Development of a Rotary-Wing Test Bed for Autonomous Flight

by

STEPHANUS GROENEWALD

THESIS PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN ELECTRICAL & ELECTRONIC ENGINEERING AT THE UNIVERSITY OF STELLENBOSCH

SUPERVISOR: DOCTOR THOMAS JONES
CO-SUPERVISOR: PROFESSOR GARTH W. MILNE

APRIL 2006

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously, in its entirety or in part, submitted it at any university for a degree.

S. Groenewald: .... Date ....

Abstract

This project developed a low-cost avionics system for a miniature helicopter to be used for research in the field of autonomous flight (UAVs).

Previous work was done on a small, electrically powered helicopter with some success, but the overall conclusion was that the vehicle was underpowered. A new vehicle, the Miniature Aircraft X−Cell, was chosen for its ability to lift a larger payload, and previous work done with it by a number of other institutions.

An expandable architecture was designed to allow sensors and actuators to be arbitrarily added to the system, based on the CAN standard. A CAN sensor node was developed that could digitize 12 channels at up to 16 bit resolution and do basic filtering of the data. Onboard computing was provided by a PC/104 based computer running Linux, with additional hardware added to interface with the CAN bus and assist with timing.

A simulation environment for the helicopter was evaluated and shown to provide a good test bed for the control of the helicopter. Finally, the avionics was used during piloted test-flights to measure data and judge the performance of both the modified helicopter and the electronics itself.

Opsomming

Hierdie projek se doelwit was om ‘n lae koste vlugdata stelsel te ontwikkel vir ‘n miniatuur helikopter wat gebruik word vir navorsing in die gebied van onbemande, autonome voertuie.

Vorige werk is gedoen op ‘n klein, elektriese helikopter waar ‘n mate van sukses behaal is, maar die algehele gevolgtrekking was dat die voertuig se enjin nie sterk genoeg was om die las te dra nie. ‘n Nuwe voertuig, die Miniature Aircraft X-Cell, is gekies vir die vermoeë daarvan om ‘n groter vrag te lig, en omdat dit gebruik word vir navorsing deur ander akedemiese institute.

‘n Uitbreibare argitektuur, gebaseer op die beheer area netwerk (CAN) is ontwikkel wat toelaat dat sensore en aktueerders willekeurig by die stelsel gevoeg kan word. ‘n CAN sensor nodus is ontwikkel wat 12 analog kanale teen tot 16 bis resolusie kan versyfer en basiese filtreering van die sensor data ook kan doen. Die aanboordrekenaar was gebaseer op ‘n standard PC/104 argitektuur met Linux as die bedryfstelsel. Addsionele hardeware is bygevoeg sodat die aanboord rekenaar met die CAN bus kon kommunikeer.

‘n Simulasie omgewing vir die helikopter is ge-evalueer en daar is gewys dat dit as ‘n goeie toetsplatform dien vir beheer van die helikopter. Die werk is afgesluit met ‘n bespreking van vlugdata om die bevoegdheid van die vlugdatastelsel te bepaal.

Acknowledgements

The following people helped to make this project a success. Special thanks go to:

- Dr. T. Jones and Prof. G.W. Milne, for constant guidance while still allowing me to make this project my own.
- Mike Davis, for the hours of flight testing every Saturday morning.
- Nicol Carstens, for the vast amount of work he had to do for this project to be possible.
- Jacob Venter, for a lot of planning sessions, late nights and hard work. And of course the afternoons of fun while flying at Coetzenburg.
- Eckhardt Kuhn, Pieter Theron, Willem Hough, Jan-Corr Roos and Johan Bijker for a lot of work- and non-work related discussions.
- Cobus Jacobs, for serious debates about work, and a plethora of unrelated conversations that make him the great friend that he is.

And thank you to Hermine, for the tremendous support, and for reading every last word.

To my mother, who would have loved to see it fly

Contents

Nomenclature

f_s	Sample rate frequency
T_s	Sample time
x, y, z	Body fame positions
u, v, w	Velocity components in the x, y and z-directions of the body fame
ϕ	Euler roll angle
θ	Euler pitch angle
ψ	Euler yaw angle
p, q, r	Roll-, pitch-, and yaw rate
a_1, b_1	First harmonic of blade flapping angles
ρ	Air density
σ	Solidity ratio (of a rotor blade)
a	Blade lift curve slope
λ_i	Inflow ratio
$Z_{\delta c}$	Collective pitch derivative
Z_w	Heave damping derivative
δ_a	Lateral flapping command
δ_b	Longitudinal flapping command
δ_c	Collective blade pitch command
δ_r	Yaw/Rudder command
$N_{\delta r}$	Yaw control derivative
N_r	Airframe yaw damping coefficient
N_{rfb}	Yaw stability-system damping coefficient
K_r	Yaw-stability-system coupling
K_{rfb}	Yaw-stability-system gain
V_{range}	Voltage range of analogue sensor
V_{noise}	Noise level of analogue sensor
N	Integer downsample ratio
B	Bandwidth

σ	Standard deviation
a_x, a_y, a_z	Translation of IMU from CG

Acronyms and Abbreviations

A/D	Analogue to digital converter
CAN	Controller Area Network
CCPM	Cyclic-, Collective Pitch Mixing
CG	Center of Gravity
COTS	Cheap, Off the Shelf
EMI	Electro-Magnetic Interference
ESL	Electronic System Laboratory
GPS	Global Positioning System
HALE	High Altitude, Long Endurance
IMU	Inertial Measurement Unit
Li-Po	Lithium Polymer
MEMS	Micro Electro-Mechanical Sensor
MIT	Massachusetts Institute of Technology
OBC	Onboard Computer
Op-Amp	Operational Amplifier
PC/104	Embedded computer standard
PSD	Power Spectral Density
RC	Remote Control
RUAV	Rotary-wing Unmanned Aerial Vehicle
TPP	Tip path plane
UAV	Unmanned Aerial Vehicle
UCA	User Control Application
USD	United States Dollar
ZAR	South African Rand

List of Figures

List of Tables

Chapter 1 Introduction

The concept of a flying, autonomous vehicle is not new. Research has been going on since shortly after the First World War. Thus it is no surprise that the major portion of research has been of a military nature. But as the technology matures, what benefits are there to the general public?

1.1 A Disruptive Technology

Unmanned Aerial Vehicles (UAV) have been called a *disruptive technology* or a *disruptive innovation*. The term refers to a new technology that initially performs worse than the dominating standard in most cases, but eventually overturns the existing dominant technology in the market. A disruptive technology comes to dominate the market by either providing features that existing technologies do not have, or by increasing its performance to supersede the existing technology [5].

By these measures, UAVs certainly seem like they could qualify as a disruptive technology. A few systems currently exist that can take-off, execute a mission (mostly surveillance) and land autonomously – qualities which are expected from any human pilot. But these systems can not handle the broad flying envelope that a trained pilot can. Factors like adverse weather handling, real-time decision making regarding the mission, and air-traffic interaction are still lacking.

Safety might be the disruptive feature: A UAV does not endanger a pilot's life during a mission in hostile or harsh environments. UAVs also bring new features that human pilots can not offer, the most significant being what is now known as HALE, or "high altitude, long endurance" flights. A HALE UAV is defined as a vehicle that has a range greater than 2000 km, flight altitude higher than 20,000 m and an endurance of greater than 24 hours [9]. A human pilot can arguably not achieve these figures, while still executing a critical mission.

The cost of UAVs can also be significantly less in cases where smaller tasks are required. UAV systems have long been sought for the advantage that they do not require employing a trained pilot to operate the system. But the real benefit is when a mission requires a small payload to be airborne. Typically, this would be a surveillance mission where the payload is imaging equipment. In this case, the pilot's weight is similar to that of the payload. A smaller aircraft can be used as the UAV to reduce the costs of the entire mission.

For UAVs to disrupt manned aircraft, it will need to replace all (or the majority of) manned military and civil aircraft – a situation that seems highly unlikely. In fact, the concept of disruption itself has been brought into doubt. John Dvorak wrote that *"…there is no such thing as a disruptive technology. There are inventions and new ideas…"* [8]. Nonetheless, UAV technology has made a definite impact on the aviation industry, and will continue to do so. Currently it enjoys considerable attention from the military, with the United States Department of Defence having spent USD 4 billion from 1990 to 2003. It will likely spend up to USD 10 billion by 2010 [10]. By February 2005, it had about 1500 operational UAVs in its inventory [19].

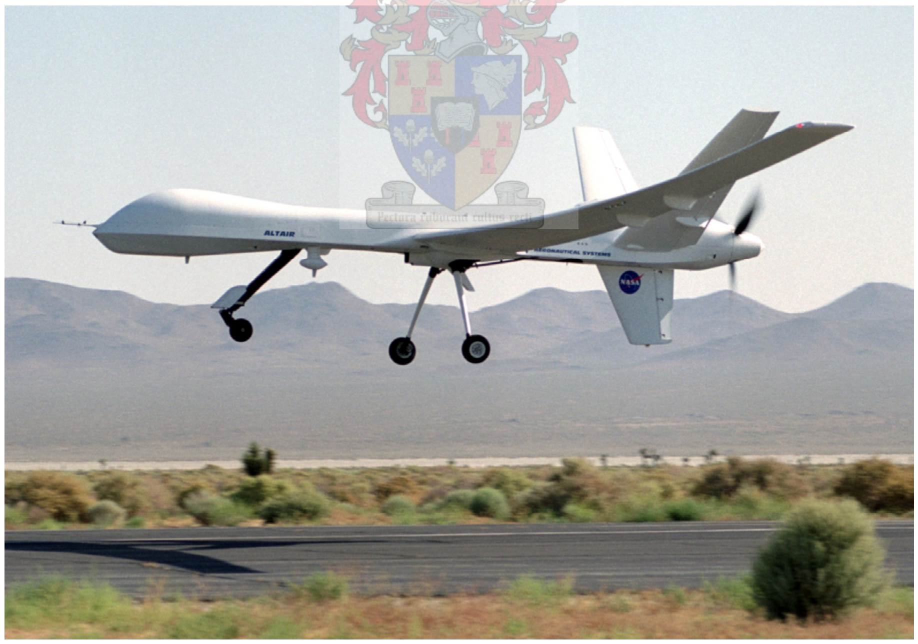


Figure 1-1: Predator B (Altair) UAV by General Atomics. The system is used for military, maritime patrol and scientific research missions[1].

Some of the military projects are finding peacetime applications, too. The Predator UAV system,

[1] NASA Photo EC03-0154-3 by Tom Tschida.

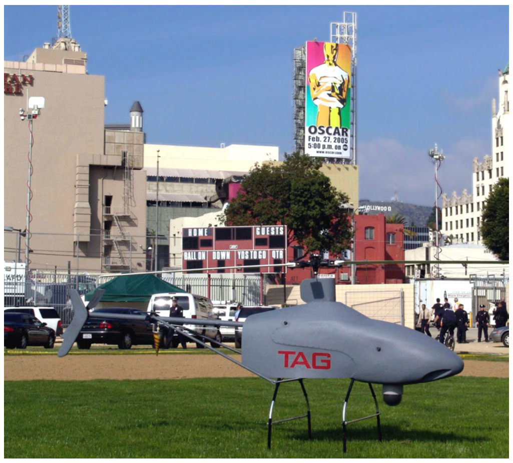
developed by General Atomics for military applications, has been used by the US Coast guard to patrol the coast in the south-western parts of Alaska [33]. NASA uses a modified version, called the Altair, for scientific missions (atmospheric sampling, etc.) at its Dryden Flight Research Centre [32].

Commercial applications are also gaining speed. NASA and Clark University established the *UAV Applications Centre* to promote the research and development of UAVs for scientific and commercial use. The centre has been involved in a number of projects providing imaging to monitor crop growth for the agriculture industry [15], [16].

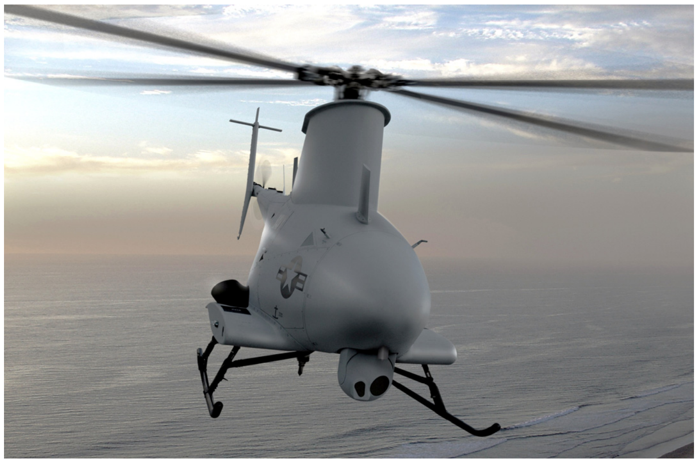
For UAVs to be embraced by industry as a tool, they will need to prove to be a cost effective solution. The commercial market does not yet fully realize the readiness and capability of this technology – many organizations do not realize that they have a requirement for UAVs. [35]. If solutions can be provided in a cost effective manner, other hurdles such as civil airspace certification and air traffic control integration will be eliminated more readily, since industry can provide a broader platform to influence government. Current proponents of certification already include organizations like the Association for Unmanned Vehicle Systems, International (AUVSI). They represent an impressive list of commercial interests, and as this list increases, so too does the influence of the organization, and the impact of research efforts.

1.2 Rotary-Wing UAV projects

Many of the UAV success stories, like the Predator, are fixed-wing aircraft. They offer structural simplicity, efficiency and better stability when compared to rotary-wing airframes. However, rotary-wing craft also offer great benefits, especially when applied as UAVs. Rotary-wing UAVs (RUAV) can take-off vertically, hover above a target site, and manoeuvre in smaller spaces. These features make them better suited for a number of short-range surveillance tasks.



TAG Series C RUAV at the Academy Awards



Northrop Grumman's Fire Scout RQ-8B

Figure 1-2: RUAV systems used for surveillance.

A RUAV designed by Tactical Aerospace Group (TAG) was used to provide aerial surveillance and assist the police force at the 77th Academy Awards [34]. Northrop Grumman developed the Fire Scout RUAV (Figure 1-2) for the US Navy to offer precision targeting support and surveillance capabilities.

Research in the RUAV field has been going on at academic institutions for more than a decade. Major contributors were the University of California at Berkeley [31], Carnegie Melon University and Massachusetts Institute of Technology (MIT) [13]. Much of the work done there made use of small to medium sized helicopters for system identification and control system design. Current research efforts are often based on their results.

Small, remote controlled (RC) helicopters are inexpensive to acquire and therefore attractive as the basis for new research in the RUAV field. They are extremely agile, but with the published system models available, control strategies have been devised to create very impressive autonomous test-beds – MIT's small X–Cell helicopter can perform autonomous acrobatic manoeuvres [14]. The vehicles are practical too: A single person can carry a normal .60 size[2] RC helicopter and transport it in a car, making flight testing easy.

[2] A "60 Size" RC Helicopter is powered by a 0.60 cubic inch (9.8 cc) methanol motor.

MIT's X–Cell 60 during an autonomous hover[3]

Berkeley's RUAV Fleet

Figure 1-3: RUAVs used for academic research.

1.3 Project Overview

After some initial investigation with regards to rotary-wing platforms [37], focused UAV research started at the engineering department of the University of Stellenbosch in 2003 with the work done by Nicol Carstens. Carstens investigated the control of a low-cost, electric RC helicopter [4] (shown in Figure 1-4). Iain Peddle designed and built the control system for a fixed-wing RC aircraft and demonstrated autonomous flight in 2004 [25].

Carstens found that a small electric helicopter does not have the capacity to comfortably lift a large enough payload, even though the avionics weighed only 0.4 kg. A side effect of the low payload capability was that the helicopter was very sensitive to shifts in the centre of gravity (CG) as sensors were added. Each component added proved to be a new challenge to keep the CG in the correct place. The result was avionics that were difficult to upgrade. All control had to be done from the ground, again because the helicopter could not carry a processor to do it onboard. This method of control introduced a severe lag in the control system that resulted in yet another challenge. Lastly, maximum flight time was approximately 4 minutes, due to limited battery life.

[3] Photo by David Dugail, 2001.

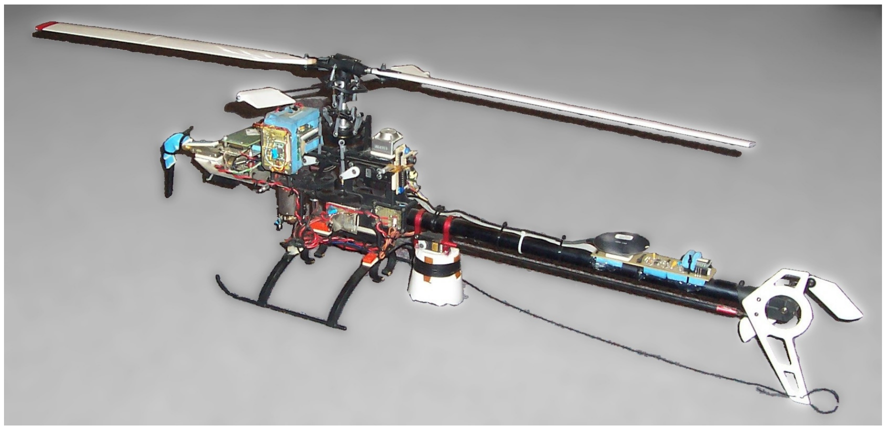


Figure 1-4: Electrically powered Voyager E with avionics.

The recommendations made after Carstens' Voyager E project became the starting point for the work done in this thesis:

- Improving the quality of sensor data by increasing digitizing resolution and speed.
- Doing all processing onboard. The processing power should be sufficient to allow a wide variety of control strategies to be used.
- Redesigning sensor mounting and the avionics package. The avionics should be modular to ease future extension and upgrading of any major component. Weight and cost should still be kept to a minimum.
- Applying the work to a platform with greatly increased payload capacity.

An RC helicopter with *glow*[4] engine was an obvious choice in terms of the payload capacity. The X−Cell of MIT has shown to be capable of acrobatic flight (in the hands of a trained RC pilot) with a 3 kg payload added [14]. This helicopter was selected early in the project. The work done by MIT and other institutions on similar sized machines was used as a guide for the avionics developed here.

1.4 Document Outline

Chapter 2 will attempt to explain the basic theory behind helicopter flight; the focus being for the

[4] RC helicopter pilots refer to glow powered motors to distinguish between glow- and spark plug engines. Glow fuel consists of Methanol (CH_3OH) as base and is usually mixed with Castor-/Synthetic Oil and Nitro Methane (CH_3NO_2) [4].

control systems engineer. The reason for this is to provide an intuitive method of reasoning when faced with the problem of helicopter control. The chapter will also discuss some of the published results relating to RC helicopter dynamics. Both linear and non-linear models will be listed.

Chapter 3 discusses the avionics system design in detail. The hardware choices were made based on the work of other, similar projects. The operating system and software required to run the various components of the system are also discussed.

Next, the hardware design is continued with the design of a generic sensor node in Chapter 4. This sensor node proved to require a significant amount of time to develop. The sensor node was designed with the focus on the type of low-cost sensors that are typically used in projects like this one.

In Chapter 5, a basic control strategy will be devised and simulated, using the models mentioned in Chapter 2. Although the primary focus of this thesis is to create an avionics system for an RC helicopter, the basic control and simulation aspect is important to verify that the avionics is adequate for the future work on the RUAV platform.

Chapter 6 verifies all design choices with measured data where possible, and Chapter 7 concludes with recommendations for improvements to the avionics and future work on the platform.

Chapter 2 Helicopter Dynamics

More than a decade of system identification in the field of mini helicopters has led to a relative abundance of published material on the subject. Some models offer a high degree of accuracy and have proven very successful as the basis for RUAV control systems. This chapter will not spend time to discuss the methods of identifying a model. It will rather give a brief introduction to helicopter dynamics, and then summarize the relevant work to this project.

2.1 Flight principles

Many texts on the topic of helicopter models seem to discard the elementary principles by which a helicopter stays airborne. Although there are excellent books on the matter of helicopter flight, they tend to develop the subject in full detail. Often, in the case of the control systems engineer, who may not have a full background in aeronautics, a brief explanation will suffice.

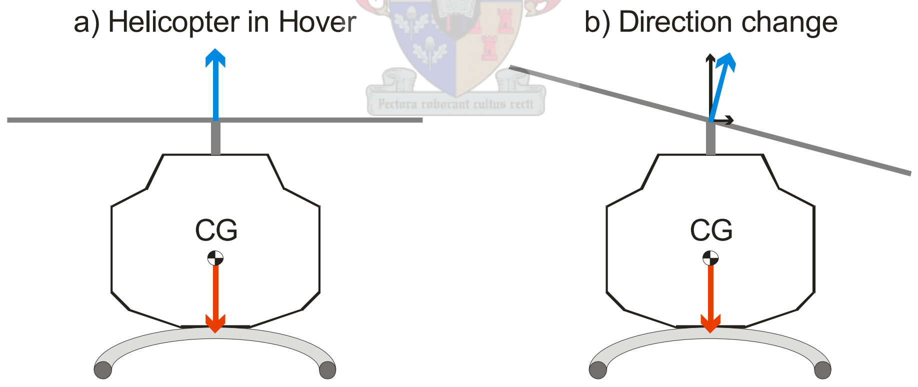


Figure 2-1: Simplified forces that act on a helicopter.

In order for a helicopter to hover, the net forces generated by the main rotor must exactly oppose the gravitational pull on centre of gravity. This situation is represented by Figure 2-1a. The main rotor can be seen as a disc producing a force normal to it that keeps the helicopter airborne. The direction of the resulting force can be changed (by some means) to accelerate the helicopter in a direction (Figure 2-1b). As an example: commanding the main rotor to tilt right will cause a resulting force to the right of the helicopter, thereby accelerating the helicopter in this direction.

If the component of the force parallel to gravity is still equal to the weight, the helicopter will maintain altitude.

The force is generated by the lift produced when the rotor blades are spun around by the helicopter's engine. A complication arises from this fact since the blades also have drag associated with them. This means that the rotor will also induce a moment on the body of the helicopter, causing it spin in the opposite direction of the main rotor blades. To counter the moment, the tail rotor is used to produce a force in the opposing direction. Unfortunately, the thrust generated by the tail rotor now acts on the body of the helicopter, causing it to accelerate along the lateral axis.

A remedy for this situation is to use the main rotor blades to produce an opposing force to that of the tail rotor. A real helicopter's main rotor will therefore always produce a force that does not only oppose gravity, but it will also produce a force to oppose the tail rotor thrust during a hover.

2.1.1 Small helicopters

Manned helicopters typically have a hinge of some sort that connects the individual main rotor blades to the hub [6]. This allows the blade to tilt upward or downward, relative to the hub, as shown in Figure 2-2a. The tip of the blade traces out a path as it rotates around the hub, called the tip path plane (TPP). The hinges allow the TPP to change its orientation and therefore change the direction of the net force produced at the hub.

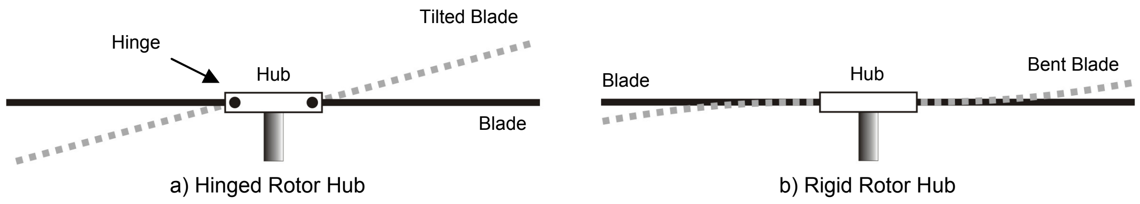


Figure 2-2: Helicopter hub configurations.

The situation is similar for small, RC helicopters, except that the hub does not have hinges. Instead, the blade itself bends as it traces out the TPP (Figure 2-2b). The angle that the tip of a blade makes relative to the hub along the lateral or longitudinal axis is called the flapping angle[5] (denoted by a_1 and b_1, respectively). The pilot is able to control these angles with the cyclic input commands (δ_a, δ_b). The collective angle of attack of the blades can be varied (by δ_c) to control the magnitude of the thrust [4]. A common method to transfer these control inputs to the main rotor is called cyclic collective pitch mixing (CCPM). Three servo actuators are spaced 120° apart and control the swashplate, the surface that forces the TPP to trace the required path. The thrust of the tail rotor can also be controlled by the pilot (δ_r).

2.1.2 Definition of symbols

The symbols that are commonly used to describe the motion of an aircraft are shown in Figure 2-3. The origin of the body axis is defined at the CG of the aircraft. Note that although the gravity vector is aligned to the vertical axis in the figure, as soon as the body changes its orientation, this will no longer be the case. Positions (x, y, z) are defined in world coordinates and place the vehicle CG at a specific location. Velocities (u, v, w) describe the motion of the CG in body coordinates. Similarly, angles (ϕ, θ, ψ) are defined in world coordinates and describe the relative alignment of the body axis to the world axis – these are then referred to as the roll, pitch and yaw of the aircraft. Angular rates (p, q, r) are defined in body coordinates.

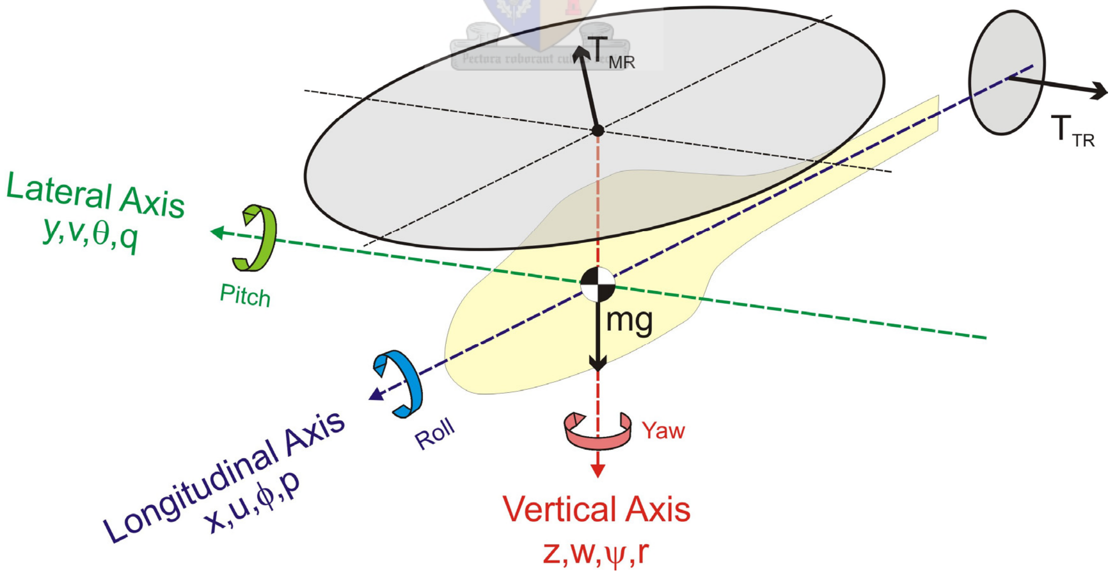


Figure 2-3: Helicopter symbols defined in the body frame.

[5] It is a bit more complicated than this: The motion of the blade can be described by mathematical functions on the lateral and longitudinal axis. The flapping angles are the first harmonics of these functions.

In the follow discussing about helicopter models and throughout this text, the above definitions will be used.

2.2 Non-Linear model

Gavrilets *et al.* [13] developed a low-order, nonlinear, dynamic model of a Miniature Aircraft X−Cell RC helicopter. The model was developed to adequately describe a miniature helicopter during aerobatic flight. Given the positive results obtained by Gavrilets with the X−Cell, it was selected as the vehicle for this project.

The exact helicopters do, however, differ slightly. The machine used by Gavrilets *et al.* [13] was the X−Cell .60 SE, equipped with a .90 size glow engine and electronic engine-speed governor. The helicopter obtained for this project was the X−Cell Fury .60 Expert [52], fitted with a .70 size engine and electronic governor (details in Appendix F). Both machines had a fast servo to control the tail rotor and share the same stiff hub. Gavrilets *et al.* [13] showed that the coupling in the hub was negligible and this led to a simpler model.

The work of Gavrilets *et al.* [12], [13], [14] will be denoted '*MIT*', while the work done here will be referred to as '*ESL*' (Electronic Systems Laboratory) whenever a concise citing is required.

2.2.1 Servo actuators

Gavrilets *et al.* [13] did some system identification of the servos used on their X−Cell to determine the bandwidth of these actuators. It must be noted again that the MIT model was used for aggressive manoeuvres, while the aim of this work is primarily to establish a basic platform for autonomous research. The bandwidth requirements are therefore lower for the ESL project. Table 2-1 compares the servos of the two X−Cells. The response and torque values are those supplied by the manufacturers. The bandwidth and damping where identified by Gavrilets *et al.* [13] for the Futaba servos (under 1 kg load for the CCPM servos, and no load for the tail rotor servo).

	Parameter	MIT	ESL
CCPM	Servo model (type)	Futaba S9402 (analogue)	JR DS8321 (digital)
	Quoted response	0.13 s / 60°	0.21 s / 60°
	Torque (4.8 V)	6.4 kg.cm	9.0 kg.cm
	Bandwidth	5.73 Hz	3.55 Hz‡
	Damping	0.5	0.6‡
Tail rotor	Servo model	Futaba S9450 (digital)	JR NES-810G (analogue)
	Quoted response	0.13 s / 60°	0.10 s / 60°
	Torque (4.8 V)	6.9 kg.cm	2.4 kg.cm
	Bandwidth	7.00 Hz	9.10 Hz‡
	Damping	0.6	0.5‡

Table 2-1: Servo comparison of X–Cell helicopters (‡ Estimated values).

The bandwidth and damping were important values for the MIT X–Cell to perform aggressive manoeuvres. For the ESL these values are less important, since the control will only focus on stabilization of the vehicle. Therefore, the bandwidth and damping was not identified, but rather estimated. The bandwidth was scaled according to the ratio between quoted responses of the servos, e.g. $5.73\text{ Hz} \times (0.13/0.21) \approx 3.55\text{ Hz}$.

The damping was based on the type of servo: Digital or Analogue. It was assumed that an analogue servo has a lower damping ratio than a digital servo, as is evident from the Futaba servos. The values were assigned accordingly.

2.2.2 Physical parameters

The helicopter's parameters where measured and compared to the MIT version. The results are shown in Table 2-2.

Parameter	MIT[6]	ESL[7]	Unit	Description
m	8.2	6.7	kg	Total Mass
I_{xx}	0.18	0.19	kg.m^2	Rolling moment of inertia
I_{yy}	0.34	0.41	kg.m^2	Pitching moment of inertia
I_{zz}	0.28	0.30	kg.m^2	Yawing moment of inertia
K_β	0.54	-	N.m/rad	Hub torsional stiffness
γ_{fb}	0.8	-	-	Stabilizer bar Lock number
$B_{\delta_{lat}}^{nom}$	4.2	-	rad/rad	Lateral cyclic to flap gain at nominal rpm
$A_{\delta_{lon}}^{nom}$	4.2	-	rad/rad	Longitudinal cyclic to flap gain at nominal rpm
K_μ	0.2	-	-	Scaling of flap response to speed variation
Ω_{nom}	167.5	-	rad.s^{-1}	Nominal main rotor (mr) speed
R_{mr}	0.775	0.775	m	Main rotor radius
c_{mr}	0.058	0.059	m	Main rotor chord length
a_{mr}	5.5	-	rad^{-1}	Main rotor blade lift curve slope
$C_{D_0}^{mr}$	0.024	-	-	Main rotor blade zero lift drag coefficient
$C_{T_{\max}}^{mr}$	0.0055	0.0053	-	Main rotor maximum thrust coefficient
$I_{\beta_{mr}}$	0.038	-	kg.m^2	Main rotor flapping inertia
R_{tr}	0.130	0.139	m	Tail rotor (tr) radius
c_{tr}	0.029	0.028	m	Tail rotor chord length
a_{tr}	5.0	-	rad^{-1}	Tail rotor blade lift curve slope
$C_{D_0}^{tr}$	0.024	-	-	Tail rotor blade zero lift drag coefficient
$C_{T_{\max}}^{tr}$	0.05	-	-	Tail rotor maximum thrust coefficient
n_{tr}	4.66	4.375	-	Gear ratio of tail rotor to main rotor
n_{es}	9.0	9.3	-	Gear ration of engine shaft to main rotor
δ_r^{trim}	0.1	-	rad	Tail rotor trim offset
S_{vf}	0.012	-	m^2	Effective vertical fin area
$C_{L_\alpha}^{vf}$	2.0	-	rad^{-1}	Vertical fin lift curve slope
ε_{vf}^{tr}	0.2	-	-	Fraction of vertical fin area exposed to t.r. induced velocity
S_{ht}	0.01	-	m^2	Horizontal fin area
$C_{L_\alpha}^{ht}$	3.0	-	rad^{-1}	Horizontal tail lift curve slope
τ_e	0.7	-	s	Engine time constant
P_e^{idle}	0.0	-	W	Engine idle power
P_e^{max}	2000	1716	W	Engine maximum power
K_{gov}	0.5	-	N.m/(rad.s^{-1})	Governor gain
T_e^{mean}	6.0	-	N.m	Engine mean torque

[6] X-Cell .60 SE with .90 engine

[7] X-Cell .60 Fury Expert with .70 engine

Parameter	MIT[6]	ESL[7]	Unit	Description
S_x^{fus}	0.1	-	m^2	Front fuselage drag
S_y^{fus}	0.22	-	m^2	Side fuselage drag
S_z^{fus}	0.15	-	m^2	Vertical fuselage drag
h_{mr}	0.235	0.350	m	Main rotor hub height above CG
l_{tr}	0.910	0.915	m	Tail rotor hub location behind CG
h_{tr}	0.080	0.095	m	Tail rotor height above CG
l_{ht}	0.710	0.735	m	Stabilizer fin location

Table 2-2: Parameter listing of X–Cell helicopters.

The moments of inertia were obtained by a torsional pendulum test [30], [13]. This procedure is difficult to set up for the X–Cell and the values obtained are rough estimates. The main rotor dimensions were almost identical, while the tail rotor parameters showed some deviation. Specifically, the tail rotor's blades and gear ratio differ. Parameters were measured or calculated where possible, but some were identified by the MIT team by more complex means. These parameters where assumed to be nearly the same for the two X–Cell helicopters.

2.2.3 Engines

The engines of the two vehicles differ in size and power characteristics. MIT used an unknown .90 size glow engine with listed maximum power P_e^{max} = 2000 W [13]. The ESL used an OS70 SZ-H .70 sized glow engine which the manufacturer specified as having an output of 2.5 hp (or 1865 W) at 16000 rpm. The engine can function in the range from 2000 rpm to 18000 rpm. These values only serve to give some indication of the relative performance and payload capacity of the two vehicles (all other parameters being equal).

The nominal blade speed of the MIT helicopter was 1600 rpm and to provide some comparison of payload capacity, the power of the ESL helicopter was roughly estimated at this speed: The gear ratio is 9.3:1 from engine to main rotor. This means that the engine rotates at 14880 rpm. The peak engine power of 1865 W was specified at 16000 rpm. Assuming no power at the idle speed of 2000 rpm, the power of was calculated at 14880 rpm with a linear curve. The approximate value of 1716 W gives a power to weight ratio of 256, compared to that of MIT's 244.

2.2.4 Hover thrust

A better way to predict the payload capacity is to calculate the maximum thrust. Carstens [4] used equation 2.1 to predict the maximum thrust (out of ground effect) for a Voyager E RC helicopter. The main rotor thrust is given by:

$$T = 4\pi R^2 \rho V_t^2 \left[A^2 r_e^2 + \tfrac{1}{3} B r_e^3 + \frac{4A\left(2A^2 - 3Br_e\right)\left(A^2 + Br_e\right)^{\frac{3}{2}}}{15B^2} \right] \qquad \textbf{2.1}$$

with

$$A = \frac{\sigma\, a_{mr}}{16}, \quad B = \frac{\sigma\, a_{mr} \theta_0}{8}, \quad \sigma = \frac{2c_{mr}}{\pi\, R_{mr}}, \quad V_t = \Omega_{mr} R_{mr} \qquad \textbf{2.2}$$

The rotor efficiency was assumed to be $r_e = 0.95$. The collective blade pitch command was set at the maximum of $\theta_0 = 10°$ (MIT control deflection limit). The results are summarized in Table 2-3.

X–Cell	ρ [kg.m^{-3}]	θ_0 [deg]	V_t [m.s^{-1}]	r_e	T_{max} [kg]	m [kg]	Thrust Ratio
ESL	1.229	10	129.9	0.95	15.7	6.7	2.34
MIT	1.279	10	125.8	0.95	15.1	8.2	1.84

Table 2-3: Thrust comparison of X–Cell helicopters.

Gavrilets *et al.* [13] cited a thrust ratio of 2.5, but as can be seen, equation 2.1 predicts a lower value. It also seems that the value that was used for V_t indicate a blade speed of 1550 rpm, rather than the claimed 1600 rpm nominal blade speed. If the parameters are adjusted slightly, specifically if the blade speed is set at 1660 rpm and the rotor efficiency $r_e = 1$, then a value of exactly 2.5 is obtained. This is not such an unreasonable adjustment for the rotor speed, since the variation of the engine speed was measured by Gavrilets as ±60 rpm during large collective variations. The rotor efficiency is modelled differently by another parameter (η_w) in the MIT model and most likely makes r_e redundant.

The value of T_{max} is used to identify C_T^{max} by the relation [13]:

$$C_T^{\max} = \frac{T_{\max}}{\rho(\Omega R)^2 \pi R} \tag{2.3}$$

With the adjustments discussed above, C_T^{max} = 0.0053 for the main rotor of the ESL helicopter.

2.3 Linear model

Mettler *et al.* [23] first presented a linear state-space model for a Yamaha R-50 helicopter. The model contained 11 states and described the helicopter during hover flight. Mettler states that small helicopters are well suited to system identification due to the dominance of the rotor in the dynamics and the absence of complex aerodynamic effects. Shim [31] designed a number of controllers based on [23] and Mettler *et al.* [22] later continued to identify a 13 state model to describe the R-50 for hover and cruise flight.

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{p} \\ \dot{q} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{a}_1 \\ \dot{b}_1 \\ \dot{w} \\ \dot{r} \\ \dot{r}_{fb} \end{bmatrix} = \left[\begin{array}{cc:cc:cc:cc:c:cc} X_u & 0 & 0 & 0 & 0 & -g & X_{a1} & 0 & 0 & 0 & 0 \\ 0 & Y_v & 0 & 0 & g & 0 & 0 & Y_{b1} & 0 & 0 & 0 \\ \hdashline L_u & L_v & 0 & 0 & 0 & 0 & L_{a1} & L_{b1} & 0 & 0 & 0 \\ M_u & M_v & 0 & 0 & 0 & 0 & M_{a1} & M_{b1} & 0 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & -1 & 0 & 0 & \frac{-1}{\tau_e} & A_{b1} & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & B_{a1} & \frac{-1}{\tau_e} & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & Z_{a1} & Z_{b1} & Z_w & Z_t & 0 \\ \hdashline 0 & N_v & N_p & 0 & 0 & 0 & 0 & 0 & N_w & N_r & N_{r_{fb}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & K_r & Z_{r_{fb}} \end{array}\right] \begin{bmatrix} u \\ v \\ p \\ q \\ \phi \\ \theta \\ a_1 \\ b_1 \\ w \\ r \\ r_{fb} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ A_{\delta_a} & A_{\delta_b} & 0 & 0 \\ B_{\delta_a} & A_{\delta_b} & 0 & 0 \\ 0 & 0 & 0 & Z_{\delta_c} \\ 0 & 0 & N_{\delta_r} & N_{\delta_c} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_a \\ \delta_b \\ \delta_r \\ \delta_c \end{bmatrix}$$

Equation 2.4: The 11 state, linear model by Mettler *et al*. [23].

Referring to equation 2.4, the 11-state vector contains the body velocities (*u, v, w*), Euler angles for roll and pitch (ϕ, θ), and angular rates (*p, q, r*). Two states (a_1, b_1) describe the flapping angles of the main rotor. Lastly, the yaw rate stability system[8] is described by a first order model (r_{fb}). The control inputs to the models are to control the TPP (δ_a, δ_b), the collective (δ_c) and the tail rotor (δ_r).

2.4 Decoupled equations

Carstens [4] proceeded to decouple the 11 state model of equation 2.4 into lower order models to describe vertical, rotational, longitudinal and lateral motion. The result will be summarized here and form the basis for control experiments in Chapter 5. All results apply to hover only.

2.4.1 Heave dynamics

A first order system model was assumed to adequate and the influence of yaw rate (r) was ignored:

$$\begin{aligned} \dot{w} &= Z_w w + Z_{\delta c}\delta_c \\ \dot{z} &= w \end{aligned} \qquad \textbf{2.5}$$

With

$$Z_w = -\frac{\rho(\Omega R)\pi R^2}{m}\frac{2a\sigma\lambda_i}{16\lambda_i + a\sigma},\ \lambda_i = \frac{\sqrt{\frac{mg}{2\rho\pi}}}{\Omega R^2} \qquad \textbf{2.6}$$

$$Z_{\delta c} = -\frac{\rho(\Omega R)^2\pi R^2}{m}\frac{8}{3}\frac{a\sigma\lambda_i}{16\lambda_i + a\sigma} \qquad \textbf{2.7}$$

The heave damping derivative for the X–Cell was calculated to be Z_w = -0.95. The value is in the same order as identified by Mettler *et al.* [22]. The collective pitch derivative is $Z_{\delta c}$ = -164.69.

2.4.2 Yaw dynamics

Influences from all but the yaw rate stability system were ignored in the simplified model:

[8] Most RC helicopters have a yaw damping system to stabilize the fast yaw dynamics and create a more stable helicopter.

$$\begin{aligned}\dot{r} &= N_r r + N_{rfb} r_{fb} + N_{\delta r}\delta_r \\ \dot{r}_{fb} &= K_r r + K_{rfb} r_{fb}\end{aligned} \qquad \textbf{2.8}$$

With the airframe yaw damping $N_r \approx -1.1$, yaw-stability damping $N_{rfb} = -22.13$, yaw control derivative $N_{\delta r} \approx -200$, yaw-stability coupling $K_r = 3.15$ and yaw-stability gain $K_{rfb} = -9.50$.

2.4.3 Pitch and Roll

These dynamics require two steps: First, describe the flapping of the rotor blades, and second, describe the pitch and roll rate of the helicopter in response to the flapping of the blades [4]. The model of Mettler *et al.* [23] is simplified by ignoring cross-coupling derivatives and realizing that the stability derivatives L_w and M_w are zero during hover. The simplified equations become:

$$\begin{aligned}\dot{b}_1 &= -\tfrac{1}{\tau_e} b_1 - p + B_{\delta a}\delta_a \\ \dot{a}_1 &= -\tfrac{1}{\tau_e} a_1 - q + A_{\delta b}\delta_b \\ \dot{p} &= L_v v + L_{b1} b_1 \\ \dot{q} &= M_u + M_{a1} a_1\end{aligned} \qquad \textbf{2.9}$$

These equations lead to second order transfer functions:

$$\begin{aligned}\frac{q}{\delta_b} &= \frac{A_{\delta b}\omega_{nq}^2}{s^2 + s/\tau_e + \omega_{nq}^2} \\ \frac{p}{\delta_a} &= \frac{B_{\delta a}\omega_{np}^2}{s^2 + s/\tau_e + \omega_{np}^2}\end{aligned} \qquad \textbf{2.10}$$

The natural frequencies and effective time constants where identified as $\omega_{nq} = 14.6$ rad.s^{-1}, $\omega_{np} = 20.1$ rad.s^{-1} and $\tau_e = 0.13$ s [4], [13].

2.4.4 Horizontal velocity dynamics

Wind gusts, constant winds and tail rotor lateral acceleration were ignored for the horizontal velocity dynamics during hover. A very simple model was finally used by Carstens [4]:

$$\begin{aligned}\dot{u} &= -g\theta \\ \dot{v} &= g\phi\end{aligned} \qquad \textbf{2.11}$$

2.5 Summary

Common conventions and definitions were presented to be used throughout the rest of this text. The new vehicle that will be used as the platform was compared to an existing machine and the parameters identified where possible. Some models were referenced as the basis for further work to control the vehicle.

Chapter 3 Architecture

The hardware and software that comprise the avionics are introduced. Key design choices are discussed, capabilities listed and future applications mentioned[9].

3.1 Background

The RUAV research projects of the past decade have mostly followed a specific path in terms of avionics. They usually had an Intel compatible, x86 processor at the heart, used a commercial inertial measurement unit (IMU) and a differential global positioning system (GPS). The operating systems varied, but were mostly a real-time system. Table 3-1 lists these traits for projects that are comparable to the airframe of this project.

Institute	Vehicle	Sensors	CPU	OS
Berkeley [31]	Kyosho Concept 60SR II	Boeing DQI-NP NovAtel RT-2 GPS	Cyrix 233MHz	QNX
CSIRO [29]	JR Ergo .60	Crossbow DMU-VG IMU Vision based Navigation (unknown) GPS Ultrasonic Sensor	AMD K6-2 300MHz	LynxOS
Georgia Tech [18]	X–Cell .60	Watson E-303 AHRS NovAtel RT-2 GPS Ultrasonic Sensor	AMD 133MHz	Linux
MIT [14]	X–Cell .60	Inertial Sciences ISIS IMU CMC Superstar GPS Barometric Altimeter	Geode 266MHz	QNX

Table 3-1: Avionics comparison of similar projects

The design goal of this project was to develop avionics that were extendable and could be used with a wide variety of aircraft, not only RUAVs. The main constraints were cost and size: Inertial

[9] The architecture was jointly developed by Jacob Venter and the author under the guidance of Dr. Thomas Jones. Credit will be given where it is due.

sensors had to be of the Micro Electro-Mechanical Sensor (MEMS) kind, since off the shelf IMU systems cost from 5000 USD upwards. The onboard computer (OBC) had to be a PC/104 [24] based system. The choice in operating system was left open, but low latency was the primary concern, while availability and local support should also be considered.

Other criteria were that a standalone GPS be used, again because the cost of differential GPS systems started at 2000 USD (NovAtel OEM4-G2L™, February 2005). The choice in supporting micro-controllers and communication standards was left open, subject to price and local support.

The avionics systems of Table 3-1 mostly have a connection topology similar to that of Figure 3-1 (basically a star topology in network terms). The CPU sits at the centre of the sensors and actuators. The connection with external devices uses the RS-232 standard. When more devices are needed, an expander board is added to the CPU bus to provide more RS-232 ports. This results in a cumbersome OBC.

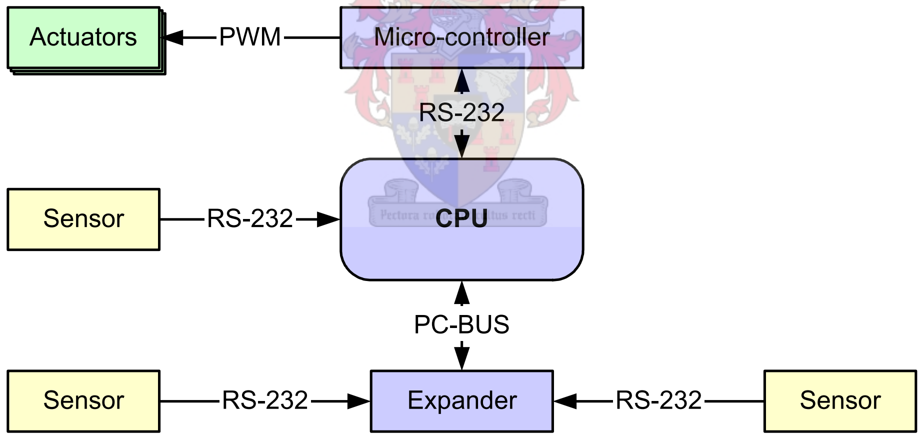


Figure 3-1: Common topology of avionics listed in Table 3-1.

The RS-232 standard makes sense as a communication standard for these systems, since most of their sensors offer a RS-232 connection. MEMS devices are analogue sensors and require some form of interface to be added; therefore the choice of communication interface is open.

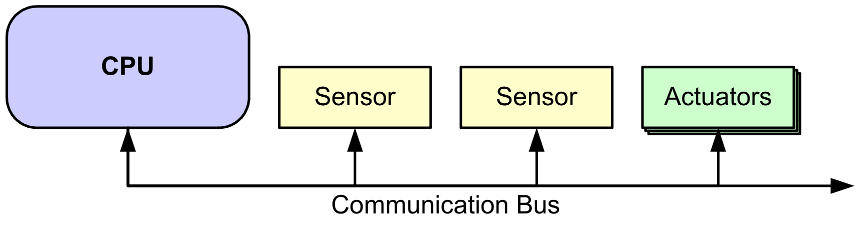


Figure 3-2: Bus architecture for an OBC.

Figure 3-2 shows an alternate configuration for the avionics. All devices are connected to a single bus. This design allows easy extension and is compact, since no large expander boards are required. The drawback is that all devices need to support the specific protocol and communication standard, meaning that COTS hardware may require additional circuitry.

3.2 Hardware

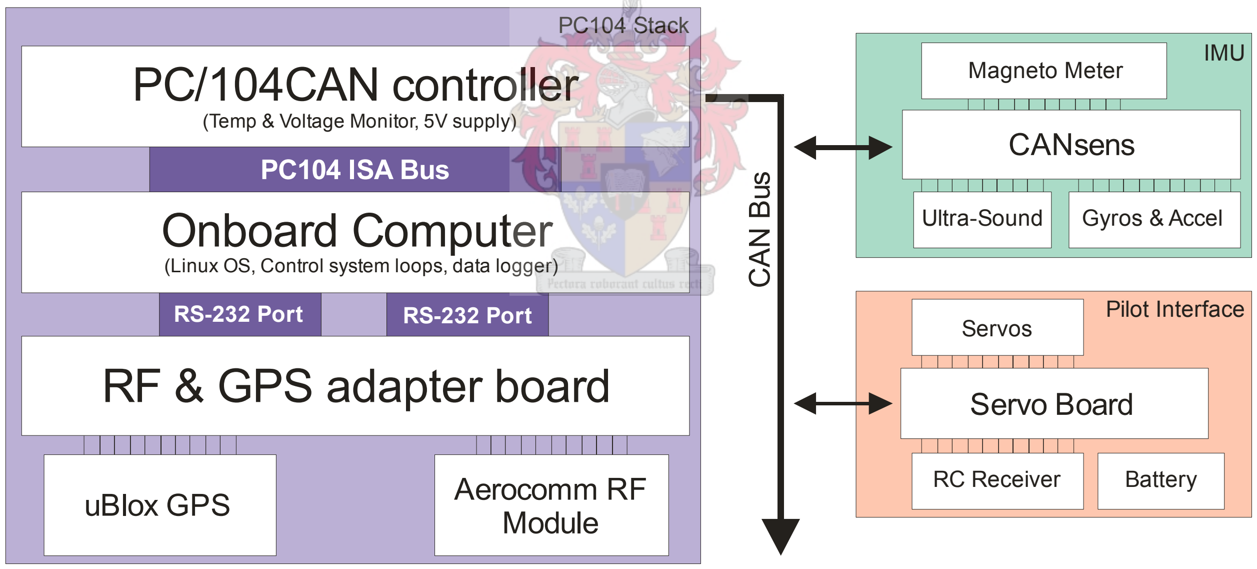


Figure 3-3: Avionics Layout.

Figure 3-3 shows the layout of the avionics. The OBC consists of a PC/104 board with a communication bridge to the CAN bus. The IMU and actuator/pilot interface are connected to this bus. The design choices that led to this layout will be discussed in the following paragraphs. A system-wide sample rate of 50 Hz was imposed, to be compatible with off-the-shelf remote control equipment. This design criterion provided the basis for most calculations when latency or bandwidth was considered.

3.2.1 CAN System

The Controller Area Network (CAN) is a serial communications protocol which efficiently supports distributed real-time control with a very high level of integrity and bit rates of up to 1 Mbit/s [44]. CAN is a packet based standard, and provides error detection and automatic retransmission. The University of Stellenbosch has done research for a number of years on satellite systems and developed a communication protocol based on CAN [45]. The Electronic Engineering department has the hardware tools and expertise to test these CAN systems and it was decided to adopt this bus architecture for the avionics.

PC/104 to CAN interface boards was considered, but these were all too expensive. A PC/104 expansion board was developed by Venter [38] to provide the bridge between the PC/104 computer and the CAN bus (hence referred to as PC/104CAN controller). The board also contains the power system, saving a significant amount of space. It can monitor the input voltage and the various voltages of the DC/DC converters that power the rest of the system. A micro-processor is at the heart of the board and provides a great deal of flexibility – something that an off-the-shelf board would not have done. The device plugs into the ISA bus of the PC/104 OBC.

3.2.2 Avionics Battery

A Lithium Polymer (Li-Po) battery was chosen as the avionics battery. These batteries have become popular with electric RC hobbyists and are widely available. The configuration used was a Kokam 11.1 V, 2000 mAh pack [48] weighing 175 grams, with dimensions 25 × 80 × 45 mm. The maximum discharge rate is rated at 30 A. The battery was chosen for the outstanding energy density offered. In Table 3-2, the Li-Po battery is compared with two batteries of similar capacity, but different chemistries. It is clear the Li-Po batter is far superior.

Battery Pack	Kokam 3 Cell Li-Po	Electrifly 10 Cell NiMH	Electrifly 10 Cell NiCd
Capacity [mAh]	2000	2000	1900
Nominal Voltage [V]	11.1	12.0	12.0
Dimensions [mm]	80×25×45	225×38×24	222×24×47
Weight [g]	175	475	570
Energy Density[10] [Wh.kg^{-1}]	126.9	50.5	40.0

Table 3-2: Energy density comparison for different battery chemistries.

3.2.3 Onboard Computer

There are a myriad of PC/104 computers on the market. Prices and features vary greatly. Although Geode based systems (as used by MIT [14]) offer less power consumption, they also offer less floating point performance, while prices tend to be the same as similar Intel machines. A solution was found in the Kontron MOPSlcd7 [49]. This PC/104+ computer offers a 300 MHz Intel Pentium III processor, upgradeable 64 Mb RAM, upgradeable 32 Mb solid state IDE drive, PCI bus, ISA bus, 2 RS-232 ports, 1 parallel port, Ethernet, 2 USB ports, keyboard, mouse and VGA connectors. The MOPSlcd7 was competitively priced when these extensive features are considered. Since it is a standard Intel architecture machine, if also offers superior software compatibility, while the Pentium III processor provides ample performance.

3.2.4 GPS and Ground-Link

The GPS used was a uBlox RCB-LJ providing updates at 4Hz with 2.5 m maximum accuracy [60]. This GPS has been supported for a while in the department and was the obvious choice. It is also the only known GPS in its price range that offers 4 Hz update rates. Similarly priced devices typically offer 1 Hz update rates.

The communication link to the ground was implemented with another device previously used in the department [25]: The AeroComm AC4486 RF Modem [39]. This is an 868 MHz device that can directly replace a RS-232 serial cable. The manufacturer specifies a maximum reliable

[10] The Energy Density of a battery can either be Wh.m^{-3} or Wh.kg^{-1}. The latter is used here since weight is more important for aircraft.

communication rate of 9600 baud [39].

Both these devices required some interface logic and voltage regulators. They where placed on a PC/104-sized PCB that sits on the PC/104 stack with the OBC and PC/104CAN controller boards.

3.2.5 Sensor Node and IMU

A sensor node was required to interface analogue sensors to the CAN bus. A small device was designed to provide up to 16 bit resolution for 8 analogue channels. This sensor node, called CANsens, was combined with MEMS sensors to create the IMU of the avionics. This sensor node also provided the interface to an ultrasonic sensor to measure height information. Full design details are discussed in Chapter 4.

3.2.6 Pilot Interface[11]

A backup pilot is essential for experiments in autonomous flight, to ensure the safety of equipment and people involved. The pilot must be able to take full control of the vehicle when the control system malfunctions. System identification and initial setup also requires that a pilot must be able to fly the vehicle while the avionics is used to gather data [13].

Roberts *et al* [28] used 5 solid state relays to ensure that the pilot can take control of the vehicle, even when the avionics loses power. For this project, however, more flexibility was required for the servo control. Specifically, the pilot interface should allow more complex (experimental) vehicles to be controlled.

Venter [38] designed a microprocessor board that could be used to read 8 channels from the pilot and control up to 16 servo actuators. The microprocessor can do complex mixing of the pilot commands, or receive servo commands via the CAN bus when the avionics has control of the vehicle. The pilot decides whether the vehicle is safe, and can override the avionics at

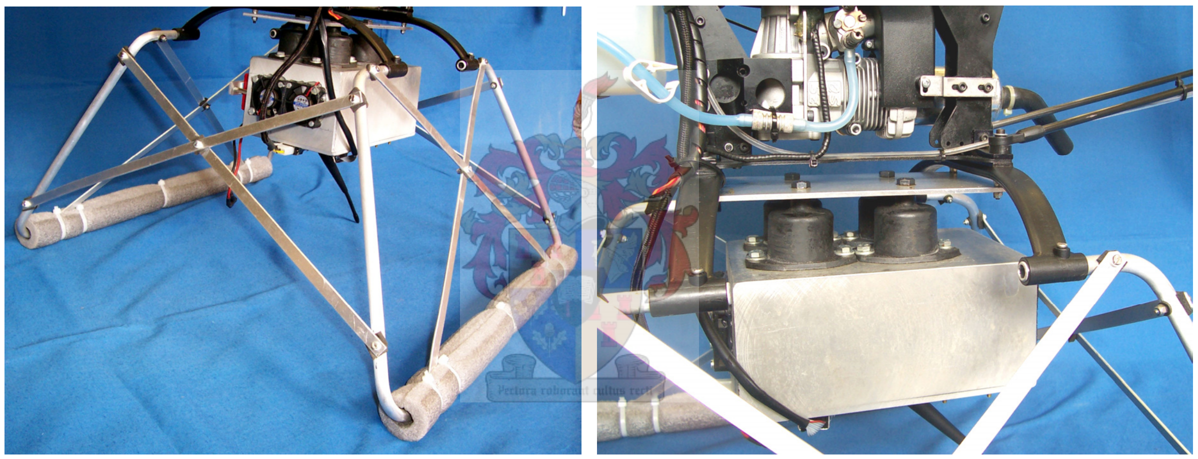
[11] Developed by Jacob Venter.

any time.

This approach could potentially be less reliable than the solid state relay implementation, and testing is essential after each software update. However, no problems were encountered during any flight test.

3.2.7 Enclosure and Mounting

Kahn [18] comprehensively documented his work on the avionics for an X–Cell. Similar to the ESL project, much of his avionics mountings were hand-made. He did extensive testing to ensure that vibration isolation was adequate.



a) Extended skids with foam. Note aluminium box for avionics b) Vibration mounts with suspended avionics

Figure 3-4: Skids and vibration mounting of X–Cell.

Kahn also noted that dc-to-dc converters caused interference (EMI) with the various sub-systems of the avionics, and with the servos of the helicopter. Since the ESL helicopter also uses dc-to-dc converters as part of the power systems, a metal enclosure was chosen to avoid this situation. Interference of the avionics components could be dealt with if a problem was encountered, but if the servos failed during a flight, the helicopter would suffer severe damage. The servos, RC receiver and pilot interface were therefore placed on the nose of the helicopter, while the metal enclosure with the rest of the avionics was mounted under the helicopter. The ultrasonic sensor was also fixed to the enclosure on the nose. New skids where made to allow avionics to hang comfortably under the helicopter. Figure 3-4a shows this layout. Foam was wrapped around the bottom part of the skids to absorb some impact during landings.

Vibration mountings were selected to primarily damp out the main rotor vibrations. The nominal blade speed is 1600 rpm, or 26.6 Hz. An off-the-self solution was found that offered around 20 dB or more attenuation at 25+ Hz [47]. The avionics was suspended from four of these dampers, as shown in Figure 3-4b.

3.2.8 Weight and cost summary

Component	Weight [g]	Price [ZAR]
uBlox GPS	16.3	R 430
GPS Antenna & Cable	102.0	R 165
AeroComm RF Modem	12.9	R 650
AeroComm Antenna	27.4	R 85
GPS & RF motherboard	40.3	R 100
PC/104 CAN Board	98.3	R 1,100
Kontron MOPSlcd7	181.8	R 5,120
Li-Po Battery	174.9	R 550
CANsens IMU	51.0	R 3,392
Ultrasonic Sensor	19.2	R 363
CAN Servoboard	45.0	R 750
Electronics Subtotal	**769.1**	**R 12,705**
Enclosure	540.0	R 110
Extended Skids	315.0	R 210
Vibration Dampers	225.0	R 220
Misc. cables, etc	78.0	R 50
Total	**1,927.1**	**R 13,295**[12]

Table 3-3: Avionics weight and cost summary.

The X−Cell used by MIT has a payload, including avionics, suspension system and modified landing gear of 3.1 kg [12], compared to the 1.9 kg of the avionics for this project (Table 3-3).

[12] 2,015 USD: Exchange rate 6.60 ZAR to 1.00 USD

Costs were kept comparatively low when it is considered that the total price of the avionics is equivalent to a single differential GPS system.

3.3 Software

3.3.1 CAN Protocol

The CAN specification [44] allows two types of identifier to implement the user's protocol: A standard identifier, which is 11 bits wide, and an extended identifier that is 29 bits wide. All packets can have 0 to 8 bytes of data. Data transmission rates can be up to 1000 kBaud. CAN implements a number of data integrity safeguards, including guarantee of latency times, error detection and error signalling, and automatic retransmission of corrupted messages. This integrity comes at the cost of overhead and can be around 100% of the amount of data transmitted [44].

The protocol implanted here uses the extended, 29 bit identifier and has the structure shown in Figure 3-5.

CAN Field	Field Name		No. Bits
CAN Identifier	Field ID	Main Type ID	4
	Field ID	Request/Reply	1
	Field ID	Subtype	8
	Source Address		8
	Destination Address		8
Data Bytes	Data Byte 0		8
	Data Byte 1		8
	Data Byte 2		8
	Data Byte 3		8
	Data Byte 4		8
	Data Byte 5		8
	Data Byte 6		8
	Data Byte 7		8

Figure 3-5: CAN packet structure using extended (29 bit) identifier.

The field functions are:

- Main Type ID: The node's function. This is a number assigned to indicate functions like *sensor*, *actuator* or *master*.
- Request/Reply: The bit is set when data is requested; unset when data is returned.
- Subtype: This indicates a specific node type. The sensor type might have a subtype of *IMU*, *Altitude*, etc.
- Source/Destination Address: A unique address for each physical node on the CAN network. The value of 255 (or 0xFF) is reserved and means '*everyone*'.

3.3.2 Operating System

Real-time operating systems are the norm in the UAV field, since they guarantee system timing irrespective of the hardware platform. Most of these systems, like *QNX* and *LynxOS,* are not free, although free equivalents like *RTLinux* do exist. However, when custom hardware is involved, the above systems require that a specific driver be created.

Having considered the time and resources involved in implementing a real-time operating system, it was decided to employ a simpler architecture: The OBC and PC/104CAN controller is considered a single device. The micro-controller of the PC/104CAN controller is responsible for the timing of the control system, while the OBC executes all the control algorithms. The OBC is aware of the timing constraint and is required to complete the calculations before the next time step. This process will be described in more detail in 3.3.3.

The main advantage of this scheme is that virtually any operating system can now be used, without any real-time extensions. *Linux* was the obvious choice for this project. The operating system provides the flexibility to modify it on any level, has wide support for networking, file systems and hardware. The core of the operating system, referred to as the *kernel*, can be configured to have any (or only) features that the user require. For an embedded system where the hardware does not change, this leads to an efficient operating system that is small and fast. Linux systems also come with a C compiler and the entire software package can be downloaded free of charge.

The creation of such an embedded Linux requires a few steps to configure and compile the system. Details are described in Appendix D and the resulting system used for the OBC has these features:

- The newest Linux kernel can be used, ensuring that the system stays current.
- The kernel has been stripped and compiles to 850 kB when compressed.
- The entire operating system occupies about 1.2 MB of the system's disk space.
- The file system used (*minix*) is simple and the system can repair damaged files automatically.
- TCP/IP networking is supported.
- Tools are included to compress files and transfer them via Ethernet.
- The system can boot via the RS-232 port, or with a normal keyboard and screen when they are connected.

3.3.3 Control Application

While the operating system is left to manage the normal functions like disk I/O, communication with the connected peripherals and network devices, it is left to the user control application (UCA) to communicate with the PC/104CAN controller and adhere to the timing imposed by it. The interface between the controller and the OBC is designed to be very similar to the I/O mapped interface of the standard PC parallel port. A control register and data register are used to communicate and pass information between the two devices. This scheme is also known as polled I/O and requires a careful software design to avoid a situation where the two devices loose synchronization.

The UCA interacts with the PC/104CAN controller, shown in Figure 3-6, in the following manner:

1. The global time synchronization packet is issued on the CAN bus by the PC/104CAN controller. This command instructs all sensor nodes to send the most recent measurements, and all actuator nodes to update to their most recently received actuation levels.
2. The nodes react and send the relevant data on the CAN bus to the PC/104CAN

controller. Assuming 5 packets at 1 Mbit/s, with each packet 13 bytes in size (4 for the identifier, 1 for data size and 8 for data) and 100% overhead, this will take (5×2×13×8)/1000000 ≈ 1ms. This puts the true data rate at around 320 kbit/s, compared to a maximum of 115 kbit/s for standard PC RS-232 ports.

3. A time-out occurs and no more CAN packets are accepted. The OBC is notified that new data is ready. Meanwhile, the OBC waits for the PC/104CAN controller to respond.

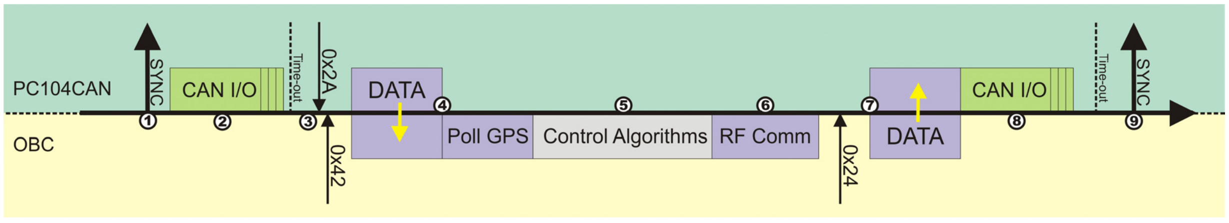


Figure 3-6: Onboard computer timing scheme.

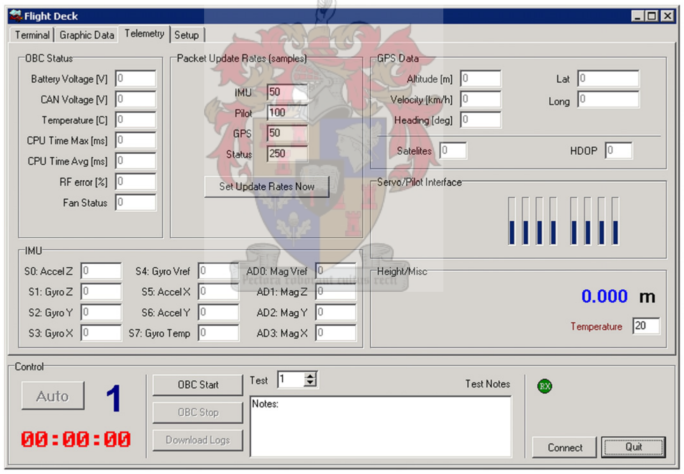
4. When new data is ready, communication is initiated and the CAN packets are copied directly to the OBC. The CAN packets are filtered and data conversion performed. The GPS is polled via the RS-232 port and data retrieved, if ready.
5. With all new sensor data ready, the control algorithms are ready to execute.
6. Any data that should be sent via the RF Link is stored in a buffer and sent in the background. The receive buffer is checked for any commands received from the RF Link and executed.
7. The OBC notifies the PC/104CAN controller that it is finished and ready to send new data onto the CAN bus, including the new actuator data calculated by the control algorithms. The OBC has the ability to place any CAN packet onto the bus. These may include commands to any or all nodes.
8. Data is placed on the CAN bus and sent to the relevant nodes. The PC/104CAN controller imposes a time-out to ensure that there is no bus activity in anticipation for the time synchronization command.
9. The global time synchronization packet is issued again and the new actuator data is applied to the actuators. The cycle repeats.

The structure of the program should remain the same for most control applications.

Modification to the control algorithms is all that would be required. The UCA also stores information about the state of the system, including some timing values. Some of these measurements and system latency are discussed in paragraph 6.2. The control software also stores all sensor data, actuator commands and system status messages to separate files for post processing.

3.3.4 Ground Station

The RF Link provides a means for the avionics to be controlled from the ground. Commands can be uploaded and status data can be returned. A simple protocol, used by Peddle [25], was extended to allow more telemetry types to be sent (see Appendix E-1). A graphic interface for Microsoft Windows was created to display the telemetry received from the avionics. Commands can be sent and telemetry update rates can be adjusted.



OBC Status

Battery Voltage [V]	0
CAN Voltage [V]	0
Temperature [C]	0
CPU Time Max [ms]	0
CPU Time Avg [ms]	0
RF error [%]	0
Fan Status	0

Packet Update Rates (samples)

IMU	50
Pilot	100
GPS	50
Status	250

GPS Data

Altitude [m]	0	Lat	0
Velocity [km/h]	0	Long	0
Heading [deg]	0		
Satelites	0	HDOP	0

Servo/Pilot Interface

IMU

S0: Accel Z	0	S4: Gyro Vref	0	AD0: Mag Vref	0
S1: Gyro Z	0	S5: Accel X	0	AD1: Mag Z	0
S2: Gyro Y	0	S6: Accel Y	0	AD2: Mag Y	0
S3: Gyro X	0	S7: Gyro Temp	0	AD3: Mag X	0

Height/Misc

0.000 m

Temperature 20

Control

1

00:00:00

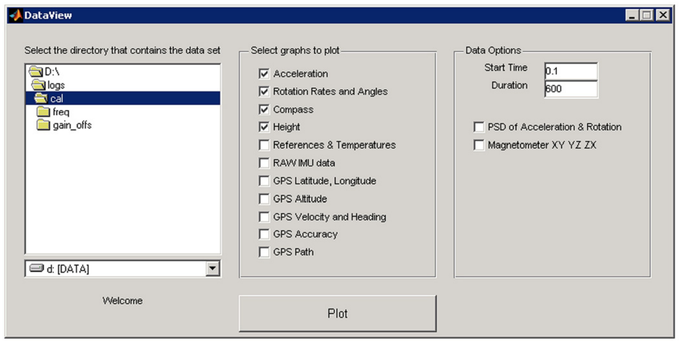
Test 1

Test Notes

Notes:

Figure 3-7: Ground Station software for Windows.

The flight data stored onboard the avionics is transferred via Ethernet cable to the users PC. A MATLAB script was created to view this data. The script presents a simple graphical interface for the user to select which sets of data to view, shown in Figure 3-8.


DataView

Select the directory that contains the data set
D:\
logs
cal
freq
gain_offs
d: [DATA]
Welcome

Select graphs to plot
- [✓] Acceleration
- [✓] Rotation Rates and Angles
- [✓] Compass
- [✓] Height
- [ ] References & Temperatures
- [ ] RAW IMU data
- [ ] GPS Latitude, Longitude
- [ ] GPS Altitude
- [ ] GPS Velocity and Heading
- [ ] GPS Accuracy
- [ ] GPS Path

Data Options
Start Time 0.1
Duration 600
- [ ] PSD of Acceleration & Rotation
- [ ] Magnetometer XY YZ ZX

Plot

Figure 3-8: MATLAB interface to view recorded flight data.

3.4 Recommendations

A low-cost avionics system was developed to meet most of the design criteria: In terms of hardware it is much cheaper than other projects mentioned, even more compact and uses newer technology. What's more, a foundation was laid to support this architecture and further develop it in the future. These improvements could be considered:

- The avionics was mounted just in front of the main rotor and the pilot noted that the helicopter handling could be better if it was moved backward somewhat. The vibration dampers and avionics mounting worked well overall, but the pilot-interface box initially caused severe vibration. The box mounting was adjusted, but it is recommended that this entire mounting scheme be revised and possibly removed from the nose to another location. In later work done by the MIT group, the avionics was mounted on the sides of the helicopter, rather than below it.
- The CAN bus is the most promising technology implemented here. The full potential of the system has not yet been exploited: It is possible to upgrade firmware via the CAN bus. This will save time and effort since it will not be necessary to dismantle the avionics to upgrade any or all of the micro-controllers connected to the CAN bus. The avionics is capable of receiving the firmware via the RF Link and then upload it to the relevant CAN node. Digital filters or actuator behaviour could be modified in this manner.
- The avionics uses a great deal of software. This makes the system flexible, but also

difficult to manage. It would be wise to attempt code sharing to eliminate some errors, especially where the CAN bus protocol is used. Already, there is shared source-code that attempts to define the protocol in a single place, but this idea should be implemented more widely to avoid time consuming debugging sessions. Data file formats, RF protocols and sensor calibration should all be defined in a single place.

- The operating system scheme proved effective, but newer version of the Linux kernel (version 2.6.12) can provide much improved real-time scheduling. This system should be investigated further. The benefit of a true real-time operating system should also not be overlooked. It could provide improved platform for academic research, but could also perform better if the sample rates are significantly increased.

Data acquisition is a key part of digital control systems. This chapter details the development of a generic sensor interface. The result is a flexible piece of hardware, called CANsens, which can be used for a wide variety of applications.

4.1 Sensor Requirements

Low cost flight control systems use numerous sensors to accomplish the task of autonomous flight. These commonly include accelerometers, rate gyroscopes, GPS hardware, pressure sensors, magnetometers and ultrasonic sensors (Figure 4-1). Vision based sensors are also common ([28], [26], [2], [1]), but are considered to be specialized sensors that require great amounts of bandwidth and processing power.

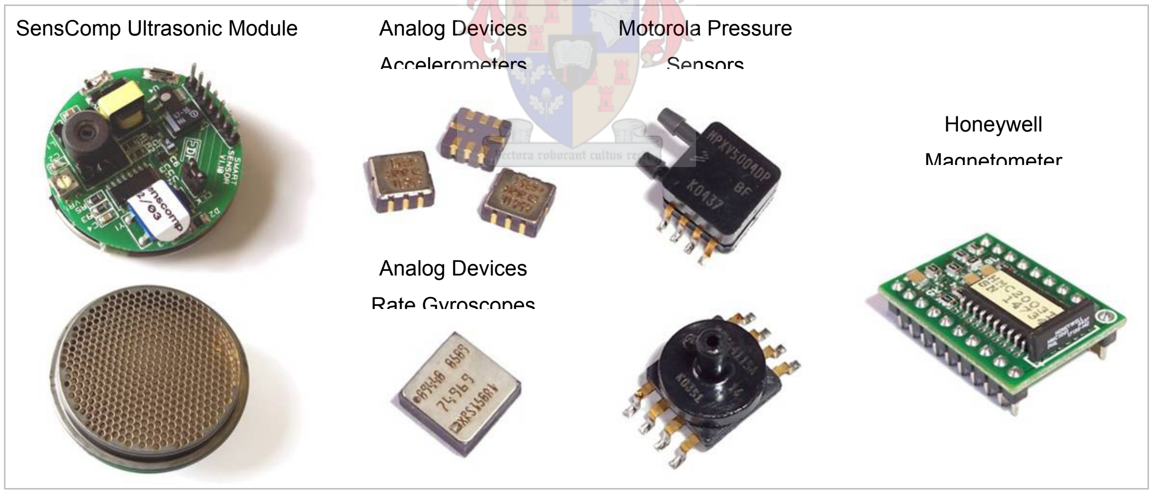

SensComp Ultrasonic Module
Analog Devices Accelerometers
Motorola Pressure Sensors
Honeywell Magnetometer
Analog Devices Rate Gyroscopes

Figure 4-1: Common analogue sensors used in low cost flight control systems.

Except for GPS hardware, these sensors all have analogue outputs that need to be digitized. They often require mounting at specific locations on the vehicle to perform optimally. Therefore, an analogue to digital (A/D) converter is required to digitize sensors at a fixed rate. It should be small, to ease mounting, and easily configured to work with any number of sensors.

Table 4-1 summarizes sensor parameters to establish digitizer requirements. The A/D resolution was calculated by finding the minimum number of bits required to scale the noise to less than 1 bit, using equation 4.1:

$$bits = \text{ceil}\left[\log_2\left(\frac{V_{range}}{V_{noise}}\right)\right] \qquad \textbf{4.1}$$

where *bits* denote the A/D resolution required, V_{range} the range of the sensor voltage, V_{noise} the (root mean square) noise associated with the sensor and *ceil* is a function to round the argument up to the nearest integer.

All the sensors of Table 4-1 have outputs of between 0V and 5V. Therefore, to simplify the analogue interface, the A/D resolution was spread across this range. The alternative is to scale the sensor output to make use of the full dynamic range of the A/D converter (assumed to be 5V). This would require more complex analogue circuitry, but allow a lower resolution A/D converter in some cases.

Sensor Type	Model Number	Maximum Bandwidth [Hz]	V_{min} [V]	V_{max} [V]	V_{noise} [mV]	A/D Resolution
Accelerometer	ADXL103	2500	0.80	4.20	1.00	13 (12)
Accelerometer	ADXL202E	6000	2.19	2.81	0.62	13 (10)
Rate Gyroscope	ADXRS401	400	1.38	3.63	3.00	11 (10)
Rate Gyroscope	ADXRS150	400	0.63	4.38	3.95	11 (10)
Absolute Pressure	MPX4115A	1000	0.20	4.79	1.00	13 (13)
Differential Pressure	MPXV5004G	1000	1.00	4.90	1.00	13 (12)
Magnetometer	HMC2003	1000	0.50	4.50	0.40	14 (14)

Table 4-1: Sensor characteristics. Noise was calculated at 100Hz bandwidth where required. A/D resolution is unscaled over 5V. Brackets indicate full dynamic resolution.

From Table 4-1 it is clear that a 14 bit A/D device would meet most requirements, but A/D devices typically introduce quantization noise, so care should be taken when selecting a device so close to the minimum specification. A 16 bit device may be a better choice.

Given that this project's aim was to develop a standard hardware platform with a PC/104 [24] computer at its core, a few PC/104 data acquisition systems were considered (see Figure 4-2).

Typical products offer 8 to 16 analogue input channels, 100 000 samples per second and 16 bit resolution. Prices range from 500 USD upwards. The main limitation of such a system is that the A/D hardware is fixed to the PC/104 stack. The sensors would require long wires to carry the signals to the PC/104 stack, possibly adding additional signal noise.

The alternative is to have local, *smart* sensors that digitize the signal at the source and then transfer the data digitally, rather than via long wires to a central A/D device.

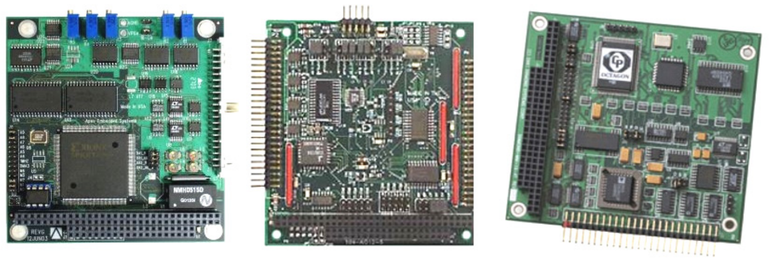


Figure 4-2: PC/104 Data acquisition devices. Left to right: Apex STX104 [42], Kontron ADIO128 [49], Octagon DMM-16-AT [53].

With the requirements established and having considered the cost of PC/104 acquisition systems, it was decided to develop custom A/D hardware.

4.2 Noise, Filters and Sampling

Low-cost sensors are typically noisy. So too are inexpensive data acquisition hardware. Add a vibrating propulsion source such as the methanol engines common to small, autonomous craft, and it becomes clear that sensor signal conditioning is essential. It would be difficult to design a filter system that will be sufficient for any application. However, the design goal of CANsens is primarily to be a generic data acquisition system for small UAVs. This places a constraint on the sample rate and bandwidth that is required. A brief discussion of the filter design for CANsens follows.

4.2.1 Sample Rate and Anti-Alias Filters

Anti-alias filters are required to attenuate noise above the Nyquist frequency [11]. They should be selected to have a minimum effect on the phase of the system bandwidth. Therefore, the choice of anti-alias filter is closely linked to bandwidth and sample rate.

An active, second order Butterworth filter is a good choice as an anti-alias filter, since it provides the flattest passband response [17] and low component count. As will be shown in 4.2.3, the phase characteristics are not important. The -3 dB cut-off frequency of a Butterworth filter is given by equation 4.2, with *R* and *C* the values of the resistors and capacitors, respectively. Refer to Appendix C-2 for the filter circuit.

$$fc = \frac{1}{4\pi RC} \qquad \textbf{4.2}$$

4.2.2 Oversampling

Noise is introduced by quantization errors of the A/D device [21] and by the sensors themselves. Averaging can be used to reduce the noise of A/D systems, with the relation between the standard deviation of the noise, σ and the amount of samples averaged, n given by equation 4.3. [20]

$$\sigma_{avg} = \sigma\sqrt{\frac{1}{n}} \qquad \textbf{4.3}$$

Averaging requires that many samples be taken in fast succession; otherwise it will introduce some phase lag to the results. A more predictable method is decimation, where the amount of oversampling and sample rates form part of the design [21]. The analogue data is sampled at a high rate (F_x) and then, in most simplest case, every N^{th} sample is taken as the output. The output sample rate is then $F_y = F_x/N$. This process introduces more aliasing, apart from the possible aliasing that the sampling process of F_x has. To prevent the secondary aliasing, another filter is added to limit the bandwidth to below $F_y/2$. The situation is summarized in Figure 4-3.

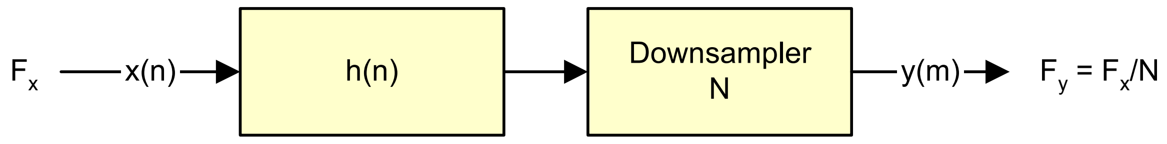


Figure 4-3: Decimation by a factor N. The sequence *x(n)* is downsampled to produce *y(m)*.

The anti-alias filter, *h(n)*, is a digital low-pass filter and should be designed to provide sufficient cut-off at $F_y/2$. The system sample rate was chosen to be f_s = 50 Hz in Chapter 3, so *h(n)* will have to attenuate the signal enough at 25 Hz. The bandwidth of the output signal is set by *h(n)* and care should be taken in its design. This digital filter will have a high sample rate (F_x) and comparatively low cut-off frequency.

4.2.3 Combined filter response

The complete filter design requires 4 parameters: The required system bandwidth *B*, the downsample ratio *N*, the analogue anti-alias cut-off frequency f_c. The sample rate is f_s = 50 Hz.

A bandwidth of around *B* = 10 Hz would be sufficient for most applications. Even MIT's aggressive X–Cell has control bandwidths of below 12.5 Hz [14]. The choice of *N* is largely dependant on the A/D device and processing capability of the acquisition system. A value of *N* = 20 implies more than 4 times ($log_2(20) = 4.32$) noise reduction, more than adequate for most applications. This implies an oversampling rate of 50 × 20 = 1000 Hz and that f_c should be below 500 Hz. To achieve good attenuation at the Nyquist frequency and make component selection easy, a value of $f_c \approx 110$ Hz was chosen.

Figure 4-4 shows the magnitude and phase response of the design. The value of *B* was lowered to 8 Hz to reduce computational complexity of the digital filter, while still providing good attenuation. The digital anti-alias filter (*h(n),* from Figure 4-3) was implemented with a second order Butterworth design. The figure shows that the analogue anti-alias filter's phase has little effect on the output within the system bandwidth.

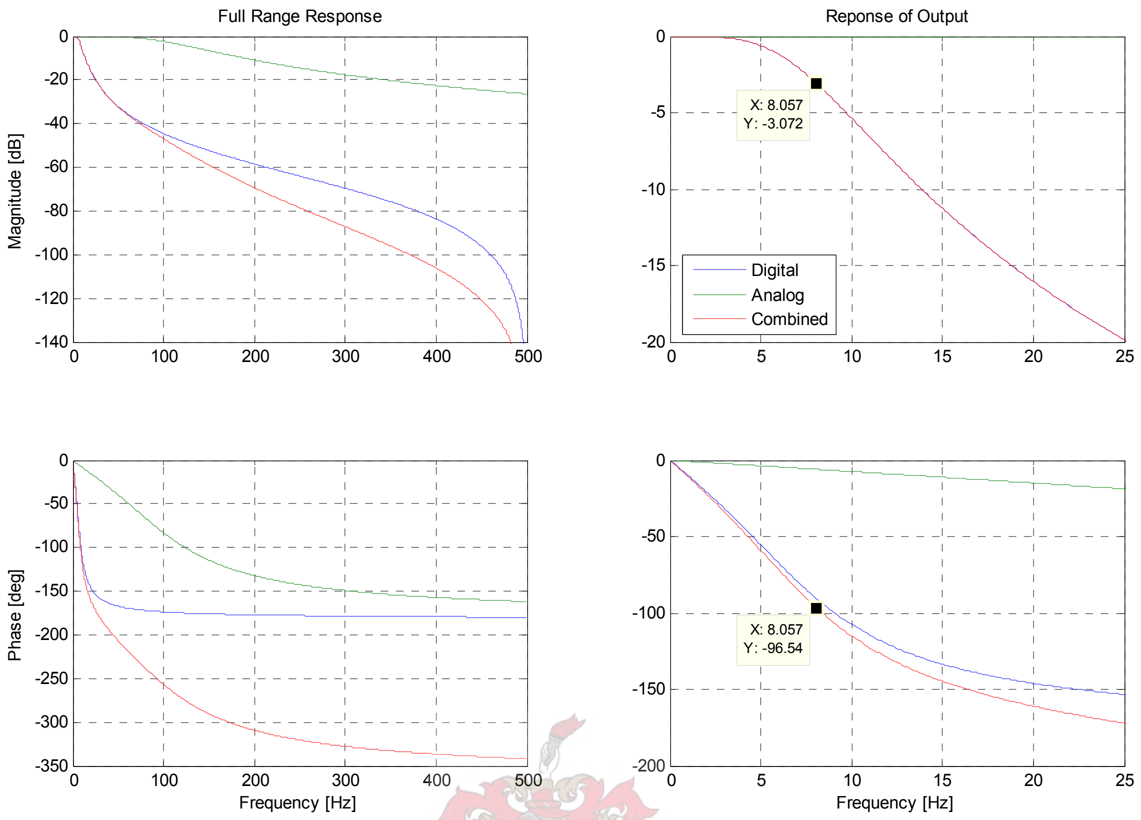


Figure 4-4: Magnitude and phase response of CANsens filter design.

4.2.4 Filter Quantization

The effect of implementing the digital anti-alias filter on a microprocessor should also be considered when looking at the frequency response. A microprocessor typically does not have any hardware to handle floating point mathematics; therefore any such operations must be emulated with integer math. A full discussion will be presented in 4.3.3, since it is heavily dependant on hardware. Although care must be taken with severe rounding errors, it can be assumed here that the constants of the digital filter can be represented with 16 bit integers. For the filter form of equation 4.4, a second order Butterworth filter has the constants listed in Table 4-2 (a_1 is normalized to 1).

$$a_1 \cdot y(n) = b_1 \cdot x(n) + b_2 \cdot x(n-1) + b_3 \cdot x(n-2) - a_2 \cdot y(n-1) - a_3 \cdot y(n-2) \qquad \textbf{4.4}$$

	Original Butterworth	16 bit Quantized	Error
b_1	0.00059496	0.00057983	2.5426 %
b_2	0.00118992	0.01190185	0.0220 %

	Original Butterworth	16 bit Quantized	Error
b_3	0.00059496	0.00057983	2.5426 %
a_2	-1.92982981	-1.92984009	-0.0005 %
a_3	0.93220965	0.93222046	0.0012 %

Table 4-2: Filter coefficient quantization.

The frequency response is shown in Figure 4-5. The magnitudes differ only slightly (the quantized filter is down 0.11 dB in the passband), while the phase difference is a maximum of 0.003 degrees over the entire range and therefore not shown.

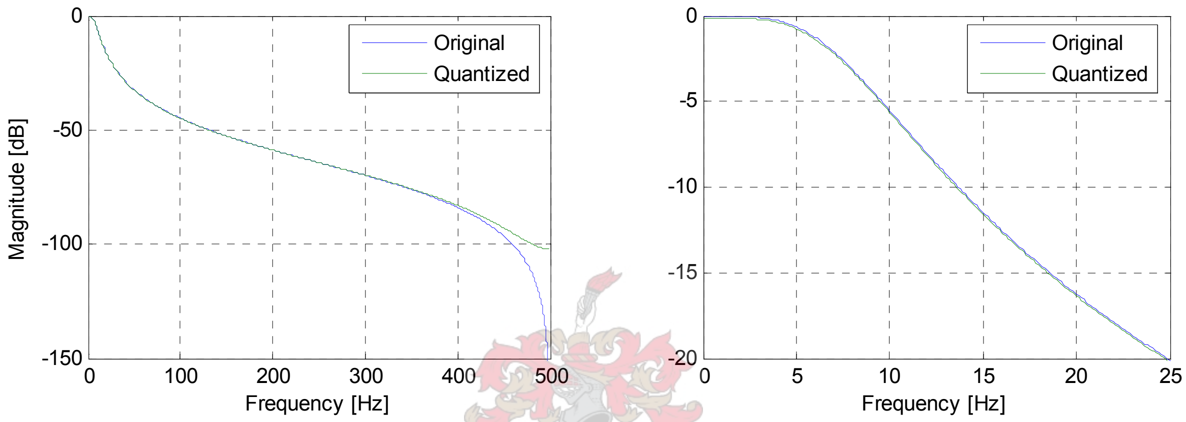


Figure 4-5: Magnitude response for quantized filter.

4.3 Design

Key areas of the design are the hardware chosen, the software structure and the filter implementation. Important design criteria are price, and using existing resources at the University of Stellenbosch's engineering department.

4.3.1 Hardware

The two key elements of the hardware design are the choice in processor and A/D device. Most 8 bit microprocessors have integrated A/D modules, with typical resolutions of 10 or 12 bits ([51], [43]). At the start of the CANsens design, Microchip's dsPIC® range [50] was still unreleased. This microprocessor is a 16 bit device, offering 12 bit A/D modules and hardware 16 bit multiplication, a feature that would greatly aid the digital filtering process.

Figure 4-6 shows an overview of the CANsens hardware layout with key components identified.

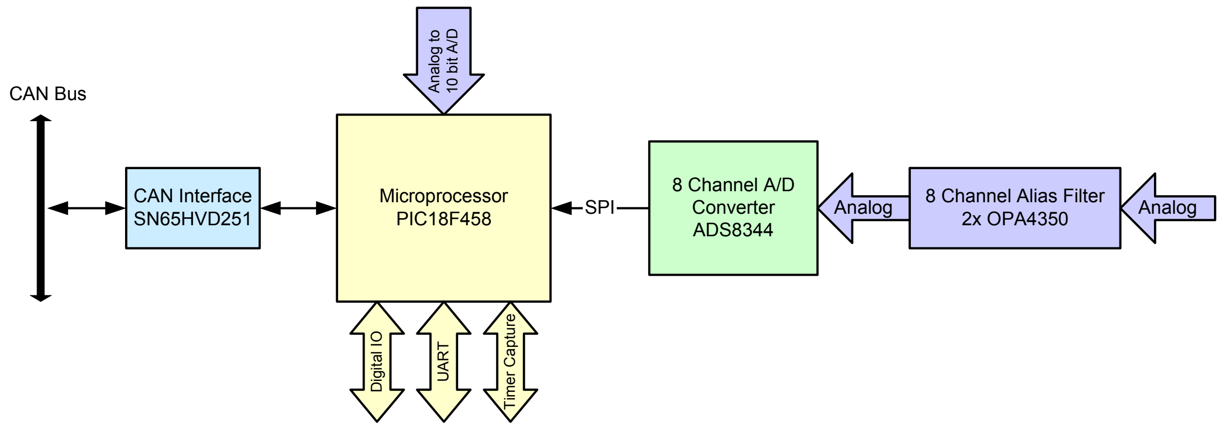


Figure 4-6: CANsens functional block diagram.

Microchip devices are in wide use in the engineering department at the University of Stellenbosch and were an obvious choice as the microprocessor for CANsens. Therefore, the 18F485 [51] was chosen for its list of features, and since no other low-cost micro-processor with a 16 bit A/D module was available. This microprocessor has 10 bit A/D modules, which are inadequate for most high resolution sensors, as seen in 4.1, but could provide auxiliary analogue measurements.

An external A/D device was needed. Initially, simultaneous sampling A/D ICs with parallel data interfaces were considered, but these were too slow, too expensive, or difficult to obtain. A solution was found in a component from Texas Instruments, the ADS8344 [56]. It is a high speed, serial A/D converter with 8 channels and 16 bit resolution. Choosing a component with 16 bit resolution leaves it up to the programmer to use all 16 bits, or discard some of the lower bits to save memory when high resolution is not needed.

A component that could be overlooked is the voltage reference for the A/D. The A/D requires a very stable reference: Assuming a 5V range, at 16 bits, this equates to $5 / 2^{16} = 61\ \mu V$ per A/D tick. Any voltage drift that the reference shows should be well below this value. The REF3140 [58] is recommended by Texas Instruments for this task. It exhibits very low temperature drift, as well as good voltage stability over time. The device provides a 4.096V reference, reducing the dynamic range that the A/D converter can use, but no other reference could be found that uses a 5V supply, provides a higher reference with less or equal drift.

The final component choice of interest is the operational amplifier (op-amp) used for the anti-aliasing filters. All the sensors of Figure 4-1 have bandwidths below 10 kHz, so most general purpose op-amps would be sufficient, but it should have very low noise figures to ensure that the sensor signal is kept as clean as possible. The op-amp must be able to drive its output down to 0V, to ensure that the dynamic range of the A/D is not reduced further. The OPA4350 [57] fits all these criteria, while also providing a compact solution by fitting 4 op-amps in one package. The analogue anti-alias filter was implemented with a second order Butterworth configuration.

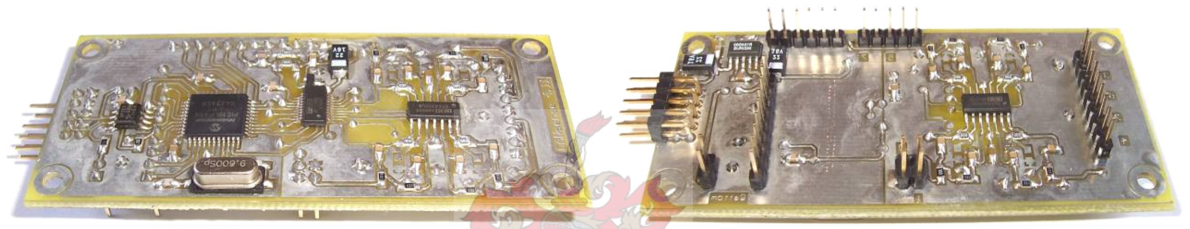


Figure 4-7: CANsens bottom layout (left) and top layout (right).

These components resulted in a small layout, with dimensions of approximately 40 × 89 mm. Two boards can be mounted next to each other on a PC/104 stack. The design allows the sensors, assembled on a separate board, to plug into the CANsens board. Figure 4-7 show an assembled CANsens node. Refer to Appendix C for further technical details about CANsens, including circuit diagrams and printed circuit board layouts.

4.3.2 Software[13]

The software onboard the PIC18F458 was mostly written in C to make the code simple to modify. When necessary, sections where re-written in assembly to optimize execution speed. Certain portions of the code where also *unrolled*. This refers to the method of taking certain blocks of code out of a loop to save variable space, at the expense of code size. The real benefit of code unrolling on the PIC18F458 is that variable indexing of arrays is avoided

[13] The full source code was too extensive to include as an Appendix, but available on the companion CD-ROM disk.

(saving about 10 instructions, as shown by Microchip's MPLab simulator). Figure 4-8 shows the main structure of the CANsens program.

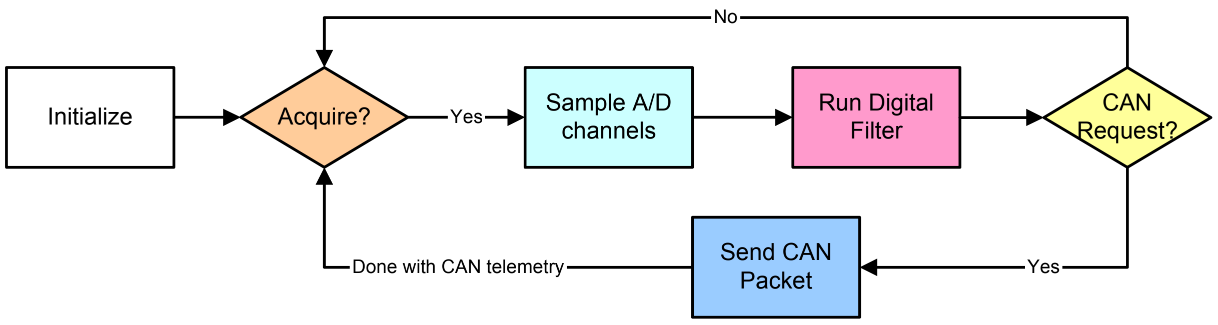


Figure 4-8: CANsens software flow diagram.

After initialization, the steps of the main loop are as follows:

- The loop waits for a flag to indicate that acquisition should start. The flag is set by a hardware timer, running at the oversampling rate (1000 Hz).
- The A/D devices are sampled. The external and internal A/D devices are sampled in parallel. This takes about 200 μs. At the output rate of 50 Hz, this seems like near-simultaneous sampling.
- The digital filters are run on all the channels and afterwards the states are updated.
- The CAN receive buffer is checked for any messages that were received. If data was requested, it is placed in the transmit buffers and transmission is initiated. If a command was received, it is executed.
- The program returns to an idle state, waiting for the next acquisition cycle.

Software interrupts are avoided to ensure that the acquisition process executes in a predictable manner.

4.3.3 Filter Implementation

From a programming perspective, the digital filter is the sum of product terms. The microprocessor can multiply two 8 bit values and store them in two 8 bit accumulators, thereby forming a 16 bit result. It also supports hardware addition and subtraction of 8 bit values.

The input from the A/D is 16 bits wide and would need 2 bytes for every channel. If the input is thought of as a number from 0 to 1 and the filter has unity gain, the output value would also be scaled from 0 to 1. This will only be true if the filter constants are scaled in the same manner: The full 16 bits should represent a value of 0 to 1. A multiply operation of 16 bits times 16 bits would produce a 32 bit result. The most significant 16 bits would represent a value from 0 to 1. The least significant 16 bits simply adds resolution to the result.

Referring to Table 4-2, the absolute value of the filter constants are in the range of 0 to 2. Conceptually, they must be normalized before being used. Multiplication or division by 2 on a microprocessor is a fast operation: The bits are simply shifted left or right. The normalization is therefore done with regards to the minimum power of two that is larger than the maximum absolute value of the filter constants, in this case 2.

The filter algorithm allows any 2nd order Butterworth filter. The cut-off frequency can be changed in this manner without rewriting the filter source code. A MATLAB script was created to generate a C header file for the 16 bit constants (Appendix G-1)

The filter code was implemented in the following manner:

- The constants of the numerator (b_1, b_2, b_3) where grouped first. They will not cause an overflow in the accumulator. Avoiding overflow checking saves time.
- Next, the large values of the denominator (a_2, a_3) are multiplied with the relevant states. Overflow is checked.
- The accumulator result is scaled to the correct range.
- The states are updated. The full 32 bit accumulator is saved for the output states to increase resolution and avoid rounding errors.

4.4 Specifications

The final CANsens node has the following specifications, listed in Table 4-3.

Parameter	Value	Notes
Supply Voltage	5V – 12V	6.5V Recommended
Supply Current	65.96 mA	Normal operation, no sensors connected, not connected to CAN bus, 6.5V supply.
16 bit Channels		
Number	8 max	Software selectable
Input voltage range	0 – 4.096V	+/- 0.2% max with [58] as voltage reference
DC Input impedance	49.5 kΩ (1.5 R)	Dependant on analogue anti-alias filter resistor R.
10 bit Channels		
Number	4 max	Software selectable
Input voltage range	0 – 5V	+/- 2% (Regulator specification [59]), With 6.5V supply
DC Input impedance	NA	Maximum recommend analogue source input impedance is 2.5kΩ [51]
Additional peripherals		
CAN		CAN version 2.0B
UART	30 – 10M baud	UART baud rate range and accuracy dependant on choice of crystal oscillator
Timer capture input	16 bit timer resolution	16 bit timer input capture is software programmable for rising/falling edge capture
Digital IO Lines	4 lines	2 shared with microprocessor programming lines
Physical parameters		
Board Dimensions	40.39mm (W) x 89.66mm (L)	
Mounting Hole Spacing	32.00mm (W) x 81.53mm (L)	3mm hole size
Weight	19.18 g	

Table 4-3: Summary of CANsens specifications.

4.5 CANsens-IMU Implementation

The most important function of CANsens is to form the basis for an inertial measurement unit (IMU). The O-Navi Gyrocube 3A [54] was considered as the sensor assembly, but it could not be imported to South Africa. The same type of sensors where assembled on a single PCB by Johan Bijker[14]. A Honeywell 3-axis magnetometer [46] was also added, along with a connection for a SensComp ultrasonic sensor [55].

The final IMU used for this project has three Analog Devices ADXL103 accelerometers [40], with a range of ±1.7g. The rate gyroscopes are Analog Devices ADXRS401 [41], with a range of ±75°/s. The IMU was vibration mounted with rubber grommets to isolate serious mechanical vibration. The standard CANsens software was extended to incorporate the ultrasonic sensor. The ultrasonic sensor has a range of 0.41m to 10.7m and returns this range as a 16 bit integer.

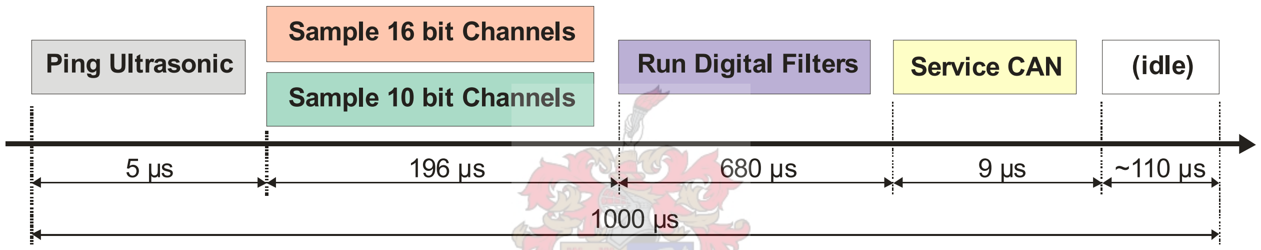


Figure 4-9: CANsens IMU timing. Values are rounded to the closest µs. (Not according to scale)

4.6 Recommendations

CANsens could become a great solution to many sensor problems. It provides a solid interface that is extremely flexible. Currently CANsens is sufficient as a data acquisition device, but it could be improved to provide even better features. Listed below are a few recommendations that the author feels should be investigated further.

4.6.1 Hardware

- The PIC18F458 can be replaced by the dsPIC30F4013. Although not pin compatible,

[14] M.Sc.Ing Student, 2005-2006, University of Stellenbosch. jbijker@sun.ac.za

this microprocessor has the same package and would require minimal modification to the PCB layout. It has the same amount of program memory, but all other features are better. The 16 bit hardware multiplication would benefit the digital filter and allow the efficient execution of higher order filters.

4.6.2 Software

- CANsens would benefit from firmware upgrading via the CAN bus. New filters could be tested without disassembling the avionics.
- The digital filter should only run on necessary channels. This would be an easy modification, and could even be switched on or off via CAN.
- The oversampling rate could be reduced to 800 Hz, instead of 1000 Hz. This would introduce a minor amount of latency, but the digital filter could be improved.

Chapter 5 Simulation

A dynamic, non-linear model of an X–Cell helicopter was discussed in Chapter 2. The proven accuracy of this model presents the opportunity to verify and refine any control strategy long before it is tested in the field. An existing implementation of this simulation will be investigated as a tool to aid the design of a controller for the X-Cell. The simulation could be a valuable component of the autonomous platform by providing a means to test the avionics A very simple controller for hover will be developed to demonstrate the capabilities of the simulation.

5.1 Simulation environment

Velez, *et al.* [36] demonstrated PID and fuzzy controllers with a computer simulation of the X−Cell helicopter. The model used was that of Gavrilets *et al.* [13], implemented on Mathworks Simulink®. The X−Cell is fully modelled, including engine governor and servo actuators. The parameters identified in [13] for the X−Cell are easily accessible and make the simulation fast to set up and customize.

Figure 5-1 shows the X−Cell simulation block. The simulation contains all the dynamics and allows access to all the states. It requires the control inputs (in degrees) to the actuators and engine throttle (the engine governor is modelled externally). Provision is also made for wind disturbances. Nearly all parameters of the simulation are configurable; symbols are clearly marked and names agree with those used by Gavrilets *et al.* [13]. Even the change in mass of the helicopter due to fuel flow is modelled. The helicopter will loose engine power when the fuel tank is empty.

Another useful tool for simulation is the AeroSim blockset [61] for Simulink®. This component offers various aeronautical simulation tools to develop dynamic 6-degree-of-freedom aircraft models. AeroSim also offers a few useful input and output functions: A standard PC joystick can be used to input real time commands into Simulink® while the output of the simulation can be sent to either Microsoft Flight Simulator or FlightGear. The latter is a 3D, open source flight simulator. These I/O features led to the inclusion of AeroSim as part of the simulation environment for the X–Cell. Since the model of Velez, *et al.* [36] is so comprehensive, none of the other functions of AeroSim were necessary. The output of the X–Cell model could be sent directly to FlightGear via the AeroSim interface.

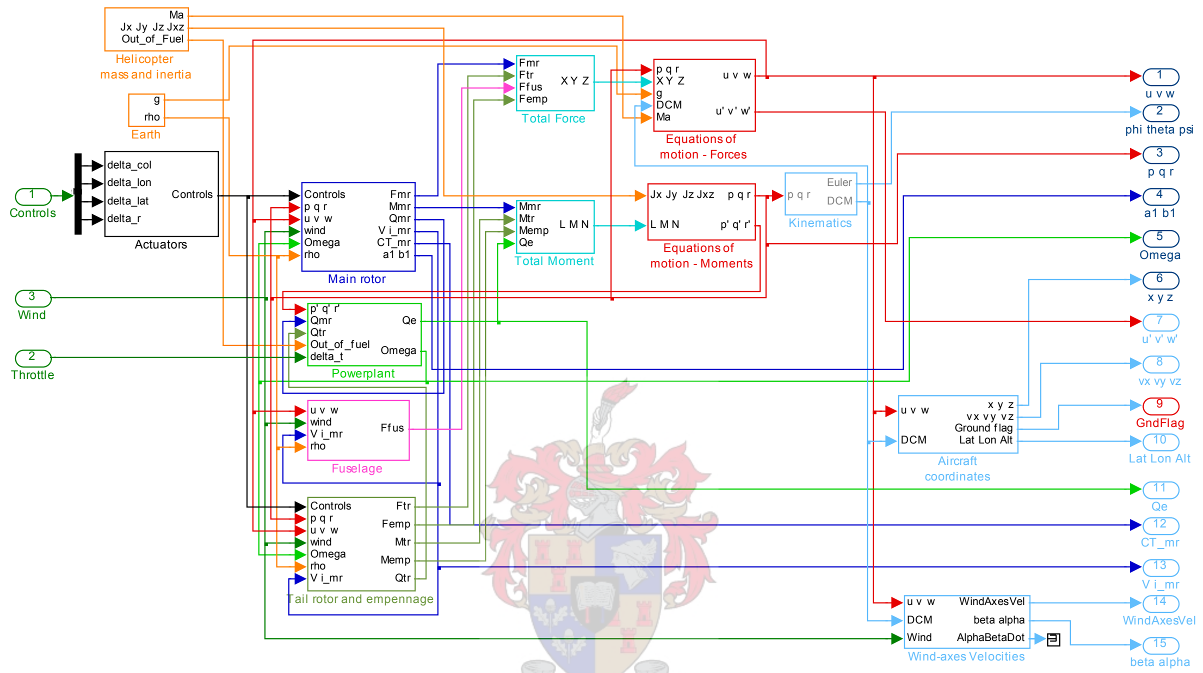


Figure 5-1: Simulink® block for the X–Cell model.

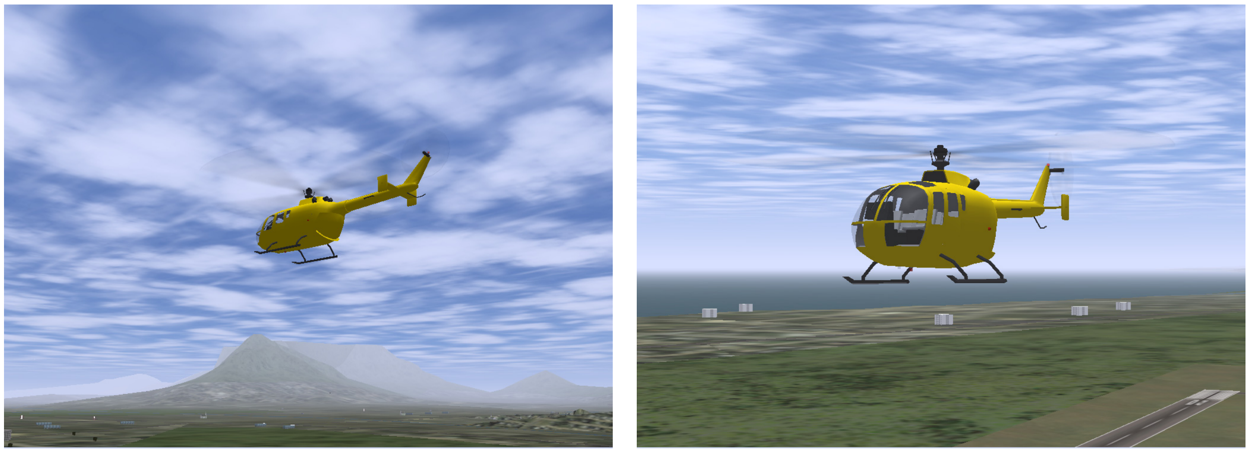


Figure 5-2: FlightGear output of Simulink® simulation with Cape Town scenery.

FlightGear requires the position of the aircraft in latitude, longitude and altitude, Euler angles and the airspeed of the vehicle. Any internal 3D model can be used to represent the vehicle. Figure

5-2 shows the output of a simulation. Scenery from most of the land masses are available, with Cape Town depicted in the figure below.

5.2 Basic Hover Controller

The simulation presents an environment to test various control strategies before risking the actual helicopter. In this way, the controllers can be fine-tuned before they are used in the field. A basic control design for hover flight will be discussed below, to demonstrate the power of the simulation. Most controllers where designed to mimic a careful pilot: The helicopter's movements are slow and docile. Fuel flow and its effects on total mass were ignored.

In Chapter 2, the work done by Carstens [4] describing a decoupled model for hover flight was summarized. Carstens used successive loop closure to systematically implement his control strategy – a controller was tested to control a specific degree of freedom while the pilot controlled the rest of the helicopter. Other controllers could then be added until all degrees of freedom were controlled.

This presents an interesting problem: The simulation does not have a simulated pilot included, making it difficult to use successive loop closure, since the simulated helicopter is not stabilized at all. Initial testing involved finding a trim condition that kept the helicopter in an open-loop hover for long enough to test the various controllers separately.

5.2.1 Altitude

The altitude controller was designed first, using the model of section 2.4.1. A simple gain would have sufficed for the single pole and integrator system, but the settling time was about 10 seconds – perhaps too slow. Lead compensation was used to improve the response. The critically damped closed-loop poles where placed to provide a settling time of around 6 seconds.

In Figure 5-3, the simulated response (and collective actuator command, δ_c) of a 1 m altitude step with the implemented controller is shown. The controller was designed in the continuous time domain and then transformed to a discrete controller with sample time of $T_s = 0.02$ s (50Hz) to simulate the architecture developed in Chapter 3. The Simulink implementation also included the initial trim value for the collective actuator (Figure 5-4). This prevents a serious transient as the simulation starts; effectively it prevents the helicopter from free-falling initially.

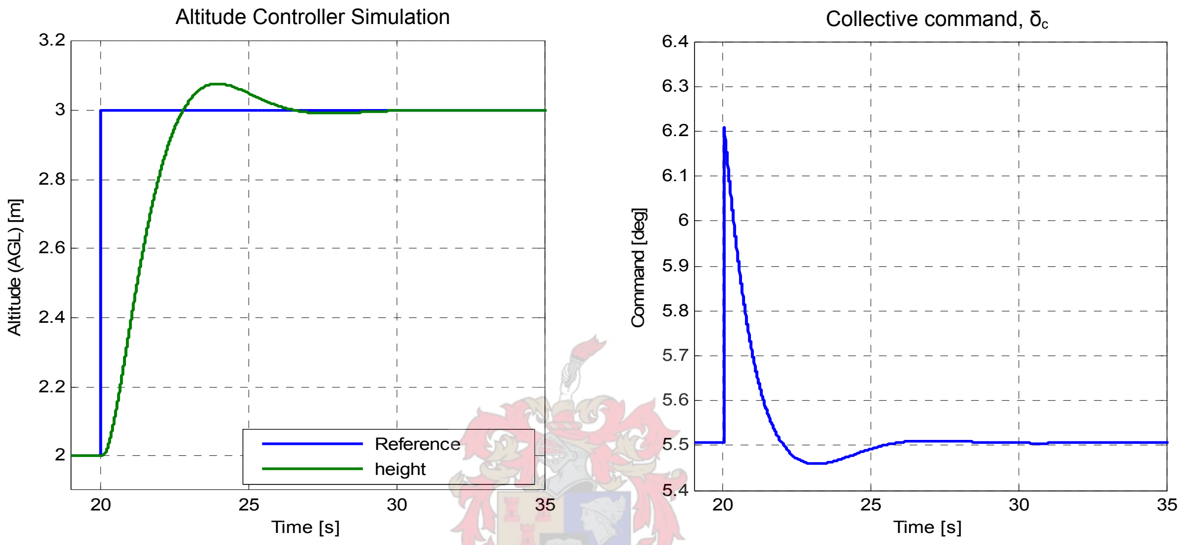


Figure 5-3: Simulated response of lead-compensator controller for altitude.

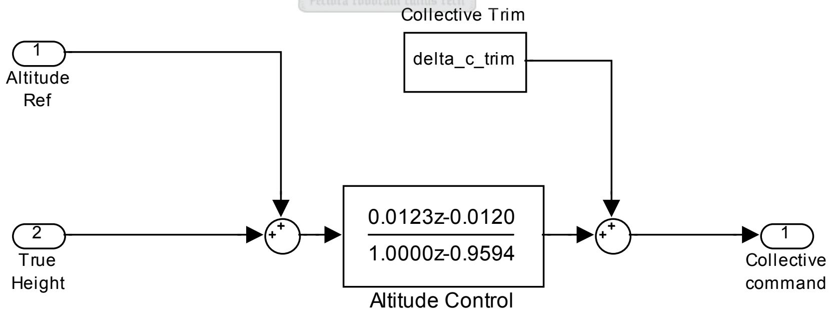


Figure 5-4: Simulink implementation of altitude controller.

5.2.2 Heading

The simulation does not include a yaw damper, since the X-Cell of the MIT team relied on its own avionics to control the fast dynamics of the tail [13]. Equation 2.8 simplifies to 5.1

(which is also equivalent to the altitude model of eq. 2.5), with the addition of the yaw angle.

$$\begin{aligned} \dot{r} &= N_r r + N_{\delta r}\delta_r \\ \dot{\psi} &= r \end{aligned} \qquad \textbf{5.1}$$

This produces the simple transfer function:

$$\frac{\psi}{\delta_r} = \frac{N_{\delta r}}{s(s - N_r)} \qquad \textbf{5.2}$$

Another lead compensator was designed to achieve a settling time of approximately 1 second. The compensator had a phase margin of 122º, and an infinite gain margin. The step response and root locus of the modelled yaw dynamics are shown in Figure 5-5. The fast response was chosen to ensure rapid reaction to disturbances. Initial experiments with the simulation indicated that the coupling from heave to yaw dynamics tend to be significant.

The simulated results are plotted in Figure 5-6, showing a settling time of 4 seconds, rather than the design value of 1 second. This indicates that the parameters of the model may be incorrect, or possibly that the model is incorrect. The response is still adequate, however.

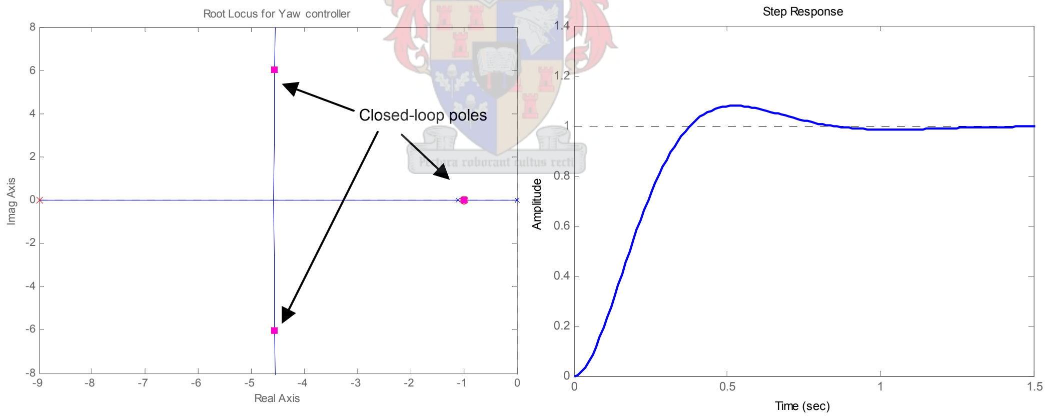


Figure 5-5: Designed response of yaw controller with linear model of .

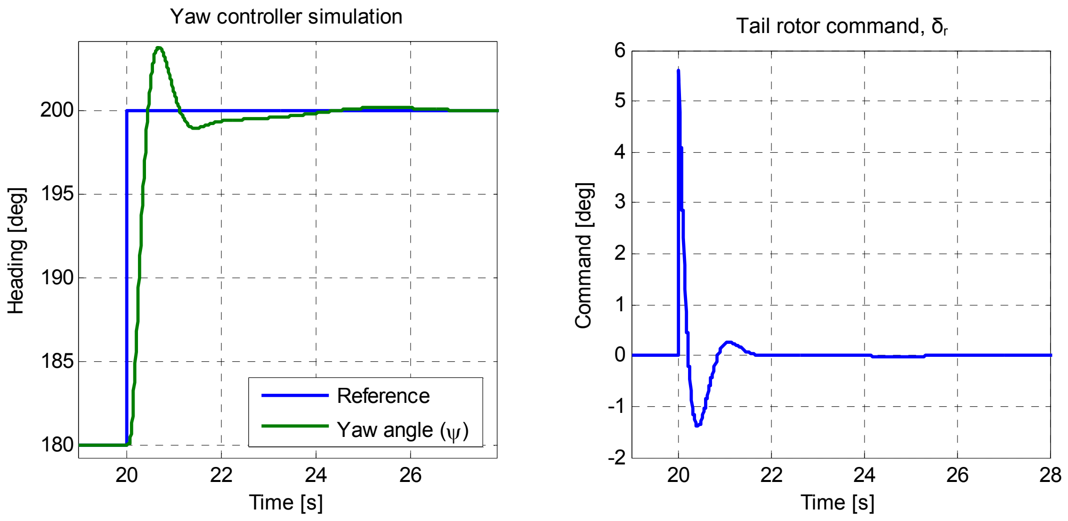


Figure 5-6: Simulated response of yaw controller.

5.2.3 Latitude and Longitude

Carstens [4] followed arguments presented by Mettler *et al.* [22] and designed the horizontal controllers for his Voyager E based on stability derivative published by Shim [31] for a Concept .60 size helicopter. Attitude dynamics for this helicopter is about 30 times faster than position dynamics. The goal is to first close in the inner attitude loop and then the slower velocity/position loops.

Carstens [4] used the simple control law of equation 5.3, where x_{ref} and y_{ref} denote the reference position inputs. The roll angle, pitch angle, longitudinal and lateral velocities were proportionally fed back to the relevant actuator command, with the gains $K_{\theta,\phi} = 0.02$, $K_{u,v} = 0.007$ respectively. The relative different in size indicate between $K_{\theta,\phi}$ vs $K_{u,v}$ indicate the aforementioned difference in dynamic response: The attitude (θ, ϕ) require larger gains to demand a larger control authority to correct change in attitude.

Proportional and integral control (PI) was employed for positions x and y, with the gains $K_{x,y} = 0.008$ (no indication of $K_{Ix,y}$ was given by Carstens). Some experimentation yielded that these values do work for the X-Cell, but a faster response, with much less overshoot, can be obtained with $K_{\theta,\phi} = 0.05$, $K_{u,v} = 0.01$, $K_{x,y} = 0.05$ and $K_{Ix,y} = 0.001$.

$$\begin{aligned}\delta_a &= -K_\phi \phi - K_v v - K_y (y_{ref} - y) - K_{Iy} \int (y_{ref} - y) dt \\ \delta_a &= -K_\theta \theta - K_u u - K_x (x_{ref} - x) - K_{Ix} \int (x_{ref} - x) dt\end{aligned} \tag{5.3}$$

Results of 1 m step commands to the respective controllers are presented in Figure 5-7. Maximum actuator commands were 0.3 degrees, and 1% settling time was 15 seconds.

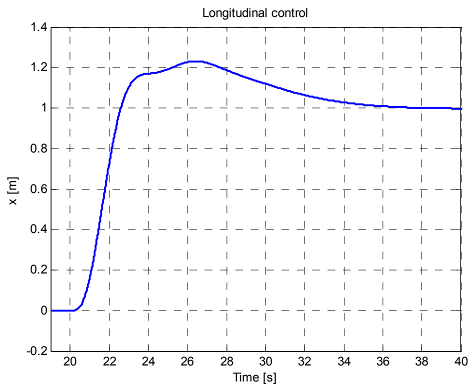


Figure 5-7: Separate 1m step inputs to lateral and longitudinal controllers

5.2.4 Coupling of dynamics

A useful feature of the simulation is that it allows the investigation of the influence that the various controllers have on one another. This cross-coupling was modelled by Mettler *et al.* [23] (refer to equation 2.4), but was ignored during the basic design presented above.

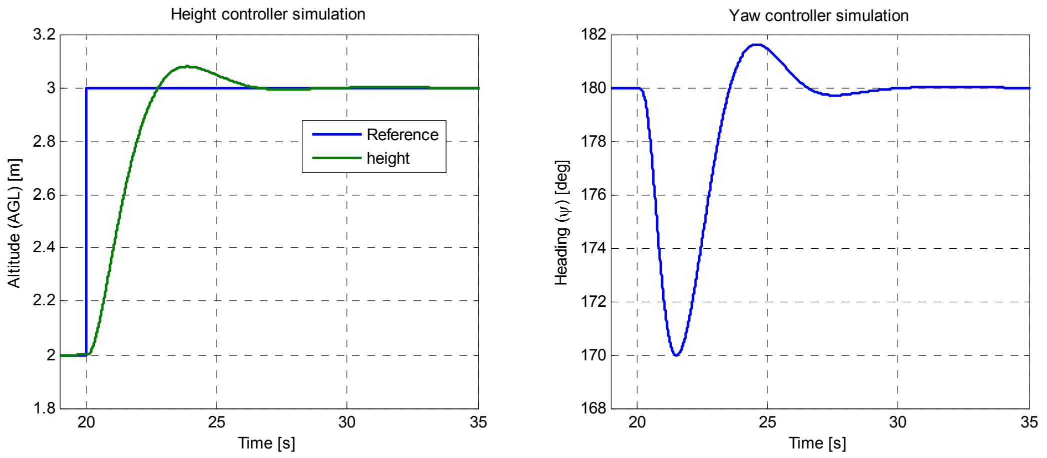


Figure 5-8: Influence of height step on heading.

Figure 5-8 demonstrates the effect that a height step has on heading. The acceleration of the main rotor induces a moment on the tail and pushes it from the reference heading. In this case, 10º is a small disturbance, but larger step commands will yield larger deflections. An easy way to remedy this is to add a simple first order, low-pass pre-filter to the reference inputs.

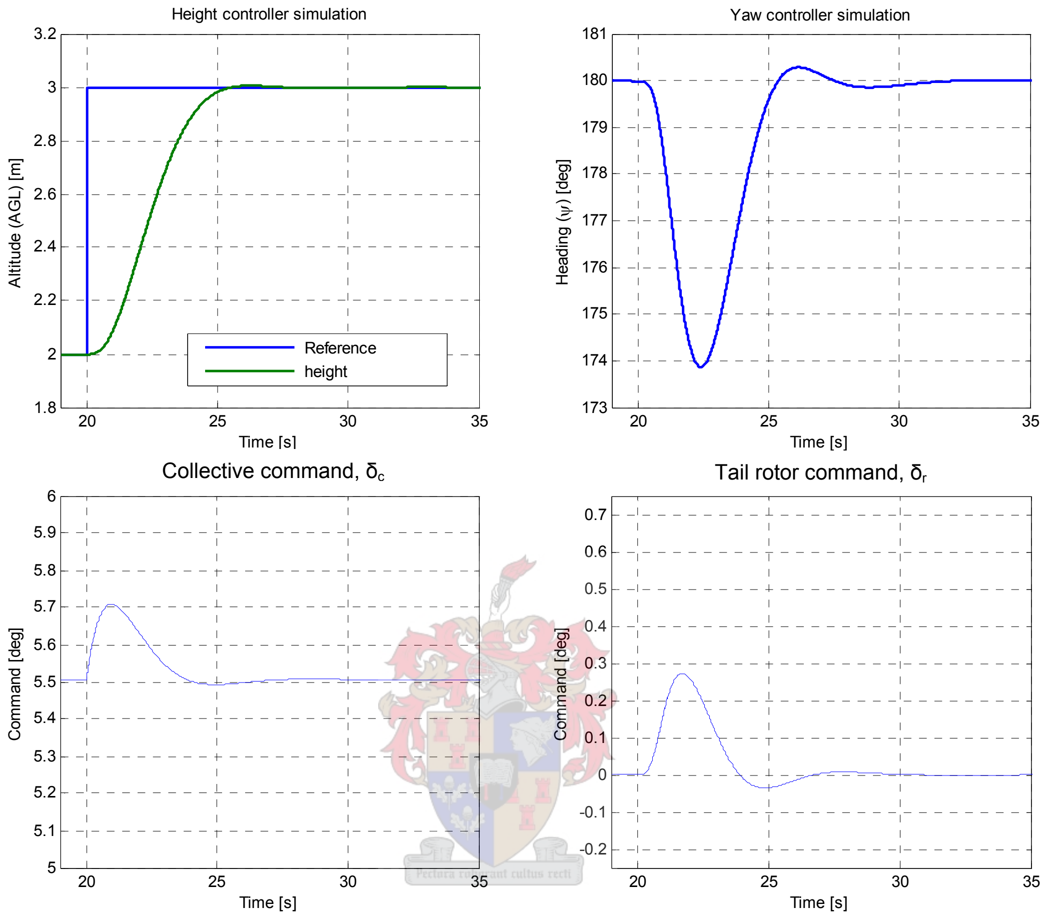


Figure 5-9: Pre-filters added to reference input. Step response and coupling is improved.

In Figure 5-9, the height response and the heading disturbance has been improved (compare to Figure 5-3 and Figure 5-6). The pre-filters have the added benefit that they smooth out the actuator command, thus creating the type of a curve a human pilot might follow. The actuator commands shown above all react instantly to a step command, but a human pilot will ease the control forward to produce an altitude step, for instance.

5.3 Recommendations

This chapter evaluated a comprehensive simulation of an X-Cell helicopter. Simple control was developed to maintain hover and provide the basis for further control strategies. The detail of the simulated model offers the opportunity to test the various effects that a new design may exhibit, without risking the helicopter.

As demonstrated in 5.2.4, the coupling effects that were ignored during the design phase could be detected in the simulation and remedied. Another benefit is that helicopter's accelerations, rates and actuator commands are available during the design. The limits of the sensors and actuators can now be taken into account during the design.

In the future, the simulation could be integrated with a hardware-in-the-loop system. This would allow the controller, running on the avionics to control the simulator. Such a system is already being designed[15], and will likely provide another level of testing before the helicopter is risked during a flight test.

[15] Willem Hough, M.Sc.Ing Student, 2005-2006, University of Stellenbosch. whough@sun.ac.za

Chapter 6 Verification and Measurement

The design criteria of the avionics will be evaluated through measurement, where possible. Some methods of calibration and accuracy will be discussed. Measurements taken during flight tests will be presented to gauge the vibration effects of the vehicle, and provide some insight to the sensor performance.

Some measurements of the bare CANsens node will be discussed to verify certain design goals. Then, the power system of the avionics will be assessed before the most important feature of the operating system environment, the latency, is discussed. The bulk of this chapter will be dedicated to the sensor data of the IMU, gathered during flight tests.

6.1 CANsens Performance

As the front-end of most sensors, it is important to verify the performance of CANsens. Calibration and noise performance are discussed. The 16 bit channels are discussed in more detail, since these channels are the primary focus of CANsens. The 12 bit channels are considered auxiliary. Additional information can be found in Appendix C-6.

6.1.1 Gain and Offset

CANsens will mostly be connected to a sensor that will require calibration to exactly relate the measured voltage with the physical quantity being measured. But it may be useful to calibrate the A/D device by itself to cater for sensors that are already calibrated, or sensors that are difficult to calibrate directly.

All the channels of a CANsens node were individually connected to a FLUKE 5100B Calibrator. A constant voltage of 0.4000V was applied for a few seconds to allow the digital filter to settle. Then a 3.6000V voltage was applied. The results of all 8 of the 16 bit channels are shown in Figure 6-1.

The data was used to do a linear calibration of the inputs and produce the relation $V_{true} = V_{meas} \times m + c$, where m is the gain and c the offset. Each sequence was averaged and then the coefficients were calculated. They are listed in Table 6-1. The MATLAB script to automatically generate this data is shown in Appendix G-2. Note that the input voltage is always attenuated by virtually the same amount (and therefore needs a gain greater than 1). This is predicted in 4.2.4 to be about 0.11 dB, or 1.013.

Channel	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Gain (*m*)	1.0132	1.0138	1.0131	1.0136	1.0135	1.0134	1.01364	1.0135
Offset (*c*)	-7.219	-13.306	-12.031	-17.864	-16.596	-0.380	-11.023	-6.081

Table 6-1: Gain and offset calibration coefficients for 16 bit channels

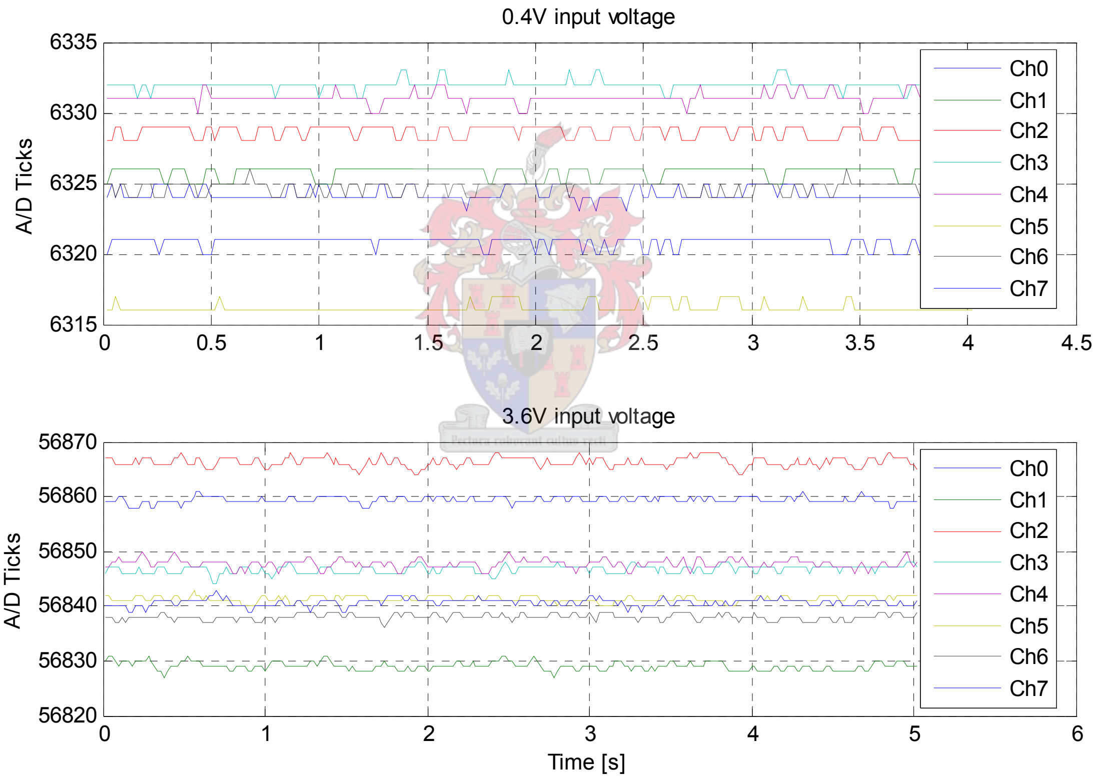


Figure 6-1: Gain and offset calibration of CANsens ('Ch *n*' indicates the nth channel).

6.1.2 Quantization Noise

Since the calibration device used in section 6.1.1 offered a low noise voltage source (<0.2µV), the measurements were also used to calculate the noise figures for CANsens. The noise of the voltage source is also assumed to have a normal distribution. The standard deviation (σ_n) is calculated for each of the 16 bit channels, and since 3σ represents 99.7% of measurements, this is taken as a good indication of the quantization noise. The results are listed in Table 6-2.

3σ value of channel	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Input: 3.6V	1.80	2.41	2.50	1.94	2.48	1.81	1.89	2.08
Input: 0.4V	1.40	1.20	1.45	1.04	1.19	1.09	1.40	1.11

Table 6-2: Three times the standard deviation (in A/D ticks) for 16 bit CANsens channels

The maximum value from Table 6-2 is 2.5, or rounded up, 3. To calculate the quantization noise in bits, the maximum power of 2 that is greater or equal to this value should be calculated –in this case 2 ($2^2 = 4 > 3$). This means that the quantization noise is 2 bits, worse case. This may be a pessimistic estimation, since the values of Table 6-2 are typically below 2, indicating only 1 bit of quantization noise. For the 16 bit channels, the resolution requirements of section 4.1 are met.

6.2 Onboard Computer

6.2.1 Power consumption

A rough estimate of the battery life can be obtained by measuring the current drawn by the avionics during operation. Figure 6-2 plots the total current of the entire avionics package while the software, described in section 3.3.3, was running. The average current is about 1.5A. The drop in current at 20 s was due to the cooling fans briefly switching off (they are temperature controlled). The scale of the figure make the measurement seem noisy, but these peaks are due to the ultra-sonic sensor operating at 50 Hz – each ultra-sonic ping causes an instantaneous current spike as the device generates the high voltage necessary for operation [55].

At the average current of 1.5 A (from Figure 6-2) and nominal battery voltage of 11.1 V, the power is 16.7 W. With the 2000 mAh battery used for the battery, this would give a maximum running time of 80 minutes. A safety margin of 10.5 V was imposed to protect the battery. At a constant discharge rate of 1.5 A, this voltage is reached at half the battery's capacity [48]. This puts a lower limit of 40 minutes on the running time.

For the above analysis, the servos where excluded since they do not draw constant current and different vehicles could have a different amount of servos.

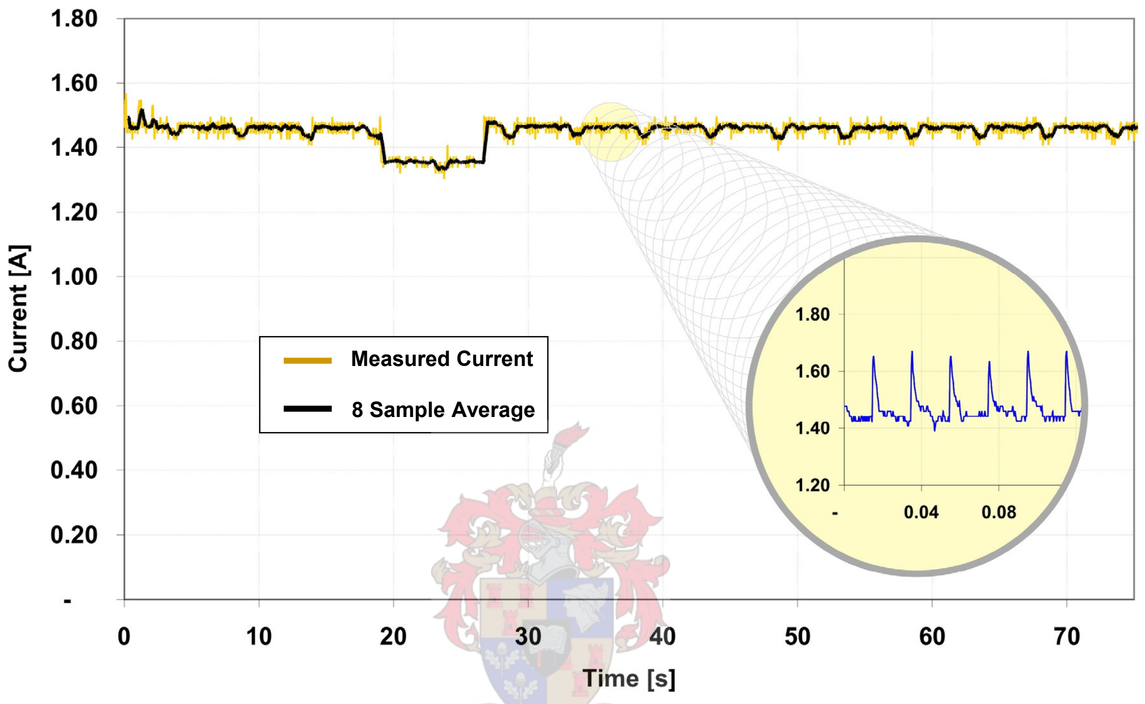


Figure 6-2: Total avionics current. Insert indicates ultra-sonic current.

6.2.2 Latency

Timing adequacy was determined by using the timing functions provided by the C-library. The time it takes for the UCA to receive data from the CAN hardware, process it, do all communication, upload the new actuator commands and return to the idle state, was measured and stored for each time step. The maximum value for all the time steps was also stored. The results of nine tests are listed in Table 6-3. The first eight tests were values taken during actual flight tests, lasting from 1 minute up to 5 minutes. The ninth test was done during a 22 minute laboratory test, chosen because it had the highest maximum time of all tests[16] done with the avionics.

Flight	1	2	3	4	5	6	7	8	9
Maximum time [ms]	4.19	4.56	4.58	4.55	4.51	5.25	4.01	4.72	6.52
Average time [ms]	0.640	0.620	0.620	0.618	0.639	0.641	0.639	0.640	0.640

Table 6-3: Measured latency of avionics during flight tests.

The average time of around 0.64 ms seems constant between the flights. The maximum time of 6.52 ms indicates that the avionics can currently cope with the processing load, but it is well above the average value. The most likely explanation is that the multitude of operating system buffers that handle the various I/O functions can potentially require service all at once. The UCA is then forced to wait while data is written to disk, input is processed from the serial devices, and data is written to the serial devices. On top of this, a specific time step may require more processing by the UCA as new GPS data becomes available, or RF commands are received. When all these events occur in a single 20 ms time step, the UCA's latency increases substantially, compared with the average value. This is where the benefits of real-time schemes come into play, since many of these systems offer some guarantee about the latency of the I/O functions.

Currently, the value of 6.52 ms (maximum) is still within limits, but the latency added to the execution time of the control algorithm must be below 20 ms, otherwise the UCA will miss the window to upload actuator commands. Such events are detected and have thus far never occurred during the normal operation of the avionics.

[16] There were 150 tests where the timing statistics were logged; including flight tests onboard the helicopter, calibration and other laboratory testing of the various systems.

6.3 IMU Calibration

The CANsens IMU was calibrated to scale the measured voltages to physical values. The avionics was mounted to a stabilized, 3-axis rate table (see Figure A-7) and calibration was done for the accelerometers and the rate gyroscopes. Axis cross-coupling was also investigated.

6.3.1 Accelerometers

The avionics was mounted close to the pivot point of the rate table to avoided measuring any centrifugal forces when rotating the table. The base of the table was assumed to be perpendicular with the gravity vector. Each of the three axes were aligned with the gravity vector and then rotated about another axis of the accelerometers, as depicted in Figure 6-3. Here, the sensitive axis was *y* and the rotation axis, *z*.

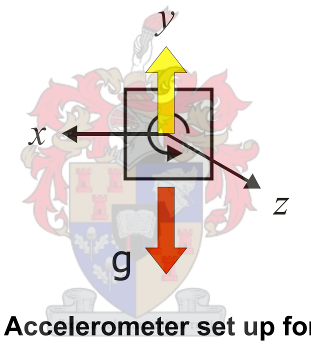


Figure 6-3: Accelerometer set up for calibration.

Note that with this configuration, the *x*-axis will also measure the gravity magnitude, but 90º out of phase: When the *y*-axis measures a maximum deflection, the *x*-axis should measure zero, since no component of the gravity vector is aligned with it. These concepts are illustrated in Figure 6-4. The *y*-axis was calibrated, with the *z*-axis the rotation axis. To automate the process, a MATLAB script was created to do the necessary calculation (see Appendix G-3). A sequence of maxima and minima are selected by hand for each calibrated axis. The script then searches for the relevant peak around these points, as indicated by the calibration points of Figure 6-4.

The scaling constant is then calculated as the average of the differences between the maxima and minima,

$$scale = \left(\frac{1}{n} \sum_{i=1}^{n} max_i - min_i \right)^{-1} \qquad \textbf{6.1}$$

where n is the total number of full deflections ($n = 5$ in the case of Figure 6-4), max_i the value of a maxima and min_i the value of a minima in equation 6.1. This procedure yielded the results of Table 6-4.

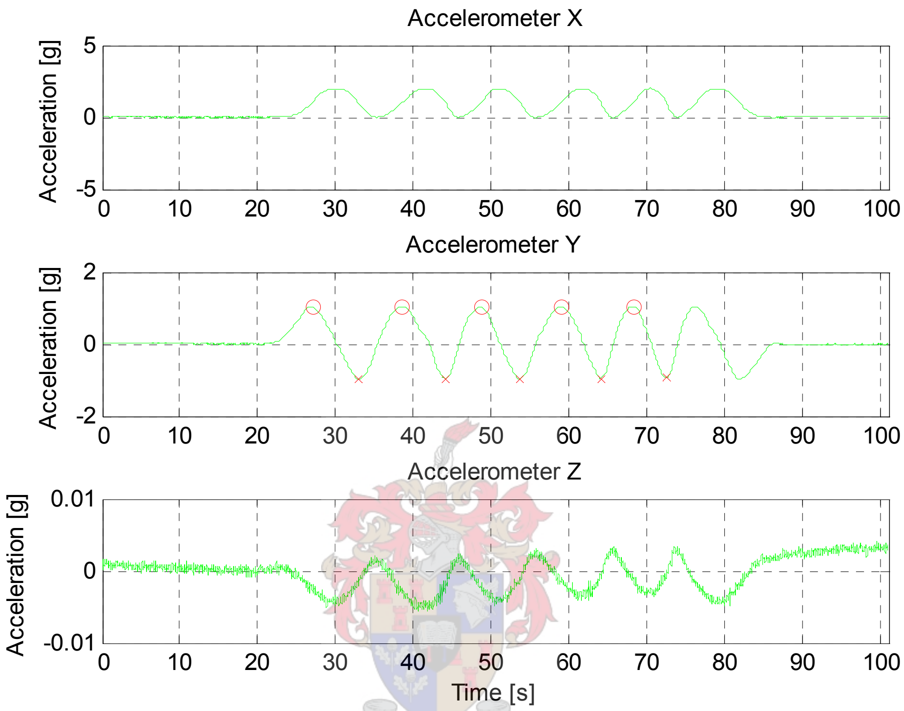


Figure 6-4: Calibration data of *y* axis. Note calibration points on Y plot.

IMU Axis	X	Y	Z
Scale factor [g/V]	1.1895	1.2027	1.1893

Table 6-4: Accelerometer Calibration.

Referring to the example of Figure 6-3, the rotation axis (z) should indicate no deflection during the measurement. A non-zero measurement on the rotation axis indicates a misalignment of the accelerometers. The angle between the two axes ($\theta_{relative}$) can be found by

$$\theta_{relative} = \arccos \frac{z}{y} \qquad \textbf{6.2}$$

and the misalignment (θ_{align}) is simply [3]:

$$\theta_{align} = \arcsin\frac{z}{y} \qquad \textbf{6.3}$$

The corresponding value of the rotation axis was taken once the MATLAB algorithm found the peak of the calibration axis. The misalignment of both the maxima and minima were calculated, and the maximum taken as θ_{align}. Results are shown in Table 6-5.

Misalignment with	X	Y	Z
X	-	1.796º	1.774º
Y	1.186º	-	0.075º
Z	0.789º	-0.119º	-

Table 6-5: Accelerometer axis misalignment.

A misalignment of 1.8 degrees from the *y*-axis to the *x*-axis would introduce a percentage error on the z-axis of 3%, or -30 dB. The misalignment is therefore small enough to ignore [3].

6.3.2 Rate Gyroscopes

The rate gyroscopes were calibrated with a similar setup as described in the previous section: The sensitive axis of the gyroscopes were aligned along the axis of rotation and the rate table was turned through a full 360º. The start (zero degrees) and end points (360 degrees) where identified and a MATLAB script calculated the scale values and the alignment errors. The measurement for the *x*-axis is shown in Figure 6-5. The calibration points are shown for the angle and the results to obtain degrees from voltage are listed in Table 6-6.

IMU Axis	X	Y	Z
Scale factor [degrees.s^{-1}/V]	68.21	67.14	69.35

Table 6-6: Rate gyroscope calibration.

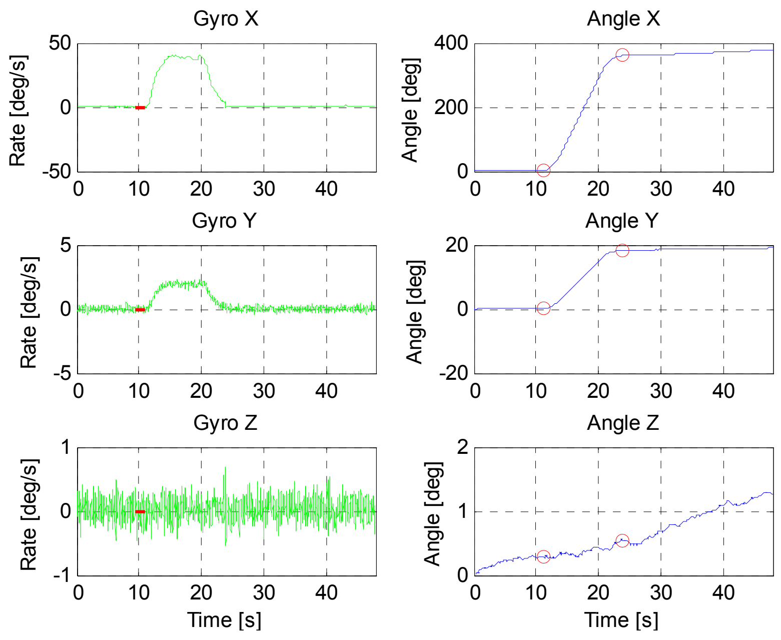


Figure 6-5: *X*-axis rate gyroscope calibration. Note coupling with *y*-axis.

Equation 6.3 can be used in the same manner here to calculate the misalignment. Table 6-7 shows that the maximum error is 2.9°, implying a 5% error. Although slightly larger than the accelerometer misalignment, this is still relatively small.

Misalignment with	X	Y	Z
X	-	-2.915°	0.018°
Y	-2.076°	-	0.106°
Z	0.644°	-1.534°	-

Table 6-7: Rate gyroscope axis misalignment.

6.3.3 Mounting

If the sensors were mounted exactly on the CG of the vehicle, the measurements would represent the accelerations and rates of the vehicle, but if they are offset by a certain distance, the general equations for 6-degree-of-freedom movement can be used to solve for the true physical values (of course assuming ideal sensors with no noise, etc). For the forces measured at the IMU,

$$\begin{aligned}
m\left[\dot{u} + wq - rv - a_x(r^2 + q^2) + a_y(pq - \dot{r}) + a_z(pr - \dot{q})\right] &= X \\
m\left[\dot{v} + ur - pw + a_x(pq + \dot{r}) - a_y(p^2 + r^2) + a_z(pr - \dot{q})\right] &= Y \\
m\left[\dot{w} + vp - qu + a_x(rp - \dot{q}) + a_y(rq + \dot{p}) - a_z(q^2 + p^2)\right] &= Z
\end{aligned} \tag{6.4}$$

where (*X, Y, Z*) represent the total force on the IMU along its three axes, and (a_x, a_y, a_z) are the translation from the CG. All other symbols are equivalent to those defined in Figure 2-3, but refer to the IMU. A similar set of equations exist for rotation rates of the body, but the measured rates and the true rates are equal if the axes of the IMU are aligned with the body axis, as is the case here.

For near-hover flight, velocities are close to zero, rates are small, derivatives also negligible and the squares of the rates are also insignificant. Since the distance the IMU was offset was relatively small (a_x = -0.08 m, a_y = 0.0 m, a_z = 0.2m), the overall effect of any corrections to the accelerometers readings would have very little effect during a hover. Some experimentation revealed that even when the helicopter was flying at low forward speed and performing gentle turns, no real effect could be seen. The corrections will therefore be omitted for the flight test data. They would apply to aggressive manoeuvres where velocities and angular rates are much higher.

6.4 Flight Test Data

Numerous flight tests were conducted to evaluate the avionics system. A trained pilot controlled the X-Cell and measurements were taken for the duration of each flight. The pilot performed some basic manoeuvres including hovering the helicopter, altitude steps and figure eight flights.

Testing was conducted on a secluded field to minimize risk to the public. Test equipment included a flight box containing the RC transmitter and a sundry of tools, a simple housing for an AeroComm RF module to serve as the ground-station interface, a Laptop to run the ground-station software (refer to Figure A-10), and a digital still camera with basic video capability.

The digital camera proved invaluable after flight tests. All flights were recorded and the video eased the identification of the various pilot commands and disturbances when analyzing the recorded data.

Below, the raw data collected during the various manoeuvres will be discussed to gauge the abilities (and shortfalls) of the avionics. All of this data was processed by the MATLAB interface mentioned in section 3.3.4. With the truly daunting amount of data collected, and plots available, this proved a very useful tool, indeed[17].

Unless otherwise stated, the governor was running at 1700 rpm (chosen by the pilot).

6.4.1 Vibration

Perhaps the most critical aspect of the avionics is its performance in the harsh, vibrating environment of a small helicopter. As a test case, a flight during a relatively steady hover will be examined. It was assumed that the only significant portion of the measurements is due to vibration. To provide some verification of the hover conditions, Figure 6-6 plots the measured height and ground velocity of the helicopter during the 12 second hover. It is clear that the altitude reading from the ultra-sonic sensor stayed within a 0.5 m range, and the GPS velocity was well below 0.5 m.s^{-1}.

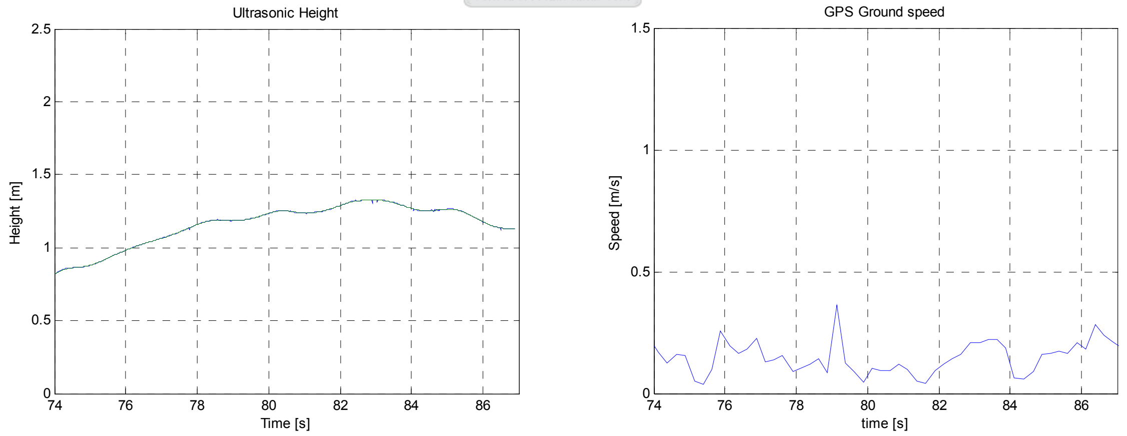


Figure 6-6: Altitude and GPS ground speed of X-Cell during a hover.

A frequency analysis (power spectral density, or PSD) was done on the accelerometers to

[17] The script was too large to include as an Appendix, but can be found on the companion CD-ROM.

identify any vibrations that may be present. From Figure 6-8 it is clear that a peak at 21.5 Hz dominates the graphs. A quick calculation reveals that this is due to the main rotor vibration: The peak is at 21.5 Hz, which can also be a signal at F_s – 21.5Hz = 28.5Hz, if the sample rate is 50 Hz. This frequency (28.5 Hz) translates into a main rotor speed of 1710 rpm, close to the governor setting of 1700 rpm. Therefore the vibration is almost certainly caused by the main rotor.

The vibration was filtered out by a second order, infinite impulse response notch filter with 2 Hz bandwidth (Figure 6-8). The wide bandwidth was chosen to accommodate some variation in the engine speed. With the filter applied, the PSD shows three other prominent peaks. These are at 7 Hz, 9.1 Hz and 13.5 Hz. The peaks were all removed by more notch filters, all with a bandwidth of 1 Hz. The result is shown in Figure 6-9.

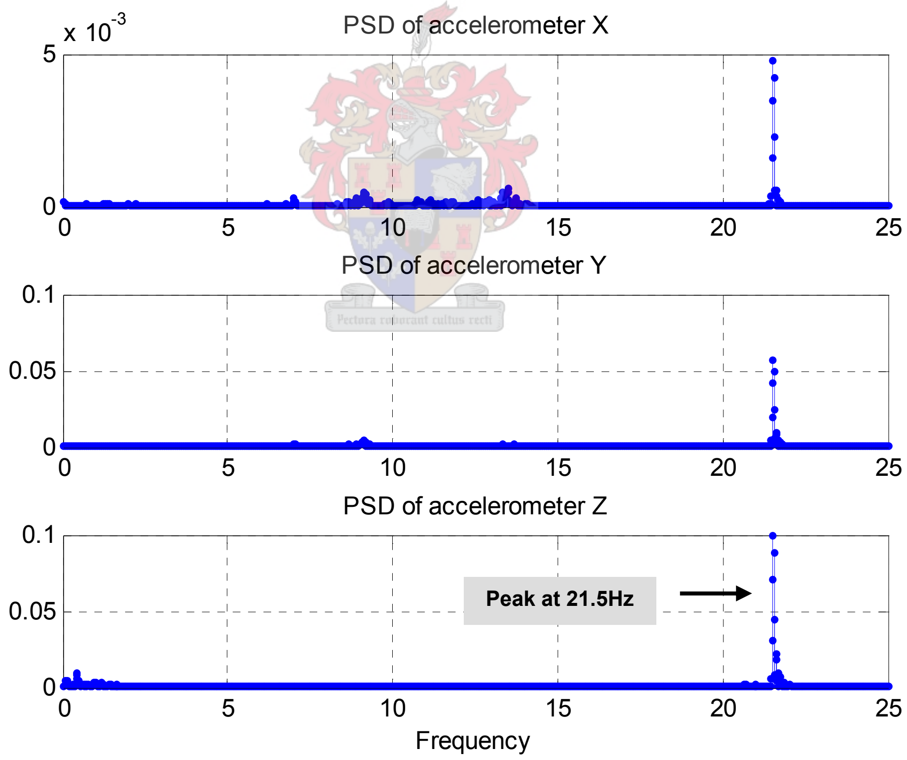


Figure 6-7: PSD of accelerometer measurements showing (aliased) rotor vibration[18].

[18] The units are omitted for the graphs since they are only used to identify the vibration peaks. The true units are [$g^2.Hz^{-1}$]

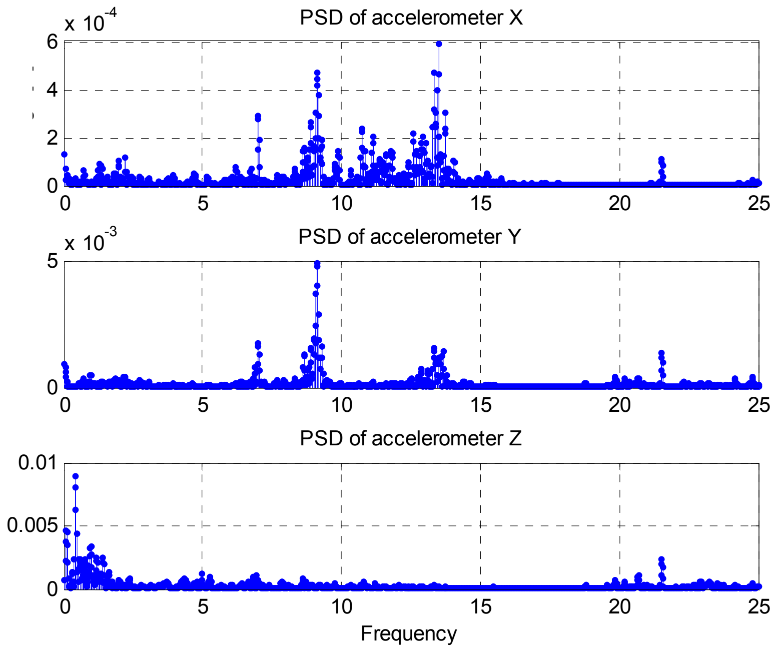


Figure 6-8: PSD of accelerometer measurements with rotor vibration removed[18].

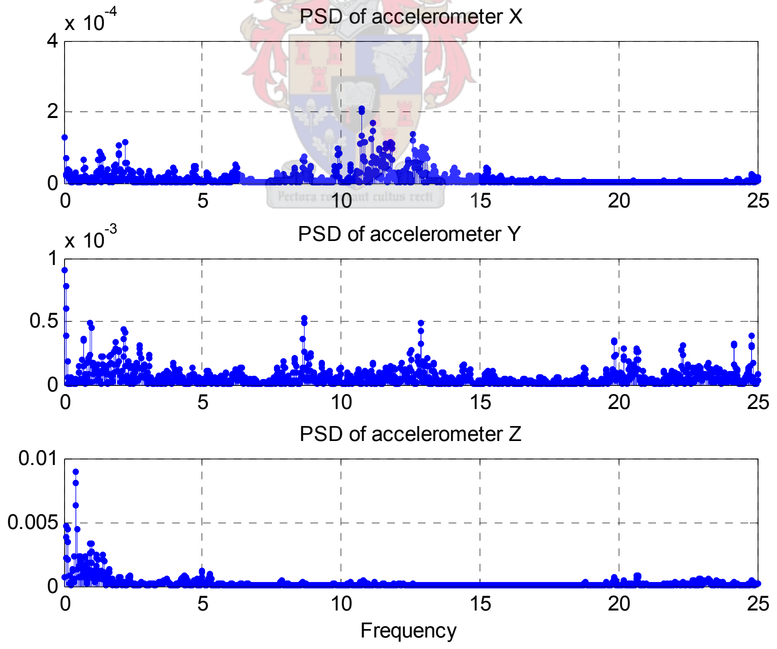
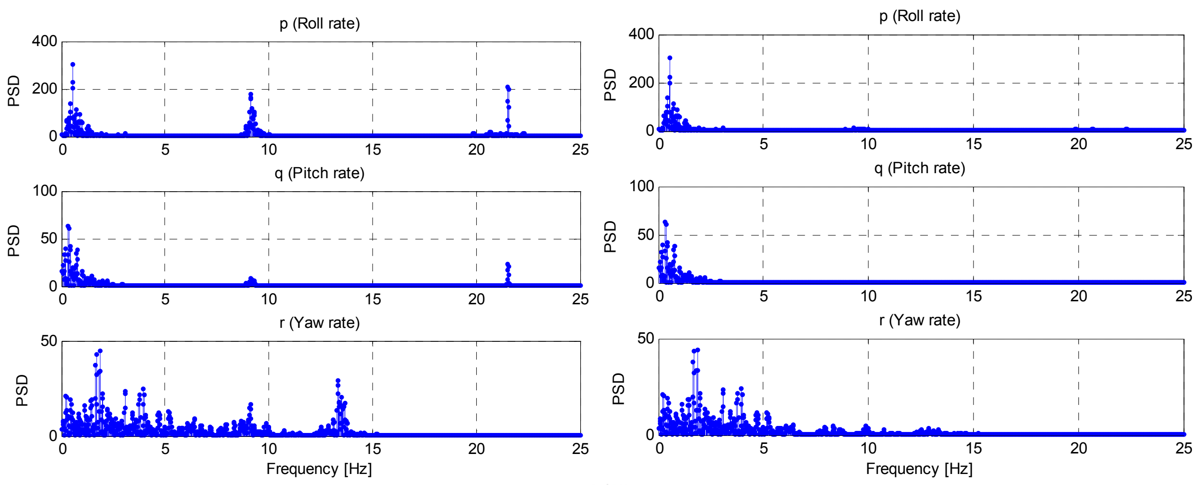


Figure 6-9: Final PSD of accelerometer with main vibration peaks removed.

The procedure can be followed for the rotation rates measured by the rate gyroscopes. The vibrations are more complex, with more resonant peaks showing on the different axes, seen in

Figure 6-10a. The engine vibration is still visible, but not the dominant effect. The 9.1 Hz vibration is also still present, along with the 13.5 Hz peak around the yaw axis. Therefore the same filters where applied as for the accelerometers. However, lower frequencies tend to dominate.



a) Unfiltered rotation rates b) Notch filters applied

Figure 6-10: Rate gyroscope vibration analysis.

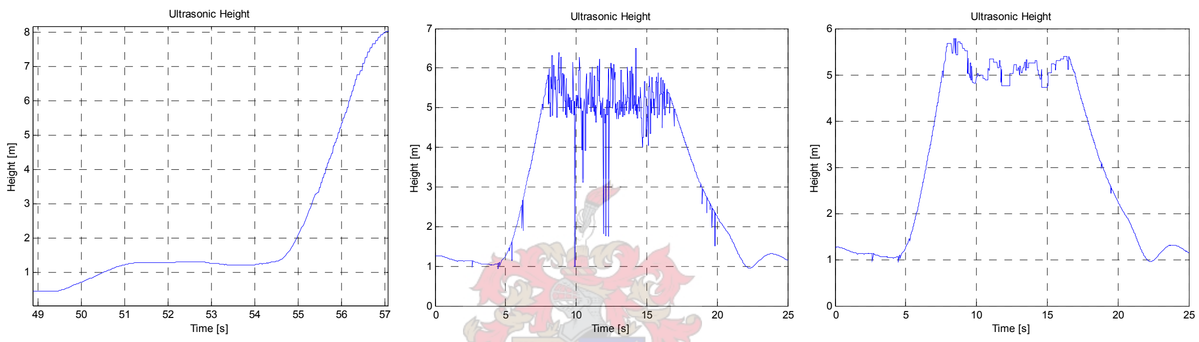
Gavrilets, *et al.* [14] found that the MIT X-Cell had two low frequency modes, "*associated with pendulum-like motion of the helicopter body in pitch and roll around rotor main axis*". The MIT frequencies where 2.7 Hz and 3.1 Hz, but the ESL helicopter exhibits even lower frequencies, due to the higher inertia [14]. Gavrilets continue by stating that the high gain of the yaw rate stability system (gyro) causes more resonances due to airframe and servo pushrod flexibility.

Notch filters add very little phase to the regions outside the stop-band. Although a low-pass filter could filter out all the higher frequency peaks identified, this filter would introduce significant phase lag to the signals. Notch filters have the disadvantage that they will also attenuate relevant information inside the stop-band. Care should be taken when they are applied.

It is clear from the discussion above that vibrations are present in the sensor data, and the data must be processed before it can be interpreted. While some vibrations can be identified, other resonant peaks have no clear source, and their origin would require further analysis.

6.4.2 Altitude Step

The data from an altitude step was analyzed to assess the avionics further. Due to an error made by the author, the anti-vibration mounting of the ultra-sonic sensor was adjusted in such a way that the sensor itself vibrated heavily during the test flight. This caused severe noise of the altitude data. Figure 6-11a shows data from a previous flight test where a positive altitude step was applied and where the ultra-sonic sensor data shows very little noise. Unfortunately, on this occasion the GPS module was disconnected.

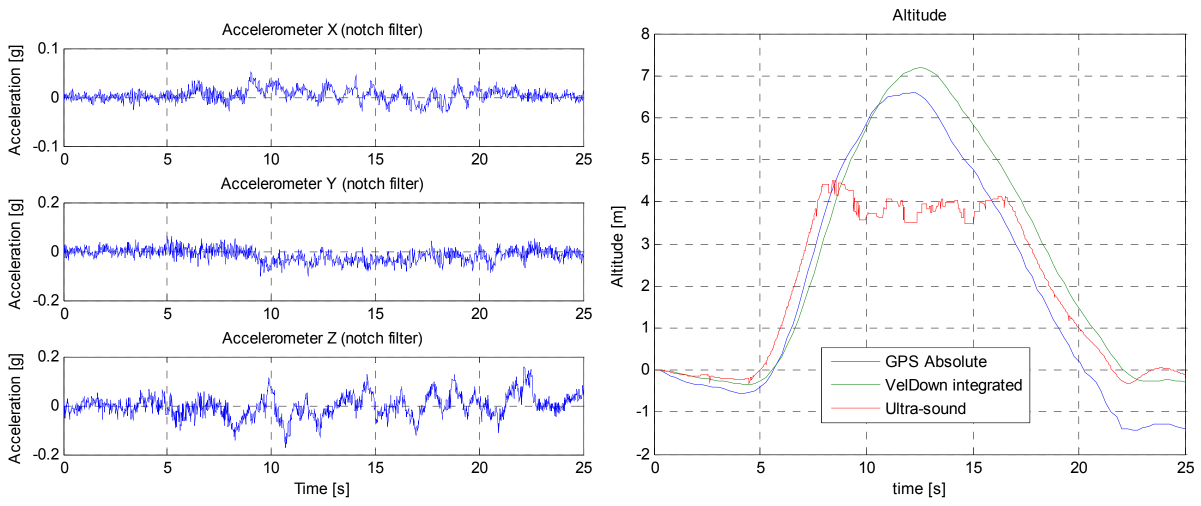


a) Ultra-sonic functioning normally b) Noise due to mounting error c) Noise gate algorithm applied

Figure 6-11: Ultra-sonic sensor readings due to insufficient anti-vibration mounting.

The plot shown in Figure 6-11b shows the noise of the ultra-sonic sensor due to the faulty anti-vibration mounting. A simple noise gate algorithm was applied where each sample is compared with the previous one to see if the acceleration is physically possible between time steps. If not, the current value was discarded and substituted with the previous one. A further refinement is added: The current value is compared with a low-pass filtered altitude sequence. This produced the result of Figure 6-11c, and will be used for the rest of this discussion.

Besides the altitude from the ultra-sonic sensor, the accelerations and rates are also relevant. Figure 6-12a plots the accelerations of the helicopter. On the *z*-axis, a small positive increase in acceleration can be seen at t = 4, and consequently the altitude increases gradually in Figure 6-12b. The GPS data is also plotted (the GPS provides absolute altitude and the vertical velocity component separately), with the zero position taken as the start of the manoeuvre. The GPS shows significant drift when compared to the relatively accurate ultra-sonic sensor, along with latency of approximately 500 ms, initially. As the altitude value drifts, an offset is introduced in the absolute altitude.



a) Unfiltered rotation rates b) Notch filters applied

Figure 6-12: Acceleration and altitude data for an altitude step.

Lastly, the angular information of Figure 6-13 indicates the corrections applied by the pilot in roll and pitch to maintain hover, while the tail drifted slowly during the manoeuvre.

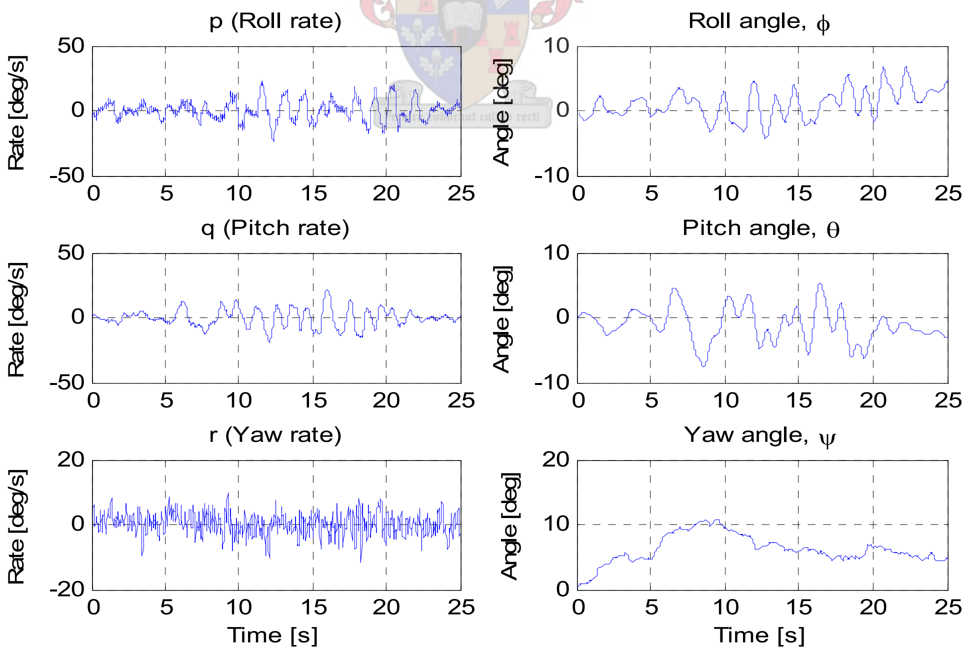
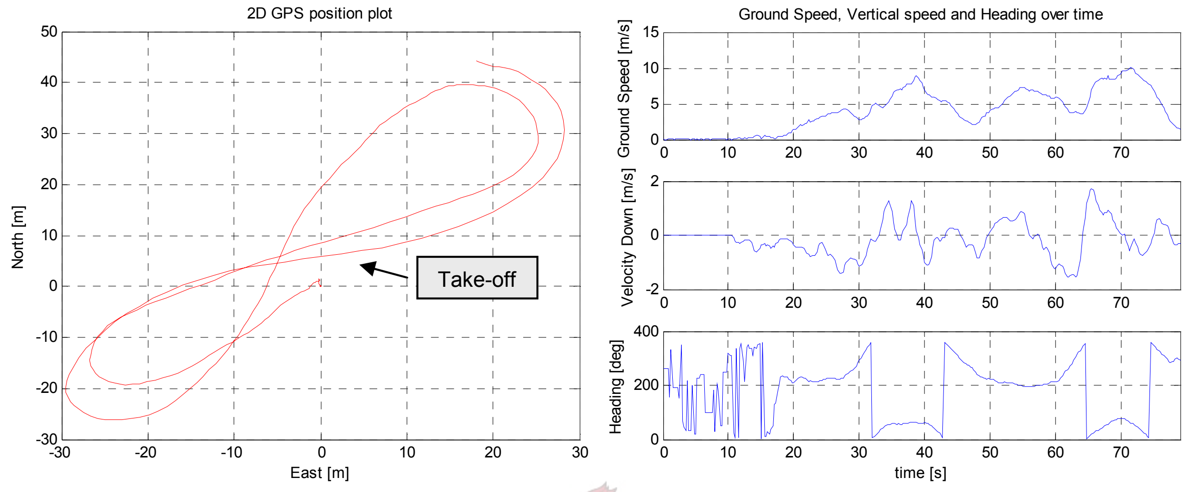


Figure 6-13: Angular rates and body angles during an altitude step.

6.4.3 Figure Eight

A number of figure eight patterns were flown to test the GPS, shown in Figure 6-14. After take-off at about t = 10 seconds, the helicopter hovered for a while and then the manoeuvre started.



a) Top view GPS path (integrated GPS velocity) b) Ground speed, vertical speed and heading

Figure 6-14: GPS path data for figure eight flight.

During the initial period, the ground speed remained low and the track direction (heading) varied substantially. As ground speed increased, the track direction settled. The 2D path is produced by integrating the GPS ground speed. The GPS performs as expected and the data will be useful during forward flight experiments.

6.5 Summary

This chapter attempted to validate the design of the avionics through various test procedures. Although it would be some time before all test scenarios are encountered, the initial testing demonstrated the capability and potential of the avionics. As with all new designs, there are improvements that could be implemented, but already the avionics performed admirably.

The low-cost sensors can only be fully evaluated ounce a control system is implement, but results look promising. The CANsens interface provided a sufficient and versatile interface, and low quantization noise.

Vibration seems to be the biggest problem, and further study is essential to improve the mechanical mounting and damping strategies employed. This ultimately falls in the domain of the mechanical engineer, but some solutions where already offered here, by means of digital notch filters, similar to those initially used by the MIT team.

Chapter 7 Conclusions

This document outlined the development of a low-cost avionics system for a small helicopter to be used as a platform for unmanned vehicle research. The selection of a suitable vehicle, design choices for the onboard hardware and overall architecture was presented, followed by the measurement of the system's performance to verify the hardware design. A simulation environment was also evaluated as a tool to aid in the platform's future use.

7.1 Overview

Based on preceding work done by Carstens [4] on an electrically powered helicopter, a more powerful methanol engined helicopter was acquired to continue research in the RUAV field. The specific helicopter chosen was the Miniature Aircraft X-Cell, used by Gavrilets *et al.* [13], [14] at MIT. Choosing a vehicle that has achieved success elsewhere proved a great advantage, since this provided this project with a clear guideline for design and expected characteristics.

The increased payload capacity of the X-Cell, compared to the previous electric helicopter of Carstens, meant that a more comprehensive avionics package could be designed. An onboard computer, in the form of a PC/104+ Pentium III based system, provides the necessary processing power for the design. An interface board facilitates communication between the OBC and the CAN bus. The bus architecture allows the future extension of the system without requiring any additional interface hardware. This implies that any number of sensors or actuators can be connected to the CAN bus, greatly enhancing the list of possible applications of the system. A CAN actuator board interfaces with a standard RC receiver and up to 16 RC servos.

The author spent considerable time developing a generic sensor node to digitize common, low-cost analogue sensors, and provide an interface to the CAN bus. This endeavour resulted in the CANsens node, a device with 8 A/D channels at 16 bit resolution, and an additional 4 auxiliary channels with 12 bit resolution. The node employs noise reduction techniques to improve resolution and offers a wide variety of peripheral devices, including a serial communication port, configurable digital I/O lines and hardware timing functions. At the time of this writing, CANsens is being used by two other postgraduate students, and one technology demonstrator project.

Linux was chosen as the operating system for the OBC, partly because of prior experience of the author, but mostly because the system design did not require a real-time operating system scheme, as often employed in similar projects. A standard operating system can draw on a wider knowledge base, and has access to more applications. These are attractive qualities for a research platform. The standard Linux kernel source was stripped down to form a smaller embedded version, called *linobc*. The result is a small and effective system, occupying 1.2 MB of disk space, while still providing basic features like TCP/IP networking, a standard file-system and shell environment.

An initial framework for the control software was developed. This is the UCA that gathers and processes sensor data, responds to ground-station commands, transmits status information to the ground, accesses the GPS, stores all data at every time step and would ultimately execute the control code. The timing statistics kept by this application showed that the architecture currently performs well within the timing constraints. The data logging capability provides easy access to the flight data, with all data stored onboard the avionics.

The avionics was mounted to the helicopter, with some modifications to the skids of the vehicle. An off-the-self vibration damping system was employed to isolate excessive vibration caused by the main rotor blades, and other modes of the helicopter. Numerous flight tests were performed and the gathered data was analyzed to gauge the performance of the vibration dampers and the sensors.

The research platform does not, however, only comprise the avionics and helicopter. A dynamic, non-linear simulation of the X-Cell, implemented in MATLAB/Simulink, was assessed. The model provides many insights into the dynamics of the helicopter - the position and orientation of the vehicle can be displayed in a 3D graphical environment, providing further understanding beyond standard, two-dimensional plots. A simple controller is included to serve as the basis for further control system study.

7.2 Improvements and Future Work

This project established a working platform to test RUAV research. The way forward would be to improve upon the work done here, implement control strategies for autonomous hover, and beyond. A discussion of the recommended work that could be done follows below. These are issues that where identified during the course of the work:

- **Vibration** noise is the main area for further work on the platform itself. Section 6.4.1 identified various resonant peaks that should be properly mechanically attenuated. Any digital or analogue filter would impose phase lag to the sensor data. Even though a notch filter adds very little phase outside the stop band, a good mechanical system would reduce or eliminate the need for such filters. This work would be more suited to the domain of the mechanical engineer.
- **Mounting** of the avionics and sensors could be improved. The extended skid design was a first attempt. Although sufficient, more compact designs are possible. The housing of the servo interface, mounted on the nose, should also be modified. No clear verification could be obtained, but it seemed that vibration of this plastic housing coupled with the skids to induce severe vibration. The problem was solved by tightening the screws of the housing, but it would be advisable to move the housing altogether. The MIT helicopter had lower skids and a larger, flat, avionics box (see Figure 1-3a).
- The **UCA** could be extended to provide better integration and portability. The framework itself was already being improved at the time of this writing, but more work could be done, perhaps to create an object orientated version that would aid in the readability of the source code.
- **Communication** with the avionics would be greatly improved if the RF link was augmented by a wireless network adapter. The avionics has a built-in LAN adapter that could be used for this purpose, but perhaps a simpler method would be to add a USB

Bluetooth dongle. These small and lightweight devices can communicate up to 100 m and would aid greatly with the retrieval of the sensor logs, making the entire process completely wireless. The current Linux kernel fully supports the Bluetooth standard, and the feature could simply be added to *linobc*.

- **Simulation** of the X-Cell model could be extended to interface with a hardware-in-the-loop (HIL) simulator. This device would allow the avionics to run control code and effectively *fly* the simulation. The CAN architecture greatly simplifies the task, since the HIL device would just be another device connected to the bus. Such a device already exists in the ESL, and testing it along with the simulation would be next the step.
- **Vision based navigation** may be an interesting project to test on the platform. The OBC has a fair amount of processing power that could be utilized by a pattern recognition algorithm to estimate vehicle states [29].

It is the author's hope that the work presented here leads to many other projects that build on it and that it may serve as an educational tool for many students to follow.

Bibliography

[1] AMIDI, O., "*An Autonomous Vision-Guided Helicopter*" Ph.D Thesis, Carnegie Mellon University, 1996.

[2] BOSSE, M.C., "*A Vision Augmented Navigation System for an Autonomous Helicopter*" M.Sc Thesis, Boston University, 1997.

[3] BROWNE, K.R.J., "*The Instrumentation and Initial Analysis of the Short-term Control and Stability Derivatives of an ASK-13 Glider*", M.Sc Thesis, University of Stellenbosch, April 2004.

[4] CARSTENS, N., "*Development of a Low-Cost, Low-Weight Flight Control System for an Electrically Powered Model Helicopter*" M.Sc Thesis, University of Stellenbosch, April 2005.

[5] CHRISTENSEN, C. M., "*The Innovator's Dilemma*", Harvard Business School Press, 1997.

[6] COYLE, S., "*The Art and Science of Flying Helicopters*", Iowa State University Press, 1996.

[7] DITTRICH, J.S., "*Design and Integration of an Unmanned Aerial Vehicle Navigation System*" M.Sc Thesis, Georgia Institute of Technology, May 2002.

[8] DVORAK, J.C., "The Myth of Disruptive Technology", www.pcmag.com, 17 August 2004.

[9] EURO UVS, "*UAVs – A Vision of the Future*", EURO UVS Publication, 2003, p. 118.

[10] EURO UVS, "*UAVs – A Vision of the Future*", EURO UVS Publication, 2003, p. 94.

[11] FRANKLIN, G.F., POWELL, J.D. and WORKMAN, M., "*Digital Control of Dynamic Systems – 3rd Edition*", Addison Wesley, 1998.

[12] GAVRILETS, V., MARTINOS, I., METTLER, B., FERON, E., "*Flight Test and Simulation Results for an Autonomous Aerobatic Helicopter*", MIT Cambridge, 2002.

[13] GAVRILETS, V., METTLER, B., FERON, E., "*Dynamic Model for X–Cell 60 Helicopter in Low Advance Ratio Flight.*", MIT Cambridge, December 2002.

[14] GAVRILETS, V., SHTERENBERG, A., DAHLEH, M. A., FERON, E., "*Avionics System For A Small Unmanned Helicopter Performing Aggressive Maneuvers*", MIT Cambridge,

2000.

[15] HERWITZ, S.R., JOHNSON, L.F., ARVESAN, J.C., "*Precision Agriculture as a Commercial Application for Solar-Powered Unmanned Aerial Vehicles*", AIAA Inc., May 2002.

[16] HERWITZ, S.R., JOHNSON, L.F., DUNAGAN, S., "*Collection of Ultra High Spatial and Spectral Resolution Image Data over California Vineyards with a Small UAV*", Symposium on Remote Sensing of Environment, October 2003.

[17] HOROWITZ, P. and HILL, W., "*The Art of Electronics – 2nd Edition*", Cambridge University Press, 1995.

[18] KAHN A.D., "*The Design and Development of a Modular Avionics System*" M.Sc Thesis, Georgia Institute of Technology, April 2001.

[19] LAMARTIN, G.F. "*Testimony of Dr. Glenn F. Lamartin*", House Committee on Armed Services, March 2005.

[20] LOEWENSTEIN, E.B., "*Reducing the Effects of Noise in a Data Acquisition System by Averaging*" Application Note 152, National Instruments Corporation, April 2000.

[21] MANOLAKIS, D.G. and PROAKIS, J.G., "*Digital Signal Processing, 3rd Edition*", Prentice Hall, 1996.

[22] METTLER, B., TISCHLER, M. B., KANADE, T., "*System Identification Modeling of a Small-Scale Unmanned Rotorcraft for Flight Control Design*", American Helicopter Society Journal, 2002.

[23] METTLER, B., TISCHLER, M.B., KANADE, T., "*System Identification of Small-Size Unmanned Helicopter Dynamics*", 55th Forum of the American Helicopter Society, Montreal, Canada, May 1999.

[24] PC/104 CONSORTIUM, "*PC/104 Embedded PC Modules*", http://www.pc104.org, October 2005.

[25] PEDDLE, I.K., "*Autonomous Flight of a Model Aircraft*" M.Sc Thesis, University of Stellenbosch, December 2004.

[26] PIPITONE, F., KAMGAR-PARSI, B. and HARTLEY, R.L., "*Three Dimensional Computer Vision for Micro Air Vehicles*", Navy Center for Applied Research in Artificial Intelligence, Washington, 2002.

[27] PRETOLANI, R., SAGGIANI, G.M., THEODORANI, B., "*A Low Cost Unmanned Helicopter Platform for Geophysical and Environmental Applications*", University of

Bologna, October 2004.

[28] ROBERTS, J.M. and CORKE, P.I., BUSKEY, G., “*Low-Cost Flight Control System for a Small Autonomous Helicopter*”, Australian Conference on Robotics and Automation, Auckland, November 2002.

[29] ROBERTS, J.M., SIKKA, P., OVERS, L., “*System Design for a Small Autonomous Helicopter*”, CSIRO MS&T Australia, October 2000.

[30] SANGHYUK, P., “*Avionics and Control System Development for Mid-Air Rendezvous of Two Unmanned Aerial Vehicles*” Ph.D Thesis, MIT Cambridge, February 2004.

[31] SHIM, H., “*Hierarchical Flight Control System Synthesis for Rotorcraft-based Unmanned Aerial Vehicles*” Ph.D Thesis, University of California – Berkeley, December 2000.

[32] UNMANNED SYSTEMS Vol. 21, no. 3, “*ALTAIR UAV Aloft*”, AUVSI, May/June 2003, p. 8.

[33] UNMANNED SYSTEMS Vol. 21, no. 6, “*Predator Supports USCG in Alaska*”, AUVSI, November/December 2003, p. 10.

[34] UNMANNED VEHICLES ONLINE, “*UAV close surveillance used at Oscars*”, Shephard Group (www.shephard.co.uk), 3 March 2005.

[35] UNMANNED VEHICLES Vol. 8, no. 4., “*From the Top*”, Shephard Group, July/August 2003, p. 33.

[36] VELEZ, C.M., BECERRA, H.M., SANCHEZ, E.N., “*Combining fuzzy and PID control for an unmanned helicopter*”, North American Fuzzy Information Processing Society, June 2005.

[37] VENTER, D. W., “HOPTUS: A Case Study” tech. rep., University of Stellenbosch, 1996.

[38] VENTER, J., “*Development of an experimental tilt-wing VTOL Unmanned Aerial Vehicle*” M.Sc Thesis, University of Stellenbosch, April 2006.

Technical References

[39] AEROCOMM, "*AC4490/AC4486 User's Manual 1.6*", AeroComm, July 2003.

[40] ANALOG DEVICES, "*ADXL103 Datasheet*", Analog Devices Inc., May 2004.

[41] ANALOG DEVICES, "*ADXRS401 Datasheet*", Analog Devices Inc., July 2004.

[42] APEX EMBEDDED SYSTEMS, "*PC/104 Computer Boards, Data Acquisition*", http://www.apexembedded.net, October 2005.

[43] ATMEL, "*ATmega128/ATmega128L Summary*", Atmel Corporation, November 2004.

[44] CAN IN AUTOMATION, "CAN Specification 2.0, Part B", CAN in Automation, Erlangen, 1999.

[45] ELECTRONIC SYSTEMS LABORATORY, "*SUNSAT 2004 Project*", University of Stellenbosch, 2004.

[46] HONEYWELL, "*HMC2003 Datasheet*", Honeywell SSEC, 2000.

[47] HUTCHINSON, "*PaulstraDYN Datasheet*", Hutchinson Worldwide, 2003.

[48] KOKAM, "*SLPB 356495 Lithium Polymer Battery Datasheet*", Kokam Engineering Ltd, Seoul, September 2002.

[49] KONTRON, "*Global Embedded Computer Leader*", http://www.kontron.com, October 2005.

[50] MICROCHIP, "*dsPIC30F Data Sheet - General Purpose and Sensor Families*", Microchip Technology Inc., December 2003.

[51] MICROCHIP, "*PIC18FXX8 Data Sheet*", Microchip Technology Inc., January 2003.

[52] MINIATURE AIRCRAFT USA, SORRENTO, FLORIDA., "*X-Cell Fury Helicopter Kit #115-103 Instruction Manual*", September 2003.

[53] OCTAGON SYSTEMS, "*Welcome to Octagon Systems*", http://www.octagonsystems.com, 2005.

[54] O-NAVI, "*Gyrocube 3A Datasheet*", O-Navi LLC, June 2003.

[55] SensComp, "*Series 600 Datasheet*", SensComp Inc., September 2004.

[56] TEXAS INSTRUMENTS, "*ADS8344*", Texas Instruments Incorporated, April 2003.

[57] TEXAS INSTRUMENTS, "*OPA350, OPA2350, OPA4350*", Texas Instruments Incorporated, February 2004.

[58] TEXAS INSTRUMENTS, "*REF3112 – REF3140*", Texas Instruments Incorporated, February 2004.

[59] TEXAS INSTRUMENTS, "*REG104*", Texas Instruments Incorporated, February 2004.

[60] UBLOX, "*RCB-LJ GPS Receiver Board Datasheet*", u-Blox AG, November 2003.

[61] U-DYNAMICS, "AeroSim Blockset version 1.1 – User's Manual", Unmanned Dynamics LLC, 2003.

Appendix A Project Photographs

A-1. Avionics

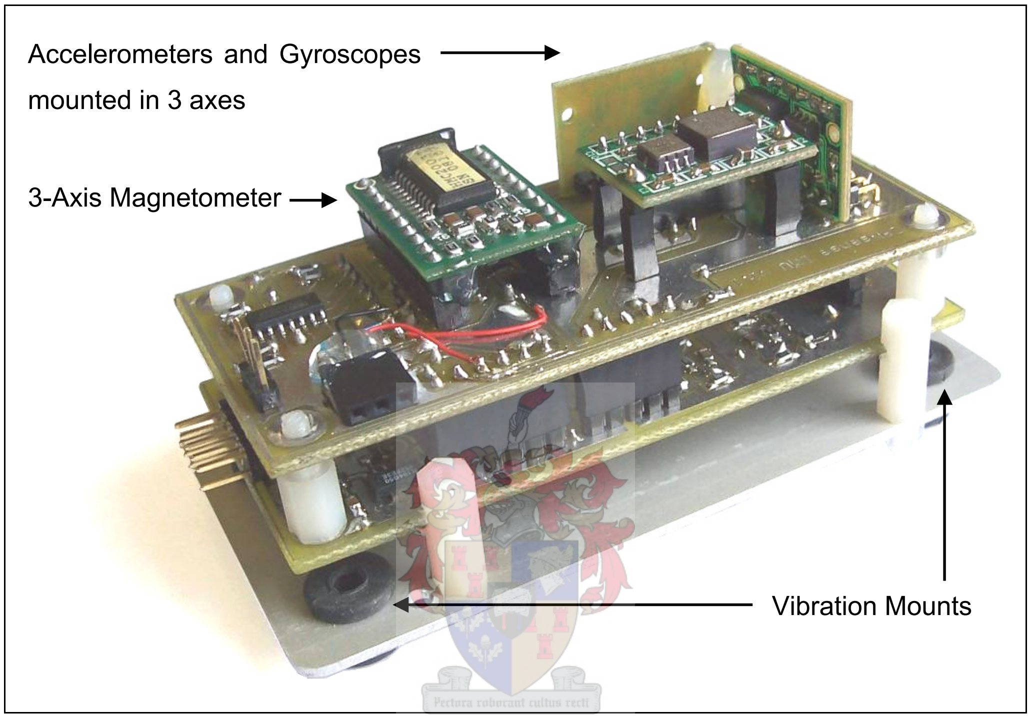


Figure A-1: CANsens IMU.

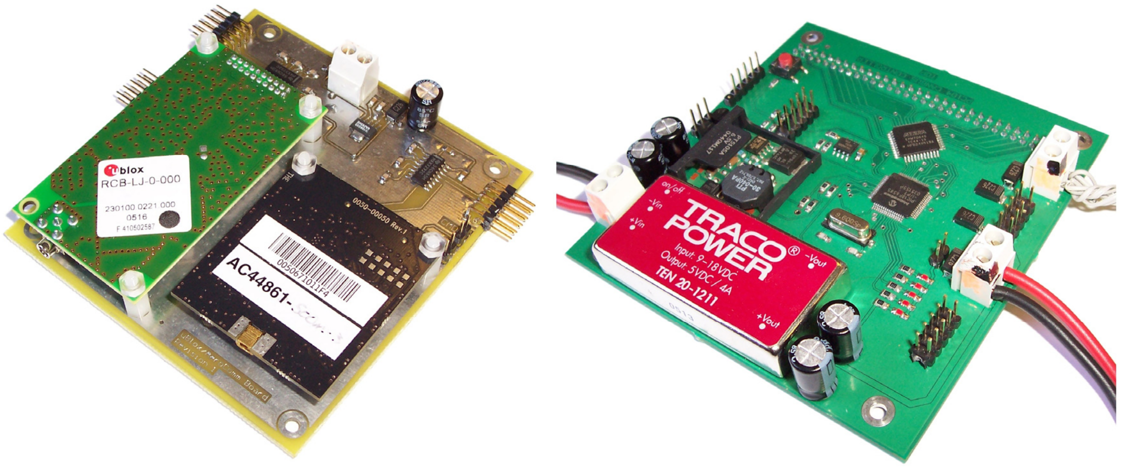


Figure A-2: PC/104 sized board for GPS & RF modules (left) and PC/104CAN controller (right)

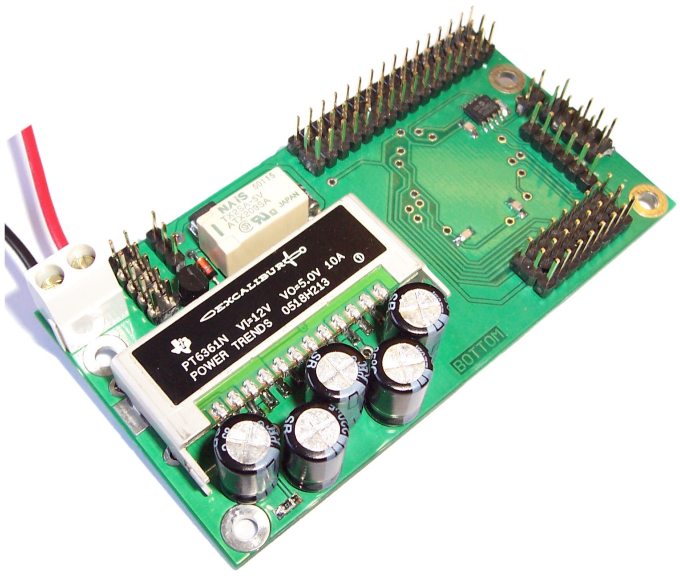


Figure A-3: CAN pilot/servo interface.

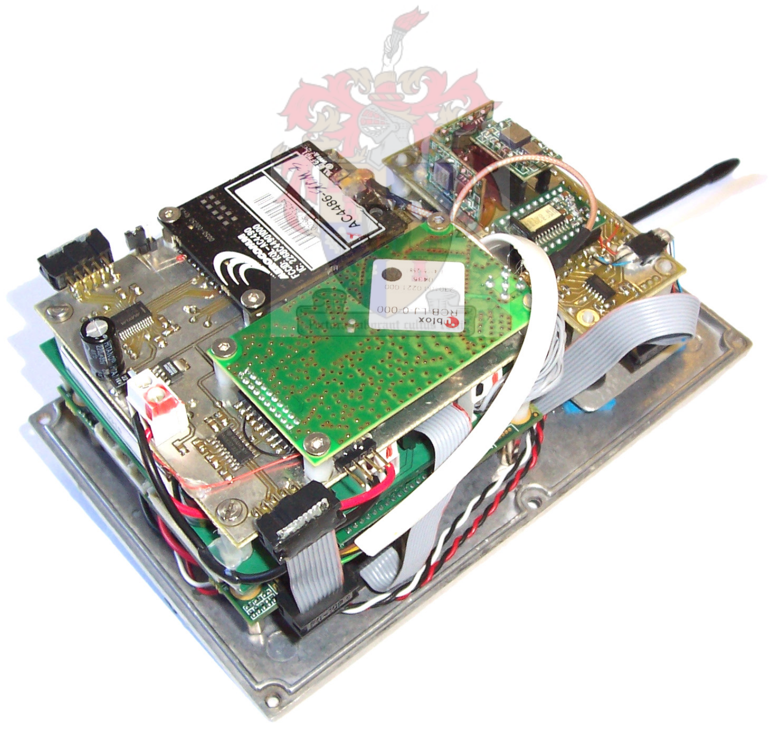


Figure A-4: PC/104 stack and IMU mount on avionics enclosure lid.

A-2. Helicopter Modifications

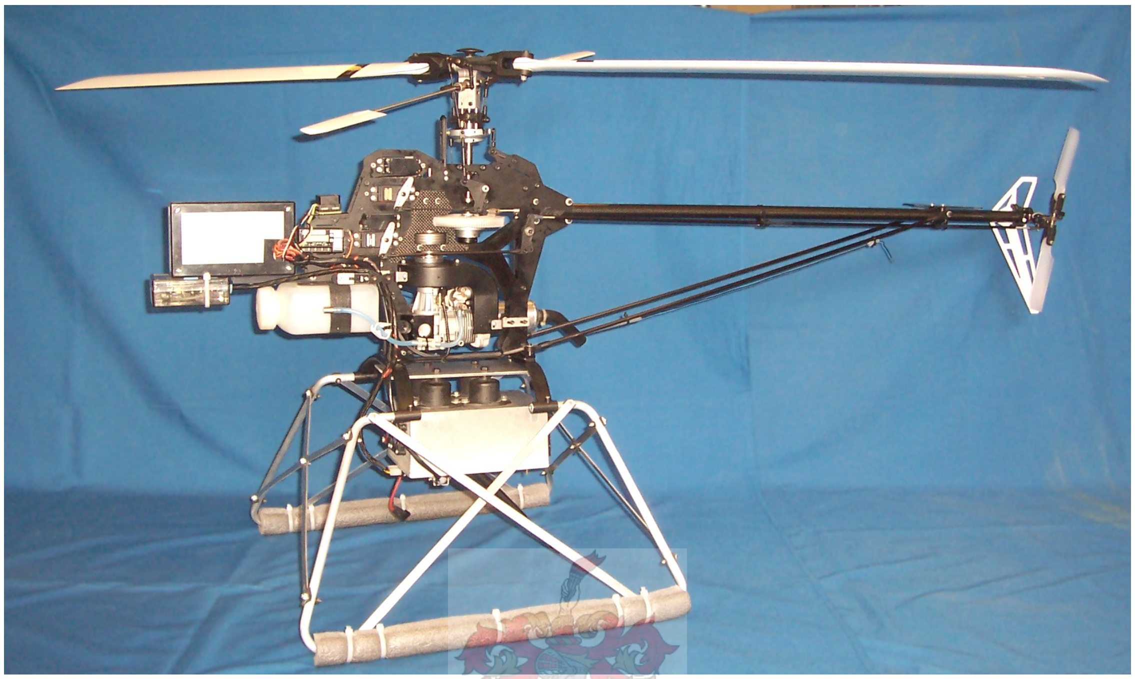
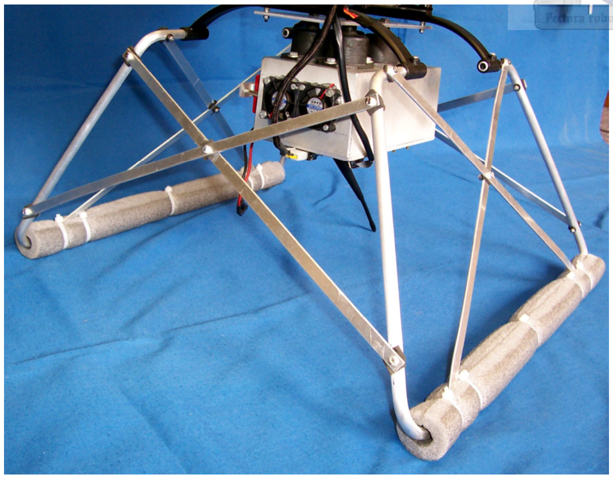
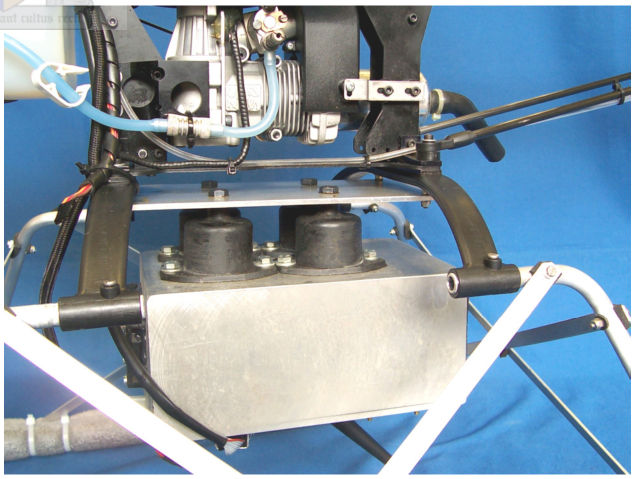


Figure A-5: Modified X–Cell 60 with avionics and extended skids fitted. White vinyl was added to the black nose and tail to make it easier to fly for the pilot. The canopy was removed since it does not affect hover performance and only adds weight.



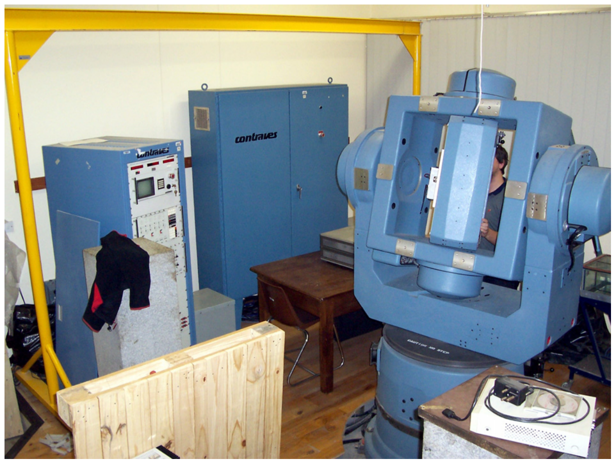
The extended skids were made from aluminum tubing and strips. The strips isolated by foam/plastic washers to avoid metal rubbing on metal, causing RF noise.



An aluminum plate was attached to the bottom of the X–Cell to form a convenient surface to fix the vibration mountings.

Figure A-6: Extended skids and vibration mounting.

A-3. Calibration with rate table


Contravez 3-axis rate table room. The rate table has some electrical problems, but can still be moved manually.

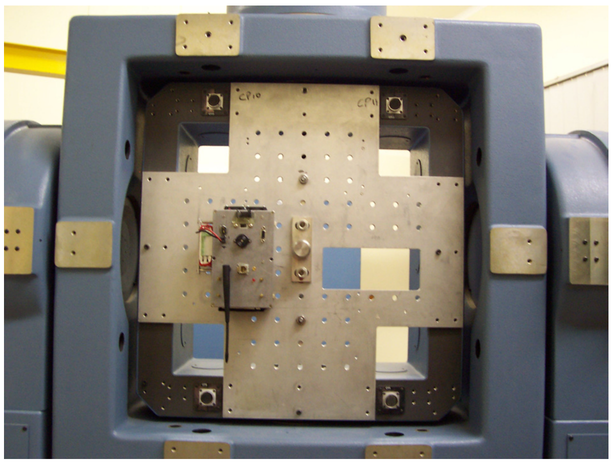

Avionics mounted to rate-table. An ordinary construction level was used to position it correctly.

Figure A-7: Rate-table calibration set-up.

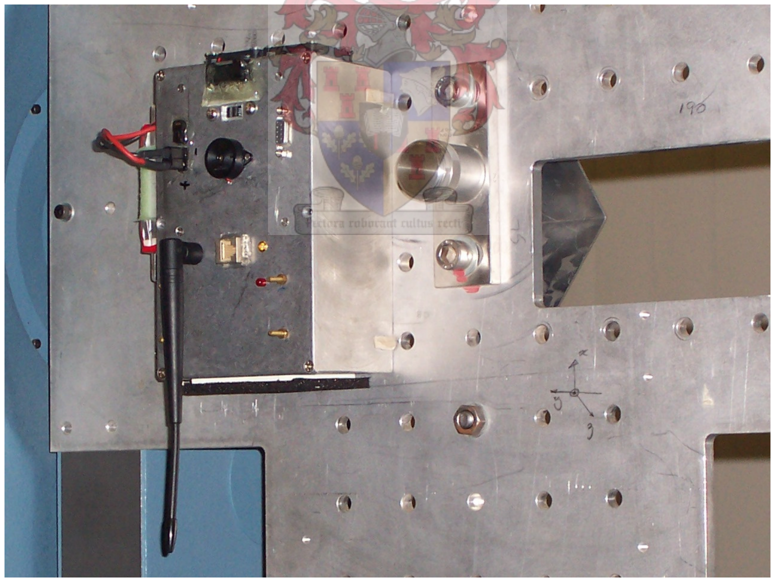


Figure A-8: Close-up of mounted avionics. Note axis marking at bottom right.

A-4. Inertia measurement

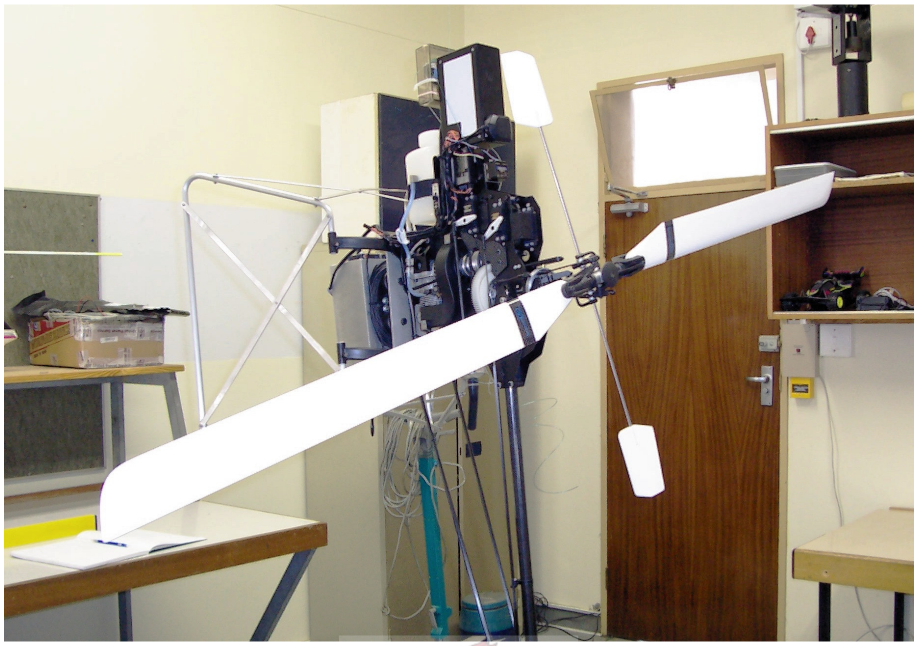


Figure A-9: The X–Cell during measurement of roll inertia. The helicopter was cumbersome to suspend for the torsional pendulum test.

A-5. Flight Tests

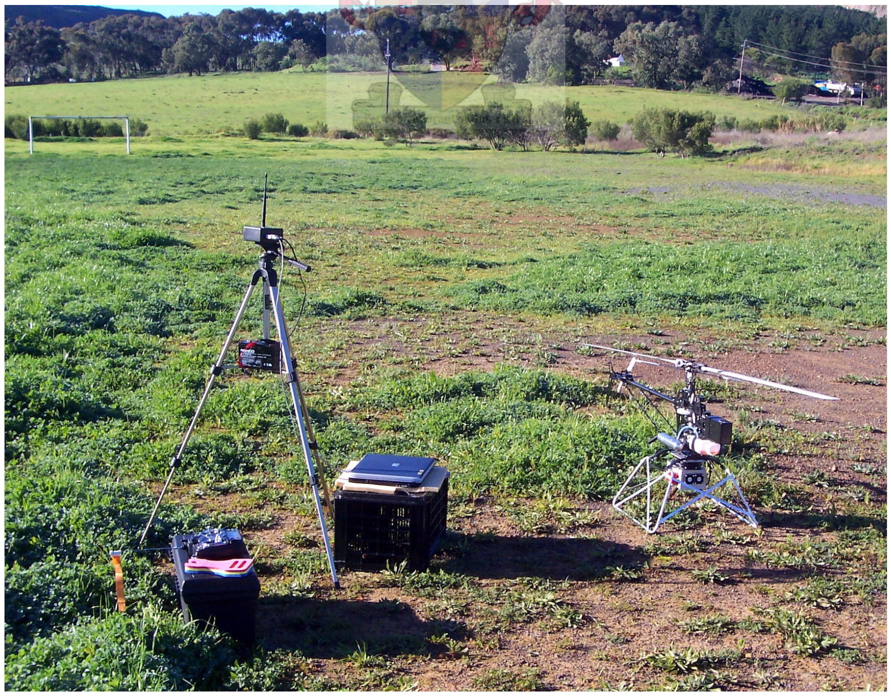


Figure A-10: Flight test equipment: Flight box and transmitter (front, left), AeroComm RF module mounted on a standard camera tripod, and Laptop for ground-station software.

Appendix B CAN Protocol

The various addresses, functions and packets of the nodes are described.

B-1. General Description

The protocol uses and extended, 29 bit, CAN 2.0B identifier. The format of the packet is listed below.

CAN Field	Field Name		No. Bits
CAN Identifier (ID)	FID	Main Type ID	4
		Request/Reply	1
		Subtype	8
	Source Address		8
	Destination Address		8
Data Bytes	Data Byte 0		8
	Data Byte 1		8
	Data Byte 2		8
	Data Byte 3		8
	Data Byte 4		8
	Data Byte 5		8
	Data Byte 6		8
	Data Byte 7		8

Figure B-1: CAN packet layout

B-2. Node Particulars

The table below contains the broad outline of the implemented CAN protocol. All nodes, except the Master node will send one packet with FID = 0x1400 when power is applied. The packet will contain zero data bytes and will be addressed to the Master node.

Message Type	FID	Comments
Time synchronization (SYNC)	`0x0101`	CANsens will store last measurement and send it immediately. Output nodes will write last values.

Real-time Control	`0x02--`
	`0x03--`
		Request (Master to Servo)

		Response (Servo to Master)
CAN Telemetry (TLM)	`0x0A--`	Request (usually when data was lost after SYNC)
	`0x0B--`	Response (to Master node)
Application specific	`0x14--`	Boot notification to Master

B-2.1. PC/104CAN Controller (Master)

- Address = `0x01`
- SYNC packet ID is `0x010101FF`
- Send SYNC every 20ms to address ALL (`0xFF`)
- Capable of sending any CAN packet from the OBC

B-2.2. CANsens IMU

- Address = `0x02`
- Will reply (TLM) with last measurement(s) if SYNC is received from Master node only
- Subtype `0x80` contains 16 bit channel 0 to 3 (ID = `0x0B800201`)
- Subtype `0x81` contains 16 bit channel 4 to 7
- Subtype `0x82` contains 10 bit channels (left aligned)
    - 16 bits per channel without ultrasonic sensor
    - 12 bits per channel with ultrasonic sensor:

Byte	0	1		2	3	4		5	6	7
Data	Lo0	Hi0	Hi1	Lo1	Lo2	Hi2	Hi3	Lo3	US_hi	US_lo

- A packet received with ID = 0x0B8F0102 will (D0 is first data byte of packet):
    - D0 = `0x10`: Set magnetometer
    - D0 = `0x20`: Reset magnetometer
    - D0 = `0x30`: Rate Gyro & Accelerometer self-test mode
    - D0 = `0x40`: Rate Gyro & Accelerometer normal mode
    - D0 = `0x50`: System reset.
    - D0 = `0x60`: Ultrasonic sensor enabled
    - D0 = `0x70`: Ultrasonic sensor disabled

B-2.3. Servoboard

- Address = `0x08`
- Main type `0x02`, subtype `0x80` to `0x83` used to set 16 servo channels. Data starts at channel 0, high byte.
- Main type `0x0A` is used to request current servo positions commanded from the PC.
- When `0x0101` is received, D0 is checked for autopilot status. PC commands or Pilots commands are written to servos, depending on pilot master switch.

B-3. Hardware Connector

The 2×5 pin header used on most CAN nodes has the pin assignment shown in Figure B-2. Pin 4 should be remove on the male connector and blocked off on the female connector to avoid connecting a node the wrong way round.

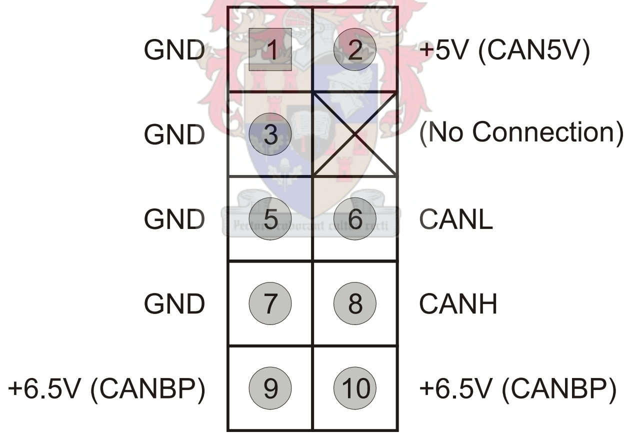


Figure B-2: 10 pin CAN Connector

Appendix C CANsens Technical Details

This appendix documents some technical aspects of the CANsens board not really relevant to the thesis, but useful to anyone working with the actual hardware.

C-1. Basic Setup

C-2. Analog Filter Details

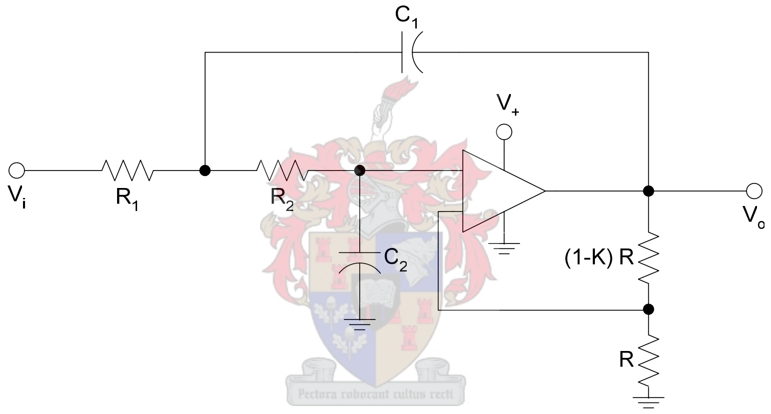


Figure C-1: 2nd Order, Lowpass Butterworth filter

Voltage -3dB point:

$$f_c = \frac{1}{4\pi RC} \quad , \quad R_1 = R_2 = R \quad C_1 = C_2 = C \qquad \textbf{A.1}$$

$$Z_{in}(s) = \frac{R}{1 - \dfrac{1}{R\left(sC + \dfrac{sC + \frac{1}{R}}{1 - sCR} + \dfrac{2}{R}\right)}} \qquad \textbf{A.2}$$

C-3. Circuit diagram

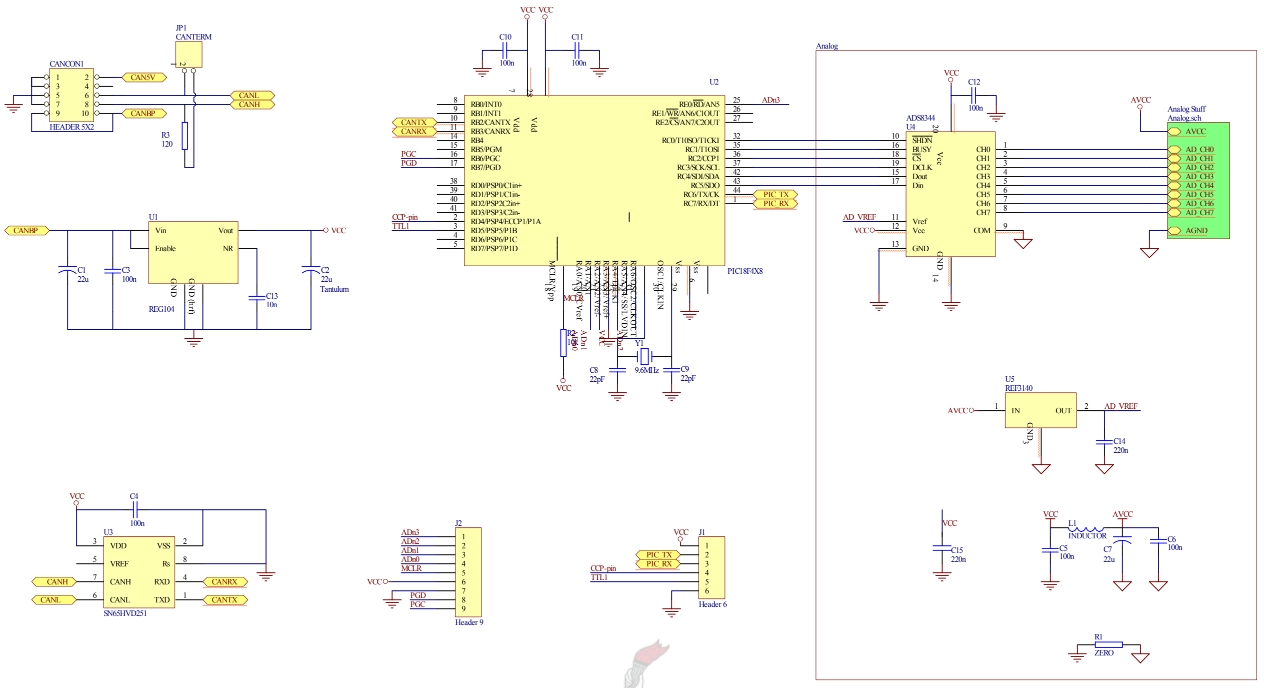


Figure C-2: CANsens schematic - digital side.

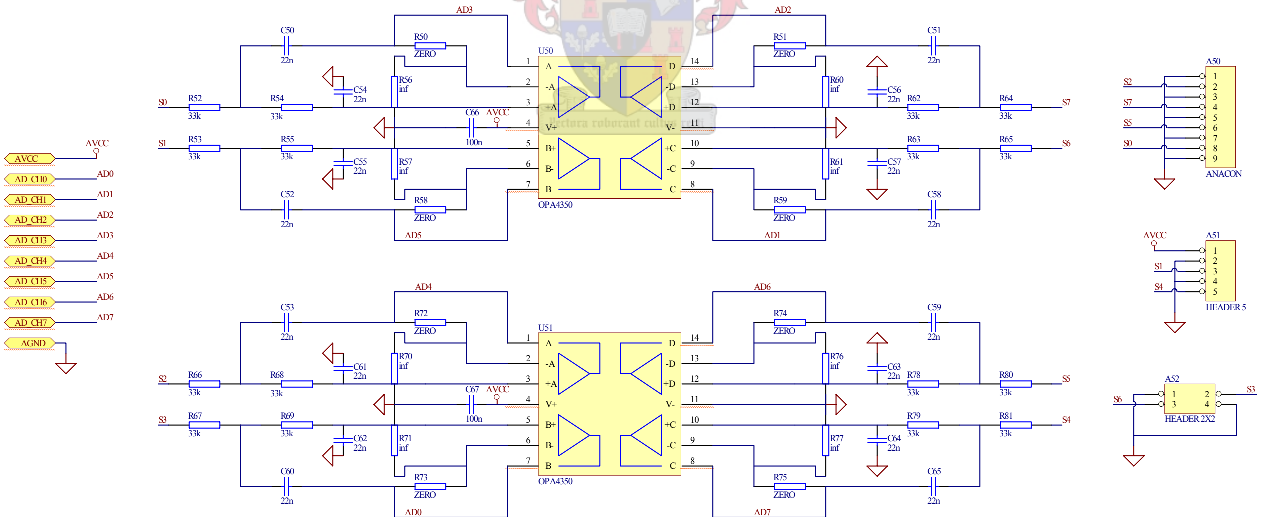


Figure C-3: CANsens schematic. Analogue side.

C-4. PCB Artwork

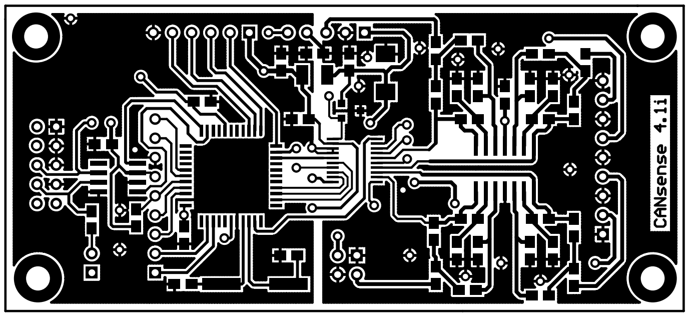


Figure C-4: CANsens PCB top layer

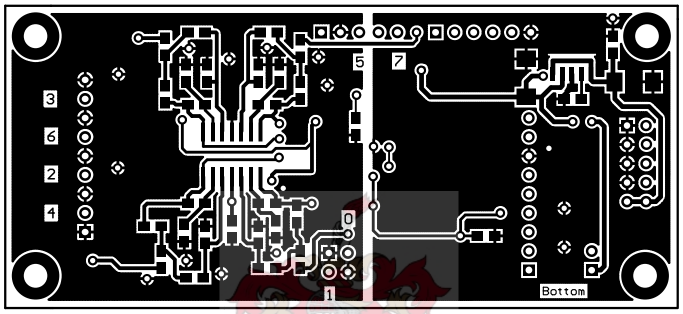


Figure C-5: CANsens PCB bottom layer (mirrored)

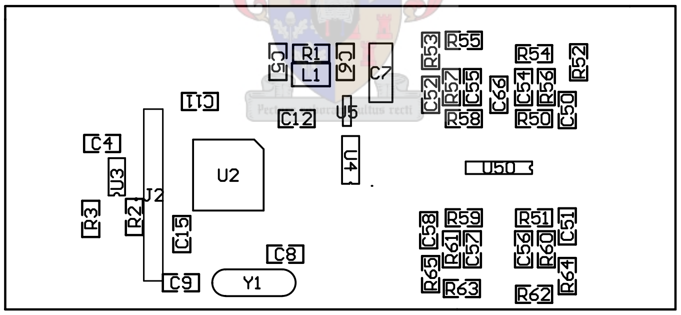


Figure C-6: CANsens PCB top overlay

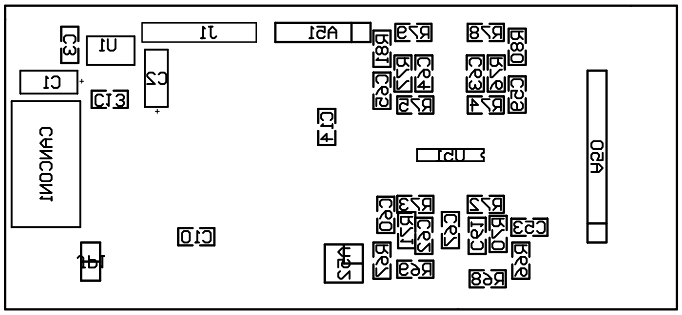


Figure C-7: CANsens PCB bottom overlay

C-5. Bill of Materials

Quantity	Comment	Description	Designator	Footprint	LibRef
9	100n	Capacitor	C3, C4, C5, C6, C10, C11, C12, C66, C67	0805	CAP
1	10k		R2	0805	RES2
1	10n	Capacitor	C13	0805	CAP
1	120		R3	0805	RES2
2	220n	Capacitor	C14, C15	0805	CAP
16	22n	Capacitor	C50, C51, C52, C53, C54, C55, C56, C57, C58, C59, C60, C61, C62, C63, C64, C65	0805	CAP
2	22pF	Capacitor	C8, C9	0805	CAP
3	22u	Capacitor	C1, C2, C7	1210, CASE C	CAPACITOR
16	33k		R52, R53, R54, R55, R62, R63, R64, R65, R66, R67, R68, R69, R78, R79, R80, R81	0805	RES2
1	9.6MHz	Crystal	Y1	X49_SMD_SML	CRYSTAL
1	ADS8344	A/D Converter	U4	PDSO-G20	ADS8344
1	ANACON		A50	HDR9X1	HEADER 9
1	CANTERM		JP1	HDR2X1	HEADER 2
1	HEADER 2X2		A52	HDR2X2	HEADER 2X2
1	HEADER 5		A51	HDR5X1	HEADER 5
1	HEADER 5X2		CANCON1	HEADER_5X2_SML	HEADER 5X2
1	Header 6	Header, 6-Pin	J1	HDR1X6	Header 6
1	Header 9	Header, 9-Pin	J2	HDR1X9	Header 9
1	INDUCTOR		L1	SMT_L	INDUCTOR
8	inf		R56, R57, R60, R61, R70, R71, R76, R77	0805	RES2
2	OPA4350	Quad Op-Amp	U50, U51	SOP14	OPA4350
1	PIC18F4X8	Microcontroller	U2	TQFP44	PIC18F4X8
1	REF3140	4.096V Reference	U5	SO-G3	REF3140
1	REG104	Burr-Brown 1A Regulator	U1	SOT223	REG104
1	SN65HVD251		U3	SO-G8	SN65HVD251
9	ZERO		R1, R50, R51, R58, R59, R72, R73, R74, R75	0805	RES2

C-6. Frequency Measurements

The filters of CANsens were tested by applying a sine wave with DC offset and amplitude 3.5 V (peak-to-peak) to a 16 bit channel. The results are plotted in Figure C-8 and Figure C-9.

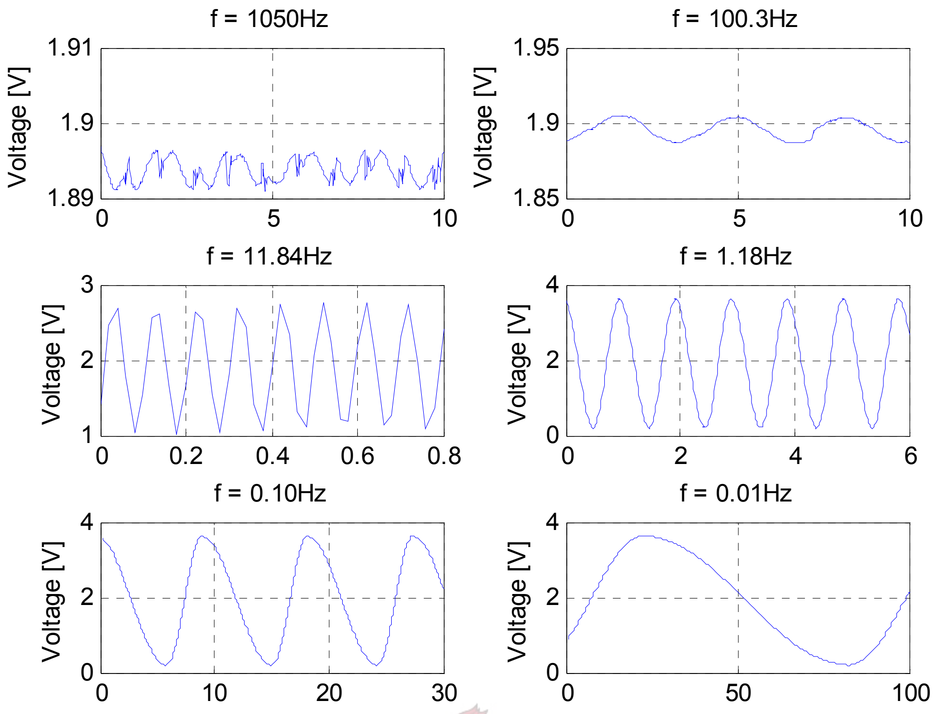


Figure C-8: Time-domain plots of different frequency inputs (time in seconds)[1].

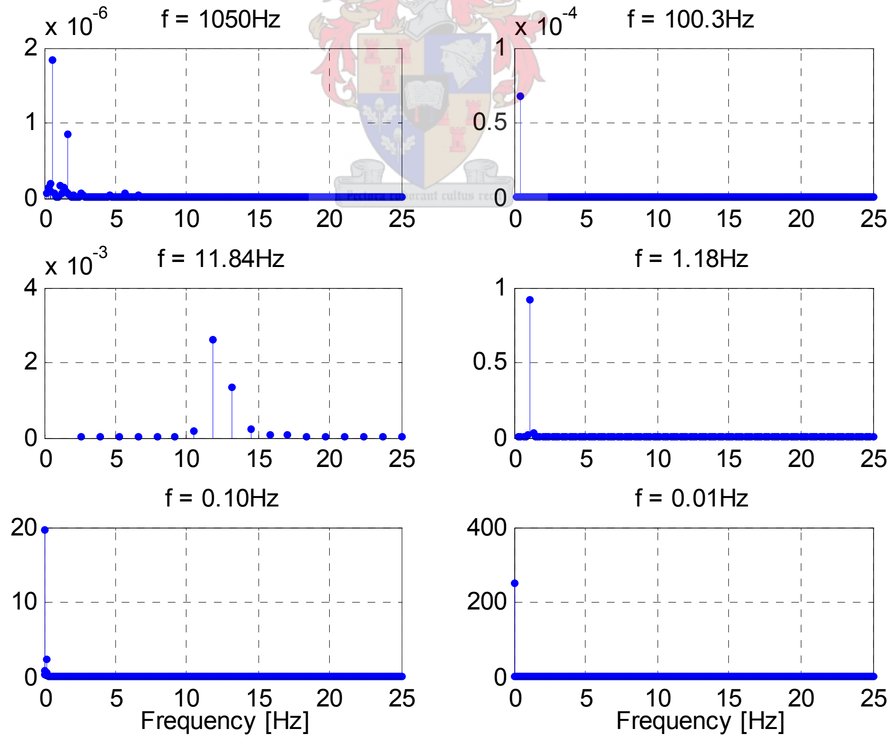


Figure C-9: Power spectral density plots for different frequency inputs.

[1] Distortion of sinus wave due to low quality signal generator.

Appendix D The toolchain for linobc

This section describes all the software and commands needed to install and use the embedded Linux system used on the OBC, called *linobc*. It does not serve as a Linux tutorial and it will be assumed that the reader is familiar with common computer terminology and perhaps some Linux conventions. Note that the information contained here should be considered a starting point. The reader is encouraged to try newer versions of the toolchain to help keep linobc current.

D-1. The Linux Development Platform

The development platform used through-out most of this project was Mandrake Linux 10, currently known as Mandriva Linux[1]. The distributors of Mandriva often focus on stability, rather than cutting edge speed and the newest features. Therefore, newer versions should work fine as a development platform, but no guarantees can be made. Installations that worked without problems were Mandrake 9.2, Mandrake 10.0, Mandrake 10.1 and Mandrake 10.2 (Limited Edition). Should the reader decide to use a different distribution, problems might occur that are beyond the scope of this document.

The chosen installation should have a recent version of *gcc* (the C compiler) and associated libraries (*glibc*, installed by default). Version 3.4.x was current at the time of this writing. An important package that is usually not installed by default is the C static libraries, typically called '*glibc-static-devel-x.y.z*' on Mandriva systems. The static libraries are required for installation and general system use, since no dynamic C libraries are installed on the embedded system. All binaries are therefore compiled static.

D-1.1. Toolchain Organization

[1] Official version at http://www.mandriva.com/, mirror at http://archive.sun.ac.za/ftp/mandrake/official/

Figure D-1 shows the organization of the toolchain directory structure. Important features are mentioned and it should provide a reasonable overview of the function of each directory.

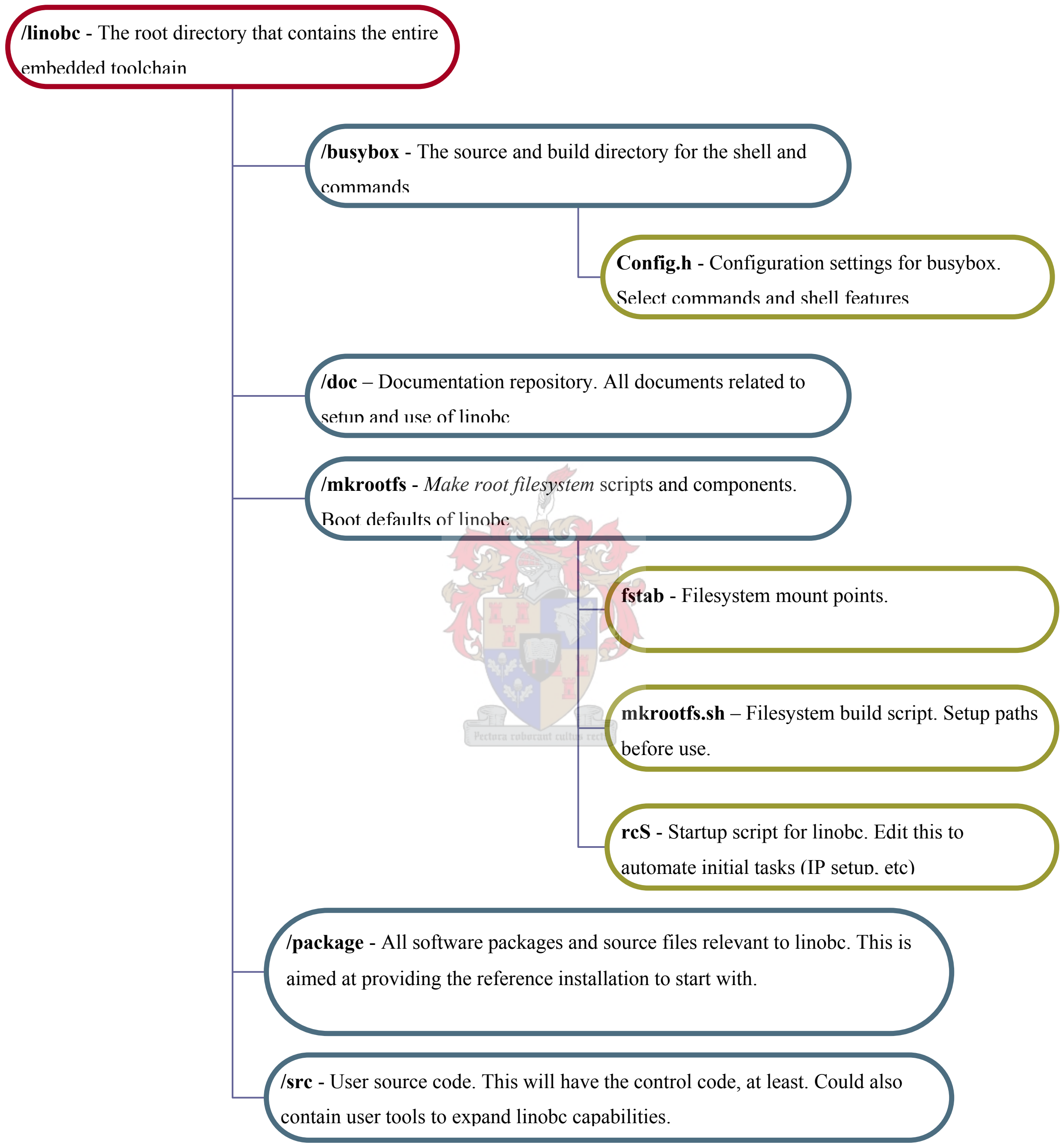

/linobc - The root directory that contains the entire embedded toolchain
/busybox - The source and build directory for the shell and commands
Config.h - Configuration settings for busybox. Select commands and shell features
/doc – Documentation repository. All documents related to setup and use of linobc
/mkrootfs - *Make root filesystem* scripts and components. Boot defaults of linobc
fstab - Filesystem mount points.
mkrootfs.sh – Filesystem build script. Setup paths before use.
rcS - Startup script for linobc. Edit this to automate initial tasks (IP setup, etc)
/package - All software packages and source files relevant to linobc. This is aimed at providing the reference installation to start with.
/src - User source code. This will have the control code, at least. Could also contain user tools to expand linobc capabilities.

Figure D-1: *linobc* Direcotory organization.

D-2. linobc Setup

The first step is to install the Linux system on the Kontron MOPSlcd7 PC104 board. The configuration used was a Pentium III Celeron 300 MHz processor, 64Mb RAM, 32Mb Chipdisk, LAN interface and serial cable on COM 1 of the MOPSlcd7.

Additional software or commands on the development platform should include:

- dd
- mkfs.minix *and/or* mkfs.ext2
- gzip
- Linux kernel source[1] 2.6.x (try to use the newest version)
- Busybox source[2]
- grub
- fdisk *and/or* cfdisk

D-2.1. Kernel configuration and building

The kernel should be extracted to a new directory with the command '*tar xvzf linux-2.6.x.tar.gz*'. It is a good idea to create a separate build directory for the kernel. Here it is assumed that this directory is called *kernelbuild* and located on the same level as the source tree. Go into the root directory of the kernel source tree (probably called linux-2.6.x) and execute the following two commands:

`make O=../kernelbuild allnoconfig` (this will generate lots of useless screen output)
`make O=../kernelbuild menuconfig` (or '... xconfig' if you prefer a graphic interface)

Once the menu system is running, select the required options for the target hardware and system

[1] Official version at www.kernel.org, mirror at http://archive.sun.ac.za/ftp/mirrorsites/ftp.kernel.org/

[2] Official version at www.busybox.net, mirror at http://archive.sun.ac.za/ftp/mirrorsites/pub/mirrors/gentoo/distfiles/busybox-1.00.tar.bz2

requirements. This is time consuming. A basic configuration might look like this:

```
General -> System V IPC
Processor -> Family = Pentium 3/Celeron (Coppermine)
Processor -> HPET Timer
Processor -> Preemptive Kernel
Processor -> Local APIC Support
Processor -> Machine Check Exception
Processor -> MTRR
Bus options -> PCI
Bus options -> ISA
Executable files -> Kernel ELF
Device Driver -> Block devices -> RAM disk
Device Driver -> Block devices -> (4096) Default RAM disk size (kbytes)
Device Driver -> Block devices -> Initial RAM disk support (initrd)
Device Driver -> ATA/ATAPI/MFM/RLL -> ATA/ATAPI/MFM/RLL support
Device Driver -> ATA/ATAPI/MFM/RLL -> Enhanced IDE/.. support
Device Driver -> ATA/ATAPI/MFM/RLL -> Include IDE/ATA2
Device Driver -> ATA/ATAPI/MFM/RLL -> Generic IDE
Device Driver -> ATA/ATAPI/MFM/RLL -> PCI IDE
Device Driver -> ATA/ATAPI/MFM/RLL -> Generic PCI bus master
Device Driver -> ATA/ATAPI/MFM/RLL -> Use PCI DMA
Networking -> Networking Support
Networking -> Networking options -> Packet Socket
Networking -> Networking options -> Unix domain sockets
Networking -> Networking options -> TCP/IP networking
Networking -> Network device support -> Ethernet 10/100 Mbit
Networking -> Network device support -> Ethernet 10/100 Mbit -> Tulip family ... -> Davicom 910x
Character devices -> Serial drivers -> 8250/16550 serial support
Character devices -> Serial drivers -> Console on .. serial port
Character devices -> Legacy PTY support
Character devices -> Max number of PTY's = 16
Character devices -> Enhanced RTC
File Systems -> Second extended fs
File Systems -> Minix fs support
File Systems -> Pseudo fs -> /proc fs
```

Note that kernel options tend to change form time to time. The above options might have moved to another part of the configuration and may not be exactly where they were at the time of this writing. The *.config* file should be saved in a safe place for future modifications. When there is an existing *.config* file available it should be placed in the *kernelbuild* directory. Once configuration is done, the kernel can be compiled with:

```
make O=../kernelbuild 2>&1 | tee ../kernelbuild/build.log
```

If it is suspected that compilation failed, be sure to check *build.log* for any problems. When compilation is done, the kernel will exist in the *kernelbuild* directory and will be ready for installation by the *mkrootfs.sh* script.

D-2.2. Compile Busybox

Busybox is used to provide the shell environment on the embedded linux installation. Extract the Busybox source code to its own directory and configure it. The *Config.h* file selects which commands and features are available. Again, this takes some time and the reader should carefully consider what commands are truly needed. Every command chosen will increase the size of Busybox. The documentation will assist in making a good choice. Currently, these commands are included:

```
ash, cat, chmod, chvt, clear, cp, dd, df, dirname, dmesg, du, echo, env, find,
free, freeramdisk, fsck_minix, gunzip, gzip, halt, hostid, hostname, ifconfig,
init, kill, killall, klogd, ln, logger, ls, kdir, mount, mv, nc, pidof,
poweroff, reboot, reset, rm, sleep, stty, sync, syslogd, tar, telnet, tftp,
umount, watchdog
```

Various other options are set in the *Config.h* file. Read it. Busybox must be compiled statically. Edit the makefile and set *DOSTATIC = true*. Compile Busybox.

D-2.3. Build the root filesystem

Once the kernel and Busybox has been compiled, the root filesystem can be built. The *mkrootfs.sh* or *mkrootfs_minix.sh* (recommended) scripts handle this task. The reader is required to set the various internal paths of the scripts as required. The scripts are commented and it should be fairly easy to follow. Be sure to log in as root or super user. The scripts will fail otherwise. As a brief explaination, the user configurable section of mkrootfs_minix.sh is listed below:

Shell variable in mkrootfs_minix.sh	Notes
`PREFX=/home/fanus/embedded/linobc`	The root of the source tree
`DEVICE=/tmp/initrd`	Place where the temporary image will be created
`MNT=/mnt/initrd`	Temporary mount point for the linobc image
`MKRTFS=${PREFX}/mkrootfs`	Directory, relative to root source, where mkrootfs is
`BBX=${PREFX}/busybox-0.60.5`	Directory for Busybox
`KRNL=${PREFX}/kernelbuild`	Directory of compiled kernel
`FSIZE=31300`	Size of Chipdisk in kBytes
`IMG=/var/lib/tftpboot/obc`	Name and location of final linobc image

Upon success, the chosen script will create the obc.gz file in a location specified by the script. The default location is /var/lib/tftpboot/obc.gz. This assumes that the tftp-server package is installed on the development platform, which is not necessarily what the reader may want.

D-2.4. Chipdisk Partitioning

Shut down the development platform and install the Chipdisk IDE drive with the 44-pin to 40-pin IDE adapter. Be sure to connect power to the adapter. The Chipdisk should be auto detected by modern BIOSes.

When the system has booted, the Chipdisk will typically be the device called /dev/hdc, but this is dependant on where the user physically installed the device. With the obc.gz created, follow these steps:

1. Run *fdisk /dev/hdc* or *cfdisk* as the root user. This command will partition the Chipdisk.
2. Create a minix filesystem with *fdisk* (or *cfdisk*). An ext2 filesystem should be created if the user has created the obc.gz file for this type of filesystem. For *fdisk*, the commands are:
    a. **d** to delete any partitions

   b. **n** to created a new partition, **p** for primary, defaults for sectors (just press enter).
   c. **t** for type. **1** for partition number one. 81 is minix, 83 is ext2
   d. **w** to write the partition to the disk
   e. **q** to quit
3. Assuming the target harddrive is /dev/hdc do: *zcat /var/lib/tftpboot/obc.gz | dd of=/dev/hdc1*
4. Run *grub*, then type
   a. root (hd1,0)
   b. install /boot/stage1 (hd1) /boot/stage2 p /boot/menu.lst
   c. quit

With these steps done, there should now be a clean embedded Linux installed on the Chipdisk. Remove it from the development platform, install it on the MOPSlcd7 and test it.

D-3. Using linobc

With linobc system installed on the Chipdisk, the MOPSlcd7 should boot over COM 1 at 9600 baud. The serial port serves as the keyboard and screen interface. Should it be required, a normal keyboard and screen can be connected to the system. Simply press ESC when the system boots and control will be passed to the real keyboard and screen.

A few things to consider when using linobc:

- The root filesystem is created in memory when the system boots. Any files saved here will be lost once the system is shut down.
- The Chipdisk harddrive is located in the */mnt/flash* directory. Anything stored here will remain after the system is shut down.
- Common system information is accessible in the */proc* directory in the form of text files.
- If the minix filesystem was chosen, linobc will check the disk for errors when it boots.
- It is a good idea to issue the command *halt* just before the system shuts down. This will ensure that all files are written to disk.

- *ifconfig* is used to change the IP of the system. No automatic IP configuration is done.
- *tftp* is the command used to send and receive files via the LAN connection.
    - To send a file: `tftp -p -l localfilename server_ip`
    - To receive: `tftp -g -r remotefilename server_ip`
- The remote PC should run a tftp server. For Windows® a good server is *Solarwinds-TFTP-Server*, for Linux, the *tftp-server* package will work fine.
- *tftp* does not preserve file permissions during transfer, so the user needs to set a file to executable before it can be run: *chmod 777 filename*
- No text editor is included by default with linobc, but simple text files can be created as follows:
    - Type `cat /dev/stdin > filename`
    - If a script (batch command file) is being created, the filename must have the extension .sh and the first line of the script must be #sh
    - Enter the commands line by line as if they were issued on the command line.
    - When done, press Ctrl+D. To abort, press Ctrl+C.

D-4. User applications

User applications need to be linked statically since no dynamic libraries are included in linobc. In the simplest case, this means passing the *–static* option to gcc when compiling, and of course having the glibc-static package installed on the development machine. A makefile can ease the coding of user applications greatly.

D-4.1. Makefile

Below is a good makefile to start with. Add the source and header files at the top, then type *make*. There are a few targets as well. Use *make clean* to remove all object files and binaries, *make export* (as root) to copy the executable to the tftp server directory, ready for linobc to copy.

```
SOURCES.c= isacan.c timer.c gps.c main.c
```

```
INCLUDES= protocol.h isacan.h timer.h gps.h
CFLAGS= -O2 -static -fomit-frame-pointer -Wall -march=pentium3 -msse
SLIBS= -lm
PROGRAM= heli
TFTPDIR= /var/lib/tftpboot

OBJECTS= $(SOURCES.c:.c=.o)

.KEEP_STATE:

debug := CFLAGS= -g

all debug: $(PROGRAM)

$(PROGRAM): $(INCLUDES) $(OBJECTS)
	$(LINK.c) -o $@ $(OBJECTS) $(SLIBS)

clean:
	rm -f $(PROGRAM) $(TFTPDIR)/$(PROGRAM) $(OBJECTS) *~

export:
	cp $(PROGRAM) $(TFTPDIR)/$(PROGRAM) -f
	chmod 777 $(TFTPDIR)/$(PROGRAM)
```

The above makefile works fine for simple source trees where there is typically a *main.c* file that includes a number of headers. When the included files start using each other, the makefile may fail, or tend to miss modifications to upper level files.

Appendix E Control Application for OBC

E-1. Telemetry Details

The protocol used for communication between ground-station and OBC is based on the implementation of Peddle [25]. A general packet has the form:

```
$ AA data * CS
```

Where,

Identifier	Details
$	Start of string delimiter
AA	Two letters for string identifier (Packet ID)
data	Binary data string
*	End of string delimiter
CS	8 bit checksum

subject to the following rules,

- The checksum is formed by taking the exclusive OR of all characters in the string between but not including the start and end of string delimiters.
- If any byte in the data string is a start or end of string delimiter, then a second data character is placed in succession so as to avoid ambiguity.
- If the checksum is calculated to be an end of string delimiter character, then it is replaced by a '+' to avoid confusion with the previous rule.

The 'data' part of the string is in binary form to increase the information density, especially for telemetry purposes where the baud rate is low.

E-1.1. Ground to OBC packets

Set telemetry rate

- Packet ID = ‘GG’
- 4 Data bytes
- Each data byte represents the update (echo) rate in samples that the specific set of data should be transmitted by the OBC. A value of 0 turns off transmission for that data set. As an example, a value of 53 will transmit the data set every 53 samples, or, at 50 Hz, 53/50 = 1.06 seconds.
- Note that the OBC will detect when the maximum bandwidth is exceeded and will ignore the telemetry rate.

Data byte	Description
0	IMU echo rate
1	PILOT echo rate
2	GPS echo rate
3	OBC STATUS echo rate

<u>Terminate OBC control software</u>

- Packet ID = ‘Gb’
- 0 data bytes
- This command is used to terminate the OBC software and can be sent by the terminal keyboard. The full packet is ‘$Gb*%’

E-1.2. OBC to Ground packets

<u>IMU telemetry</u>

- Packet ID = ‘II’
- 26 data bytes
- 12 channels with 16 bit data, starting with Ch0 high byte.
- Last 2 bytes contain ultrasonic data.

<u>PILOT telemetry</u>

- Packet ID = ‘PP’
- 16 data bytes
- 8 channels of 16 bit pilot commands, Ch0 high byte first.

GPS telemetry

- Not fully implemented
- Packet ID = 'SS'
- 15 data bytes
- Only available when in NMEA mode

OBC STATUS telemetry

- Packet ID = 'AA'
- 13 data bytes

Data byte	Description
0	Battery Voltage
1	PILOT echo rate
2	GPS echo rate
3	OBC STATUS echo rate

Appendix F X–Cell component details

A concise listing of the specific parts used to complete the X–Cell helicopter.

F-1. Components

- Helicopter Kit: Miniature Aircraft X–Cell Fury .60 Expert (*S/N: 10181279*)
- Engine: OS70 SZ-H .70 helicopter glow engine. Power = 2.5hp @ 16000 rpm
- Main rotor blades: GCT 680 mm, Symmetric, Carbon Fiber
- Governor: Futaba GV-1
- Rate Gyro: Futaba GY-401
- Servos
  - CCPM: JR DS8321, Digital, Metal gears, 0.21s/60°, 9.0 kg.cm
  - Tail rotor: JR NES-810G, Super Analogue, Nylon gears, 0.10s/60°, 2.5 kg.cm
  - Engine Throttle: JR DS8201, Digital, Nylon gears, 0.19s/60°
- RC Receiver: JR PCM R700, 7 channel. Frequency: 35.120 MHz.
- RC Flight battery: 4.8V 1100 mAh Ni-Cd, 4xAA cells.

Appendix G Selected MATLAB® scripts

G-1. CANsens Filter Constants (filter_code.m)

```
fs = 1000;   % sample rate
fc = 7.9;
[B, A] = butter(2, fc/(fs/2));

% convert to normalized value
B1 = abs(B(1))*2^15;                    B2 = abs(B(2))*2^15;                    B3 = abs(B(3))*2^15;
A2 = abs(A(2))*2^15;                    A3 = abs(A(3))*2^15;

% convert to 16 bit value with ROUND
B1_hi = round(B1/256);                  B1_lo = round(B1-B1_hi*256);            B1r = (B1_hi*256+B1_lo)/2^15;
B2_hi = round(B2/256);                  B2_lo = round(B2-B2_hi*256);            B2r = (B2_hi*256+B2_lo)/2^15;
B3_hi = round(B3/256);                  B3_lo = round(B3-B3_hi*256);            B3r = (B3_hi*256+B3_lo)/2^15;
A2_hi = round(A2/256);                  A2_lo = round(A2-A2_hi*256);            A2r = (A2_hi*256+A2_lo)/2^15;
A3_hi = round(A3/256);                  A3_lo = round(A3-A3_hi*256);            A3r = (A3_hi*256+A3_lo)/2^15;

fid = fopen('filter.def', 'w');
fprintf(fid,'//---------------------------- Filter Coefficients ----------------------------\r\n');
fprintf(fid,'// Normalized from 0..2, absolute values only. a1 assumed to be 1.000\r\n');
fprintf(fid,'// a1*y(n) = b1*u(n) + b2*u(n-1) + b3*u(n-2) - a2*y(n-1) - a3*y(n-2)\r\n');
fprintf(fid,'//\r\n');
fprintf(fid,'#define B1_hi %3.0f\r\n#define B1_lo %3.0f\t\t\t// B1 = %5.0f, %8.8f  (%7.4f%% error)\r\n', B1_hi, B1_lo, B1_hi*256 + B1_lo, B(1), (abs(B(1)-B1r)/B(1))*100);
fprintf(fid,'#define B2_hi %3.0f\r\n#define B2_lo %3.0f\t\t\t// B2 = %5.0f, %8.8f  (%7.4f%% error)\r\n', B2_hi, B2_lo, B2_hi*256 + B2_lo, B(2), (abs(B(2)-B2r)/B(2))*100);
fprintf(fid,'#define B3_hi %3.0f\r\n#define B3_lo %3.0f\t\t\t// B3 = %5.0f, %8.8f  (%7.4f%% error)\r\n', B3_hi, B3_lo, B3_hi*256 + B3_lo, B(3), (abs(B(3)-B3r)/B(3))*100);
fprintf(fid,'#define A2_hi %3.0f\r\n#define A2_lo %3.0f\t\t\t// A2 = %5.0f, %8.8f  (%7.4f%% error)\r\n', A2_hi, A2_lo, A2_hi*256 + A2_lo, A(2), (abs(abs(A(2))-A2r)/A(2))*100);
fprintf(fid,'#define A3_hi %3.0f\r\n#define A3_lo %3.0f\t\t\t// A3 = %5.0f, %8.8f  (%7.4f%% error)\r\n', A3_hi, A3_lo, A3_hi*256 + A3_lo, A(3), (abs(abs(A(3))-A3r)/A(3))*100);
fprintf(fid,'//-------------------------------------------------------------------------------\r\n');
fclose(fid);
```

G-2. CANsens 16 bit Calibration

```
% Script to Calibrate 16bit CANsens Gain and offset corrections

close all;
clear all;
figure;

T = 1/50;
cal_ch = [1 5 2 6 3 7 4 8];
lo_cal = [2 2 2 2 2 2 2 2]; % t where low calibration will be taken, ends at t+2
hi_cal = [30 30 20 20 20 20 20 20]; % t where high calibration will be taken, ends at t+2

lo_dat = [];
hi_dat = [];


for n = 1:8
   % 16bit Calibration data. 0.40000V Low, 3.6000V High
   d = csvread(strcat('D:\logs\cal\gain_offs\0',num2str(n-1),'\imu.csv'));
   t = d(:,1)*T;

   % printf('Total data time: %5.2f\n', t(length(t)));
   y = d(:, 2)*256 + d(:,3);       % 1, S0: Accel -Z
   y = [y d(:, 4)*256 + d(:,5)];   % 2, S4: Gyro Vref
   y = [y d(:, 6)*256 + d(:,7)];   % 3, S1: Gyro +Z
   y = [y d(:, 8)*256 + d(:,9)];   % 4, S5: Accel +X
   y = [y d(:, 10)*256 + d(:,11)]; % 5, S2: Gyro +Y
   y = [y d(:, 12)*256 + d(:,13)]; % 6, S6: Accel -Y
   y = [y d(:, 14)*256 + d(:,15)]; % 7, S3: Gyro +X
   y = [y d(:, 16)*256 + d(:,17)]; % 8, S7: Gyro Temp
```

```
    lo_dat = [lo_dat y( (lo_cal(n)/T):((lo_cal(n)+4)/T) ,cal_ch(n))];
    hi_dat = [hi_dat y( (hi_cal(n)/T):((hi_cal(n)+5)/T) ,cal_ch(n))];

    subplot(4,2,n);
    plot(t,y(:,cal_ch(n)));
    ylabel(strcat('Ch ',num2str(n)));
    hold on;
    grid on;
end

figure;
subplot(2,1,1)
plot((1:length(lo_dat))*T,lo_dat); ylabel('A/D Ticks');
legend('S0','S1','S2','S3','S4','S5','S6','S7');
title('0.4V input voltage');
grid on;
subplot(2,1,2)
plot((1:length(hi_dat))*T,hi_dat); ylabel('A/D Ticks');
legend('S0','S1','S2','S3','S4','S5','S6','S7');
title('3.6V input voltage');
grid on;
xlabel('Time [s]');


lo_avg = median(lo_dat)*4.096/2^16;
hi_avg = median(hi_dat)*4.096/2^16;

m = (3.6 - 0.4)./(hi_avg - lo_avg)     % gain error
c = (3.6 - m.*hi_avg)/4.096*2^16       % offset error in ticks

% lo_avg = mean(lo_dat)*4.096/2^16;
% hi_avg = mean(hi_dat)*4.096/2^16;
%
% m = (3.6 - 0.4)./(hi_avg - lo_avg)     % gain error
% c = (3.6 - m.*hi_avg)/4.096*2^16       % offset error in ticks

% Create header file
fid = fopen('cal16.h', 'w');
fprintf(fid, '// File generated with Matlab 7.0.1 - cal_imu.m\n');
fprintf(fid, '// This gives the constants to calculate the true input value\n');
fprintf(fid, '// from the A/D value in the form AD_true = GAIN*AD_out + OFFS\n\n');
for n = 1:8
    fprintf(fid, '#define CAL_S%d_OFFS %10.8f\n',n-1,c(n));
end
fprintf(fid, '\n');
for n = 1:8
    fprintf(fid, '#define CAL_S%d_GAIN %10.8f\n',n-1,m(n));
end
fclose(fid);
```

G-3. Calibration and Alignment of Accelerometers

```
% Script to calibrate IMU on rate table: Rates
close all;
clc;
clear all;

% sample period
T = 1/50;

%ax = [3 2 1; 1 1 2; 3 2 3];                 % measurement axis
names = ['X' 'Y' 'Z'];
align = zeros(3,3);


cal_angle = 360./[5.3185 5.3469 5.1948];
n_avg = 2./T;                   % number of samples to use for average

n_zero1 = 15./T;
n_zero2 = 20./T;

hi_g = floor([25.7 43.25 53.9 63.4 74.05; 27 38.5 48.7 59 68.3; 31 43.4 54 73.6 89.4]./T);
```

```
lo_g = floor([34.7 49.05 59.4 69.45 78; 33 44.1 53.7 64.05 72.5; 23.1 37.5 48.5 60 79]./T);
n_src = 50; % number of samples to search for peak;

accel_hi = [];
accel_lo = [];
a_tst = [3 2 1];  % axis tested in order
a_alg = [1 3 2];  % coupling axis 1 test order
b_alg = [2 1 3];  % coupling axis 2 test order

for k = 1:3
  d = csvread(strcat('e:\tesis\logs\rate_table\',num2str(k+18,'%2.2d'),'\imu.csv'));

  % initial time vector
  t = d(:,1)*T;

  % set time window
  w1 = 10;
  w2 = 600;
  % sanity check
  if (w1/T) > length(t);
    w1 = 0;
  end;
  if (w2/T) > length(t);
    w2 = floor(length(t)*T);
  end;

  % set time window
  t = d((w1/T):(w2/T),1)*T-w1;

  % printf('Total data time: %5.2f\n', t(length(t)));
  y = d(:, 2)*256 + d(:,3);       % 1, S0: Accel -Z
  y = [y d(:, 4)*256 + d(:,5)];   % 2, S1: Gyro +Z
  y = [y d(:, 6)*256 + d(:,7)];   % 3, S2: Gyro +Y
  y = [y d(:, 8)*256 + d(:,9)];   % 4, S3: Gyro +X
  y = [y d(:, 10)*256 + d(:,11)]; % 5, S4: Gyro Vref
  y = [y d(:, 12)*256 + d(:,13)]; % 6, S5: Accel +X
  y = [y d(:, 14)*256 + d(:,15)]; % 7, S6: Accel -Y
  y = [y d(:, 16)*256 + d(:,17)]; % 8, S7: Gyro Temp

  % swap 04152637 error
  z = y;
  %y = [z(:,1) z(:,3) z(:,5) z(:,7) z(:,2) z(:,4) z(:,6) z(:,8)];

  % normal channel mix ???
  y = [z(:,1) z(:,5) z(:,2) z(:,6) z(:,3) z(:,7) z(:,4) z(:,8)];


  % scale to voltages
  y = y((w1/T):(w2/T),:)./(2^14);


  % extract accel
  accel = [y(:,6) y(:,7) y(:,1)];     % Accel X, -Y, -Z
  accelcal = mean(accel(n_zero1:n_zero2,:));
  %sc_accel = 1.15;    % acceleration scale

  for w = 1:5
    v = accel( (hi_g(k,w)-n_src):(hi_g(k,w)+n_src),a_tst(k));
    accel_hi(k,w) = max(v);
    v = accel( (lo_g(k,w)-n_src):(lo_g(k,w)+n_src),a_tst(k));
    accel_lo(k,w) = min(v);
  end
end


a_scale = (mean(accel_hi')-mean(accel_lo'))/2

%--------------------------------------------------------------------------
for k = 1:3
  d = csvread(strcat('e:\tesis\logs\rate_table\',num2str(k+18,'%2.2d'),'\imu.csv'));

  % initial time vector
  t = d(:,1)*T;

  % set time window
  w1 = 10;
  w2 = 600;
  % sanity check
  if (w1/T) > length(t);
    w1 = 0;
```

```
end;
if (w2/T) > length(t);
    w2 = floor(length(t)*T);
end;

% set time window
t = d((w1/T):(w2/T),1)*T-w1;

% printf('Total data time: %5.2f\n', t(length(t)));
y = d(:, 2)*256 + d(:,3);       % 1, S0: Accel -Z
y = [y d(:, 4)*256 + d(:,5)];   % 2, S1: Gyro +Z
y = [y d(:, 6)*256 + d(:,7)];   % 3, S2: Gyro +Y
y = [y d(:, 8)*256 + d(:,9)];   % 4, S3: Gyro +X
y = [y d(:, 10)*256 + d(:,11)]; % 5, S4: Gyro Vref
y = [y d(:, 12)*256 + d(:,13)]; % 6, S5: Accel +X
y = [y d(:, 14)*256 + d(:,15)]; % 7, S6: Accel -Y
y = [y d(:, 16)*256 + d(:,17)]; % 8, S7: Gyro Temp

% swap 04152637 error
z = y;
%y = [z(:,1) z(:,3) z(:,5) z(:,7) z(:,2) z(:,4) z(:,6) z(:,8)];

% normal channel mix ???
y = [z(:,1) z(:,5) z(:,2) z(:,6) z(:,3) z(:,7) z(:,4) z(:,8)];


% scale to voltages
y = y((w1/T):(w2/T),:)./(2^14);


% extract accel
accel = [y(:,6) y(:,7) y(:,1)];     % Accel X, -Y, -Z
accelcal = mean(accel(n_zero1:n_zero2,:));

% Scale to proper physical values
accel(:,1) = (accel(:,1) - accelcal(1)) / a_scale(a_tst(1));
accel(:,2) = (accel(:,2) - accelcal(2)) / a_scale(a_tst(2));
accel(:,3) = (accel(:,3) - accelcal(3)) / a_scale(a_tst(3));

% search and save exact maximum g location to calculate misalignment
c = [];
mx = [];
b = [];
for w = 1:5
    v = accel( (hi_g(k,w)-n_src):(hi_g(k,w)+n_src),a_tst(k));
    c = [c v'];
    for q = 1:length(v)
        if v(q) == max(v)
            mx = [mx q+length(b)];
            break;
        end
    end
    v = accel( (hi_g(k,w)-n_src):(hi_g(k,w)+n_src),a_alg(k));
    b = [b v'];

end

% Calculate misalignment of high calibration point
ms = asind(b./c);

figure;
subplot(2,1,1);
plot((1:length(c))*T,ms,'b');
hold on;
plot((1:length(c))*T,[c],'r');
plot(mx*T,c(mx),'go');
plot(mx*T,ms(mx),'go');
ylim([-5 5]);


a_align = mean(ms(mx))

% search and save exact maximum g location to calculate misalignment
c = [];
mx = [];
b = [];
for w = 1:5
    v = accel( (lo_g(k,w)-n_src):(lo_g(k,w)+n_src),a_tst(k));
    c = [c v'];
    for q = 1:length(v)
```

```
        if v(q) == min(v)
            mx = [mx q+length(b)];
            break;
        end
    end
    v = accel( (lo_g(k,w)-n_src):(lo_g(k,w)+n_src),a_alg(k));
    b = [b v'];

end

% Calculate misalignment of low calibration point
ms = asind(b./c);

subplot(2,1,2);
plot((1:length(c))*T,ms,'b');
hold on;
plot((1:length(c))*T,[c],'r');
plot(mx*T,c(mx),'go');
plot(mx*T,ms(mx),'go');
ylim([-5 5]);

a_align = mean(ms(mx))
```