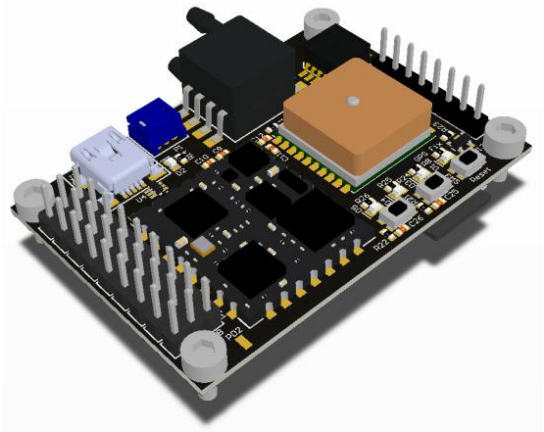
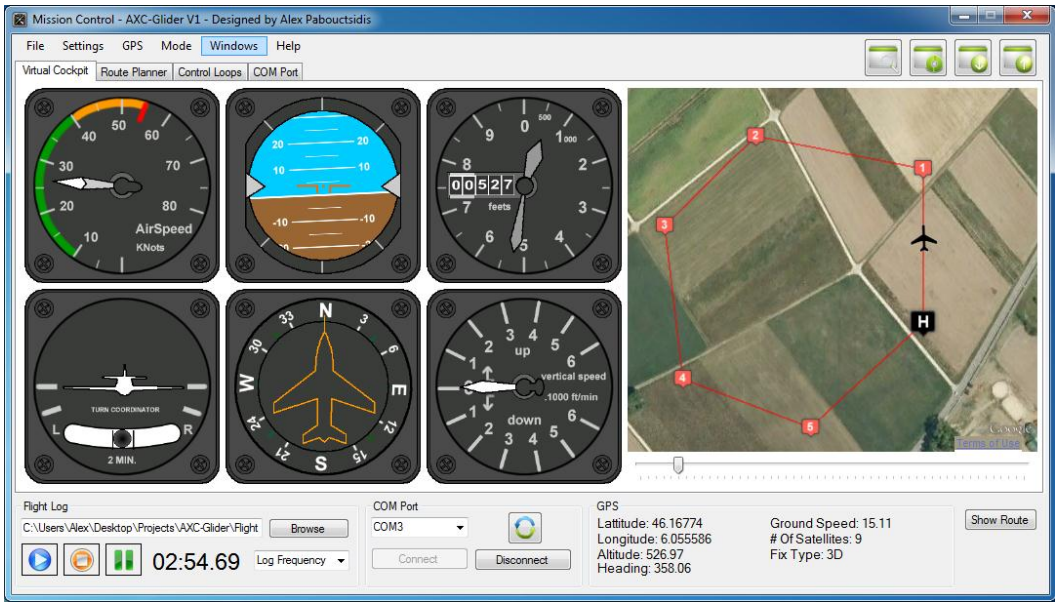
h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

AUTONOMOUS CROSS COUNTRY GLIDER: SYSTEM IDENTIFICATION AND CONTROL

Alexandre Pabouctsidis
Microengineering– Microsystems
hepia - HES·SO
June, 2011

Mentor: J.-M. Allenbach





1. TABLE OF CONTENTS

2. Introduction 4
3. Available hardware 7
3.1 AXCG system 7
3.2 AXCG autopilot board 8
3.2.1 Low level embedded software 10
3.3 Model aircrafts 11
4. Aircraft flight dynamics 13
4.1 Coordinate systems 13
4.2 Control inputs 14
4.3 Aerodynamic forces and moments 15
4.4 Stability and control derivatives 16
5. Theoretical aircraft model 18
5.1 Obtaining AXN floater wings geometry using 3d scans 19
5.2 XFLR5 aircraft model and analysis 21
5.3 AVL stability and control derivatives 25
5.4 Simulink model and simulation with flightgear 27
6. Control loops architecture 31
7. System identification and controller design 36
7.1 Ailerons → Roll identification 36
7.2 Elevator →Pitch identification 38
7.3 Attitude PID gains 39
7.3.1 Aileron PID Gains 39
7.3.1 Elevator PID Gains 40
7.4 Roll closed loop → Bearing 41
7.5 Pitch closed loop → Airspeed 44
7.6 Navigation PID gains 47
7.6.1 Roll PID gains 47
7.6.2 Pitch PID gains 50
7.7 Safety throttle curve 51
8. Embedded programming 53
8.1 Navigation functions 53
8.2 Control loop functions 55
8.1 High level tasks 55

8.1.1 Attitude Controller Task 56
8.1.2 Read and Parse Buffers Task 57
8.1.3 Navigate Task 57
8.1.4 Airspeed Task 59
8.1.5 Pressure Data Task 60
8.1.6 Record Data Task 61
8.1.7 Radio Timeout Task 62
8.1.8 Mission Control Task 63
8.1.9 GPS Initialize Task 64
8.1.10 Flight Mode Task 65
8.2 Processing time and runtime statistics 65
8.3 Important bugs/errors and fixes 66
8.3.1 Radio Parsing Error 66
8.3.2 AHRS centrifugal force sensitivity 67
9. PC User interface: Mission Control 68
9.1 AXCG ↔ PC communication protocol 68
9.1.1 Commands 68
9.1.2 Error Codes 68
9.1.3 Register Map 68
9.1.4 YPR Data 69
9.1.5 GPS Data 70
9.1.6 Pressure Voltage Data 70
9.1.7 Cycle Read Registers 71
9.1.8 Cycle Read Frequency 71
9.1.9 Mode 72
9.1.10 SD Log Frequency 72
9.1.11 Attitude PID Frequency 73
9.1.12 Aileron PID 73
9.1.13 Elevator PID 74
9.1.14 Navigation PID Frequency 74
9.1.15 Roll PID 75
9.1.16 Pitch PID 75
9.1.17 Servo Endpoints 76
9.1.18 Max Bank 76
9.1.19 Max Pitch 77

9.1.20 Waypoints 77
9.1.21 Cruise Speed 78
9.1.22 GPS Altitude Error 78
9.1.23 Throttle Altitude Threshold 79
9.1.24 Pitch Offset 79
9.1.25 Radio Timeout 80
9.2 Mission Control PC Interface 81
9.2.1 Google Earth implementation 81
9.2.2 Virtual Cockpit and Data Log playback 82
9.2.3 Route Planner 83
9.2.4 Uploading and Downloading Parameters 84
9.2.5 COM Port tab 85
10. Test Flights and Results 86
10.1 Fly by wire test 86
10.2 Autopilot mode 87
11. Next Step 95
11.1 Hardware Revision 95
11.1.1 Absolute Pressure Sensor 95
11.1.2 Low cost AHRS 95
11.1.3 Zigbee wireless module 96
11.2 Autopilot Functionality 96
11.3 Mission Control improvements 97
12. Conclusion 98
13. List of figures and tables 99
14. Digital appendix folders 101
15. References 103

2. INTRODUCTION

The autonomous cross country glider (AXCG) project is the design from scratch of an autopilot capable of autonomously flying a model glider using GPS waypoint navigation.

The project was inspired after watching an online video of a father and son launching a high altitude weather balloon with a HD camera attached – Space Balloon[1].

Figure 1: Space Balloon Freeze Frame

Once the balloon reached an altitude of about 30km, it burst and the camera dropped back down to earth with a parachute. The entire voyage was filmed and produced stunning footage.

The original idea of the AXCG project was to, like the space balloon video, launch a high altitude weather balloon but with an RC model glider instead of a parachuted camera. Once the balloon explodes due to the lack of atmospheric pressure, the glider would then fly a very long preprogrammed trajectory.

Due to the complexity of such a project, only the core essence of the project was retained: designing an autonomous glider capable of long range navigation, but without the weather balloon launch and high altitude navigation.

The final objective of the AXCG project was set as the following:

- A glider is launched from the top of a cliff or mountain with preprogrammed GPS waypoints
- As soon as the glider is launched, it follows the programmed trajectory as best as it can
- Once the glider reaches its final waypoint, it circles in the air waiting for the user to take command of the glider using a standard RC (i.e. Radio Controlled) Radio.
- Once the user has command of the glider, he or she safely lands the glider.

The project started earlier this school year as a semester project. During the semester project, the autopilot hardware and low level software functions were completed, leaving the implementation of the high level control and navigation functions, as well as the user PC interface to configure the flight and control parameters of the autopilot.

[1] http://www.vimeo.com/15091562

The bachelor project was successfully completed with an RC aircraft that was able to autonomously navigate around a preprogrammed circuit that was entered using a custom made user PC Interface. To achieve this, the following points were covered during the bachelor project:

- Study of the flight dynamics of a model aircraft
- Computation of theoretical aircraft stability and control derivatives coefficients for a Simulink model
- System identification and controller design using flight data logs
- Implementation of embedded control and navigation tasks on the autopilot
- Design of a user Interface using C# .NET framework and Google earth via HTML JavaScript

Figure 2 is picture of the autopilot mounted in the aircraft that was used throughout this project. The servo connectors of the aircraft are all directly connected to the autopilot, as well as the radio receiver. The rubber tubes are connected to the pitot tubes outside under the wing, and to the pressure sensor of the autopilot.

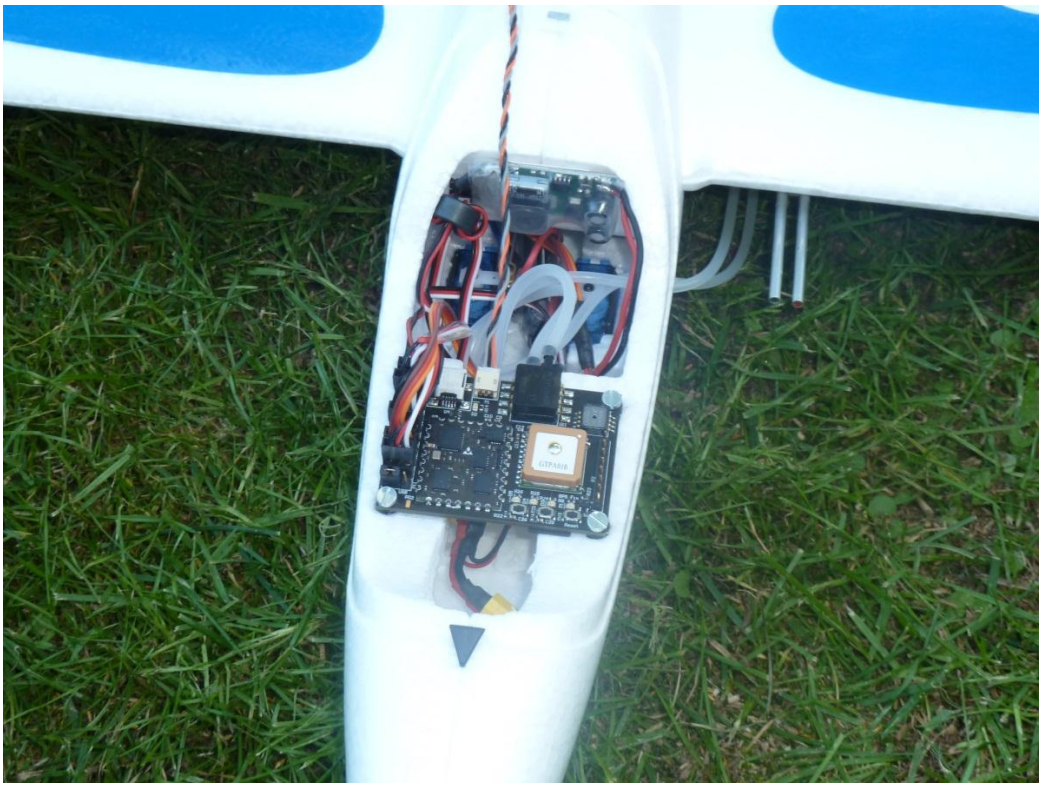


Figure 2: AXN Floater with AXCG system

The overall system is represented in Figure 3. The aircraft is mounted with and a radio receiver and the autopilot board which commands the aircraft controls. While in the air, the user sends commands to the autopilot using a radio transmitter. When the aircraft is on the ground, the autopilot board can be connected to a computer via USB, where

the user can then configure the autopilot using a custom program. In addition, the autopilot is capable of storing flight data on a micro SD card which can then be read on a computer.

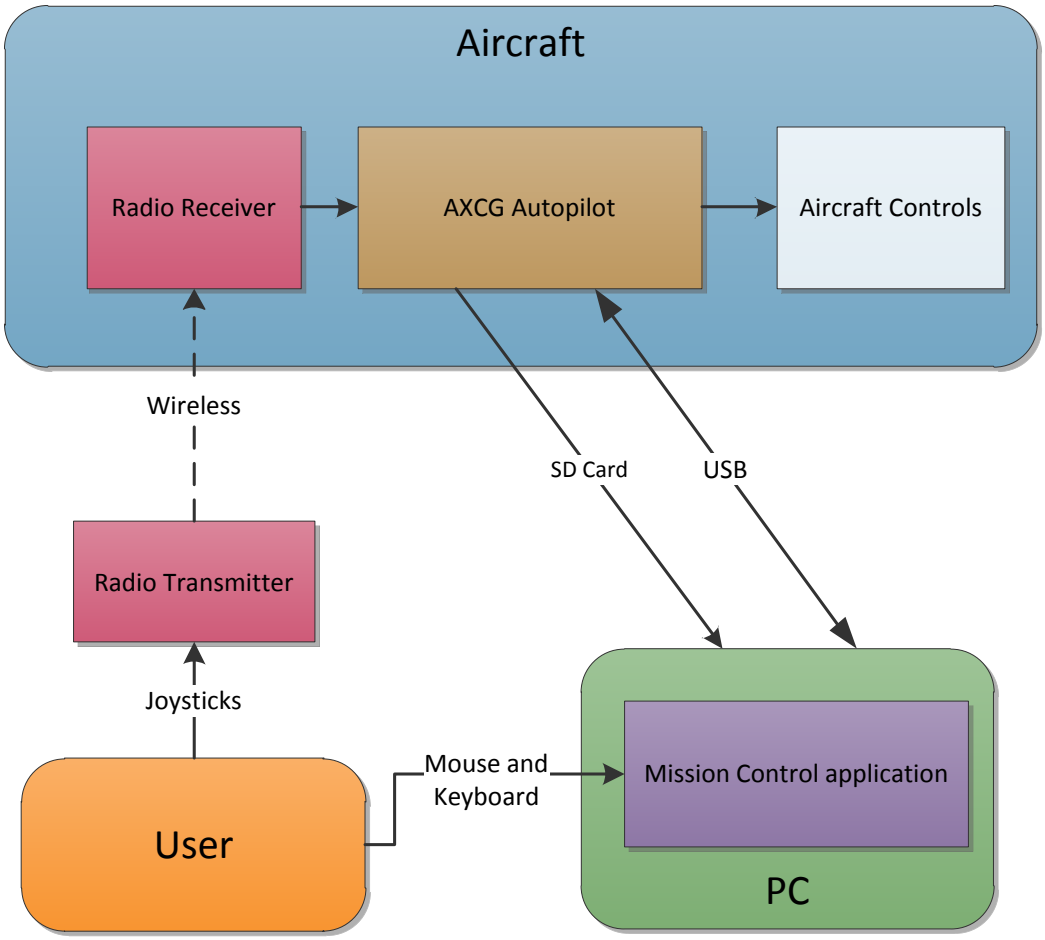


Figure 3: Global overview

3. AVAILABLE HARDWARE

As stated previously, the project started as a semester project and much was already completed during that time. The hardware and low level embedded programming was accomplished, creating a solid foundation on which to continue the bachelor project. This will be a small summary of what was accomplished, and what the starting point for the bachelor project was.

3.1 AXCG SYSTEM

The AXCG (autonomous cross country glider) autopilot system is composed of the following:

- A central autopilot PCB
- 3S lithium-polymer battery that powers all the electronics
- A speed controller with an internal BEC (Battery Eliminator Circuit) that powers the electric motor as well as provides a programmable 5-6V to the servos and the autopilot PCB
- A Spektrum satellite radio receiver
- Servos to control the aircraft control surfaces.

Figure 4: Spektrum Receiver

The Spektrum satellite radio receiver is bound via radio frequency to a radio transmitter that the operator uses to send commands to the autopilot board. Data from the receiver is sent to the autopilot using a standard 3.3V TTL serial protocol. The servos and throttle are controlled using PWM (Pulse Width Modulation) signals with varying pulse widths (1ms to 2ms).

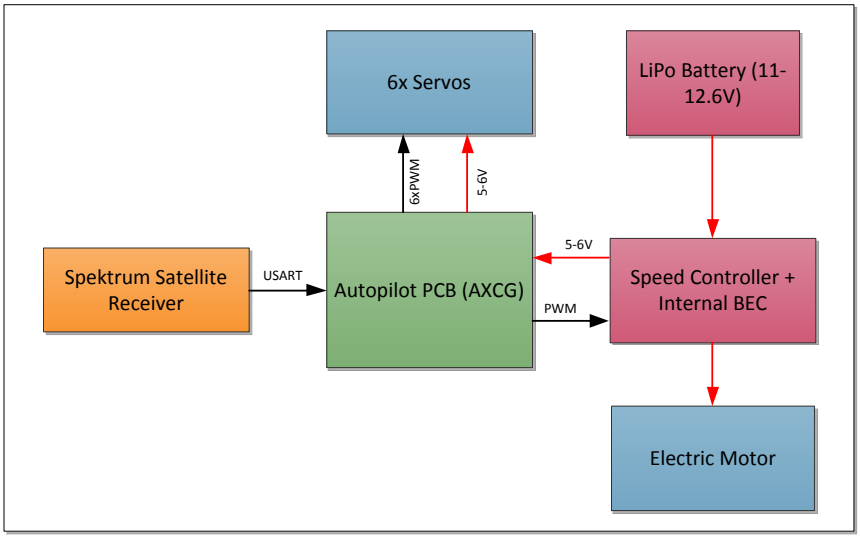


Figure 5: AXCG Autopilot System

3.2 AXCG AUTOPILOT BOARD

At the heart of the autopilot is a STM32 ARM Cortex-M3 Microcontroller running at 72 MHz. The PCB has 7 PWM outputs to command servos and/or speed controllers. The autopilot also has a special connector to directly connect to the Spektrum satellite receiver. It also possesses a micro SD memory card slot for in flight data logging, as well a small I²C EEPROM to store flight and configuration parameters. To control the autopilot on the field, the autopilot includes a few LEDs and buttons. A USB port is present on the board for communication with a PC.

The autopilot is fitted with the following sensors: Vectornav VN-100 AHRS, Mediatek 3329 10Hz GPS module, MPXV5004DP differential pressure sensor, and the MPXH6101A6T1 absolute pressure sensor.

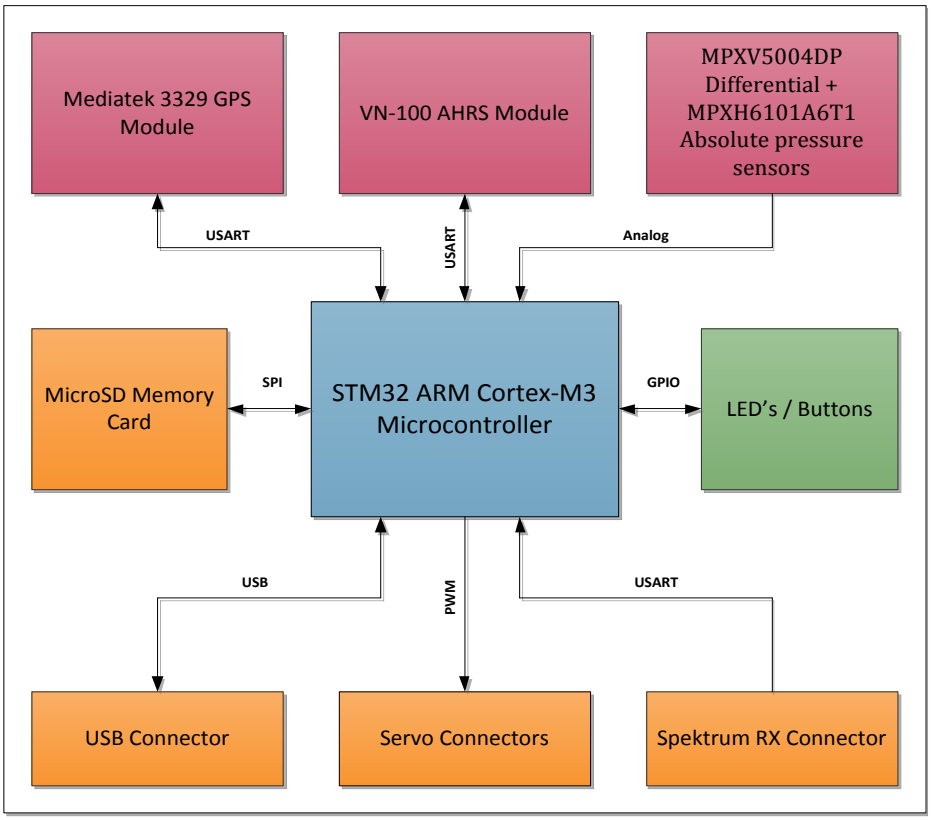


Figure 6: AXCG Autopilot PCB Diagram

The end result of the autopilot PCB was a compact double side mounted autopilot with sophisticated sensors, all of which providing accurate measurements required to control an aircraft.

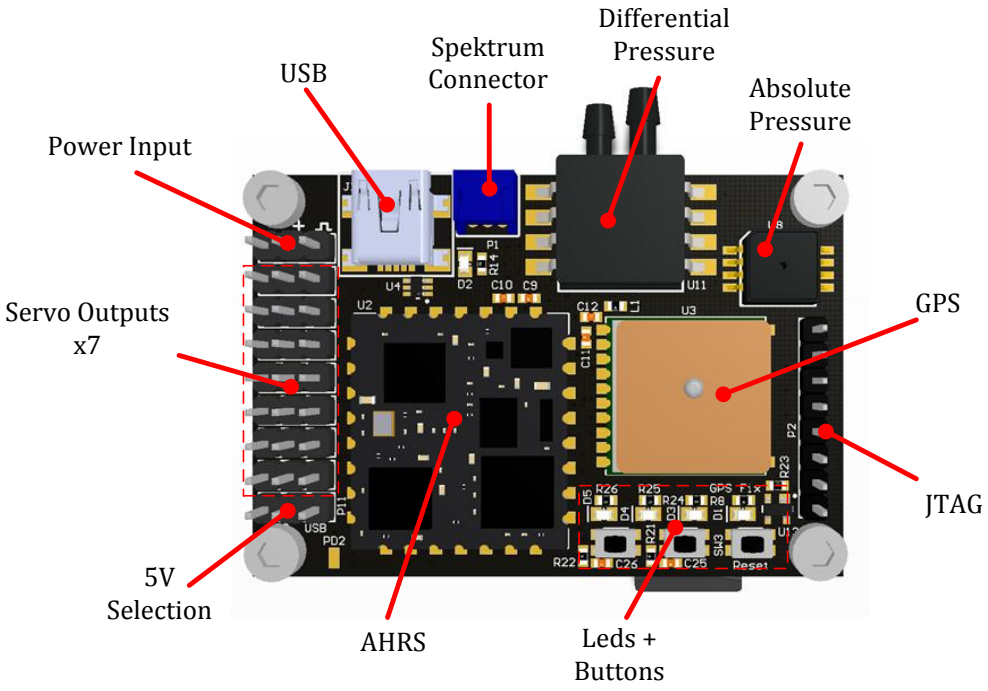


Figure 7: AXCG Autopilot PCB – TOP

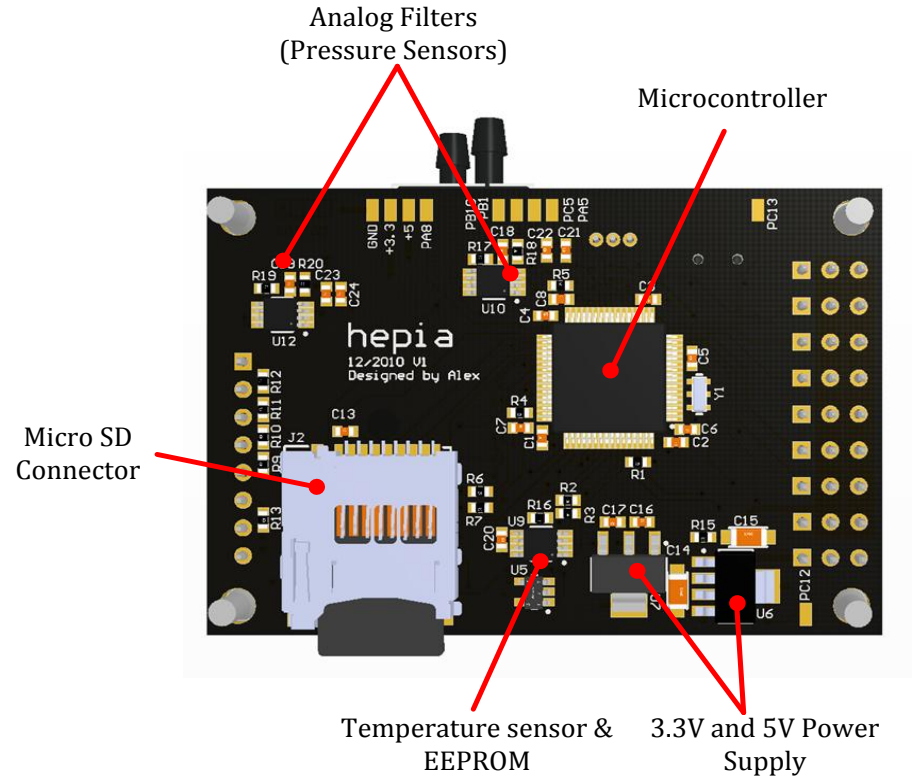


Figure 8: AXCG Autopilot PCB – Bottom

3.2.1 LOW LEVEL EMBEDDED SOFTWARE

Much of the low level functions to access and control the various sensors and components were implemented during the semester project. The code was written in a structured hierarchal format, starting with ST's peripheral library with which low level functions to access and control all aspects of the autopilot were written.

In addition, a FAT32 library was implemented to create, read, and write files on the micro SD card.

Finally, a free RTOS (Real Time Operating System) called FreeRTOS was implemented that allowed the quick implementation of high level routines and tasks. It also allowed creating of structured code making it more robust, and easier to debug.

This provided a very solid foundation for the project, making it easy to write structured and robust routines for all aspects of operation such as the control loops and the user PC interface.

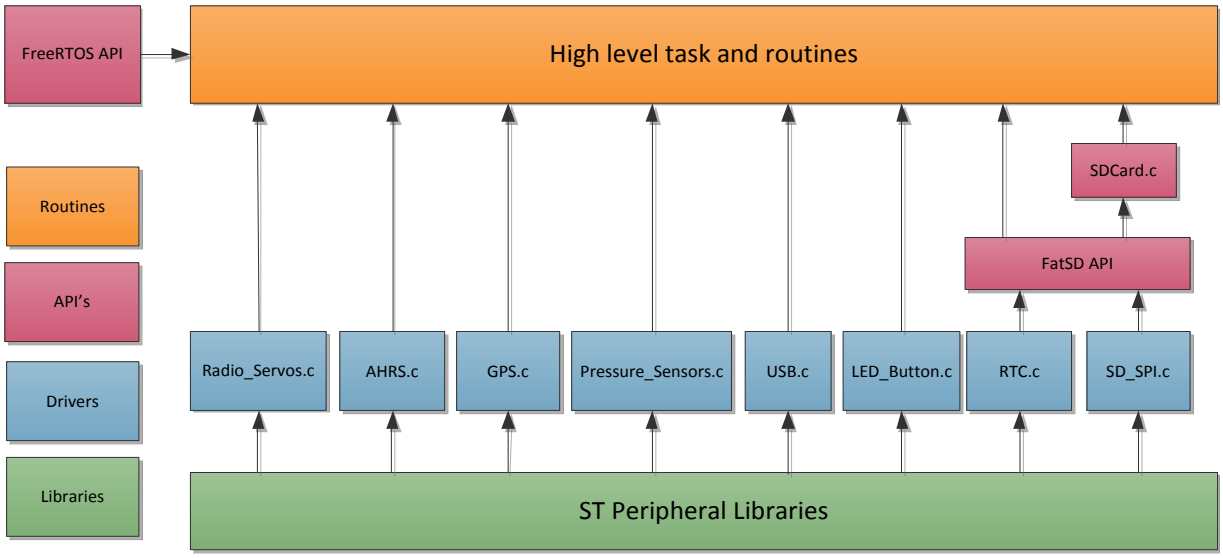


Figure 9: Embedded Software Architecture

3.3 MODEL AIRCRAFTS

Two standard RC aircrafts were selected and ordered during the semester project. The main aircraft is the Multiplex Cularis glider, with a 2.6m wingspan, an electric motor and folding propeller. It has 6 control surfaces: 2x Ailerons, 2x Flaps, an Elevator, and a Rudder, allowing precise control of the aircraft. The glider has a large wingspan and a low wing load which, theoretically, should give it a good glide ratio. The electric motor was used during field tests (eliminating the need of a secondary tow plane) as well as a safety thruster if the glider would drop too low during in autopilot mode.

Here are its specifications:

- Wingspan : 2.61 [m]
- Length : 1.26 [m]
- Flying Weight : ~1.7 [kg]
- Wing Loading : 30 [g/dm^2]
- Wing Area : 55 [dm^2]

Figure 10: Multiplex Cularis

Due to the large size of the Cularis, for logistic reasons, smaller and cheaper aircrafts were ordered that would be used to perform field tests to validate certain parts of the software before testing it on the much larger and more expensive Cularis. The secondary aircraft is the AXN Floater from hobbyking.com, which can be ordered for $50 with everything included except the battery and radio receiver. Its small size and low price made it a great choice for field tests.

The AXN is a 1.6m wingspan electric airplane, with ailerons, an elevator and rudder. The propeller is mounted on the back reducing the risk of a prop strike, making the aircraft more resilient to crashes,

After a crash of the original Cularis because of a radio parsing error (see 8.3.1) the entire project was completed using only the AXN Floater, as they were easier to replace. A total of three aircrafts were used after damaging two in various crashes.

Only after validating the entire code and methodology on the small AXN floater would the steps be repeated with the larger Cularis. Unfortunately, due to the short time frame, the Cularis never flew with the autopilot.

Figure 11: AXN Floater

4. AIRCRAFT FLIGHT DYNAMICS

4.1 COORDINATE SYSTEMS

An aircraft has six degrees of freedom, three degrees to define the position of the aircraft, and three to define the orientation in space of the aircraft, or attitude. Aerodynamic forces and movements are generally defined in one of two possible co-ordinate systems:

- Wind axes
  X axis – Positive in the direction of incoming air
  Y axis – Perpendicular to X axis
  Z axis – Positive downwards, and perpendicular to XY plane
- Inertial or body axes
  X axis – Positive forward through the nose of the aircraft
  Y axis – Perpendicular to X axis, positive to right of X axis
  Z axis – Positive towards the earth, perpendicular to XY plane

A third coordinate system is used as a reference frame when describing the attitude of the aircraft:

- Earth axes
  X axis – Positive towards the north
  Y axis – Positive towards the east
  Z axis – Positive towards the earth, perpendicular to XY plane

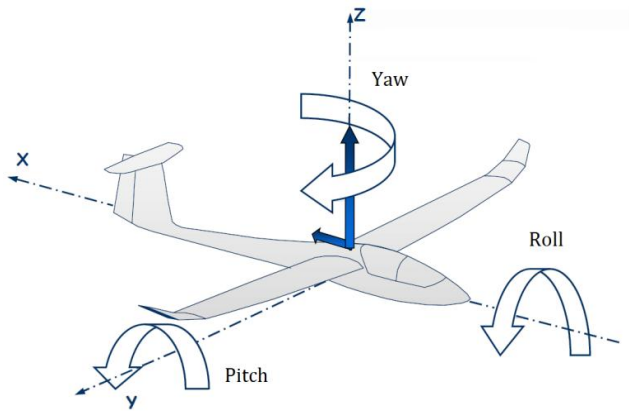


Figure 12: Yaw, Pitch, and Roll rotations around body axis

In flight dynamics, yaw, pitch and roll angles describe the absolute attitude angles relative to the earth fixed reference, and the changes in attitude (or movement) relative to the current orientation of the aircraft. They are defined as the following:

- Yaw – Angle of the of Z body axis relative to the north (X earth axis). Rotation around the Z body axis.
- Pitch – Angle of Y body axis relative to the horizon (Y earth axis). Rotation around the Y body axis
- Roll – Angle of X body axis relative to the horizon (X earth axis). Rotation around the X body axis

In general, the body reference frame is not aligned with the earth reference frame unless flying level. The body orientation can define by a quaternion, rotation matrix (DCM), or three Euler angles (as was described above with yaw pitch and roll angles).

4.2 CONTROL INPUTS

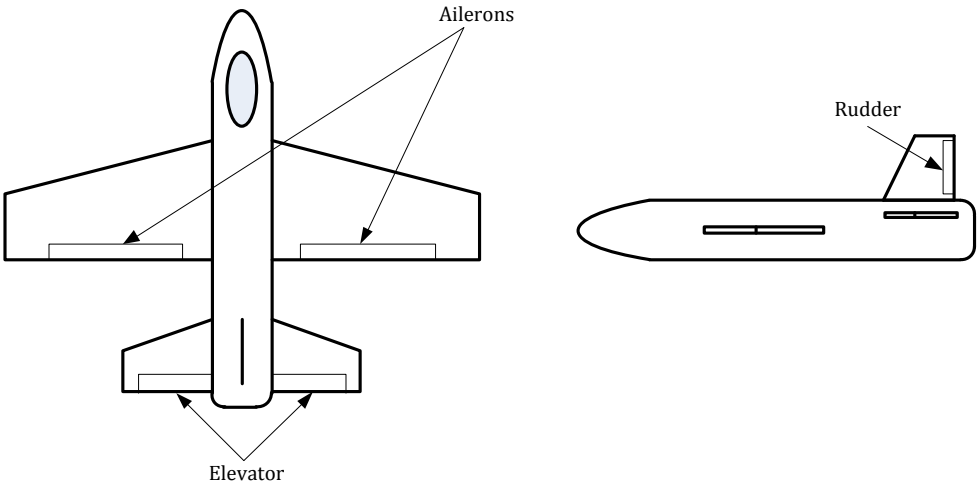


Figure 13: Aircraft Controls

A typical fixed wing aircraft is made of a main wing, a horizontal stabilizer, vertical stabilizer, and a fuselage. The typical or minimum flight controls of such a configuration are the following:

- Ailerons – Mounted on the trailing edge of each wing which move in opposite directions. Raised ailerons reduce lift and lowered ailerons increases lift causing the aircraft to roll left or right.
- Elevator – Mounted on the trailing edge of the horizontal stabilizer. They move up and down together (unlike ailerons). Raising the elevator pushes on the tail causing the aircraft's nose to pitch up. The opposite is true when lowering the elevator, causing the aircraft to pitch down.
- Rudder – mounted on the trailing edge of the vertical stabilizer. Deflecting the rudder left or right causes the nose to yaw left or right.

4.3 AERODYNAMIC FORCES AND MOMENTS

Force and moments applied to the aircraft can be described in both the wind reference frame and body reference frame. Forces and moments are described as coefficients that depend on the dynamic pressure, the wingspan, the wing surface, and the MAC (mean aerodynamic cord).

The moments are defined around the body axes, and are noted as: N for a yawing moment, L for a rolling moment, and M for a pitching moment. Figure 14 shows the moments around the three body axes.

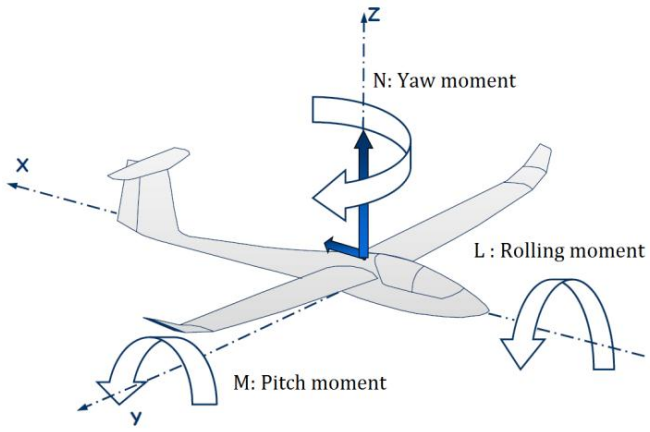


Figure 14: Aerodynamic moments

The forces are often noted lift, and drag which are defined on the wind axes, with the lift being the force perpendicular to the wind vector, and the drag being the force parallel to the wind vector. The forces can also be described using the body axis and they are noted Fx, Fy, and Fz.

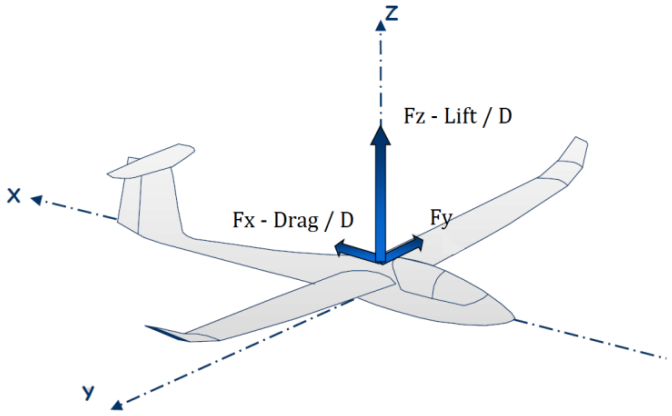


Figure 15: Aerodynamic forces

The forces and moments are expressed by dimensionless C coefficients, multiplied by the dynamic pressure and reference surface area (usually the surface of the wing). The dynamic pressure is defined by:

$$q = \frac{1}{2}\rho V^2 \quad (4.1)$$

q = dynamic pressure [Pa]
ρ = fluid density [kg/m^3]
V = fluid velocity [m/s]

When working with body axes, the three force and three moment coefficients are Cx, Cy, Cz, Cl, Cm, Cn. These coefficients depend on the operating point of the aircraft defined by the angle of attack, the Mach number, Reynolds number etc. how to obtain these coefficients is another subject entirely and will not be explained in this report.

Once the coefficients are known, one can easily calculate the exerted force applied to the aircraft at its center of gravity. Example of calculating the force exerted on the x body axis:

$$F_X = qS_{ref}C_X[N] \quad (4.2)$$

q: dynamic pressure [Pa]
S_{ref}: reference area [m^2]
C_X: X force coefficient [-]

The same is applied to moments, with the difference that an extra parameter is added: the cord reference length (width of the wing). Example for a pitching moment:

$$M_L = qS_{ref}C_{ref}C_L\,[Nm] \quad (4.3)$$

q: dynamic pressure [Pa]
S_{ref}: reference area [m^2]
C_{ref}: reference cord length [m]
C_X: X force coefficient [-]

Like the forces, the moments are applied to the center of gravity of the aircraft.

4.4 STABILITY AND CONTROL DERIVATIVES

Knowing the forces and moments exerted on the aircraft at a specific operating point is only half of the story, we also need to know how these forces and moments change when certain flight parameters are modified, such as the angle of attack of the aircraft, or what happens when the control surfaces are deflected.

This is where the stability and control derivatives come in. Stability derivatives describe how the force and moment coefficients change when the flight condition parameters of the aircraft are changed, such as the angle of attack, the side slip, as well as angular rotations (also known as body rate dampening).

Control derivatives describe how the forces and moments change according to the deflection of control surfaces, such as the increase in lateral (Y - body axis) force when some rudder is applied etc.

Here is an example of the shorthand notation of a derivative: $C_{M_{de}}$. The first subscript 'M' indicates it's a pitching moment (as shown in Figure 14). The second subscript '_de' stands for the elevator, meaning that this derivative indicates the change in pitching moment in response to elevator input.

Derivatives can be determined for all three forces and moments, coupled with every possible flight parameter. During this project we only worked using seven flight parameters totaling three control derivatives, and four stability derivatives:

Control derivatives:

- Aileron input '_da'
- Elevator input '_de'
- Ruder input '_dr'

Stability derivatives:

- Pitch rate '_q'
- Roll rate '_l'
- Yaw rate '_r'
- Sideslip '_b'

Using a control derivative, an analytical linearized (around a specific operation point) transfer function can be determined. For example, knowing the C_M_de and C_M_q derivatives, we can express a simplified elevator to pitch transfer function:

Using equation (4.3) for C_M_de and C_M_q and using Newton's law, we can write this differential equation:

$$eC_{M_{de}}C_{ref}S_{ref}q + {C_{ref}}^2 S_{ref}C_{M_q}q\dot{\theta} = I_{yy}\ddot{\theta} \quad (4)$$

e: elevator deflection angle [deg]
q: dynamic pressure [Pa]
I_{yy}: moment of inertia for pitch axis [kgm^2]
C_{ref}: reference cord length [m]
S_{ref}: reference surface area [m]
C_M_de: Elevator pitch moment derivative [Nm/deg]
C_M_q : Pitch body rate dampening [Nms/deg]

Converting to s-space using the Laplace transform gives us (assuming no initial conditions):

$$eC_{M_{de}}C_{ref}S_{ref}q + {C_{ref}}^2 S_{ref}C_{M_q}q\theta s = I_{yy}\theta s^2 \quad (4.4)$$

We then extract the θ/e transfer function:

$$\frac{\theta}{e} = \frac{\frac{C_{M_{de}}C_{ref}S_{ref}q}{I_{yy}}}{s^2 - \frac{{C_{ref}}^2 S_{ref}C_{M_q}q}{2I_{yy}}s} \quad (4.5)$$

The result is 2nd order transfer function with one pole, and one integration term. The same principle can be applied for the elevator and ruder controls, which also produces 2nd order transfer functions with one pole and integration.

The equation can then be expanded by adding the actuator (servo is our case) transfer function, which adds another pole.

This chapter was extremely simplified as covering all aerodynamic principles of a flying aircraft is not the subject of this project, but rather to explain some of the notions used when modeling the aircraft in Simulink.

5. THEORETICAL AIRCRAFT MODEL

When working with such a dynamic platform, simulation is an essential part of designing an autopilot. The initial idea when the project started was to design a complete Simulink model of the aircraft, with which the autopilot algorithm and control loops could be implemented and tested without risking any actual hardware.

During the semester project, an example of a full size general aviation aircraft model was found in Simulink's aerospace tool box. The aircraft could be flown using a joystick and the user could visualize the flight on Flight Gear, which is an open source flight simulator with which Matlab can interface and display a 3d aircraft and scenery.

This served as a great starting point for the Simulink model, as the flight dynamics of a large scale aircraft and a small RC airplane remain the same. After studying the full size aircraft Simulink model, it was concluded that all that was needed to change the model so it corresponded to the AXN Floater were its stability and control derivatives as well as its moment of inertia tensor. What had to be done was to compute these theoretical stability and control derivatives for multiple angles of attack (creating and aero model), and then replace those computed with those in the example model.

Computing such coefficients is no easy task, as even highly sophisticated CFD (Computational Fluid Dynamics) tools often provide erroneous results. Luckily, during the semester project a free and open source tool called XFLR5[2] was discovered. It is an analysis tool for airfoils, wings and planes operating at low Reynolds numbers, which was designed for small scale RC aircraft and gliders. Entering the wing geometry and airfoils of the aircraft, XFLR5 computes theoretical force and moments curves, or polars. It can also perform a stability analysis providing longitudinal and latitude modes pole locations.

Unfortunately, extracting all stability and control derivatives only using XFLR5 is difficult, and was not achieved. That is when another open source tool was discovered called AVL[3]. Using AVL, it was much easier to calculate stability and control derivatives at multiple operating points (differing the angle of attack and airspeed). The problem resided in the user interface that is entirely console based, as entering the airplane geometry is done using a special description language. XFLR5 however, is capable of exporting an aircraft geometry to AVL by generating the needed description files and calculating an estimated moment of inertia tensor and mass.

For this reason, both tools were used together. XFLR5 was used to enter the airfoils, wing geometries, and mass. AVL was then used for computing the stability and control derivatives for different angles of attack at a specific airspeed.

The method seemed sound, but relied on one very important factor: it required accurate dimensions of the wings and airfoil. This was done using optical 3D scans of the wing, which were then imported in pro-engineer and measured.

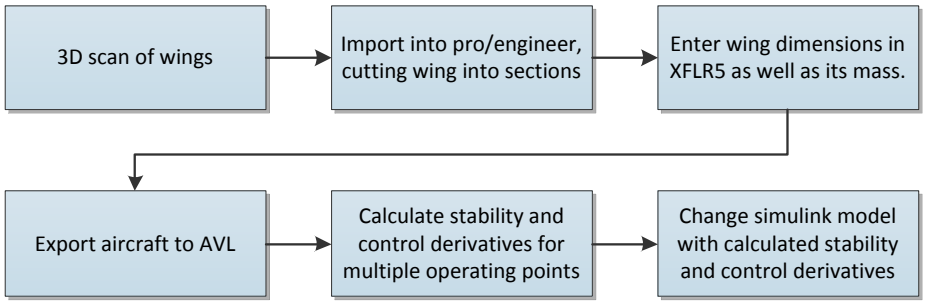


Figure 16: Aircraft model design flow

[2] http://xflr5.sourceforge.net/xflr5.htm
[3] http://web.mit.edu/drela/Public/web/avl/

5.1 OBTAINING AXN FLOATER WINGS GEOMETRY USING 3D SCANS

Figure 17 shows the final result of the optical 3D scan. The small holes on the surface are due to small stickers that were applied to the wing that served as reference points during the scan.

Figure 17: AXN Floater 3D wing

To enter the wing correctly in XFLR5, the wing had to be separated into sections, with each section defined by its position, cord length, and offset. To do so, a skeleton was overlaid over the top projection of the wing by hand, which provided good enough detail but without adding too many sections.

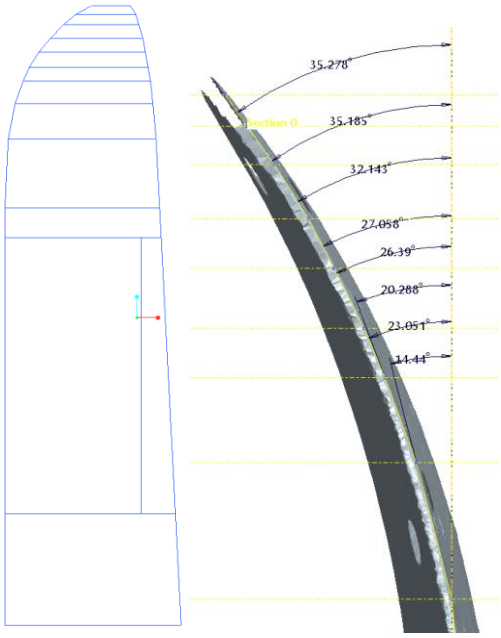


Figure 18: AXN Floater Wing skeleton and dihedral

In addition to the position, cord, and offset the section can also have a dihedral angle. The wing itself is horizontal in reference to the fuselage, but the tips of wing do have a slight curve. Thus another side view projection was added with the angles of each section. We now have everything required to create the wing in XFLR5. Figure 18 shows the wing skeleton with the different sections and the dihedral angles at the tip of the wing.

Moving on was horizontal stabilizer. Figure 19 shows the 3d scan obtained for the horizontal stabilizer:

Figure 19: AXN Floater Elevator 3D scan

Like for the main wing, the same steps were repeated, of creating a skeleton overlay with the least amount of sections possible while still remaining accurate. In Figure 20, we see the final skeleton of the horizontal stabilizer.

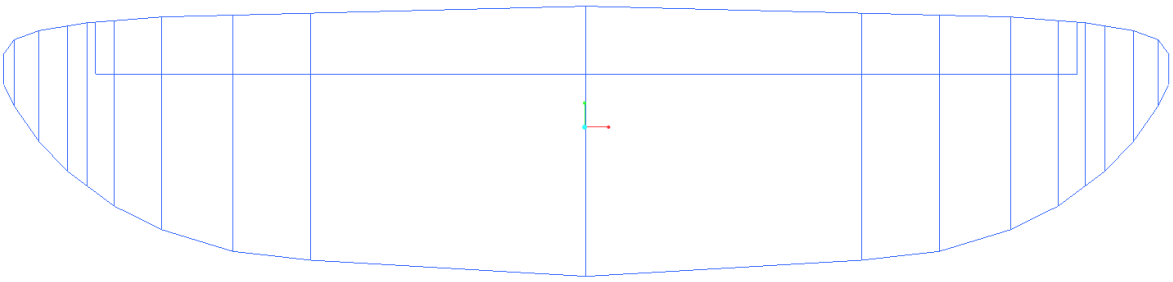


Figure 20: AXN Floater Elevator skeleton

Two unknowns remained, and they were the main wing and horizontal stabilizer airfoils. The 3d models were cut perpendicular using a cross-section view, which made the air foil visible. Figure 21 and Figure 22 both show the main wing cross section and the horizontal stabilizer cross section respectfully.

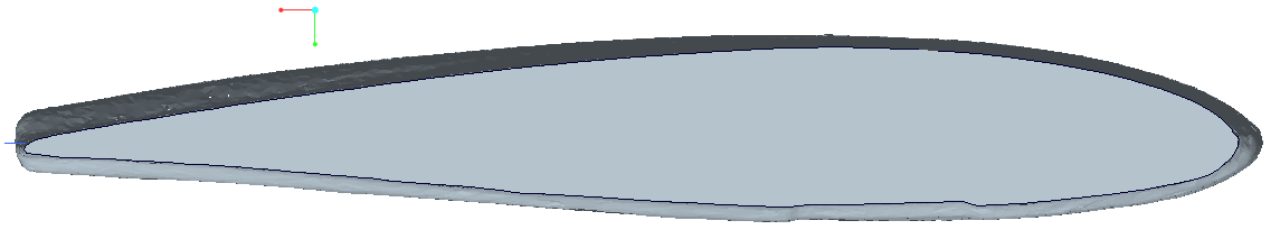


Figure 21: AXN main wing cross-section

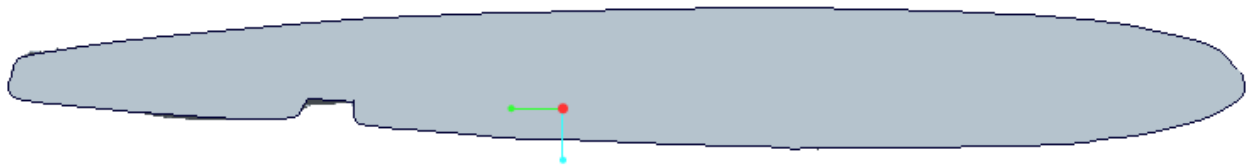


Figure 22: AXN horizontal stabilizer cross-section

Unfortunately, obtaining an accurate curve or point plot using pro/engineer was difficult, and only a visual approximation was used.

With the skeletons of the main wing and horizontal stabilizer, as well as the dihedral of the main wing tips, the wing geometries were completely defined and could be entered in XFLR5.

5.2 XFLR5 AIRCRAFT MODEL AND ANALYSIS

The first step of creating a XFLR5 plane model is entering the air foils of the different wings.

The airfoils were approximated starting with a standard symmetrical NACA 0012 for the horizontal stabilizer and vertical stabilizer, and a NACA 2412 for the main wing. They were then slightly modified within XFLR5, by changing the max thickness as well as the camber, so that they looked as close as possible to the cross sections obtained using the 3d scan.

Variants of the foils had to be created with hinges, which represent the ailerons , elevator and rudder. Figure 23 shows the two airfoils and their variants with different flap angles, and hinge positions.

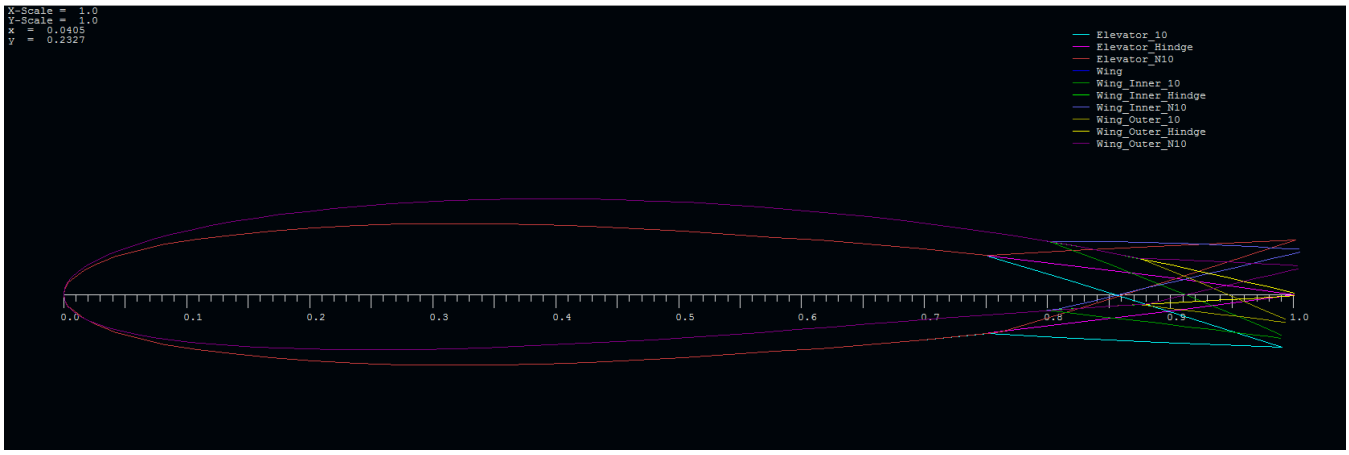


Figure 23: XFLR5 Airfoils

After entering the airfoils, XFLR must run an analysis using Xfoil to generate polars for different Reynolds numbers ranging from 10,000 to 500,000 covering the entire flight envelope of the aircraft. The important polars generated are the Cl-Cd, Cl-Alpha, Cm-alpha, and Cl-Cd/alpha. In Figure 26 we can see the type of graphs generated by Xfoil.

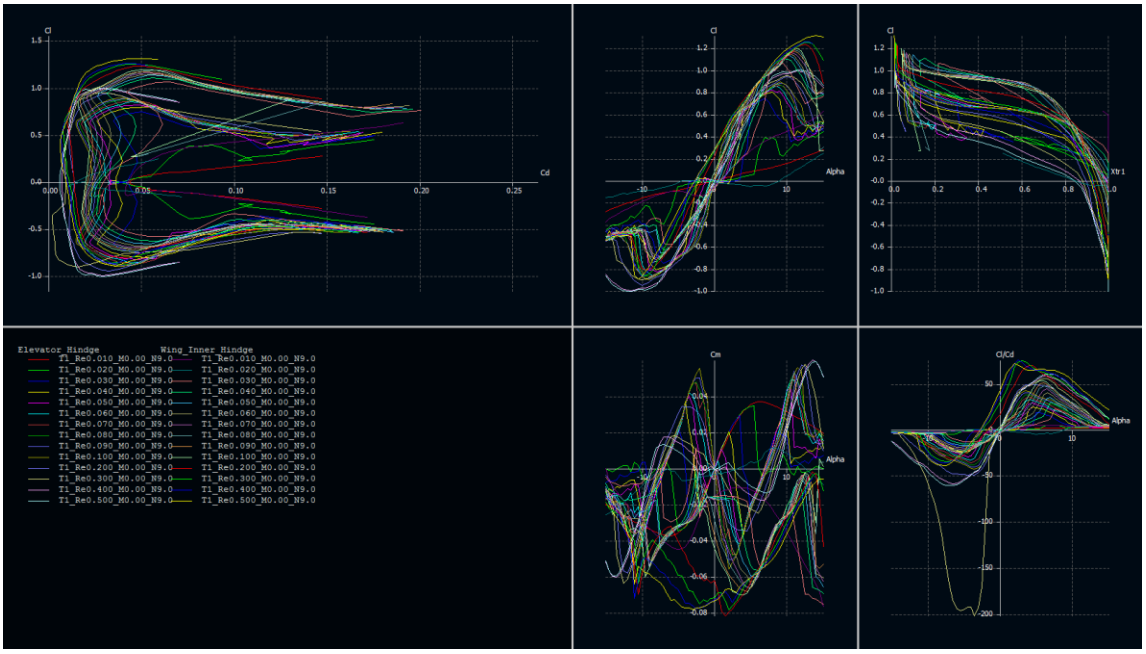


Figure 24: Xfoil polar analysis

The next step is to enter the wing geometries, using the measurements of the wing skeleton obtained during the 3D scans. The wing is entered by sections, defined by the position, cord, offset, dihedral and twist angle. Figure 25 shows the wing as it was entered in XFLR5.

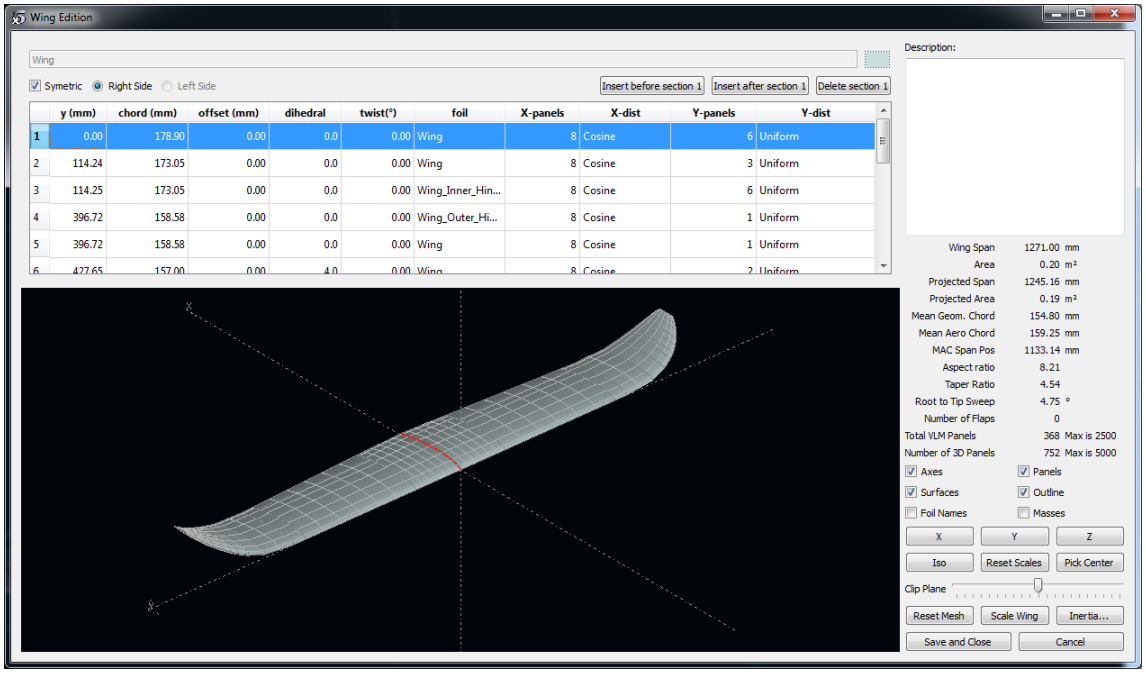


Figure 25: XFLR5 AXN main wing

The same steps were repeated for horizontal and vertical stabilizers, after which a new airplane is defined. In the airplane, the wings and stabilizers are imported. At this point, the mass of the wings can be defined, as well as point

masses. The point masses represent the electronics, servos, motor, batteries, etc. It then calculates an estimated moment of inertia tensor for the aircraft. After reading a comparison of XFLR5 predictions with experimental results[4], it was concluded that the addition of the aircraft body, or fuselage, is more of a nuisance than help. This approximation is acceptable for a glider, as the wing span is very important compared to the size of the fuselage. Additionally, adding the body of the aircraft would have been difficult, and for those two reasons it was chosen not to include the body in the analysis.

After the airplane was completely defined, a fixed airspeed analysis was completed. The airspeed was fixed at an arbitrary 7 m/s, and the force and moments acting on the plane were then computed for different angle of attacks ranging from – 10 to 20 degrees. Figure 26 shows the graphic result of such analysis, with the pressure distribution along the wings, the airstream, as well as the downwash.

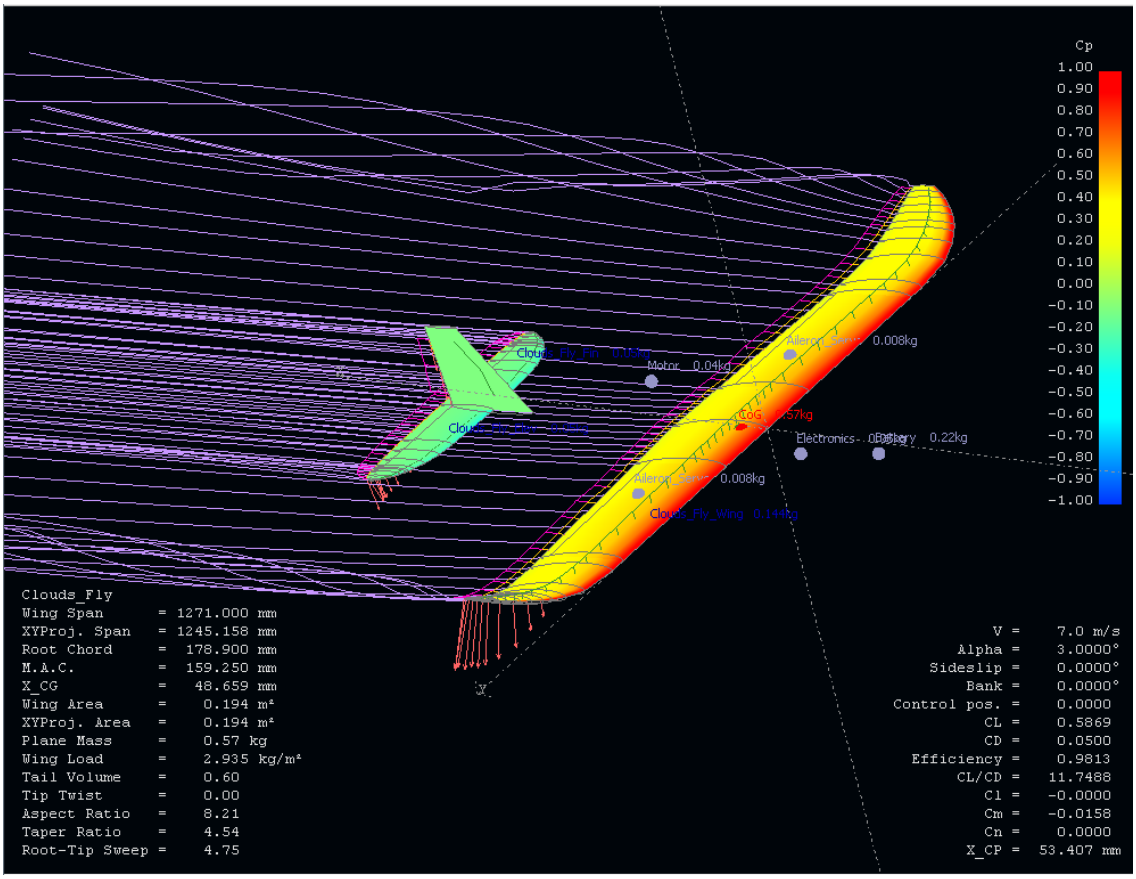


Figure 26: XFLR5 AXN Floater Analysis

More importantly, it also calculates the polars of the entire aircraft. Figure 27 shows various polars of the aircraft giving insight on the various forces and moments coefficients, and how they change vs. the angle of the attack.

[4] Deperrois A. 2009, *About XFLR calculations and experimental measurements*

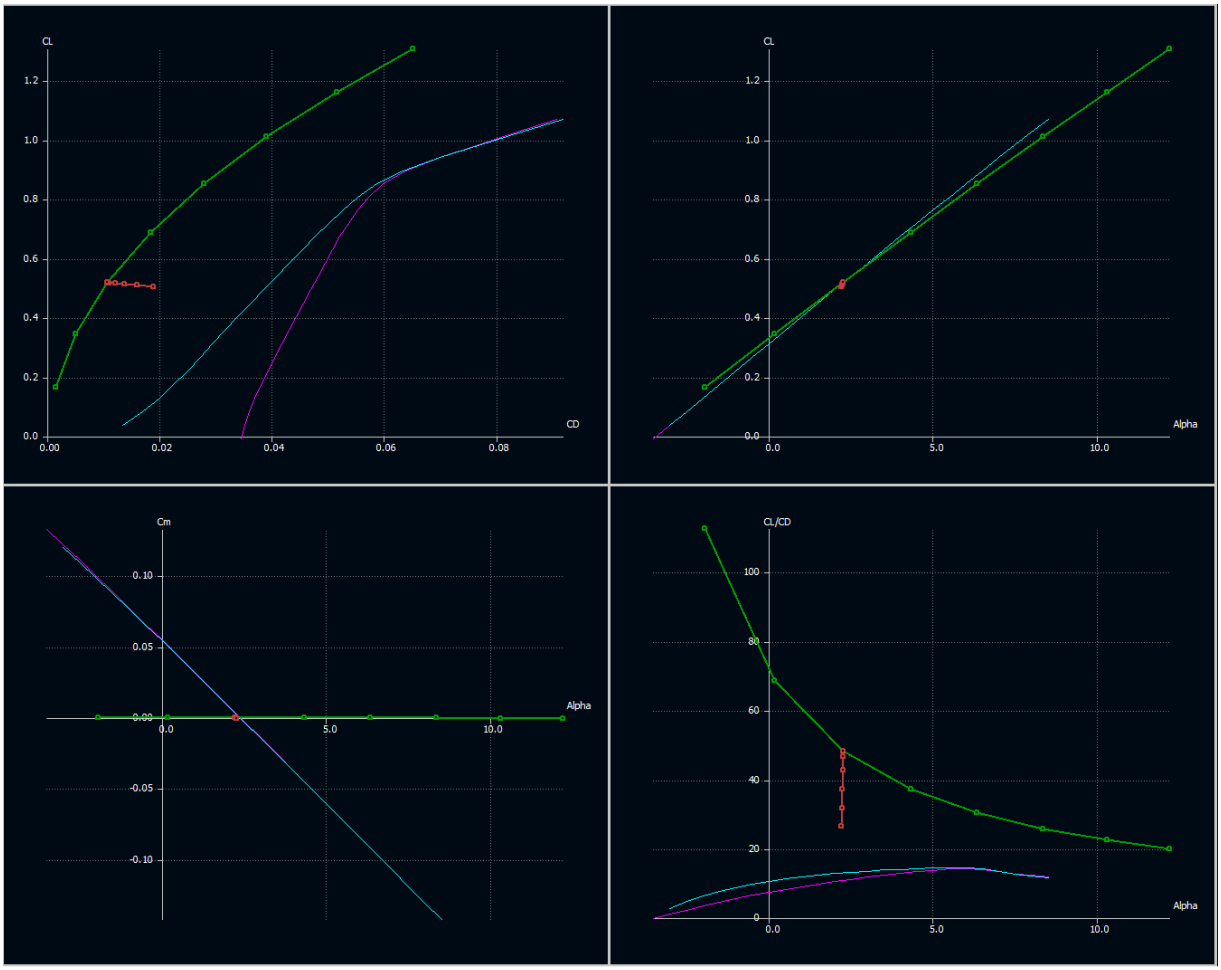


Figure 27: XFLR5 AXN Floater polars

Unfortunately, these polars are not sufficient to model the entire aircraft, as they only provide a fraction of the total coefficients required for the Simulink model.

It was at this point that the aircraft was exported using the XFLR5's built in AVL export function which creates an AVL aircraft description file (.avl), and mass file (.mass). The .avl file contains a written description of the aircraft geometry, and the .mass contains the moment of inertia tensor calculated in XFLR.

5.3 AVL STABILITY AND CONTROL DERIVATIVES

After importing the .avl and .mass file, we can plot the aircraft in AVL making sure that the aircraft still looks correct. Figure 28 shows the aircraft that was imported via the .avl description file. The airfoils are not displayed, but the overall wing shape matches that which we saw in XFLR5.

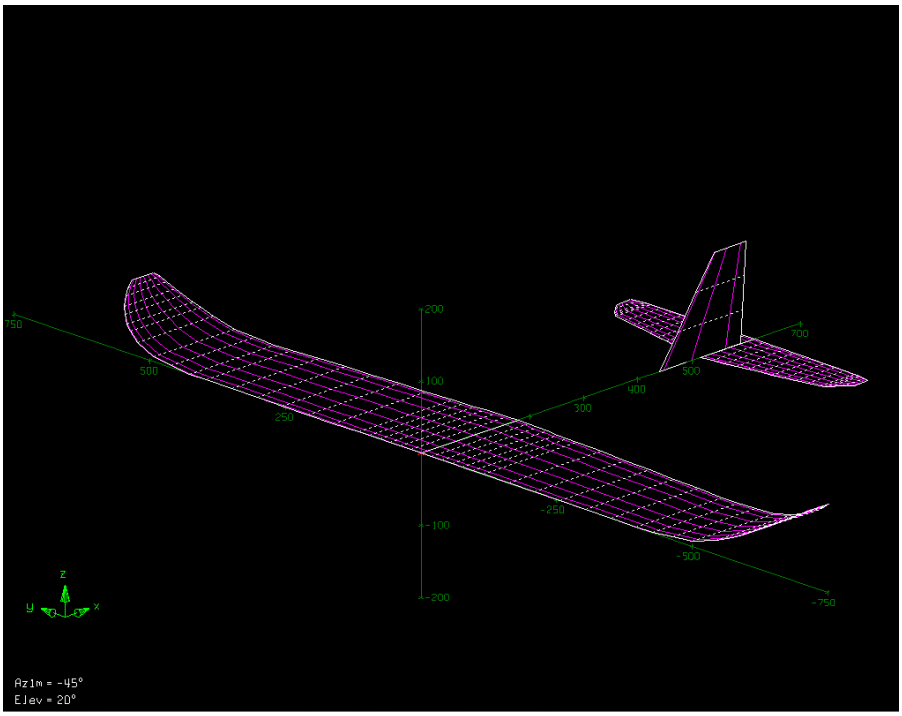


Figure 28: AVL AXN Floater geometry plot

After importing the files, operating point analyses were completed for different angles of attack. Each operating point analysis was configured at a constant airspeed of 7 m/s, and a specific angle of attack. An operating point analysis was ran for a wide range of angle of attacks ranging from -20 degrees to 20 degrees. Each analysis returned the force and moment coefficients on the x,y,z body axis, as well as the stability and control derivatives in the body axis.

Cx,Cy,Cz are the force coefficients, and Cl,Cm,Cn are the moment coefficients. These six coefficients describe all the forces and moments that are acting on the aircraft at a given time.

Stability derivative coefficients are defined by a force or moment coefficient follow by a '_q', '_p', or '_p' subscript, with q, p, and r representing the angular rates around the pitch (q), roll (p), and yaw (r) axis. For example, Cm_q represents the addition to the Cm coefficient, proportional to the pitch rate.

Control derivatives coefficients, like the stability derivatives, describe how the force or moment coefficients change but due to the angles of the control surfaces (ailerons, elevator, rudder, etc). The subscripts are '_de' for elevator, '_da' for ailerons, and '_dr' for rudder. For example, Cz_de represents the added vertical force applied to the z axis, proportional to the elevator deflection angle.

Using AVL, all the force and moment coefficients as well as derivatives were calculated for different angle of attacks.

Stability and control derivatives and total forces and moments vs. the angle of attack – AXN Floater Aero model:

Alpha	Cz	Cx	Cm	Cz_q	Cx_q	Cm_q	Cz_de	Cm_de	Cy_b	Cy_p	Cy_r	Cy_dr
-20	1.36995	0.41486	0.45702	-9.12957	-2.80251	-14.76661	-0.00657	-0.02008	-0.18813	-0.351905	0.079568	0.001474
-15	0.98471	0.22113	0.37299	-9.36694	-2.18183	-15.20493	-0.00696	-0.02127	-0.20097	-0.295995	0.107029	0.001567
-10	0.56163	0.0841	0.27529	-9.53303	-1.54453	-15.52753	-0.00725	-0.02215	-0.21228	-0.237833	0.133675	0.001636
-5	0.11354	0.00793	0.16691	-9.62656	-0.89549	-15.73197	-0.00743	-0.02271	-0.22197	-0.17786	0.159304	0.00168
-2	-0.16166	-0.00752	0.09811	-9.64754	-0.50247	-15.79722	-0.00749	-0.02289	-0.22698	-0.141197	0.174111	0.001693
0	-0.34592	-0.00507	0.05114	-9.64683	-0.23963	-15.81667	-0.0075	-0.02293	-0.22998	-0.116534	0.18372	0.001697
2	-0.52977	0.00758	0.00354	-9.63437	0.023509	-15.81685	-0.0075	-0.02292	-0.23269	-0.091728	0.193105	0.001696
5	-0.80281	0.04552	-0.06852	-9.59368	0.418058	-15.781	-0.00746	-0.02281	-0.23623	-0.05432	0.206738	0.001686
10	-1.24325	0.15813	-0.18841	-9.46752	1.072562	-15.62522	-0.0073	-0.02233	-0.24068	0.008306	0.228183	0.001649
15	-1.65384	0.32936	-0.3049	-9.2693	1.718902	-15.35053	-0.00704	-0.02153	-0.24331	0.07087	0.247892	0.001586
20	-2.02211	0.554	-0.41445	-9.00054	2.352161	-14.95901	-0.00668	-0.02042	-0.24408	0.132894	0.265713	0.001499

Alpha	Cl_b	Cl_p	Cl_r	Cl_da	Cl_dr	Cn_b	Cn_p	Cn_r	Cn_da	Cn_dr	Cy_da
-20	0.000782	-0.45388	-0.12649	-0.00313	0.000071	0.102746	0.267294	-0.05347	0.000352	-0.000585	0.000262
-15	-0.02307	-0.47261	-0.06877	-0.00328	0.000071	0.085155	0.194673	-0.05478	0.000254	-0.000622	0.000252
-10	-0.04961	-0.48774	-0.01053	-0.00339	0.00007	0.071972	0.120571	-0.05567	0.000149	-0.000651	0.000238
-5	-0.07803	-0.49916	0.047786	-0.00344	0.000068	0.063598	0.045552	-0.05614	0.000041	-0.000669	0.00022
-2	-0.09564	-0.50419	0.082631	-0.00345	0.000066	0.060995	0.000337	-0.05622	-2.4E-05	-0.000675	0.000208
0	-0.10748	-0.50678	0.18372	-0.00344	0.000064	0.060288	-0.02982	-0.05618	-6.7E-05	-0.000676	0.000199
2	-0.11935	-0.50875	0.128723	-0.00342	0.000063	0.06041	-0.05993	-0.05608	-0.00011	-0.000676	0.00019
5	-0.13706	-0.51054	0.162893	-0.00339	0.00006	0.062142	-0.10495	-0.0558	-0.00017	-0.000673	0.000175
10	-0.16587	-0.51042	0.218804	-0.00328	0.000055	0.069103	-0.1793	-0.05499	-0.00027	-0.000659	0.00015
15	-0.19302	-0.50641	0.27305	-0.00313	0.00005	0.080961	-0.25227	-0.05376	-0.00036	-0.000634	0.000123
20	-0.21771	-0.49855	0.325218	-0.00294	0.000044	0.097355	-0.32333	-0.05212	-0.00044	-0.0006	0.000096

Datum Coefficients **Body Rate Dampening** **Ailerons** **Elevator** **Rudder**

Table 1: AXN Stability and control derivatives, and datum coefficients

5.4 SIMULINK MODEL AND SIMULATION WITH FLIGHTGEAR

As stated earlier, the initial model was obtained from a Matlab example for a large scale aircraft. After obtaining the theoretical stability and control derivatives, they were entered in the Simulink model as look up tables (with the angle of attack as the input).

The aircraft is structured in two main blocks, the environment block, and the aircraft block. The environment block simulates the atmosphere around the aircraft such as the air temperature, air density, as well as wind and turbulence. The aircraft block calculates the flight parameters such as the airspeed, angle of attack, side slip angle, etc. Then using those flight parameters it calculates the force and moment coefficients using the look up tables and the control inputs. The coefficients are then translated to actual forces and moments using the reference wingspan, reference cord length, reference surface area and dynamic pressure.

External forces such as gravity are then added, and the sum is connected to a 6DOF (degrees of freedom) block which represents the aircraft (position, velocity, attitude, etc).

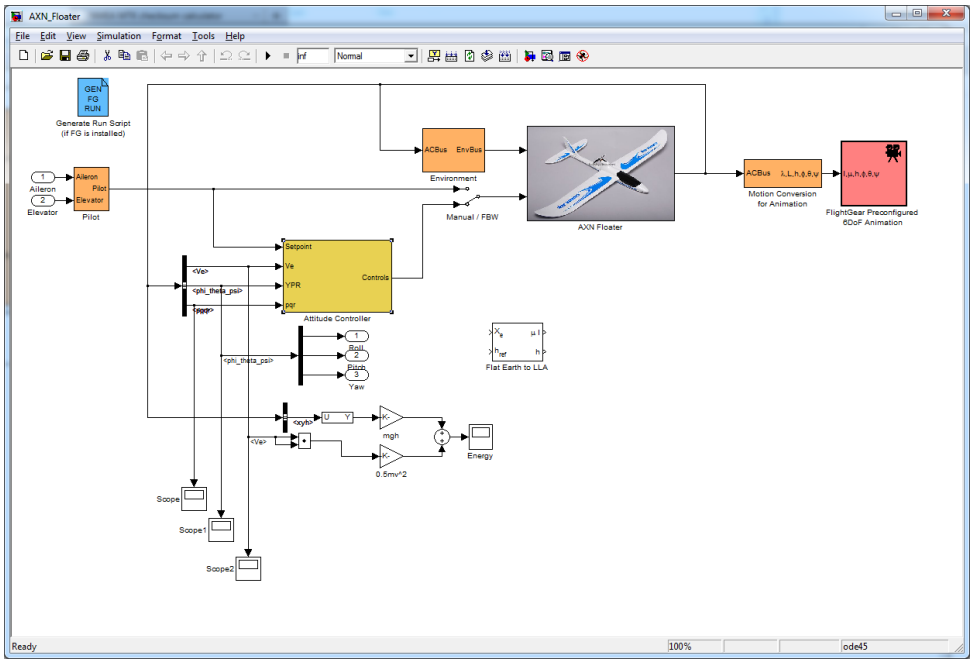


Figure 29: AXN Simulink – Top

Figure 29 shows the top view of the Simulink model with the two main aircraft and environment blocks. Controlling of the aircraft actuators (servos) could be done using either a joystick, or the attitude controller block which was used test different control loops.

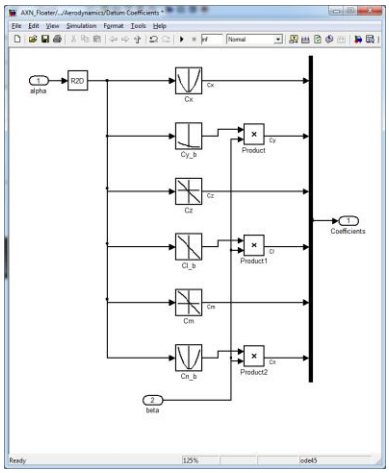


Figure 30: AXN Simulink - Datum Coefficients

Figure 30 shows the datum coefficients block, which calculates the total forces and moments applied to the aircraft regardless of the control inputs. The block contains look up tables using the angle of attack with data corresponding to Table 1: AXN Stability and control derivatives, and datum coefficients (blue).

In addition to the datum coefficients block, there are three control blocks with the control derivatives of the ailerons, rudder, and elevator. Figure 31 shows the aileron control derivatives block, which by using the angle of attack, looks up the coefficients which are then multiplied by the ailerons deflection angle. The results are then added to the global force and moments coefficients. Some derivatives contain 0 meaning that the ailerons have no effect on that coefficient, such as Cz (vertical force). This block is repeated for the elevator and rudder.

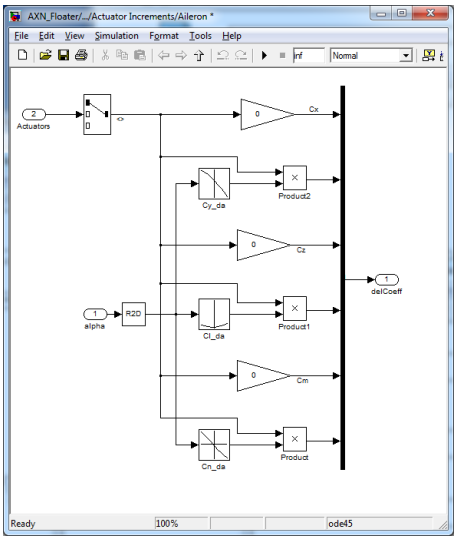


Figure 31: AXN Simulink – Aileron control derivatives

In addition to the aircraft model blocks, external blocks which serve as the control loops were added. These made it possible to test different ideas and control algorithms (as seen in Figure 29). Figure 32 shows one of the configurations that were tested, which ended up being the one used to the end and is explained in chapter 6.

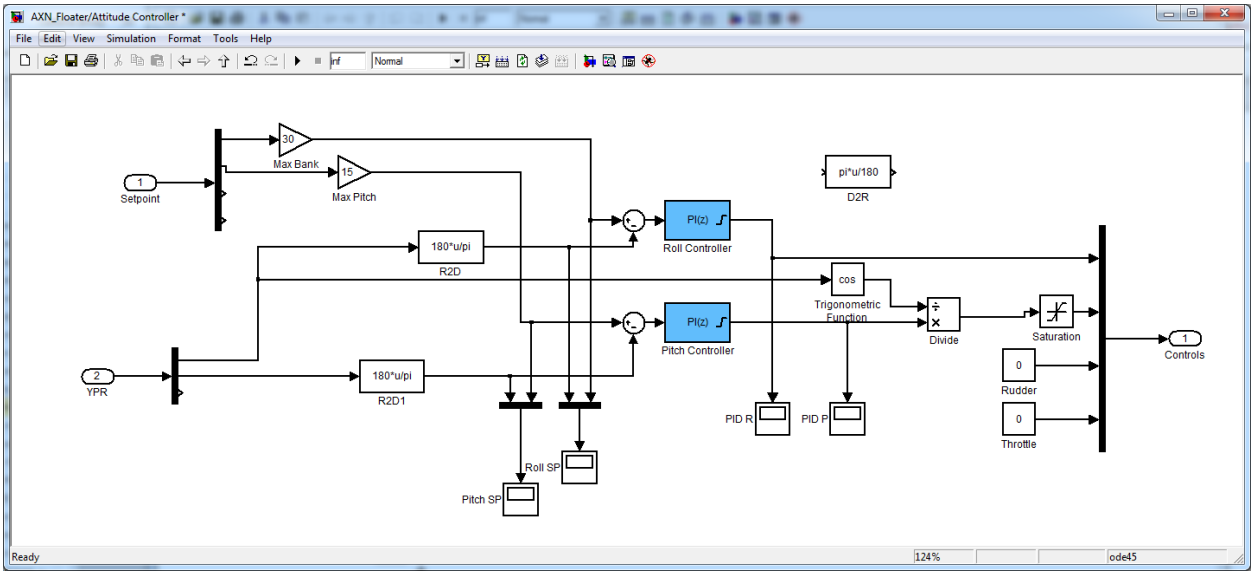


Figure 32: AXN Simulink - Attitude Controllers

Matlab is able to interface with FlightGear, an open source flight simulator. For the AXN model, flight gear is used to display the aircraft in a scene that allows us to visualize the aircraft flying in real time. Flight Gear in this case serves only as a convenient way to visualize the aircraft and nothing more; absolutely all calculations are done within the Simulink. Adding a joystick, we were able to fly our 100% theoretical aircraft in real time just like any other flight simulator. Figure 33 shows the flightgear visualization. The aircraft in the picture is an arbitrary model which does not reflect the AXN floater, and only serves as a representation.

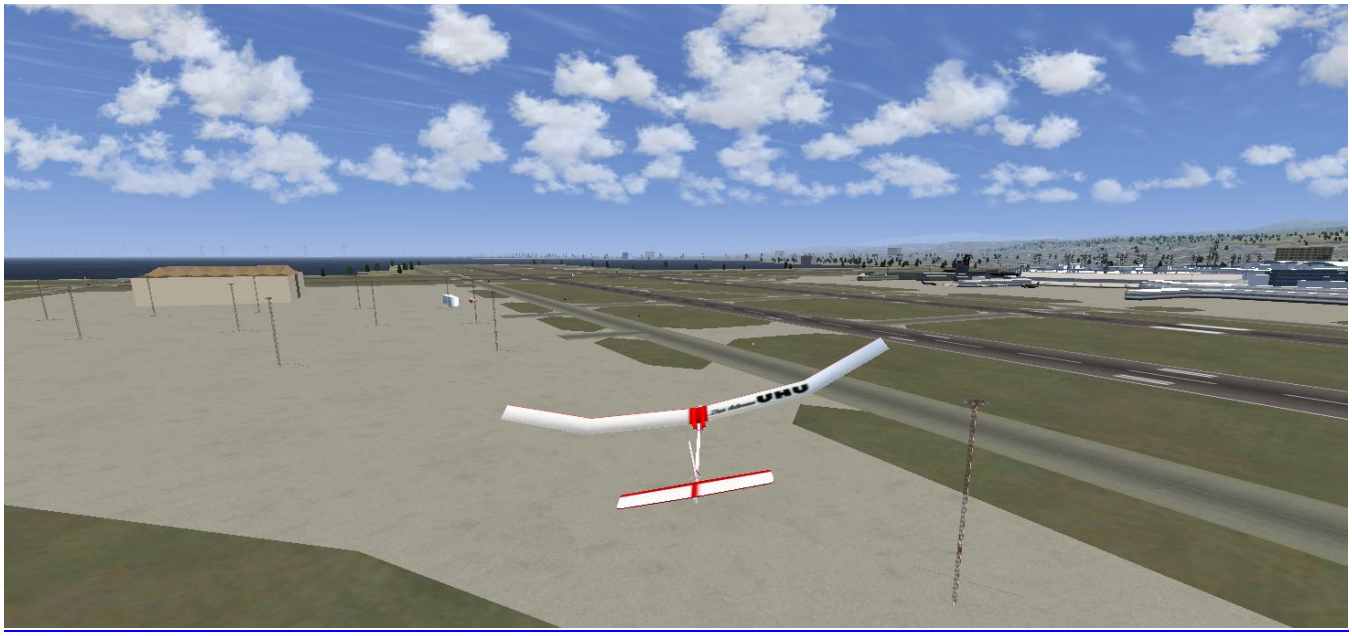


Figure 33: Flight Gear visualization

The glider model flew and responded as expected from a small RC plane, but the overall handling did not match what was observed when flying the actual AXN floater in real life. Additionally, the model had some errors, especially

regarding the drag force as the model could glide for a very long time and for unrealistic distances. The model would have to be corrected for it to match the real life AXN in order for it to be used for designing the control loops. The initial plan was to correct the model using experimental data from manual flight logs. Unfortunately, correcting the model based on experimental data is no easy task as only a limited number of data points are measured and stored, while still lacking some very important measurement such as the angle of attack (which is the primary parameter for the entire Simulink model). It later turned out to be much easier to identify smaller SISO (single in single out) systems for every control surface (aileron, elevator, and rudder). For these reasons, the model was left as is and not improved any further. However, the frame work is in place and it still remains a valid method of simulating algorithms should an aircraft operate in more extreme conditions that simply cannot be easily tested in real time (such as high altitude weather balloon launches).

Although the Simulink model was not directly used when designing the control loops of the aircraft, it provided great insights in the dynamics of the aircraft, how the aircraft reacts to control inputs, the cross coupling between the different axes and control surfaces etc. It still simulates a small scale aircraft (even if not the AXN Floater) and could be used to try different types of control loop options, and see how well they worked. These findings were a big factor in the choices made when designing the control loops in chapters 6 and 7.

6. CONTROL LOOPS ARCHITECTURE

The overall control loops architecture was split in two cascaded loops. They are called the attitude control loop and the navigation control loop.

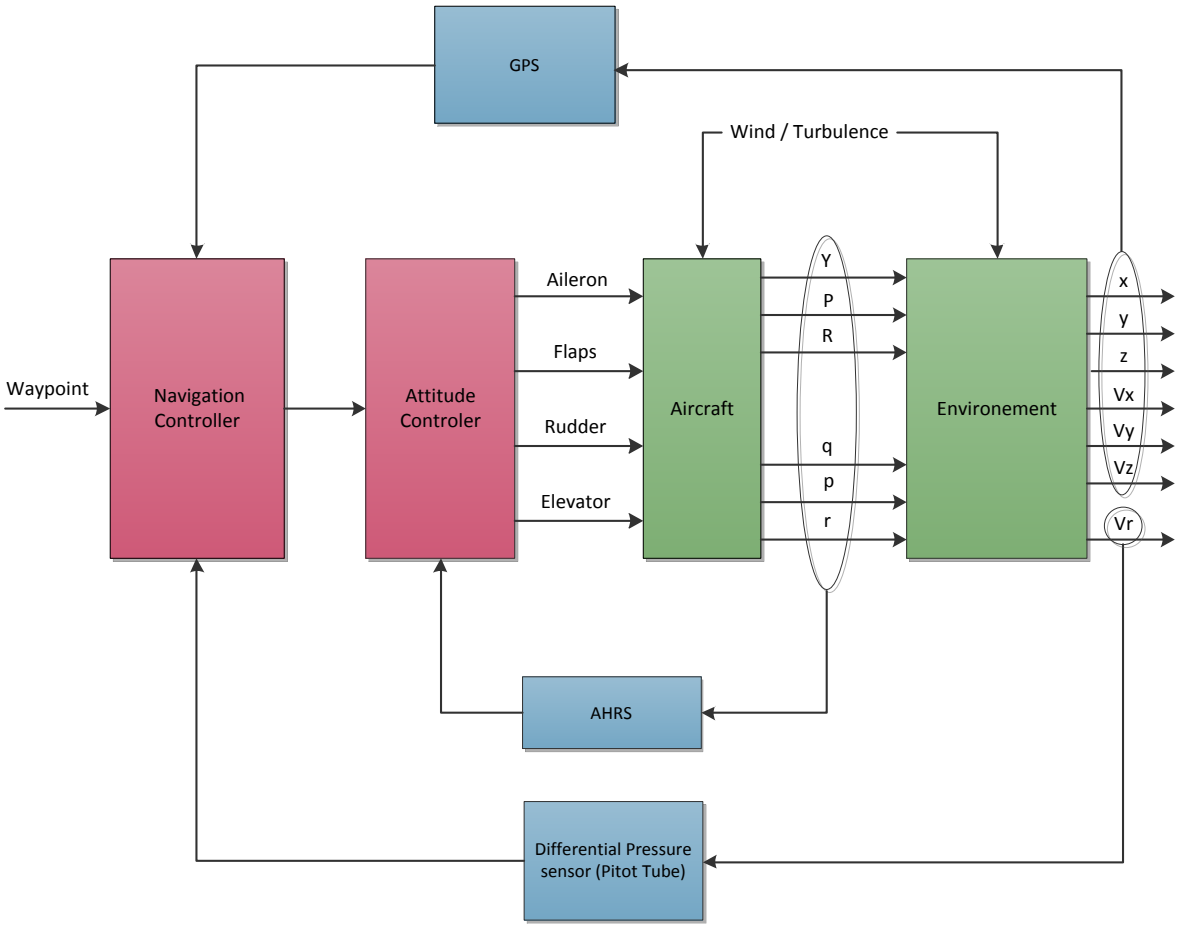


Figure 34: Top control loops architecture

The attitude control loop is charged with the task of controlling the attitude, or orientation in 3d space, of the aircraft. The feedback is done using the AHRS (attitude and heading reference system) sensor which measures the aircraft attitude under different formats, such as Euler angles, a quaternion, or a direction cosine matrix.

The navigation loop is the second outer loop which calculates the setpoint of the attitude loop based on the current and desired position of the aircraft, as well as its airspeed. The feedback is done using a GPS to measure the position and heading of the aircraft, and a differential pressure sensor with a Pitot tube is used for the airspeed.

The initial idea was to design a state space controller as they are very suitable for MIMO (Multiple In Multiple Out) systems such as ours. The problem with designing a state controller was that it required an accurate state space model of the aircraft that would be difficult to obtain experimentally.

It was then decided to use SISO (Single In Single Out) controllers in parallel, with one controller per axis (pitch and roll). This decision was based on the hypothesis that the cross coupling between the roll and pitch axis was either negligible, or of known quantity. To do so the AHRS was set to Euler angles, with each angle serving as a feedback for one attitude control loop. Euler angles were chosen as they allowed a quick implementation of the control loops when

using the parallel SISO configuration. It must be known though that using Euler angles have their limits. If the aircraft is pointed either directly up or down, the control surfaces (ailerons, elevator, and rudder) no longer act on the correct axis. Example: when flying perfectly vertical, the ailerons no longer affect the roll angle of the aircraft, but the yaw angle. The aircraft must therefore be confined to a specific flight envelope, meaning it cannot fly in extreme orientations such as vertical.

By considering the aircraft is confined to small (<45 degree) rotations, the roll angle is controlled using only the ailerons, and the pitch angle of the aircraft is controlled using the elevator. This control loop configuration was tested within the Simulink model (with some auto tuned PID gains) and it stabilized the aircraft even in sharp turns or dives.

One important point that had to be considered was the cross coupling of the axes. That is the output of one control surface such as the elevator having an effect on the yaw of the aircraft, or how the roll of the aircraft changes the elevator → pitch transfer function.

In our case, the roll angle of the aircraft reduces the effect of the elevator acting on the pitch angle of the aircraft.

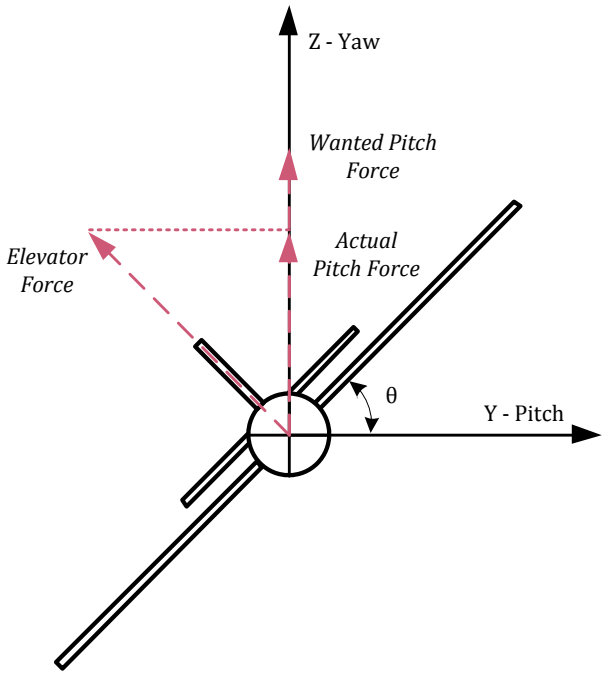


Figure 35: Elevator force pitch projection – body axes

Consider Figure 35 above with an aircraft turning with a 45 degree bank angle. If the pitch controller is using static non adaptive gains that were designed with a level aircraft in mind, the moment applied to the pitch axis is proportional to the projection of the elevator force on the z axis. These forces are drawn as the elevator force, the actual force applied on the Z-axis (which translates to a Y axis pitching moment), as well as the wanted pitch force which is the force that would be exerted if the aircraft was flying level.

There are two solutions to correct this roll angle dependency. The first possibility is to use an adaptive controller instead of a static one. This would mean that the gains or coefficients of the controller would change corresponding to the roll angle of the aircraft. Doing so would require knowledge of how the transfer function changes with the roll angle which is difficult to obtain experimentally.

If we accept the hypothesis that the elevator force is linearly proportionate to the elevator deflection angle (as described with control derivatives in 4), we can correct the deflection directly at the output of the controller. By dividing the output of the controller by the cosine of the roll angle, we are increasing the elevator deflection angle so that the projected force on the Z-axis is the actual wanted force that the controller was designed to apply. This method works regardless of the controller used and is simple to implement. As an added bonus, should the aircraft roll over on its back the cosine term inverts the pitch output, which is desired.

The final attitude control loops with the cosine roll dependency correction gives us the following:

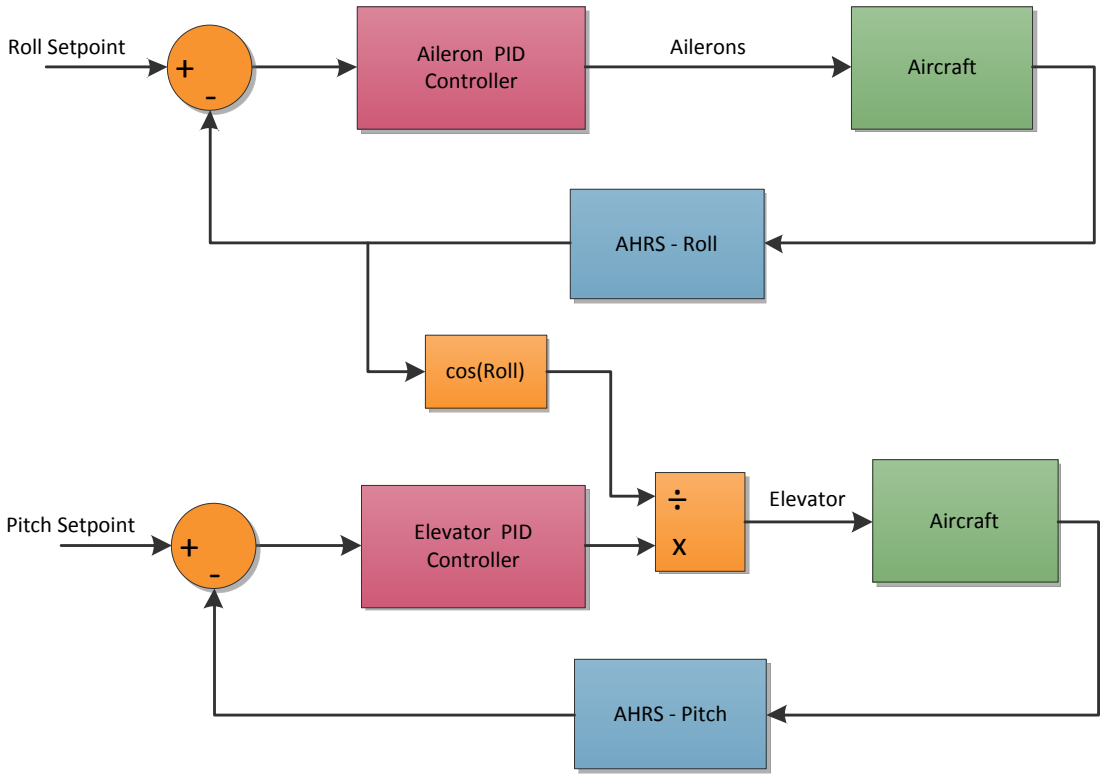


Figure 36: Attitude control loops

With the attitude control loops defined, we can move on to the navigation loops. The navigation controllers are higher level loops that control the setpoints of the attitude controller based on the aircrafts current and desired bearing, as well as its airspeed.

Considering the user will input GPS waypoints, the bearing setpoint will need to be calculated based on the waypoint and the aircrafts current latitude and longitude (see 8.1).

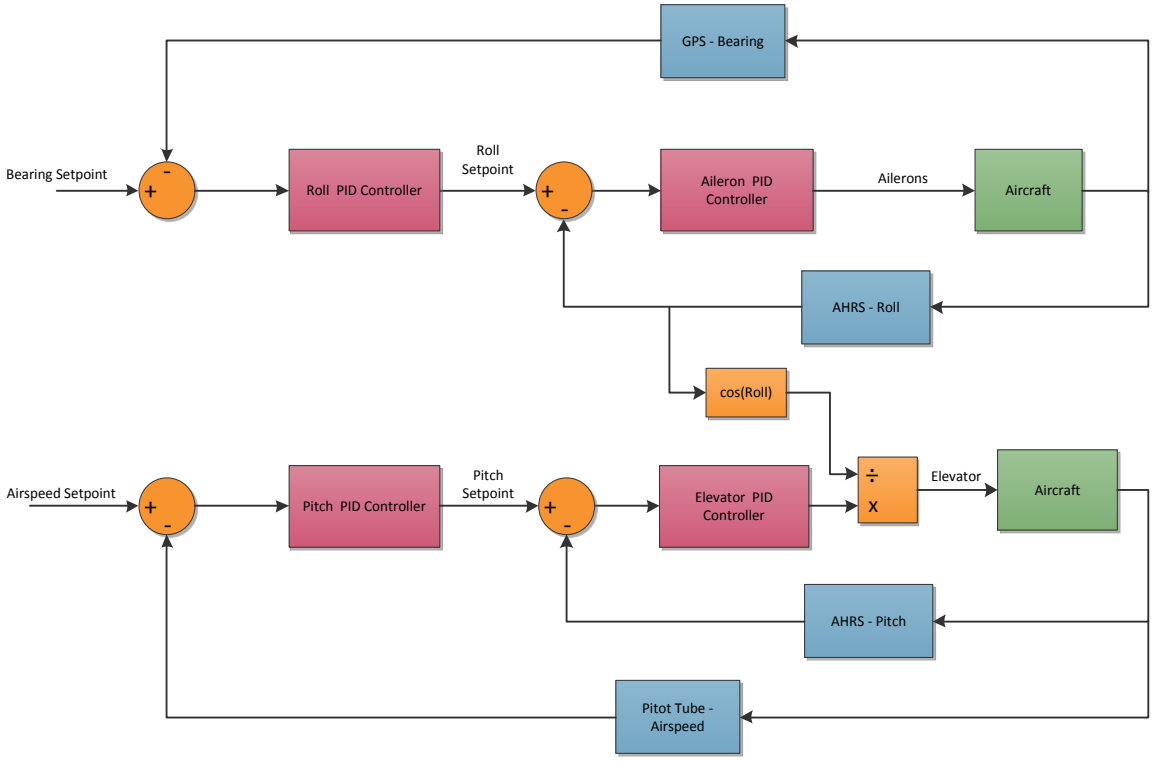


Figure 37: Attitude and Navigation control loops

For testing purposes, the autopilot was programmed with two operating modes:

Auto – The aircraft is entirely controlled by the autopilot, and thus requires that a trajectory be entered in the autopilot. Figure 37 represents the control loops when operating in auto mode

Fly by wire (FBW) – in FBW mode, the navigation control loops are removed and replaced with user inputs via the radio. The joystick position on the transmitter is translated to a roll or pitch setpoint angle by multiplying the position value by the max pitch or roll angle, as shown in Figure 38. In this mode the user still maintains control of the aircraft using the radio, but through the attitude loops.

Having these two operating modes made it possible to split the controller design process in two, giving the possibility to test and validate the attitude controller before designing the navigation controllers and testing the aircraft in autopilot mode.

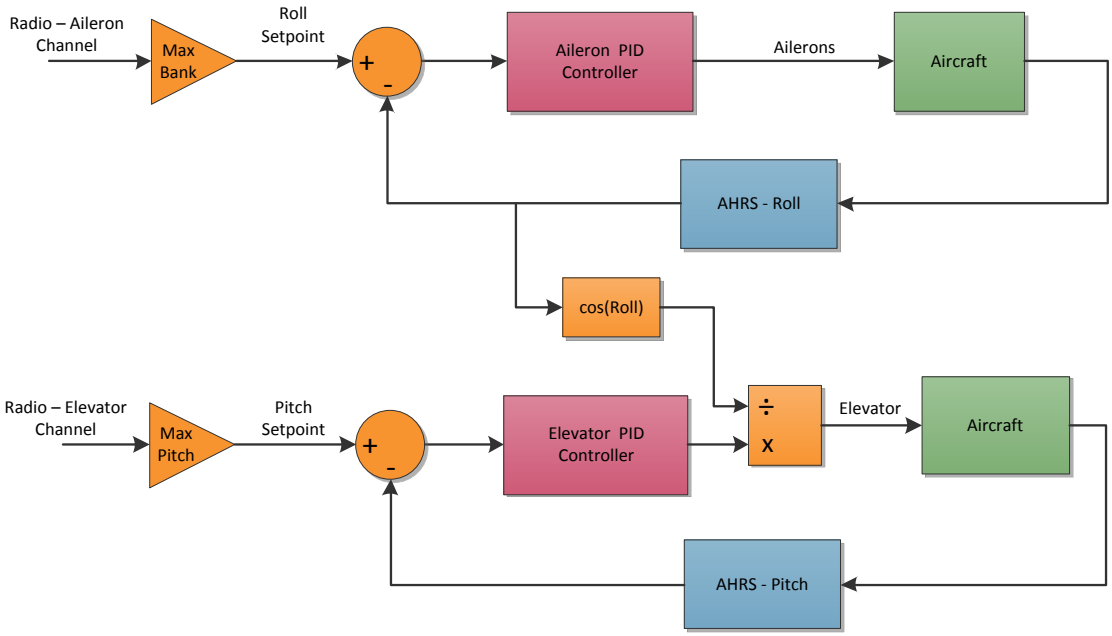


Figure 38: Attitude and Fly-by-wire control loop

7. SYSTEM IDENTIFICATION AND CONTROLLER DESIGN

The system identification process was done using experimental data obtain during test flight via the flight logs, in conjunction with the identification toolbox from Matlab. During flight, the yaw, pitch, and roll angles as well as the angular speeds were recorded. Additionally, all radio commands were also stored, allowing the reproduction of input and output plots of the system. For all data logs, the resolution was configured at 100Hz.

As explained in 6, the control loops work independently of each other with one control loop per axis, with a total of four PID controllers, two in the attitude control loops and two in the navigation control loops. This meant that a total of four SISO systems had to be identified in order to compute the gains of each of the PID controllers.

The four systems were:

- Ailerons → Roll : How the roll of the aircraft responds to aileron servo inputs
- Elevator →Pitch : How the pitch of the aircraft responds to elevator servo input
- Roll closed loop → Bearing: How the heading, or bearing of the aircraft changes with the bank or roll of the aircraft
- Pitch closed loop → Airspeed: How the airspeed of the aircraft changes with the pitch of the aircraft

7.1 AILERONS → ROLL IDENTIFICATION

Obtaining data to determine the Ailerons →Roll system was a simple fact of flying the aircraft in manual mode, and recording the roll angle and angular rate as well as the radio aileron input.

To facilitate the identification using the toolbox, it was decided to determine the Ailerons → Roll angular rate transfer function, and then add the integrator term manually.

Multiple manual flight tests were done, and here was the data segment that was used to identify the system:

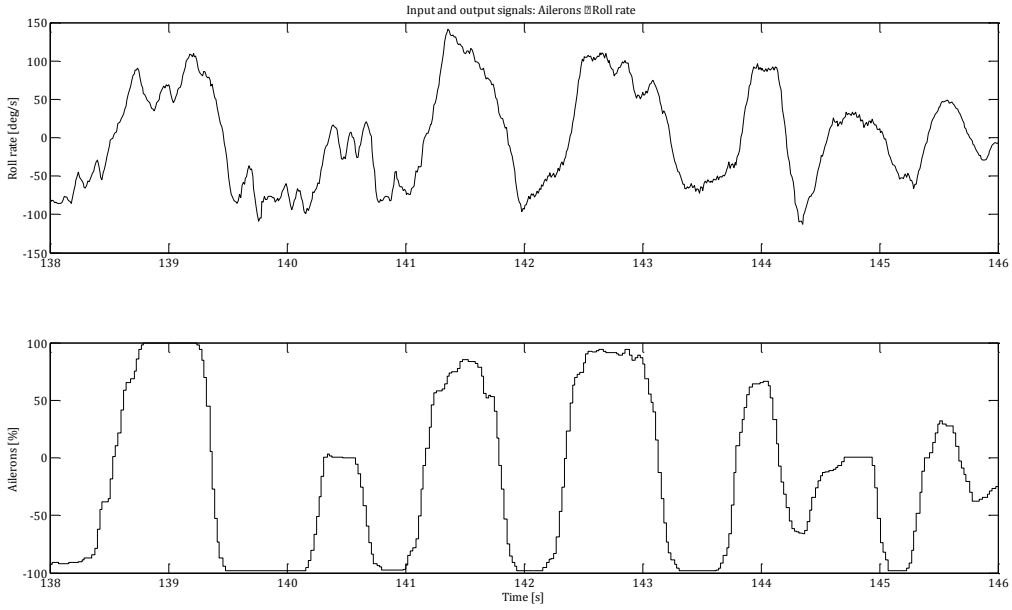


Figure 39: Ailerons → Roll rate data segment

The transfer function obtained was:

$$\frac{Roll\ Rate}{Ailerons} = 0.46418 \cdot \frac{1 + 0.63894s}{(1 + 0.19328s)(1 + 0.21571s)} \left[\frac{\frac{deg}{s}}{\%}\right] (7.1)$$

Here is the simulation result performed using another data segment:

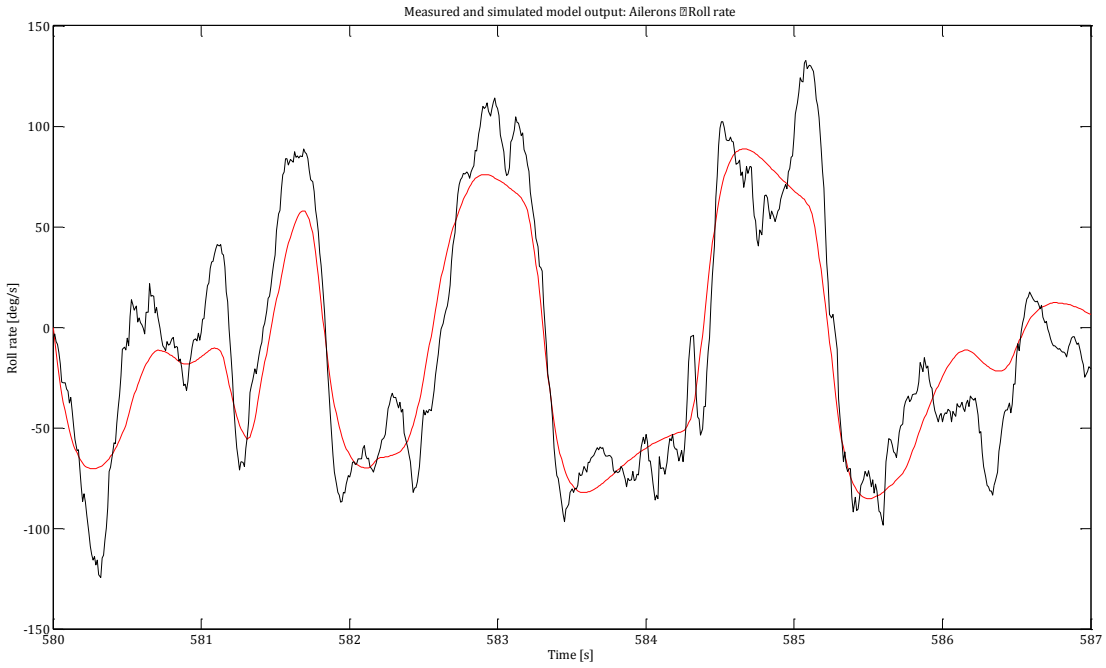


Figure 40: Ailerons→ Roll rate Simulation

By adding an integrator term, we obtain the Ailerons → Roll transfer function which was what we are looking for:

$$\frac{Roll}{Ailerons} = 0.46418 \cdot \frac{1 + 0.63894s}{s(1 + 0.19328s)(1 + 0.21571s)} \left[\frac{deg}{\%}\right] \quad (7.2)$$

7.2 ELEVATOR →PITCH IDENTIFICATION

The Elevator → Pitch transfer function was obtained exactly the same way as the Ailerons → Roll transfer function, using manual flight data logs, and using gyroscope data and adding an integrator term by hand rather than using Pitch data. Here is the data segment that was used to identify the system:

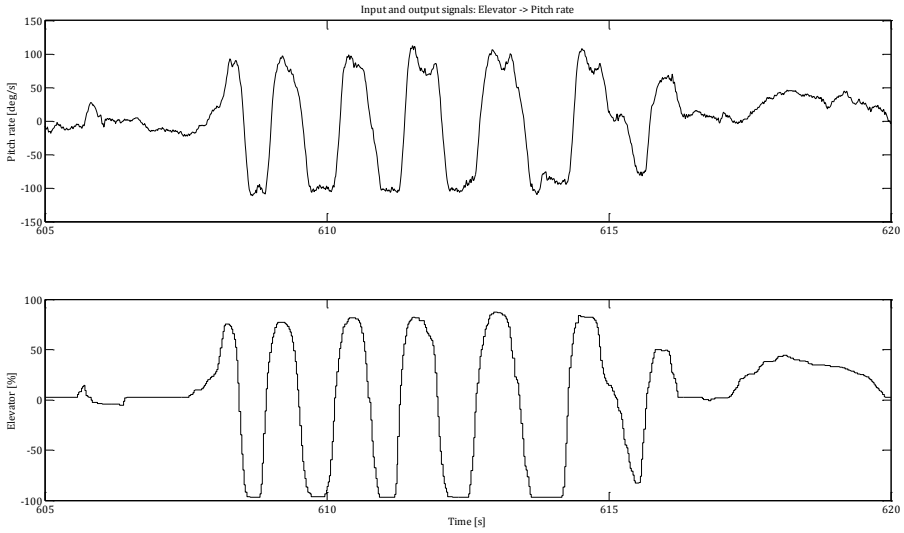


Figure 41: Elevator → Pitch data segment

The transfer function computed by Matlab is:

$$\frac{Pitch\ Rate}{Elevator} = 1.0129 \cdot \frac{1 + 0.14418s}{(1 + 0.11246s)(1 + 0.22297s)} \left[\frac{\frac{deg}{s}}{\%}\right] \quad (7.3)$$

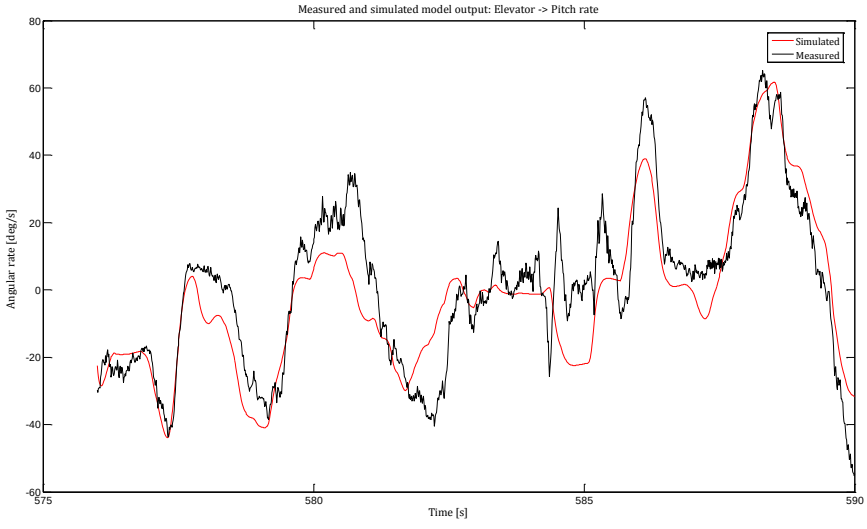


Figure 42: Elevator → Pitch rate Simulation

By adding an integrator term, we obtain the Elevator → Pitch transfer function:

$$\frac{Pitch}{Elevator} = 1.0129 \cdot \frac{1 + 0.14418s}{s(1 + 0.11246s)(1 + 0.22297s)} \left[\frac{deg}{\%}\right] \quad (7.4)$$

7.3 ATTITUDE PID GAINS

Now that we have the two attitude SISO transfer functions in continuous time, we can calculate the gains of a discrete PID controller.

The first step was to convert our continuous time transfer functions to discrete z-space transfer functions using the c2d(...) function from Matlab. The attitude control loop work at an arbitrary user chosen frequency. For these calculations a frequency of 50Hz was chosen as it is the maximum servo refresh frequency.

After converting to z-space we could apply standard PID tuning techniques.

7.3.1 AILERON PID GAINS

The computed z-space discrete transfer function is:

$$g_{arz} = \frac{0.0013472\ (z + 0.9465)\ (z - 0.9692)}{(z - 1)\ (z - 0.9115)\ (z - 0.9017)} \quad (7.5)$$

If the aircraft is balance correctly, it should not be under the influence of a constant moment on the roll axis. Additionally, having small steady state errors are acceptable as long as they're only apparent when the aircraft is turning, and not when trying to achieve level flight (roll setpoint = 0). For these reasons, a simple P controller was initially calculated and tested, which turned out to perform remarkably well.

The Kp gain was calculated using the Matlab sisotool and automated PID tuning. The Kp was calculated at 8.2683, producing the following closed loop step response:

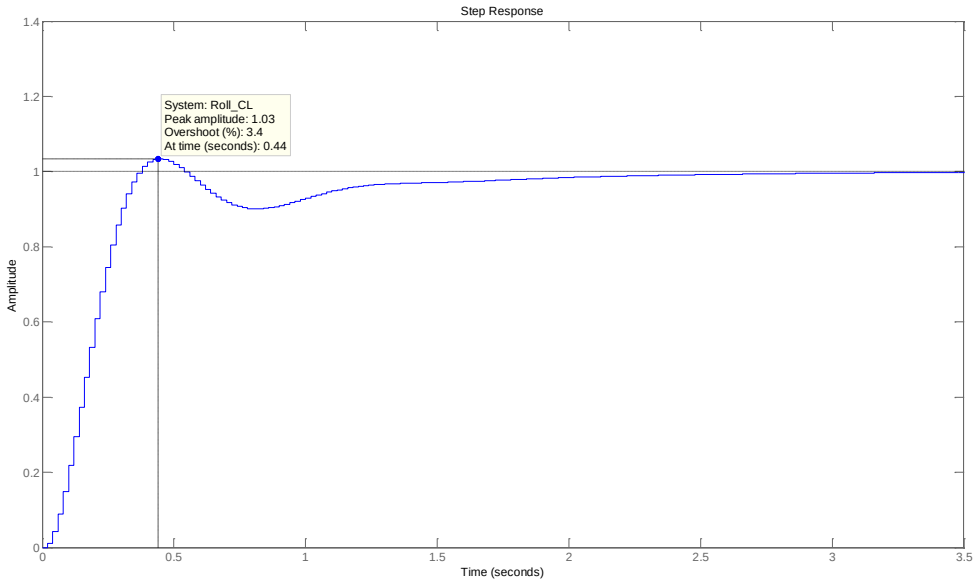


Figure 43: Aileron PID step response

7.3.1 ELEVATOR PID GAINS

The computed z-space discrete transfer function is:

$$g_{epz} = \frac{0.0025856\ (z + 0.9272)\ (z - 0.9142)}{(z - 1)\ (z - 0.8705)\ (z - 0.8371)} \quad (7.6)$$

If we refer back to 5, we know that unless the aircraft is trimmed correctly for a certain cruise speed, it will either want to pitch up or down. Changes in the center of gravity also change the balance of the aircraft and change the trim required. This tendency to pitch up or down translates into a constant moment that is always applied to the aircraft (unless trimmed, or flying at very specific airspeed) causing a steady-state error.

This problem is solved by using a PI controller as it corrects against steady-state errors. Essentially, the integral term of the PI controller acts as the aircraft trim, adding an offset to the elevator output so that it flies regardless of the airspeed or center of gravity.

A PI controller was calculated using Matlab sisotool GUI and the automated PID tuning. The following PI controller was calculated:

$$grz = \frac{5.5734\ (z - 0.9979)}{(z - 1)} \quad (7.7)$$

The closed loop step response can now be traced:

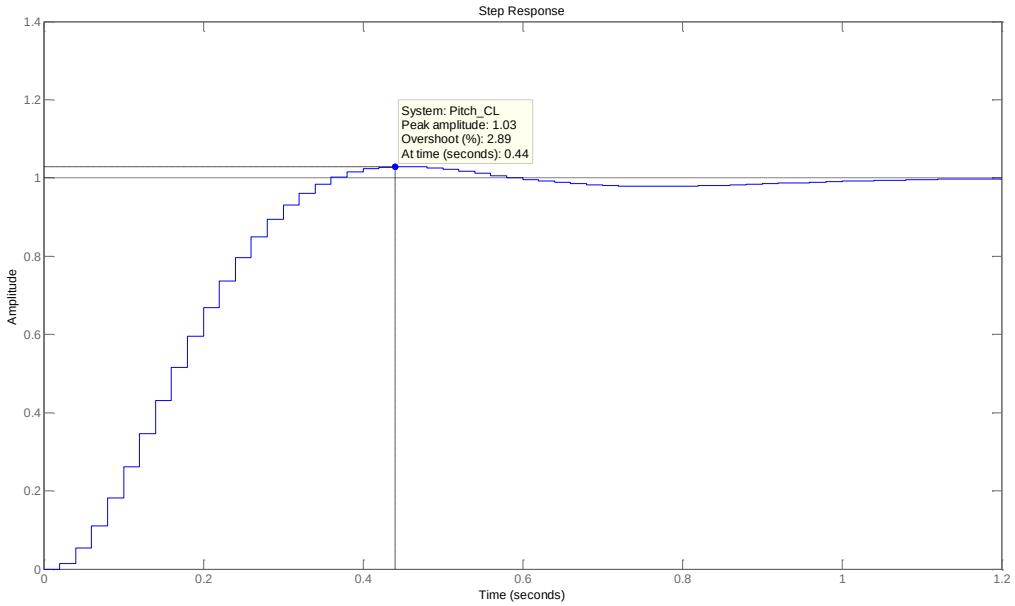


Figure 44: Elevator PID step response

From the PI transfer function the Kp and Ki gains need to be calculated which will be used to configure the autopilot:

$$Kp = 5.5734 \cdot 0.9979 = \ 5.5617$$

$$Ki = 5.5734 \cdot (1 - \ 0.9979) = 0.0117$$

7.4 ROLL CLOSED LOOP → BEARING

With the lack of appropriate data at the time, the initial transfer function was calculated theoretically based on the equation of the rate of turn of an aircraft vs. the roll angle.

The turn radius of an aircraft can be assimilated to a car turning on a slanted circuit. We assume that the aircraft is in a balanced turn meaning the aircraft has no vertical acceleration, and that the speed is constant throughout the turn.

The turn radius is given by:

$$R = \frac{V^2}{g \cdot tan(roll)} \ [m] \quad (7.8)$$

Knowing the radius of the turn and the speed, we can calculate the time it would require to travel a complete circle:

$$T = \frac{2\pi R}{V} \ [s] \quad (7.9)$$

Knowing the time and the radius, we can now easily calculate the turn rate in degrees per second:

$$\omega = \frac{360}{T} = \frac{180 \cdot g \cdot tan(roll)}{V \cdot \pi} \ \left[\frac{deg}{s}\right] \quad (7.10)$$

Now that the theoretical turn rate vs. the roll angle was determined, the next step is obtaining the roll closed loop transfer function. There were two possible ways to approach this problem.

The first method was to use the open loop aileron → roll transfer function and compute the theoretical closed loop function with the PID gains calculated in 7.3.

The second method consisted of using data from fly by wire tests, where the aircraft operated in closed loop mode and use the system identification toolbox as done previously to determine the closed loop transfer function.

The second method was chosen as it was one step closer to reality and would simplify the process as the aileron → roll loop operated at a different frequency than the roll → bearing loop. Using the identification toolbox would supply a continuous transfer function (in s space), that could then be converted to z space with the roll → bearing loop frequency.

Figure 45 shows segment of data, which was obtained during a fly by wire flight test, used to identify the system

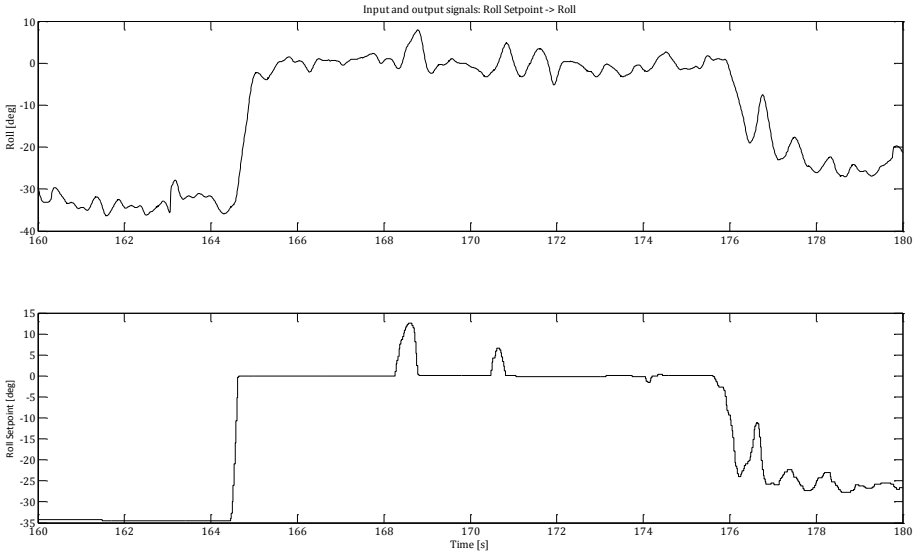


Figure 45: Roll closed loop data segment

Different settings were used concerning the number of poles and zeros. The final result contains two poles and a zero, giving us the following transfer function:

$$\frac{Roll}{Roll\ Setpoint} = 1.0129 \cdot \frac{1 + 0.14418s}{(1 + 0.11246s)(1 + 0.22297s)}\ [-] \quad (7.11)$$

Here is the simulation result performed using another data segment:

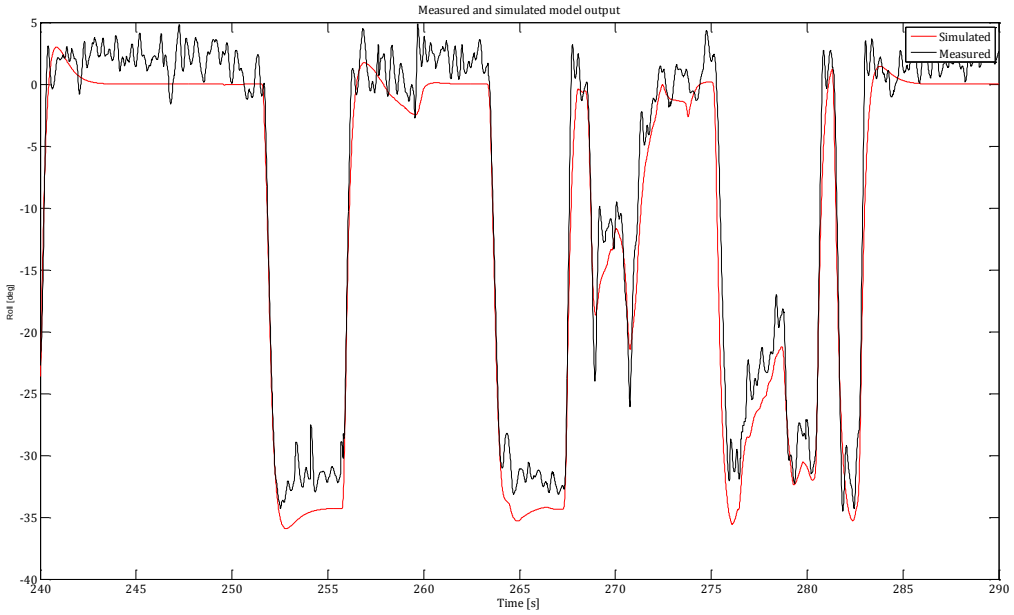


Figure 46: Roll closed loop simulation

We can now calculate the theoretical roll closed loop → bearing transfer function. However, the tan(roll) in equation 7.10 is problematic, and we must use a Taylor series approximation. Both second and third order approximations were tested and gave the same result, so only the second order was retained. We also have to add an integrator term to obtain the bearing from the angular rate.

The final theoretical equation is given by:

$$Bearing = \int \frac{180 \cdot g \cdot (roll + \frac{roll^3}{3})}{V \cdot \pi} \ [deg] \quad (7.12)$$

By replacing the roll variable with the roll closed loop transfer obtained previously, we can calculate the wanted roll closed loop → bearing transfer function:

$$\frac{Bearing}{Roll\ Setpoint} = \frac{89.7262\ (s + 9.497)\ (s + 9.666)\ (s^2 + 17.8s + 79.66)\ (s^2 + 20.53s + 105.8)}{s\ (s + 9.666)^4 (s + 9.497)^4} \ [-] \quad (7.13)$$

This theoretical model was used to calculate the initial PID gains, which were then used for the final flight test in autopilot mode.

However, after performing the final autopilot flight test, enough data was obtained to directly identify the roll closed loop → bearing transfer function with the identification tool box.

Do note that the bearing angle is the absolute unwrapped angle (-670 degrees = -1 complete turn and -310 degrees), which was required to have a continuous curve for the system identification. Here is the data used to identify the system:

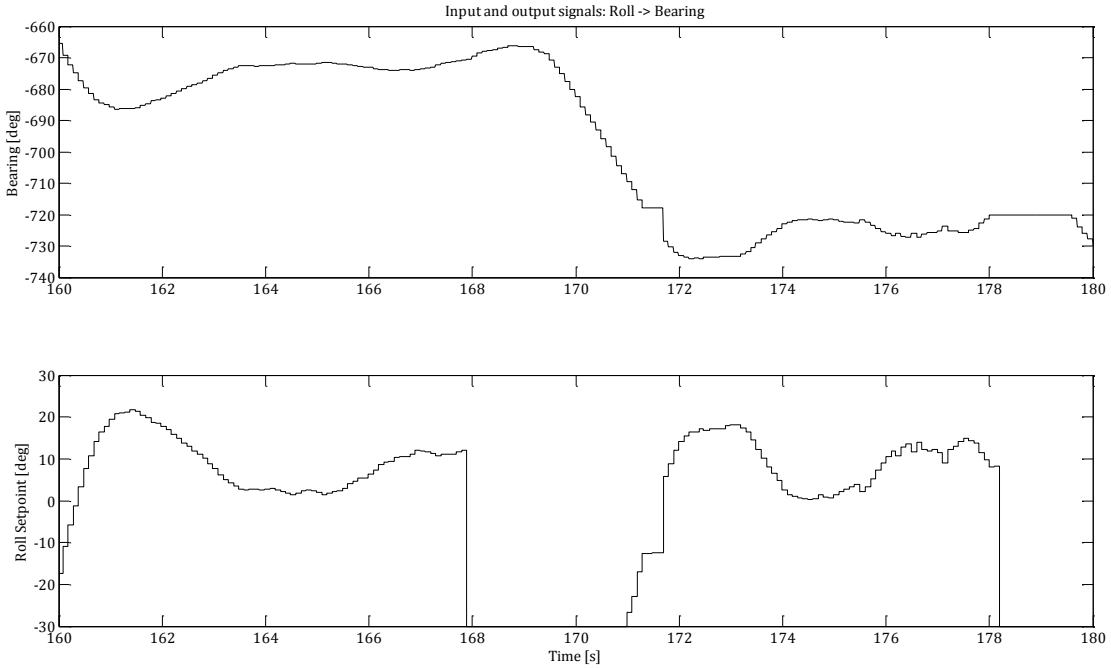


Figure 47: Roll → Bearing data segment

The result obtained is a third order transfer function with an integrator term and a zero:

$$\frac{Bearing}{Roll\ Setpoint} = -2372.6 \cdot \frac{1 - 141.74s}{s(1 + 1.0755s)(1 + 5.947 \cdot 10^5 s)}\ [-] \quad (7.14)$$

Here is the simulation result performed using another data segment:

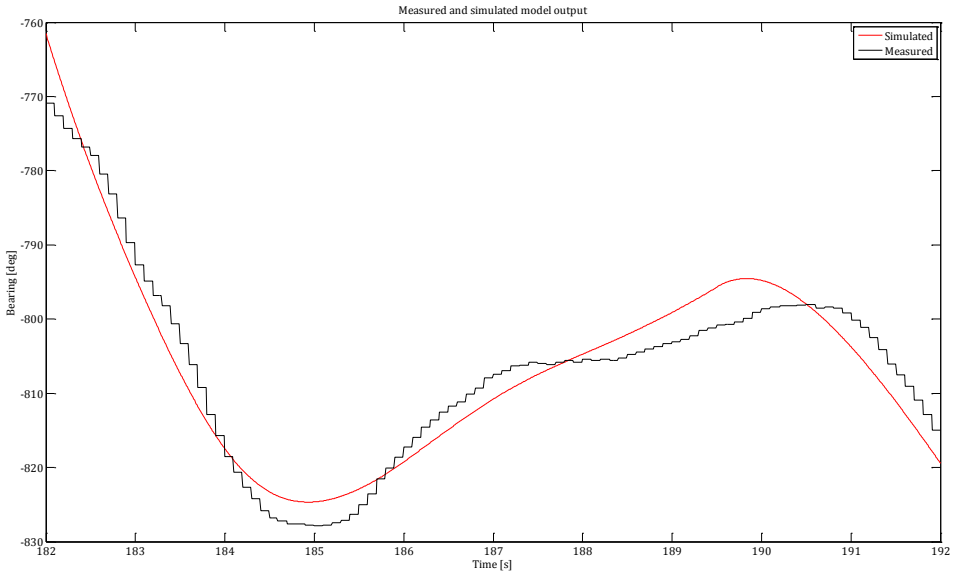


Figure 48: Roll → Bearing simulation

7.5 PITCH CLOSED LOOP → AIRSPEED

Obtaining acceptable results for the Pitch closed loop → Airspeed transfer function proved to be more difficult than the rest. The airspeed measurement is easily disturbed by turbulence and wind. In addition, the only segments of data that can be used are parts where the throttle is cut off, because we are interested in the transfer function when the aircraft is gliding, as that will be its main point of operation (only using the throttle as a safety net).

Doing so, without considering the effects of the throttle means that the throttle will act as an external perturbation when used, and it will be up to the controller to respond correctly.

The most optimal way of obtaining the Pitch closed loop → Airspeed transfer function would be to directly calculate it using data logged while the aircraft was being flown in Fly By Wire with the throttle cut off. Unfortunately, logs when the aircraft flew in FBW mode with the throttle cut off were scarce, and the few logs that did meet the criteria produced erratic results when using the identification tool box. There were only two data segments that could produce realistic results, but they were obtained when the aircraft was in manual (pitch and roll open loop) mode.

The identification was then split in two parts, starting by identifying the roll in closed loop using FBW data logs, and then the Pitch open loop → Airspeed using the two data segments. By putting the two transfer functions in series we would then obtain the Pitch closed loop → Airspeed transfer function.

For the pitch closed loop transfer function. The following data was used:

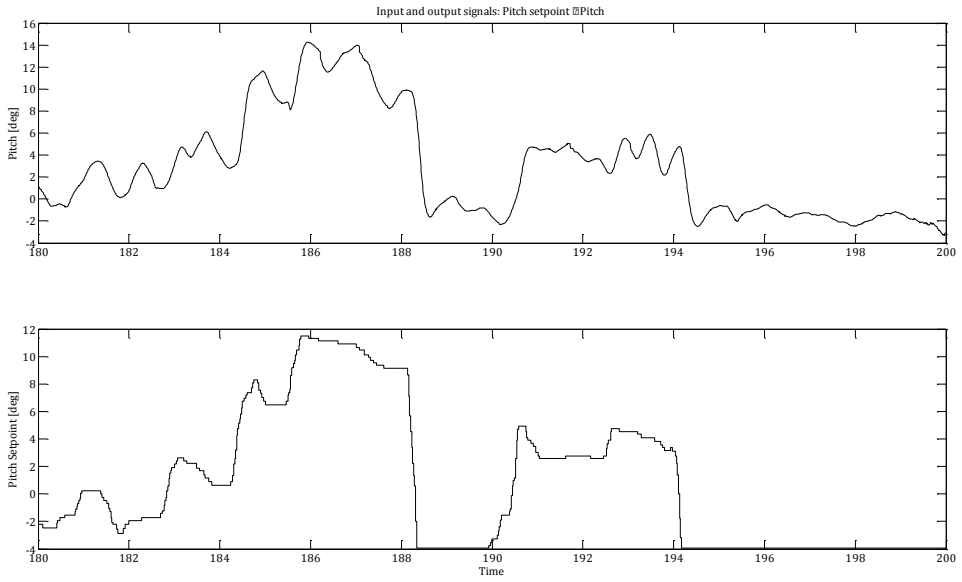


Figure 49: Pitch closed loop data segment

The transfer function computed by Matlab is:

$$\frac{Pitch}{Pitch\ Setpoint} = 0.99644 \cdot \frac{1}{(1 + 0.1053s)(1 + 0.10346s)}\ [-] \quad (7.15)$$

Here is the simulation result performed using another data segment:

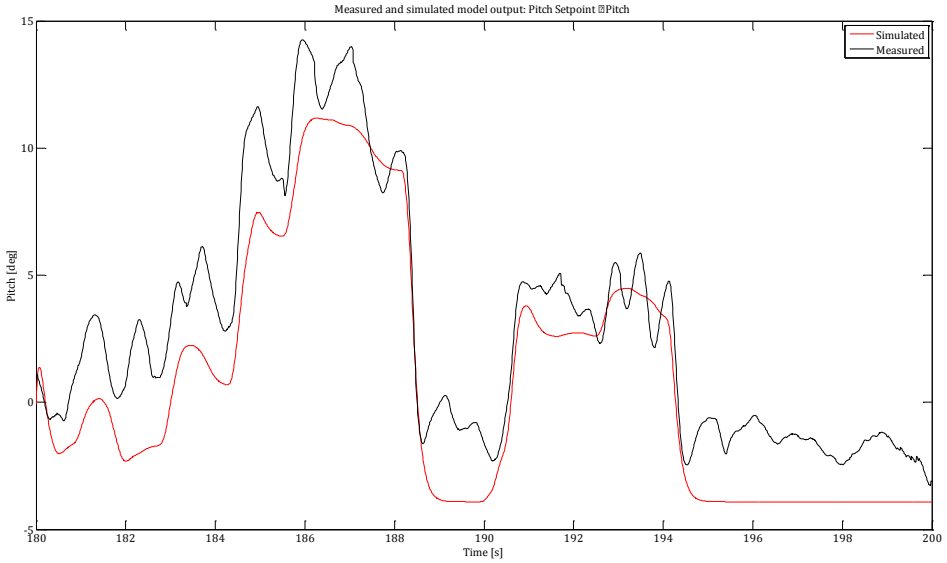


Figure 50: Pitch closed loop simulation

Now that the pitch closed loop has been identified experimentally, we move on to the Pitch open loop → Airspeed transfer function. Here is the data used to identify the system:

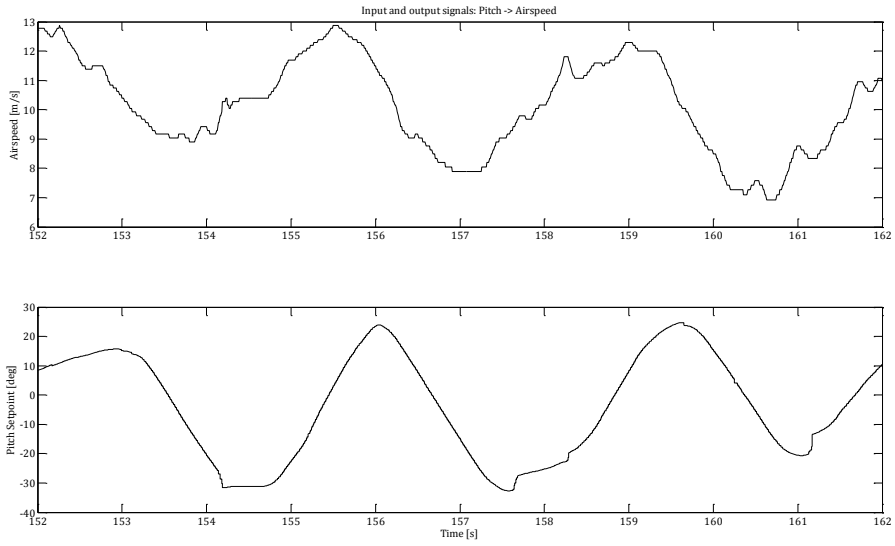


Figure 51: Pitch → Airspeed data segment

The result obtained is a second order transfer function:

$$\frac{Airspeed}{Pitch} = -693.71 \cdot \frac{1 + 15.092s}{(1 + 4286.3s)(1 + 17.076s)} \left[\frac{\frac{m}{s}}{deg}\right] \quad (7.16)$$

Here is the simulation result performed using another data segment:

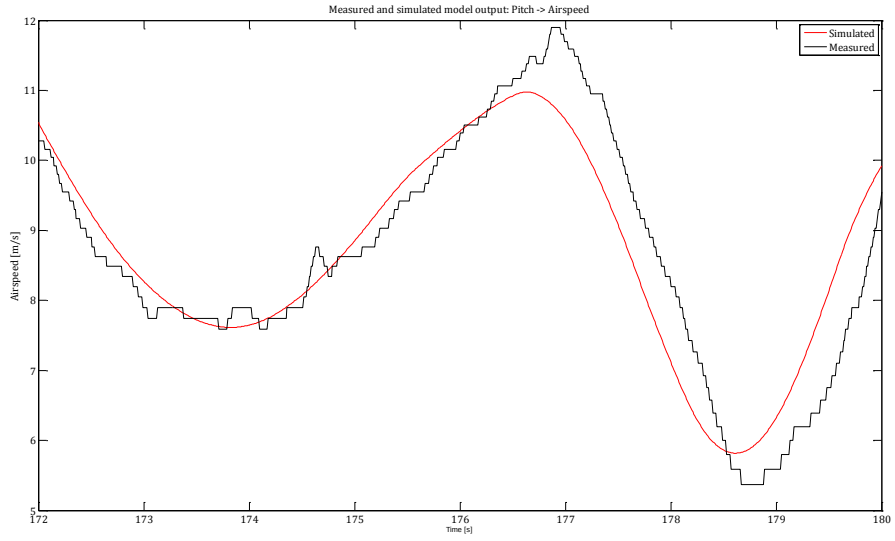


Figure 52: Pitch → Airspeed Simulation

What we noticed from the data simulation results are that the simulated curve reacts faster than the measurement. This means that the model created by Matlab is faster and more reactive than the real world, and must be considered when calculating the PID gains.

By adding the transfer functions in series we obtain:

$$\frac{Airspeed}{Pitch\ Setpoint} = -691.2\frac{1 + 15.092s}{(1 + 4286.3s)(1 + 17.076s)(1 + 0.1053s)(1 + 0.10346s)}\left[\frac{\frac{m}{s}}{deg}\right] \quad (7.17)$$

7.6 NAVIGATION PID GAINS

We repeat the same process as for the attitude gains, starting by converting the continuous s-space transfer functions to discrete z-space transfer functions.

Due to the fact that that the bearing is measured by the GPS, the roll control loop can only run at 10Hz which is the refresh rate of the GPS.

For the airspeed we can select any frequency, but since the airspeed measurement is filtered by a low pass with a 5 Hz cutoff frequency, a frequency of 20 Hz was chosen.

7.6.1 ROLL PID GAINS

As stated previously, two methods were used to determine the transfer function, the theoretical and experimental.

The theoretical method was used to calculate the initial gains that were used on the final autopilot test flight. Following that flight, a new gain was calculated using newly obtained flight data.

We will start with the theoretical transfer function. The computed z-space discrete transfer function is:

$$g_{rbz} = \frac{0.0094766\ (z + 2.373)\ (z - 0.3869)\ (z - 0.3804)\ (z + 0.1616)\ (z^2 - 0.7151z + 0.1284)\ (z^2 - 0.8195z + 0.1686)}{(z - 1)\ (z - 0.3869)^4\ (z - 0.3804)^4} \quad (7.18)$$

The transfer function possesses an integration term (Z-1) and eight very small poles. This meant that no poles need to be simplified by the PID controller, and a simple P controller will suffice.

To determine the Kp gain, a root locus plot was traced:

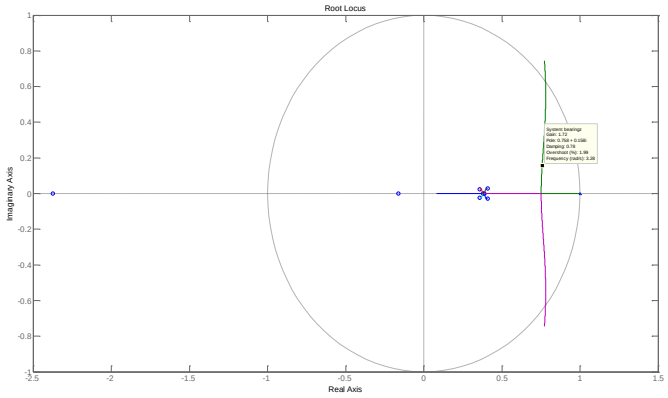


Figure 53: Roll Root Locus 1

We obtain a gain of 1.72 giving us an overshoot of 2.07%. Using the newly calculated Kp gain, the closed loop step response can now be traced:

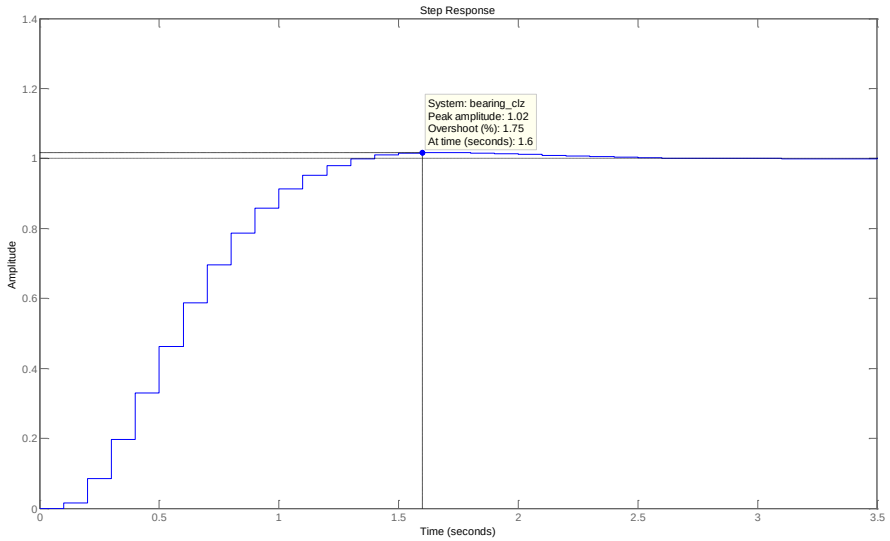


Figure 54: Roll PID step response 1

The same method was repeated with the experimental transfer function. The computed z-space discrete transfer function is:

$$g_{rbz} = \frac{0.0025438\ (z + 0.9693)\ (z - 1.001)}{(z-1)^2\ (z - 0.9112)} \quad (7.19)$$

The transfer function has a single integration (with a zero almost simplifying the other z-1), and a pole in 0.9112 that could be simplified.

However, when trying to simplify the pole with a PI the system turns unstable. When tried with a PD controller, the calculated Kp gain obtained is around 50 for a 5% overshoot, which seems extremely high considering that a previous flight test was achieved with good results using the initial P controller with a gain of 1.72, additionally using the derivative term is to be avoided as it is sensitive to noise.

Based on the fact that the first autopilot flight worked very well using a simple P controller and only had a mild overshoot, a third P regulator was finally calculated using the root locus plot, shown in Figure 55

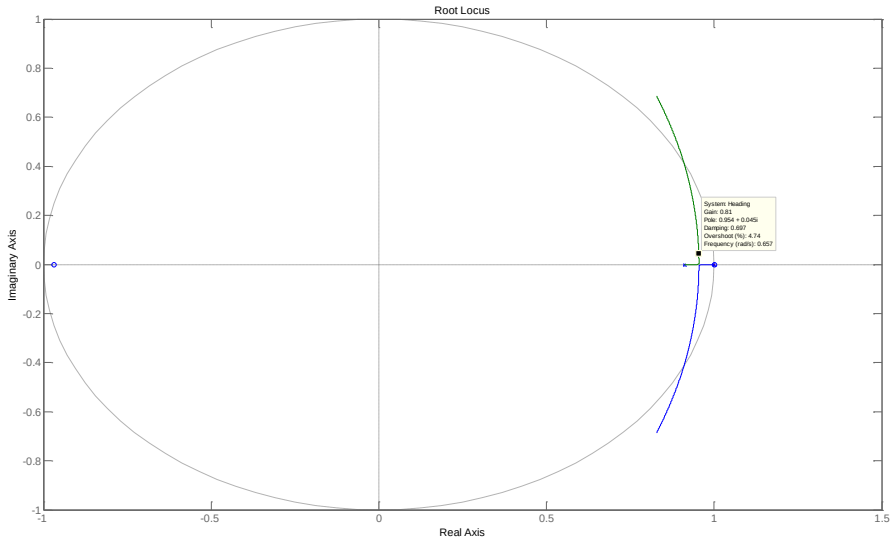


Figure 55: Roll Root locus 2

A gain of 0.81 giving us an overshoot of 4.74% was obtained. Considering the flight result with the previous gain of 1.72 (see 10) performed adequately (albeit with an overshoot), this new gain seems entirely plausible but still remains to be tested.

As previously, we can now trace the closed loop step response:

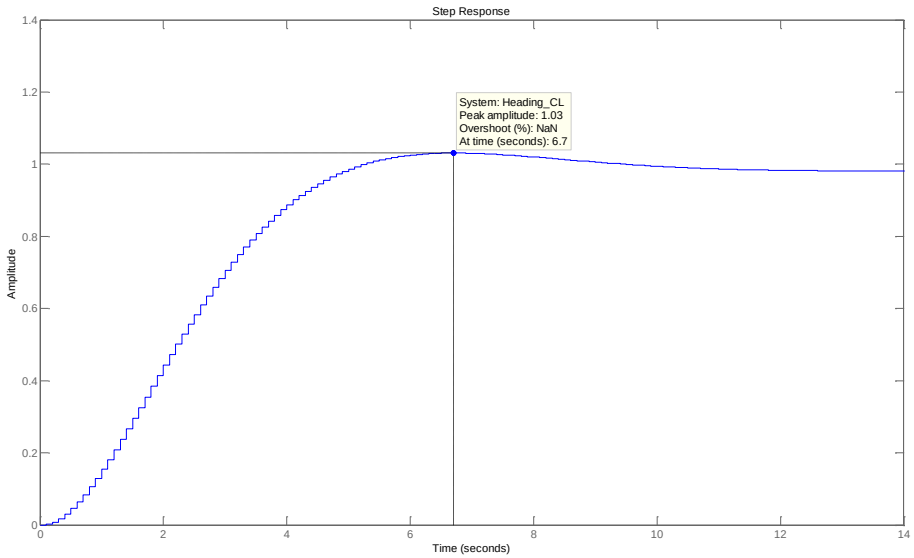


Figure 56: Roll PID step response 2

7.6.2 PITCH PID GAINS

The computed z-space discrete transfer function is:

$$g_{paz} = \frac{-0.00021576\ (z + 2.957)\ (z - 0.9967)\ (z + 0.2095)}{(z - 1)\ (z - 0.9971)\ (z - 0.622)\ (z - 0.6168)} \quad (7.20)$$

The system has an integrator term and, two small poles that cannot be simplified (0.622 and 0.6168). The third pole at 0.9971 is currently simplified by the zero in 0.9967. This means that a P controller would satisfy our requirements.

The gain is calculated using a root locus plot:

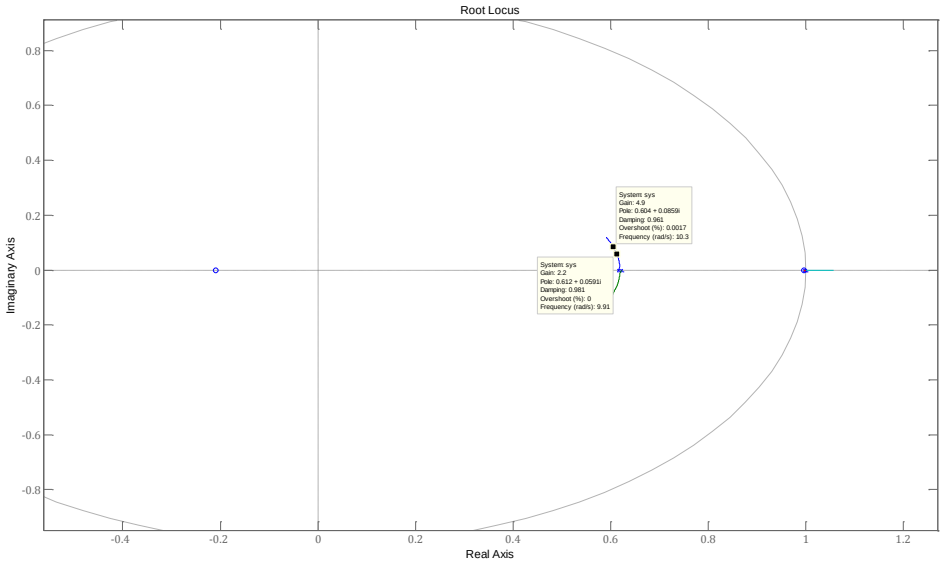


Figure 57: Pitch Root locus

Due to the fact that the pole is simplified by a close zero, the gain can theoretically reach as high as 20-30 with a 5% overshoot. However, if in reality the zero or pole shifts, and they no longer simplify each other, the system may become unstable. Additionally, a gain of 20-30 seems very high intuitively (a gain of 30 would translate as an increase of 30 degrees of pitch for an error of 1m/s). For these reasons, a much smaller gain that provides practically no overshoot was chosen.

The initial gain that was used for the first autopilot flight test was 2.0. After the successful autopilot flight, a gain of 5 was tested. The gain of 5 would cause some oscillations whenever a small gust of wind or turbulence would disturb the aircraft, and the gain was brought back down to 2.

7.7 SAFETY THROTTLE CURVE

Although the autopilot was designed to work with a glider, meaning the aircraft has no controlled propulsion system, if the aircraft is mounted with a motor it should be able to use it as a security option should the aircraft drop too low.

The throttle control is not done using a conventional control loop, but using a simple throttle curve that controls the throttle based on two variables: the minimum altitude, and an altitude threshold.

The minimum altitude is the altitude that the aircraft must absolutely not drop below. The throttle threshold is a distance above the minimum altitude under which the throttle will start to kick in.

When the aircraft is above the minimum altitude + throttle threshold, the throttle is cut off and the aircraft is gliding. When the aircraft is equal or below the minimum altitude, the throttle is set to 100%. When the aircraft is situated between these two limits, the throttle follows a simple linear curve, shown in Figure 58.

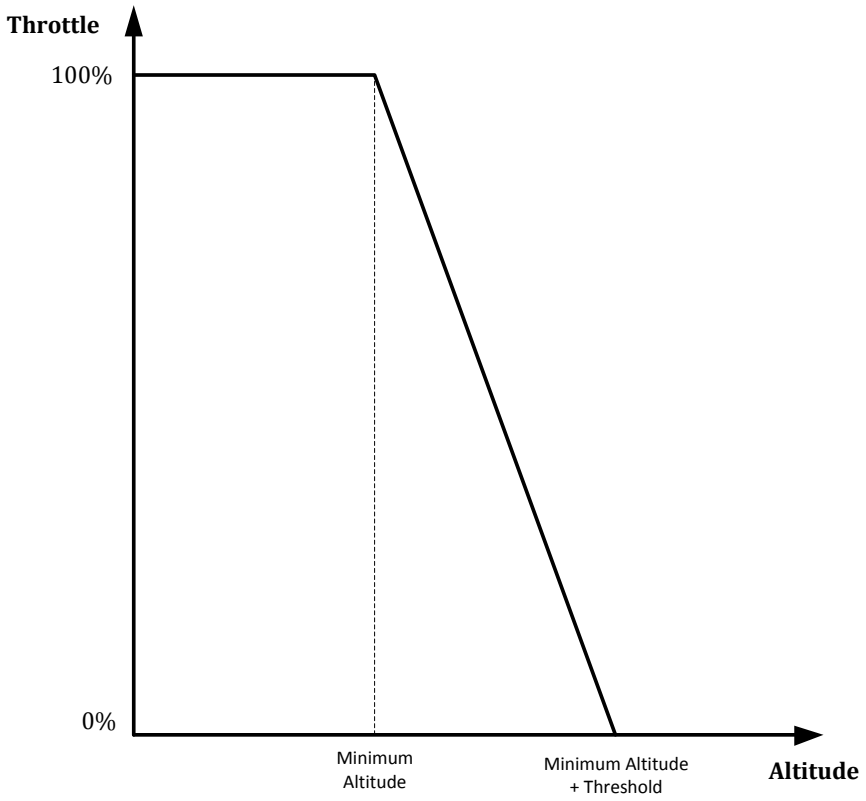


Figure 58: Throttle Curve

The curve can be express analytically with a simple line equation, with two unknowns: the slop and the offset.

$$Throttle = a \cdot Altitude + b \quad (7.21)$$

The slope of the line is easily determined and is given by:

$$a = -\frac{100\%}{Threshold} \quad (7.22)$$

The offset is calculated by using a known data point of the curve and solving for b:

$$100\% = -\frac{100\%}{Threshold} \cdot Minimum\ Altitude + b \ \rightarrow b = 100\% + \frac{100\%}{Threshold} \cdot Minimum\ Altitude \quad (7.23)$$

We then obtain the final equation:

$$Throttle = \frac{100\%}{Threshold} \cdot [Minimum\ Altitude - Altitude] + 100\% \ \left[\frac{\%}{m}\right] \quad (7.24)$$

When adding or increasing the throttle the aircraft gains airspeed and since the aircraft's airspeed is regulated with the pitch controller, the aircraft will pitch up in attempt to keep the airspeed constant at the configured cruise speed. By increasing the pitch, the aircraft will then gain altitude which was the purpose of increasing the throttle in the first place. Although the altitude is not directly controlled using the pitch of the aircraft, the end result of increasing the aircraft's altitude is still obtained.

In the event that the autopilot is not used in a glider, but a motorized aircraft instead, the throttle safety curve will act as the throttle and altitude regulator. It will be up to the end user to choose a large enough throttle threshold that allows the aircraft to find equilibrium altitude, where the aircraft remains level with a certain amount of throttle.

This allows the autopilot to be used in a normal motorized aircraft even though it was programmed to be used with a glider. However, it is not the optimal solution as the aircraft's altitude is not properly regulated; only the fact that the aircraft will fly above the minimum altitude can be guaranteed. The actual altitude at which the aircraft will fly depends on the equilibrium point that is specific to each aircraft as well as the cruise speed configured by the user.

8. EMBEDDED PROGRAMMING

As stated previously, all low level functions, as well as a RTOS were implemented previously during the semester project. This greatly simplified the embedded programming portion of this project as only a few of high level tasks had to be implemented.

The program makes use of two interrupt routines, one for the GPS USART and another for the radio USART. The GPS USART interrupt routine simply stores the received bytes in a circle buffer and transmits bytes that are in the output circle buffer, emulating a DMA. This was done because the DMA channel for the GPS USART was already used for the SD card SPI, which is much more data intensive. The radio USART routine reads and parses the data directly in an interrupt function, making sure the user always has command of the aircraft even directly after a reset.

In addition to the high level tasks, some new functions had to be created that are used by the attitude and navigate control tasks, such as calculating the bearing between two waypoints, or implementing the PID function.

The navigation and control functions are located in navigate.c and control.c files respectively.

8.1 NAVIGATION FUNCTIONS

There are three functions used for the navigation task, they are:

- Get_Bearing – Calculates the bearing between two waypoints
- Get_Bearing_Error – Calculates the shortest bearing error between two bearings
- Get_Distance_2D calculates the distance in meters between two waypoints]

Calculating the distance and bearing on an elliptical earth requires the use complicate equations, such as the haversine formula for calculating the distance. However, these accurate equations are calculation intensive, and when working on an embedded microcontroller with software floating point they are best avoided.

When working with small distances, we can use an equirectangular projection of the earth, and use equations such as Pythagoras theorem for calculating the distance, and a simple arctangent for the bearing.

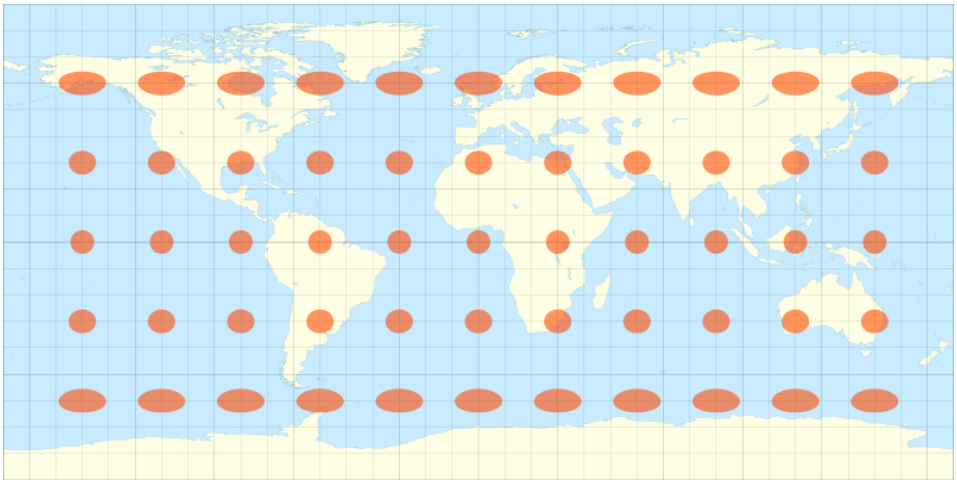


Figure 59: Equirectangular projection

The get distance function takes in two locations (1 and 2) defined by their latitude and longitude. It then calculates the latitude and longitude variation, ΔLat and ΔLong. The ΔLong term is multiplied by a corrective term defined by the equirectangular project. It is multiplied by the cosine of the average latitude angle between both waypoints.

$$\Delta Lat = (Latitude2 - Latitude1) \quad (8.1)$$

$$\Delta Long = (Latitude2 - Latitude1) \cdot cos\left(\frac{latitude1 + latitude2}{2}\right) \quad (8.2)$$

After obtaining the latitude and scaled longitude variation, it is a simple matter of applying Pythagoras theorem. This returns the distance in degrees, but we are interested in the distance given in meters. Knowing that the average circumference the earth is 40075.16 [km], we divide the circumference by 360 degrees, and multiply the result to our distance in degrees:

$$Distance = \sqrt{\Delta Long^2 + \Delta Lat^2} \cdot \frac{40075160}{360} [m] \quad (8.3)$$

The get bearing function works the same way. It then calculates the latitude and longitude variation, ΔLat and ΔLong. It then applies the equirectangular project scaling on the ΔLat by dividing the result by the cosine of the average latitude between both waypoints.

$$\Delta Lat = \frac{(Latitude2 - Latitude1)}{cos\left(\frac{latitude1 + latitude2}{2}\right)} \quad (8.4)$$

$$\Delta Long = (Longitude2 - Longitude1) \quad (8.5)$$

The bearing is then calculated by applying the arctangent of the x (latitude) and y (variation). The result is then converted to degrees. The angle calculated is using the normal trigonometry angles with 0 degrees equal to east, while we want 0 degrees to point to north. So we need to add 90 degrees to the result.

$$Bearing = 90 + atan2(-\Delta Latitude, \Delta Longitude) * 57.2957795 \, [deg] \quad (8.6)$$

Due to the circular nature of degrees there are always two ways to go from one bearing to another. An extra function called get bearing error was added that takes two bearing angles, and calculates the shortest angle between the two. It starts by simply subtracting both angles, and checks whether or not it is larger than 180 (clockwise rotation) or smaller than -180 (counterclockwise rotation). If that is the case, it means that the subtraction result provided the longer way, and must be correct by either adding or subtracting 360 degrees.

If the bearing error is larger than 180 degrees, 360 degrees are subtracted from the result, and if the error is less than -180 degrees, 360 degrees are added to the results.

8.2 CONTROL LOOP FUNCTIONS

The control loops functions consist of the PID function that works with a structure grouping the PID gains, saturation limits, as well as the controller states. A second function was created to initialize the PID controller states. The PID structure contains the Kp, Ki, and Kd gains, as well as the upper and lower output saturation limits. It also contains two state variables X_R and e_1 used in the PID algorithm.

The following PID algorithm was used, with the error 'e' as an input:

$U_{id} = X_R + (K\text{p} + K\text{i} + K\text{d})e - K\text{d}\, e_1$ - Calculate ideal output without saturation.

If U_{id} > Saturation limits - Check for either upper or lower saturation
    U_{cm} = Saturated U_{id}
Else
    $U_{cm} = U_{id}$
End

$e_{lim} = \text{e} - (u\text{id} - u\text{cm}) / (K\text{p} + K\text{i} + K\text{d})$ - calculate antiwindup term
$X_R = X_R + K\text{i}\, e_{lim}$ - update previous integral term
$e_1 = \text{e}$ - update previous error

Return U_{cm} - Return PID output result

When calling the PID functions, the user enters the error input, and a pointer to the specific PID structure.

8.1 HIGH LEVEL TASKS

The entire embedded program runs within ten independents tasks. Data is shared between tasks by using global variables. The priority of each task was chosen as to make sure that variables are not also written by one task when being used by another. A higher priority task will always run before a low priority task until it finishes or yields.

All the tasks run periodically except for the GPS initialize task that is only ran once upon startup. The frequencies of the different task are for the most part fixed, except for the attitude controller task and record data task. These two tasks have variable frequencies which are defined by the user using a configuration register.

Here are all the tasks running the autopilot and their priority levels (higher number translates into higher priority):

Task	Priority	Frequency [Hz]
Attitude Controller Task	8	User Defined
Read Parse Buffers Task	7	400
Navigate Task	6	10
Airspeed Task	6	20
Pressure Data Task	5	100
Record Data Task	4	User Defined
Radio Timeout Task	3	10
Mission Control Task	2	100
GPS Initialize Task	1	-
Flight Mode Task	1	20

Table 2: Task priorities and frequencies

Before launching the task scheduler, all peripheral are initialized, the SD card is mounted so that the program can create, read, and write to files. Lastly all configuration registers are loaded from the external EEPROM.

In addition to these fixed tasks, there is a Cycle Register Read task that reads a choice of three registers periodically. This task is deleted if the user selects no registers, and is recreated if the user selects at least one register to read cyclically. These cyclically read registers are selected using the "Cycle Read Registers" register defined in 9.1.7.

8.1.1 Attitude Controller Task

The Attitude Control task is the highest priority task. This was chosen so that the attitude data would remain constant throughout the control task without the risk of the global variable, containing the AHRS data, changing during the PID calculations.

When the task is initially started the PID structures are initialized with the PID gains in the user register, as well as the saturation levels defined by the servo end points user register.

Every tick, the task checks in what mode the autopilot is working it, and if it's in either FBW or autopilot mode, its computes and sets the new servo position. When the aircraft is in manual mode it jumps down to the task delay, where it will be called X [ms] later, where X depends on the user defined control loop frequency.

Additionally, when changing back to manual mode after operating in FBW or Auto, and then going back to FBW or Auto, the PID structures are reinitialized in order to reset any possible offsets due to the integration term of the PID controllers. This is done using a simple variable that compares the current mode and the previous mode, and initializes the structure accordingly.

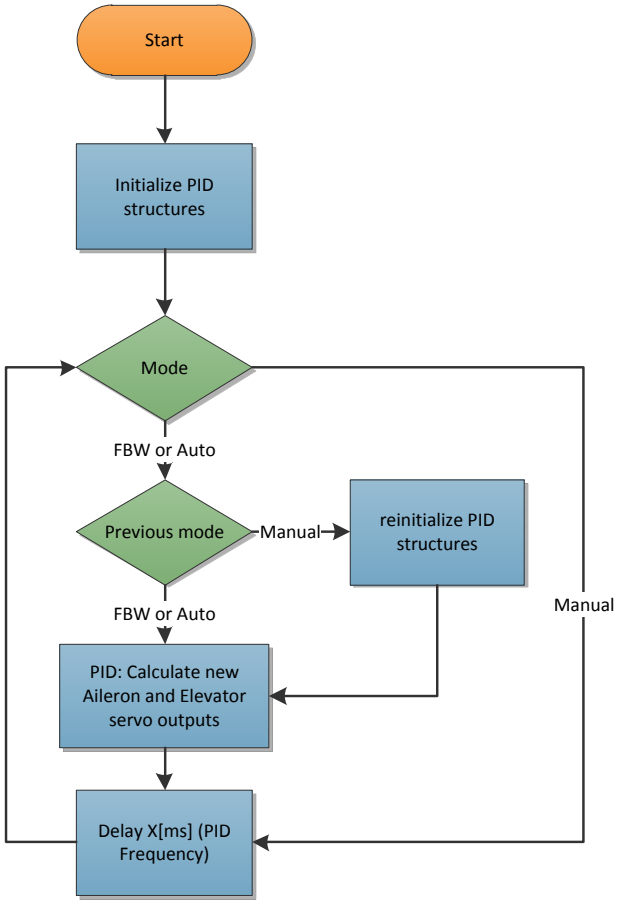


Figure 60: Attitude controller task

8.1.2 READ AND PARSE BUFFERS TASK

Data from both the AHRS and GPS are sent using a standard USART asynchronous serial bus. All bytes received are stored in a circular buffer using a DMA, and the buffers must be read and parsed from time to time. The Read and Parse Buffer task is in charge of reading and parsing the AHRS and GPS buffers as well as synchronizing the navigate task when new GPS data is received.

The task runs at a high frequency of 400Hz as to operate at a higher frequency than the maximum AHRS data rate of 200Hz. A message from the AHRS is defined by a string that starts with a '$' and ends with the newline character '\n'. As the GPS is operating in binary mode, a message is defined by its length of 37 bytes and its two first synchronization bytes (0xD0D0). At every tick, the AHRS and GPS buffers are read and checked if a complete message was received. If that is the case, the message is then parse to extract the data contained within the ASCII, or binary strings and then stored in global structure variables. When a complete GPS message is received, the navigate task is unblocked using a semaphore.

Once completed, the task is delayed for 2.5 [ms]. If a full string was neither received from the GPS nor the AHRS, the task jumps to the end and is delayed.

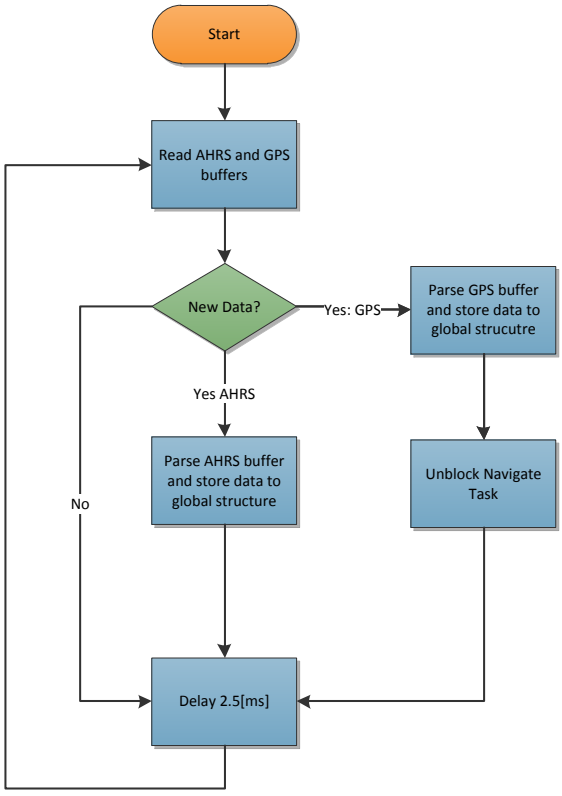


Figure 61: Read Parse Buffers Task

8.1.3 NAVIGATE TASK

The navigate task contains the Roll → Bearing control loop as well as the throttle control. The task uses a semaphore to synchronize itself with the Read Parse Buffer task so that it only runs when new GPS data is presented. The task begins by checking the flight mode of the autopilot. For testing purposes, the autopilot has two auto modes: Auto_A and Auto_B. In Auto_A, the throttle is not controlled by the autopilot but by user on the ground using the radio. In the second Auto_B mode, the entire aircraft, as well as the throttle, is controlled by the autopilot and the user has no effect

what so ever on the aircraft (he can only retake command by going in manual mode using a switch on the radio). These two modes allowed the throttle curve and the Roll → Bearing control loop to be tested independently in order to reduce risks of crashing during the first flight tests.

Each waypoint is described by the following information: waypoint number, latitude, longitude, minimum altitude, radius, and next waypoint. Since the next waypoint is actually defined in the current waypoint, it allows the user to create infinite loops or circuits, by setting the next waypoint to a previous waypoint. The user can use up to 32 waypoints. The waypoint limit was fixed at 32 to simplify the storing and reading of the waypoints in the EEPROM memory blocks.

The task begins by checking the flight mode. If it is in either Auto_A or Auto_B mode, it calculates the distance to the next waypoint. If the distance is equal or smaller than the radius defined for that waypoint, the current waypoint number is set to the next waypoint.

After checking the distance to the waypoint, and changing the waypoint if needed, it calculates the bearing error and calculates the new roll setpoint using the PID function. It is important to note that the autopilot was initially programmed with point to point navigation only; it does not try to fly following a line between each waypoint. This simplified the navigation task. The Bearing is measured using the GPS and not the AHRS, compensating for any cross winds, as the GPS measures the true heading in which the aircraft is flying regardless of the actually attitude of the aircraft.

If the aircraft is operating in Auto_B mode, in addition to doing everything described previously, the task compares the aircrafts current altitude to the waypoint minimum altitude + threshold. If the altitude is below, it calculates the throttle output using the throttle curve described in 7.7.

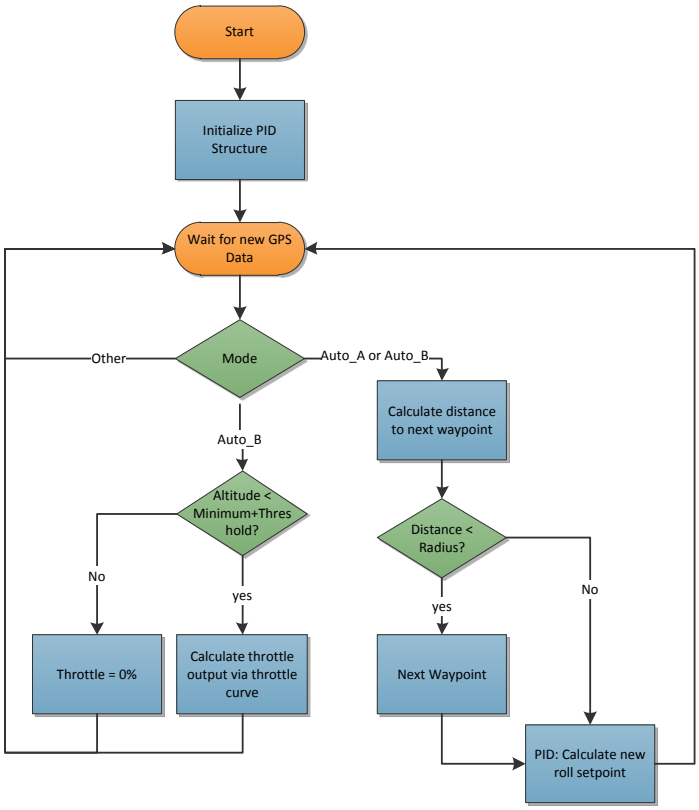


Figure 62: Navigate Task

Once those operations are completed the task is set to a block state waiting to be unblocked by the Read Parse Buffers task.

8.1.4 AIRSPEED TASK

The Airspeed task contains the second navigation control loop, the Pitch PID controller. The task starts by initializing the PID structures with gains in the user register, as well as the saturation levels defined by maximum and minimum pitch registers.

Every tick, the tasks checks the operating mode of the aircraft. If Auto_A or Auto_B is active, the task computes the airspeed in m/s from the pressure ADC value using the following equation:

$$Airspeed = \sqrt{1.2 \cdot 2 \cdot (V_{DPres} - V_0)}$$

With V_{Dpres} the pressure sensor voltage in millivolts, and V_0 the zero airspeed voltage.

It then computes the air speed error compared to cruise speed setpoint which is defined in a user register, and calculates the new pitch setpoint using the PID function.

Once completed, the task is delayed 50 [ms] before restarting.

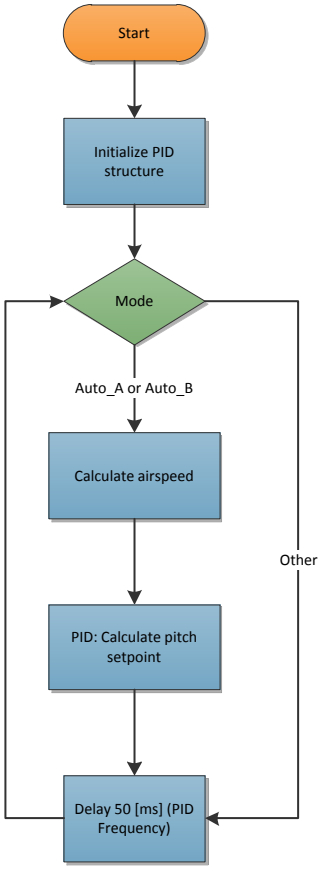


Figure 63: Airspeed Task

8.1.5 PRESSURE DATA TASK

The task starts by initializing the digital FIR filter of the pressure sensors. The FIR filter was implemented using the DSP (Digital Signal Processing) functions in the CMSIS (Cortex Microcontroller Software Interface Standard) library. The gains of the filter were calculated using Matlab. The filter has a very low cut off frequency as to remove as much noise as possible as well as soften the air speed measurement variation due to gust of winds and turbulence.

The filter was designed with a sampling frequency of 100Hz, with a cutoff frequency 5Hz and a stop frequency of 10Hz at – 60db. The filter runs at 100Hz due to the fact that the analog RC filter cutoff frequency is at 50Hz, and in order to respect Shannon's theorem the minimum frequency the filter can run at is 2x50 = 100Hz.

Using the filter builder tool the coefficients of a FIR filter were calculated, resulting in a 46 order filter:

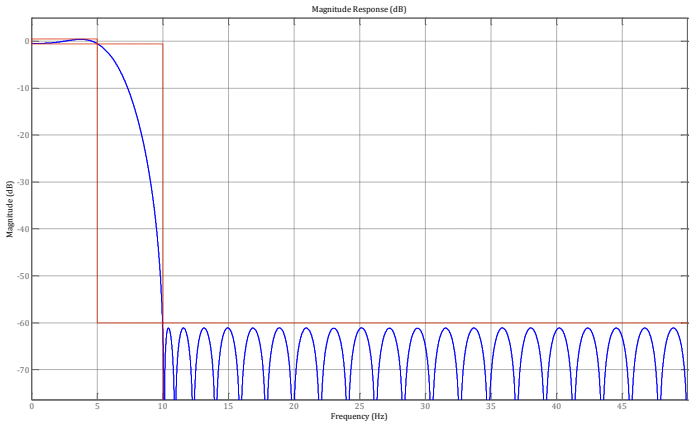


Figure 64: FIR Filter frequency response

After initializing the FIR filter, the task reads the ADC values that are stored in a circular buffer using a DMA and filters them using a CMSIS filter function. The result is then stored in a global register as a voltage in millivolts.

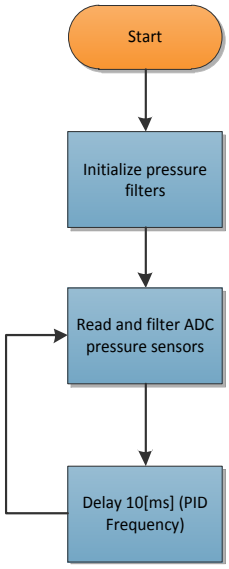


Figure 65: Pressure Data task

8.1.6 RECORD DATA TASK

The record data task makes use of the sdcard.c functions (written during the semester project) which create new files with incrementing names (DATALOG#001, DATALOG#002,...), as well as provides a function to write data to the file. The task itself controls at which frequency data is written to the file, as well as the user button that starts or stops recording. To indicate to the user whether the autopilot is recording or not a led is blinked at 1Hz.

One of the initial problems with the Record data task was the requirement of closing the file for it to be saved correctly on the SD card. If the SD card was removed, or the autopilot board was reset before closing the file, the file will look empty when opened on a computer.

To solve this problem, the FatFs library has a function called f_sync() that flushes the cached information of a writing file. When doing so, even if the file is not properly closed, the file can still be read using a PC with the information intact. Unfortunately calling the f_sync function takes a fair amount of time execute, and if called after every write operation (default 100Hz), the task starves the lower priority tasks of operating time. For this reason, the f_sync() function is only called at 1Hz.

When the task starts, it creates a file and begins recording. This was implemented in the event of an in air reset, or a damaged autopilot after an aircraft crash, so that the flight would still be recorded.

At every tick the task checks if it is recording (by using a Boolean). If it is currently recording it writes a new line in the log file and toggles on or off the led at 1Hz. At the same time it checks the state of the push button, looking for a rising edge. If the task detects a rising edge it either closes the file if was already recording, or creates a new log file if it was not.

It finishes with a delay of X [ms] which is defined by a user register, but is usually set at 10ms (100Hz) by default.

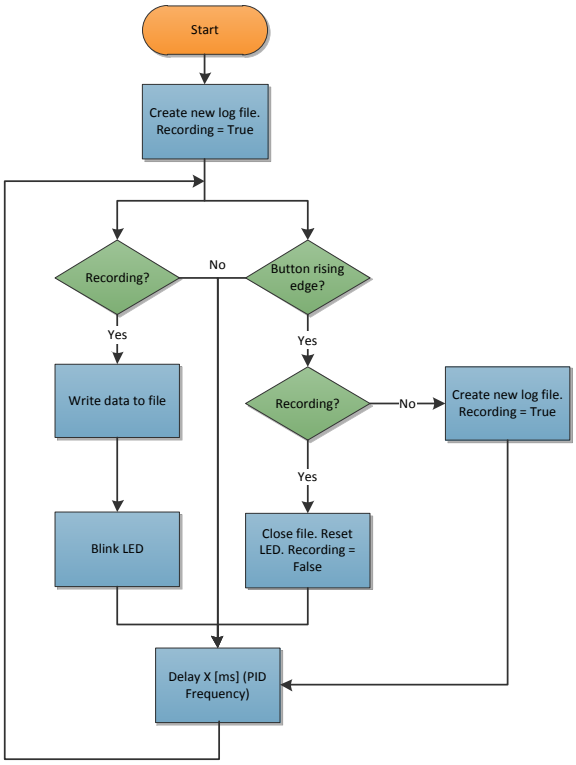


Figure 66: Record Data Task

The following parameters are stored in the log file:

- Yaw, Pitch, and Roll angles, and angular rates
- X,Y,Z accelerations
- GPS Latitude, Longitude, Altitude, Fix mode, Satellites in view, Heading, Ground speed
- Absolute and differential pressure
- Radio channels 1-7
- Roll and pitch setpoints
- Flight mode and current waypoint number
- Throttle, Aileron, and elevator servo outputs

8.1.7 RADIO TIMEOUT TASK

The radio timeout task was added after the crash of the Cularis which was due to the loss of control of the aircraft. As a security measure, a task was implemented which checks if the radio receiver is still receiving correct data (the radio commands).Since the Spektrum receiver sends new data every 20ms or 50 Hz, if no new data is detected within 100ms it does one of the following user selectable actions:

- Manual Neutral – All servos are set to their neutral position and the throttle is cut off
- FBW Neutral – The airplane is set to FBW mode, with the roll and pitch setpoint both set to 0 degrees
- Return Home – The aircraft is set to Auto_B mode, and the current waypoint is set to 0

The radio timeout is configured by the user via a register.

The detection of new radio data is done using a global Boolean variable. Every time a correct message is receive from the Spektrum receiver, it is set to true, and the end of the radio timeout task the Boolean is set to false. If the task starts and the Boolean is still set to false, that means no new radio data was received to set it back to true, and it can be concluded that the radio is no longer functioning. This offers a very quick reaction time of 1/10th of a second.

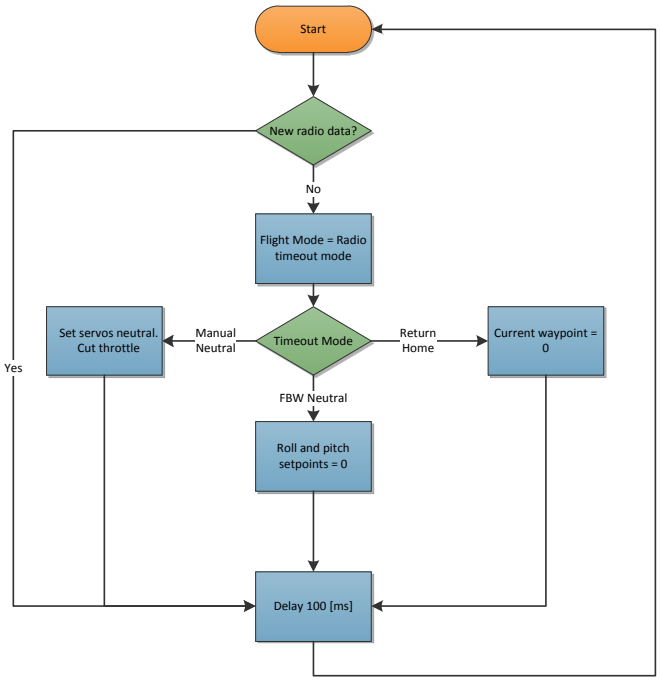


Figure 67: Radio Timeout task

8.1.8 MISSION CONTROL TASK

The mission control task handles all PC to autopilot communication. A custom NMEA protocol was defined during the semester project, with which to read sensor data and configure the autopilot using registers. A complete detailed register map can be found at 9.1.

In conjunction with this task, two functions were written: Register_Read(…) and Register_Write(…). The register read function takes the register number as an input and returns a string formatted according to the AXCG protocol. The register write function takes the ASCII data string (characters following the command and register number) directly, decodes it, and then stores the values in the global variables which serve as registers.

The mission control task determines what type of command was received (register read / write, store to EEPROM, etc), and calls the appropriate functions with the register number parameter.

In addition to writing and reading the registers, the task handles the Cycle Register Read task which is used to periodically read sensor data registers which can then be display on the Mission Control PC interface.

In order to use Vectornav's Sensor Explorer PC utility, which is used to configure the AHRS parameters, an extra mode was added to the mission control task. In this mode, all strings received from the AHRS are relayed and sent to the USB out buffer and all strings starting with "$VN…" are sent to the AHRS module. This essential turns the autopilot into a USB ↔ serial converter.

The task starts by reading the USB input buffer. If a complete message was received (detected by presence of a newline character '/n'), it continues on to decode the message. It starts with the header, by looking for either "$AX…" or "$VN…". Commands directed to the autopilot start with "$AX", and commands directed to the AHRS start with "$VN" allowing manual configuration of the VN-100 using a terminal emulation program on the PC, or using the Sensor Explorer PC interface. If neither one of these headers were recognized, it returns an error code 3 "$AXERR,3*..".

If the header starts with $AX, it reads the following three command characters looking for either RRG to read a register, WRG to write to a register, or SEE to store the configuration registers' content to the onboard EEPROM.

If it is either RRG or WRG, it reads the 2 character register number and then calls either the Register Read or Register Write function. When writing to the "Cycle Read Registers" register, it checks the values and if all of them are '0' it deletes the Cycle Read Register tasks, otherwise it creates a new instance of the Cycle Register Read task. If the command is SEE, it calls the EE_Store() function which stores all configuration registers to the EEPROM, which are then loaded every time the autopilot is turned on or restarted.

This cycle is repeated every 10ms (100Hz).

See section 9.1.1 for a list of the commands

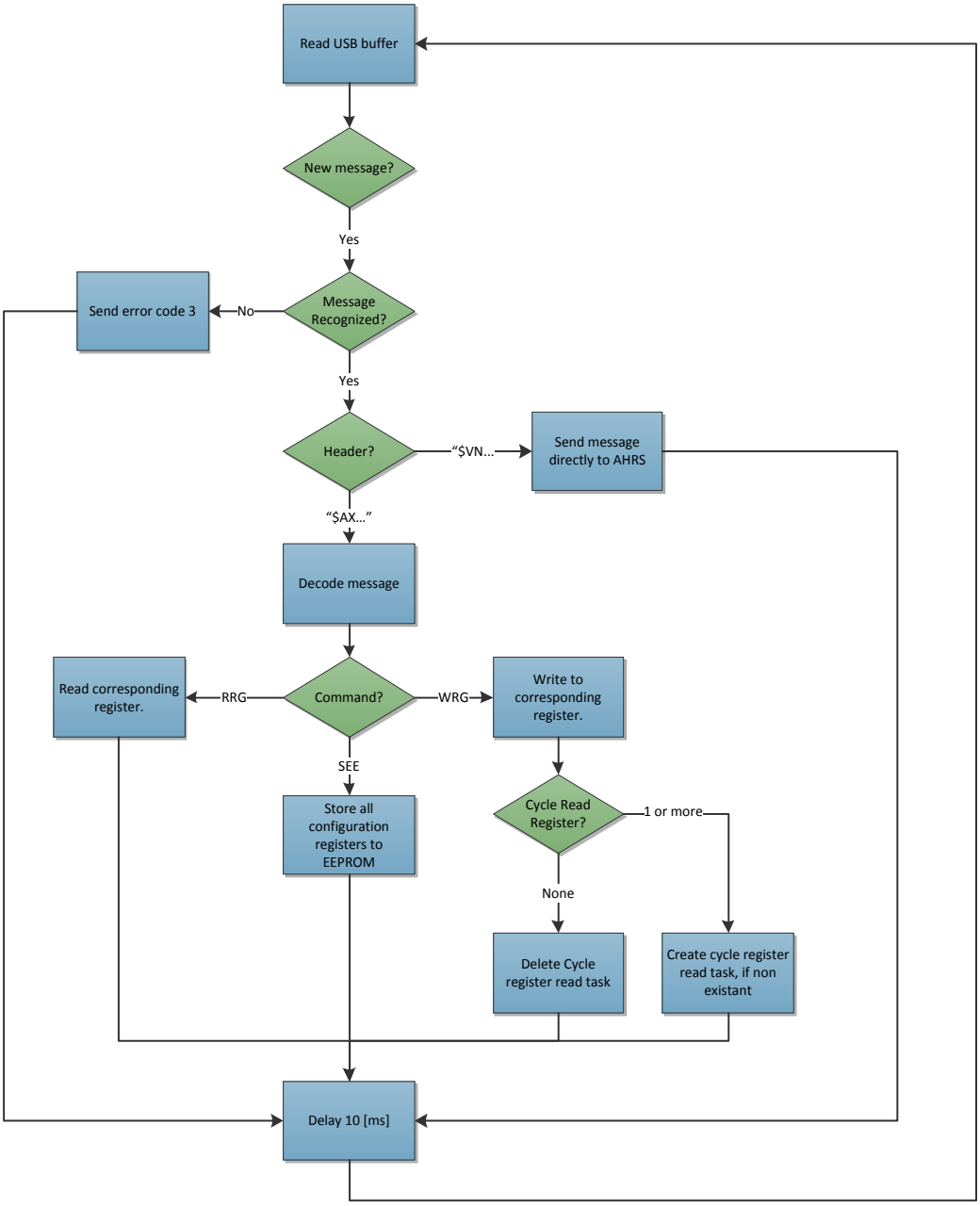


Figure 68: Mission Control task

8.1.9 GPS INITIALIZE TASK

The GPS initialize task is ran only once. The reason it was implemented as a task was because it required long time delay times, and using the RTOS is an easy way to create these delays without wasting processing time.

The GPS initialize tasks sends 3 strings to the GPS whenever the program is started. These strings configure the GPS in binary mode, running at 10Hz, with SBAS (satellite-based augmentation system) enabled.

Binary mode is a second data protocol option of the GPS that allows the GPS to send all relevant data directly in binary instead of the NMEA ASCII strings. This greatly simplifies the parsing code, and reduces the number of USART interrupts received by the GPS.

The following strings are sent to the GPS module:

- $PGCMD,16,0,0,0,0,0*6A – Configure in binary mode
- $PMTK220,100*2F – Configure to 10Hz refresh rate
- $PMTK313,1*2E – Enable SBAS

8.1.10 Flight Mode Task

The flight mode task is a simple task that reads the user push button and increments the flight mode every time a rising edge is detected. The flight mode is stored in a global variable which is then used by the other tasks. Once the flight mode reaches Auto_B, it returns to manual on the next button rising edge.

8.2 Processing time and runtime statistics

When developing an embedded system it can be very difficult to estimate the actual time the processor requires to perform a certain task. One of the greatest advantages of using FreeRTOS is that it provides run time statistics such as how much memory each task uses or how much processing time is used up by each task.

These runtime statistics showed that using floating point variables for the attitude and navigation controllers, even without hardware floating point, required actually very little processing time. It also showed that even with every task running at full load, there still is about ~80% processing time left, leaving much room for future expansion using the same microcontroller. Table 3 shows a worst case runtime statistics with the processing time spent per task. These statistics were produced under these conditions:

- Autopilot in Auto_B mode
- Recording active at 100Hz
- Cycle reading of YPR, GPS, and pressure registers at 30Hz

Task	Processing Time
Attitude Controller	<1%
Read Parse Buffers	6%
Navigate	<1%
Airspeed	<1%
Pressure Data	1%
Record Data	9%
Radio Timeout	<1%
Mission Control	<1%
Flight Mode	<1%
IDLE	80%

Table 3: Task Processing Time

The most time consuming task is the record data task and that is largely due the f_sync function explained previously, even though f_sync is called very second. Without calling the f_sync function the record data task drops to 4-6%. Nonetheless, the processor still has 80% processing time left over which means that it is not a problem as long as the lower priority tasks are not critical to the flight navigation.

This does not include time spent in interrupt routines. However, both interrupt routines were kept very short as to minimize the overhead time.

The total code size of the firmware is 102.8 Kb, out of the total 256Kb available in the microcontroller.

8.3 IMPORTANT BUGS/ERRORS AND FIXES

8.3.1 RADIO PARSING ERROR

This bug caused the crash of one of the Cularis. It was due to an error in the radio parsing function created during the semester project.

The Spektrum receiver periodically sends 16 bytes at 50Hz with each channel coded on two bytes. It was thought that the first two bytes were constant and served synchronization bytes: 0x03 and 0x01.

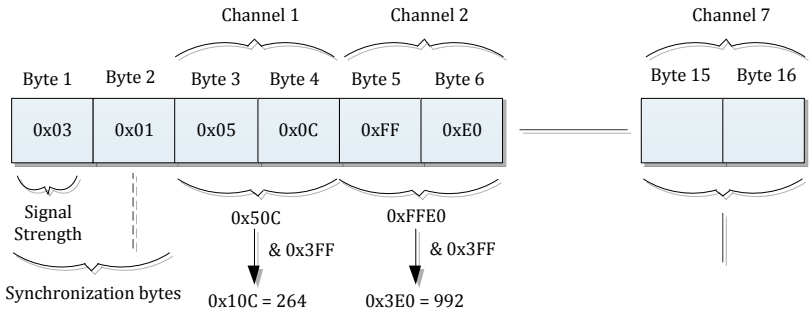


Figure 69: Spektrum data protocol

It turned out that the first byte is not constant, but in fact represents the signal strength of the radio. The first byte can change between 0x03 for full strength, 0x02 for medium and 0x01 for low.

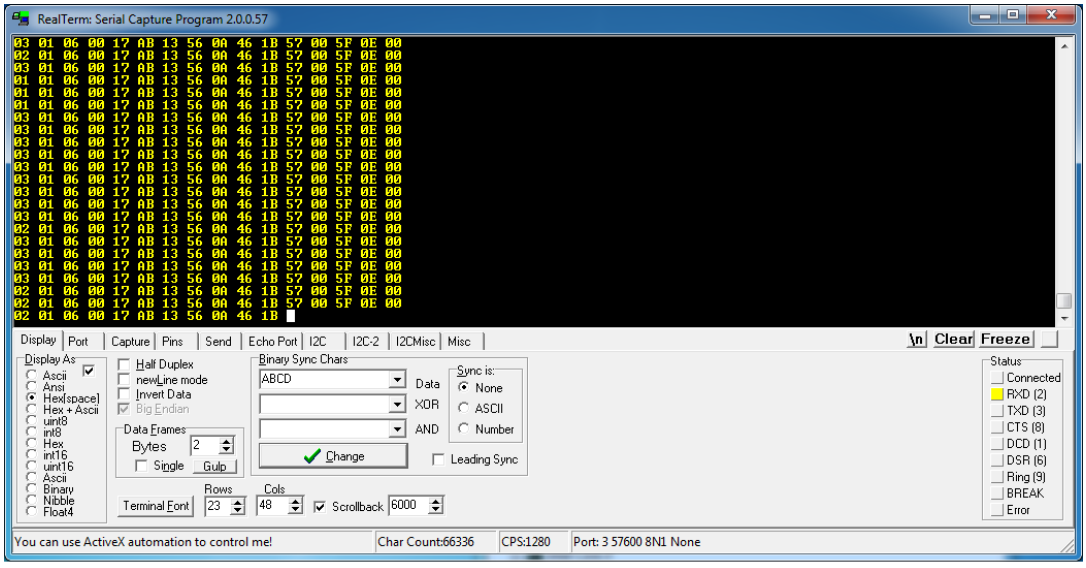


Figure 70: Spektrum data stream - varying signal strength

In Figure 70 we can see that the first byte changes between 0x03, 0x02 and 0x01. This was obtained when the radio was moved far away from the receiver reducing the signal strength.

The old parsing functions would synchronize only with the 0x03, which caused a signal loss if the signal strength dropped anywhere below ‘strong’ or 0x03

This is exactly what happened during a fatal Cularis flight, translating into a loss of control of the aircraft and eventually a nose first crash into the ground, with extensive damage to the autopilot board.

The solution was simply to accept 0x02 or 0x01, in addition to the 0x03, and then to look the second synchronization byte 0x01 that follows.

8.3.2 AHRS CENTRIFUGAL FORCE SENSITIVITY

During initial FBW tests, the aircraft did not behave as expected. In those tests, when the aircraft was set into a turn by increasing the roll setpoint and then released by setting the setpoint back to zero, the aircraft would continue to bank and turn. When reviewing the data logs afterwards, the PID control loops showed that the roll setpoint and the measured roll angle were on top of each, meaning that the PID was working, or it thought it was working.

There could be only one explanation, and that was that the AHRS we providing erroneous information. It turned out that the AHRS was prone to the centrifugal acceleration which is presented anytime the aircraft turns, which meant that during turns the attitude estimation was completely wrong.

The solution was to use the dynamic disturbance filter option of the VN-100. The acceleration dynamic filter compares the norm of the accelerometer measurements to the gravity acceleration (~-9.81), and computes the difference. The AHRS then changes the Kalman filter measurement variance gains on the fly according to the error. This meant that as soon as the AHRS was influenced by an external acceleration (causing the overall norm to increase or decrease) the AHRS would ignore accelerometer data.

Adding the dynamic filter solved this problem, and the following FBW test flight proved to be successful.

9. PC USER INTERFACE: MISSION CONTROL

9.1 AXCG ↔ PC COMMUNICATION PROTOCOL

Continuing on the protocol defined during the semester project, new registers had to be added to the autopilot, to configure the flight parameters of the aircraft as well as the control loop gains and waypoint trajectory, etc. In addition, the register numbers are now coded with two characters instead of one, increasing the register limit from 10 to 100.

Additionally a new command was added to store flight configuration registers (registers 09 to 23) in an external EEPROM. When the autopilot is turned on or restarted, the registers are loaded with the values stored in the EEPROM.

9.1.1 COMMANDS

Description	Command	Response
Register Read	$AXRRG,XX *CS\n	$AXRRG,XX ,D1,D2,Dx*CS\n
Register Write	$AXWRG,XX,D1,D2,Dx*CS\n	$AXWRG*CS\n
Store to EEPROM	$AXSEE*4A	$AXSEE*4A

Data fields are represented by Dx. The number of Data fields and their width in bytes depend on the register being written or read. Consult the Register Map for details. 'CS' contains a 2 byte checksum of the string starting after the '$' until the '*'. All commands must be terminated with an end of line character '\n'. If this character is not present the AXCG will not recognize the command string.

9.1.2 ERROR CODES

If there's any error regarding the command sent, the ACXG will return an error code indicating the source of the error. The string format is the following:

$AXERR,X*CS
Where X can take a value between 1 and 9. Every value corresponds to a specific type of error described in the table below.

Error Name	Error Code	Description
Invalid Register Number	1	Read / Write to an invalid register number
Bad Checksum	2	Checksum mismatch
Invalid Command	3	Cannot recognize command

9.1.3 REGISTER MAP

Register Name	Register ID	Read/Write	Width Bytes	Reset Value
YPR Data	01	R	7,7,7	N/A
GPS Data	02	R	9,9,5,4,1	N/A
Pressure Voltage Data	03	R	4,4	N/A
Cycle Read Registers	04	R/W	1,1,1	0,0,0
Cycle Read Frequency	05	R/W	3	20
UI Mode	06	R/W	1	0
SD Log Frequency	07	R/W	3	20
Servo Positions	08	R	4,4,4,4,4,4,4	N/A

Register Name	Register ID	Read/Write	Width Bytes	Reset Value
Attitude PID Frequency	09	R/W	3	Last Stored Value
Aileron PID	10	R/W	5,5,5	Last Stored Value
Elevator PID	11	R/W	5,5,5	Last Stored Value
Navigation PID Frequency	12	R/W	3	Last Stored Value
Roll PID	13	R/W	5,5,5	Last Stored Value
Pitch PID	14	R/W	5,5,5	Last Stored Value
Servo Endpoints	15	R/W	1,4,4	Last Stored Value
Max Bank	16	R/W	2	Last Stored Value
Max Pitch	17	R/W	3,3	Last Stored Value
Waypoints	18	R/W	2,8,8,5,4,2	Last Stored Value
Cruise Speed	19	R/W	4	Last Stored Value
GPS Altitude Error	20	R/W	4	Last Stored Value
Throttle Altitude Threshold	21	R/W	4	Last Stored Value
Pitch Offset	22	R/W	6	Last Stored Value
Radio Timeout Mode	23	R/W	1	Last Stored Value

9.1.4 YPR DATA

Number	Name	Width	Field Description
1	Yaw	7	Yaw angle in milli degrees. Contains a + or – sign. 1 digit is used for the sign, the 6 others for the value.
2	Pitch	7	Pitch angle in milli degrees. Contains a + or – sign. 1 digit is used for the sign, the 6 others for the value.
3	Roll	7	Roll angle in milli degrees. Contains a + or – sign. 1 digit is used for the sign, the 6 others for the value.

READ EXAMPLE

Command	Response
$AXRRG,01*73	$AXRRG,01,+102520,+010500,-057320*71\n

Field	Value
Yaw	$+102520*10^{-3} = 102.52$ [Deg]
Pitch	$+010500*10^{-3} = 10.5$ [Deg]
Roll	$-057320*10^{-3} = -57.32$ [Deg]

9.1.5 GPS DATA

Number	Name	Width	Field Description
1	Latitude	9	GPS latitude position given in micro degrees
2	Longitude	9	GPS longitude position given in micro degrees
3	Altitude	5	GPS Altitude given in centimeters
4	Ground Speed	4	Speed relative to the ground (earth reference frame) in centimeters per second
5	Satellites	1	Number of satellites in view
6	GPS Fix	1	Type of GPS fix: 1 : No Fix 2: 2D Fix 3: 3D Fix

READ EXAMPLE

Command	Response
$AXRRG,02*70	$AXRRG,02,046284383,12028438,42574,1223,3*6C\n

Field	Value
Latitude	$046284383*10^{-6}$ = 46. 284383 [Deg]
Longitude	$12028438*10^{-6}$ = 120.284383 [Deg]
Altitude	$42574 *10^{-2}$ = 425.57 [m]
Ground Speed	$1223*10^{-2}$ = 12.23 [m/s]
GPS Fix	3= 3D Fix

9.1.6 PRESSURE VOLTAGE DATA

Number	Name	Width	Field Description
1	Differential Pressure Voltage	4	Voltage output from the differential pressure sensor. Given in millivolts
2	Absolute Pressure Voltage	4	Voltage output from the absolute pressure sensor. Given in millivolts

READ EXAMPLE

Command	Response
$AXRRG,03*70	$AXRRG,03,1246,3090*6C\n

Field	Value
Differential Pressure Voltage	1246 [mV]
Absolute Pressure Voltage	3090 [mV]

9.1.7 CYCLE READ REGISTERS

The AXCG is capable of sending data via USB automatically at a fixed frequency defined in the Cycle Read Frequency register. "The Cycle Read Register" is used to define what data the AXCG should send. The three registers of choice are the "YPR Data register", the "GPS Data register", and the "Pressure Voltage Data" register. Writing a '1' in the corresponding field will enable the cyclic read for that register.

Number	Name	Width	Field Description
1	YPR	1	If = 1, the YPR Data register will be read and sent cyclically at the frequency defined in Cycle Read Frequency register
2	GPS	1	If = 1, the GPS Data register will be read and sent cyclically at the frequency defined in Cycle Read Frequency register
3	Pressure Voltage	1	If = 1, the Pressure Voltage Data register will be read and sent cyclically at the frequency defined in Cycle Read Frequency register

WRITE EXAMPLE

Command	Response
$AXWRG,04,1,1,0*4F\n	$AXRRG*5B\n

Field	Value
YPR	True
GPS	True
Pressure Voltage	False

9.1.8 CYCLE READ FREQUENCY

To define at which frequency the AXCG should send the register data (defined in Cycle Read Registers), the user has access to the Cycle Read Frequency register which dictates at what frequency the registers should be read.

Number	Name	Width	Field Description
1	Frequency	3	Frequency at which the register defined in the "Cycle Read Registers" Register should be read. Defined in Hertz, with a range from 0Hz to 999Hz

WRITE EXAMPLE

Command	Response
$AXWRG,05,050*4F\n	$AXRRG*5B\n

Field	Value
Frequency	50Hz

9.1.9 MODE

The AXCG is capable of operating with two different PC applications. The first is the custom made Mission Control used to program waypoints and other various settings of the autopilot. The other is the Sensor Explorer application from VectorNav that is used for the VN-100 development board, which was adapted so it can be used with the AXCG. The Sensor Explorer application is used to tune the Kalman filter and change the various settings of the AHRS module. In order to use one application or the other, the user has to define in which mode the AXCG needs to operate in, by writing a 1 or 0 in the Mode register.

Number	Name	Width	Field Description
1	Mode	1	0 : Mission Control Mode 1 : Sensor Explorer Mode

WRITE EXAMPLE

Command	Response
$AXWRG,06,0*5D\n	$AXRRG*5B\n

Field	Value
Mode	0 = Mission Control

9.1.10 SD LOG FREQUENCY

The AXCG has a micro SD card slot that is used for data logging of flight control data. The user chooses at which frequency data is stored by writing in the SD Log Frequency register. Any input between 0 to 999Hz is accepted. But anything over 200Hz would have little use, as the on board sensors have a maximum data output of 200Hz or less.

Number	Name	Width	Field Description
1	Frequency	3	Micro SD card data log frequency. Defined in Hertz, with a range from 0Hz to 999Hz

WRITE EXAMPLE

Command	Response
$AXWRG,07,100*5D\n	$AXRRG*5B\n

Field	Value
Frequency	100 Hz

9.1.11 Attitude PID Frequency

Register determines the frequency of the attitude control loops (Aileron PID and Elevator PID). The attitude control loops commands the servo outputs, which are refreshed at 50 Hz. It is therefore recommended to have this loop run at 50Hz. anything higher will have little effect as the servos will not be able to react quickly enough.

Number	Name	Width	Field Description
1	Frequency	3	Attitude control loop frequency: 0-999 Hz

Write Example

Command	Response
$AXWRG,09,050*5D\n	$AXRRG*5B\n

Field	Value
Frequency	50Hz

9.1.12 Aileron PID

Register contains the gains of the Aileron PID in the attitude control loops. The aileron PID controls the roll of the aircraft using the aileron servos. The PID controller is defined by its three gains: Kp – Proportional gain, Ki – Integral gain, and Kd – Derivative gain. The gains are codded in fixed point with 2 decimal values.

Number	Name	Width	Field Description
1	Kp	5	Aileron PID proportional gain. Fixed point 2 decimal values
2	Ki	5	Aileron PID integral gain. Fixed point 2 decimal values
3	Kd	5	Aileron PID derivative gain. Fixed point 2 decimal values

Write Example

Command	Response
$AXWRG,10,05020,00723,00000*5D\n	$AXRRG*5B\n

Field	Value
Kp	50.2
Ki	7.23
Kd	0

9.1.13 Elevator PID

Register contains the gains of the Elevator PID in the attitude control loops. The elevator PID controls the pitch of the aircraft using the elevator servo. The PID controller is defined by its three gains: Kp – Proportional gain, Ki – Integral gain, and Kd – Derivative gain. The gains are codded in fixed point with 2 decimal values.

Number	Name	Width	Field Description
1	Kp	5	Aileron PID proportional gain. Fixed point 2 decimal values
2	Ki	5	Aileron PID integral gain. Fixed point 2 decimal values
3	Kd	5	Aileron PID derivative gain. Fixed point 2 decimal values

Write Example

Command	Response
$AXWRG,11,01500,00350,00632*5D\n	$AXRRG*5B\n

Field	Value
Kp	15
Ki	3.5
Kd	6.32

9.1.14 Navigation PID Frequency

Register determines the Frequency of the navigation control loops (roll PID and pitch PID). The navigation control loops command the setpoints of the attitude control loops. The roll PID is linked to the gps, meaning it refreshes every time new GPS data is received, which is fixed at 10 Hz. Therefore only the pitch PID can have its frequency changed, which is done via this register.

Number	Name	Width	Field Description
1	Frequency	3	Attitude control loop frequency: 0-999 Hz

Write Example

Command	Response
$AXWRG,12,020*5D\n	$AXRRG*5B\n

Field	Value
Frequency	20Hz

9.1.15 Roll PID

Register contains the Roll PID in the navigation control loops. The roll PID controls the roll setpoint of the aileron PID loop. The PID controller is defined by its three gains: Kp – Proportional gain, Ki – Integral gain, and Kd – Derivative gain. The gains are codded in fixed point with 2 decimal values.

Number	Name	Width	Field Description
1	Kp	5	Aileron PID proportional gain. Fixed point 2 decimal values
2	Ki	5	Aileron PID integral gain. Fixed point 2 decimal values
3	Kd	5	Aileron PID derivative gain. Fixed point 2 decimal values

Write Example

Command	Response
$AXWRG,13,00830,00012,00000*5D\n	$AXRRG*5B\n

Field	Value
Kp	8.3
Ki	0.12
Kd	0

9.1.16 Pitch PID

Register contains the Roll PID in the navigation control loops. The roll control loop controls the roll setpoint of the aileron PID loop. The PID controller is defined by its three gains: Kp – Proportional gain, Ki – Integral gain, and Kd – Derivative gain. The gains are codded in fixed point with 2 decimal values.

Number	Name	Width	Field Description
1	Kp	5	Aileron PID proportional gain. Fixed point 2 decimal values
2	Ki	5	Aileron PID integral gain. Fixed point 2 decimal values
3	Kd	5	Aileron PID derivative gain. Fixed point 2 decimal values

Write Example

Command	Response
$AXWRG,14,00830,00012,00000*5D\n	$AXRRG*5B\n

Field	Value
Kp	8.3
Ki	0.12
Kd	0

9.1.17 Servo Endpoints

The servo endpoints register sets the limits of the minimum and maximum position the servos can take. Absolute limits are -500 (minimum) and +500 (maximum). This registered is used to limit the throw of the servos in case there are physical obstructions.

Number	Name	Width	Field Description
1	Channel	1	Channel Number (0-7)
2	Minimum	4	Minimum servo position (-500 to 500). Contains a + or – sign.
3	Maximum	4	Maximum servo position (-500 to 500). Contains a + or – sign.

Write Example

Command	Response
$AXWRG,15,1,-450,+500*5D\n	$AXRRG*5B\n

Field	Value
Channel	1
Minimum	-450
Maximum	+500

9.1.18 Max Bank

Registers contains the maximum bank angle of the aircraft when operating in fly by wire or automatic mode. The bank angle is symmetric - there's is no differentiation between left and right turns.

Number	Name	Width	Field Description
1	Bank Angle	2	Maximum bank when in FBW or Auto mode, in degrees

Write Example

Command	Response
$AXWRG,16,35*5D\n	$AXRRG*5B\n

Field	Value
Bank Angle	35

9.1.19 MAX PITCH

Registers contains the min and max pitch limits of the aircraft when operation in FBW or auto mode

Number	Name	Width	Field Description
1	Min Pitch	3	Minimum pitch angle allowed, in degrees. Contains a + or – sign.
2	Max Pitch	3	Maximum pitch angle, in degrees. Contains a + or – sign.

WRITE EXAMPLE

Command	Response
$AXWRG,17,-15,+25*5D\n	$AXRRG*5B\n

Field	Value
Minimum pitch	-15
Maximum pitch	+25

9.1.20 WAYPOINTS

The Waypoints register is actually a table of registers. The autopilot accepted a maximum of 32 waypoints. Each waypoint is defined by the follow: Waypoint number, Latitude, Longitude, Altitude, Radius, and the next waypoint. The user can only read or write to one waypoint at a time.

Number	Name	Width	Field Description
1	Waypoint Number	2	The Waypoint number. Can take any value between 00 and 32.
2	Latitude	8	Definition of the waypoint's latitude. Given in micro degrees ($x10^6$)
3	Longitude	8	Definition of the waypoint's longitude. Given in micro degrees ($x10^6$)
4	Altitude	5	Minimum altitude the aircraft can drop too when heading to this waypoint. Given in centimeters ($x10^2$)
5	Radius	5	Radius of the waypoint. Given in centimeters ($x10^2$)
6	Next	2	The next waypoint to go to once the aircraft reaches the current waypoint

WRITE EXAMPLE

Command	Response
$AXWRG,18,04,42041732,06378920,50252,1000,05*5D\n	$AXRRG*5B\n

Field	Value
Waypoint Number	4
Latitude	$42041732 \cdot 10^{-6} = 42.041732$ [°]
Longitude	$06378920 \cdot 10^{-6} = 6.378920$ [°]
Altitude	$50252 \cdot 10^{-2} = 502.52$ [m]
Radius	$1000 \cdot 10^{-2} = 10$ [m]
Next	5

9.1.21 Cruise Speed

Register contains the cruise speed at which the aircraft will try to fly when operation in automatic mode.

Number	Name	Width	Field Description
1	Speed	4	Airspeed setpoint when in autopilot mode. Given in [cm/s]

Write Example

Command	Response
$AXWRG,19,1250*5D\n	$AXRRG*5B\n

Field	Value
Cruise speed	$1250 \cdot 10^{-2} = 12.5$ [m/s]

9.1.22 GPS Altitude Error

The altitude of the aircraft is determined using the GPS. Unfortunately the accuracy of the GPS in the z axis (altitude) is much less reliable. The GPS altitude error register is an added security to add fix error estimation to the GPS altitude. The final altitude is calculated by taking the GPS altitude and subtracting the GPS altitude error.

Number	Name	Width	Field Description
1	Altitude Error	4	GPS altitude error estimation, of which to subtract to the altitude given by the GPS. Given in [cm]

Write Example

Command	Response
$AXWRG,20,0800*5D\n	$AXRRG*5B\n

Field	Value
Altitude error	$800 \cdot 10^{-2} = 8$ [m]

9.1.23 Throttle Altitude Threshold

The throttle altitude threshold defines when the aircraft will start to apply throttle as a security measure for getting to close to the waypoint's minimum altitude. The throttle is applied linearly starting at the Waypoint Altitude + Throttle Threshold, and is at a 100% when it reaches the minimum altitude. Refer to 7.7 for more details.

Number	Name	Width	Field Description
1	Throttle Threshold	4	Altitude above the minimum altitude for which the autopilot will start applying throttle. Given in [cm]

Write Example

Command	Response
$AXWRG,21,0550*5D\n	$AXRRG*5B\n

Field	Value
Altitude error	$550 \cdot 10^{-2} = 5.5$ [m]

9.1.24 Pitch Offset

It is not always possible to mount the autopilot PCB perfectly level in the aircraft. For this reason a Pitch offset registers was added that allows the user to add an offset to the pitch angle to correct for any possible misalignment. The final pitch angle is the sum of the pitch measurement from the AHRS plus the pitch offset.

Number	Name	Width	Field Description
1	Pitch Offset	6	Pitch value of which to add to the pitch obtained from the AHRS. Given in millidegrees.

Write Example

Command	Response
$AXWRG,22,+03500*5D\n	$AXRRG*5B\n

Field	Value
Altitude error	$3500 \cdot 10^{-3} = 3.5$ [°]

9.1.25 RADIO TIMEOUT

In the event that the aircraft should lose the radio signal, the autopilot enters in radio timeout mode and performs various actions depending on the setting in the Radio Timeout register. There are three options:

- Servos Neutral and Throttle off
- FBW Neutral setpoint and throttle off
- Return home.

Number	Name	Width	Field Description
1	Timeout Mode	1	0 – Manual Neutral : All servos are set to their neutral position and the throttle is cut off 1 – FBW Neutral: Aircraft enters fly by wire mode, with the roll and pitch setpoints set to 0. Throttle is cut off. 2 – Return Home : Aircraft enters autopilot mode, and the current waypoint is set to 0

WRITE EXAMPLE

Command	Response
$AXWRG,23,1*5D\n	$AXRRG*5B\n

Field	Value
Timeout Mode	1 – FBW Neutral

9.2 MISSION CONTROL PC INTERFACE

It was absolutely critical to have an intuitive, easy to use, yet powerful PC interface to configure the autopilot with all its operating modes and options, as well as to view sensor data or replay a flight. Thus the Mission Control application was created. Mission Control was designed in C# .Net using Microsoft visual studios as a Windows Forms Application.

The application uses a tabbed format with three mains tabs

- Virtual cockpit – This tab is used to view sensor data (AHRS, GPS, and pressure / airspeed), as well as playing back a flight using a data log. (Figure 72)
- Route planner – This tab is used to create a waypoint trajectory of which the autopilot will try to follow. (Figure 73)
- Control Loops – This tab contains all the flight parameters such as the PID gains for the four controllers, the control loop frequencies, the control loop limits or saturation levels, the servo endpoints, as well as the radio timeout mode. (Figure 74)

A fourth tab is present which is used for debugging purposes. It serves as a serial interface to view the raw data sent and receive from the autopilot, as well as giving the possibility to manually send commands.

On the outside of the tabs, there is the flight playback box with the file browser and the play/pause/stop buttons. There is also the COM Port setting to select which COM port the autopilot is using and to either connect or disconnect the autopilot.

The development of the Mission Control application can be separated into three key parts:

- Google earth integration and interaction – JavaScript functions, route planner and moving map
- AXCG communication protocol - loading and storing parameters, as well as virtual instruments
- Flight playback from a data log file

9.2.1 GOOGLE EARTH IMPLEMENTATION

The implementation of Google earth was the most difficult part of designing the application as it is the most critical instrument, and not directly integrated with C#. The Google earth windows are actually integrated web browsers displaying a custom made html file running Google earth using Google's API. In order to interact with Google earth, custom made JavaScript functions in the html files had to be written that make use of Google API in order to display icons, waypoints, as well as extract data, and events (such as mouse double click, and drag and drop).

The following functions were written in JavaScript, which could then be called through the web browser in C#:

- set_plane(latitude,longitude,heading) – Creates, positions, and rotates an airplane icon on the map according to its latitude and longitude position, and heading
- set_camera(latitude,longitude,altitude) – Moves the camera view to the latitude and longitude position, and at a specific altitude
- set_home(latitude,longitude,altitude,clamped) – Creates and positions the home icon according to its latitude and longitude position, and altitude. Additionally there is the option to clamp the icon, meaning to either force it on the ground, or letting it float in the air at the defined altitude
- add_waypoint(latitude,longitude,altitude,index,clamped) – Same as the set_home function with the key difference of the index number, which is used to choose the number displayed on the icon
- add_line(latitude,Longitude,Altitude) – Adds a line continuing from that last waypoint. Used to draw looped trajectories

In addition to these JavaScript functions, there are two Google earth events (defined within the html file) defined that callback a C# function. These events are:

- Double click – Calls JSDoubleClick_ with the latitude and longitude of the click
- Placemark(waypoint) drag and drop – calls JSDragDrop_ with the latitude and longitude of where the placemark was dropped, as well the place mark reference

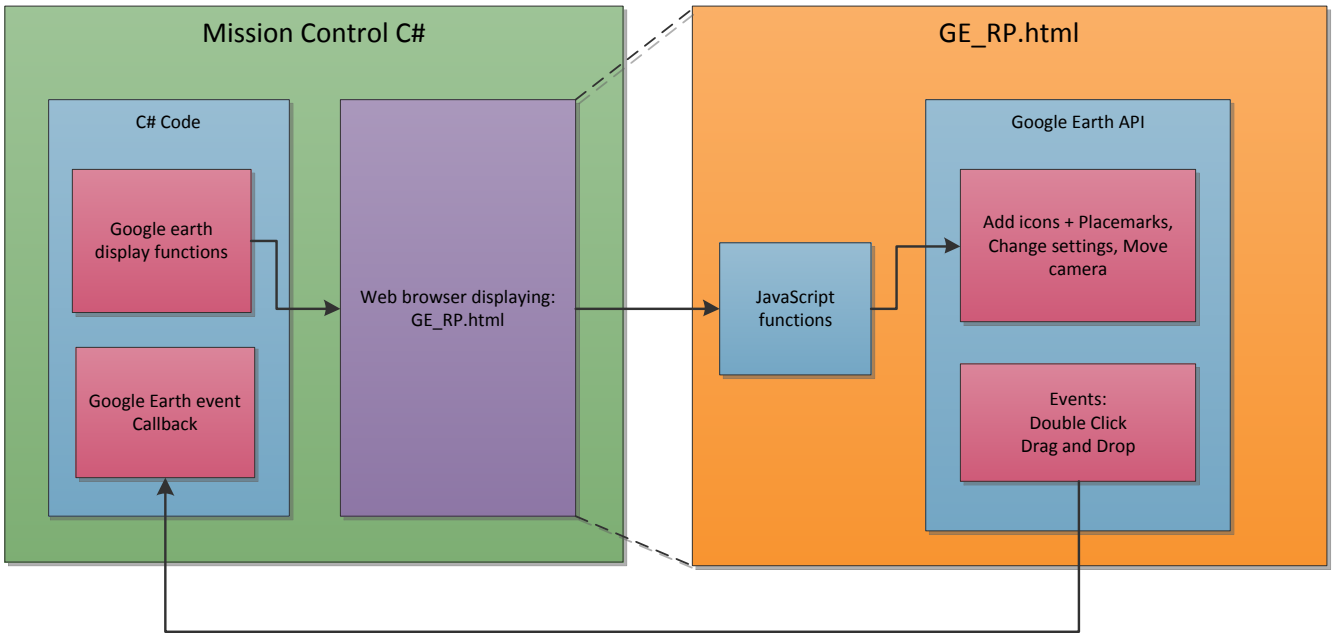


Figure 71: Google earth implementation via html JavaScript

Using these JavaScript functions, and callback events, it was possible to create a GPS moving map to view the GPS as well as replaying a flight log and to add/move/remove waypoints in the route planner tab. Figure 71 shows the interaction between the C# application the web browser / html file.

9.2.2 Virtual Cockpit and Data Log playback

The main tab is called the virtual cockpit, as it contains all the instruments you would typically find in a full scale aircraft such as the "six pack" (airspeed, artificial horizon, altitude, turn indicator, heading, and vertical speed) and the GPS moving map. The 6 instrument objects were taken from another open source application called HappyKillmore's Ground Control Station found on www.diydrones.com. It was a simple matter of importing the objects into the project, where they can then be controlled using class functions.

The virtual cockpit can display data from two different sources. The first is directly from the autopilot via USB. When the autopilot's Cycle Read Registers (register 04) is configured correctly, all sensor data is displayed using the instruments (AHRS – artificial horizon and heading, etc.), as well as an airplane icon which is displayed on the moving map using the set_plane and set_camera functions. All this is done within the Parse_String function.

The Parse_String function was written in order to decode AXCG strings from the autopilot and store the values in global variables as well displaying them using the instruments in the virtual cockpit tab, the data grid array waypoints in the route planner tab, and the flight parameters text boxes in the Control loops tab.

The second possible source is a data log file. Using a file browser, the user can select a .txt file, which the program will open once selected. The flight playback is done using a timer which runs at the same frequency as the data log (usually

100Hz). On every tick a line is read and the string is parsed, converting and storing the data in global variables. It is then displayed using the instruments and GPS moving map exactly as before. The timer is set to disabled by default, and is enabled when the user clicks the play button. Pressing the pause or stop button will disable the timer again. The timer period is defined by using a combo box with different standard frequencies such as 200,100, and 50 Hz, with the period equal to 1/frequency.

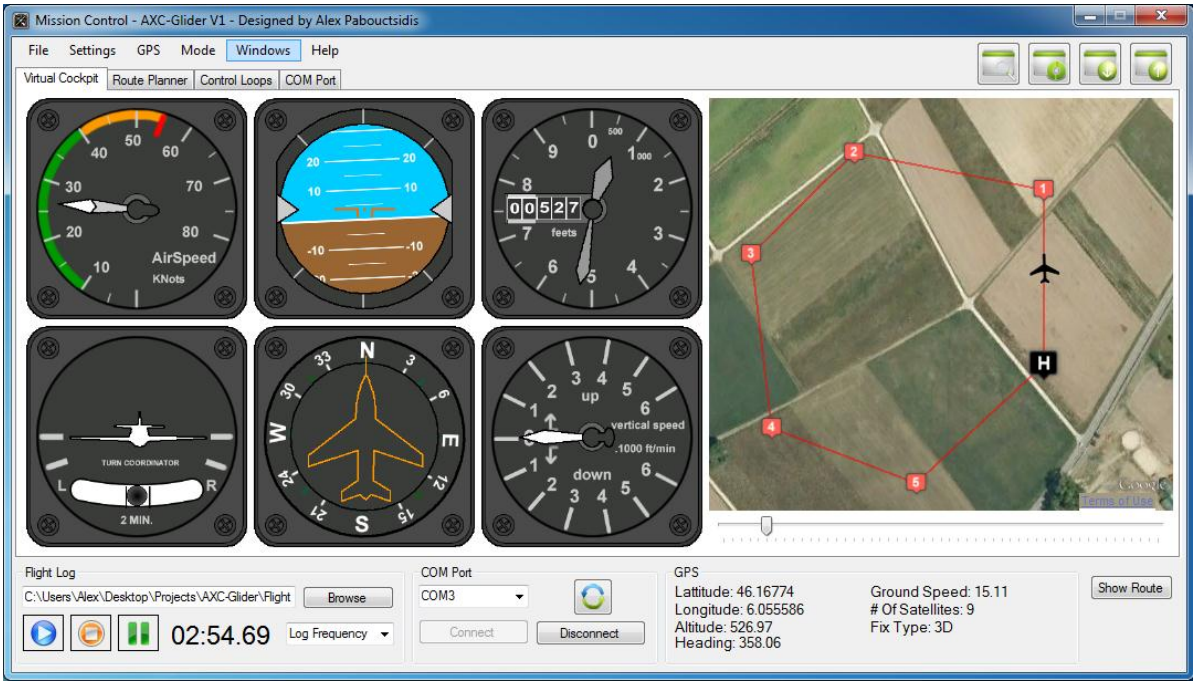


Figure 72: Virtual cockpit tab with flight playback

9.2.3 ROUTE PLANNER

The route planner works using a data grid array object containing 6 columns, with each column corresponding to a parameter of the waypoints.

When the user double clicks on the map, the JSDragDrop_ C# function is called with the location of the double click. A new row is then added to the data grid array with the latitude, and longitude of the double click, and the default altitude and radius defined in the boxes below the map. The next waypoint parameter is automatically set to the next waypoint (+1) and the waypoint number is set to the position in the grid array. For this reason the array is unsortable, as to not change the order of the waypoints.

A function called GE_Update was created that clears everything on the map and then reads the data grid array line by line, adding the waypoint icon and line. The GE_Update function is called within a 'cell change' event of the data grid array object. This means that whenever a single cell is changed, no matter who changed it (user, or other internal function), the entire map is redrawn.

The end user can create an infinite loop at the end of the trajectory by changing the next waypoint number to an earlier waypoint. The GE_Update function will close the loop by adding a line to the next waypoint. However, it will only do so for the last waypoint, so if any waypoint is added after the loop, it will not be displayed properly (however the aircraft will still fly the trajectory as intended).

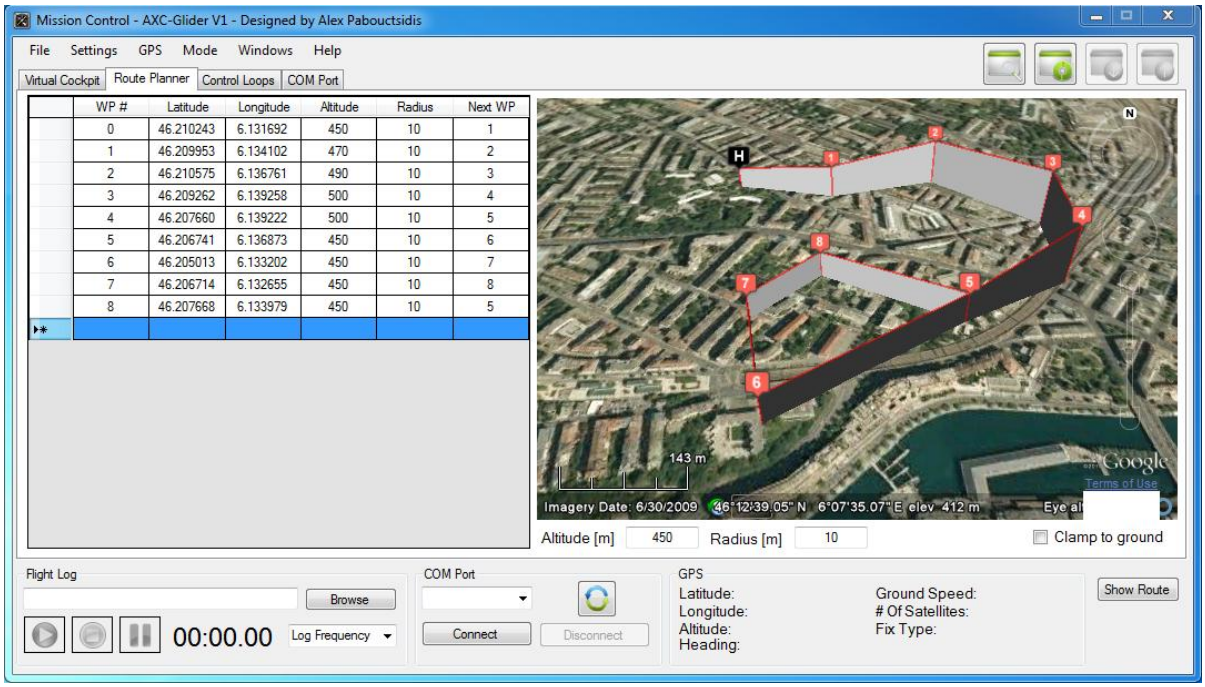


Figure 73: Route planner tab

9.2.4 Uploading and Downloading Parameters

The Control loops tab is simply composed of organized text, numeric up down, and combobox objects. In this tab the user can configure the following:

- Aileron, Elevator, Roll, and Pitch PID gains
- Attitude PID frequency
- Navigation limits (Roll and Pitch PID saturation limits)
- Cruise speed
- GPS altitude error estimate
- Pitch offset, to compensate if the AXCG autopilot is not level inside the aircraft
- Radio Timeout mode
- Servo Endpoints

When the user presses the download data icon at the top left, the application starts sending register read commands ($AXRGG,XX) starting at 09 all the way to 23. After every read command sent, the thread is paused for 20ms, leaving enough time for the autopilot to respond. The received strings are then sent to the Parse_String function that decodes displays the string register values to the corresponding textbox / combobox.

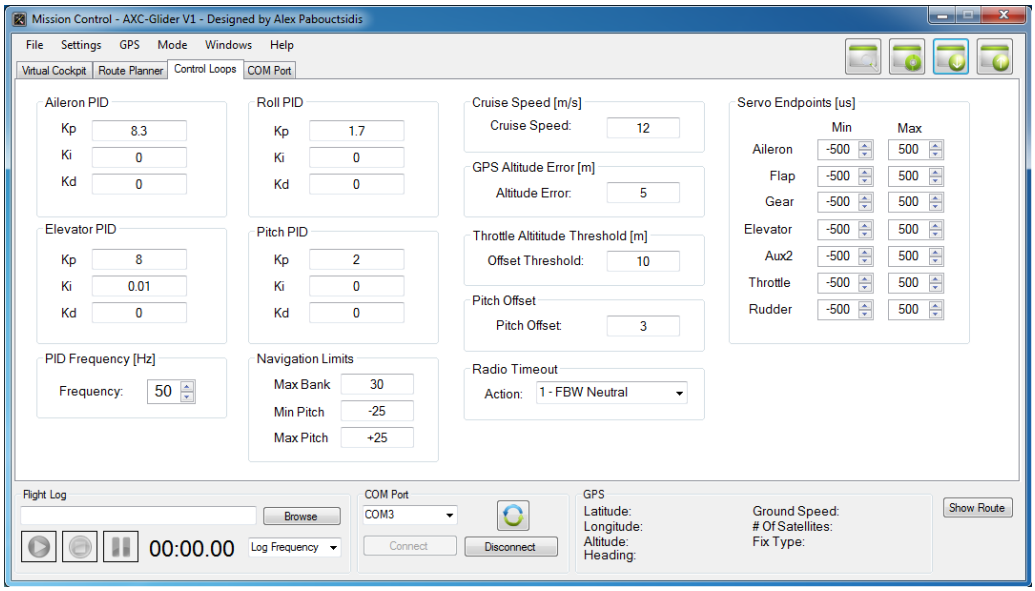


Figure 74: Control loops tab

To upload the data to the autopilot the user presses the upload button on the top left. The application then encodes the register values as defined by the AXCG communication protocol ($AXWRG,XX,...) and writes the configuration registers (09-23) one by one. After completing the writing sequence, it finishes by sending the store to EEPROM command $AXSEE*4.

9.2.5 COM PORT TAB

For debugging purpose, a fourth tab called COM port was added. This serves as a terminal emulator to view all raw data sent and received from the AXCG autopilot. The user can directly send any string using the send string text boxes. There are already some useful predefined commands, such as enabling cycle reads, changing the cycle read frequency, or changing the UI mode.

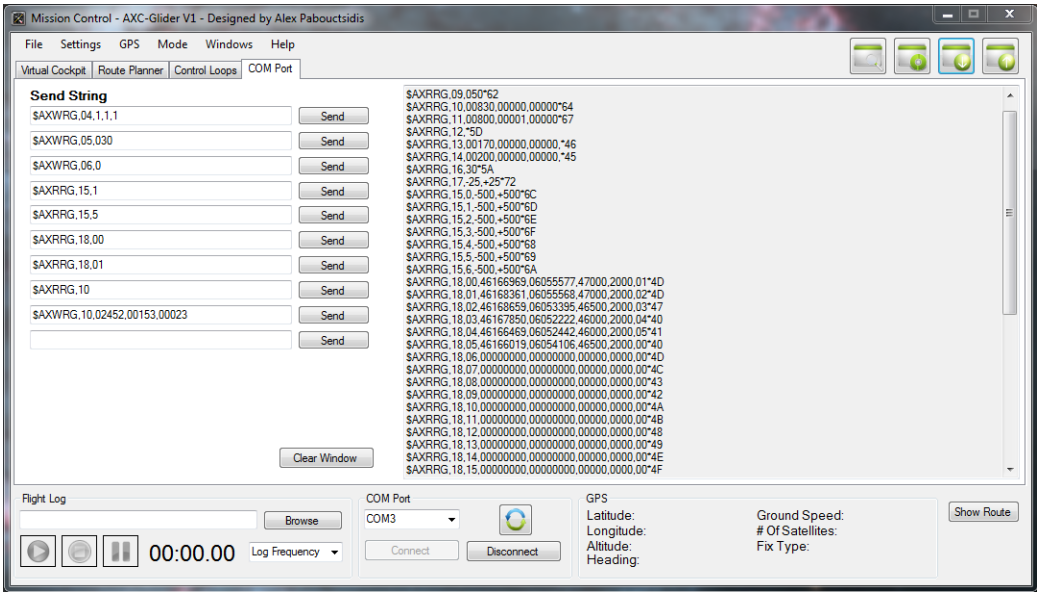


Figure 75: COM Port tab

10. Test Flights and Results

Throughout the course of this project, a great number of manual, and fly by wire tests were completed, with some of those resulting in crashes. In total two AXN Floaters were heavily damaged but reparable, and one Cularis was crashed due to a loss of control and resulted in total destruction of the fuselage.

However, with each crash, a problem was fixed, or a lesson learned and eventually the aircraft was able to fly in FBW and worked admirably well. Quickly following the successful FBW tests (as the autopilot code had already been implemented), the autopilot Auto_A and Auto_B modes were tested and completed with success.

10.1 Fly by wire test

The flight parameters for the first successful FBW were the following:

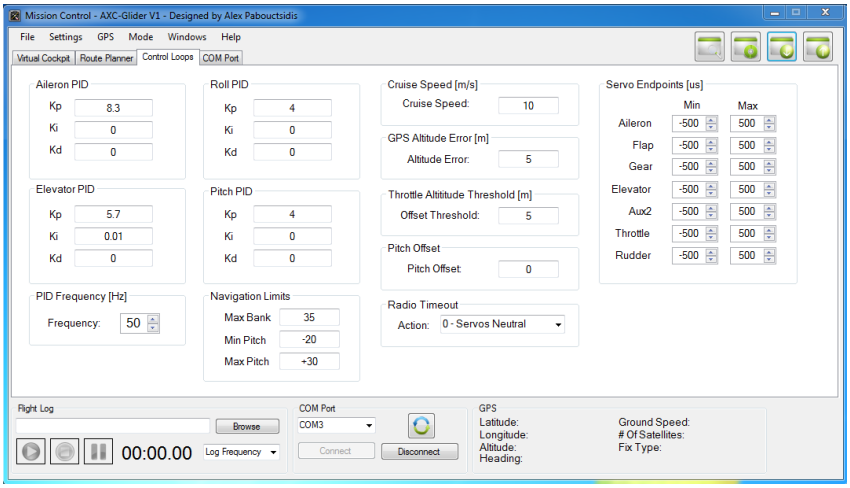


Figure 76: FBW Test - parameters

Figure 77 shows the roll and pitch data log:

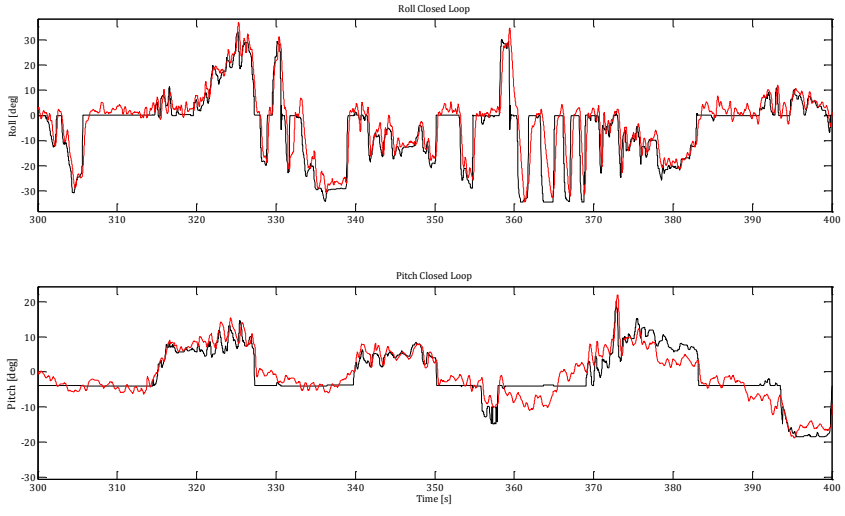


Figure 77: FBW Test - Attitude controllers

At the top we have the roll setpoint and measurement and on the bottom the pitch setpoint and measurement. The first thing we noticed is that for the roll, both curves follow each very well, indicating that the PID control is doing a good job of controlling the roll angle. We do however notice a small positive steady state error which means that the aircraft was not perfectly balanced.

As for the pitch angle, the aircraft would sometimes have a bit of trouble returning back to level flight. The elevator PID kp gain was increased to 8 (from 5.7), hoping that this would reduce the steady state error when returning to level flight.

This test validated the attitude control loops, and the next step was to test the system in autopilot mode.

10.2 AUTOPILOT MODE

The first autopilot mode test flight was done using Auto_A, which meant the throttle was still controlled manually. With a throttle set to about 40-50% the aircraft would keep its current altitude. During the flight, the aircraft would sometimes miss the waypoint and circle back, eventually reaching it and continuing on to the next (without any user intervention). The radius was then increased to 20m from 10m, and then Auto_B mode was tested for the first time.

The first Auto_B test had a slight problem with throttle control. The throttle curve was not being used, and the throttle only kicked in to 100% when the aircraft hit the minimum altitude and then back to 0% when it was above. This caused the aircraft to jump up and glide down as it navigated around the circuit.

A small fix was applied on the field, and the final test flight followed. Figure 78 shows the parameters programmed during the autopilot tests.

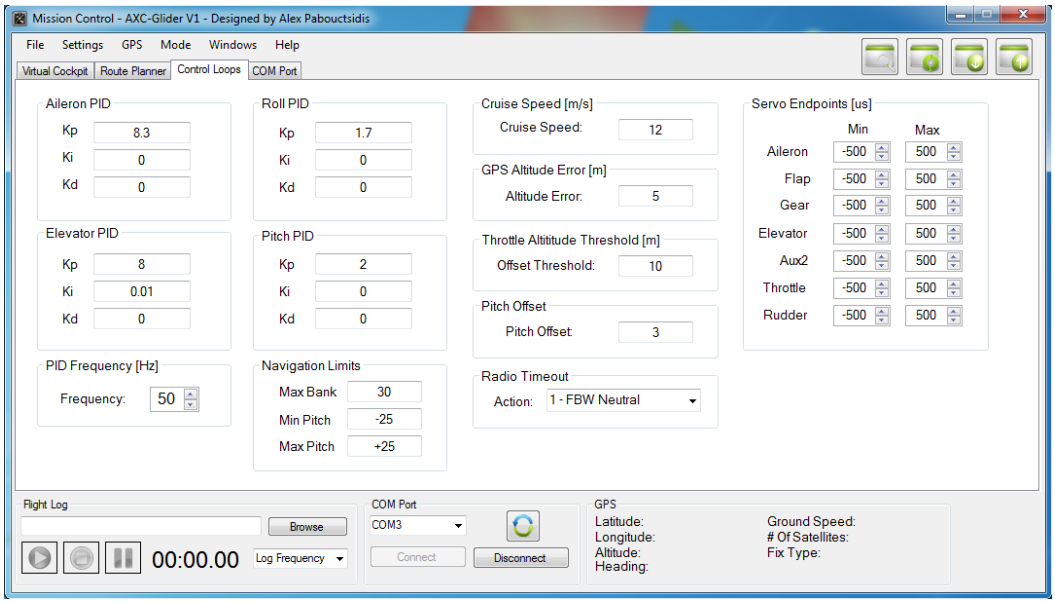


Figure 78: Autopilot Test - parameters

The waypoint entered formed a small circuit with varying altitudes, ranging from 460m to 470m. The waypoint radiuses were still set at 20m based on the change after the first Auto_A test flight.

Figure 79 shows the route planner tab with the programmed waypoints that were used for this autopilot test. The altitude was set to go up and back down about 10m, with steps of 5m between waypoints.

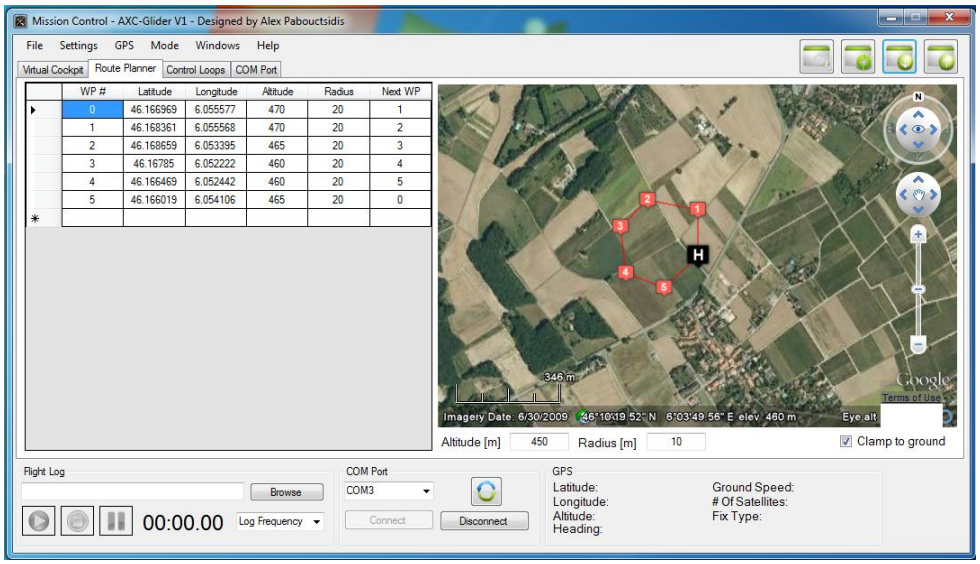


Figure 79: Autopilot Test - waypoints

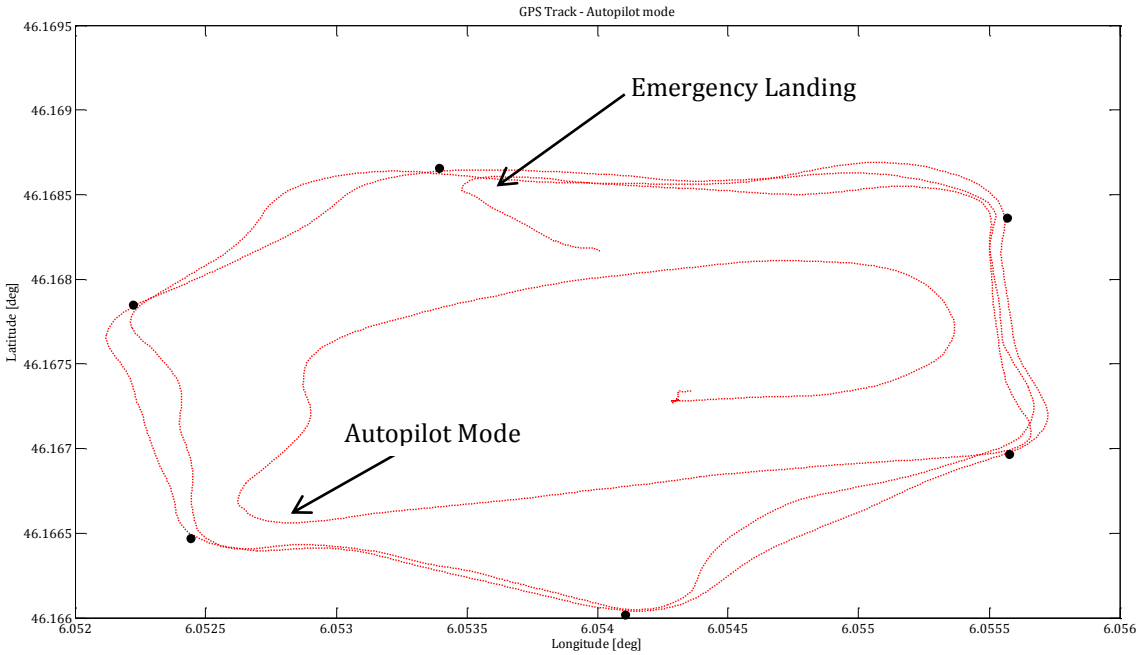


Figure 80: Autopilot test GPS track

Figure 80 shows the waypoints and the trajectory the aircraft flew during the test. The aircraft was launched manually and visually aligned towards the first waypoint. At that moment, the autopilot was set to Auto_B mode, and the rest was up to the autopilot. The aircraft corrected itself with the correct heading right way and directed itself straight to the home waypoint. Once reaching the first waypoint, the aircraft turned and leveled off in the direction the second

waypoint, just as planned. The aircraft continued to navigate the programmed circuit without any human intervention what so ever, completing two circuits. Halfway into the third circuit the battery reached a too low voltage level the speed controller automatically cut the motor, which resulted in an emergency landing.

What we can initially conclude from the GPS track data log is that the overall navigation algorithm works, and works well. Even with a cross wind the autopilot managed to fly to every waypoint, using the GPS heading instead of the AHRS heading.

However, we notice that during every waypoint change the aircraft overshoots the bearing setpoint by a small amount. This is probably due to a too high Roll Kp gain. This was mentioned in 7.6.1 and a new Kp gain was calculated which remains to be tested.

Figure 81 shows the bearing setpoint, and the GPS heading over time. For visualization purposes, the angles were unwrapped (corrects the angle by adding +360 everyone an absolute jump occurs), producing a continuous curve.

There are two important points that can be concluded by observing this graph. First of all, it confirms the earlier statement about the overshoot, as it is very clear that with every waypoint change the autopilot slightly overshoots the bearing setpoint. This should be improved with the new Roll Kp gained calculated in 7.6.1

Second, sometimes the aircraft takes a fair amount of time to reach the bearing setpoint and in some cases not far before reaching the next waypoint. There are two easy ways this could be improved. First of all, is increasing the distance between each waypoint, giving the aircraft plenty of distance to attain the proper heading. The second option would be to increase the max Roll angle. For this test the max roll angle was set at 30 degrees, which could be increased to 45 degrees. Increasing the bank angle will increase the maximum rate of turn providing tighter turns. Theoretically, the Roll PID gain would remain unchanged as the PID was calculated without taking into account output saturation in the first place, but this still remains to be tested.

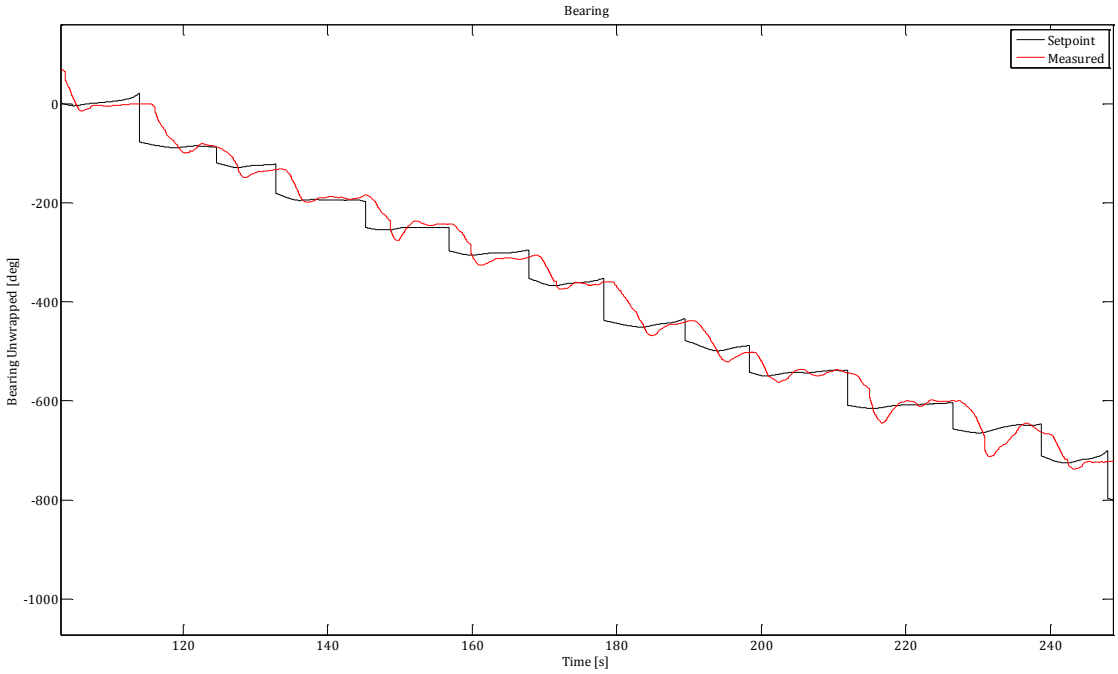


Figure 81: Autopilot Test - Bearing

Figure 82 shows the pitch and roll setpoints computed by the navigation controller, as well as the aircrafts response:

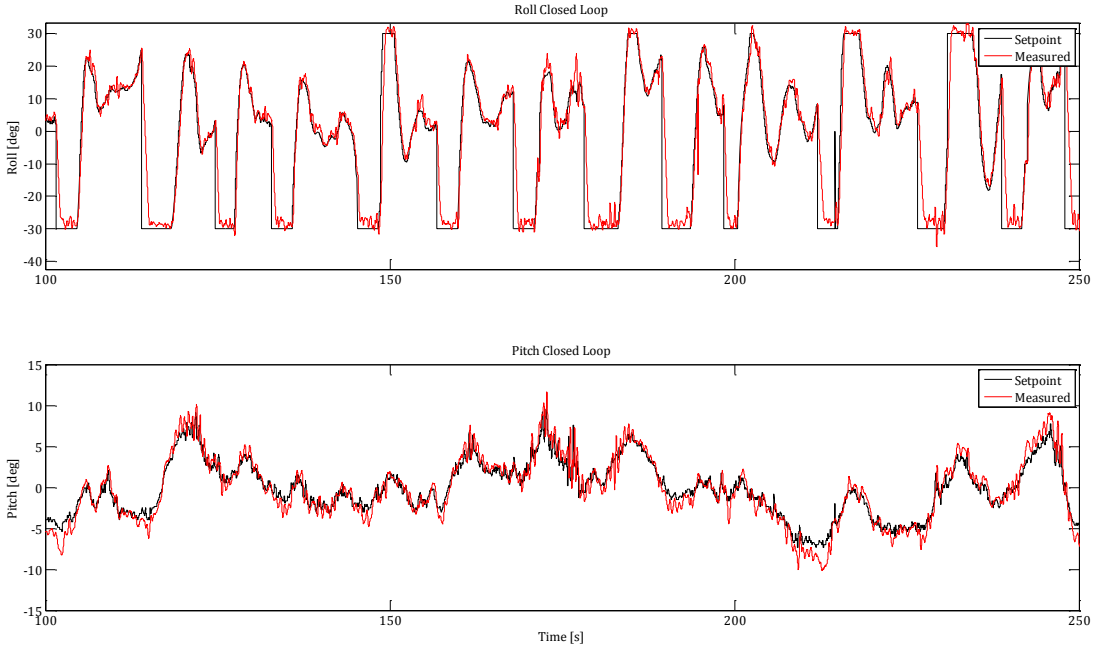


Figure 82: Autopilot Test - Attitude controllers

What we notice is the pitch tracks the setpoint much more closely than compared to the FBW test. This can be linked to how the airspeed control loops works. Since the pitch setpoint is based on the airspeed, the aircraft should always attain the setpoint. This was not the case for the FBW mode as the user could give a high pitch setpoint (15-25 degrees) which the aircraft could simply not attain with the current amount of throttle applied, as it would loss too much speed and the nose would naturally want to drop down.

The roll tracks the setpoint beautifully, which shows that the controller design process using experimental system identification was effective as the aileron PID gains were never changed to anything other than the initial computed gains.

Concerning the altitude and throttle control, there was a protocol error between the Mission Control application and the AXCG autopilot firmware, which caused the throttle threshold to equal 100m instead of the 10m entered on Mission Control. Nonetheless, the aircraft flew without any problems, albeit at a much higher altitude than what was expected.

Looking at Figure 83 we can see how the minimum altitude, the minimum altitude + Threshold, and the GPS altitude changed over time. The first thing we notice, Is the altitude remained more or less constant at about 520m ±10m. At that altitude the throttle varied between %60 and 45%, which corresponds to the equilibrium point around which the aircraft remains at its flight level.

However, with the programmed flight level changes and throttle changes, the aircrafts altitude was very slow to respond, if at all. With the current system setup, the aircraft should have gained some altitude when the minimum altitude increased, and dropped when the minimum altitude decreased, which we cannot discern in the flight log. This

might be because of the large 100m threshold value, which with 5m waypoint steps translates into small throttle steps, and a slow response.

This does prove that the separated but coupled airspeed and throttle control loops do suffice to keep the aircraft airborne and level, but is indecisive regarding waypoint altitude changes. A new test needs to be complemented with larger waypoint altitude changes, and a smaller throttle threshold to gain more insight on the throttle curve performance when changing flight levels.

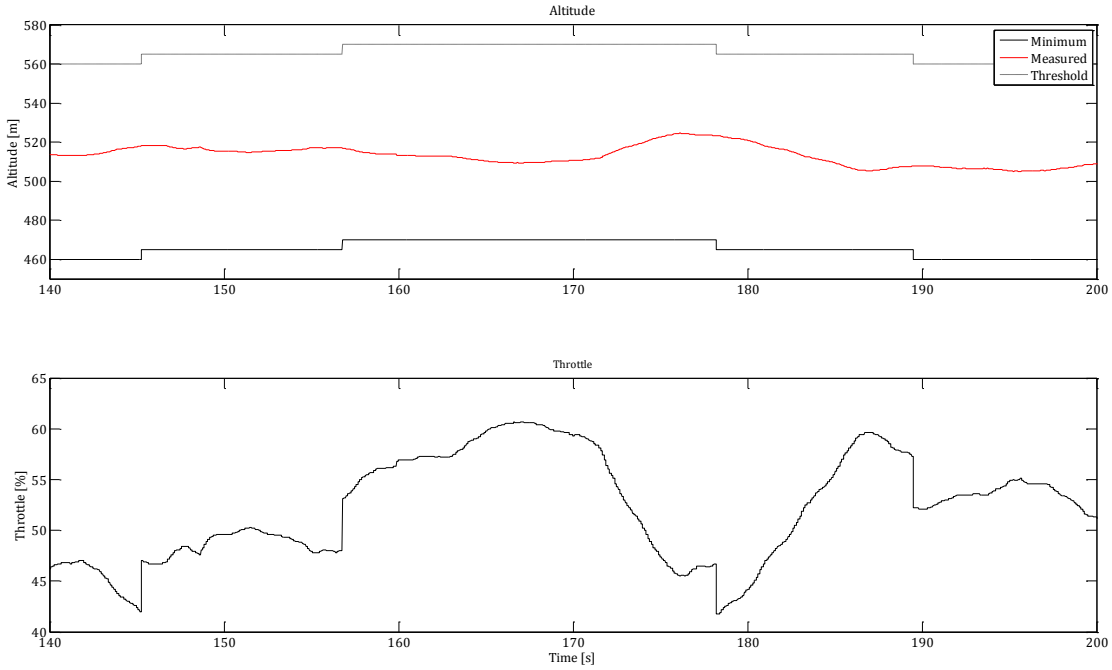


Figure 83: Autopilot test - Altitude and Throttle

In Figure 84, we can see at the top the airspeed measurement and cruise speed setpoint over time. On the bottom we see the pitch of the aircraft overtime.

The two graphs, Airspeed and Pitch, resemble each other as the pitch setpoint is a factor times the airspeed error (airspeed control loop) which is normal. We can see that when the aircraft's airspeed went above the 12m/s setpoint, the aircraft would pitch up, and when under 12 m/s it would pitch down. However, the autopilot had a hard time retaining the 12m/s cruise speed. This may be due to the throttle which acts as a perturbation for the airspeed controller, which was not taken into account.

The airspeed tracking could possibly be improved by modifying the PID gains, with gains specifically designed to be robust again external perturbations. Otherwise an integration term could be added to the PID which would improve the steady state error, which is clearly present.

Even with this steady state error, the aircraft still flew within a confined airspeed of 9-16 m/s which is sufficient to keep it airborne and flying correctly, which in the end is what is essential.

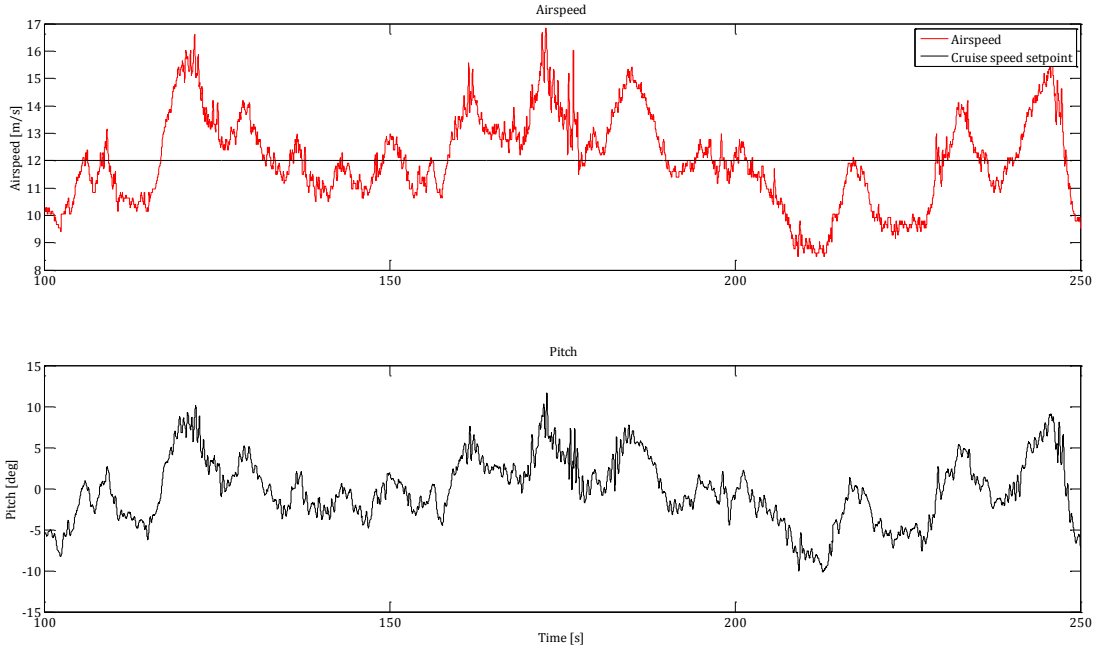


Figure 84: Autopilot Test - Airspeed Control Loop

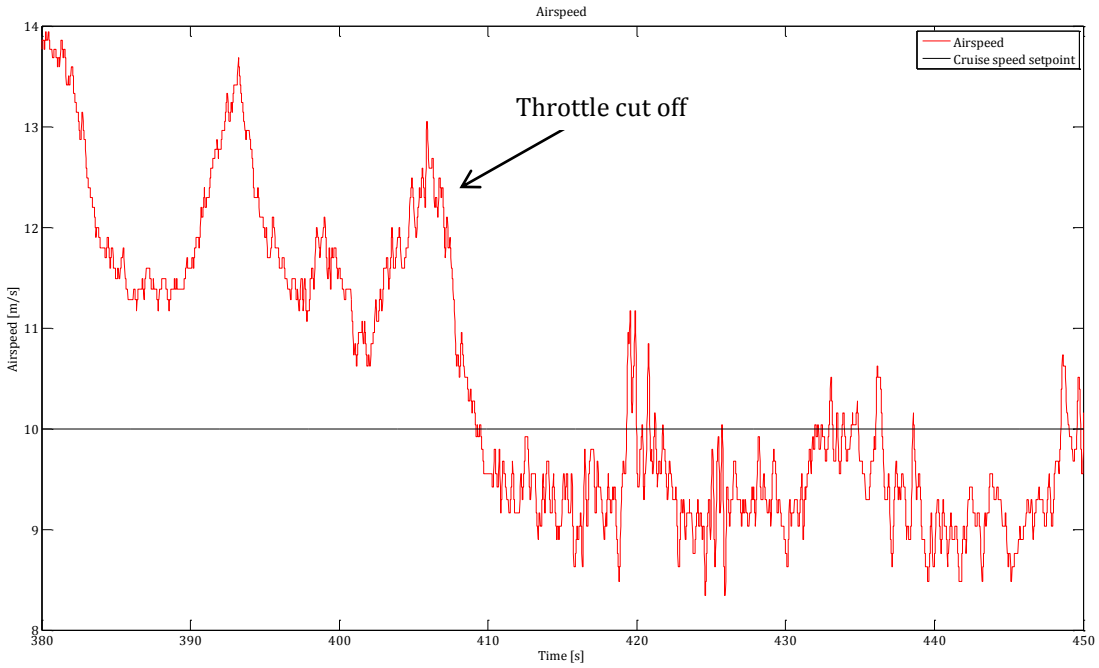


Figure 85: Auto_A Test - Airspeed

Looking back at the Auto_A flight log there were long portions of the flight where the throttle was cut off. Figure 85 shows the airspeed of an Auto_A test where the throttle was controlled manually. At ~405 seconds, the throttle was cut off the aircraft continued to navigate while gliding. What we notice is when the throttle was on (set to about 50%) the aircraft had a large steady state error of about 3-4 m/s above the cruise speed.

Once the throttle was cut off, the airspeed dropped and settled to about 9.5m/s ±2m/s. This confirms the observation of the airspeed during the Auto_B test, in that the throttle is a non-negligible perturbation for the airspeed controller, and that there is always a steady state error. The airspeed PID gains need to be adapted, by taking into consideration the throttle when computing the gains. Alternatively, the controller can be replaced with a more sophisticated polynomial robust controller.

The Pitot tube was never calibrated in a controlled airstream, which could bring the question if the airspeed is actually correct. To gain some insight Figure 86 was plotted comparing the GPS ground speed and Pitot tube airspeed.

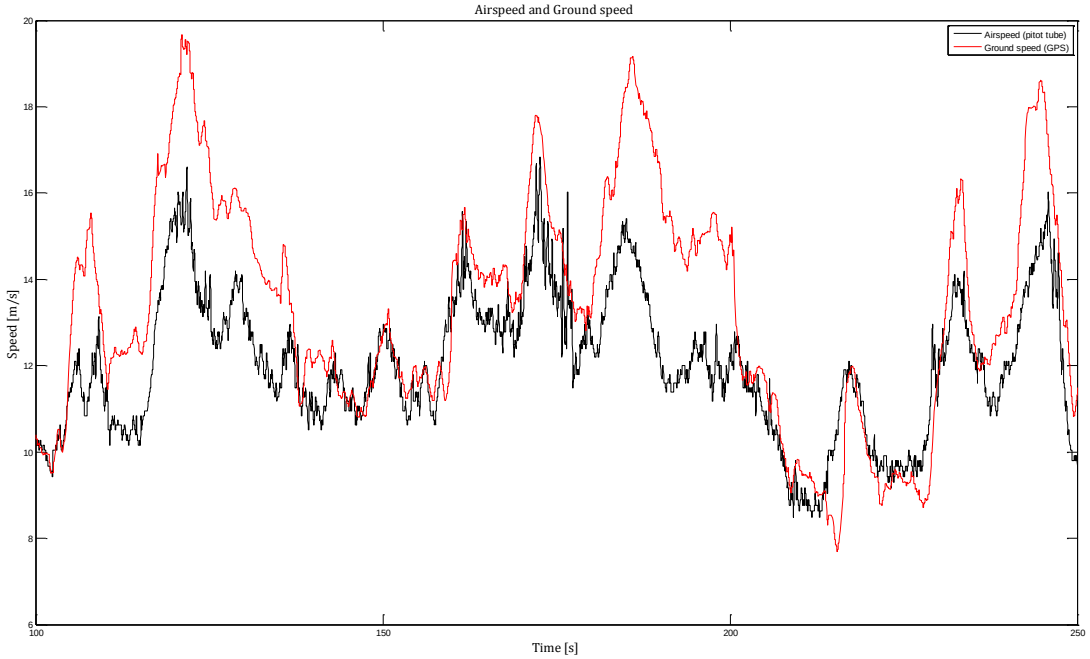


Figure 86: Airspeed vs. Groundspeed

The graph shows that the overall dynamic variations of the two follow each other nicely. There are at some points an offset up to ~3m/s, but those are most likely due to the wind that was non negligible during the test flight. At this point we assume that the Pitot tube reports an accurate airspeed reading, or is off by a negligible amount. To be full proof, a calibration test inside a controlled air flow (such as a wind tunnel) should be performed.

11. NEXT STEP

11.1 HARDWARE REVISION

Throughout this project the hardware proved to be more than adequate for the autopilot needs. No change is absolutely required, but there are always possibilities for improvements, be it for cost or functionality.

11.1.1 ABSOLUTE PRESSURE SENSOR

The only thing that was miscalculated during the initial design during the semester project was the absolute pressure sensor. This was bypassed by using the GPS to measure the altitude, but it has its limitation especially regarding accuracy.

Other absolute pressure sensors that are specifically design to accurately measure the altitude (and vertical speed) were discovered: The Bosch BMP180 and BMP085. The datasheet specifications state that these sensors can measure the altitude within 25cm, which would be perfect for the AXCG autopilot. Additionally, these are digital sensors which communicate using an I^2C bus, increasing the immunity to noise.

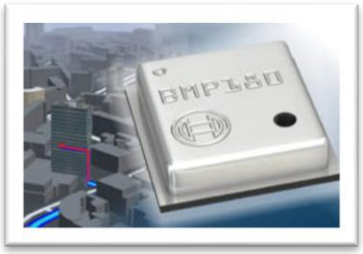


Figure 87: BMP180

11.1.2 LOW COST AHRS

The total BOM of the autopilot is around $550 (unitary price), with $500 just for the AHRS. Although the VN-100 works extremely well, is easy to configure and comes with advanced tools, finding a lower cost alternative could greatly reduce the total BOM cost. With the rapid evolution in MEMS technologies, we are starting to find chips with sensors such as accelerometers and gyroscopes with integrated signal processing, all integrated on a signal chip.

One such MEMS sensor is the Invensense MPU-6000. This sensor carries a 3-axis accelerometer, a 3-axis gyroscope and an onboard motion processor running a Kalman filter. It can also connect to an external 3D compass or magnetometer to complete the AHRS. Considering these MEMS are used in tablets and smartphones, they are very competitively priced.

Figure 88: Invensense MPU-6000

Another option would be to design the onboard AHRS from scratch using the same MEMS sensors on the VN-100. The actual BOM price is quite low; the added value is all in the digital signal processing (extended Kalman filter). Designing a custom AHRS would reduce the BOM cost significantly, but would be very difficult to implement properly.

11.1.3 ZIGBEE WIRELESS MODULE

There are still some unused pins left over on the microcontroller, including a serial Tx and Rx pin. It would be possible to add an external Zigbee wireless module with which the microcontroller communicates with using the two left over serial pins. On the receiving end would be a user and a computer with which he can view, and update the autopilot system in real time.

The XBee pro module allows a direct link between two modules with very long ranges when in line of site. Basing on the already existent PC to autopilot communication protocol, the Mission Control task can be expanded to use both the USB and the serial data from the XBee module. This would be easy to implement and require very little code change of the autopilot firmware.

Figure 89: Xbee Pro Zigbee module

11.2 AUTOPILOT FUNCTIONALITY

As stated previously, the autopilot is only programmed to work with a glider (no altitude control), so far. One major feature that could be easily implemented would be an option where the user can specify if he is working with a glider or a motorized aircraft.

When working with a glider, the airspeed is controlled using the pitch and the altitude is approximately controlled using the throttle curve, if the aircraft drops too close to the minimum altitude. When working with a motorized airplane, the airspeed would be controlled using the throttle, and the altitude controlled with the pitch of the aircraft.

The autopilot system could also be expanded with more navigation options such as loitering (circling in the air for a predesignated amount of time, before continuing on track),

We can also imagine a more intelligent system capable of making decisions on its own such as on the fly trajectory changing under certain conditions (such as an emergency landing in the closest field in the event of a low battery).

A ‘on track’ navigation mode could be added in addition to the point-to-point navigation that is currently implemented.

11.3 MISSION CONTROL IMPROVEMENTS

One feature that Mission Control is lacking is the possibility to save and load flight parameters and trajectories to a local file on the computer. The buttons to do so are already in place, but the feature was not yet implemented.

Another feature would be to open the instruments in external resizable windows, such as the GPS moving map or six pack flight instruments. The GPS moving map could also be improved to replay a flight log in 3D, with a 3D model aircraft.

In conjunction with a wireless module on the autopilot board, the program could be modified to allow a wireless connection to the autopilot to view real time data, as well as changing flight parameters on the fly (literally).

There are still some non-essential portions of the code that need to completed, such as the waypoints in the route planer that don't re-enumerate themselves when a waypoint is deleted, or the cycle read register window that still has no effect. The plane icon on the moving map when in playback mode flashes due to the continuous deleting and drawing, and could be improved.

It could also use some minor bug fixes such as the crash that sometime occurs when trying to disconnect the COM port when the autopilot is in the midst of sending data (typically when in cycle reading registers, or in sensor explorer mode).

12. CONCLUSION

During the course of the last 8 weeks, a theoretical model of an RC aircraft was computed using XFLR5 and AVL and then simulated using Simulink. Using experimental flight data and the identification tool box from Matlab, SISO systems were identified for various control surfaces and axes allowing the corresponding controller gains to be calculated.

A user PC interface with Google earth implementation was developed that allows the end user to quickly and easily change all flight parameters, such as PID gains, the waypoint trajectory, servo limits, etc. Additionally, a flight playback feature was implemented that allows the end user to replay a flight while viewing all relevant flight information such as the aircraft attitude, airspeed, position, heading, altitude, etc. all displayed on realistic aircraft cockpit instruments.

High level tasks were implemented using the RTOS on the microcontroller that handles everything from flight navigation such as the four PID control loops, to radio timeout security, to handling all PC communication via user registers and the custom defined communication protocol.

The project ended with a final flight test where a small circuit was defined using the Mission Control PC interface and uploaded to the autopilot. The aircraft flew remarkably well despite the error in the throttle threshold register value, and was able to complete the circuit three times without any user intervention before the battery died and the throttle was automatically cut off.

The next steps of the AXCG project would be to add software functionality to the autopilot such a new airplane or glider operating mode option, and extra navigation functions such as loitering, “on track” navigation, etc. To improve the altitude measurement the current absolute pressure sensor should be changed, with a more precise digital pressure sensor.

The commercial potential for radiosonde retrieval was analyzed, showing that there is a market for such a product, but a study on the theoretical success rates must be completed to determine maximum BOM cost and whether or not it is technologically and commercially viable.

13. LIST OF FIGURES AND TABLES

Figure 1: Space Balloon Freeze Frame 4
Figure 2: AXN Floater with AXCG system 5
Figure 3: Global overview 6
Figure 4: Spektrum Receiver 7
Figure 5: AXCG Autopilot System 7
Figure 6: AXCG Autopilot PCB Diagram 8
Figure 7: AXCG Autopilot PCB – TOP 9
Figure 8: AXCG Autopilot PCB – Bottom 9
Figure 9: Embedded Software Architecture 10
Figure 10: Multiplex Cularis 11
Figure 11: AXN Floater 12
Figure 12: Yaw, Pitch, and Roll rotations around body axis 13
Figure 13: Aircraft Controls 14
Figure 14: Aerodynamic moments 15
Figure 15: Aerodynamic forces 15
Figure 16: Aircraft model design flow 18
Figure 17: AXN Floater 3D wing 19
Figure 18: AXN Floater Wing skeleton and dihedral 19
Figure 19: AXN Floater Elevator 3D scan 20
Figure 20: AXN Floater Elevator skeleton 20
Figure 21: AXN main wing cross-section 20
Figure 22: AXN tail wing cross-section 21
Figure 23: XFLR5 Airfoils 21
Figure 24: Xfoil polar analysis 22
Figure 25: XFLR5 AXN main wing 22
Figure 26: XFLR5 AXN Floater Analysis 23
Figure 27: XFLR5 AXN Floater polars 24
Figure 28: AVL AXN Floater geometry plot 25
Figure 29: AXN Simulink – Top 27
Figure 30: AXN Simulink - Datum Coefficients 28
Figure 31: AXN Simulink – Aileron control derivatives 28
Figure 32: AXN Simulink - Attitude Controllers 29
Figure 33: Flight Gear visualization 29
Figure 34: Top control loops architecture 31
Figure 35: Elevator force pitch projection 32
Figure 36: Attitude control loops 33
Figure 37: Attitude and Navigation control loops 34
Figure 38: Attitude and Fly-by-wire control loop 35
Figure 39: Ailerons → Roll rate data segment 36
Figure 40: Ailerons→ Roll rate Simulation 37
Figure 41: Elevator → Pitch data segment 38
Figure 42: Elevator → Pitch rate Simulation 38
Figure 43: Aileron PID step response 39
Figure 44: Elevator PID step response 40
Figure 45: Roll closed loop data segment 42
Figure 46: Roll closed loop simulation 42

Figure 47: Roll → Bearing data segment 43
Figure 48: Roll → Bearing simulation 44
Figure 49: Pitch closed loop data segment 45
Figure 50: Pitch closed loop simulation 45
Figure 51: Pitch → Airspeed data segment 46
Figure 52: Pitch → Airspeed Simulation 46
Figure 53: Roll Root Locus 1 47
Figure 54: Roll PID step response 1 48
Figure 55: Roll Root locus 2 49
Figure 56: Roll PID step response 2 49
Figure 57: Pitch Root locus 50
Figure 58: Throttle Curve 51
Figure 59: Equirectangular projection 53
Figure 60: Attitude controller task 56
Figure 61: Read Parse Buffers Task 57
Figure 62: Navigate Task 58
Figure 63: Airspeed Task 59
Figure 64: FIR Filter frequency response 60
Figure 65: Pressure Data task 60
Figure 66: Record Data Task 61
Figure 67: Radio Timeout task 62
Figure 68: Mission Control task 64
Figure 69: Spektrum data protocol 66
Figure 70: Spektrum data stream - varying signal strength 66
Figure 71: Google earth implementation via html JavaScript 82
Figure 72: Virtual cockpit tab with flight playback 83
Figure 73: Route planner tab 84
Figure 74: Control loops tab 85
Figure 75: COM Port tab 85
Figure 76: FBW Test - parameters 86
Figure 77: FBW Test - Attitude controllers 86
Figure 78: Autopilot Test - parameters 87
Figure 79: Autopilot Test - waypoints 88
Figure 80: Autopilot test GPS track 88
Figure 81: Autopilot Test - Bearing 89
Figure 82: Autopilot Test - Attitude controllers 90
Figure 83: Autopilot test - Altitude and Throttle 91
Figure 84: Autopilot Test - Airspeed Control Loop 92
Figure 85: Auto_A Test - Airspeed 93
Figure 86: Airspeed vs. Groundspeed 94
Figure 87: BMP180 95
Figure 88: Invensense MPU-6000 95
Figure 89: Xbee Pro Zigbee module 96
Figure 90: Vaisala RS92-SGP Radiosonde ..... **Error! Bookmark not defined.**
Table 1: AXN Stability and control derivatives, and datum coefficients 26
Table 2: Task priorities and frequencies 55
Table 3: Task Processing Time 65

14. DIGITAL APPENDIX FOLDERS

DATASHEETS

Folder contains the datasheets for all the components on the autopilot board.

FIRMWARE CODE

- Drivers – low level driver source and header files
- FreeRTOS – FreeRTOS C and header files
- Libraries – Peripheral, FatSD, CMSIS, and DSP_Lib source and header files
- src - main source files
- inc - main header files

FLIGHT LOGS

One folder per flying session, numbered incrementally. Each contains all the flight logs for that session, also numbered incrementally. Flight logs can be directly imported in Matlab and sorted using the AXN_Datalog.m script.

REFERENCES

Folder contains most of the digital references in PDF format, if available.

MANUALS

Folder contains manuals for programming on the STM32, the GPS binary protocol, tuning the VN-100, and the XFLR5 guidelines.

MATLAB

- 'Ident' sessions – contain all work sessions for the Matlab identification toolbox
- Scripts – contains general scripts, for PID gains and importing the flight log
- Simulink model – contains AXN_Floater.mdl Simulink model and Load_AXN.m which contains the model flight parameters

MISSION CONTROL

Folder contains MissionControl.sln Visual C# project file, as well as GE_RP.html and GE_VC.html for Google earth files. The .html files must be placed in C:\, unless changed in the C# project.

PROGRAMS – XFLR5 AND AVL

- XFLR5 – The XFLR5 program that can be executed without installation. Project files are in 'XFLR5 and AVL Models'
- Avl.exe – AVL executional. AXN floater aircraft files (.avl and .mass) are in' XFLR5 and AVL Models'
- XFLR5 and AVL Models – contains the 3D scans in .stl and .asm formats, the AVL aircraft files and results, and the XFLR5 aircraft model

15. REFERENCES

Pabouctsidis, A. 2011, *Autonomous Cross Country Glider: Hardware design and low level embedded software*, Project, hepia.

Allenbach J.-M. 2010, *Systèmes asservis volume 4 – Asservissements digitaux*

Babister A W: *Aircraft Dynamic Stability and Response*. Elsever 1980

Deperrois A. 2010, *Stability Analysis with XFLR5*

Deperrois A. 2009, *Survol des base Aero et XFLR5*

Deperrois A. 2009, *About XFLR calculations and experimental measurements*

Deperrois A. 2011, *Guidelines for XFLR5*

http://www.movable-type.co.uk/scripts/latlong.html

http://www.diydrones.com/

http://xflr5.sourceforge.net/xflr5.htm

http://web.mit.edu/drela/Public/web/avl/

http://www.princeton.edu/~stengel/MAE331Lectures.html

http://adg.stanford.edu/aa241/stability/sandc.html