M.Sc. Thesis

Development of a UAV ground control station

Philip S. Anderson

2002

Acknowledgements

The author would like to acknowledge all those in the Flumes department at Linköping University who have provided assistance and support throughout this project. In particular, many thanks go to Cameron Munro whose enthusiasm and patience have enabled this project to be a reality. Many thanks must also go to Ulf Bengtsson who provided much assistance with the practical aspects of the project. Thanks must also go to Petter Krus for making this opportunity possible both by making the exchange possible and by providing the financial budget for the thesis. Much gratitude must also be expressed to Santi Haya Leiva for his assistance with various experimental and practical aspects of the project. The author is grateful for the assistance and patience of Pascal Spadone and Gernot Ziegler who's great computing knowledge provided essential assistance. Finally many thanks go to Monsa, Damian, Joan, Dalphine and Nicolas with who have all had to put up with the authors bad habits and provided an excellent working atmosphere.

Abstract

Unmanned air vehicles are nowadays seen as an area of great importance in the aerospace industry. An essential part of these aeronautical systems is the ground control station or GCS. This is the unit on the ground that sends and receives signals from one or several airborne units. These are normally very complicated systems that require many personnel and a lot of computing power. The aircraft design group at Linköping university is currently in the process of developing a fully functional small scale UAV. This thesis deals with the design and development of the GCS to control and analyse this aircraft. The overall design methodology has been to keep things cheap and simple. The software that has been used has enabled the development to be quick and simple. Initially the primary control for the UAV has been the integration of a commercial joystick. The main data that has been displayed on the flight data screen comes from a sensor on the aircraft known as an attitude and heading reference sensor or AHRS. The cockpit displays that have been used are ActiveX controls that require only the input from the AHRS and other sensors. These have been inserted into the main programming language that is LabVIEW – a graphical programming language. The whole system has been placed in an aluminium rack and secured to the floor of a VW Caravelle – the car in which the final system will be set up.

The project has been a success in that the feasibility of the system has been demonstrated. All available hardware components have been integrated into LabVIEW and the system performed as expected when installed in the van.

Nomenclature

AHRS	Attitude and Heading Reference Sensor
GCS	Ground Control Station
UAV	Unmanned Air Vehicle
UCAV	Unmanned Combat Air Vehicle
HOTAS	Hands On Throttle And Stick
USB	Universal Serial Bus
LCD	Liquid Crystal Display
CRT	Cathode Ray Tube
VDU	Visual Display Unit
AH	Artificial Horizon

Contents

1 Introduction

1.1 Description of task

The work involved in this project will involve the design and development of a ground control station (GCS) suitable for real time control and display of flight test data from a small-unmanned air vehicle (UAV). The system is to be designed to be transportable in a VW Caravelle van to the location of test flights, so it will need to be self sufficient in terms of power generation. It is anticipated that the GCS shall incorporate two terminals, one for the mission controller (who shall monitor the flight, and were necessary take control of the UAV), the other for real time display of flight test data. This may later be converted to a navigation terminal.

Key requirements are for the system to use off the shelf components wherever possible to keep the cost down and minimise the amount of work required. Initially it is anticipated that that the following will form part of the system:

- 2 x TFT monitors (to keep down weight, space and power consumption)
- 2 or more PCs for flight control, data acquisition and display
- Commercial joysticks for flight control
- Rack to mount system and allow ready mounting and removal from the van

The key problem to be resolved will be the interfacing of the different hardware and software components in the system. It will be necessary to start from a simple baseline system and build up to a fully functional system, identifying and solving problems as they occur.

The basic software will be through the LabVIEW data acquisition, control and visualisation software. This is a graphical programming language for which a number of libraries exist to allow the rapid development of the software interfaces and GUI. It is anticipated that the most flexible solution to the visual front-end of the system will be ActiveX controls, such as the aircraft instrumentation controls available from Global Majic Software (www.globalmajic.com).

The vision is to have a system where the mission controller has a complete cockpit display with all necessary aids with which to fly the aircraft remotely. These would include basic instrumentation and warning indicators such as;

- Basic control instruments:
    - Artificial Horizon
    - Turn Coordinator
    - Compass
    - Airspeed Indicator
    - Altimeter

- Warnings and system monitoring
    - Exceedences (airspeed, AoA, sideslip, engine temp, engine RPM, acceleration)
    - Fuel or batteries
    - Timer

Furthermore, it is anticipated that a live video feed will be incorporated, although it is not known how useful this will be as a flying aid. Other possibilities include a data-driven "virtual view" of the flight vehicle, indicating its current angular position.

The onboard sensor package consists of an air data system (pitot static system for airspeed, alpha and beta vanes for angle of attack and yaw measurement, Watson AHRS (attitude and heading reference system) which allows inertial measurements (angular rates and accelerations) and longer term, a radar altimeter. These sensors provide data to an onboard data acquisition, control and telemetry system based on PC104 architecture. This data is then relayed to the GCS. The exact definition of the telemetry system is as yet undefined, although it is reasonable to assume that it will consist of RS-232 links.

The system should enable Level 3 autonomy to be achieved through the GCS. This means that the system should allow everything from human-in-the-loop manual control to autonomous flight along predefined waypoints.

Key interfaces that should form part of the initial project work include:

- Joystick-computer-LabVIEW-control uplink
- ActiveX controls in LabVIEW
- Incoming telemetry data-LabVIEW

While the software component will clearly be a major part of the work, selecting and integrating the necessary hardware will also be a necessary task. In particular, it is important to realise that there is always a delay between ordering and receiving delivery of hardware so this must be considered in the overall time plan.

It should be noted that this project forms part of an on going development effort and hence there are areas that do not form part of this project – such as definition and development of the telemetry system, which will be conducted in parallel with this project.

1.2 Key Design Considerations

The overall emphasis on the design requirements during this project was to keep things cheap and simple. In terms of hardware, at all possible opportunities, components that could be reused from existing hardware in the department should be. This has lead to a final system that is maybe not as elegant as it might be, but also one that is relatively cheap. The software 'LabVIEW' that has been used was chosen due to its quick and flexible application, particularly for new users. In addition to the main programming language, simple ActiveX add-ons have been used to simplify the software even further. The result has been to create a prototype system that has been relatively quick to develop and has been very cost effective.

1.3 System overview

The ground control station will for part of a larger UAV system in ongoing development at LiTH. As well as the GCS, a large part of the system is the airborne part. The whole system can be seen in schematic form in fig 1.2. The airborne part will most likely be housed in an aircraft named LUCAS (Linköping University Combat Air System). This is being continually developed with in the department. An early version of this aircraft can be seen in fig 1.1. The schematic in fig 1.2 shows that there are certain data acquisition devices all of which need to be incorporated in some way in displays in the ground control station. As well as the standard analogue sensors there will also be an Attitude & Heading reference system (AHRS), which measures various angular positions and rates. This data is all collected and processed by the airborne hardware on a PC104 set-up. This data is then transmitted down to the GCS using some, as yet, undefined telemetry link. Although nothing has been confirmed about the telemetry subsystem at this stage it is being assumed that it will be using an RS232 connection though the serial port of the PC. Three monitors are shown to display the aircrafts data, although initially this thesis will be concentrating on the flight display and data display. Input for the GCS will come in the form of a commercial HOTAS throttle and joystick, the connection for which will be either USB or gameport. The joystick coordinates should be taken by the GCS and transmitted to the UAV using the same telemetry system as the aircraft data.

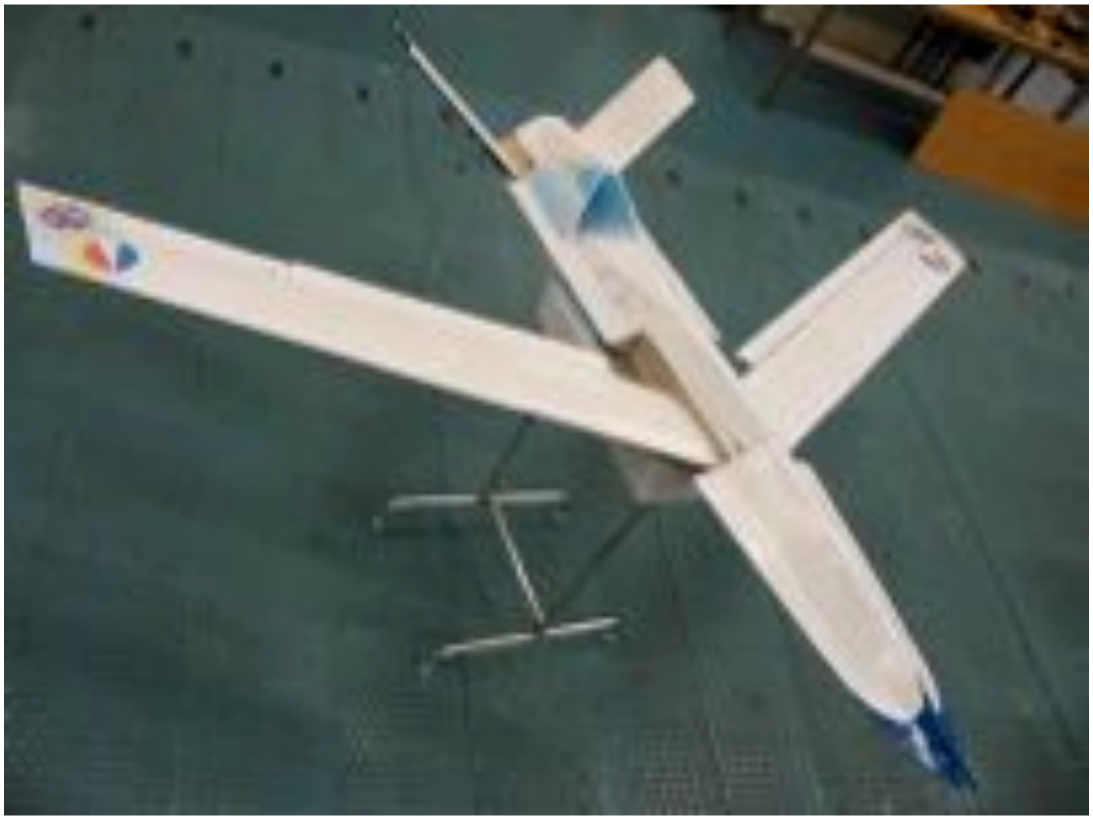


Fig 1.1: LUCAS aircraft

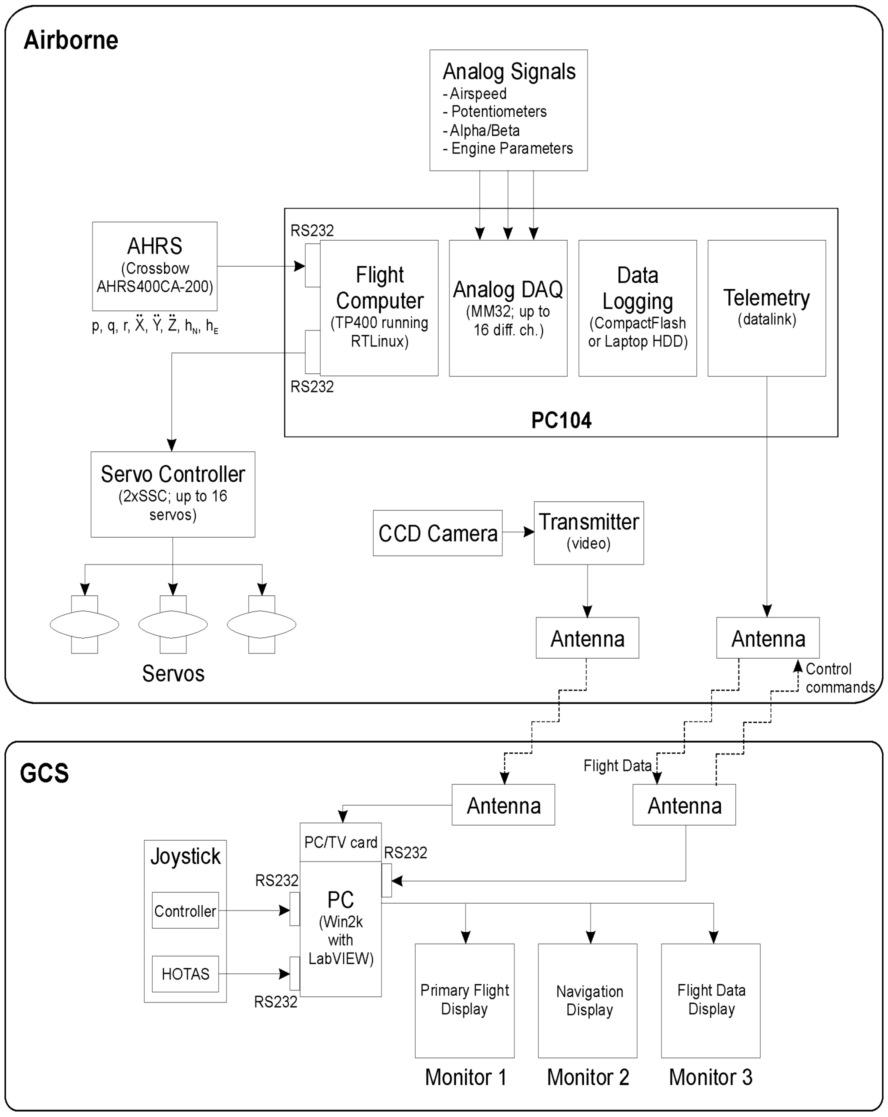


Fig 1.2: Schematic diagram of complete system

1.3 Other Components in Development

There have been a number of other components developed in parallel with this GCS. The first of these is the auto pilot system that is located in the aircraft. This will hopefully provide automatic control of the aircraft when the joystick is not controlling it. At the same time as this thesis was being carried out, another thesis was being undertaken to develop the aircraft instruments and datalink subsystem. The component that probably has the most relevance to the design of the GCS is a flight simulator. It is hoped that in the future, the GCS will be able to control this simulator in exactly the same way as it controls the actual aircraft.

2 Background

2.1 Role of a GCS

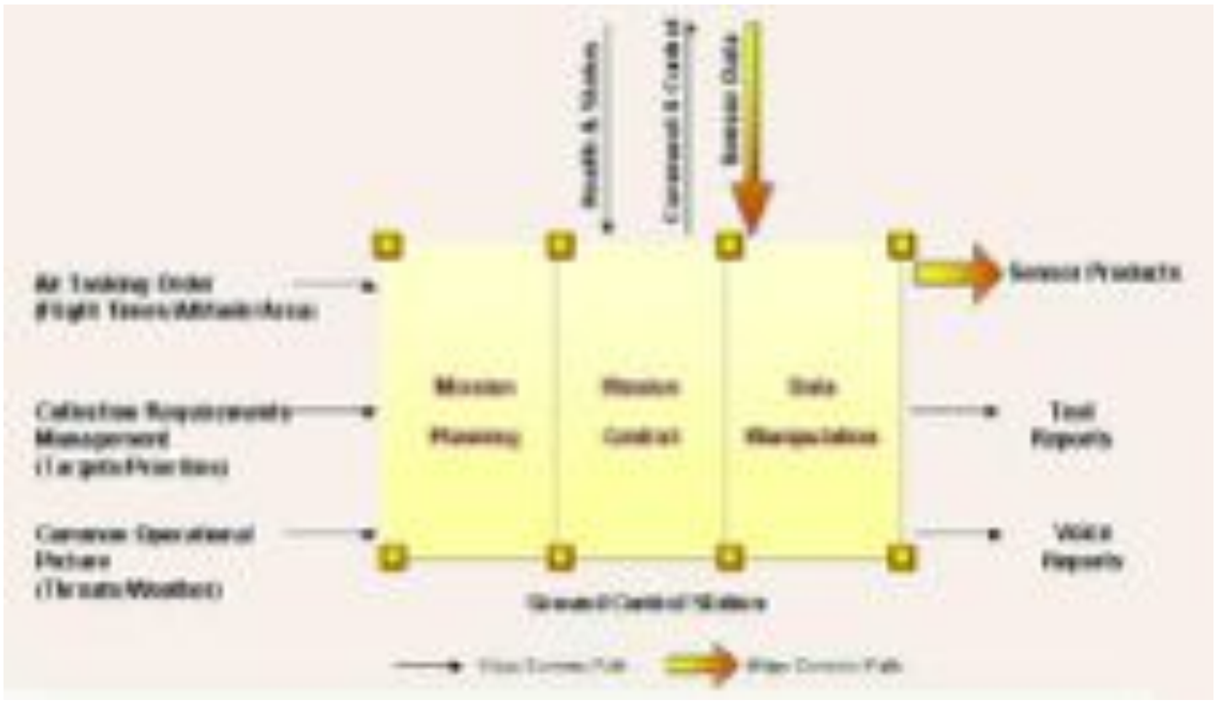


Fig 2.1: Typical GCS functions

The ground control station (GCS) is the hub of an unmanned aircraft system. It is what processes the incoming data and sends control instructions to the aircraft. A typical military ground control station (GCS) will envelope three main functions: mission planning, mission control and data manipulation.

- Mission planning: Will be involved with identifying and prioritising targets. This function will be involved with examining the sensor capabilities compared with the selected targets. The communications available will be identified in this function and sensor use identified. A map of any potential threats and an analysis of the weather will commonly be carried out. From this information a flight plan will be developed.

- Mission control: Will involve all necessary intervention while the aircraft is on the ground. It will control the launch of the aircraft and then be responsible for the aircrafts navigation and any mid flight rerouting. This function will be responsible for the monitoring and control of any payload. Finally this section will be responsible for the recovery of the aircraft.

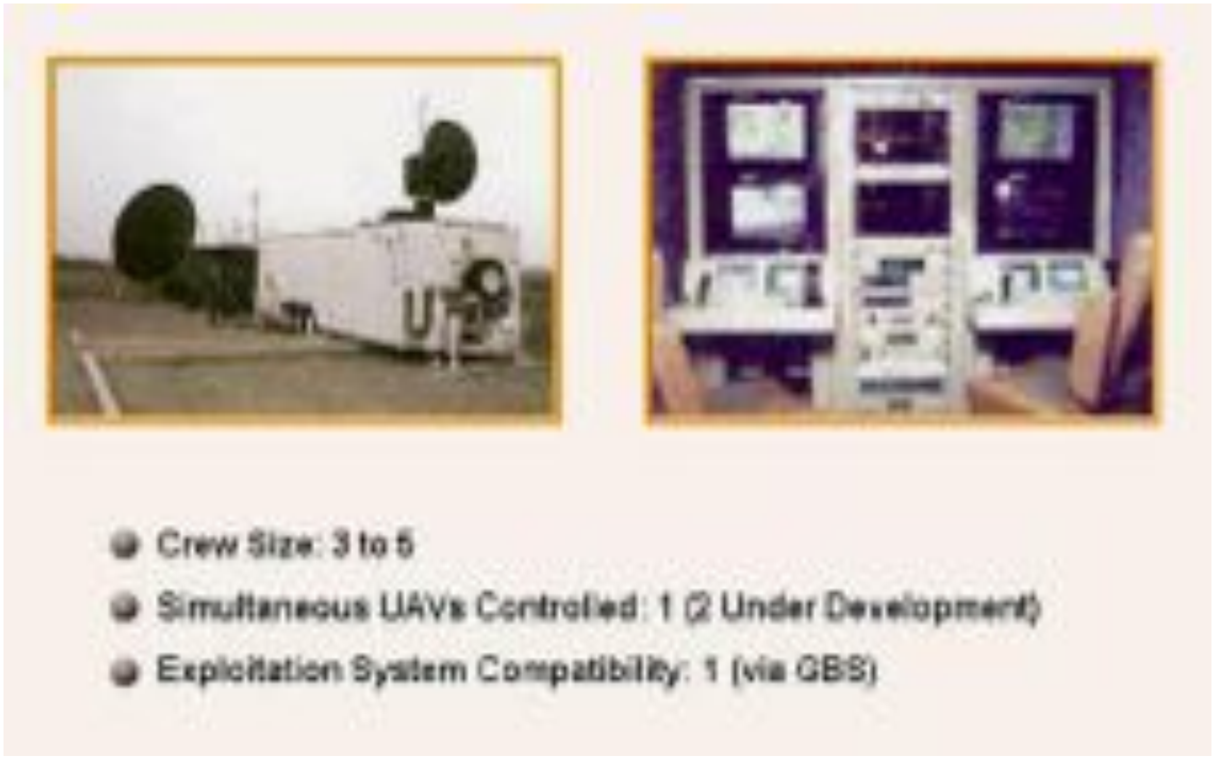
- Data manipulation: it is assumed that on most UAV missions, some form of data will be collected. This will need to be processed, exploited and archived during flight and/or post flight. The data manipulation function therefore forms an integral part of most UAV GCS functions.

For small scale UAV's, such as the LUCAS, the mission planning function is not such a dominant part of the overall system. The details of the sub-functions in the mission control function will depend on the size, function and level of autonomy in the UAV in question. There are three levels of autonomy in UAV design. These are as follows;

Piloted Flight	Example
o Direct	MAV's
o Augmented	Pointer
o Automated Phases	Pioneer/UCARS
• Autonomous Flight	
o Preprogrammable (only)	CL-289
o Reprogrammable	Global Hawk
o Onboard decision making authority	UCAV
• Cooperative Flight	
o Wingman	No known UAV's at present
o Similar aircraft formations	"
o Dissimilar aircraft formations	"

2.2 Examples of current military GCS's

2.2.1 Predator System

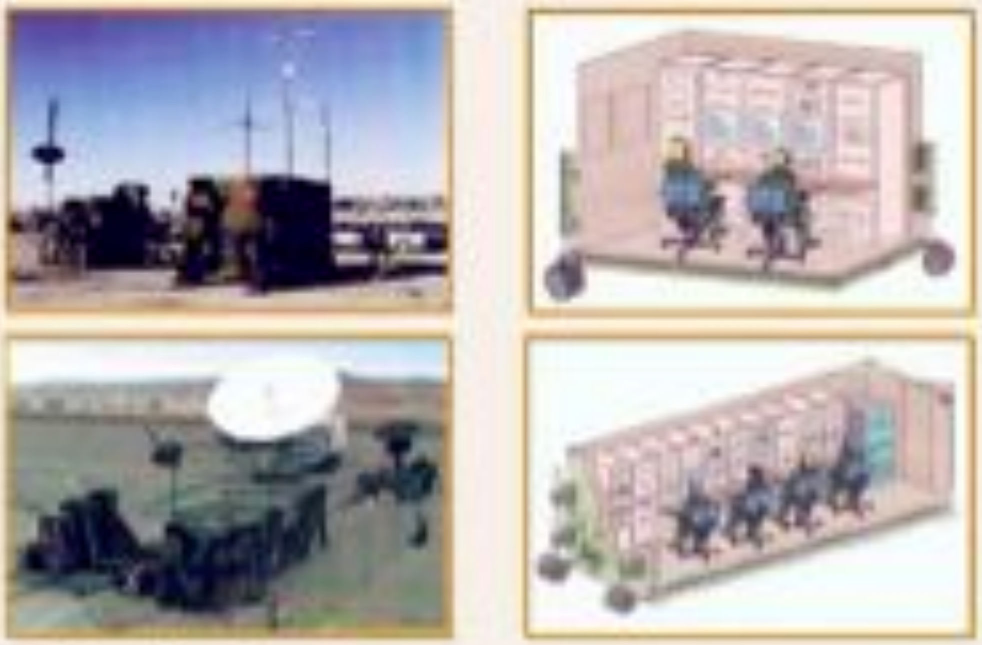


- Crew Size: 3 to 5
- Simultaneous UAVs Controlled: 1 (2 Under Development)
- Exploitation System Compatibility: 1 (via GBS)

Fig 2.2: Predator GCS

A pilot in a ground control station flies the Predator remotely. In addition to the pilot, there are three sensor technicians on duty monitoring the flow of information from the vehicle. This information comes from three sources, a TV camera, and infrared camera and onboard radar. The radar can be operated simultaneously with either the TV camera or the infrared camera, depending on visibility conditions. There is a C-band line-of-sight data link or a Ku-band satellite data link for beyond-line-of-sight operations. It is more correct to think of the predator as a system rather than an aircraft. The complete system consists of a ground station, four UAVs, the main satellite dish for the ground link and a team of operators. The UAVs themselves can be broken down into small transport containers, while the ground station module can travel in a C130. The aircraft are powered by the turbocharged four-cylinder Rotax 914 engine that is used on some more expensive microlight (ultralight) aircraft.

2.2.2 Global Hawk System



- Crew Size: 6
- Simultaneous UAVs Controlled: 3 (Under development)
- Exploitation System Compatibility: 3

Fig 2.3: Global Hawk GCS

The Global Hawk's ground stations include the Mission Control Element (MCE) and the Launch and Recovery Element (LRE). The MCE is the Global Hawk's ground control station for operations involving reconnaissance. It consists of four workstations: mission planning, sensor data and processing, air vehicle command and control operator (CCO), and communications. The

Mission Commander is the fifth crewmember, responsible for overall mission management. The LRE includes a mission planning function as well as air vehicle command and control. During operations where the two units are split, the senior operator will function as mission commander until air vehicle control is taken up by the MCE.

2.3 Smaller Systems

The previous two examples have illustrated two large, complicated systems. There are also a number of much smaller UAV systems. One example of one of these smaller systems is the Pointer. This is a hand launched reconnaissance aircraft. The GCS that controls this aircraft is a one man operated portable system. It is designed to support manoeuvre battalion commanders or other military personnel users needing short-range intelligence. This UAV has no autonomous ability. The pointer GCS can be seen below in fig 2.4.

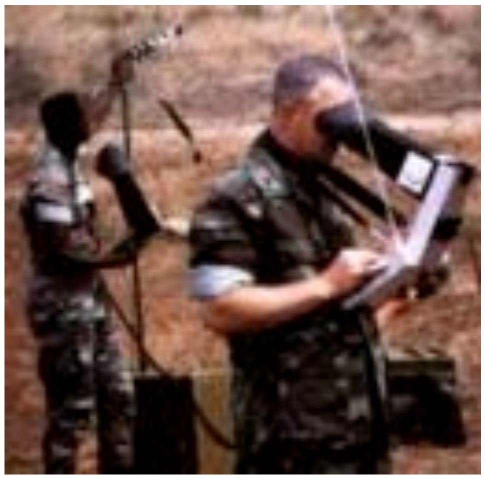


Fig 2.4: Hand held Pointer GCS

2.4 Summary

There are numerous UAV systems currently on the market ranging from small model aircraft to full sized, fully autonomous jet aircraft. Most larger UAV's are being developed with a military perspective in mind. All UAVs must have a ground control station of some sort. Many ground control stations can be said to perform some or all of three main functions: Mission planning; Mission control; and Data manipulation. The level to which each of these functions are implemented in a particular system, depends a lot on the level of autonomy in that system. Two examples of large-scale military UAVs currently in operation are the Predator and the Global Hawk. By looking at a range of UAVs it can be seen that the system mission and the level of autonomy dictate the GCS architecture.

3 Power Budget

3.1 Calculations

Initially it was considered that all displays would be displayed on two VDU's. As money was always a concern, it was decided to initially work with CRT displays as opposed to LCD displays. That way, hardware already in the department could be re-used. To start with, an initial guess of 80 W was made for the power consumption of 1 monitor and it was known that the CPU consumed 60 W of power. It should be noted that while CPU usually refers to the actual chip, for simplicity CPU in this thesis will from now on refer to the whole of the central box and everything in it. With two monitors that is a total of 160 W for both monitors. This is assuming 17" monitors as it should be noted that size of the screen very much dictates the power consumption. The power would be supplied from one or more DC car batteries. It was thus realised that some form of transformer/s would be needed. Both the monitors and CPU normally run off of a 230V AC supply. So it may seem initially that only one transformer was needed. However, the CPU then has another internal transformer that converts the 230V AC to a multi-voltage DC supply. It was thus realised that the efficiency would be higher if two transformers were used: one for the monitors and one for the CPU, which would replace the internal one. A summary of the transformers used is thus as follows:

- A CPU DC-DC transformer for the CPU with an estimated efficiency of 70 %
- A DC-AC transformer (12V –230V) for the monitors with an estimated efficiency of 85 %

Including these efficiencies, the power consumption in the monitors and CPU were thus;

$^{160}\!/_{0.85} + {}^{60}\!/_{0.7} = 188.2W + 85.7W$ Respectively.

Initially the option of a 12V battery supply was analysed. With a 12V supply, the current drawn is as follows;

- Current from the monitors is $^{188.2}\!/_{12} \approx 16A$
- Current from the CPU is $^{85.7}\!/_{12} \approx 7A$

It was desirable to reduce the current consumption further in order to prolong the endurance. This could be done by increasing the voltage. However it was only practical to do this for the CPU supply. It was easy enough to obtain a DC-DC transformer with an input of 24V as this was the standard. However the university owned a standard AC transformer with an input of only 12V. It was seen as too expensive to buy a new 24V one so the voltage was only increased for the CPU

- Current from the monitors is $^{188.2}\!/_{12} \approx 16A$
- Current from the CPU is $^{85.7}\!/_{24} \approx 3.6A$
- Total current $\approx 20A$

It was then necessary to work out the endurance of the system, but first, the actual current drawn was measured, so the endurance would be a more accurate value than the purely theoretical value. The experimental results of the current drawn are as follows;

CPU

On start up the CPU draws 4A
After that it draws 3.6A

MONITORS

One monitor draws 12A off of one battery

This gives a total, actual current draw of 28A. The batteries that will initially be used for testing have a capacity of 60Ah when new. Unfortunately as only one AC transformer was available, both the monitors must be connected to one battery. This therefore means that the endurance is not twice the capacity of one battery as most of the current is being drawn from only one battery. As the current drawn by the monitors is far greater than that drawn by the CPU and to add a factor of safety it was decided to calculate endurance based only on one battery. Thus the theoretical and measured endurances are as follows;

	Theoretical	Measured
Current	20A	28A
Endurance using 60Ah batteries	3 hours	2 hours 9 mins

It has been agreed that a condition of 3 hours endurance is required. It can be seen that if the measured values are used then this does not give a long enough endurance. With a current of 28A and a required time of 3 hour, the required battery capacity is 28 x 3. This gives a battery of 84Ah. The closest battery to this value, available from Biltema is an 88Ah battery. It is thus recommended that this be the battery that is used in the final design

3.2 Conclusion

It has been agreed that a condition of 3 hours endurance is required. Therefore for the final system larger capacity batteries will be required. Either that, or more batteries will be required to be wired in parallel over the monitor section. However as weight is a concern, it is suggested that larger capacity batteries should be used in the final system. However, for the testing of the system for this thesis the current batteries have sufficed. The final set-up can be seen in figure 3.1 below.

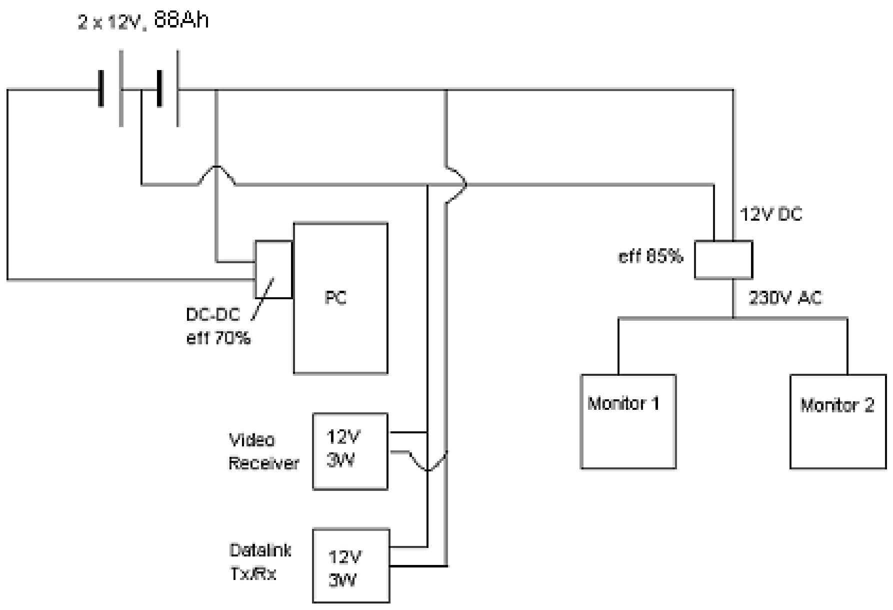


Fig 3.1: System wiring

4 Hardware

4.1 VW Caravelle

The GCS will need to be placed and secured inside a VW Caravelle. Therefore as part of this thesis a support rack will need to be designed and built. There are certain criteria and limitations on the design of this rack. Firstly the rack must be quick to take in and out of the van. This is due to convenience requirements and time constraints on the availability of the van. However a more limiting constraint is the size of the van. The dimensions of the van can be seen in fig A.1. One of the most crucial dimensions it can be seen is the size of the door.

4.2 PC

To start with, one PC will be used in the set-up. This PC has a Pentium II 333MHz CPU. It will be running windows 2000, and the software being used to develop the GCS is a graphical programming language called LabVIEW. LabVIEW has been widely adopted throughout industry, academia and research labs as the standard for data acquisition software.

The single CPU unit is being used to display the system graphics on 2 CRT VDU's. For this it has been required to insert a second graphics card into the PC. This has simplified the system in terms of the number of components and the amount of software. A slight disadvantage if that it takes slightly more computer power than with a single graphics card. It was decided that the benefits in terms of weight and simplicity out weighed the performance gains from using multiple CPU's

4.3 Power

For the purposes of testing in this thesis two 12V 60Ah batteries are supplying the power. Although it is recommended that 88Ah batteries should be used when the system is implemented for real. To supply power to the two batteries an 800W DC-AC converter has been used. The input to this is fixed at 12V so both monitors are powered from one of the batteries. To power the CPU unit a 24V DC-DC transformer has replaced the internal power supply. This requires the CPU to be wired across two of the 12V batteries in series.

4.4 AHRS

An attitude and heading reference sensor measures the majority of the aircraft data that is to be displayed on the GCS. This is manufactured and supplied by Watson industries. For the purposes of testing the GCS system, this has been connected directly to the COM port using RS232 wiring.

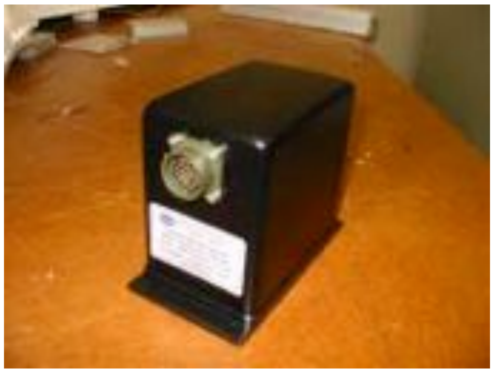


Fig 4.1: AHRS

4.5 Mounting rack

So that the GCS system can be mounted securely in the Caravelle, a supporting rack has been constructed. As money was a concern, this has been constructed mainly from scrap material found in the department. It has been constructed from “FlexLink” aluminium bars. These are particularly practical due to the fact that all dimensions of the rack can be easily and quickly adjusted. The rack was constructed using the dimensions that can be seen in appendix A. It has been design to be fixed to the floor of the car by taking out a row of seats and fixing it using the bolts that normally hold the seats in place. One of the most important features of the rack was that it should secure the monitors very well, so that they have no chance of moving when driving. The completed rack can be seen in figure 4.2 below.

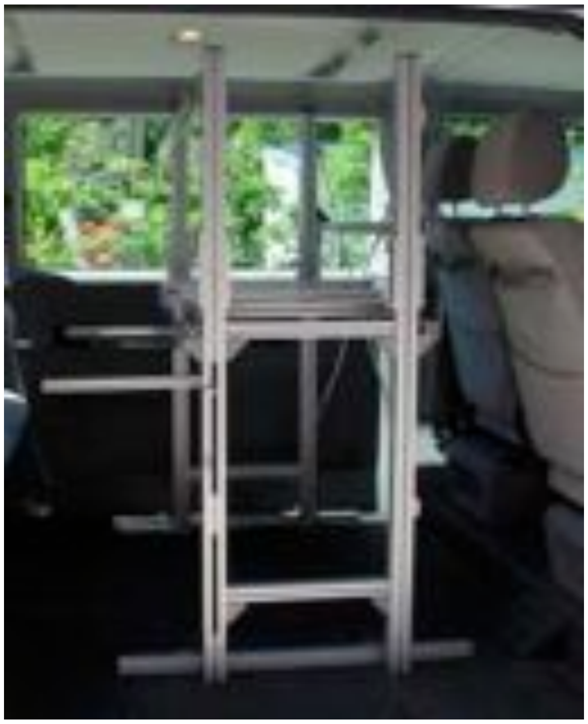


Fig 4.2: GCS support rack

5 Cockpit Displays

There is a general accepted way of thinking that says there are six main instruments, which give an overall picture of the aircrafts flight condition. These six primary instruments are airspeed indicator, altimeter, artificial horizon, direction indicator, vertical speed indicator and turn and bank indicator. It is important to group these instruments in the right way in order to let the pilot clearly interpret the situation without confusion. Over the years many different layouts have been experimented with, however there is now one accepted layout, which all modern aeroplanes adopt. This is known as the basic T and can be seen in fig 5.1. This is based on the assumption that out of the above six instruments the following are seen is the most important; airspeed, artificial horizon, altimeter and course indicator

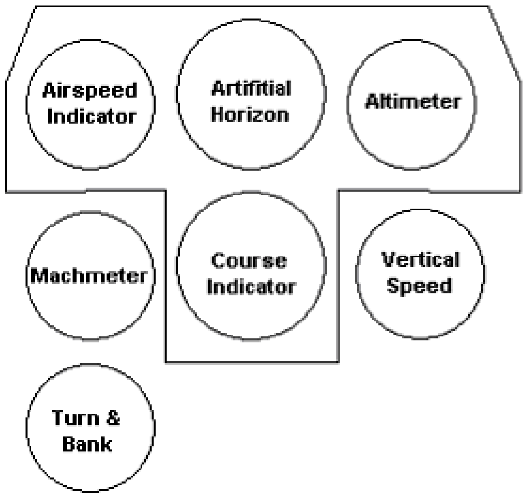


Fig 5.1: Basic T layout

The AH occupies the central position due to the fact that it displays attitude in all axis. It is therefore used as the primary instrument. Since airspeed and altitude are directly related to attitude that can be seen on the AH then these indicators lie directly beside the AH. Change in direction is initiated by banking the aircraft. To see the change in direction, some sort of course indicator is required. This instrument supports the interpretation of roll attitude, and is therefore placed directly below the main display of roll; the AH. This layout can be seen in nearly all aircraft with a traditional (non glass) cockpit (Fig 5.2).

5.1 Indicators Descriptions

5.1.1 Artificial Horizon

This indicates the pitch, bank and heading attitude of the aircraft. Pitch, bank and heading attitude are represented by one moving element. This is a surface that symbolises the natural horizon. This moves in three axes to indicate the change in all three parameters simultaneously. A fixed horizontal line on the indicator represents the aircraft. In a traditional aircraft, this indicator employs a displacement gyroscope whose spin axis is maintained vertical by a gravitational sensing device similar to a pendulum.

5.1.2 Airspeed Indicator

This is effectively the display of a very sensitive pressure gauge that measures the difference between pitot and static pressure. To avoid high-speed errors, present day airspeed indicators are calibrated to the law that includes a compressibility error;

$$p = \tfrac{1}{2}\rho V^2 \left(1 + \tfrac{1}{4}\frac{V^2}{a_0^2}\right)$$

This is an indicator that is directly driven from pressure measurements and therefore from the air data system.

5.1.3 Heading indicator

This is an indicator that shows the aircrafts direction as a magnetic compass bearing. In traditional aircraft it will often show information regarding radio navigation systems such as VOR or ADF. However the system that is being developed with this GCS is too small a scale for this information to be useful, initially anyway. The basic indicator being used in this case consists simply of one moving part. The bearing indications move around a fixed symbol that represents the aircraft, so that the direction of travel is always at the top. A similar indicator can be seen in the Robin DR400 (top left fig 5.2).

5.1.4 Altimeter

This is a pressure-sensing device that requires direct input from the pressure sensing system. The display consists of one rotational pointing device that moves in full circles around a fixed set of indications. One full rotation of the pointer represents 10000 ft. There is also a digital read out in the centre of the display.

5.1.5 Vertical Speed indicator

Also known as a rate-of-climb indicator, this is the third of the primary group of pitot-static flight instruments. This is another very sensitive pressure gauge. It is designed to indicate the rate of altitude change from the change in static pressure alone.

5.1.6 Turn coordinator

The turn co-ordinator shows the yaw and roll of the aircraft around the vertical and longitudinal axes. The airplane symbol shows the rate of turn, not bank angle. If the airplane symbol is on the first line this is called a standard rate turn, and means that the aircraft is turning at 3 degrees per second. This is particularly important information for instruments flying in the clouds and having to hold or make timed turns.

The ball or inclinometer, is like a ball on the end of a string, if a turn is made with too much rudder then the ball (and occupants) feel thrown sideways (a skid), and with insufficient rudder the ball would fall into the turn (slip). Using rudder and the inclinometer means that turns can be smooth and the occupants do not feel thrown one way or another, like a car on an angled road.

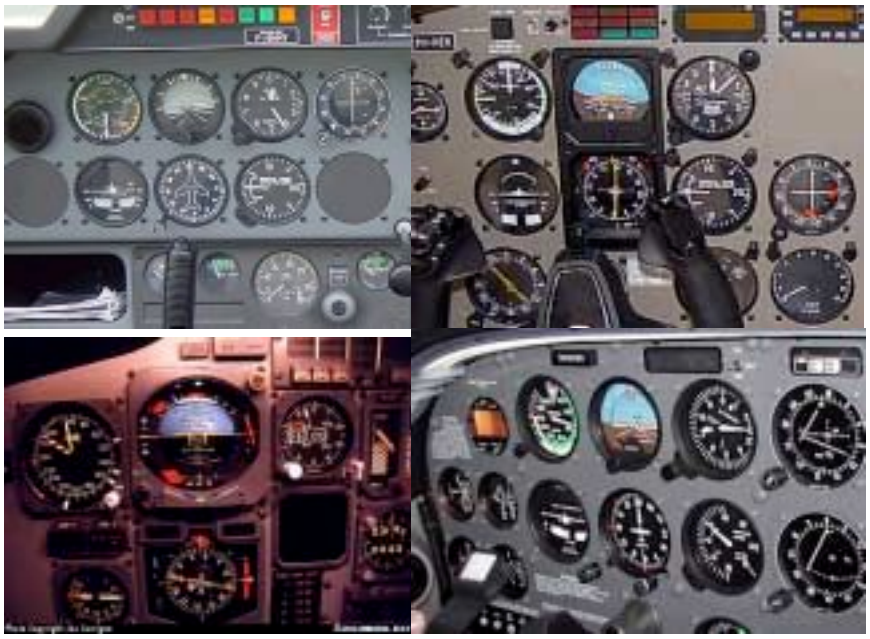


Fig 5.2: Basic T displays from top left clock wise; Robin DR400; Piper Archer; Cessna 172; Boeing747.

5.2 Software

The displays in the ground station will be developed from ActiveX displays. 6 main displays were used and they were arranged using the basic T layout as described above. The result can be seen in Fig 5.3. The ActiveX controls that were available were: airspeed indicator, altimeter, artificial horizon, direction indicator, vertical speed indicator and turn coordinator. The controls are developed by Global Majic Software and can be used as a stand-alone application. For the GCS they were interfaced relatively easily with LabVIEW. All other secondary displays have been created using the indicators present in LabVIEW.

As well as the six main displays there was also a requirement for various other flight control displays and warnings. These are; alpha and beta angular display, throttle setting, motor temperature warning, acceleration exceedance, battery warning and stall speed warning. On the second VDU the following systems analysis displays were required;

- 3 x angular rates ($\dot{p},\dot{q},\dot{r}$)
- 3 x accelerations $(\ddot{x},\ddot{y},\ddot{z})$
- Motor temperature

- Control surface position

5.3 Choice of displays

Obviously the best display for all of these things will be a different format. LabVIEW unfortunately does not have a great selection of displays however it does have what is sufficient for the purposes of this thesis. Each of the displays was selected based on a little bit of common sense but mainly what was available in the LabVIEW libraries.

5.3.1 Scale

One important thing to consider is the scale and the number of graduation marks. It is important that the marks are chosen so that they give the user a quick and accurate interpretation of the reading. If there are too few marks then important information may be lost as the dial is too hard to read. If there are too many marks then this may give the observer a false sense of accuracy. I.e. you may be able to easily take a reading to more accuracy than the accuracy of the instrument. A general rule of thumb used in the design of most instruments is to split the scale so that the marks represent 1, 2 or 5 decimal multiples. Another aspect to consider is whether to chose a linear or non-linear scale. All displays in the GCS will be measuring parameters that are best represented with a linear scale so this has not really been considered in the layout of the displays. Next one should be considering whether it is more appropriate to use, circular scales, straight scales or a digital scale. In standard aircraft it is usually most appropriate to use circular scales. However straight scales can often save space and fit into an arrangement more appropriately. In new aircraft, with the more widespread use of glass cockpits, standard circular displays are proving less common.

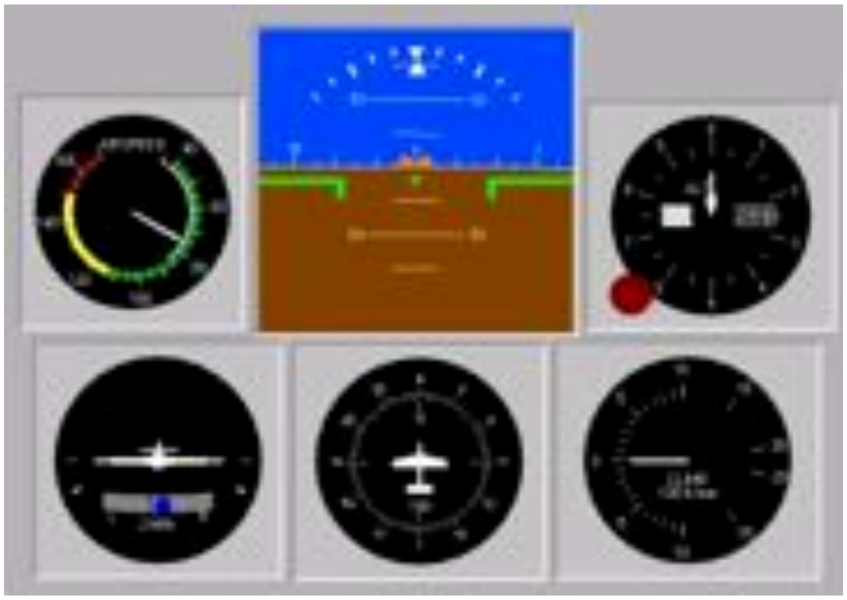


Fig 5.3: GCS layout of flight control instruments.

5.3.2 α & β

The angle of attack and sideslip angle are measures of angles. The best way for an observer to interpret angles on an artificial display is by using some sort of rotary pointer. So these will be displayed on the GCS's VDU using one of the basic rotary displays from LabVIEW.

5.3.3 Throttle setting

This can generally be interpreted on a scale from 0 to 100%. To display throttle setting a straight, linear scale has been chosen. The logic behind this is that the throttle is operated in a linear manner i.e. the control is moved backward and forward. It was felt that seeing the same movement on the screen would enable the pilot to interpret the signal more clearly.

5.3.4 Acceleration

There was a requirement for acceleration exceedance warning. While it was absolutely necessary to have some sort of binary warning in the form of a light or sound, it was felt that it was also important to see this situation approaching. Therefore another straight, linear scale has been chosen to display acceleration in the Z direction. It was assumed that this would be the axis with the highest acceleration. The choice of display for this was made mainly to improve the aesthetics. It was felt that a more symmetrical cockpit panel would be easier on the pilots eyes and would not be as confusing as lots of different formants in different places.

5.3.5 Motor temperature

The traditional device for measuring temperature is a thermometer. In the majority of complex machines, including aircraft, the norm for the output of a thermometer is the use of a rotary pointer display. The choice of display for the motor temperature has been chosen to as closely as possible represent this expected method of displaying temperature. It is known that if a person is used to seeing something then seeing that thing again is taken for granted. In other words the person can interpret what he sees, without thinking too much, compared to an unknown format.

5.3.6 Battery life

This is based on a timer. Is should be known how long the batteries will last and this is programmed into the code before the flight. The remaining time is then displayed in a similar display as that used by the thermometer.

5.3.7 Stall warning

There is no need for an analogue display for this as what is important is velocity. This is already being displayed on the six main instruments. Instead there should be a light/sound, which comes on just before the stall velocity is reached.

5.3.8 Flight test/Systems observer

All information required here will be displayed in real time on LabVIEW graphs displays. Since there is limited space, then some things will have to be displayed on the same graph. It has been decided to display the three rates on one graph and the three accelerations on another graph. Motor temperature will be on another graph and all the control surface positions will be on another graph. These plots are all accompanied with instantaneous digital readouts. See chapter 8 for a more detailed analysis of the flight data displays.

5.4 LabVIEW integration

5.4.1 ActiveX instruments

The main flight control instruments are ActiveX objects. This means that there is very little to do to get them working in LabVIEW other than simply adding the inputs. There are two blocks that need to be added in addition to the ActiveX block. We need to define the property that the input is for by adding a property node. There needs to be an invoke node that tells the display to refresh every time the loop is run. The initial task in getting the displays working was to introduce some arbitrary input such as a sine wave and check that the output was correct. Fig 5.4 shows this set-up for the Artificial horizon for three axis including the require property node. The result from this was a clear sinusoidal motion in three axes. So there is nothing more to displaying the output on an ActiveX control than what can be seen in Fig 5.4. However it did take considerable experimenting to figure this out. The AH being the most important display was the first to be set-up. Once this was working all the other displays were added in the format of fig 5.4 and wired in exactly the same way.

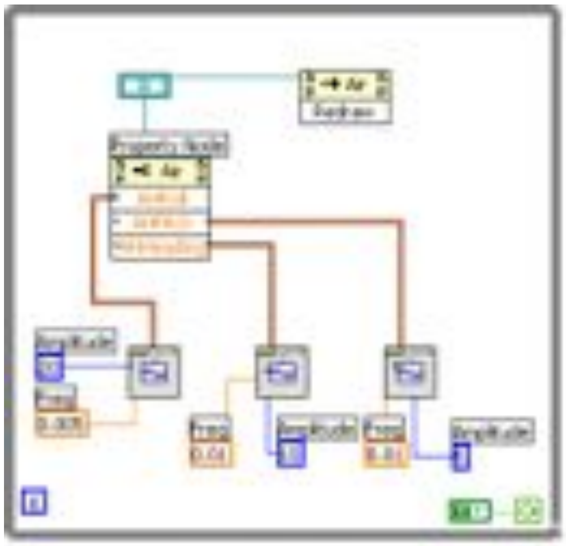


Fig 5.4: First test set-up of ActiveX control.

5.5 Summary

Six main aircraft instrument displays have been used on the instruments screen. These are based on ActiveX technology the dramatically simplifies the situation. The six primary instruments have been arranged in a patter based on the basic-T layout. This is a layout that can be seen on many of today's light aircraft. Several of these aircrafts cockpits have been studied before finalising the cockpit layout in the GCS. As well as the 6 primary displays, several other displays and warnings have been created, using built in displays from the LabVIEW program.

6 Joystick

6.1 Selection

The joystick that has been used is a Saitek X36F flight stick and a Saitek X35T throttle. This was selected both because it was one of the few sticks that had enough functionality and due to its quality. This is a system that is to be transported around in a van with perhaps quite harsh environmental conditions. For this reason it was important to have all the components well made and of a high standard. This is a dual port joystick i.e. it has both a gameport connector and a USB connector.

Fig 6.1: Saitek Joystick similar to the one used

6.2 Gameport

So initially there were two options: the joystick could either be connected using the gameport in the sound card or by USB. With standard software, USB would be the obvious choice due to its increased performance and flexibility. However there initially seemed to be some problems integrating this technology with LabVIEW. All documentation on LabVIEW seemed to indicate that there was no general and set way of calling USB devices in LabVIEW . It seemed that there was quite a complicated procedure which involved communicating with Microsoft's DirectX technology. As this was something that was beyond the technical expertise of the author0, the possibility of using USB was looking disappointingly unlikely. So the first thing that was tried was to get a more basic joystick and connect this via the game port.

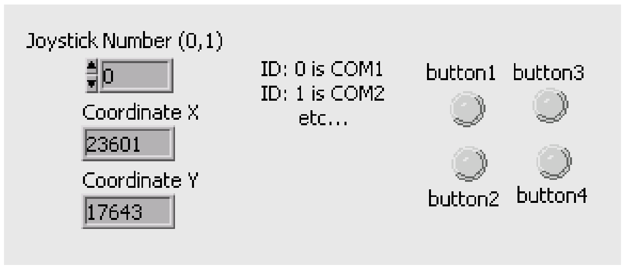


Fig 6.2: Display from basic gameport joystick VI.

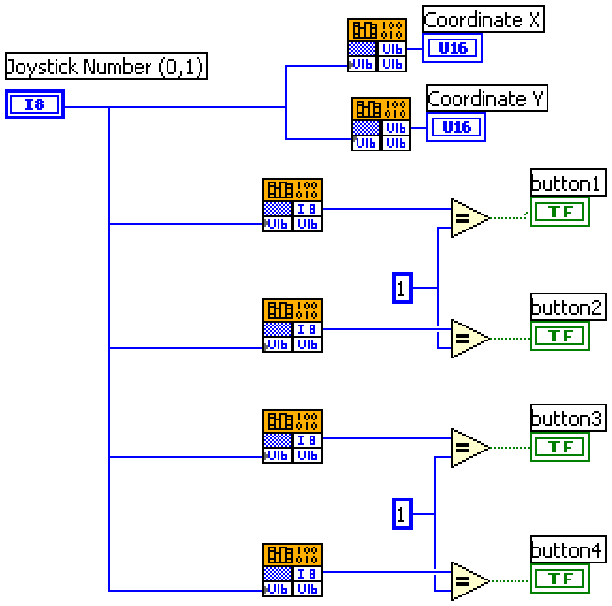


Fig 6.3: Code for basic gameport joystick VI.

The joystick used was one of the most basic Saitek controllers. The basis for the joystick code was obtained from the NI developer's zone web site. The VI[1] came with a universal dll file for a two-axis joystick. The code and the display for this can be seen above in figures 6.2 and 6.3. Apart form the input and output displays in this VI, the main structure purely comes from blocks, which call dll files. These blocks are known as a call library function. The main set-up involved in these functions are selecting the desired dll file and then selecting the desired function from that file. It so happens that the dll that was obtained with this VI was a very simple one. It gave high-level access to all the inputs from a simple joystick. In other word each of the above call library functions had one of the following functions associated with it:

- GetJoyX
- GetJoyY
- GetBtn1

[1] Written by Pavel Charny of "Automated Control Systems" Moscow, Russia

- GetBtn2
- GetBtn3
- GetBtn4

This VI worked perfectly with the simple 2-axis stick. However for the GCS at least four axes are required (ailerons, elevator, rudder and throttle). This meant one of two things either, rewriting the source code for the simple dll, or finding a new dll. As the source code was unavailable then this left only the latter option. So, a very long search was begun trying to find a dll, which would give high-level access to at least a 4-axis joystick. It could have been possible to write the required shared library in C, if Knowledge of C was available. However this was far too complicated for the time available in this project. A long time was spent searching for this elusive library file before it became obvious that it wasn't going to be possible to obtain one.

6.3 USB

6.3.1 USB background

USB (Universal Serial Bus) is a relatively new technology that is supposed to make connecting peripherals a lot simpler and have far faster data transfer. The USB technology was an unknown quantity before this thesis so it was interesting to learn of some of its advantages.

6.3.2 USB advantages

Simple to connect
In most cases USB greatly increases the simplicity in connecting an external device. It requires no terminator settings or id numbers. In theory all that is required is a device driver and the device is ready to operate. It is available on a wide range of peripherals including joysticks. This increased simplicity isn't that applicable to joysticks as most normal joysticks are fairly simple to set-up any way.

Performance
A more noticeable advantage when it comes to joysticks is the much-increased performance over a standard connection. USB devices can transfer data at up to 12 megabits per second – almost 50 times faster than traditional serial connections.

Instant access
A capability known as "hot-plug" means that the joystick (and other USB devices " can be plugged and unplugged while the computer is running. The device is detected automatically and no set-up is required.

Greater expandability.
Traditionally only one device could be connected to port at anyone time. With USB not only can devices be swapped over instantly but also multiple devices can be plugged into one port. Using several hubs, USB can accommodate up to 127 devices simultaneously.¨

6.4 Final Implementation

The key piece in the development of the GCS is to get the joystick interfaced with LabVIEW. The joystick will initially be the main control for the aircraft and it is vital that this is quickly a fully functioning part of the system.

After a long time with no success with the serial library method it was decided to look at USB again despite it seeming more or less impossible. Using the method of the call library function, as in fig 6.3, it would be necessary to find out how all the X36 drivers worked and how to call them in the right order. The X36 joystick comes with around 25 different dll files and it is not at all obvious what all of them do.

6.4.1 ActiveX

It was at this stage that another idea was developed. It was known that the ActiveX software had proved very simple in displaying input information for the flight control instruments. The same company that developed the flight instruments ActiveX controls also developed a joystick ActiveX. This is the solution that got the joystick talking to LabVIEW. Using this ActiveX object in exactly the same way as for the instruments the coordinates on all four axes could be read. It also included support of several other joystick features including several buttons and a POV[2] control. Although the final set-up of this solution was very simple, it took a long time to get it working. There was an unexplainable software problem for a long time that meant the software was not connecting with the joystick. To this day it is unknown what was causing the problem. It is only know that a lot of playing with various parameters fixed the problem. It should be noted that the ActiveX joystick control is not a driver for the joystick. It relies on there being an appropriate driver installed for the joystick being used. It is quite probable that this had something to do with the joystick not being recognised. However USB's compatibility and flexibility advantages can still be utilised as the joystick is detected automatically.

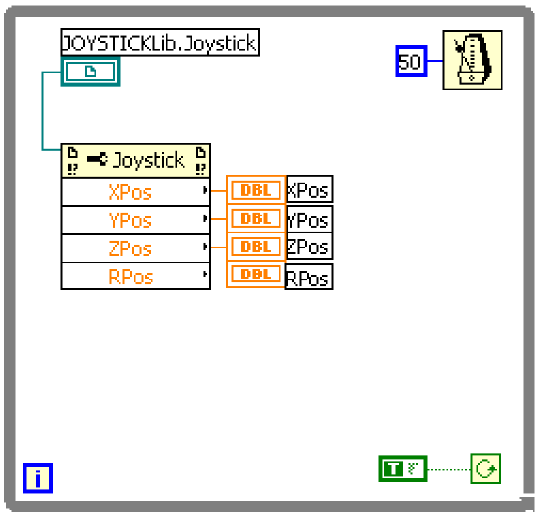


Fig 6.4: ActiveX joystick control

[2] Point of view: button on the joystick that allows the user to select one of eight compass directions.

Fig 6.4 shows how the GMS ActiveX is wired to obtain outputs in numeric form. For the joystick no invoke node is required – simply a property node with all the properties that are desired. At the initial stage of the project, simply the above 4 axis were used.

6.5 Testing Joystick Data Transmission

Except for the USB joystick, all data transfer to and from the ground station will by via the serial port in RS232 format. It is there for important to understand how this works and be able to implement it in LabVIEW. So, although the joystick coordinates are being read into the CPU via USB they will have to be sent to the aircraft via an RS232 system. It was decided that it was necessary to test the system simultaneously reading the input from the USB and writing the data to the COM port.

6.5.1 RS232 Background

The RS-232C standard is one of the oldest physical communication standards in computer world. The standard is a low-cost form of serial communication that sends bits down a copper wire in a robust way. It was originally defined for connecting peripherals such as terminals and printers to computer systems. Nowadays, the computer-to-computer link with a so-called null modem cable is commonly used.

Communication as defined in the RS-232C standard is an asynchronous serial communication method. The word serial means, that the information is sent one bit at a time. Asynchronous comes from the fact that the information is not sent in predefined time slots. Data transfer can start at any given time and it is the task of the receiver to detect when a message starts end ends.

6.5.2 Loopback test

A common test to check that the serial port on any computer is working is called a Loopback test. This involves fitting a Loopback adapter to the serial port on the computer. This has the RS232 transmit and receive lines shorted out. The process involves sending a signal out of the serial port waiting a few milliseconds for this to complete then reading what has come back to the computer via the adapter. A LabVIEW VI was found on the Internet that accomplished this task. This simply transmitted a series of characters, i.e. text and then displayed it again. If the same text was received as was transmitted, then the user could tell that the COM port was working properly. This acquired VI was taken and modified so that Joystick data could be sent and received via a Loopback test. The result of this modified program can be seen in fig 6.5.

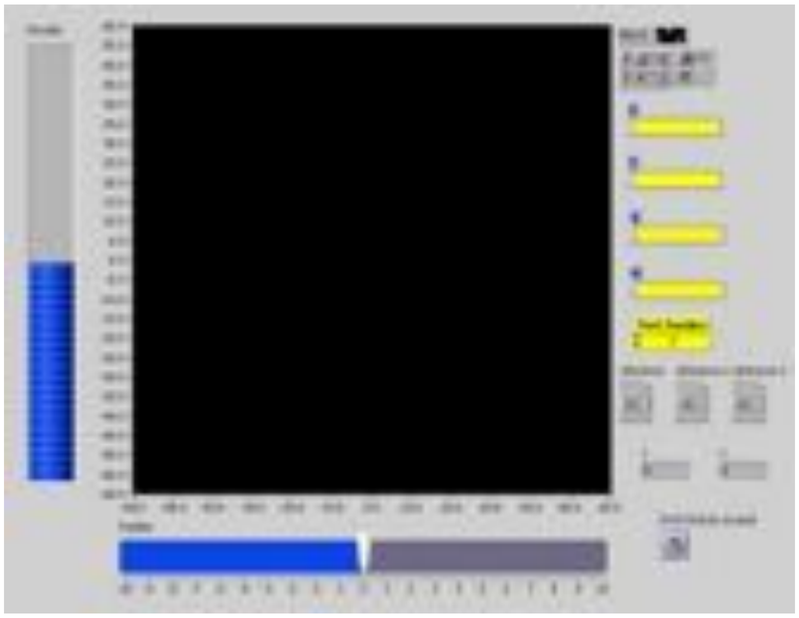


Fig 6.5: Output from Joystick Loopback test.

Fig 6.6 shows the code for the initial joystick Loopback test. It can be seen that there are 4 sequential parts to this code. First the port must be initialised. This involves opening the port and setting the desired settings for that port. In the next stage, the joystick data is written to the port. This involves, first, combining the four coordinates into one long string. This one string is sent to the port. Writing the data can take a few milliseconds so the next frame in the sequence is there to give a pause that allows all the writing to be completed. After the writing has been completed, the data can then be read back in again, in the last frame. The long string is read and it then needs to be broken up into the four axis components. This is done by taking the length of each component – measured during the write phase – and make this an offset for which to read a substring from the long string. We then have 4 strings. So that these can be displayed in a graphical format these strings need to be turned into a numerical format. This is done using a few simple conversion functions and array building functions.

6.5.3 Null-modem

Having completed a successful loop back test the next stage was to send the joystick signals to a second computer and read them there. To do this the above Loopback test was modified slightly. It was obvious that the Loopback VI would need to be split so that one computer (the one with the joystick connected) would be running the serial write section and the other one would be running the serial read section. The main problem is that the Loopback test involved a substring length that is measured in the write phase and passed forward to the read phase. As the two phases are not connected when the program is split, another way of unbundling the string will has had to be developed. The way that this has been solved has been to introduce a character after every number to indicate where one number ends and the other starts. This character was arbitrarily chosen as s.

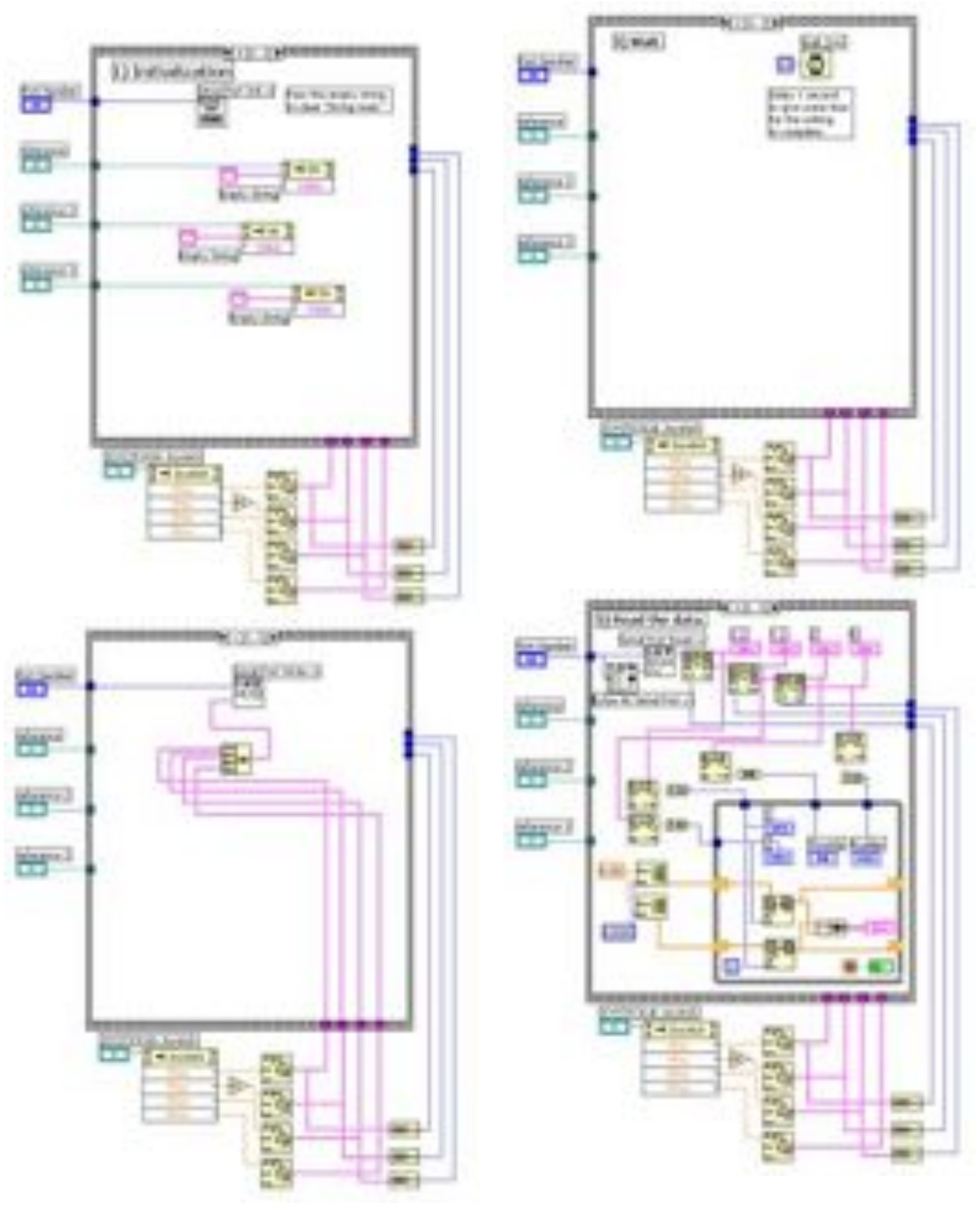


Fig 6.6: LabVIEW code from the joystick Loopback test

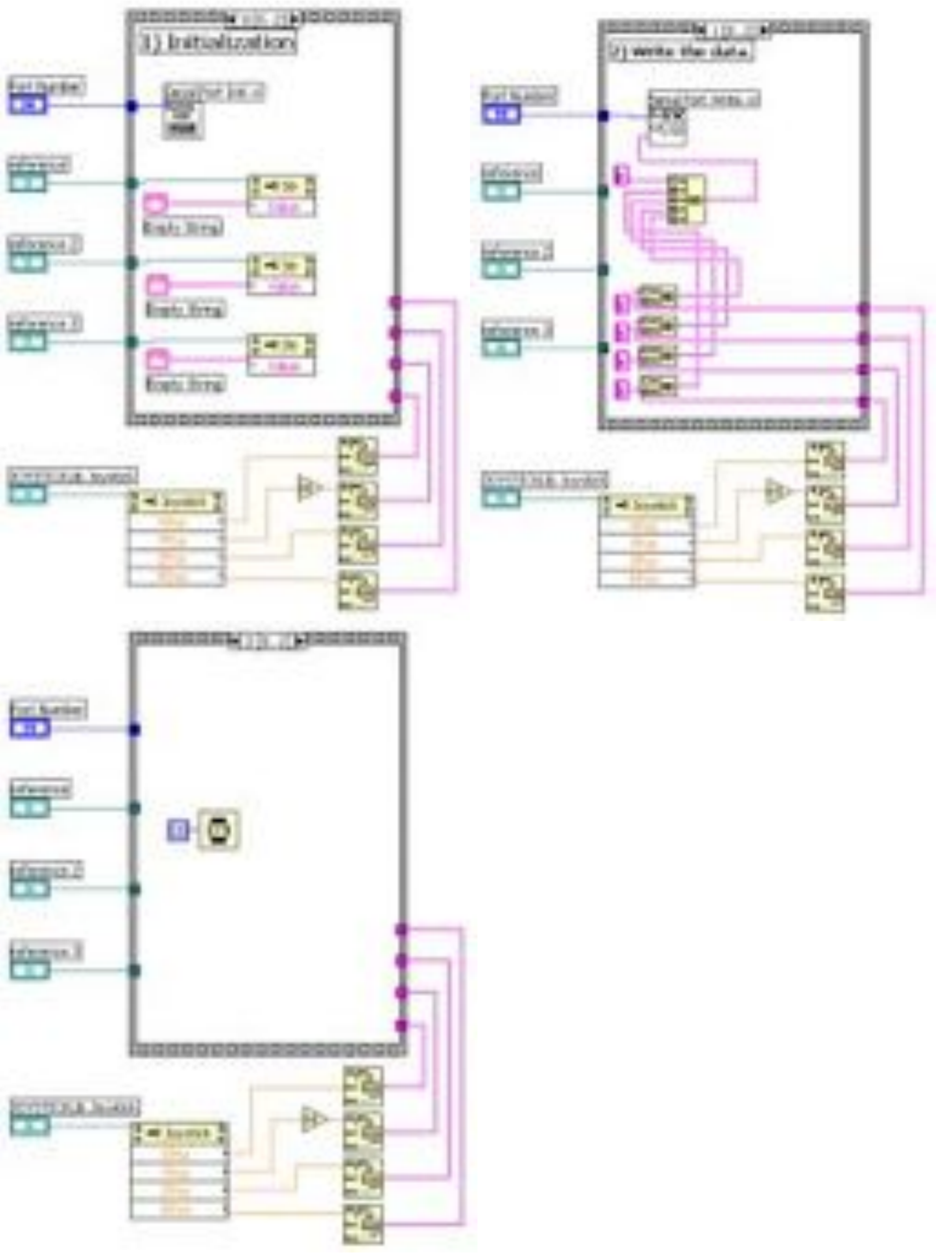


Fig 6.7: LabVIEW code for the Null modem test

Simple null modem without handshaking

Fig 6.8: Null modem cable wiring

Therefore the new write phase involves inserting an "s" after every coordinate number. So the read phase then needed to be adapted to search for this marker character and split the string based on the markers positions. The next aspect to be addressed was the fact that the data arriving at the second computer isn't necessarily being read starting at the start of a long string. It was also necessary then to mark the start of the long string with another marker character. This was arbitrarily chosen as the letter p. Fig 6.7 then shows the program used to send the data from the joystick. Fig 6.9 shows the VI run simultaneously on the second computer to receive the data.

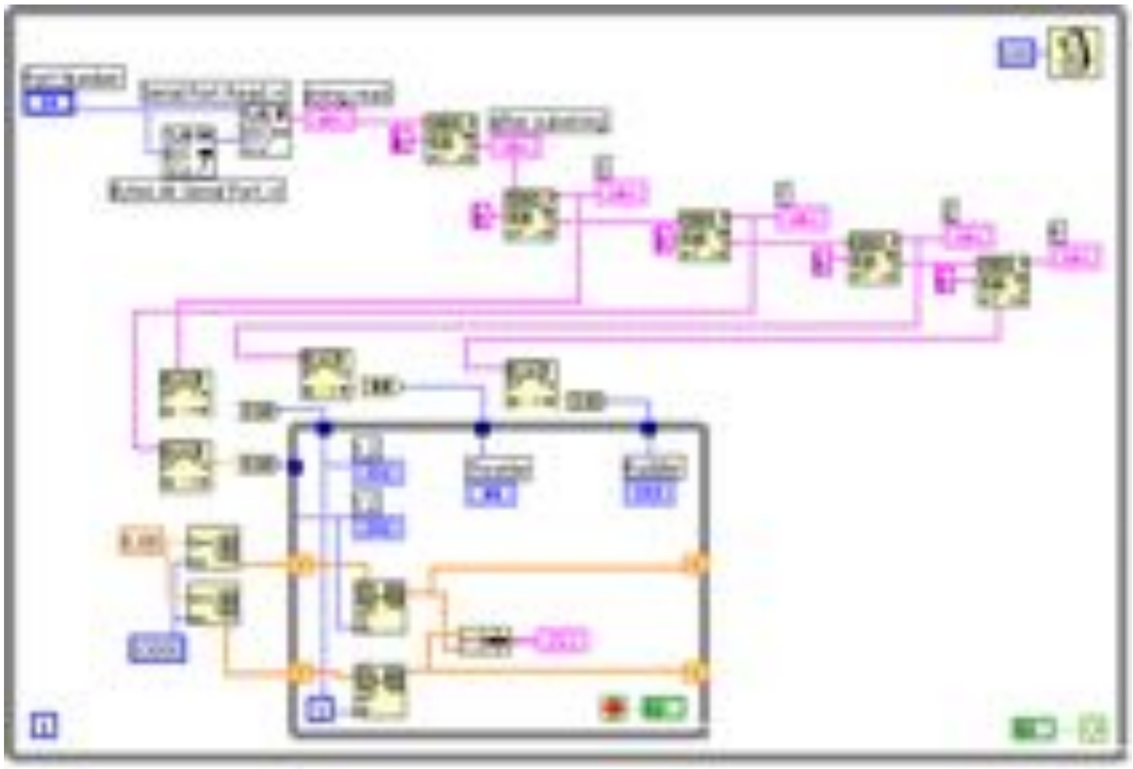


Fig 6.9: Code used to receive the joystick data

6.5.4 Accuracy of Loopback and Null-modem.

There are two possible factors contributing to inaccuracies and deviation in the error in its calibration. It is also likely that there are errors being developed in the transmitting and receiving of data using the RS232 link. There is certainly data being wasted due to the way in which the Loopback test works. This is because there is a delay that lets the data be written. Although this reduced the amount of data flow it doesn't necessarily reduce the accuracy of that data which is transmitted.

It has been noticed that there is occasionally a small bug in the interface of the joystick with the ActiveX control and LabVIEW. This involves the signal getting all jumpy and inconsistent. Shutting down and loading up LabVIEW again fixes this. This is obviously not ideal but there is no indication of why this is happening so a fix has not been able to be developed.

The governing factor for the speed at which this section can run, and ultimately the definition of the refresh rate, is the rate at which data is transferred. This is known as the baud rate, and in the case of the joystick signals this has been set at 9600 Kb/sec. This was seen as a fair compromise between the speed at which the program will run and the amount of processor power that is used.

6.6 Summary

The joystick that is used in the GCS is a Saitek X36 flight stick. After looking at both Gameport and USB connection methods, a solution was finally obtained by reading the joystick in LabVIEW via a USB connection. This has been achieved by using another ActiveX control. This quickly and easily allows the joystick coordinates to be read in LabVIEW. To test how easy it is to write these coordinates to the COM port, several tests have been performed: first using a Loopback test and then by using a Null Modem cable. Eventually it was possible to display the joystick coordinates on the screen of a second laptop using the Null Modem. This was seen as an effective testing method as it uses the same transmission system as will be used in the final system: RS232.

7 Control of other displays and functions

7.1 Battery power/timer

The initial approach to this has been to adopt the procedure that is used with model aircraft. The simplest way to keep track of endurance is to set up a timer. This involves entering a time at the start and the timer then counts down from this time. This time must be calculated from the known endurance of the batteries in the aircraft. So what we need for the GCS is a front display of time left, and software to control the count down. The only user input for this will be a decimal input of the time required in minutes and a start stop switch so that the timer can be run independently of the rest of the program. The display has already been shown, the source code for the timer will now be detailed.

7.1.1 Timer Requirements:

- User should enter desired time in minutes
- Must count down from this time and display on a dial and a digital display
- Digital display should be in the format (minutes:seconds:tenths)
- Timer should stop when it reaches zero.
- A warning light and buzzer should sound with one minute left
- There should be a start stop switch to control the timer independently of the rest of the program

7.1.2 Solution:

The first criterion is satisfied by means of a digital control on the front panel. This value is then multiplied by 60 to turn it into seconds. This is then added to the real time at the starting moment. We then subtract from this the real time at a succession of moments after this to get the time passed. We then create a percentage by dividing by the total time and multiplying by 100.

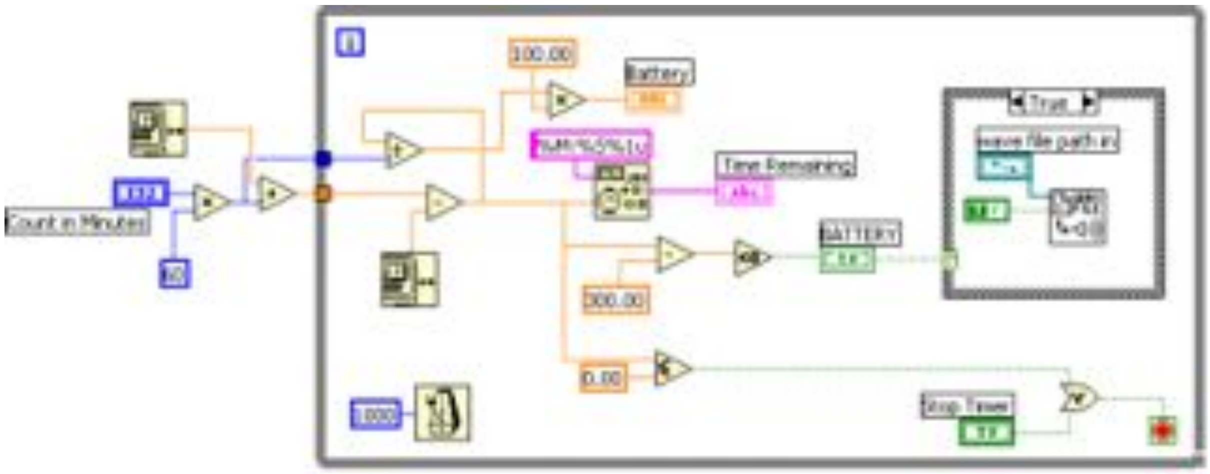


Fig 7.1: Code for the timer section

This gives the numerical value to be displayed on the indicator dial. Also required is the time in the prescribed digital format. The easiest way to display this is to turn the signal into a string and format it that way. The next aspect of the timer is to get the warning at 1 minute left (or any other desirable time). This is done simply by subtracting the desired number of seconds and initiating the warning when this new signal reaches zero. The loop that controls the timer can be stopped either when the stop button is pressed or when the original time (before subtraction of 60 seconds) left reaches zero. The final aspect that needed to be added was a delay. This was not added until the timer subsystem was combined with all the other subsystems. The timer works fine without it but when other things need to be done, if there is no pause, then the time takes up far too much computer time and the whole system grinds to a halt. It was decided that there was no need for a timer accuracy of more than 1 second, so this is the delay between displays that has been added.

7.2 Motor temperature sensor

Although there was an initial requirement to incorporate a motor temperature indicator, there is as yet no defined hardware to work with on this. It has therefore not been able to incorporate this into the GCS apart from the definition of the graphical output.

7.3 RPM meter

Although not a requirement in the initial definition, it is known that there is on going work on an RPM meter. It would thus be desirable to include this as a display on the cockpit screen. It was hoped that enough work would have been carried out on the actual sensor to be able to incorporate the reading into the GCS's displays. Unfortunately, time in both this thesis and the development of the RPM meter has prevented this from being implemented.

7.4 Summary

With the present state of the available hardware it has really only been possible to program code for the battery timer section. This has been done with a simple while loop that compares the time inside the loop with that from the start.

8 Flight data display

8.1 Layout

When considering the layout of the flight data display, one has been aware that there is limited screen space and all of the data needs to fit on one screen. Yet it needed to remain clear and easily readable. The first thing that was decided was what data was going to be displayed on which plots. It was obvious that there was not going to be enough room on the screen to have a separate plot for all the desired information. It was initially a thought that there could be multiple data on some plots. There are two main ways of doing this in LabVIEW. First one can have several data lines on one plot or the plot can be split into individual sub plots. With the first style of layout, one must be aware of the data range of all the plots. It is not the best solution to have multiple lines on the same graph where the different sets of data are in different orders of magnitude. This is because if you have something that has a substantially greater range than another set of data then the small data range will not be seen clearly.

As an example it was initially thought that the three accelerations could be displayed in this way on one graph. However when one considers the forces that are present on an aircraft, it can be seen that the Z-axis acceleration will usually be higher than that experienced on the other two axes. If these parameters were plotted on the same graph, then this difference in magnitude would mean that the size of the Z-axis acceleration would detract from the clarity of the other two acceleration plots. It can also be seen from the raw AHRS data (see chapter 9) that X and Y accelerations are centred on zero for steady level flight but during steady level flight the Z acceleration value is centred on a value of -1. This is due to the acceleration due to gravity. This factor again reduces the clarity from putting these parameters on the same plot. It was concluded from this analysis that the accelerations could not be displayed together. While it might be relatively clear to display the X and Y accelerations on one plot, it was decided that there was enough room for separate plots and this would result in a far clearer display. There was also a requirement for an acceleration warning. This has been implemented as a visual and aural warning on the cockpit display screen.

The second major set of displays on this screen are the angular rates. It was expected that on average these would have values of relatively the same magnitude as each other. However like before, it could be seen that there was just enough room to display them individually. This was preferable, as it was important to maximise clarity at all possible opportunities.

The next possibility for a multiple plot was the data from the three control surfaces. It was presumed that the deflections of these would be in the same order of magnitude as each other so these three have been placed on the same graph display. The next thing to decide was whether to use the single screen with multiple lines or the multiple split screen display for the two screens with multiple plots. It was decided that the split screens would be possible as the deflections would be small values for the majority of the time.

Although they were not required on the flight data display, it was decided to include plots of roll, pitch and heading for the sake of completeness. As there was only room for two more plots, it was decided to combine the plots of Pitch and roll, as these would usually have the same order of magnitude.

It can be seen that there is a lot of data to be displayed on the flight data screen. It maybe possible to condense the data so that it does not take up so much room. However, when doing this there is a high chance that the clarity and definition of a lot of the elements of the data may be lost or degraded. It has been decided that the best solution was to arrange the displays in a grid pattern. The grid has been arranged so that related parameters are in the same row and the more important data is at the top of the screen..

There are two elements that are both optional and can be separated from each graph. These are the legend and scroll bars. It was concluded that both of these are essential. The legend may not be so important for most plots but it is absolutely necessary for the roll/pitch and control surface plots, where there are multiple lines on one figure. The scroll bar is essential for all plots as it is important in long flights to be able to look back at the data from all of the flight in an uncluttered format. When the scroll bar is moved, one can access the readings from any point in the flight. So that space is used efficiently, these two elements are not placed directly at the side of the corresponding plot. So that all the plots can be placed with an efficient use of space, all the legends and scroll bars from each row are placed at the right side of that row. The upper most legend/scroll bar always refers to the right most plots.

8.2 Programming the displays

There isn't a lot of complicated programming behind the displays on the flight data screen. The plots are simply standard LabVIEW functions. These are simply wired to the outputs from the AHRS section (see chapter 9). The plots are displayed using a Double-precision floating-point representation. That means that the representative numeric has a capacity of 64 bits.

8.3 Scale of plots

The Scale on the Y-axis of all the flight data plots is a fairly important variable to be considered. LabVIEW plots have the option to automatically scale, however this continuously scales to the instantaneous values and thus is not as clear as it might be. It has been concluded that all plots should have a fixed scale on the Y-axis. The scales have been chosen to represent the normal operating range of the aircraft. It has been assumed that, at least for the initial flight, the aircraft will be restricted to relatively smooth and conservative manoeuvres. Therefore scales on the roll and pitch plots have initially been restricted from the maximum possible range of 360 degrees. Several hours of hand testing and a reasonable amount of 'engineering judgment' have gone into deciding the maximum value for most of the displays. The range of all the plots can be seen below.

• Roll/ Pitch	-	± 60
• Heading	-	+ 363 to - 3
• Z acceleration	-	+1 to -3
• X/Y acceleration	-	± 3
• Angular rates	-	± 100
• Control surfaces	-	± 3
• AoA/Sideslip	-	± 10
• Engine temp	-	indefinable at present time

It should be noted that although these values have been inputted into the program in its present condition, they could be changed by any user in a matter of seconds.

9 AHRS

9.1 Configuration

The primary sensor in the aircraft is an attitude and heading reference system (AHRS). This sensor uses solid-state technology to provide the user with measurements of angular attitude, magnetic compass heading and linear acceleration. The sensor is driven by a microprocessor that is used to undertake signal conditioning, undertake the coordinate transforms and provide the compensation needed to correct for the 15 degrees per hour rotation of the earth. One important factor to note is that the accelerations that are obtained from the sensor are resolved into Earth-axes thus making them independent of sensor attitude changes.

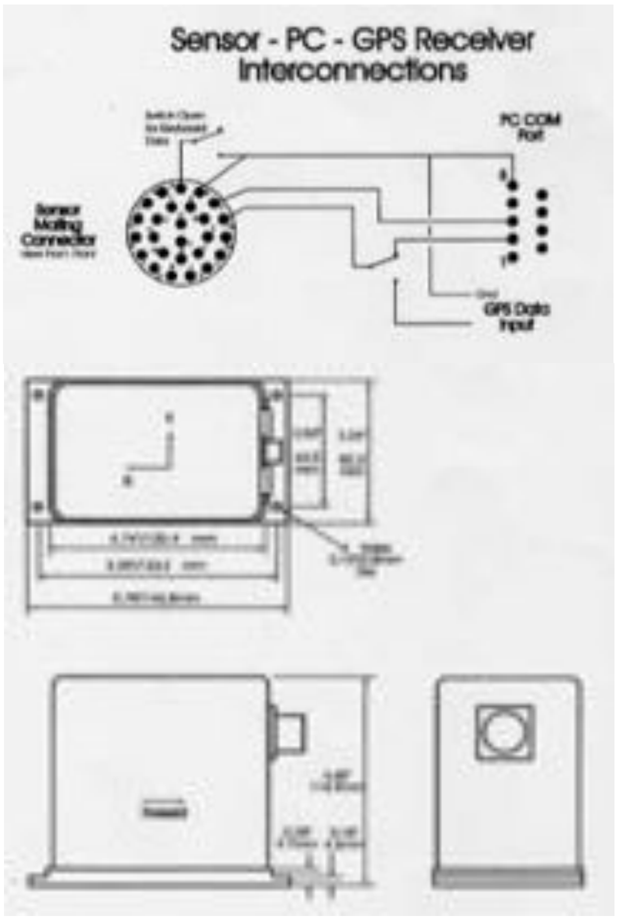
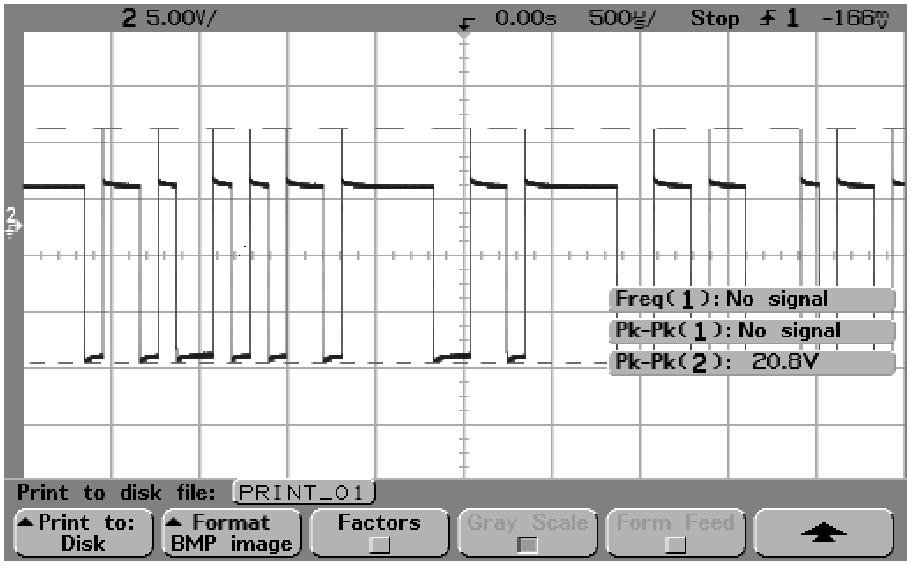


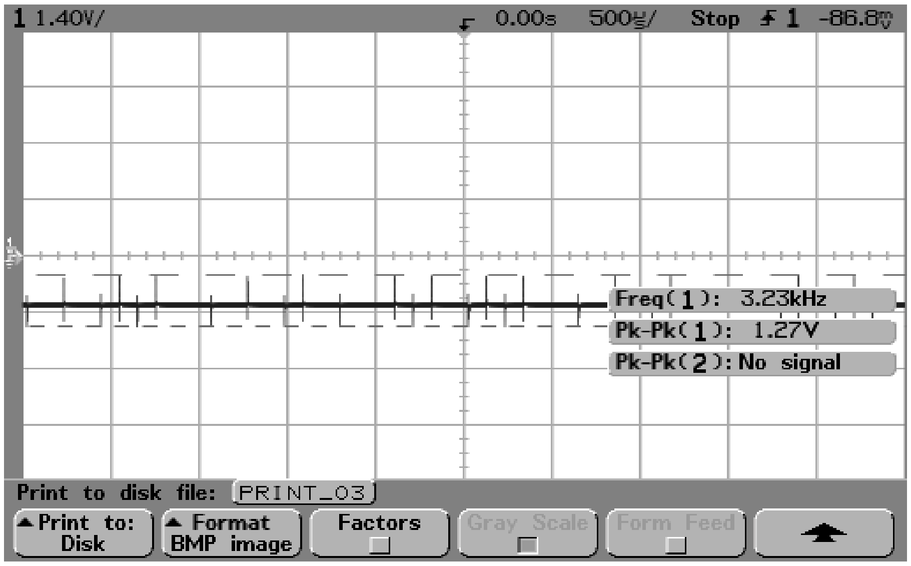
Fig 9.1: Watson industries AHRS

The AHRS that was supplied from Watson Industries came with some software that can be used to calibrate the sensor and display the data coming out of the data. This was seen to be the starting point when the sensor was first delivered. However there were problems getting the sensor interfaced with this software. Initially the sensor RS232 cable had been made according to the following diagram that was supplied with the sensor. To try to solve the problem of no data, an oscilloscope was used to read the signals coming down both wires. The signal from the RS232 transmit wire can be seen in Fig 9.2 and the signal from the receive wire is displayed in Fig 9.3. So, from these readings it can clearly be seen that the reason the computer was not receiving any data was that there was no signal in the receive wire. Yet there was a signal in the transmit wire were one was not expected. It was then realised that the diagram in Fig X was wrong and the transmit and receive wires had been connected to the wrong terminals. The wire originally on pin two should have been connected to pin 3 and vice versa.



Freq(1): No signal
Pk-Pk(1): No signal
Pk-Pk(2): 20.8V

Fig 9.2: The signal from the RS232 transmit wire before correction

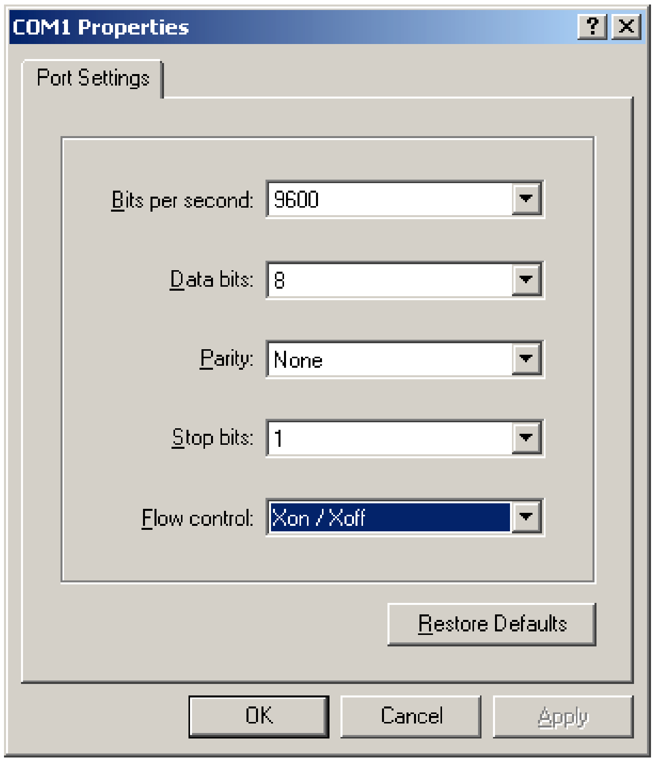


Freq(1): 3.23kHz
Pk-Pk(1): 1.27V
Pk-Pk(2): No signal

Fig 9.3: The signal from the receive wire before correction

When these wires were swapped over, the sensor produced data that could be read on the screen as seen in fig 9.5. It can be seen that the total amount of data coming out of the sensor consists of 17 columns. This is not all needed, but certainly the first 6 columns are essential to this project. These are roll, pitch, yaw, x acceleration, y acceleration, and z acceleration. The initial task for data extraction was to write code in LabVIEW that would display the first 6 columns of data.

The next problem that was encountered with the AHRS was that it only seemed to be outputting 4 columns of data in any application other than the supplied software. Initially data was read in Hyper Terminal, Matlab and LabVIEW. In all of these only 4 attributes were obtained. However with the supplied software the output was as shown in Fig 9.5. It was thus assumed that the software that was supplied contained some sort of commands that told the AHRS to send more data. After a long period of fruitless experimental attempts, access to the internal flash memory was obtained using a sequence of commands in Hyper Terminal. During the course of this project, two-computer login ids have been used. For some reason data could only be written to the AHRS using one of these id's. This was one reason why it took so long to discover how to access the flash memory in the AHRS. It is also important to get the settings exactly right, particularly the flow control, which must be set to Xon/Xoff. The figure below (Fig 9.4) shows the port settings that must be employed to be able to communicate with the AHRS.


COM1 Properties

Port Settings

Bits per second: 9600

Data bits: 8

Parity: None

Stop bits: 1

Flow control: Xon / Xoff

Fig 9.4: AHRS port settings

Once the correct port settings have been established then command access is obtained by entering the key sequence "! <Space> <space> <space>". One must then wait approximately 10 seconds for the sensor to start sending data again and then the following commands become available:

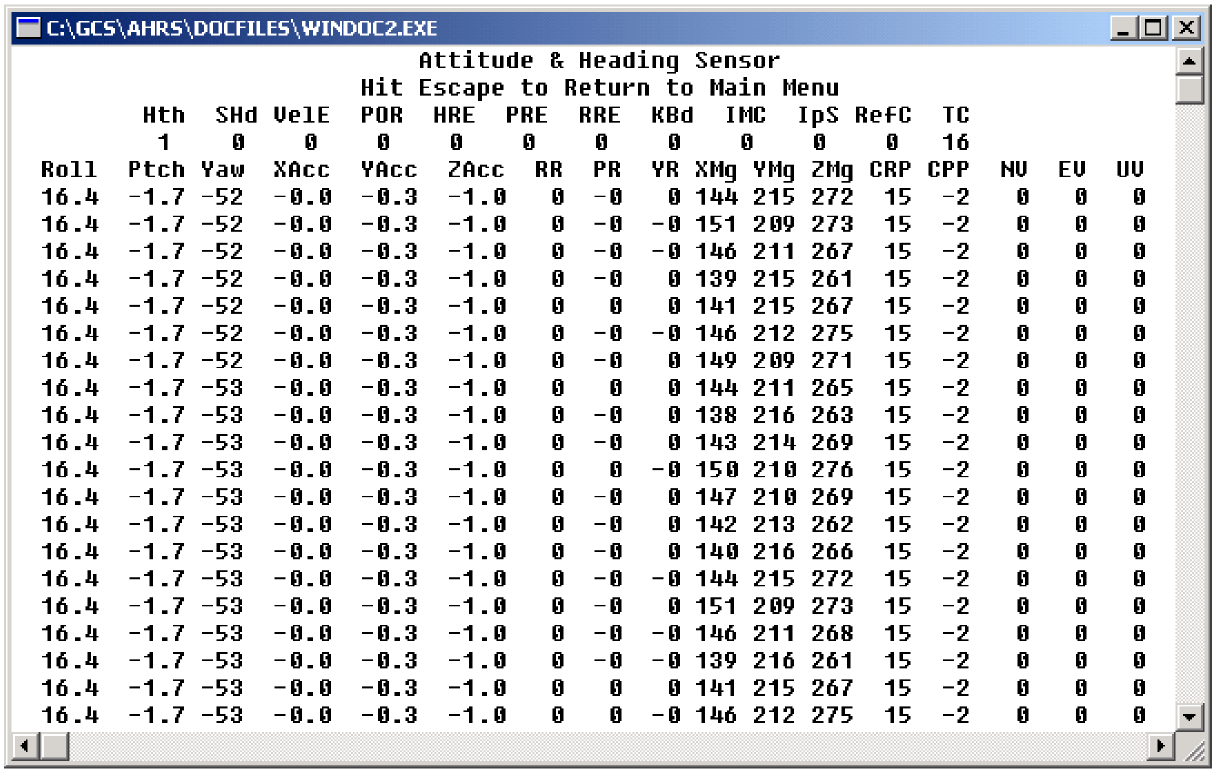
'!<spacebar>'	:- Re-initialises the integrators after a violent manoeuvre
"	:- Makes user-selected options permanent and updates sum-check
'&'	:- Requests User Options Menu
':'	:- Sets and clears single frame mode
'_'	:- Sets serial data output to ASCII mode
'F'	:- Sets inertial mode
'H'	:- Sets CM mode
'I'	:- Clears reference mode

'K'	:- Clears CM and inertial mode
'R'	:- Sets reference mode
'S'	:- Test EEPROM Sum-check
'^'	:- Set output to binary mode

Most of these are not important and do not feature in this project except for '&' which is the command that requests the user options menu for the sensor. Having selected this command, it was then possible to select the outputs that were desired. The outputs that were selected were;

- Pitch
- Roll
- Heading
- X Acceleration
- Y Acceleration
- Z Acceleration
- X angular rate
- Y angular rate
- Z angular rate
- Heading rate

When selecting outputs, one was aware that there was limited space on the flight data monitor and that the more outputs there are then the slower the program will run. As all the data comes in one string, then the more data there is, then the longer that string will be, and thus the greater the time that must be waited between readings.



```
                          Attitude & Heading Sensor
                      Hit Escape to Return to Main Menu
       Hth  SHd VelE  POR  HRE  PRE  RRE  KBd  IMC  IpS RefC  TC
        1    0    0    0    0    0    0    0    0    0    0   16
Roll  Ptch Yaw  XAcc  YAcc  ZAcc  RR  PR  YR XMg YMg ZMg CRP CPP  NV  EV  VV
16.4  -1.7 -52  -0.0  -0.3  -1.0   0  -0   0 144 215 272  15  -2   0   0   0
16.4  -1.7 -52  -0.0  -0.3  -1.0   0  -0  -0 151 209 273  15  -2   0   0   0
16.4  -1.7 -52  -0.0  -0.3  -1.0   0  -0  -0 146 211 267  15  -2   0   0   0
16.4  -1.7 -52  -0.0  -0.3  -1.0   0  -0   0 139 215 261  15  -2   0   0   0
16.4  -1.7 -52  -0.0  -0.3  -1.0   0   0   0 141 215 267  15  -2   0   0   0
16.4  -1.7 -52  -0.0  -0.3  -1.0   0  -0  -0 146 212 275  15  -2   0   0   0
16.4  -1.7 -52  -0.0  -0.3  -1.0   0  -0   0 149 209 271  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0   0   0 144 211 265  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0  -0   0 138 216 263  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0  -0   0 143 214 269  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0   0  -0 150 210 276  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0  -0   0 147 210 269  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0  -0   0 142 213 262  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0  -0   0 140 216 266  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0  -0  -0 144 215 272  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0  -0   0 151 209 273  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0  -0  -0 146 211 268  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0  -0  -0 139 216 261  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0   0   0 141 215 267  15  -2   0   0   0
16.4  -1.7 -53  -0.0  -0.3  -1.0   0   0  -0 146 212 275  15  -2   0   0   0
```

Fig 9.5: AHRS output from the supplied software.

9.2 Data in LabVIEW

The data being received from the AHRS consists of ASCII characters using the RS232 interface. The data is coming in through COM1, so the starting point was to experiment with some of the basic serial communication VI's. The starting point was to look at an existing program in the LabVIEW examples called : LabVIEW<-> Serial.

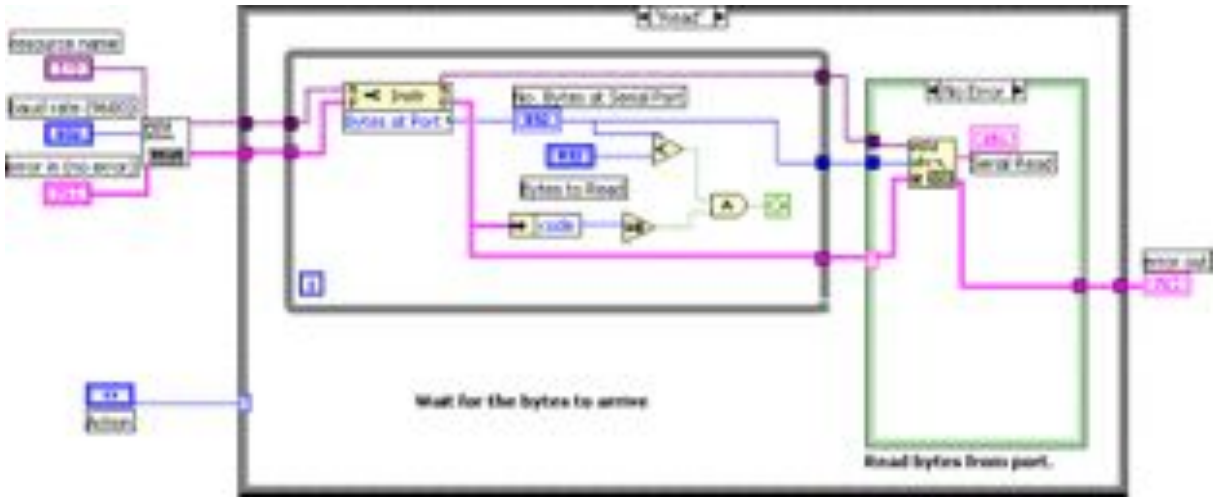


Fig 9.6: Basic Serial read VI

This VI was designed to read directly what was being transmitted into the serial port on the computer. It also enabled data to be written to the transmit line of the COM port. This was also necessary, as it was required to send signals to the AHRS to alter its settings. The code for this starting VI is shown in Fig 9.6. It consists of two parts: Waiting for the data to arrive and then reading it using a standard read function. It also shows a limiting factor in this program. The way this works is that it requires the user to select a number of bytes to be read, it then waits for this amount of bytes to become available before reading them. Now with the AHRS this would not be a viable way of working the procedure as the device is continuously sending data and thus it was necessary to develop code to continuously read the data.

The first modification to be made was to take out the while loop just for waiting for the bytes and just let the computer read the bytes waiting at the port, however many there are. It still needs to be able to read continuously however, so a while loop is inserted, which encompasses the property node and the read section. This enables the data to be continuously read although it isn't much use in a continuous stream of characters. What must be done is to put a delay in the loop that is just long enough so that 1 line of data is read every iteration. This is because it takes a certain amount of time for the data to arrive at the port and if it is being continuously read then it is not possible to interpret. So once one line of data has been read it then needs to be split up into the column components and then turned from strings into numerics. The procedure is the same as that used to read the joystick data. First of all the start of the line needs to be identified. This is made easier by the way the data is formatted. Every line is prefixed by one of 4 different letters. Depending on what mode the sensor is in, every line starts with either "R", "r", "I", or "i". Thus the 'Match Pattern' function is used to search for these indicators and then out put a string as whatever follows the initial character. Research was then carried out to discover that each of the columns always take the same amount of room in terms of number of character. Therefore the

'String Subset' function was used to split the string into subsets based on fixed off sets and lengths. This gave the required data as strings. These were then turned into numerics using the 'Fract/Exp String To Number' function. The final program used to read the data in LabVIEW can be seen in fig 9.7. This shows all 10 parameters being displayed as digits and in graphical format as a scrolling plot. Also the two ActiveX controls that are directly driven from the AHRS data have been wired in to the program.

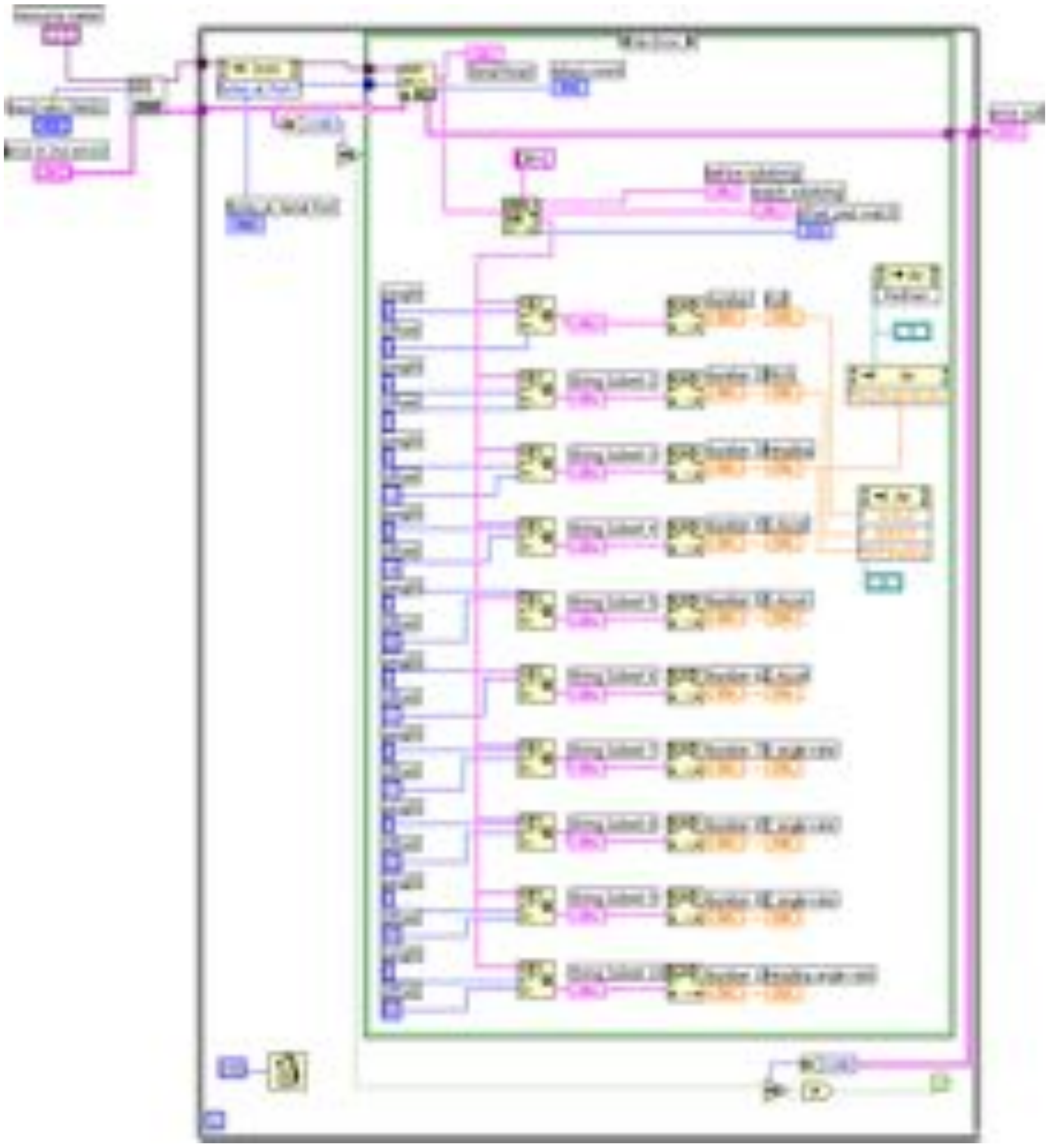


Fig 9.7: Code for decrypting the AHRS data. Including wiring to two ActiveX controls.

10 Writing data to a spreadsheet

10.1 Programming

Having run the program several times, it can be seen that the data available from the plots is not sufficient for analysis of the flight. What is needed is a permanent record of the data that occurred during the flight.

There where two obvious options for this;

- Create a text file that outputs the raw data collected in long string form.
- Break the string up so that it can be read using a spreadsheet.

Since the latter option is no more complicated and is far more flexible, then this was the obvious choice. So for a permanent record, the data from one run of the program is saved as a file can be read in a spreadsheet such as excel. This has been accomplished by first creating a 2D string array: one dimension for each of the parameters and the other dimension for each iteration of the while loop. The first dimension is simply created by building the array with all the parameters inside the while loop. Wiring the first dimension to the outside of the while loop using an indexing node creates the second dimension. This array is then converted to a format that can be read by a spreadsheet. This is done by using the 'array to spreadsheet string' function. This works by inserting tabs between the numbers, which means the spreadsheet can distinguish what goes in each cell. When using the function, it is important to specify the format of the numbers that are to be used. In this case the format is that of a fractional format. Also when reading the file it is important to specify the correct decimal point symbol. There were problems initially, as LabVIEW creates strings where the decimal point is represented by a full stop and the Swedish version of excel being used represents the decimal point with a comma. Therefore when opening one of the files using excel it was important to specify that a full stop was being used.

One small, yet important part of the code is the button that breaks the while loop. If there was nothing other than an error to stop the while loop then the program would never finish building the array and the file would never be saved. It was therefore important to include a button that would break the AHRS sub routine of the program. This is a Boolean switch that is wired to the continue node of the while loop. When this is pressed, the program stops receiving data from the AHRS and saves everything it has collected so far in a user-defined file.

So that graphs can be easily drawn using the spreadsheet data and so that the values have a reference, it has been arranged that the first row of the array contains a time stamp. This and all the other features of this section can be seen below in Fig 10.1. In this, 'X' represents everything up stream in the system that is required to obtain numerics of the ten parameters. It can be seen from this, that the time stamp, which is in the first row of the build array function, is created from two get time functions. It is simply the difference between the real time at the start and the real time during that particular iteration.

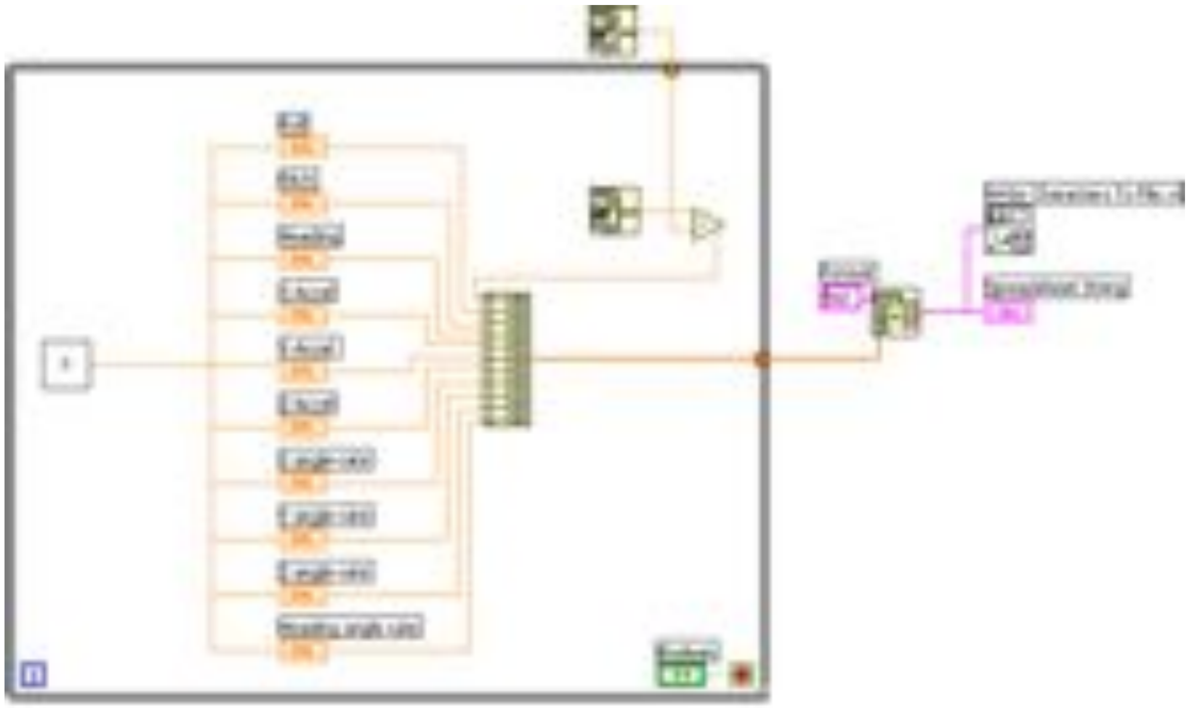


Fig 10.1: Components of the write to spreadsheet section

10.2 Data logging tests

Several tests have been performed with the code to check that it works as it should. Example results can be seen in appendix C. This test involved moving the AHRS first in a rolling motion, then in a pitching motion and finally in a yawing motion. The results of this test were plotted in an excel graph which can be seen below (Fig 10.3). In the author's opinion, this plot is a little clearer than the LabVIEW plots (Fig 10.2) for post processing analysis, especially if heading were to be separated from the other two. However it is realised that is sort of thing very much comes down to personal opinion. The advantage of the system developed here is that the user has a choice of two methods of analysis. That of LabVIEW and that of the excel data.

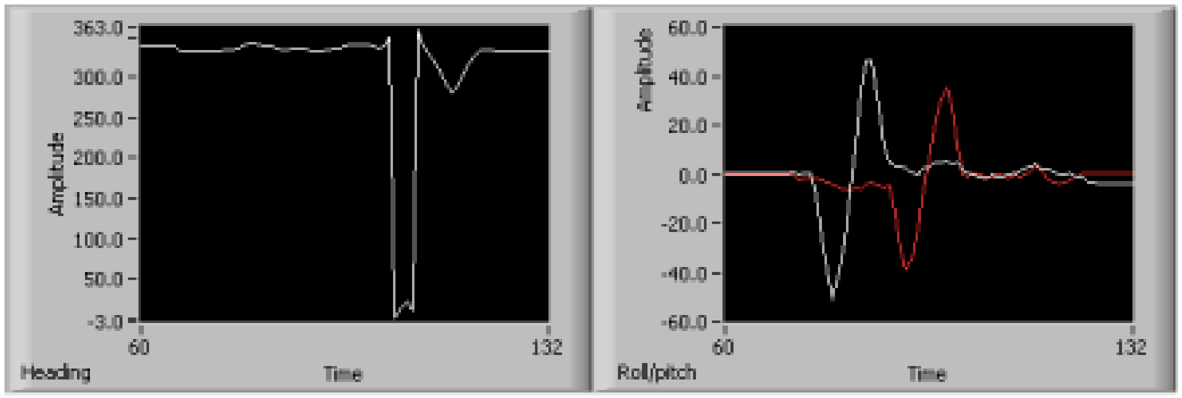


Fig 10.2: LabVIEW output from data logging test

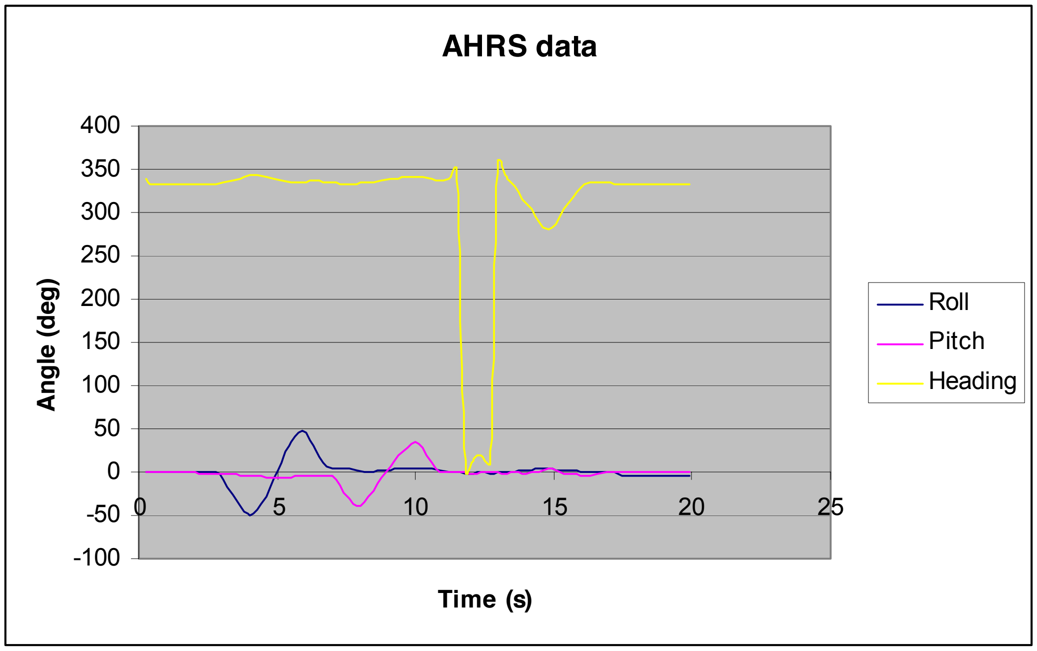


Fig 10.3: Example excel plot of recorded data.

10.3 Summary

The reason for the ' Write to spreadsheet' section was that it was desirable to have a more permanent record of what happened during the flight compared to LabVIEW's temporary memory. The method has been to create an array of double-precision numerics that are then turned into a spreadsheet string. The results can then be viewed and implemented in excel.

11 Programming of Aircraft instruments

11.1 Turn Coordinator

After the initial applications of the AHRS, the next stage was to use the attitude data to program the inclination ball in the turn coordinator. Unfortunately at this stage, the bank indicator could not be programmed. As this indicates a two-minute turn, then velocity data is required from the air data system. This is as yet an undefined part of the greater system. However the accelerations and roll of the AHRS can be used to give a display from the inclination ball. The position of this ball is a function of vertical acceleration, lateral acceleration and roll angle. What is essentially being represented is, if one imagines a little ball inside the AHRS sitting in a curved surface.

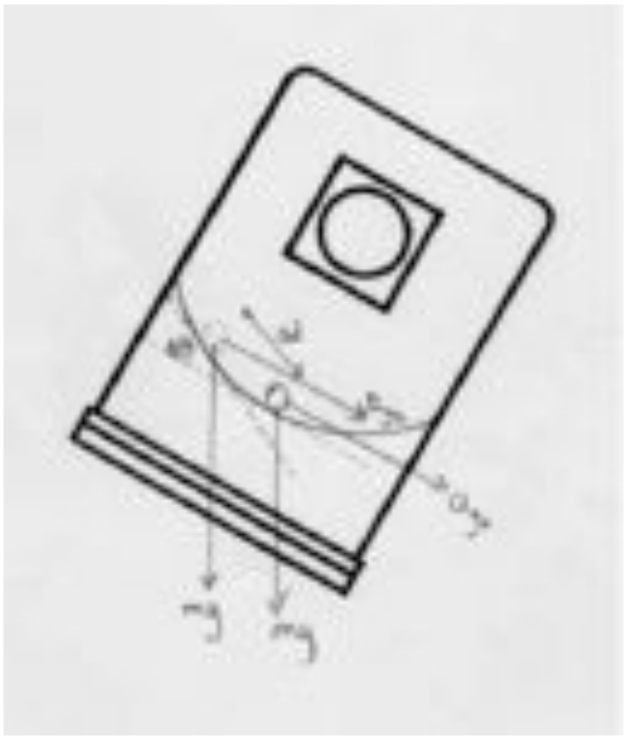


Fig 11.1: TC inclinometer ball representation

The method that has been adopted for finding the balls position is to calculate the total acceleration. The acceleration on the ball comes from two sources; the gravitational acceleration pulling the ball down and the centrifugal acceleration pushing the ball up. The diagram above shows the two accelerations involved. The y acceleration shown is the centripetal acceleration as measured by the AHRS. It can be seen that if the ball rests in the centre then this is the acceleration at a tangent to the surface, which is the acceleration required. If the ball moves from the centre then the measured acceleration will have to be resolved to find the acceleration that is at a tangent to the curved surface. For example if the ball moves a distance delta x along the surface, then the tangent to the surface will have moved through an angle of delta theta. The required tangential component will then be the measured acceleration multiplied by cosθ. The full method is given below;

1) Start with initial ball position d_0
2) Calculate $\Delta\theta$ as follows:

$$\Delta \theta = \frac{\Delta d}{r}$$

   Where r is the radius of curvature
3) Add this to the roll angle to get total theta

$$\theta = \text{roll} + \frac{\Delta d}{r}$$

4) Calculate total acceleration on the ball at that angle

$$a_T = \cos\theta - mg\sin\theta$$

5) From this acceleration calculate the distance travelled (Δd)

$$\Delta d = a_T \Delta t$$

   Where delta t is a variable in the LabVIEW code.
6) Find new starting

$$d = d_0 + \Delta d$$

7) Start gain at step 2 with this new position

Initially, this was hard to employ in LabVIEW, as one cannot simply join up the formulas with wires in the flow pattern as LabVIEW then encounters a closed loop that it cannot solve. What is required is something called a shift register. For this, a while loop surrounds what is required to be iterated. On the left side and on the right side of this, a block of a shift register is placed. In this case it is the value of d that is required to be carried though and re-evaluated. Therefore instead of creating a physical loop with the LabVIEW wires, the start value of d is wired to the left shift register and after the calculations have been performed the new value is wired to the right shift register. This means that on the next iteration of the while loop, the value on the right side becomes the value on the left side. An initial value can be set for the first iteration by wiring a constant to the left side outside the loop. However this is not required in our situation with the turn coordinator.

It is notable that in these equations there are two main variables: delta t and mg. Delta t should be fixed to the time delay that is present in the while loop, this ended up being 200ms. The variable mg was obtained experimentally by connecting the AHRS and changing the value until a realistic looking display was obtained. A reasonable value for mg was found to be 0.01.

These equations represent only a rough initial model of the turn coordinator ball. The main element that is missing from this method is any form of viscous damping. A viscous liquid damps the movement of the ball in a turn coordinator. This is not modelled in anyway in the coordinators code. It was found however that the inherent sluggishness of the software and processing time of the CPU created reasonable damping as a positive side effect.

11.2 Other Instruments

Unfortunately it has not been possible to do any programming of the remaining instruments: airspeed indicator, altimeter and vertical speed indicator, as these all require data from the air data system, which at present is not available to work with. However there is at present, ongoing work with an ultrasonic altimeter, which it is hoped may form part of future work.

12 Final System

All of the previously detailed components of the GCS were assembled in a final system. When the software was working properly it was installed in the VW Caravelle to check that it all fitted in place. The final system consists of one main VI that is stretched across two screens. . The final displays for the cockpit screen and for the flight data screen can be seen in figures 12.1 and 12.2 respectively. Full documentation, as produced by LabVIEW can be found in appendix D

12.1 Method of testing

In the final design, all the data will be going through one data link transmitter/receiver. However, for the purposes of testing the working components of the GCS at this stage, the hardware has been plugged directly into, the back of the computer. A computer has been used that has two COM ports. Into one of them, is plugged the AHRS and into the other, is plugged a null modem as described in chapter 6. When the final system was tested, this null modem was plugged into the back of a laptop. The final LabVIEW code that was used in the laptop can be seen in appendix D. The CPU, the AHRS and the monitor were all powered off of the 60Ah batteries as shown in figure 3.1. It was only possible to test one monitor in the van, due to the fact that an error developed with the second graphics card.

12.2 Optimisation

The software contains some essential time delays that allow time for data transfer. There for there are several areas in it that require experimental and logical optimisation. There are some parts of the program that do not need to be continually updated, and thus a slower refresh rate is

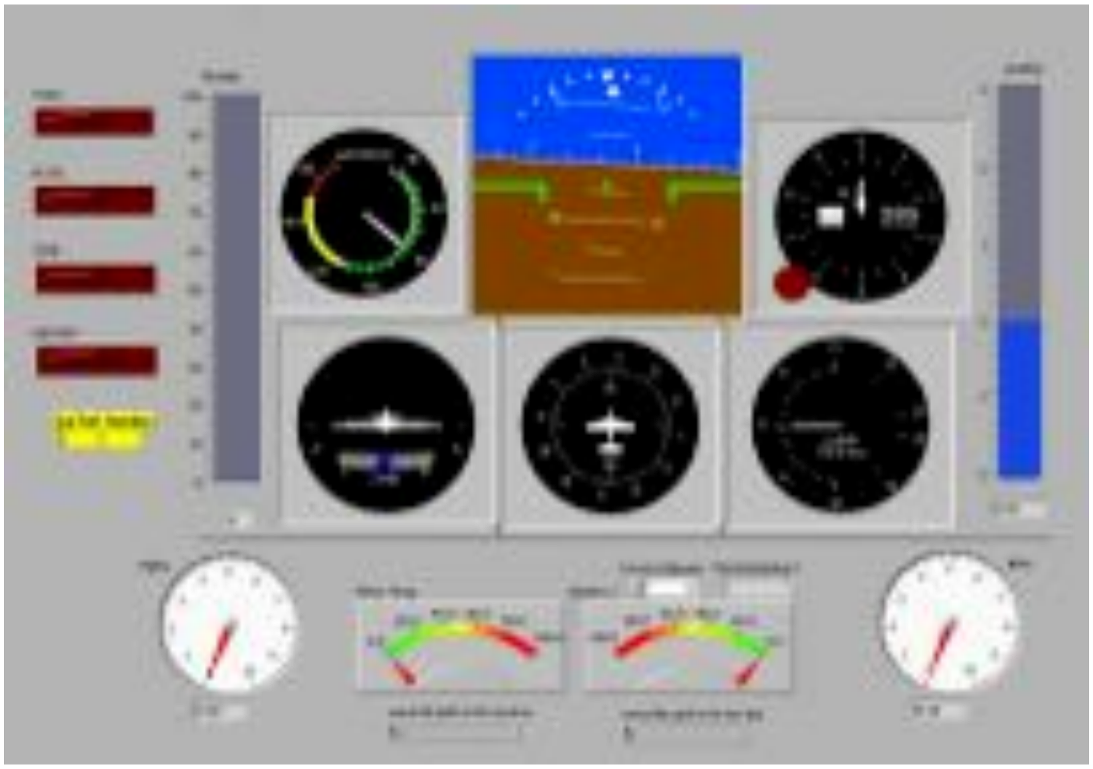


Fig 12.1: Final Cockpit display

possible in order to reduce the amount of time spent on unessential program sections. For example, it was felt that the timer that shows the time left in the batteries does not need to show the time with a greater accuracy that one second. Therefore in this part of the VI, a time delay of 1000 milliseconds has been introduced between repeats of the while loop.

The second section that does not need to be updated that fast is the monitoring of the engine temperature. This because it is not expected that the temperature will be capable of changing particularly quickly. This section has therefore been given the same refresh rate as that of the timer section: 1000 ms.

The refresh rate of the two main sections is influenced by other factors. With the sections that transmit joystick coordinated and receive AHRS data, it is desirable to have the refresh rate as fast as possible. This is so that the software generates a smooth display. However, there is a maximum refresh rate that is governed by the speed at which data can be transmitted using the RS232 format. With the section that gathers data from the AHRS, every loop of the while loop must wait for enough time for one whole line of data to be written to the buffer. There is nothing that can be done to change this speed, as it is predetermined by the software configuration in the AHRS. It was found that the lowest time period that must be incorporated as a delay in this section to let all data be received is 200 ms. This means that the cockpit displays and flight data displays are refreshing at a rate of 5Hz. This unfortunately results in quite a jerky display.

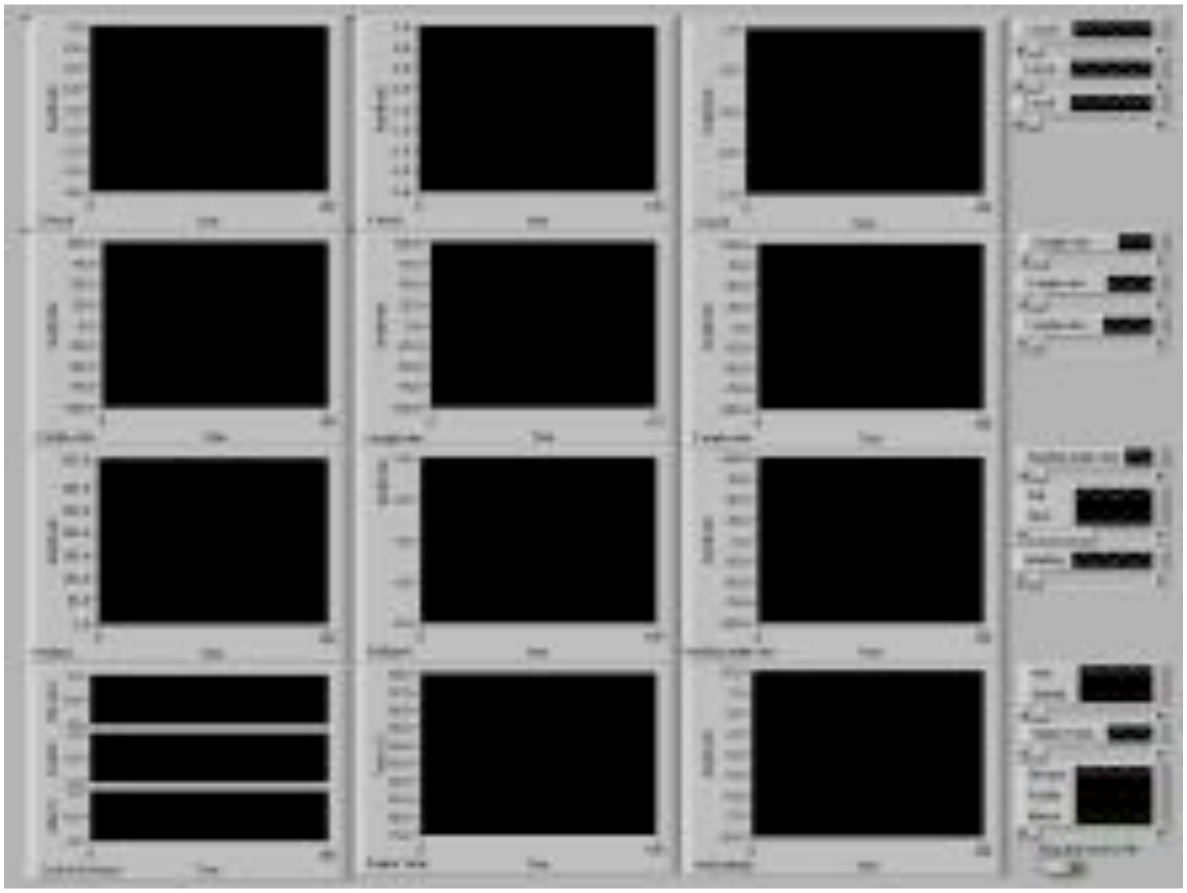


Fig 12.2: final flight data display

Data is sent from the AHRS at a standard speed of 9600 Kb/sec and this is the limiting factor in the quality of the displays on the GCS. It can be seen that one way of solving this problem would be to develop a way of reading the data one column at a time. Although a lack of time has prevented this being accomplished during this thesis.

The second section that is limited by the speed at which data can be transmitted is the joystick coordinates section. However this is not limited so much by the main code, but how the system is being tested. This is because the limiting factor comes from the speed at which the data can be read by the laptop. Generally, data can be transmitted a lot faster than it can be read. However, despite the fact that the real system will not involve a laptop, it is assumed that the same data transmission hardware will be used and thus the same problem will exist on the aircraft. It was found that the data in this section could involve a faster refresh rate than that of the AHRS and that only a pause of 100 ms was required. This is due to the fact that only four columns of data are being transmitted from the joystick as opposed to 7 from the AHRS.

12.3 Set-up in the van

The method of installation in the van is as follows;

- Remove central row of seats
- Lift the bare rack into the van and fix it in place using 4 M10 bolts. The rack is fixed to the left set of bolts from the double seat and the left set of bolts from the single seat. The method of fixing the rack to the floor can be seen in fig 12.3.

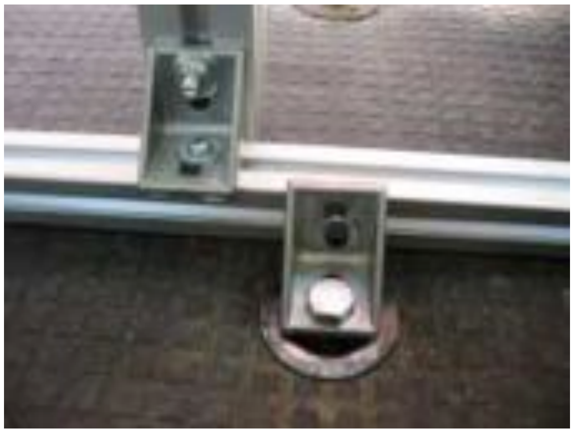


Fig 12.3: Attachment of rack to car floor

- Both the monitors are fixed in place by moving the bar in front of them and the two bars below them.
- The CPU, batteries and AC transformer are then placed on the lower shelf and secured with some sort of bungee cord.
- Everything is then wired up and switched on.

As expected, there was a slight lack of space and it is recommended that the smaller of the available personnel should carry out the installation. However, with a few practice attempts, it should be possible for the system to be installed fairly rapidly.

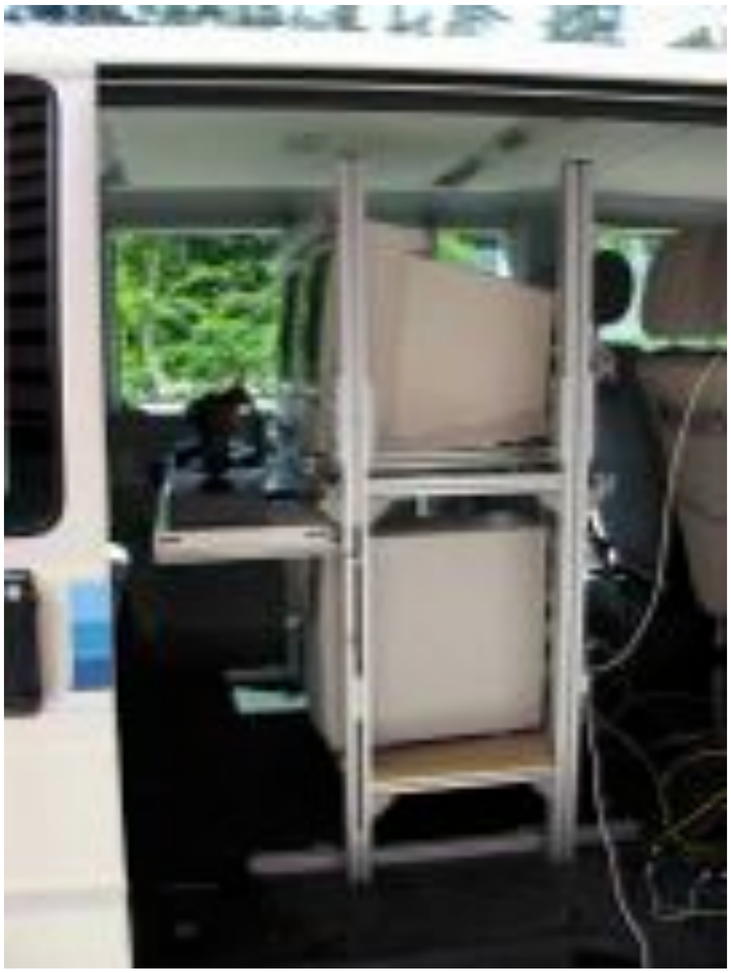


Fig 12.4: Installed GCS

12.4 Software

The final software at the end of this thesis, consists of five sections. Three of these sections are completed, the other two are just temporary as there is insufficient knowledge about the hardware for them to be completed. These two sections are the control surface monitoring section and the motor temperature section

12.4.1 Joystick section

This consists of a sequence structure with three parts. First the correct port is initialised, using the users desired port input. A small pause of 10 milliseconds allows the port to be fully initialised. Then the joystick coordinates are converted to strings. These strings are then joined together but before they are joined, the individual coordinates are marked with a marker character so that the string can be split into its individual coordinate components at the other end. This long string is then written to the users desired COM port.

12.4.2 AHRS data section

In this section an alternative user defined COM port if initialised using another sub VI. The input to this also consists of the desired baud rate and a path for any errors. This then provides a reference input for the read serial function. In its current set-up, this simply reads the number of bytes waiting at the port. This function is inside the loop that contains a pause of 200 milliseconds. So for every iteration of the loop it waits for enough time for at least a whole line of data columns to be written. The start of the line is then found by matching one of the characters that always appears at the start of each line. The string is then split into the various constituent components of data by assuming that they occur in the same place in the string every time. These sub strings are then turned into fractional numerics. The AHRS data is then wired to all the displays on the flight data screen. It is also used to power the output of the artificial horizon, heading indicator and the inclinometer in the turn coordinator. The sub VI 'TC' contains the arithmetic for the turn coordinator calibration. The Z-axis acceleration data is used to produce a visual and aural warning if a predefined acceleration is exceeded. There remain several unwired components in the bottom right of the section. Unfortunately nothing can be done with these until a clearer picture of the hardware is obtained. When the save button is pressed, then the loop stops and the data is saved to a spreadsheet file that can be viewed and manipulated using a program such as excel.

12.4.3 Timer section

The time remaining is obtained by subtracting the real time in that iteration from the real time at the start plus the desired run time. This is turned into a percentage and displayed on a dial that displays values from 0 to 100. This is then wired to a visual and aural warning, which are triggered at a predefined time from the end. In the present set up this is a constant of 300 seconds from the end. The loop is refreshed every second and can be stopped by pressing a switch on the front panel.

12.4.4 Control surfaces and engine temperature section

These sections have been included so as to get the front panel looking as it should be. Although the inputs bear no resemblance to reality. The inputs to the control surface readouts are manual inputs. That of the motor temperature comes from a random dumber generating VI.

13 Conclusion

13.1 Summary of work completed

The four joystick coordinated have been read in LabVIEW. This has been tested inside a single VI, using a Loopback adapter through the serial port and to another computer using a null modem. All required displays and indicators have been arranged and positioned on two 17″ screens using a single VI that is stretched between the two screens. The 7 most important columns of data have been read from the AHRS and displayed in LabVIEW. These have been used to display accurately the output of both the flight data graphs and to the ActiveX control that use the attitude data. A timer system has been set up that is used as a measure of the battery life left in the system. Visual and aural warnings have been implemented for both the Z-axis acceleration and for when the batteries are running low. A small series of functions have been implemented so that the AHRS data throughout the flight is recorded and saved to a spreadsheet compatible file. The CPU and two screens have been wired to two 12V car batteries using a couple of transformers – a DC/DC converter for the CPU and a DC/AC converter for the two screens. An aluminium rack has been constructed to hold everything. All the components were then set up and secured in the VW Caravelle that will be used in the final set-up.

13.2 Conclusions from development

One of the first lessoned learnt, during the development process was that the USB system should be used for the joystick communication rather the through the Gameport. It took a while to get the joystick section working, but once it was, it proved a very simple and versatile system.

The other noticeable conclusion that was draw during the development was that the ActiveX controls are very powerful tools for the quick development of software. Without these controls, the GCS could not have been completed nearly as quickly.

13.3 Conclusions from final system test

It was known before hand that weight would always be an issue and this was confirmed during the installation of the system in the van. It would have been preferable to have the screens and CPU fixed to the rack before it was placed in the van however this would have made the whole structure too heavy. It could be seen that the monitors were responsible for the majority of the weight. Therefore, despite the extra cost it has been concluded that it would probably be preferable to use LDC displays. This would also have the added bonus of increasing the endurance of the system. One problem with this is that it is quite hard to obtain 17″ TFT screens which is the size that would be required.

Other than slight weight and size constraints there were no other physical conclusions to draw from the final system. However there was one fairly significant result to note from the software. It could be seen that the slow refresh rate on all the data transmission produced a display that was far more sluggish and unclear than was desirable. It also meant that control would be compromised, as the pilot would not have highly responsive control over the aircraft. Since the primary aircraft being used with this system is, LUCAS – a scaled attack aircraft, it would be desirable to have a more responsive control. The solution to this would be to either use an alternative to RS232 or to program the code so that on column is read at a time. The latter method would almost certainly work for reading the AHRS data as its transmission speed cannot be changed. However it is unlikely that transmission would be speeded up by sending one joystick coordinate at a time. What would need to be changed is the code on the laptop that is reading the joystick coordinates. One alternative that may speed the system up would be to use two CPU's – one for the transmission of the joystick signals and one for reading the AHRS. However, as it has been seen, the majority of the sluggishness comes from the way the data is transmitted and a second CPU is unlikely to speed the system up enough to justify the increased complexity and weight.

One more problem with the current software design is that when one or both of the aural warnings are played then the system dramatically slows down. It can be seen that playing wave files while continuing to run the VI takes more CPU power than is available. The task manager was open while the VI was run during testing and it was noticed that at all times the CPU usage was on 100 %, thus indicating that perhaps a faster computer was required.

13.4 Future work

As much work has been completed in this thesis as was possible given the conditions. However there are still a number of things that need to be done. Firstly there are several aircraft sensors that still need to be incorporated into the GCS when they are defined. Once the air data system is available it will need to be incorporated so that the primary static/dynamic pressure indicators can be programmed. This system is required so that a display can be obtained for the air speed indicator, the altimeter, the rate of climb indicator and the turn coordinator. To more sensors that are required to be integrated into the GCS are the RPM counter and the motor temperature sensor.

More hardware integration is required in the form of transmitting the data using the radio data link system. But at the time of completing this thesis, the data link system has still some on going work being carried out. It would also be desirable to actually get a definition of the software in the aircraft and get the joystick signals actually being processed and outputted to some real control surfaces.

It can also be seen that the software will require some improvements. It is desirable to get the data being read slightly faster. It is anticipated that this could be done by reading the data columns individually instead of a line at a time. An additional display that could form part of future would is a virtual aircraft attitude indicator that shows the instantaneous position of the aircraft based on a graphical representation.

14 References

[1] Pallett, E.H. "Aircraft Instruments" Longman Scientific & Technical, Essex, 2nd ed. 1991

[2] Wells L.K. Travis J. "LabVIEW for everyone" Prentice Hall, New Jersey 1997

[3] "http://www.ni.com//" Website of the LabVIEW manufacturers

[4] Michelson R. Newcome N. "21st Century Aerial Robotics 2001" Course Notes

[5] Wiener E.L.. "Human Factors in Aviation", San Diego Academic press1988

[6] "http://www.airliners.net/" Website with aeronautical picture database.

[7] "http://www.fas.org/" Federation of American Scientists web site

[8] "http://www.lammertbies.nl/" Personal website with info on serial communication

15 Appendix A

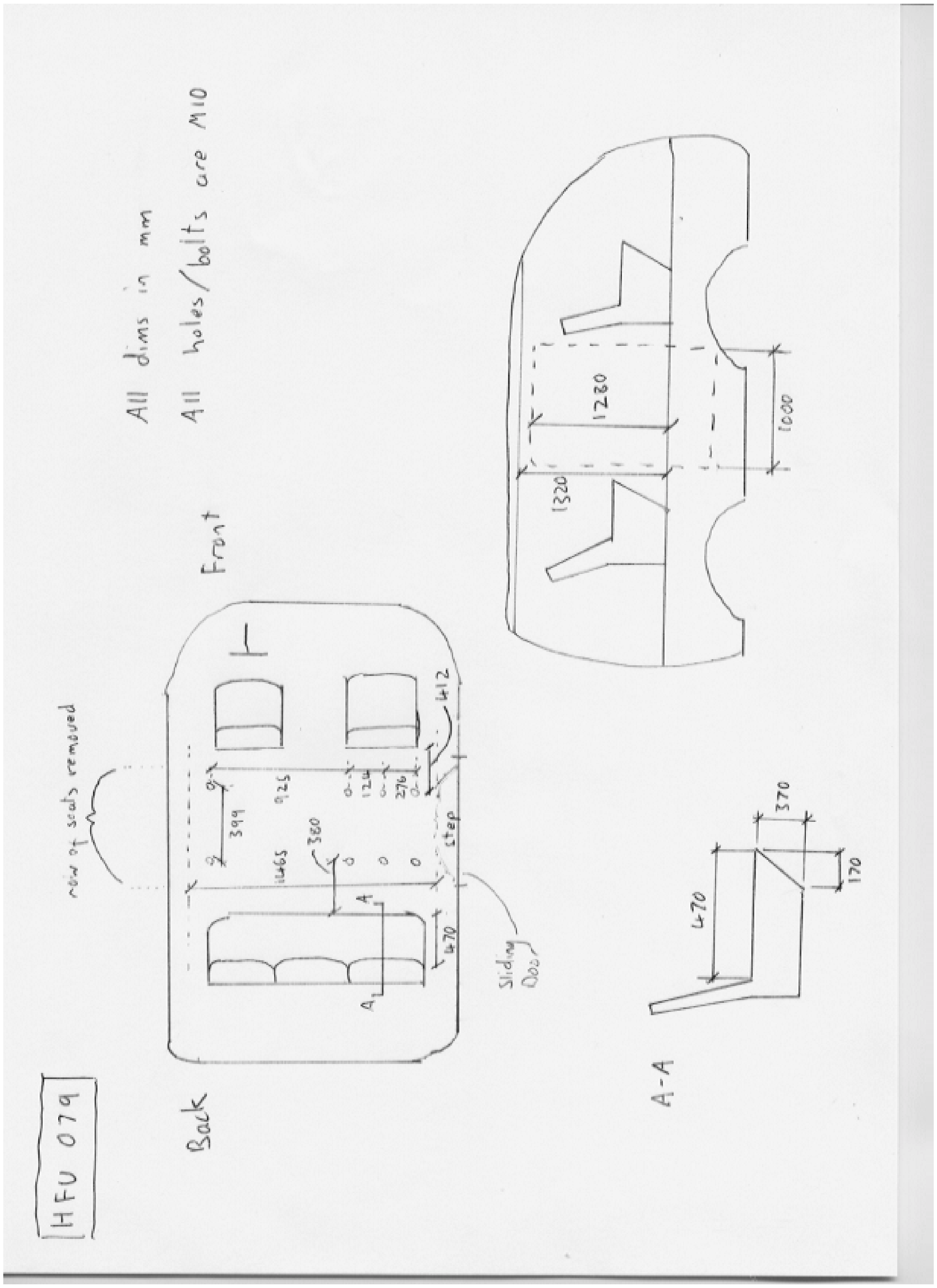


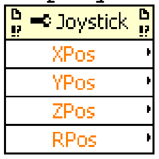
Fig A.1: Van Dimensions

16 Appendix B

LabVIEW Functions, controls and indicators

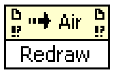
Control	Indicator	Data Type
SGL	SGL	Single-precision floating-point numeric
DBL	DBL	Double-precision floating-point numeric
I8	I8	Signed 8-bit integer numeric
I16	I16	Signed 16-bit integer numeric
I32	I32	Signed 32-bit integer numeric
U8	U8	Unsigned 8-bit integer numeric
U16	U16	Unsigned 16-bit integer numeric
U32	U32	Unsigned 32-bit integer numeric
abc	abc	String
▭	▭	Path
TF	TF	Boolean
▦	▦	Cluster – Encloses several data types. Pink colour indicates the elements are different types
▭	▭	Reference Number. Also represents an ActiveX control
I/O	I/O	I/O Name – Passes DAQ channel names, VISA resource names, and IVI logic names you configure to I/O VIs to communicate with an instrument or DAQ device

Property Node



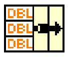
Sets (writes) or gets (reads) ActiveX object property information.

Invoke Node



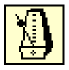
Invokes a method or action on an ActiveX object.

Bundle



Assembles all the individual input components into a single cluster.

Wait until next ms multiple



Waits until the value of the millisecond timer becomes a multiple of the specified millisecond multiple.

Get date/time in seconds

Returns a time-zone-independent number that contains the number of seconds that have elapsed since 12:00 a.m., Friday, January 1, 1904, Universal Time.

Format Date/Time

Gives you the ability to display the date and time in a format you specify.

VISA read

Reads the specified number of bytes from the specified device or interface.

VISA write

Writes data to the specified device or interface.

Match pattern

Searches for regular expression in string beginning at offset, and if it finds a match, splits string into three substrings.

String Subset

Returns the substring of the original string beginning at offset and containing length number of characters.

Fract/Exp to number

Interprets the characters 0 through 9, plus, minus, e, E, and the decimal point (usually period) in string starting at offset as a floating-point number in engineering notation, or exponential or fractional format and returns it in number.

Snd play wave file .vi

This VI plays a PC audio waveform file from disk.

Serial port init.vi

Initialises the selected serial port to the specified settings.

Serial port write

Writes the data in string to write to the serial port indicated in port number.

Bytes at serial port

Returns in byte count the number of bytes in the input buffer of the serial port indicated in port number.

Serial port read

Reads the number of characters specified by requested byte count from the serial port indicated in port number.

Initialise array

Creates an n-dimensional array in which every element is initialised to the value of element.

Replace array subset

Replaces an element or array in an array at the point you specify in index. When you wire an array to this function, the function resizes automatically to display index inputs for each dimension in the array you wired. If you do not wire an index for a dimension, Replace Array Subset replaces all the elements in that dimension.

Concatenate strings

Concatenates input strings and one-dimensional arrays of strings into a single, output string. For array inputs, this function concatenates each element of the array.

17 Appendix C: AHRS test data

Time	Roll	Pitch	Hd	X acc	Y acc	Z acc	X rate	Y rate	Z rate	Hd-rate
0.301	0.7	-0.1	338.3	0	0	-1	0	0	0	0
0.441	0.2	-0.2	332.5	0	0	-1	0.4	-0.7	0	0
0.671	0.3	-0.2	332.5	0	0	-1	-0.1	0.3	0	0
0.982	0.3	-0.1	332.5	0	0	-1	-0.3	0.2	0	0
1.302	0.2	-0.1	332.5	0	0	-1	-0.2	-0.4	0	0
1.583	0.5	-0.1	332.4	0	0	-1	-0.3	-0.7	0	0
1.883	0.9	-0.1	332.4	0	-0.01	-1	0.3	0.2	0.1	0.2
2.183	-0.2	-2.5	333	-0.02	0	-1.02	0.4	-2.6	-1.4	-2.1
2.484	0.7	-1.6	332.8	-0.02	-0.01	-1.02	-2.8	1.8	6.2	2.9
2.794	0.4	-1.6	333.3	-0.02	-0.01	-0.98	-19	1.4	6.2	5.2
3.085	-8.8	-1.6	334.3	-0.03	0.06	-0.98	-46.5	-3.9	7.4	4
3.385	-26.2	-2.8	337	-0.04	0.25	-0.94	-46.7	-7.3	7.9	7.1
3.686	-39.4	-3.3	339.7	-0.05	0.5	-0.84	-40.9	-6.8	6	9.6
3.986	-51	-4.1	342.6	-0.05	0.69	-0.7	-14.4	1.5	4.9	-0.6
4.286	-42.7	-5.4	342.5	-0.07	0.73	-0.67	42	2.2	-9.7	-1.1
4.587	-27.5	-6.3	341.4	-0.09	0.58	-0.78	68.2	-2	-2.8	-9.2
4.877	-6.7	-6.6	339.7	-0.11	0.3	-0.92	44.6	4	-8.6	-3.6
5.178	11.1	-5.7	337.6	-0.1	0.01	-0.96	87.9	-15.7	-9.1	-12.7
5.478	34.8	-6.2	335.6	-0.1	-0.35	-0.89	35.7	5.8	0.6	4
5.779	46	-3.9	334.2	-0.09	-0.57	-0.8	17.9	4.3	4.4	7.5
6.079	46	-3.6	334.7	-0.06	-0.69	-0.71	-53.4	5.5	11.3	11.9
6.38	27.1	-4.4	335.9	-0.07	-0.56	-0.81	-34.5	-3.2	-8.8	-11
6.68	11.5	-5.4	335.7	-0.08	-0.34	-0.91	-47.4	1.1	-2.3	2.9
6.98	4.5	-4.8	335.5	-0.08	-0.15	-0.96	-5.7	-19.4	-3.5	-10.3
7.291	3.4	-15.4	333	-0.16	-0.07	-0.98	3.3	-42.8	-4.3	-4.4
7.581	3.6	-30.8	332.2	-0.38	-0.07	-0.91	-6.8	-30.2	8.2	3.3
7.882	2.3	-38.7	332.3	-0.54	-0.05	-0.83	-3.7	-12.1	8.5	16.1
8.182	0.1	-33.8	333.8	-0.59	-0.03	-0.82	-13.5	29.3	20.2	17.3
8.483	-0.7	-22.3	335.7	-0.48	0	-0.87	18.2	51.6	-0.8	-2.7
8.783	2.3	-6.2	336.2	-0.24	-0.02	-0.95	7.3	41.1	1.8	2.2
9.083	2.7	5.5	338.3	-0.04	-0.04	-0.99	19.5	45.6	9	10.3
9.384	4.6	18.9	339.9	0.17	-0.05	-0.96	21.1	33	-6.6	-0.9
9.684	4.6	28.4	340.4	0.36	-0.07	-0.91	14.3	27.3	2.1	8.2
9.985	5.3	35	341.9	0.51	-0.06	-0.84	-0.2	4.6	-3	-7.2
10.285	3.8	27.3	340.5	0.54	-0.06	-0.86	-0.9	-51.8	-10.9	-10.7
10.586	5	11.5	339	0.36	-0.04	-0.92	-2.7	-53.5	-2.3	-7.8
10.886	1.5	-0.2	336	0.08	-0.03	-0.99	-9.3	1.3	-9.6	0.3
11.186	0.7	-0.9	339.5	0.01	-0.04	-1	5.5	6.2	41.6	50.4
11.487	0	0.4	350.4	0	-0.01	-0.99	-14.6	-4.9	39.4	17.9
11.787	-1.1	-0.5	0.5	0	0.01	-1	8.6	0.3	32.1	32.8
12.068	-1.5	-1.1	14.2	0	0.01	-0.99	5.1	-7.1	52.3	58.9
12.388	-0.2	-0.8	20.5	-0.01	0.04	-1	-9.5	-4.5	-3.2	-11.9
12.679	-1.3	-0.7	11.2	-0.01	0.03	-1	4.2	6	-37.9	-37.2
12.979	-0.8	-0.4	358.7	0	0.03	-0.99	-2.9	-7.2	-58	-57.3
13.29	-0.7	-1.8	338.5	-0.02	0.01	-1	2.6	9.8	-49.7	-45.4
13.58	0.6	-0.1	329.8	-0.01	0	-0.99	-1.3	4.8	-31.2	-39
13.88	1.4	-1.2	315.8	0	0	-0.99	-3.5	-7.8	-55.6	-51.6
14.171	2.4	0	304.1	0	-0.02	-1.01	-4.3	19.3	-35.6	-36
14.481	3.4	0.9	289.2	0.01	-0.02	-0.99	-1.9	-9.9	-33.5	-29.1
14.782	4.1	3.4	280.9	0.03	-0.07	-0.98	-6.7	-3.1	-3.4	-1
15.082	1.5	1.4	286.5	0.04	-0.06	-1.01	4.2	-15	57.2	48.1
15.383	2	-2.2	303.7	0	-0.04	-1	-3.7	-0.2	49.9	55.5
15.683	1.1	-3.2	318.1	-0.03	-0.02	-0.99	0.3	-4.7	52	52.1
15.983	0	-4.2	328.6	-0.05	0	-0.98	9.6	-8.1	45.6	47.9
16.284	-0.6	-3.4	334.7	-0.06	0.02	-0.99	-3.2	8.1	-8.2	-10.3
16.584	0.1	-2.8	334.7	-0.04	0	-0.98	-5.5	11.4	-3.5	-4
16.885	-1	-0.8	333.7	-0.02	0.01	-1.01	-0.3	3.5	1.1	-0.7
17.185	-0.9	0.2	332.9	0	0	-1.01	-7	-0.5	0.1	0.1
17.486	-3.4	0.2	332.9	0	0.03	-1	-10.7	-3.2	-0.2	0
17.786	-3.6	0.1	332.9	0	0.06	-1	0	-0.1	0	-0.1
18.096	-3.7	0.1	332.9	0	0.06	-0.99	-0.2	0	0	0
18.367	-3.7	0.1	332.8	0	0.06	-0.99	-0.2	0	0	0
18.697	-3.8	0.1	332.8	0	0.06	-0.99	-0.1	0	0	0
18.988	-3.8	0.1	332.7	0	0.06	-0.99	-0.1	0	0	0
19.288	-3.8	0.1	332.6	0	0.06	-0.99	-0.1	0	0	0
19.579	-3.9	0.1	332.6	0	0.06	-0.99	-0.1	0	0	0
19.879	-3.9	0.1	332.5	0	0.06	-0.99	-0.1	0	0	0

18 Appendix D: Full Code documentation

GCSf.vi

Connector Pane

GCS

Front Panel

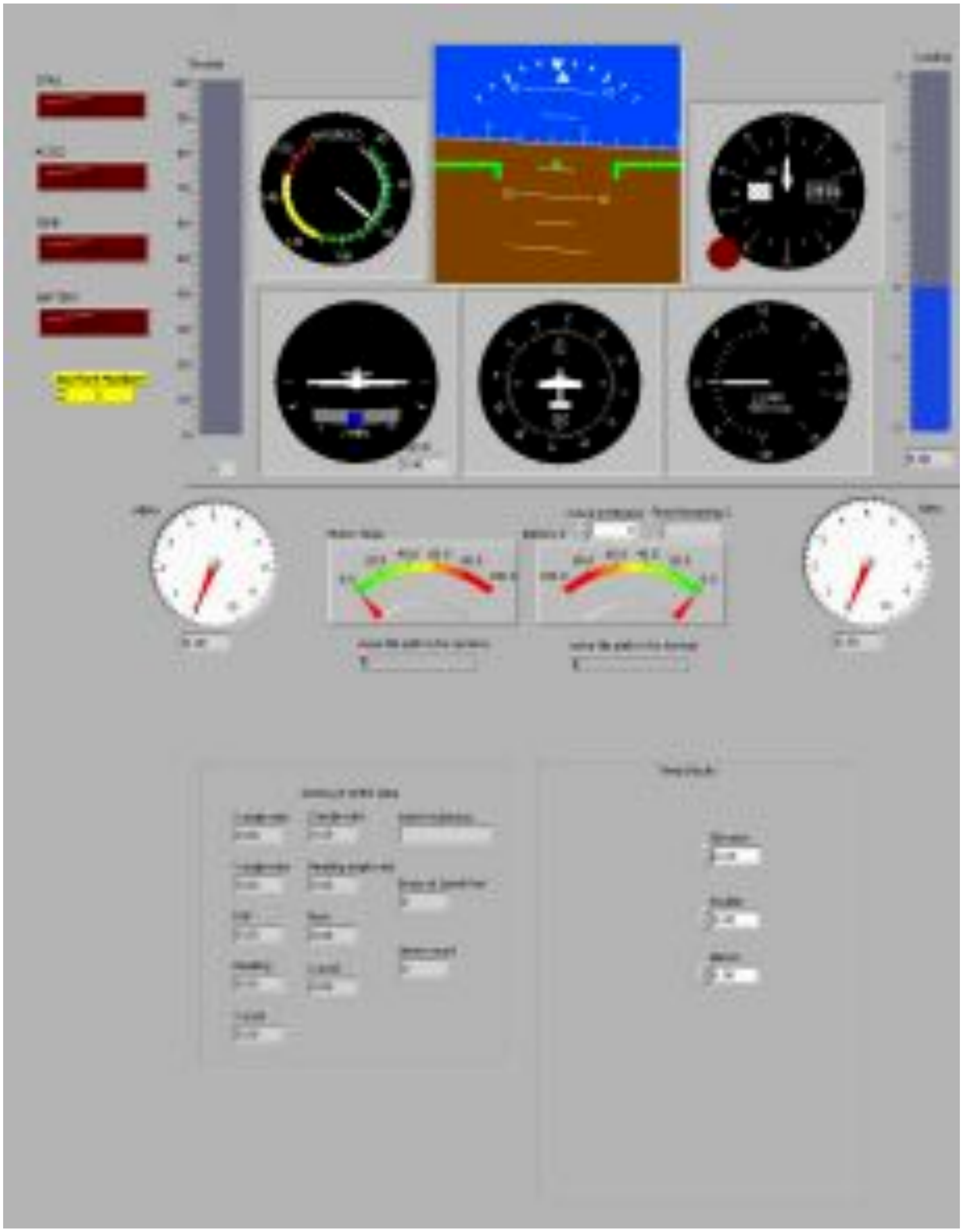


Fig D.1: Final front panel, including sections not normally visible

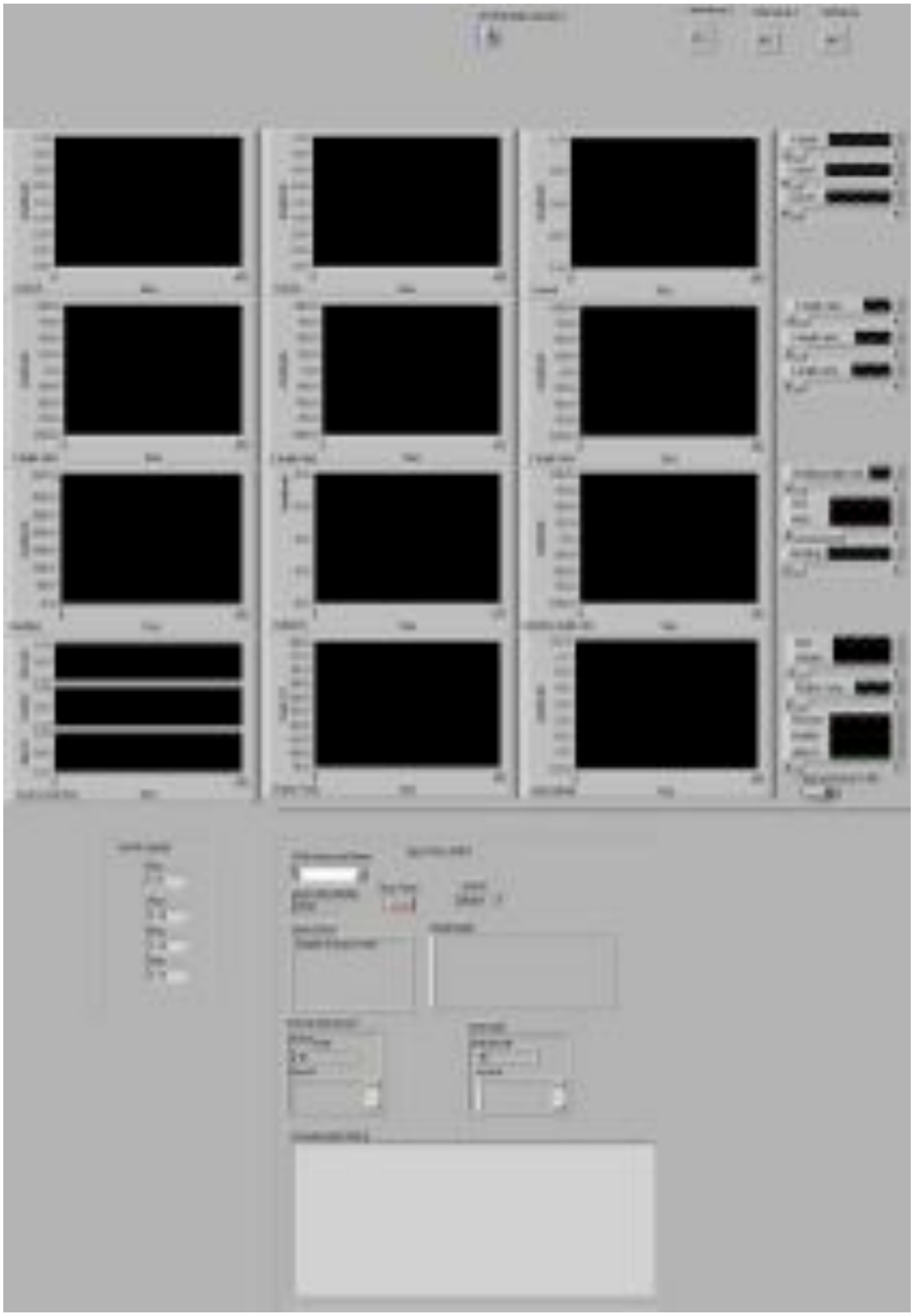


Fig D.2: Final front panel, including sections not normally visible

Controls and Indicators

AH ActiveX for the artificial horizon

Altitude ActiveX control for the Altitude indicator

Air Speed ActiveX control for the air speed indicator

Climb rate ActiveX control for the climb rate indicator

Heading ActiveX control for the heading indicator

Turn Coordinator ActiveX control for the turn coordinator

Aileron Temporary inputs for the control surface

Rudder Temporary inputs for the control surface

Elevator Temporary inputs for the control surface

Count in Minutes Input for battery life

wave file path in for low bat wave file path in is the path of the file where the sound data is stored.

Stop Timer

baud rate (9600)

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

> status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> code The code input identifies the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> source The source string describes the origin of the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

	Action Dictates whether data should be written to or recieved from the AHRS
abc	Serial Write The Serial Write control contains the string written to the selected serial port if Write is selected on the action control. The string is normally an instrument-specific command for the device connected to the serial port.
	wave file path in for accel ex wave file path in is the path of the file where the sound data is stored.
TF	Stop and save to file
	reference 3 Used to initiate joystick sub routine
I8	Joy Port Number The port to which the joystick data is to be sent
	reference 2 Used to initiate joystick sub routine
	reference Used to initiate joystick sub routine
	JOYSTICKLib.Joystick 2 ActiveX control for the joystick
I/O	VISA Resource Name Port from which the AHRS data is being read
DBL	Alpha Cockpit display
DBL	Beta Cockpit display
DBL	Loading
U8	Throttle Cockpit display
TF	STALL Cockpit warning - should be linked to angle of attack and/or airspeed
TF	ACCEL Cockpit warning
TF	TEMP
TF	BATTERY Cockpit warning
DBL	X-accel Flight data display
	Roll/pitch Flight data display
	Control Surfaces Flight data display
DBL	Engine Temp Flight data display
DBL	AoA/sideslip Aircraft data display

SGL X Pos Joystick coordinate

DBL Motor Temp Flight data display

DBL Battery 2 Cockpit display

abc Time Remaining 2 Digital cockpit read-out

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

> TF status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> I32 code The code input identifies the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> abc source The source string describes the origin of the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

U32 Bytes at Serial Port Data waiting to be read

U32 return count Amount of data that has been read

DBL Y accel Flight data display (non essential digital)

DBL X accel Flight data display (non essential digital)

DBL Heading Flight data display (non essential digital)

DBL Pitch Flight data display (non essential digital)

DBL Roll Flight data display (non essential digital)

abc match substring Shows that the start of the data line has been found

abc Serial Read The Serial Read indicator displays the string returned from the selected serial port if the Read choice is selected from the Action control. The string is usually returned from the instrument or device connected to the serial port.

SGL	Y Pos Joystick coordinate
SGL	R Pos Joystick coordinate
SGL	Z Pos Joystick coordinate
DBL	Heading angle rate Flight data display
DBL	Heading angle rate Flight data display (non essential digital)
DBL	Z angle-rate Flight data display
DBL	Z angle-rate Flight data display (non essential digital)
DBL	Y angle-rate Flight data display
DBL	Y angle-rate Flight data display (non essential digital)
DBL	X angle-rate Flight data display
DBL	X angle rate Flight data display (non essential digital)
DBL	Z-Accel Flight data display
DBL	Z accel Flight data display (non essential digital)
DBL	Heading Flight data display
DBL	Y-Accel Flight data display
abc	Concatenated String Joystick data that is being transmitted

Block Diagram

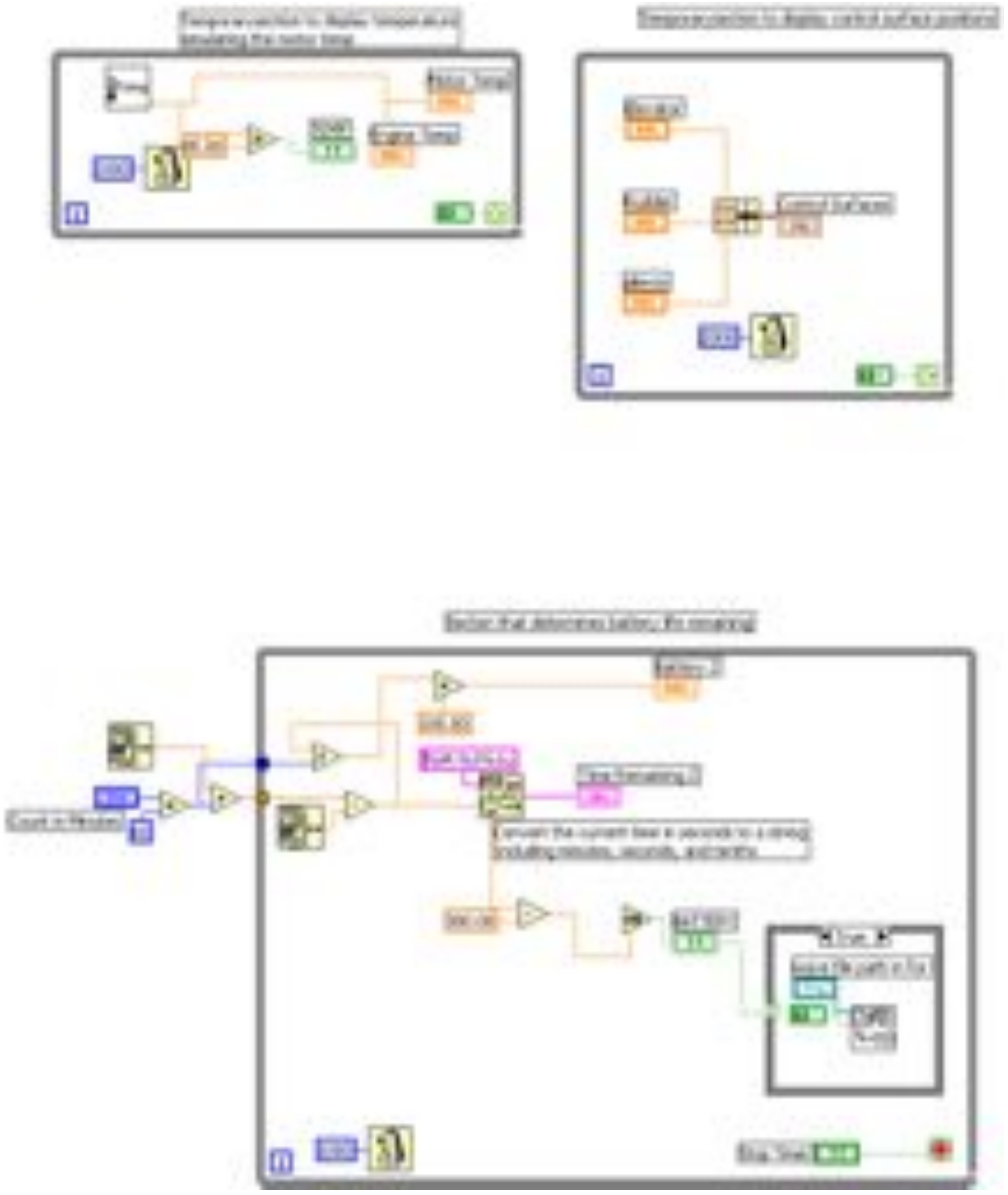


Fig D.3: Final block diagram of main program

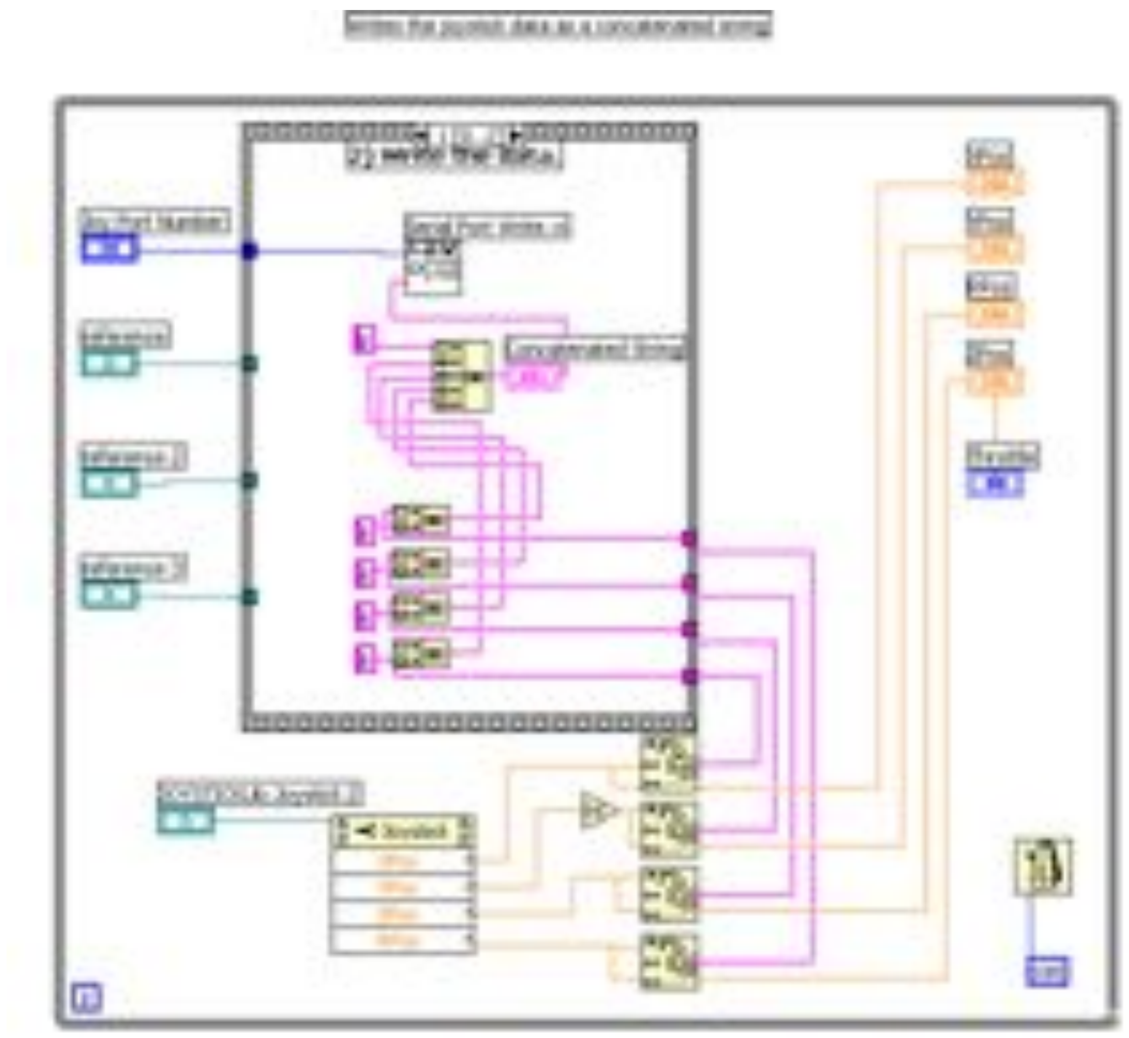


Fig D.3: Final block diagram of main program

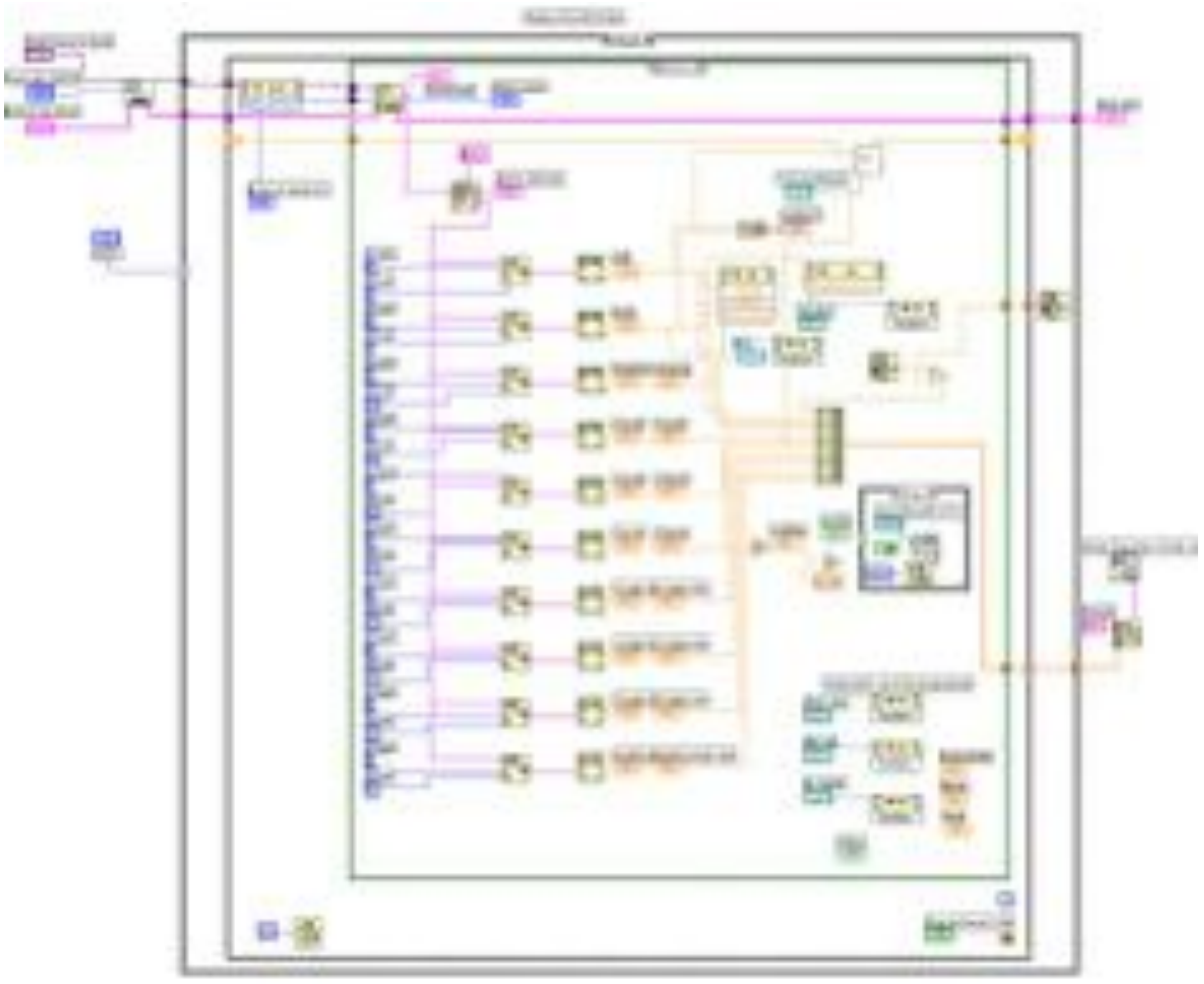


Fig D.3: Final block diagram of main program

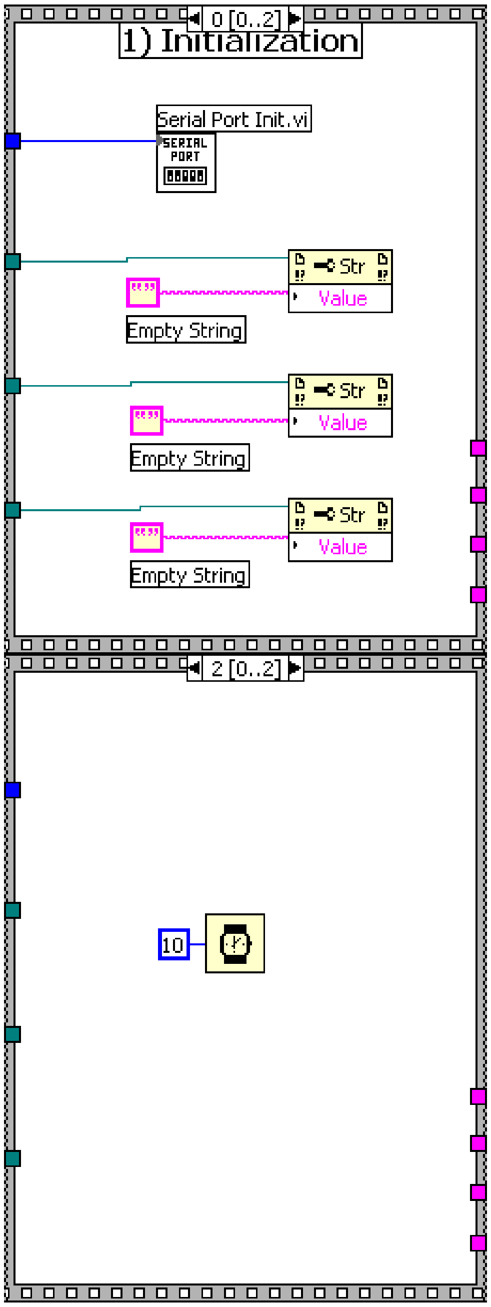


Fig D.3: Final block diagram of main program

List of SubVIs

Snd Play Wave File.vi
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\sound\lvsound.llb\Snd Play Wave File.vi

VISA Configure Serial Port
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\Instr_visa.llb\VISA Configure Serial Port

Write Characters To File.vi
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\Utility\file.llb\Write Characters To File.vi

TCsub2.vi
\\Alfen\Flyg\Phian\TCsub2.vi

Serial Port Init.vi
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\INSTR\Serial.llb\Serial Port Init.vi

Serial Port Write.vi
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\INSTR\Serial.llb\Serial Port Write.vi

Digital Thermometer.vi
C:\Program Files\National Instruments\LabVIEW 6\activity\Digital Thermometer.vi

History
"GCSf.vi History"
Current Revision: 66

Position in Hierarchy

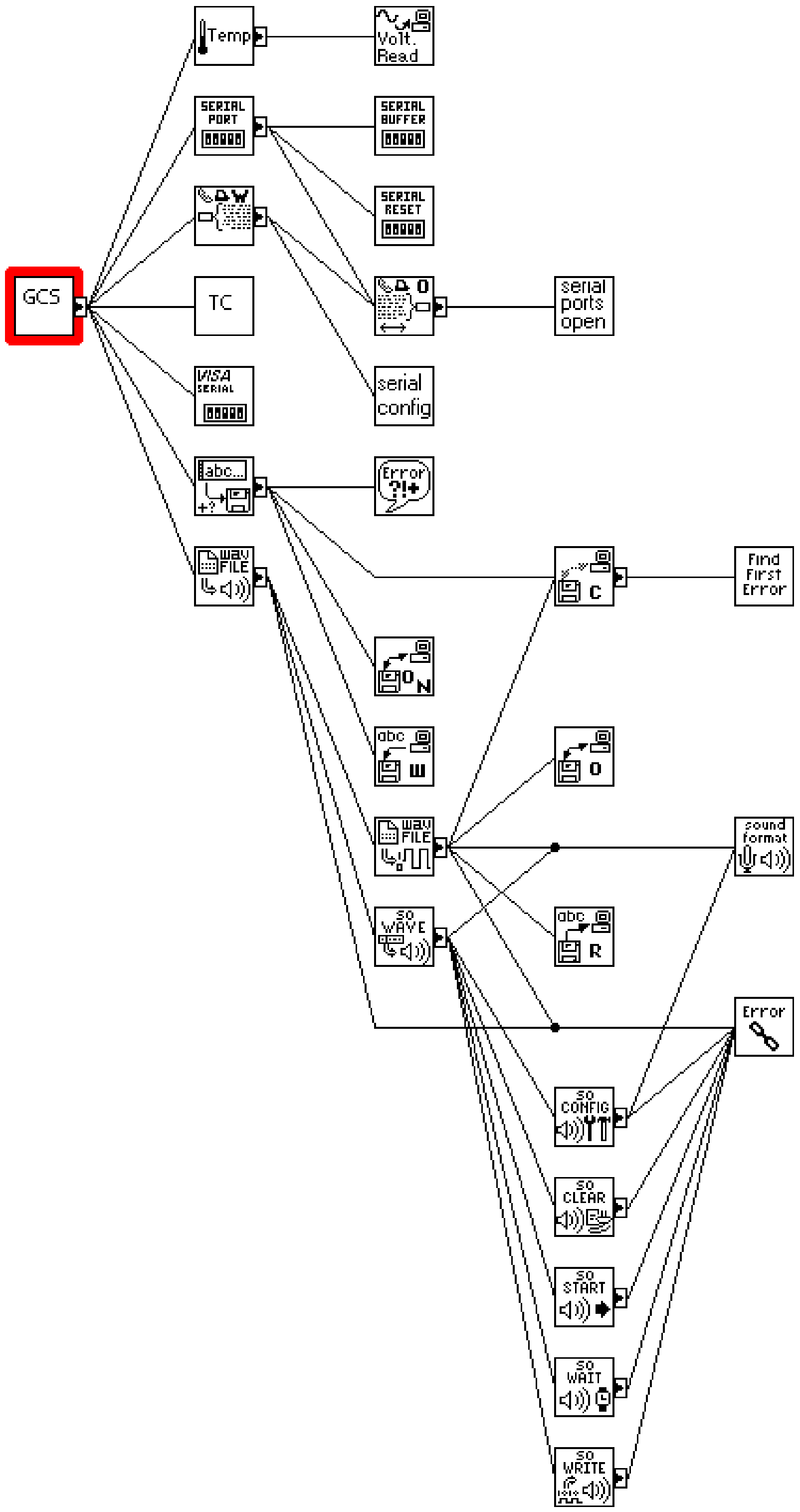
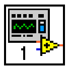


Fig D.4: Hierarchy of sub VI's in main program

joy recieve.vi

Connector Pane



Front Panel

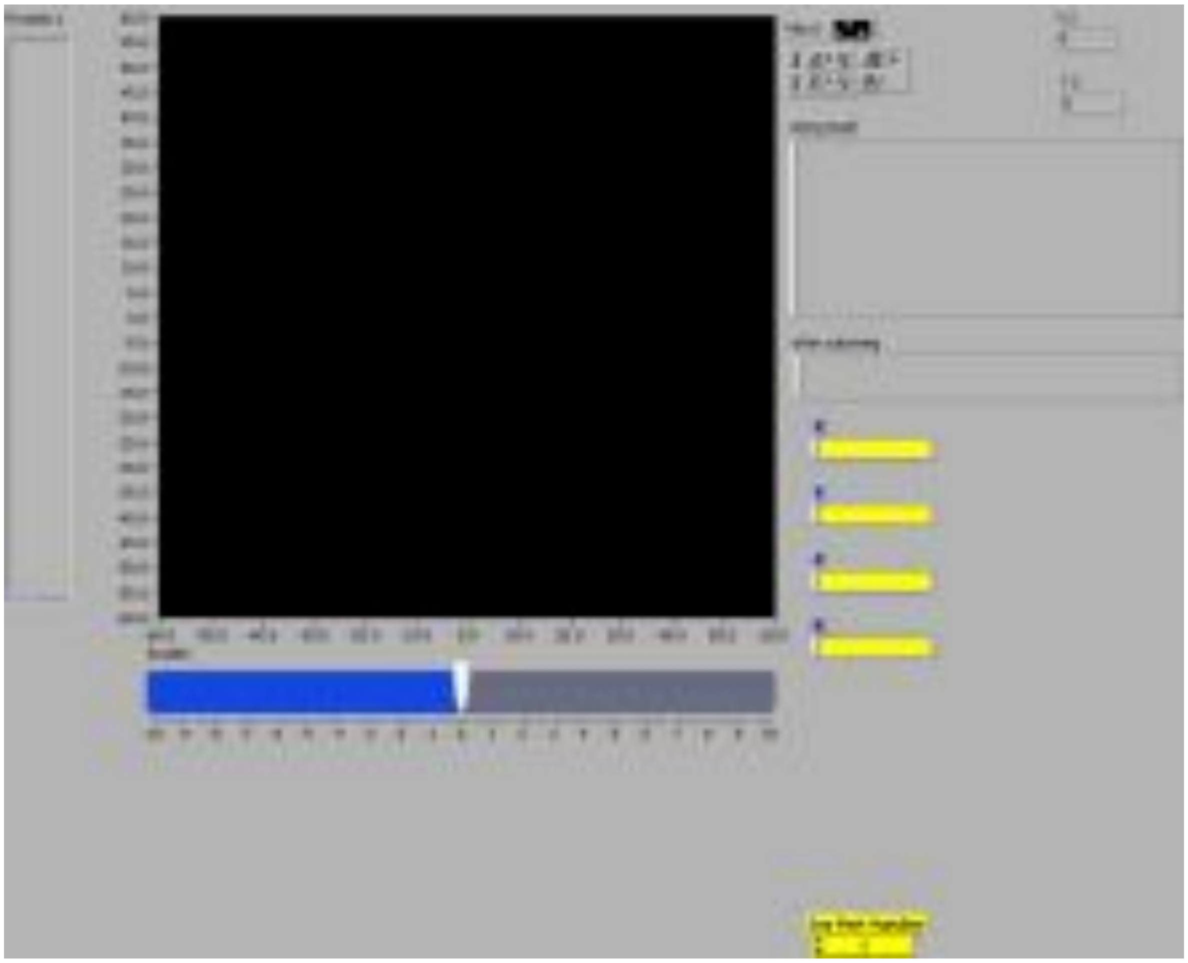


Fig D.5: Front panel from the program used on the laptop to receive the data from the joystick

Block Diagram

Fig D.6: Block diagram from the program used on the laptop to receive the data from the joystick

Fig D.6: Block diagram from the program used on the laptop to receive the data from the joystick