	70	0	0	270
65.6	7	0	0	70	0	0	270
65.7	7	0	0	70	0	0	270
65.8	7	0	0	70	0	0	270
65.9	7	0	0	70	0	0	270
66	7	0	0	70	0	0	270
66.1	7	0	0	70	0	0	270
66.2	7	0	0	70	0	0	270
66.3	7	0	0	70	0	0	270
66.4	7	0	0	70	0	0	270
66.5	7	0	0	70	0	0	270
66.6	7	0	0	70	0	0	270
66.7	7	0	0	70	0	0	270
66.8	7	0	0	70	0	0	270
66.9	7	0	0	70	0	0	270
67	7	0	0	70	0	0	270
67.1	7	0	0	70	0	0	270
67.2	7	0	0	70	0	0	270
67.3	7	0	0	70	0	0	270
67.4	7	0	0	70	0	0	270
67.5	7	0	0	70	0	0	270
67.6	7	0	0	70	0	0	270
67.7	7	0	0	70	0	0	270
67.8	7	0	0	70	0	0	270
67.9	7	0	0	70	0	0	270
68	7	0	0	70	0	0	270
68.1	7	0	0	70	0	0	270
68.2	7	0	0	70	0	0	270
68.3	7	0	0	70	0	0	270
68.4	7	0	0	70	0	0	270
68.5	7	0	0	70	0	0	270
68.6	7	0	0	70	0	0	270
68.7	7	0	0	70	0	0	270

	68.8	7	0	0	70	0	0	270
	68.9	7	0	0	70	0	0	270
	69	7	0	0	70	0	0	270
	69.1	7	0	0	70	0	0	270
	69.2	7	0	0	70	0	0	270
	69.3	7	0	0	70	0	0	270
	69.4	7	0	0	70	0	0	270
	69.5	7	0	0	70	0	0	270
	69.6	7	0	0	70	0	0	270
	69.7	7	0	0	70	0	0	270
	69.8	7	0	0	70	0	0	270
	69.9	7	0	0	70	0	0	270
	70	7	0	0	70	0	0	270
	70.1	7	0	0	70	0	0	270
90 degree	70.2	7	0	2	74	0	30	270
Right Turn	70.3	7	0	3	72	0	40	270
	70.4	7	0	4	71	0	60	270
	70.5	7	0	5	71	0	70	270
	70.6	7	0	5	71	0	80	270
	70.7	7	0	4	71	0	80	270
	70.8	7	0	3	71	0	80	270
	70.9	7	0	3	71	0	80	270
	71	7	0	2	71	0	80	280
	71.1	7	1	0	71	0	80	300
	71.2	7	1	0	71	0	80	302
	71.3	7	2	0	71	0	80	304
	71.4	7	2	0	71	0	80	306
	71.5	7	2	0	71	0	80	308
	71.6	7	2	0	71	0	80	310
	71.7	7	2	0	71	0	80	312
	71.8	7	2	0	71	0	80	314
	71.9	7	2	0	71	0	80	316
	72	7	2	0	71	0	80	318
	72.1	7	2	0	71	0	80	320
	72.2	7	2	0	71	0	80	322
	72.3	7	2	0	71	0	80	324
	72.4	7	2	0	71	0	80	326
	72.5	7	2	0	71	0	80	328
	72.6	7	2	0	71	0	80	330
	72.7	7	2	0	71	0	80	332
	72.8	7	2	0	71	0	80	334
	72.9	7	2	0	71	0	80	336
	73	7	2	0	71	0	80	338
	73.1	7	2	0	71	0	80	340
	73.2	7	2	0	71	0	80	342
	73.3	7	2	0	71	0	80	344
	73.4	7	2	0	71	0	80	346
	73.5	7	2	0	71	0	80	348
	73.6	7	2	0	71	0	80	350
	73.7	7	2	0	71	0	80	352

	73.8	7	2	0	71	0	80	354
	73.9	7	2	0	71	0	80	356
	74	7	2	-2	71	0	80	358
	74.1	7	1	-4	71	0	40	360
	74.2	7	1	-6	71	0	30	360
	74.3	7	0	-1	71	0	20	360
	74.4	7	0	0	71	0	10	360
	74.5	7	0	0	71	0	5	360
	74.6	7	1	0	71	5	0	360
	74.7	7	1	0	74	5	0	360
	74.8	7	1	0	75	5	0	360
	74.9	7	1	0	75	5	0	360
	75	7	1	0	75	0	0	360
	75.1	7	1	0	75	0	0	360
Return to Level	75.2	7	1	0	75	0	0	360
flight	75.3	7	0	0	75	0	0	360
Upwind leg	75.4	7	0	0	75	0	0	360
	75.5	7	0	0	75	0	0	360
	75.6	7	0	0	75	0	0	360
	75.7	7	-2	0	75	0	0	360
	75.8	7	-2	0	75	0	0	360
	75.9	7	-2	0	80	5	0	360
	76	7	-2	0	80	5	0	360
	76.1	7	-2	0	80	0	0	360
	76.2	7	-2	0	80	0	0	360
	76.3	7	-2	0	80	-5	0	360
	76.4	7	-2	0	80	-5	0	360
	76.5	7	-2	0	80	-5	0	360
	76.6	7	-2	0	80	-5	0	360
	76.7	7	-2	0	80	-5	0	360
	76.8	7	-3	0	85	-5	0	360
	76.9	7	-3	0	85	-5	0	360
	77	7	-3	0	85	-5	0	360
	77.1	7	-3	0	85	-5	0	360
	77.2	7	-3	0	85	-10	0	360
	77.3	7	-3	0	85	-10	0	360
	77.4	7	-1	0	85	-10	0	360
	77.5	7	-1	0	85	-10	0	360
	77.6	7	-1	0	85	-10	0	360
	77.7	7	-1	0	85	-10	0	360
	77.8	7	-1	0	75	-10	0	360
	77.9	7	-1	0	75	-10	0	360
	78	7	-1	0	75	-10	0	360
	78.1	7	-1	0	75	-10	0	360
	78.2	7	-1	0	75	-10	0	360
	78.3	7	-1	0	75	-10	0	360
	78.4	7	-1	0	75	-10	0	360
	78.5	7	-1	0	75	-10	0	360
	78.6	7	-1	0	75	-5	0	360

78.7	7	-1	0	75	-5	0	360
78.8	7	-1	0	75	-5	0	360
78.9	7	-1	0	75	-5	0	360
79	7	-1	0	75	-5	0	360
79.1	7	-1	0	75	0	0	360
79.2	7	-1	0	75	0	0	360
79.3	7	-1	0	70	0	0	360
79.4	7	1	0	70	0	0	360
79.5	7	1	0	70	5	0	360
79.6	7	1	0	70	5	0	360
79.7	7	1	0	70	5	0	360
79.8	7	3	0	70	5	0	360
79.9	7	3	0	70	5	0	360
80	7	3	0	70	10	0	360
80.1	7	3	0	70	10	0	360
80.2	7	3	0	75	10	0	360
80.3	7	3	0	75	10	0	360
80.4	7	3	0	75	10	0	360
80.5	7	3	0	75	10	0	360
80.6	7	3	0	75	10	0	360
80.7	7	3	0	75	10	0	360
80.8	7	3	0	75	10	0	360
80.9	7	3	0	75	10	0	360
81	7	3	0	75	0	0	360
81.1	7	3	0	75	0	0	360
81.2	7	0	0	75	0	0	360
81.3	7	0	0	75	0	0	360
81.4	7	0	0	75	0	0	360
81.5	7	0	0	75	0	0	360
81.6	7	0	0	70	0	0	360
81.7	7	0	0	70	0	0	360
81.8	7	0	0	70	0	0	360
81.9	7	0	0	70	0	0	360
82	7	0	0	70	0	0	360
82.1	7	0	0	70	0	0	360
82.2	7	0	0	70	0	0	360
82.3	7	0	0	70	0	0	360
82.4	7	0	0	70	0	0	360
82.5	7	0	0	70	0	0	360
82.6	7	0	0	70	0	0	360
82.7	7	0	0	70	0	0	360
82.8	7	0	0	70	0	0	360
82.9	7	0	0	70	0	0	360
83	7	0	0	70	0	0	360
83.1	7	0	0	70	0	0	360
83.2	7	0	0	70	0	0	360
83.3	7	0	0	70	0	0	360
83.4	7	0	0	70	0	0	360
83.5	7	0	0	70	0	0	360
83.6	7	0	0	70	0	0	360
83.7	7	0	0	70	0	0	360

83.8	7	0	0	70	0	0	360
83.9	7	0	0	70	0	0	360
84	7	0	0	70	0	0	360
84.1	7	0	0	70	0	0	360
84.2	7	0	0	70	0	0	360
84.3	7	0	0	70	0	0	360
84.4	7	0	0	70	0	0	360
84.5	7	0	0	70	0	0	360
84.6	7	0	0	70	0	0	360
84.7	7	0	0	70	0	0	360
84.8	7	0	0	70	0	0	360
84.9	7	0	0	70	0	0	360
85	7	0	0	70	0	0	360
85.1	7	0	0	70	0	0	360
85.2	7	0	0	70	0	0	360
85.3	7	0	0	70	0	0	360
85.4	7	0	0	70	0	0	360
85.5	7	0	0	70	0	0	360
85.6	7	0	0	70	0	0	360
85.7	7	0	0	70	0	0	360
85.8	7	0	0	70	0	0	360
85.9	7	0	0	70	0	0	360
86	7	0	0	70	0	0	360
86.1	7	0	0	70	0	0	360
86.2	7	0	0	70	0	0	360
86.3	7	0	0	70	0	0	360
86.4	7	0	0	70	0	0	360
86.5	7	0	0	70	0	0	360
86.6	7	0	0	70	0	0	360
86.7	7	0	0	70	0	0	360
86.8	7	0	0	70	0	0	360
86.9	7	0	0	70	0	0	360
87	7	0	0	70	0	0	360
87.1	7	0	0	70	0	0	360
87.2	7	0	0	70	0	0	360
87.3	7	0	0	70	0	0	360
87.4	7	0	0	70	0	0	360
87.5	7	0	0	70	0	0	360
87.6	7	0	0	70	0	0	360
87.7	7	0	0	70	0	0	360
87.8	7	0	0	70	0	0	360
87.9	7	0	0	70	0	0	360
88	7	0	0	70	0	0	360
88.1	7	0	0	70	0	0	360
88.2	7	0	0	70	0	0	360
88.3	7	0	0	70	0	0	360
88.4	7	0	0	70	0	0	360
88.5	7	0	0	70	0	0	360
88.6	7	0	0	70	0	0	360
88.7	7	0	0	70	0	0	360
88.8	7	0	0	70	0	0	360

88.9	7	0	0	70	0	0	360
89	7	0	0	70	0	0	360
89.1	7	0	0	70	0	0	360
89.2	7	0	0	70	0	0	360
89.3	7	0	0	70	0	0	360
89.4	7	0	0	70	0	0	360
89.5	7	0	0	70	0	0	360
89.6	7	0	0	70	0	0	360
89.7	7	0	0	70	0	0	360
89.8	7	0	0	70	0	0	360
89.9	7	0	0	70	0	0	360
90	7	0	0	70	0	0	360
90.1	7	0	0	70	0	0	360
90.2	7	0	0	70	0	0	360

```
% Code for Determining System Transfer Function

% wn = 0.5;
% z =0.1;
% step (gs)
data = xlsread (' data.xls');
t = data (:,1);

x= data (:,2);
y= data (:,3);
error = zeros (1001,1001);
% Hs = tf (1,[1 2*wn*z wn^2]);
% O = lsim (Hs, x,t);
% subplot (211);
% plot (x,y);
% title (' Original');
%
% subplot (212);
% plot (x,O);
% title (' Simulated');
% i = 1;
% ii = 1;
for wn = 0:1:100
   for z = 0:1:100
```

Item	Quantity	Cost
Model airplane kit	1	90.00
Lipo battery	1	30.00
Futaba radio system	1	180.00
Electronic speed controller	3	200.00
Brushless motor	3	200.00
Lipo battery charger	1	30.00
Extra wiring + servo extensions + push rods	-	58.00
Pressure sensor	3	150.00
Accelerometer	3	150.00
Compass module	2	100.00
1.5V AA batteries	4	10.00
Miscellaneous		32.00

```
      Hs = tf (1,[1 2*wn*z wn^2]);
      O = lsim (Hs, x,t);
      error (i,ii) = sum ((y - O).^2);
      ii = ii+1;
   end
   i= 1+1;

end
save ' Results.mat' error;
```

Total	1230.00

Appendix E: *Cost of Materials*