University of Southern Queensland

Faculty of Engineering and Surveying

The Autonomous Unmanned Aerial Surveillance Vehicle
Navigation and User Interface

A dissertation submitted by

Soz Deja Knox

in fulfilment of the requirements of

Courses ENG4111 and ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Electrical and Electronic)

Submitted: October, 2005

Abstract

There has been a steep growth, throughout the World, in civilian UAV applications. UAVs offer cost effective alternatives for many applications, as they enable objectives be carried out without the risk to pilots. In terms of crop surveillance, they offer a cheap alternative to satellite and manned aerial imagery. These current techniques are limited due to cloud cover, and poor image capture resolution. By flying a UAV at low altitudes, these limitations may be greatly reduced. With the recent financial commitment from the Queensland State Government, to the funding of a UAV research and development facility, the future of UAV technology and its potential market is being recognized. Current UAV navigation and guidance packages on the market are prohibitively expensive. This paper describes the design and implementation of a cheap navigation and user interface for the specific purpose of carrying out surveillance over a pre-determined flight path in a UAV. The system has been designed assuming a suitable automatic flight control system is available, for receipt of the guidance outputs.

By using a HC12 microprocessor, GPS, compass and transceivers, in conjunction with a navigation algorithm, guidance of the UAV is made possible by producing heading error and altitude to feed to the automatic flight control system of a UAV. A digital camera and electronic trigger unit are used for taking surveillance photos over the pre-determined path. A user interface has been designed for the entering of four waypoints (desired path) and includes a telemetry downlink for real time visual indication of UAV status, including desired heading, current heading, altitude, position, GPS link status, downlink communication status, and raw data. A prototype of the system is designed, implemented and built, with appropriate risk control measures applied as identified in the project Risk Assessment.

The prototype system described in this dissertation successfully carries out navigation, guidance and surveillance over a pre-determined flight path. Simulated testing using a trolley guided by hand, communicating wirelessly to a ground control station with user interface, has proven the successful design, implementation and final integration of the individual elements.

Improvements and modifications are suggested that may enhance the current features of this system, however, successful implementation of the prototype system in this project has proven that UAV surveillance can be carried out, using cheap but effective methods.

UNIVERSITY OF SOUTHERN QUEENSLAND

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof G Baker
Dean
Faculty of Engineering and Surveying

DISCLAIMER PAGE

Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Soz Deja Knox

Student Number: 0039911442

_____
Signature

_____
Date

Acknowledgements

I would like to thank the following people and organisations who have made completion of this project possible.

- My supervisors, Mark Phythian and LEUT Kathryn Burr who managed to answer my questions or point me in the right direction over and over again throughout the year,
- Terry Byrne for his technical expertise, enthusiasm and for the excellent idea in the first place,
- The Royal Australian Navy who sponsored me financially in the past three years. I would not have been able to focus so much of my energy into this project, thereby gaining as much as I have through this process, without the freedom provided me.
- Craig Littleton, Dean Thompson and Michael Costa for putting up with me everyday, and
- My two beautiful dogs- Rooster and Dex.

Soz Knox

University of Southern Queensland

October 2005

Table of Contents

List of Figures

List of Tables and Equations

Nomenclature

ABR	Australian Book of Reference
ABS	Acrylonitrile Butadiene Styrene
A/D	Analogue to digital
Algorithm	Step by step procedure for solving a problem
ANSI C	American National Standards Institute for C programming language
ASCII	American Standard Code for Information Interchange
Assembler	Translator from assembly language to machine language
AS/NZS	Australian New Zealand Standard
Autonomous	Self governing; self controlling; self contained
Avionics	Aviation electronics
Baud rate	The rate at which data flows
BCD	Binary Coded Decimal
Binary	Number system consisting of zeros and ones only
Bit	Binary Digit- smallest unit of storage in a computer
Byte	A group of eight data bits
C	Standardised programming language
CAN	Controller Area Network
CASA	Civil Aviation Safety Authority
CASR	Civil Aviation Safety Regulations
Compiler	Translates a high level language into machine language
CR	Carriage Return
C++	An extension of the C programming language (object oriented)
DDR	Data Direction Register
DGPS	Differential Global Positioning System
DME	Distance Measurement Equipment
EEPROM	Electrically Erasable Programmable Read Only Memory
EMI	Electromagnetic Interference

Flash	Non-volatile memory
ft	Foot
GCS	Ground Control Station
GIS	Geographical Information System
GND	Ground potential
GPS	Global Positioning System
GUI	Graphical User Interface
ha	Hectare
HC12	16 bit microprocessor
Hz	Hertz
IBM	International Business Machines Corporation
ICC12	ImageCraft Integrated Development Environment (Compiler)
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineers
Java	Object oriented programming language
KB	Kilo Byte
LAN	Local Area Network
LF	Line Feed
LORAN	Long Range Navigation system
LSBF	Least Significant Bit First
Mbits/s	Mega bits per second
MHz	Mega Hertz
MISO	Master In Slave Out
MOSI	Master Out Slave In
NASA	National Aeronautics and Space Administration
NDB	Non-directional beacon
NMEA- 0183	National Maritime Electronic Association standard
OEM	Original Equipment Manufacturer
Payload	Instruments or equipment carried by an aircraft
PC	Personal Computer (usually IBM compatible)
Pseudorange	Distance measurement based on correlation
RAM	Random Access Memory
RF	Radio Frequency
RMC	Recommended Minimum Sentence (GPS)
RPV	Remote Piloted Vehicle
RS232	Electrical signal specification

RTS/CTS	Request to send/ clear to send
SCI	Asynchronous Serial Communications Interface
SCLK	Clock signal
Servo	Electro-mechanical device that moves control surfaces
SLAM	Simultaneous Localisation and Mapping
SPI	Serial Peripheral Interface
Telemetry	Data stream of measured values
UAV	Unmanned Aerial Vehicle
USQ	University of Southern Queensland
UTC	Universal Time Coordinated
VDC	Direct Current Voltage
Vin	Input Voltage
VOR	VHF omnidirectional range
V2X	Compass Module
Waypoint	A specific location defined by GPS coordinates
xxxBR	Baud Rate Register
$GPGGA	Global Positioning System fixed data sentence
λ_1	Longitude of initial position
λ_2	Longitude of desired position
Φ_1	Latitude of initial position
Φ_2	Latitude of desired position

Chapter 1

Introduction

An Unmanned Aerial Vehicle (UAV) is a powered aerial vehicle sustained in flight by aerodynamic lift and guided without an onboard crew. It may be expendable or recoverable and can fly autonomously or be piloted remotely.

Autonomous UAVs require systems that enable them to maintain a stable attitude and the ability to follow desired trajectories. These avionic systems require continuous gathering of data from sensors (and human inputs), and the ability to process this information to guide and control the UAV through the generation of actuator commands. The avionics package necessary to carry out these functions includes a processor, sensors, software and data link hardware (Dittrich 2002).

Navigation and guidance of a UAV over a pre-determined path assumes that the existing system is capable of carrying out autonomous straight and level flight. The existing avionics package is then supplemented with extra sensors and software to include navigation and guidance capability.

Surveillance and user interface capabilities of a UAV require the addition of camera technology as well as data link hardware connected to some form of user enterable information platform. This allows for the inclusion of telemetry downlink of data regarding UAV status as well as a platform for viewing the images taken during flight.

1.1 History of UAV Development

Unmanned Aerial Vehicles have been around since the dawn of aviation, and Australia has been developing some form of UAV since the late 1940s including the highly successful GAF Jindivik target drone (Wong & Bil 2003). Remotely Piloted Vehicles (RPVs) have been used in niche military applications, and whilst many believed they would take over various roles of piloted aircraft, the fact that they require a skilled pilot on the ground has limited their growth (Wong & Bil 2003). Current UAV technology however, allows for fully autonomous systems to be employed. It has been the development of lightweight, inexpensive and compact sensors and microprocessors as well as a fast growing world-wide UAV knowledge base to which Wong and Bil (2003) attribute the steep growth in the market for civilian UAV applications. UAVs have unmatched qualities that often make them cost effective methods of carrying out objectives that may be either highly risky to pilots or where their presence isn't necessary.

Recently, the Queensland State Government committed financial assistance to aid in a UAV research and development facility in Brisbane. As stated by Premier Beattie (Dept. of State Development, Trade and Innovation. 2005):

> "UAVs comprise a significant aerospace research and development opportunity that is not being pursued by any other State."

This commitment from the State Government proves the relevance and value of the project represented by this dissertation, and endorses the University of Southern Queensland's promotion of UAV development as an ongoing project.

Regardless of the application, any UAV developed needs to operate in non-segregated civilian airspace (McManus 2004). As a consequence, regulators are presently working on policy to enable integration of UAVs into civilian airspace. In the Civil Aviation Safety Authority (CASA) Regulation (1998), it is stated that:

- UAV certification is not required for those weighing less than 150kg and flying below 400ft (approx. 1312 m). This is considered a 'small' UAV.
- If flying outside these parameters then certification is required and continuous communications are necessary.

The CASA regulation does not specify a certification process. In fact, there is presently no guidance on how to carry out certification in spite of its requirement under the conditions listed above.

'Small' UAVs have no restrictions imposed upon their operation. The operator of the UAV is responsible for ensuring that the UAV is operated safely and remains clear of potential low level traffic, structures, powerlines etc (CASA 1998).

1.2 Project Background

The Autonomous Unmanned Aerial Surveillance Vehicle – Navigation and User Interface is a follow-on project initiated by Michelle Keefe (2003). The primary requirement for this project was to develop a functional prototype of an autonomous radio-controlled aircraft for the purpose of conducting aerial surveillance over a pre-determined path. This project was carried out up to the point of semi-autonomous flight (straight and level), remote direction controlled by the user.

As a consequence, the Autonomous Unmanned Aerial Surveillance Vehicle – Navigation and User Interface project was proposed to provide the navigation solutions for full autonomous flight, integration of an image capture system and a ground control station with a graphical user interface (GUI) for entering flight path data and display of telemetry information during flight. The emphasis is on low cost to enable the prototype to be a viable alternative to current satellite surveillance of crops. This project has been carried out in conjunction with 'The Autonomous Unmanned Aerial Surveillance Vehicle- Autonomous Control and Flight Dynamics' (Littleton 2005).

1.3 Project Aim

The aim of this project was to develop a navigation algorithm, image capture system and user interface to integrate into a fully functional prototype of an autonomous Unmanned Aerial Vehicle (UAV) capable of carrying out surveillance over a pre-determined flight path.

1.3.1 Operational Requirements

The overall requirements of the system were as follows:

- The system shall provide a user interface to allow entering of waypoints to determine a flight path.
- The system shall provide navigational guidance information to allow autonomous flight over this pre-determined flight path.
- The system shall allow capturing of image information over pre-determined points on the flight path.
- The system shall be integrated into a fully functional prototype UAV.

1.3.2 Performance Requirements

For the purpose of this project, certain assumptions were made regarding the scope of the UAV's performance. These assumptions were:

- The aircraft will be remotely controlled by a user during take-off and landing.
- The distance that the aircraft is expected to fly during the autonomous control phase will be no more than 500m from the Ground Control Station (GCS) laptop.

- The UAV will not be flown over 400ft (1312m) and will not weigh more than 150kg.

- The camera chosen for surveillance will be limited to a finite number of pictures based on its current memory capability and resolution.

The flight path chosen includes no more than four waypoints as a means of ensuring that the distance from the GCS mentioned above is not exceeded.

1.3.3 Constraints

As this was a University of Southern Queensland Faculty-sponsored project, resources and costs associated are limited by those available to the faculty. As such, every effort was made during the system design phase to utilise available hardware and software. Where possible, the off the shelf solutions chosen, balance low cost with acceptable quality.

The payload limitation of the remote controlled aircraft chosen for this project represented the primary design and performance constraint. The model aircraft to be used originally did not have a very large payload capability. The cost of these aircraft is proportional to the payload capabilities. As a consequence, all hardware for the project was chosen to be as small and light as possible to achieve acceptable levels of performance.

1.4 Project Objectives

The objectives of this project in accordance with the specification, as included in Appendix A, were as follows:

1. Research information for the design and implementation of common Unmanned Aerial Vehicle (UAV) GPS guided navigation algorithms and user interfaces.

2. Research requirements for the design and implementation of an image capture system including hardware, interfacing requirements and payload limitations.

3. Research communication alternatives to interface Ground Station (user interface) with UAV for uplink of navigation algorithm and downlink of UAV telemetry.

4. Design, develop and test individual systems, i.e. navigation algorithm, image capture system and user interface.

5. Design interface for the individual systems and integrate to enable surveillance over a pre-determined path with non real-time entering of flight path and real-time telemetry downlink.

6. Construct prototype UAV (using a model aircraft) and integrate systems with 'Autonomous Control and Flight Dynamics' project being carried out by Craig Littleton.

1.5 Project Methodology

The methodology adopted in this project is shown in Figure 1. In this methodology the overall UAV system requirements are analysed and allocated to software and hardware sub-system design stages. The requirements for each sub-system were analysed, designed, built and tested in stages. By using a staged approach, the project could be systematically carried out and appropriately documented to enable review between stages and thorough evaluation during the testing phases. When problems were identified during the development and testing phases of each stage, redesign and further testing was carried out. When the hardware and software sub-systems development proved successful, integration and testing the system as a whole was confidently carried out. The original project timeline is included in Appendix I for reference. This includes a Gantt Chart of all the project activities and the times anticipated for implementation.

Once analysis of the UAV system level requirements was performed, hardware selection for the UAV was carried out. Hardware selection in a UAV environment requires that the sensors, microprocessor and data link hardware be chosen with the specific operational requirements in mind. The selection of this hardware and its integration into the final system is discussed in Chapters 6 and 7.

Similarly, once the UAV system requirements were allocated to the software sub-system, design and development of both a navigation algorithm and user interface were carried out. Sensor data was integrated into the navigation solution through the use of algorithms implemented in software and programmed into the navigation UAV microprocessor. The implementation of the navigation algorithm as well as the development of the Ground Control Station and user interface is discussed in Chapter 8.

A hardware and software test plan was developed during the sub-system design phases. It was crucial that any existing faults be found prior to integration of the whole system to enable systematic fault analysis be carried out with as few dependencies as possible. Also, once the overall system was integrated, a thorough test procedure was executed to ensure the system was sufficiently mature for integration into a prototype UAV. This testing was carried out by simulating the UAV flight path using a trolley that could be manoeuvred by hand. The specific details of this testing is given in Chapter 9.

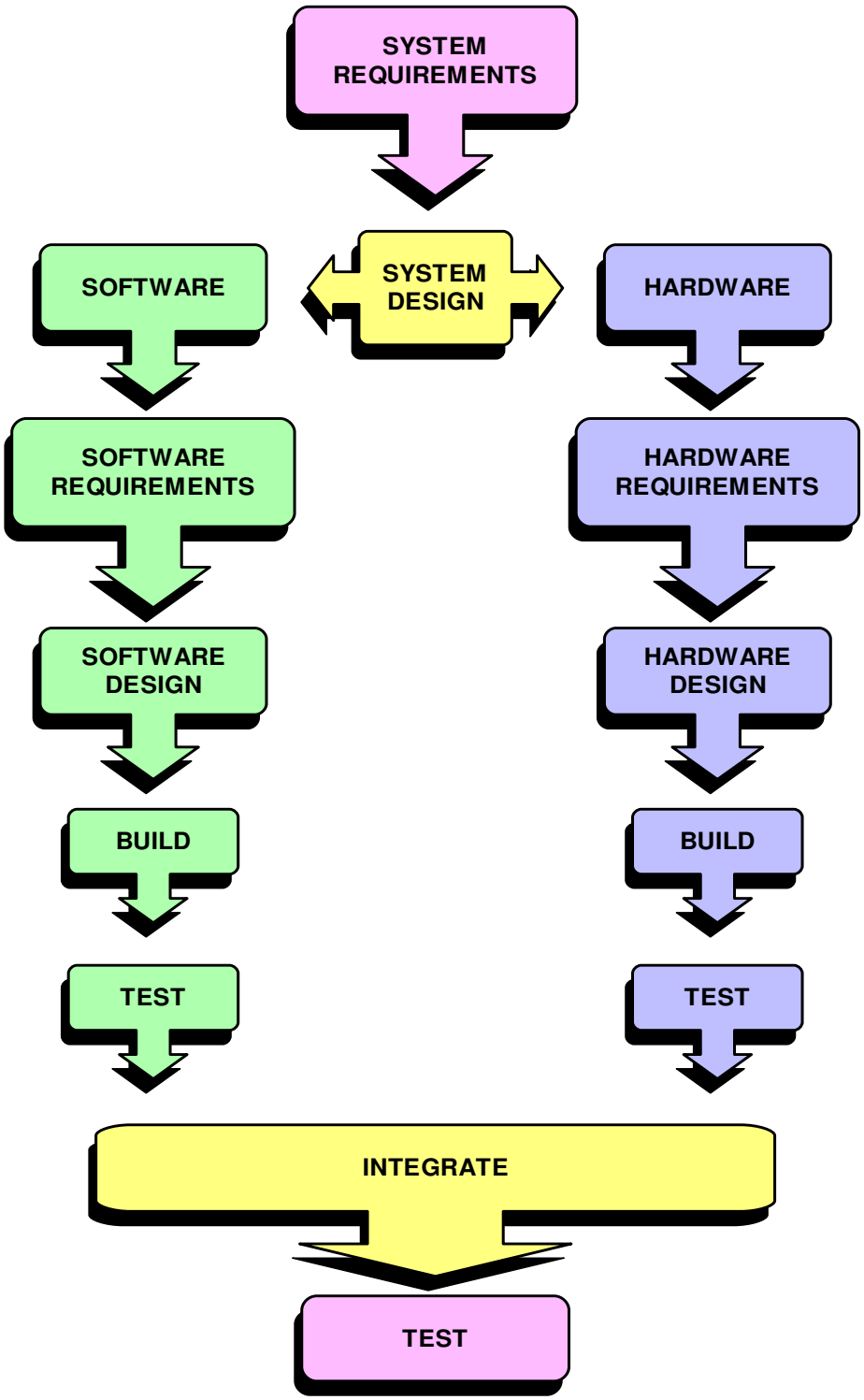


Figure 1- System Development Methodology

1.6 Aspects of Ethical Responsibility

The development of a cheap, fully autonomous UAV capable of carrying out surveillance over a pre-determined flight path may offer an affordable and sustainable alternative to current means of crop surveillance. If used, it will alleviate the safety issues inherent in human-flown aircraft used for this purpose. It is a technical task that has the potential to endanger the public if the technology is used for the wrong purposes. The ease and low cost may allow the technology to be used by those with less innocuous objectives. The UAV has the potential to be used as an autonomous weapon or for intrusive surveillance. Also, by its nature, the UAV is an aircraft operating in domestic airspace and as such, there exists a legal responsibility to ensure it is operated safely and effectively within that airspace.

The performance requirements for this project do not exceed the height or weight restrictions imposed on UAVs by the CASA 1998 regulation, and as such, operation of the UAV will not require certification. This does not mean that the prototype UAV will not be capable of flying at those altitudes and as such, it is important that the prototype UAV be considered potentially dangerous and only be operated by engineering staff/ students under strict supervision and in a safe environment.

The prototype UAV is not a toy and should not be treated as such. Any possible future marketing of the product to the government or to farmers would need to include thorough warnings associated with the above-mentioned concerns and educational materials outlining regulatory issues as well as detailed instructions on its use.

1.7 Dissertation Structure

This dissertation is organised as follows:

Chapter 2 covers the main findings of a literature review carried out on the current state of navigation and guidance methods used within the aviation industry. It discusses and investigates viable methods to achieve guidance of a UAV over a pre-determined path.

Chapter 3 includes the literature review carried out on the image capture system and user interface. It discusses the efficacy of using UAVs for crop surveillance, and some of the methods available to implement an image capture system. The purpose and function of the user interface is discussed, along with the benefits of the chosen Programming language. The Ground Control Station hardware requirements necessary to implement the user interface is described.

Chapter 4 gives a description of the system architecture design concept as decided upon, based on the literature review carried out. The chapter also presents a resource analysis carried out based on the conceptual design.

Chapter 5 is a risk assessment carried out on the project and its activities. An evaluation of these risks and the necessary control measures is tabulated.

Chapter 6 describes in detail the navigation and guidance system hardware used, and how it was developed. The development tools used to aid in the implementation of the hardware are described, and individual testing of the components is discussed.

Chapter 7 explains the selection and implementation of the image capture and user interface hardware used in this project. Testing of each of these components is discussed, and an integrated system hardware wiring diagram is provided.

Chapter 8 describes the system software design and implementation carried out. The chapter documents each of the programs necessary to carry out the navigation algorithm and hardware interfacing. The user interface design is described and final implementation shown. Testing of the individual functions is discussed.

Chapter 9 outlines the integrated system test methods, results and discussion necessary for system verification. Where required, options for problem rectification are canvassed, and subsequently implemented.

Chapter 10 analyses the results of the final system tests in relation to the project specification, aim and objectives.

Chapter 11 begins by revisiting the relevance of this area of research and development in the current market. The overall performance of the system is discussed with reference to the preceding chapters. Recommendations are made for future improvements and/or possible modifications to the final system.

Chapter 2

Navigation and guidance

Navigation is the determination of the position and velocity of a moving vehicle (Kayton & Fried 1997). Guidance refers to the act of steering toward a destination of known position from the aircraft's present position. Therefore, navigation is a necessary step towards aircraft guidance. Navigation systems can be categorised as either positioning or dead-reckoning where positioning systems measure the state vector (the three components of position and the three components of velocity make up a six-component state vector) without regard to the path travelled by the aircraft in the past. Dead-reckoning navigation systems derive their state vector from a continuous series of measurements relative to an initial position. The guidance loop for any UAV is required to generate guidance commands from the UAV states and the desired waypoint information. These commands are passed to the flight control loop which is then responsible for controlling the actuators and servos of the UAV control surfaces.

Fully functional navigational and guidance packages are currently on the market. They are capable of carrying out waypoint navigation along a pre-determined path and include the hardware and software interfaces. The Pegasus Advanced Precision Aerial Delivery System includes a digital flight computer which processes information from the GPS receiver, air speed sensor and compass. These sensors provide the inputs to a guidance algorithm that provide the signals to the servo actuators to enable interception of a pre-programmed flight path (FXC Corporation 2005). This system, like many UAVs already on the market, are prohibitively expensive for use in civilian surveillance applications (Schulze, Abramson & Rogan 2004). Cornell University undergraduate

students developed a UAV package for use in local research activities. The development saw a modern cost-saving approach to developing a UAV and its subsystems. Their budget, however, was US $15 000. According to the Department of Defence (2005), the Avatar UAV and ground station cost US $40 000. The Cornell University packge, whilst a vast improvement on the previous cost of UAVs, still represents a large investment. They use a GPS receiver and pressure sensors to detect position and velocity state, and an autopilot system capable of flying a series of GPS coordinates. Their autopilot includes a powerful onboard computer running Windows XP. This, in addition to the sensor and communication hardware, represents a large payload and as a consequence, the use of a large heavily modified remote controlled aircraft.

Clearly, there is a lack of low cost navigational and guidance packages on the market that can be integrated into small scale aircraft such as UAVs.

Radio-based navigation methods used in manned aviation are usually based on terrestrial non- and omni-directional beacons (NDB, VOR) and/or measurement of distances (DME, LORAN) (Dittrich 2002). It is the high accuracy, simplicity and availability of GPS that Dittrich (2002) suggests is the reason GPS has become the standard positioning system for unmanned aircraft. The question then becomes whether to use single point GPS which is subject to atmospheric errors, or differential GPS (DGPS) which uses a second stationary GPS receiver to correct these errors. The accuracy of single point GPS is within 10 to 30 metres (Stefan 2000). The accuracy of DGPS is within 10 metres. DGPS requires a reference station at a known location that receives the same signals as the GPS receiver being used. This reference station processes its GPS measurements, deriving pseudorange and pseudorange-rate errors with respect to its accurately known location, and then transmits these corrections to the user who is then able to apply these corrections to their measurements. The decision to use DGPS then, will depend on the application and the need for precise location information. If it is only necessary to come within 10 to 30 metres of a given waypoint, then single-point GPS is sufficient and provides a lower cost solution, as it not only avoids the necessity for a reference station, but also the hardware necessary to communicate error data from one GPS to the other.

2.1 GPS

The fully operational GPS satellite constellation comprises 24 satellites as described in the American Federal Radio Navigation Plan (1996). GPS receivers use ranging code from at least four GPS satellites to calculate their instantaneous position and velocity. Most GPS receivers output their data in NMEA- 0183 format. This format includes a variety of transmitted sentences/ messages that provide data for navigational purposes. These sentences are transmitted in ASCII code, each beginning with a dollar sign ($) and ending with a carriage return and linefeed (<CR><LF>). Data is comma delimited. The sentence extracted for use depends on the required application with the recommended minimum sentence being the RMC message (Stefan 2000). The RMC message provides the following information:

- Universal Time Coordinated (UTC),
- GPS status (indicating whether the incoming GPS data is valid),
- Latitude, Northern or Southern hemisphere,
- Longitude, East or West indicator,
- Speed (From 0000.0 to 1851.8 knots),
- Course over ground (000.0 to 259.9 degrees),
- Date, and
- Magnetic Variation (000.0 to 180 degrees).

The following is an example of a GPS transmitted RMC message:

$GPRMC,1030804.374,A,0411.8650,N,10326.3480,E,0.00,3.85,010304,,*03

The majority of GPS receivers update their data every second with a 4800 bps baud rate. This sampling rate represents a significant limitation. If an aircraft is travelling at 45km/h and the GPS receiver is updating every second (1 Hz) then the aircraft will travel 12.5 metres between position measurements. Integrated inertial sensors are often combined with GPS as they offer high frequency information with short term accuracy. As GPS provides low frequency information with long term accuracy, the sensors can be used to complement each other. A widely used technique to integrate inertial sensors (such as Gyros) and GPS, is the use of a Kalman Filter. A Kalman Filter is an error estimator that is used to control complex dynamic systems. It effectively predicts, filters

and smooths the data from the available sensors to correct for their inherent errors (Merminod 1989). Kalman Filtering techniques are used by military aircraft (Royal Australian Navy S-70B-2 helicopters) to ensure accurate position information over long distances. These systems are useful for long range flying, but are not considered necessary for short flights where errors in the inertial sensors and low frequency updating of GPS are not considered critical. Unfortunately, whilst GPS can provide long term stability and accuracy, its reliability is not guaranteed as it relies on a clear line of sight from the receiver to at least four satellites to guarantee a position fix. This is another reason why many UAV applications see the use of GPS complementing inertial navigation systems; ensuring availability of position information regardless of the weather or environmental conditions. When designing UAVs with high manoeuvrability which may impede the GPS receiver to satellite signals, it is possible to add a second redundant GPS receiver to ensure that position information is possible regardless of whether the UAV is upright or not.

2.2 Compass

When using GPS to provide heading information, there is a necessity for an extra sensor to provide attitude information. The GPS provides heading information but only at a 1Hz rate. A faster update rate can be found using a compass, which can be used to supplement the GPS with a higher heading sampling rate.

A popular compass on the market is the HMR3000 Digital Compass Module. This compass provides heading, pitch and roll for use in navigation and guidance systems and is recommended for use in unmanned vehicles (Honeywell Sensor Products n.d.). The HMR3000 costs $675 and therefore, was not considered a viable option for this project.

New Mexico Tech students used a Vector 2X compass in designing their Golfing Robot (2002). They incorporated the compass into their project as a means of providing heading information to supplement their GPS. The Vector 2X- Compass Module is a low cost, light weight 2-axis compass and magnetic sensor module. As the application notes suggest (1998), GPS system backup azimuth data is a possible application of the Vector 2X. The compass uses a synchronous serial port for communications with the

host device .This is compatible with Motorola Serial Peripheral Interface (SPI) capabilities. The output of the Vector compass is accurate to 2° with a resolution of 1°. The format of the heading output can either be Binary Coded Decimal (BCD) or Binary (Precision Navigation Inc 1998). The sample rate in low resolution mode is between 5 and 10 Hz. This sampling rate will enable the navigation algorithm designed for this project to make heading error calculations between GPS position fixes. i.e. heading error = desired heading – current heading. Refer to Figure 2 for pictorial representation.

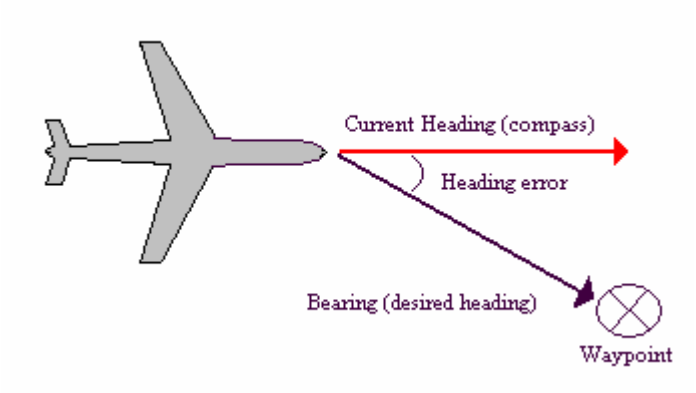


Figure 2- UAV heading and bearing

2.3 Navigation Strategy

Guidance systems for UAVs require commands be sent to servos to enable steering towards a destination. The destination can be provided to the system as a set of waypoints, i.e latitude and longitude of each waypoint. This information is used in this project in conjunction with the GPS measurements of current position to determine a path towards the next waypoint. Dead reckoning calculations will not be used as they assume flat-Earth approximations (Kayton & Fried 1997). Therefore, distance is calculated using the Great Circle Navigation formulae (Williams n.d.). Since the Earth is a sphere, the shortest path between two points is calculated using the great circle distance, which corresponds to an arc linking two points on a sphere.

The great circle distance d between two points with coordinates $\{\Phi_1(lat_1), \Phi_2(lat_2)\}$ and $\{\lambda_1(lon1), \lambda_2(lon_2)\}$ is given by:

$$d = \text{acos}\left(\sin(\Phi_1) \text{ x } \sin(\Phi_2) + \cos(\Phi_1) \text{ x } \cos(\Phi_2) \text{ x } \cos(\lambda_1 - \lambda_2)\right)$$

Equation 1- Great Circle Distance Formula

The initial true course or bearing calculation tells the UAV which way to go. It is defined as the angle measured horizontally from north to the current direction of travel (Stefan 2000). With the great circle distance calculated as in Equation 1, the bearing c is:

$$c = \text{acos}\left(\frac{\sin(\Phi_2) - \sin(\Phi_1) \text{ x } \cos(d)}{\cos(\Phi_1) \text{ x } \sin(d)}\right)$$

Equation 2- Bearing

The result of this bearing calculation must be qualified by testing whether or not $\sin(\lambda_2 - \lambda_1)$ is negative. If negative, the true course is determined by $360^o - c$. Otherwise, the UAV will end up at the waypoint but it will take the long way round the globe.

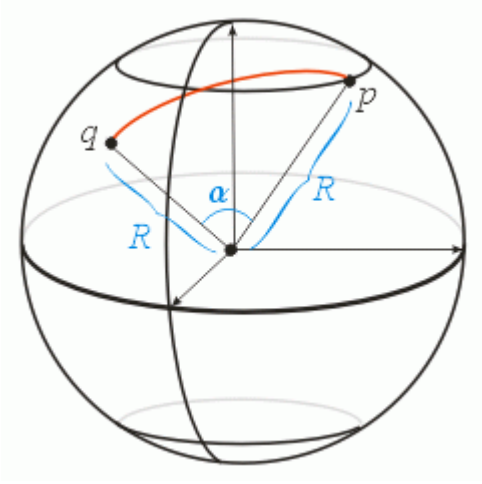


Figure 3- Great Circle Distance

Chapter 3

Image Capture and User Interface

Wong and Bil (2003) believe that UAVs developed for crop monitoring (surveillance capabilities) represent 80% of the market potential for UAV civilian applications within Australia. Presently only 10% of crops are monitored per annum by manned aircraft. A project carried out by Jensen, Apan, Young, Zeller and Cleminson (2003) in the Toowoomba area identified the feasibility of carrying out crop observations with the use of a remote controlled aircraft. They recognised that previous research on crop observations using satellites and aerial imagery presented limitations including repeatability, cloud cover, cost and poor spatial resolution. By using a radio controlled aircraft capable of flying at low altitudes, these limitations are greatly reduced.

3.1 Image Capture

UAVs have been used for surveillance and crop monitoring throughout the world. In 2002, NASA's solar powered UAV was used to conduct a proof-of-concept mission above the 1500 ha plantation of the Kauai Coffee Company in Hawaii. High resolution colour and multi-spectral imaging payloads were used in conjunction with a local area network (LAN) using unlicensed radio frequency for camera control and imagery downlink. The colour images collected were useful for mapping invasive weed outbreaks and for revealing irrigation and fertilisation anomalies (Herwitz et al. 2003).

The Swedish Defence Research Agency has conducted extensive research into military UAV surveillance applications including navigation-aided image processing, such as

Simultaneous Localisation and Mapping (SLAM), target tracking, and detection of moving objects (Nygards, Skoglar & Ulvklo 2003). Clearly there is an identifiable market for UAV surveillance and image processing capabilities. The challenge is to set up surveillance capabilities with appropriately high resolution while keeping costs to a minimum.

The first step in ensuring cost is kept to a minimum is to avoid the use of real time image processing. Real time image processing requires the image capture system on board the UAV be capable of sending its images directly (via LAN or RF) to the ground control station laptop. This relies not only on a reliable data link between the UAV and ground control station, but also a data link with a high enough bandwidth to enable image transfer with the required resolution. The bandwidth requirements for digital images and compressed video range in transmission rate from 1 to 10 Mbits/s (Sheldon 2001). This is due to Hartley's law that states that the greater the bandwidth, the greater the information that can be transferred from source to destination (Beasley & Miller 2005). Unfortunately, the larger bandwidth required means a greater cost of the transceivers, so it wasn't possible to include real time control capabilities in this project.

The simplest method to incorporate an image capture system is to include a light weight digital camera on-board the UAV. The digital camera requires high enough resolution to enable image capturing of crop details as well as zoom capability to ensure that the altitude of the UAV does not adversely affect the picture quality. The next issue of concern is how to trigger the camera while in flight. This can be done through mechanical or electrical triggering. i.e. servos can be used to manually press the shutter button, or the camera can be modified to include an electrical trigger unit that bypasses the shutter button. An electrical trigger unit was considered preferable as it avoids unnecessary vibration caused by mechanically moving parts, and also any electromagnetic interference (EMI) that may be caused by the servos. A digital camera without automatic shut off capability is also desirable as this can cause the camera to switch itself off after a certain amount of time and as such, would require the use of a more complex bypass circuit be used.

The memory capability of the camera is a limiting factor in the number of images taken during flight. A program is included in the navigation microprocessor to trigger the camera to take photos at some predetermined location or time, i.e. to take a photo either

at certain time intervals, or at waypoint locations. In this case, the system will take photos at pre-determined positions. The images can be downloaded to the laptop once the UAV has landed. A laptop can be used to view the images taken during flight.

3.2 User Interface and Communications

The purpose of the human-UAV interface is to provide a familiar platform for the user to enter flight path waypoints, as well as providing a visual indication of real-time UAV status during flight. Images captured during flight can be viewed via an appropriate interface. A laptop is an obvious choice for use as part of the Ground Control Station (GCS) due to its portability. The interface must be easy to use and provide adequate information without overwhelming the user. Whilst there are impressive user interfaces being used with UAVs that include real-time control of the UAV and the capability to view images taken during flight while the UAV is still in the air, it was decided that the most basic level of control would be implemented. That is, the GCS is used as a receiver of telemetry once the UAV is in the air with no capability of varying the flight path during any surveillance mission. This ensures that the UAV's flight path activity does not rely on the integrity of the communications link to carry out any given mission. The user interface is designed to ensure that the user is not required to have any level of UAV technical expertise to understand the functions and status indications. As such, a dial compass and heading error indicator have been included for ease of reading the incoming data, and as a way of emulating the aircraft cockpit in an attempt to add an instinctive feel to the viewing of UAV status.

The programming language used to design the graphic user interface (GUI) is Java. C++ was considered as an option but it was the re-usability and cross platform compatibility of Java that made it the best choice. Future improvements could see the inclusion of third party Geographical Information System (GIS) software that may provide real-time monitoring of geographical areas if used in conjunction with real-time imagery and map viewing capability. The GCS allows the user to enter the required flight path waypoint information as well as keep abreast of GPS status, UAV position, communication link status, UAV heading error, altitude and current heading.

The communications downlink to the GCS must be capable of reliably transmitting UAV telemetry. Real- time images taken during flight could be transmitted to the GCS but this was considered beyond the scope of this project due to the large communication bandwidth required to transmit and receive good quality, high resolution digital images. As such, image viewing is not included in the GUI design.

Chapter 4

System Architecture

Based on the literature review carried out, the hardware available and the project specification, the chapter presents the overall system design concept and architecture as shown at the highest level in the system development methodology of Chapter 1. A block diagram of the system is shown in Figure 4. The details of the design and testing of these systems is discussed in depth in the following chapters.

4.1 Original Concept

Before UAV takeoff, the Ground Control Station (GCS), consisting of a laptop PC (Dell Inspiron 510m), GUI and XStream 900MHz Transceiver will be connected to the UAV via a RS232 wireless link. Once connected, flight path information consisting of four latitude and longitude waypoints will be entered into the GUI, where they will be sent to the 'Navigation HC12' on the UAV. The 'Navigation HC12' microprocessor is connected to the Garmin GPS 35 LP TracPak™ antenna/receiver, the V2X 2-axis compass module, an electronic trigger unit and Minicam TDC-32 camera and XStream 900 MHz Transmitter.

Programmed into EEPROM of the 'Navigation HC12' is the 'Waypoint Path Planner Algorithm', which uses the user-inputted waypoints, GPS and compass signals to determine the distance and heading error to the next waypoint. This information is then ready to pass to the Automatic Flight Control HC12 (Littleton 2005).

The UAV is controlled by a user during takeoff, on a separate radio-controlled communications channel. Once in the air, the UAV is switched to autonomous flight at which time the 'Navigation HC12' commences provision of heading error information from current position to the first waypoint. The 900MHz transceiver connected to the HC12 will downlink telemetry information to the GUI including latitude, longitude, altitude, satellite and receiver status, heading error and compass heading information.

The camera is electronically triggered once over every waypoint by the 'Camera Trigger Program Module' programmed into the 'Navigation HC12' EEPROM. Once all four waypoints have been passed within a five metre radius, the UAV control is switched back to remote-control by the user, where it can then be landed. Surveillance photos can then be accessed on the GCS (serial cable connection).

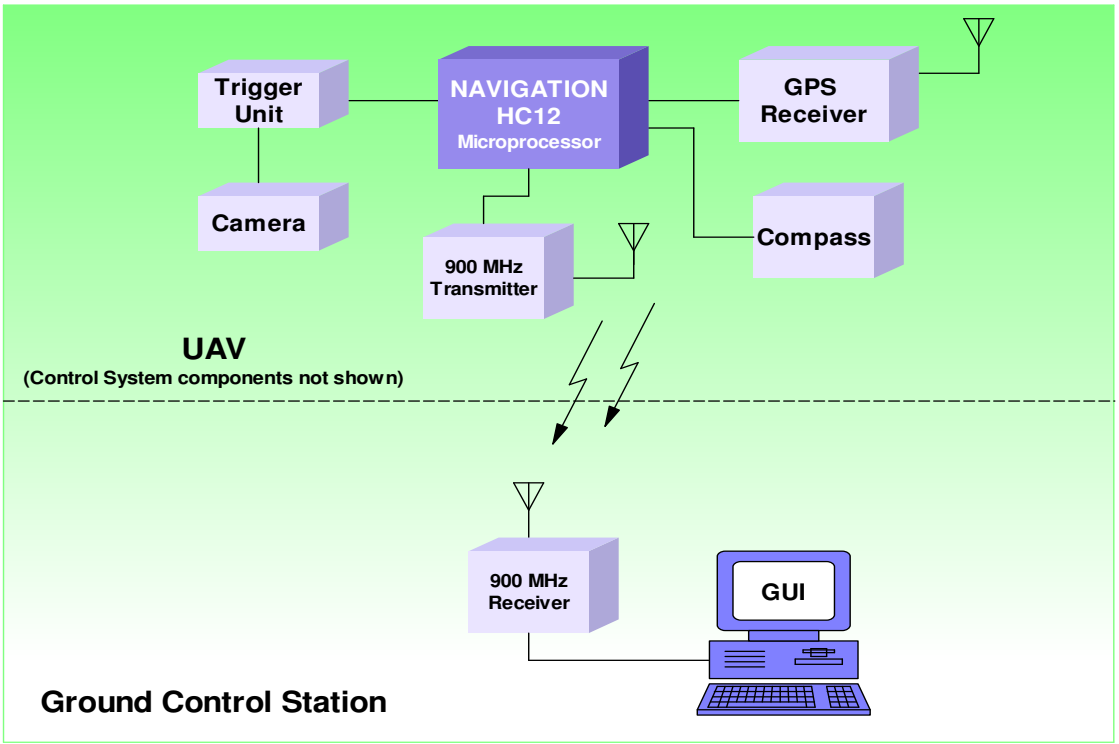


Figure 4- System Block Diagram

4.2 Resource Analysis

To build a prototype autonomous UAV navigation and user interface, there are many components and resources required to fulfil the objectives up to and including the final UAV system integration. The resources used include supporting personnel: my supervisors and Engineering faculty technical staff. The university provided a room and equipment to carry out all testing and the provision of miscellaneous parts: soldering equipment, power supplies, cables, passive components, oscilloscope and multimeters. The library, Institute of Electrical and Electronics Engineers (IEEE) and Engineers Australia have provided invaluable educational resources. The hardware resources selected for use in this project are shown in Table 1. The prices for components that were not bought specifically for this project have not been included.

Table 1- Hardware Resources Used

Component	Brand	Price or Provider	Ease of Use
GPS 35- HVS	Garmin TrakPak	USQ	
Compass	Vector 2X	$155	
Microcontroller	M68HC12D60	USQ	
Laptop	Dell Inspiron 510m	Soz	
Digital camera	Minicam TDC-32	USQ	
Electronic Trigger Unit	Custom Built	USQ	
Transceivers	XStream 900MHz	$510	

Excellent
Good
Average
Poor
Bad

The software resources selected for use in this project are shown in Table 2.

Table 2- Software Resource Used

Language	Package	Price or Provider	Ease of Use
C	Microsoft Visual C++	Soz	
JAVA	NetBeans	Soz	
C compiler	Imagecraft ICC12	Demo	
Card 12 Assembler	TwinPEEKs Monitor	USQ	

The total cost of resources bought specifically for this project was $665. A test trolley was manufactured out of recycled materials. The model aircraft currently available for use is the GW/SlowStick. The specifications for this remote controlled aircraft are shown in Table 3. This aircraft was used in Michelle Keeffe's project in 2003. It was chosen because of its simplicity, low cost and ease of use. Whilst it is an inherently stable aircraft, its payload capabilities have meant that the complete navigation and guidance system has not been installed on the UAV. Model aircraft alternatives were investigated but it became clear that the cost of a new aircraft with higher payload capabilities was not feasible this year. All other resources required were available from early on in the life of the project.

Table 3- GW/SLOWSTICK aircraft

Characteristic	Value
Length	954 mm
Wing Span	1176 mm
Wing Area	32.64 dm^2
Flying Weight	405~440 g
Wing Loading	12.4~13.5 g/dm^2
Battery Required	11.1 V lithium polymer
Power System	brushless motor 1200mAh
Radio Required	2~4 Channel Radio

Full integration of the system into a UAV requires appropriately stable, non-vibrating platforms for the protection of the avionics equipment. These resources were not fully

investigated as they depended very much on the UAV for which the system would be integrated. A suitable remote control aircraft was not available, so these types of protection were not designed. A custom made test trolley was manufactured to carry out testing of the Navigation and User Interface system. This trolley is a basically a plank of wood with four wheels, a GPS tower and a broom like handle for pushing. The final navigation circuit was placed in a protective housing to keep it free from dust.

The need for an experienced remote control aircraft pilot was identified during the risk assessment carried out in Chapter 5. Any risk to personnel and equipment due to inexperience was considered unacceptable. Therefore, any final prototype testing of the system in a UAV was to be carried out by an experienced pilot for take-off and landing, and that the pilot be available in case automatic control needed to be switched back to manual control during flight.

Chapter 5

Risk Assessment

Aviation activities are inherently dangerous. Whilst UAVs offer a safe alternative to piloted aircraft by avoiding the necessity for personnel to be exposed to the risk of flying, they then require completely reliable autonomous and/or remote control to ensure the UAV does not represent a risk to people, equipment or structures on the ground. The risk of aircraft failure in a general sense is not acceptable due to the severe consequences. For this reason, all general aircraft are designed with redundant flight critical systems. This ensures that if a system required for safe flight fails, a back-up system will take over with no reduction in performance. This enables the mission to be completed or safe landing of the aircraft and as such, safe return to ground of all personnel and equipment. This project has been carried out with safety as a major priority, and a risk management plan was implemented early on to ensure all risks were mitigated where necessary.

The risk management process adopted by this project follows the methodology outlined in AS/NZS 4360:2004. Its main elements are shown in Figure 5 below. The project context has been outlined in Chapter 1.

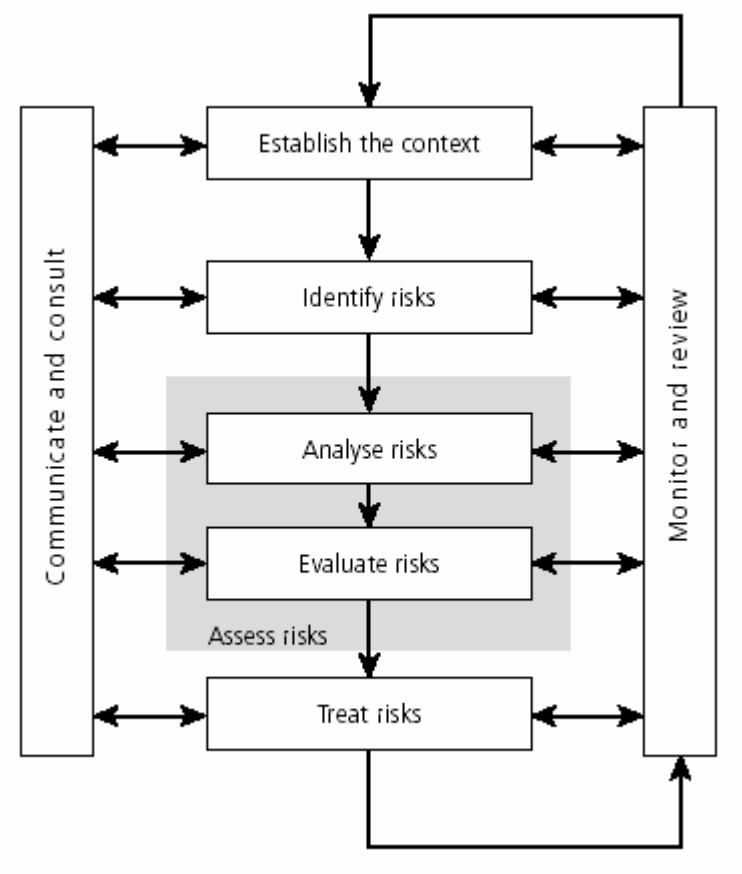


Figure 5- Risk Management Process Overview

5.1 Identifying Risks

Risk identification is concerned with 'What can happen and how?'. Once identified, risks can be actively managed by the risk management process. Like all risk management activities, risk identification was ongoing during the life of the project and was not limited to those identified early in the project life.

5.2 Evaluating Risks

A qualitative method for evaluating the level of risk posed by an event has been adopted for this project. The risk analysis consists of first assessing the *likelihood* of the event happening and the potential *consequences* should that event occur. Criteria for grading likelihood and consequences are provided in Table 4 and Table 5, respectively.

Table 4- Likelihood Rating Criteria

LIKELIHOOD RATING	DESCRIPTION
Rare	May occur only in exceptional circumstances
Unlikely	Could occur at some time
Moderate	Might occur at some time
Likely	Will probably occur in most circumstances
Almost Certain	Expected to occur in most circumstances

Table 5- Consequence Rating Criteria

CONSEQUENCE RATING	DESCRIPTION
Insignificant	Inconvenience. Minimal or no impact.
Minor	• Small but acceptable degradation in performance.. • Unable to meet an intermediate milestone but able to meet all major milestone dates.
Moderate	• Tolerable but significant degradation in performance. • Unable to achieve one major milestone date.
Major	• Significantly degraded performance to the point where project success is questionable. • Unable to achieve more than one major milestone date.
Severe	• Unacceptable performance resulting in project failure. • Unable to achieve acceptance milestone date.

The likelihood and consequence grading are combined to form a single risk level matrix shown in Table 6. This matrix has been taken from the RAN Aviation Safety Manual (ABR 5147 Annex C), and is the standard Department of Defence risk matrix. The risk levels have been categorised as:

LOW. The risk event, were it to occur, may have some minor undesirable consequences for the project, but little effect on its perceived or actual overall success;

MEDIUM. The risk event, were it to occur, would be likely to have some undesirable consequences for the project, but would be unlikely to cause it to fail;

HIGH. The risk event, were it to occur, would have a significant impact on the perceived or actual success of project, and could even cause the project to fail; and

EXTREME. The risk event, were it to occur, would probably cause the project to fail, or give rise to unacceptable circumstances (such as frequent adverse user acceptance and operational failure) attributable to the project.

Table 6- Risk Level Matrix

Consequence / Likelihood	*Insignificant*	*Minor*	*Moderate*	*Major*	*Severe*
Almost Certain	**Medium**	**Medium**	**High**	**High**	**Extreme**
Likely	**Medium**	**Medium**	**Medium**	**High**	**Extreme**
Moderate	**Low**	**Medium**	**Medium**	**High**	**High**
Unlikely	**Low**	**Low**	**Medium**	**Medium**	**High**
Rare	**Low**	**Low**	**Low**	**Medium**	**Medium**

5.3 Risk Control

By analysing the risks identified in accordance with the above mentioned matrix, applicable measures of control have been designed and implemented where possible. A full risk assessment is included in Table 7. The risk assessment includes sources of risk, their potential consequences and the likelihood that those consequences will occur. Risk control measures are included. These risks have been monitored and reviewed through the life of the project.

Table 7- Risk Assessment

Risk Description	Likelihood	Consequence	Risk Level	Risk Control Measure
Risks to Project Completion				
Limited access to resources	Moderate	Major	**High**	Assess resource requirements early to ensure availability
Breach timeline, schedule or major milestones	Moderate	Major	**High**	Allocate extra time for all activities
Risks to Personnel and Equipment				
Provision of incorrect heading information	Moderate	Severe	**High**	Provide adequate shielding to the compass to prevent magnetic anomalies
GPS satellite information drops out	Likely	Major	**High**	Ensure all flights are carried out in an area clear of obstruction and on a clear day- design an emergency back-up system to inform of satellite drop-out and allow UAV control to be taken over by remote control user
Loss of telemetry downlink signal	Moderate	Moderate	**Medium**	Ensure the UAV is flown within the range of the transceivers
Incorrect entering of waypoints	Moderate	Moderate	**Medium**	Repeat user-entered information to the user and verify the latitude and longitude points
Power failure , i.e GPS and compass data lost- loss of camera function, loss of HC12 function-	Moderate	Moderate	**Low**	Check all power supplies are fully charged and serviceable prior to flight- perhaps include backup
Power failure , i.e laptop GCS	Unlikely	Insignificant	**High**	Ensure the laptop battery is fully charged prior to any flight
Collision with person	Moderate	Major	**High**	Ensure all flights are carried out in areas clear of any unnecessary persons
UAV gets in the wrong hands and is used as a weapon	Rare	Severe	**Medium**	Ensure the UAV is only used by those involved in the project and for the purposes outlined in the objectives

Unskilled pilot used for take-off landing and for emergency control of UAV	Likely	Moderate	**Medium**	Ensure any pilot is experienced and that all CASA guidelines are followed
UAV is flown above 400ft	Moderate	Moderate	**Medium**	Ensure the operator of the UAV is aware of the CASA regulations that limit altitude.

Chapter 6

Navigation System Hardware Design and Implementation

This chapter describes the selected navigation and guidance system hardware and how it was developed. The components selected for use are those shown in the system architecture block diagram in Chapter 4. They are now discussed in detail. The requirements and limitations as outlined in Chapter 1 have been analysed to ensure any design falls within these guidelines. Development tools employed throughout the project for successful design and implementation of components is detailed. The testing of each of the components is described along with any interface considerations necessary for successful integration.

6.1 Flight Computer

The brain of the Navigation and User Interface system is the 68HC(9)12D60 microcontroller unit. It is a 16 bit member of Freescale's HC12 Microcontroller family and offers additional features that aid in the development process. In particular, the microcontroller card is equipped with the TwinPEEKS monitor program which allows easy download and programming of Flash and EEPROM memory, without the need for any additional hardware tools (Elektronikladen 2005). Other features include:

- 60 KB Flash

- 1 KB EEPROM
- 2 KB RAM
- SPI, 2 x SCI
- Enhanced Capture Timer
- 4 Channel Pulse Width Modulator
- 16 Channel 10 bit A/D Converter
- 8 MHz operation at 5V
- CAN 2.0A/B bus interface with CAN driver

Appendices B and C show the block diagram and schematic of the 68HC(9)12D60. The HC12 has been labelled 'Navigation HC12' to differentiate it from the HC12 used in the Automatic Flight Control portion of the prototype UAV (Littleton 2005) The 68HC(9)12D60 is shown in Figure 6.

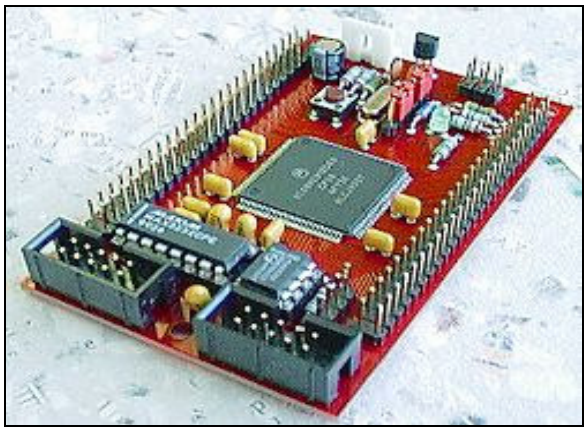


Figure 6- Card 12

The port used for the majority of the hardware interfacing is Port S. The Navigation HC12 uses its two asynchronous serial communication interfaces (SCIs) and one synchronous peripheral interface (SPI) as shown in Figure 7.

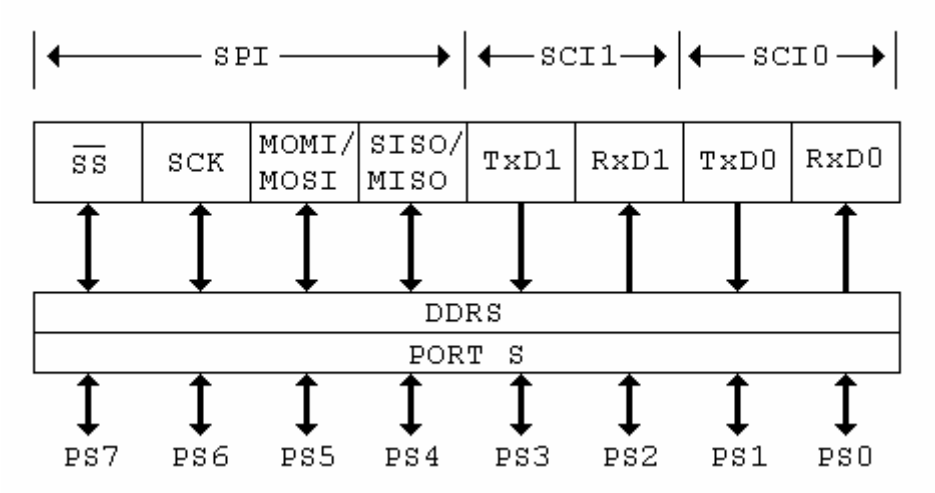


Figure 7- HC12 Port S

The GPS is connected to SCI1 and the compass is connected to the SPI. The Ground Control Station laptop is connected via SCI0 through the 900 MHz transceiver used for uploading the initial user-selected waypoints and then for telemetry downlink of UAV status whilst in flight. The CAN bus has been set up to send data from the Navigation HC12 to the Automatic Flight Control HC12. The input/output ports A and B are used for controlling the compass and camera trigger module where necessary. These ports are controlled by assigning them either an input or output by writing a 1 (HIGH) or a 0 (LOW) to the corresponding bit of the data direction register (DDRA/B). On reset, DDRA and DDRB are set to $00 which means they are inputs by default. For example, to make bits 3-0 of PORTA inputs, and bits 7-4 outputs, it is necessary to write 0x0f to DDRA. Then, to send data to the output pins, it is as simple as writing directly to PORTA. When reading from PORTA, input pins will return the value of the signals on them (0 = 0V, 1 = 5V); output pins will return the value written to them. A block representation of PORTA and its data direction register are shown in Figure 8.

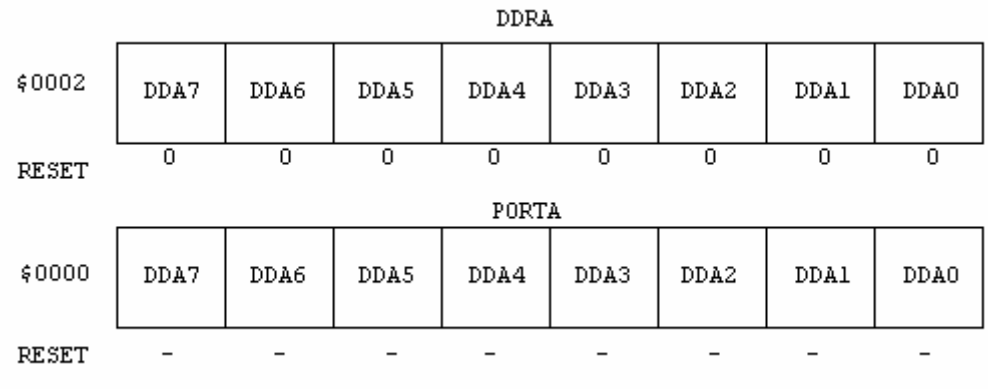


Figure 8- DDRA and PORTA

The HC12 has capabilities above and beyond the necessities of this project, but its ease of use and processing power offer a platform for a reliable embedded navigation system for the prototype UAV. The navigation and guidance 'Waypoint Path Planner Algorithm' and 'Camera Trigger Module' are programmed into the EEPROM and the user-entered waypoints are stored into RAM for every mission to be carried out. The 'navigation HC12' is programmed using C language for two reasons. Firstly, a suitable HC12 C-compiler demo was available for use, and secondly, C is an easy and simple programming language to carry out the tasks required. Assembly language programming was not considered viable due to the complexity of the trigonometric functions to be used in the navigation algorithm. Details of program implementation are discussed in Chapter 8.

6.1.1 SPI

The Serial Peripheral Interface (SPI) is used primarily for synchronous serial communication between a host microprocessor and peripherals. It can also be used to connect two HC12 microprocessors together, but this feature has not been used in this project. A data byte can be shifted in and/or out one bit at a time. The SPI in the HC12 contains the four necessary signals for SPI communication. Two SPI modules connected in a generic Master-Slave configuration are shown in Figure 9.

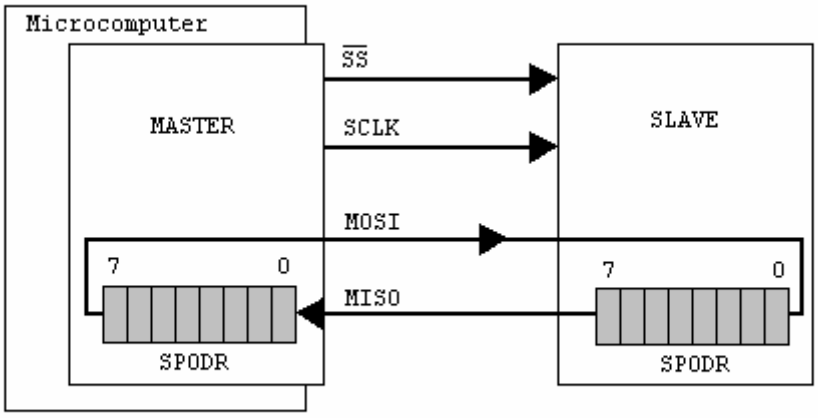


Figure 9- HC12 SPI configuration

In the master SPI, the bits are sent out of the MOSI pin and received in the MISO pin. The bits are shifted out and stored in the SPI data register, SP0DR, and are sent out most significant bit first (bit 7). When bit 7 of the master is shifted out through the MOSI pin, a bit from bit 7 of the slave is being shifted into bit 0 of the master via the MISO pin. After 8 clock pulses or shifts, this bit will end up in bit 7 of the master. In the HC12 the least significant bit is sent out first by setting the LSBF bit to 1 in the SPI Control Register. The clock which controls how fast the bits are shifted out and into the SP0DR, is the signal SCLK at PS6. The frequency of this clock is controlled by the SPI baud rate register, SP0BR. The SS pin must be low to select a slave. This signal can come from any pin on the master, including its SS pin when it is configured as an output. The SPI registers in the HC12 are shown in Table 8.

Table 8- HC12 SPI registers

Name	Register Address	Description
SP0CR1	00D0	SPI Control Register 1
SP0CR2	00D1	SPI Control Register 2
SP0BR	00D2	SPI Baud Rate Register
SP0SR	00D3	SPI Status Register
SP0DR	00D5	SPI Data Register

SPI transmission is always initiated by the master with the applicable peripheral device referred to as the slave. SCLK has eight different frequencies that can be selected depending on the application and the timing requirements of the slave peripheral. These frequencies are selected with bits SPR2: SPR0, as indicated in Table 9.

Table 9- SPI clock rate selection

SPR[2:0]	Divisor	Frequency
000	2	4.0 MHz
001	4	2.0 MHz
010	8	1.0 MHz
011	16	500 KHz
100	32	250 KHz
101	64	125 KHz
110	128	62.5 KHz
111	256	31.25 KHz

The details of SPI communication implementation are discussed in Chapter 8, which describes the program necessary to initiate SPI communication with the compass.

6.1.2 SCI

The 68HC12 has two Serial Communication Interfaces, each providing a Transmit Data and a Receive Data signal suitable for a RS232 interface. Each SCI can be configured for eight or nine data bits, the most significant can be configured as an even or odd parity bit which is generated automatically on transmission and checked on reception. There is always a single stop bit. Transmission data rates can be selected independently for each SCI, however, the transmit and receive rate for a single SCI must be the same (Almy 2004). The SCI1 connection to the GPS is configured for reception of data only. The SCI0 connection to the Ground Control Station is set up for reception of data to allow for the user-entered waypoints to be downloaded to RAM on start up. The SCI0 connection is then configured (via software) for transmission of data to enable telemetry downlink to the Ground Control Station. Whether transmitting or receiving, the following process must be used to initialise communications with the SCI and its peripherals:

1. The baud rate must be set. If the HC12 SCI baud rate is not selected to match the baud rate of the peripheral it is connected to, the transmission and reception of data will not be reliable if it in fact works at all. SCIxBDH and SCIxBDL are

two 8-bit registers considered together as a 16-bit baud rate control register. Table 10 details the necessary values to be set in the SCI Baud Rate Control Register. This table follows the following formula for calculating the necessary enterable Baud Rate (BR).

$$BR = \frac{MCLK}{16 \text{ x } SCI\ Baud\ Rate}$$

Equation 3- Baud Rate Calculation

BR is then the value written to bits [12:0] of SCIBDH.

Table 10- Baud Rate Generation

Desired SCI Baud Rate	BR Divisor for MCLK = 8 MHz
110	4545
300	2273
600	833
1200	417
2400	208
4800	104
9600	52
14400	35
19200	26
38400	13

The GPS defaults to 4800 baud. This means that to receive data from the GPS, the SCIBDH gets set to 104 decimal. The Ground Control Station can communicate at a variety of baud rates as dictated by the laptop PC. The baud rate selected and set on both the laptop and SCI0 is 9600 baud and as such, the SCIBDH is set to 52 decimal.

2. The next step is to configure the SCI control registers for the desired SCI parameters, i.e. transmission or reception or both. In each SCI connection, SCxCR1 has been set to all zeros. This sets up the SCI to transmit and receive normally with both high and low drive capability and a character format mode

of one start, eight data and one stop bit. SCxCR2 is set to 12 decimal to enable transmit and receive anytime.

3. The next step requires the SCI status register be polled to check for either a Transmit Data Register Empty Flag (TDRE = 1) or a Receive Data Register Full Flag (RDRF = 1). When transmitting data, once TDRE = 1, the SC0DRH and SC0DRL data registers may be written to for sending data to the applicable peripheral (in this case the Ground Control Station). When receiving data, once RDRF = 1, the SC1DRH and SC1DRL may be read from.

6.1.3 CAN

The Controllable Area Network (CAN) is a serial communication protocol that supports distributed real-time control applications. The bus in this project is used as the connection between the 'Navigation HC12' and the 'Automatic Flight Control HC12' for the transmission of heading error and altitude.

Though conceived and defined by BOSCH in Germany for automotive applications, CAN is not restricted to that industry. The CAN protocol fulfills the communication needs of a wide range of applications, from high-speed networks to low cost multiplex wiring (Huang 2003).

The Freescale Scalable CAN module (MSCAN) is an advanced communications controller, implementing the CAN protocol, with these features as described in the Motorola Advanced Information Booklet (2000):

- Implementation of CAN version 2 parts A and B
- Standard (11-bit) and extended (29-bit) data frames
- 0 to 8 bytes data length
- Programmable bit rate up to 1 Mbps
- Support for remote frames
- Double buffered receive
- Triple buffered transmit with internal prioritization using a "local priority" concept

- Flexible maskable identifier filter supports alternatively two full size extended identifier filter, four 16-bit filters, or eight 8-bit filters
- Programmable wakeup functionality with integrated low-pass filter
- Programmable loopback mode supports self-test
- Separate signaling and interrupt capabilities for all CAN receiver and transmitter error states (warning, error passive, bus-off)
- Programmable MSCAN clock source (either the CPU bus clock or the crystal oscillator output). Low-power sleep mode.

CAN is divided into data link and physical layers. The physical layer defines how the signals are actually transmitted; bit timing, bit encoding and synchronization.

Information on the CAN bus is sent in fixed formats of different but limited lengths. When the bus is free, any connected node may start to transmit a new message. For the application required in this project, the only nodes connected will be the automatic flight control system and the Navigation and User Interface systems. The Navigation and User Interface system will transmit heading error and altitude in the required format (as a character array) onto the CAN bus so that the Flight Control System can access it when ready. The CAN system does not specify node addresses in a message. Instead, it uses an identifier to describe the content of the message. The identifier does not indicate the destination of the message; instead, the meaning of the data is described. The nodes connected to the bus network are able to decide by message-filtering whether the data is to be read by them or not. This feature is not required in this application as the Automatic flight control system will always be the recipient of the transmitted messages and as such, these filters have been disabled in the software implementation described in Chapter 8.

6.2 Development Tools

The development tools that were paramount for the successful implementation, testing and debugging of the system were the TwinPEEKS monitor program and ImageCraft's ICC12 integrated development environment.

6.2.1 TwinPEEKs Monitor Program

The TwinPEEKs monitor program is a ROM Monitor utility which supports uploading of user program code to the on-chip EEPROM of the HC12. The Monitor Program is able to display and modify the memory contents of the HC12 and can also be used to commence execution of the user program in real-time.

The Monitor occupies 2 KB of EEPROM Code space leaving 2 KB for user program code. The Monitor is pre-programmed into the EEPROM of the 68HC(9)12D60 development board so the Monitor Program can communicate with the Monitor when the board is powered up.

TwinPEEKs allowed the programming of the final navigation solution to be uploaded to the HC12 EEPROM in assembly language source file code, as is produced by the ImageCraft compiler discussed below. The memory map of the 68HC(9)12D60 is shown in Table 11.

Table 11- 68HC(9)12D60 Memory Map

$0000 - $01FF	MCU Control Registers
$0200 - $07FF	2 KB RAM
$0C00 - $0FFF	1 KB EEPROM
$1000 - $7FFF	28 KB (lower) Flash Memory Array
$8000 - FFFF	32 KB (upper) Flash Memory Array Boot Block (containing Monitor code): $E000 - $FFFF

As shown in Table 11 the Monitor code resides in memory address $E000 - $FFFF. After reset the Monitor checks whether pins PH6 and PH7 on the HC12 are connected or not. Without a connection (default setting), the Monitor jumps to the address of the Monitor code mentioned above. If there is a connection, the Monitor jumps to address

$8000. In this way it is possible to automatically start a user program without changing the Reset Vector, which resides in the write-protected Boot Block area.

6.2.2 ImageCraft ICC12 V6

The ImageCraft C ICC12 Development Environment is a program for developing HC12 microcontroller applications using the ANSI standard C language. Its main features are:

- An intuitive Windows 95/NT native Integrated Development Environment (IDE) with integrated editor and project manager. Source files are organized into projects. Editing and building can be done wholly within the environment. Compile time errors are displayed in the status window, and with a click of the mouse button, you can jump to the lines that cause the errors in the editor window. The integrated project manager generates a standard makefile that you can view and use directly if desired.

- The IDE drives an ANSI C command line compiler that is normally transparent in operation. However, if you wish, you can interact with the compiler directly using the command prompt program. The compiler is a set of native 32-bit programs and understands long file names.

ICC12 enables the development, compilation, linking and debugging of C source code and produces an assembly language source file version of the C source code that is executable on the HC12 microprocessor. The benefits of this software cannot be overstated. The amount of time and effort required in algorithm implementation and hardware integration has been greatly reduced by avoiding the need to develop assembly language source files from scratch.

6.3 Sensors

The navigation algorithm uses the Great Circle Distance formula to calculate the distance and bearing to the next waypoint. It requires current position information and

current heading information to then calculate the heading error (and thus amount of turn required) in order to ensure the UAV heads towards the next waypoint. The hardware design and integration issues of these sensors are now discussed with the software implementation discussed in Chapter 8.

6.3.1 GPS

The GPS used in this project is the GARMIN GPS 35 LP TracPak$_{TM}$, model GPS35-HVS. This product is a receiver with an embedded antenna that provides one second navigation updates and low power consumption. This GPS offers two RS-232 compatible full duplex communication channels and user-selected baud rates that allow maximum interface capability and flexibility. The GPS GGA message is used to provide position information to the system as well as telemetric information; altitude and GPS status to the Ground Control Station GUI. Data parsed from the GPS to the microprocessor is GPS status, time, latitude, longitude, and altitude. The altitude measurements are passed directly to the control system microcontroller without any manipulation. The GPS status is used to indicate to the Ground Control Station any loss in quality of the received GPS message. By including this feature, the high level risk identified in the risk assessment shown in Chapter 5 is mitigated. The user is informed of a loss of GPS signal quality, in which case the UAV control can be handed back to manual remote control for safe landing of the UAV. Whilst the GPS receiver offers reliable long term position information, its susceptibility to environmental interference means that the reliability of the GPS can only really be guaranteed on non-cloudy days away from any physical obstructions that may impose a shadow on the GPS receiver/ antenna. An extract from the GPS35-HVS specification detailing the NMEA sentences received is included in Appendix D. The following is a summary of the more pertinent features:

- Full navigation accuracy provided by the Standard Positioning Service (SPS),
- Compact design ideal for applications with minimal space,
- High performance receiver tracks up to 12 satellites while providing fast first fix and low power consumption,

- Internal clock and memory are sustained by a rechargeable memory backup battery. The battery recharges during normal operation. This redundant feature ensures the reliability of data received from the GPS.
- User initialisation is not required ensuring ease of use and an 'off the shelf' like solution.
- Two communication channels and user selectable baud rates allow for maximum interface capability and flexibility, and
- Flexible input voltage levels of +6.0 VDC to 40 VDC unregulated supply.

The GPS unit weighs 124.5 grams (GARMIN Corporation 2000). This represents a major drawback for integration of the final system into a prototype UAV because of the payload limitations imposed by the smaller, less powerful remote control aircraft available. The GPS unit represents the majority of the weight of the final Navigation and User Interface system.

Interfacing the GPS with the HC12 required a surprisingly small amount of effort. The 35-HVS comes as a complete GPS receiver and antenna in a waterproof packaged unit. Typical application architecture as outlined in the technical specification is shown in Figure 10.

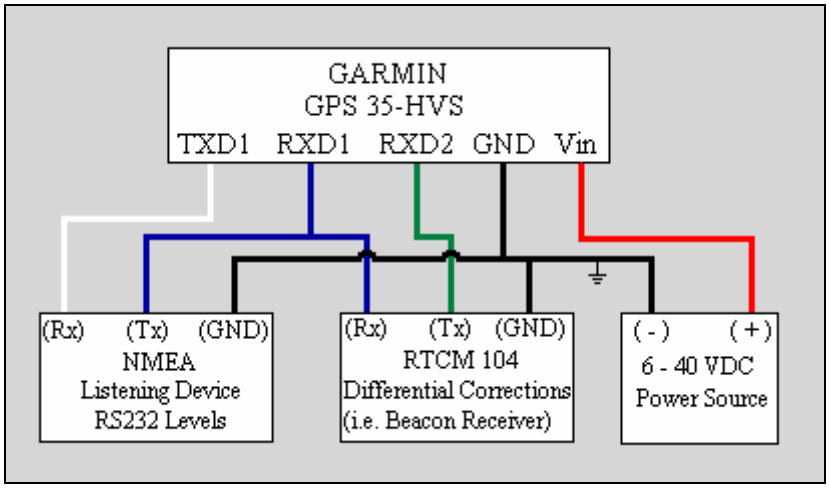


Figure 10- Typical GPS 35-HVS Application Architecture

The only wire used to connect the GPS to the HC12 is TXD1. The other wires connected to the system are the GND and Vin pins which are connected to a common ground and 12 VDC supply respectively. Fifteen seconds after power up, the GPS

begins data acquisition and this data is then sent to the HC12 SCI1, as discussed in section 6.1.2 of this chapter. The software implementation details are shown in the Navigation Algorithm code included in Appendix G.

6.3.1.1 GPS Testing

After analysis and high-level design of the hardware, testing of the individual components and their integration with the rest of the system was carried out. The functionality of the GPS and its interface with the HC12 was first tested by developing a small assembly language program to initialise SCI communications with the GPS. Through the TwinPEEKS monitor program running on a laptop, GPS sentences were captured into a text file, and the first successful interface verified. The assembly language program and raw captured GPS data are included in Appendices E and F.

6.3.2 Compass

The V2X Compass Module is used in this project to provide an increased update frequency of heading calculations, enabling the provision of heading error to the Automatic flight control system at a faster rate than is possible with only GPS. The V2X is a 2-axis magnetometer that measures the magnetic field in a single plane. This plane is the plane created by its two sensors, which are perpendicular to each other on the board (shown in red in Figure 11). The heading is calculated with respect to the front of the board. One limitation of the V2X is that the compass must measure the Earth's field in a plane that is level. This level is analogous to the plane parallel to the surface of a glass of water, i.e. when the glass of water tilts, the water's surface will remain level (PNI Corporation 1998). This means that if the compass is not level, the calculated heading will have errors related to the tilt of the board from level. Unfortunately, in a UAV application, there are times when the aircraft will tilt. Specifically, when the rudder of an aircraft is moved causing the aircraft to yaw, one wing will advance and the other will retreat. The faster moving wing produces more lift than the other which will cause a roll in the same direction as the yaw. This will mean that during any turn, there will be slight errors in the calculated heading. For the

purposes of the Navigation and User Interface system prototype, this error is not critical. In future prototypes, a gimballed compass could be used which will eliminate this tilt error.

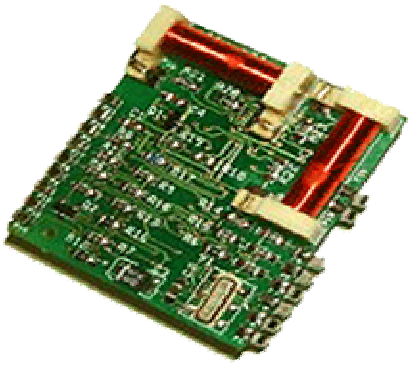


Figure 11- Vector 2x Compass Module

The V2X requires SPI communications and can operate in three different operating modes. These are master mode, slave mode or raw mode. For the purpose of the navigation algorithm application, the V2X is operated in slave mode to operate effectively with the HC12 in a master-slave relationship as outlined in section 6.1.1. In slave mode, the HC12 must clock the data out of the V2X and as such, must provide the clock (SCLK) as an input to the compass. The maximum rate of clocking out data is 1 MHz for reliable data acquisition. The output data format can be chosen to be in binary coded decimal (BCD), binary or raw output. BCD output format was chosen for this project. The algorithm for interpreting the data is shown in the compass.c program in Appendix G. The pin outs on the V2X are shown in Figure 12.

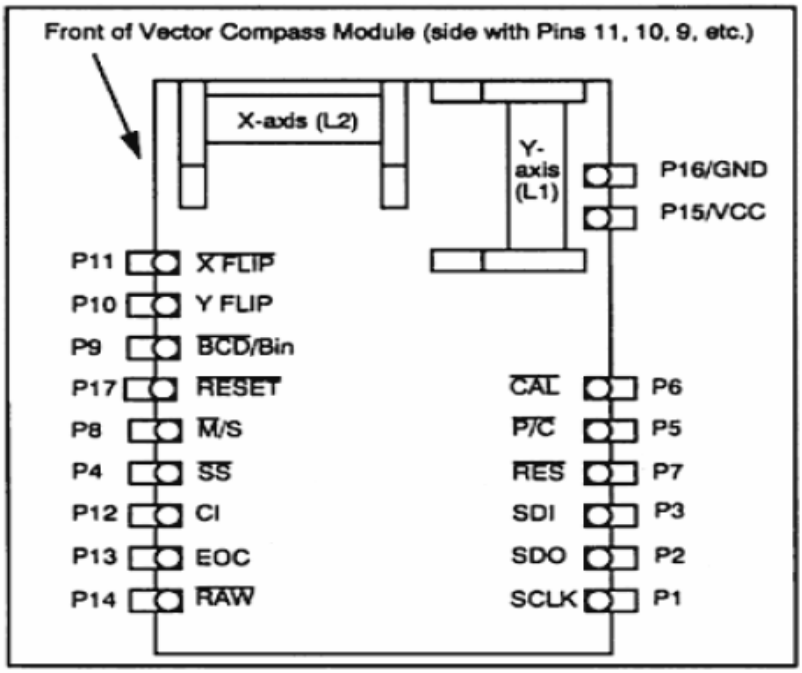


Figure 12- V2X Compass Board Layout

The functions of the pins used to interface the V2X to the navigation system are shown in Table 12.

Table 12- V2X Compass Pin Connections

Pin Name	Function	Setting
SCLK	Serial Clock	Input
SDO	Serial Data Output	Output
$\overline{SS}$	Slave Select	Input
$\overline{P}$/C	Poll/ Continuous	Input
$\overline{CAL}$	Calibration	Input
$\overline{RES}$	Resolution	Input- Low for low resolution
$\overline{M}$/S	Master enable/Slave enable	Input- High for slave operation up
$\overline{BCD}$/Bin	Data output format selection	Input- High for binary
Y FLIP	Flips the Y axis	Input- Low for normal setting
CI	Calibration Indicator	Output
EOC	End of calculation indicator	Output
VCC	Power	Connected to a common +5 V
GND	Ground	Connected to a common ground

The timing diagram that was followed to ensure successful data acquisition is shown in Figure 13.

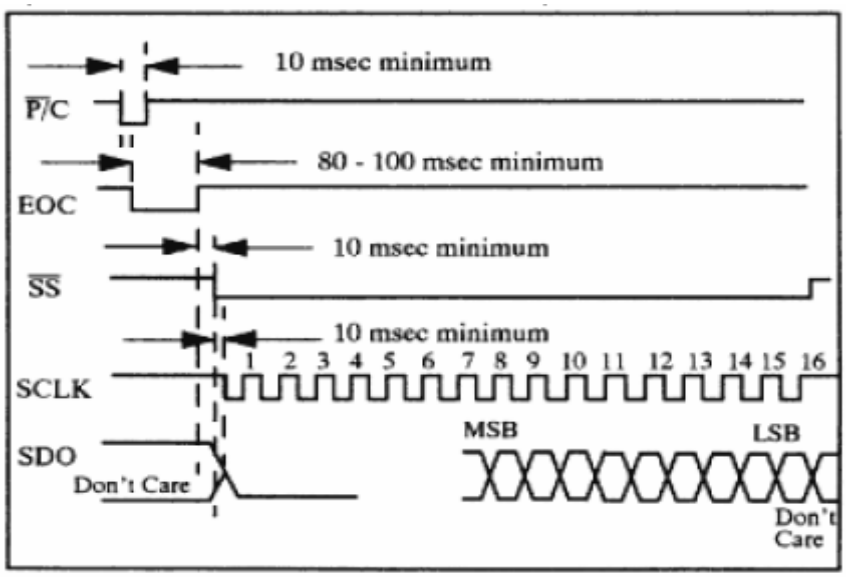


Figure 13- Data Clock Timing Diagram

This diagram shows the order of precedence in which signals must be controlled, and the necessary timing to ensure reliable data acquisition from the compass.

6.3.2.1 Compass Calibration

Due to the fact that the compass determines heading by sensing the Earth's magnetic field, external magnetic fields can become a source of disruption for the compass and therefore, produce incorrect heading data. By calibrating the V2X, the static magnetic fields in the UAV can be compensated for. This calibration is limited, in that it is not capable of compensating for any dynamic magnetic fields, but is useful for ensuring initially stable and reliable heading data.

The Calibration pin ($\overline{CAL}$) is used to calibrate the V2X. The steps carried out in order to calibrate the V2X as outlined in the Application Notes (PNI Corporation 1998) are:

- $\overline{P}/C$ must be high,
- Point the host system with V2X mounted in any direction,

- Toggle $\overline{CAL}$ low for 1 ms (must be high when not in calibration mode),
- Toggle $\overline{CAL}$ high again,
- Rotate the host system 180^0,
- Toggle $\overline{CAL}$ low for a minimum of 10 ms, and
- Toggle $\overline{CAL}$ high again (to remain there during normal operation) to complete calibration.

The V2X only has volatile memory and as such, when the power is removed, the calibration settings are lost. In order to ensure the settings are not forgotten, the host system can calculate and save the calibration settings. This feature was not used in this project because a permanent platform for the Navigation and User Interface system was not established. Calibration of the compass is, therefore, carried out after power up and prior to normal operation.

6.3.2.2 Compass Testing

Testing of the compass was carried out in conjunction with the software used to parse the data coming from the compass. This interface software is described in Chapter 8. The compass, HC12 and laptop were connected so that the output of the compass could be monitored. The compass was first placed in the 0^0 position (north facing). This position was confirmed by a handheld compass placed next to the V2X compass. By moving the V2X left and right, the changes in the compass readings were compared to those of the handheld compass. The readings were within 1^0 of the handheld compass. This result was better than the expected 2^0 of accuracy as outlined in the specification. The readings frequencies were as expected, with no sudden changes or lag in heading updates.

6.3.2.3 Mitigation of Incorrect Heading Provision Risk

The provision of incorrect heading information was identified as a risk in the risk assessment carried out in Chapter 5. The control measure identified at the time was to

provide adequate shielding to the compass to prevent magnetic anomalies. The best way to avoid magnetic anomalies in the compass is to make sure it is as far away from any power wires or motor leads that may induce electromagnetic fields. The compass placement in the final Navigation and User Interface prototype was made with these considerations in mind. A sealed acrylonitrile-butadiene styrene (ABS) case is used for housing the final circuit, because its non-magnetic properties make it useful for magnetic compass housings. Carrying out calibration of the compass alleviates the static magnetic anomalies caused by the surrounding system components and platform.

Chapter 7

Image Capture and User Interface Implementation

This chapter describes the hardware used to implement the image capture system to enable surveillance over the pre-determined path and the platform used as the basis of the user interface. The chapter details the Ground Control Station as laid out in the system architecture design of Chapter 4, then the final system integration as discussed in the overall hardware system design is shown.

7.1 Ground Control Station

The Ground Control Station (GCS) as the name suggests, is the part of the system that remains on the ground when the UAV is flown on a surveillance mission along a predetermined path. The GCS consists of a laptop, a transceiver and a Graphical User Interface (GUI). The transceiver was originally intended to be used as only a receiver with the downlink of waypoints to occur on the ground with a direct wire connection from the laptop to the HC12. The receiver was then going to be connected to the laptop and accompanying transmitter connected to the HC12 for downlink of telemetry data during flight. However, the need for connection and disconnection of the transceivers is negated by using the GCS transceiver as a transmitter while the UAV is on the ground for downloading the user-entered waypoints, and then using it as a receiver once the UAV is in the air. This reduces the amount of set up time for any given surveillance

flight and ensures that any possible hardware connection errors due to system configuration changes are avoided.

The laptop used for the GCS in this project is an IBM compatible Dell Inspiron 510m. This laptop was used due to its availability but because the GUI design has been implemented in Java, the GCS can be implemented on any computer that has the Java runtime environment. The only necessary requirements for the computer selected is that it has a nine pin RS-232 serial connection and that it be portable enough to carry to the site where the surveillance mission is to be carried out.

7.1.1 Data link

To enable communications between the in-flight navigation system and the GCS, a pair of wireless transceivers is used to download the user-entered waypoints from the GCS to the HC12. Once this is done, the GCS transceiver is used as a receiver for the down link telemetry. Conversely, the transceiver attached to the navigation HC12 is used as a transmitter.

The transceivers chosen are the MaxStream 9XStream 900 MHz Wireless OEM Modules. They use an RS-232 interface to communicate with the host system. Each transceiver has a transmission power output of 140 mW and operates at long ranges (11 km with a dipole antenna), due to their excellent receiver sensitivity of –110 dBm. Additionally, the 9XStream uses proprietary filtering technology to provide interference immunity, including 70 dB of cell phone and pager rejection (10 million times attenuation) (MaxStream 2005).

The simplicity of these transceivers makes them a perfect solution for the UAV wireless needs. They are a light weight, ‘plug and play’ solution with no configuration required. Whilst small modifications were required to enable the connection of power leads to the batteries used for testing, the transceivers interfaced perfectly with the system as though a direct wired link were being used. The transmitter is connected to SCI0 of the HC12 and the receiver is connected to the laptop RS-232 connection on the laptop (GCS). The relative size of one of the modules is shown in Figure 14 (MaxStream 2005).

Figure 14- 9XStream 900 MHz Wireless Module

7.1.1.1 Transceiver Testing

The transceivers were tested to ensure that the range required, as laid out in the performance requirements of Chapter 1, could be met. Specifically, the transceivers were required to have a range of 500m as the UAV was to be flown no more than 500m from the GCS. This range falls within the expected 11km as detailed in the transceiver specification. The validation of the transceivers' transmission was considered necessary in spite of the requirements falling well within the published limits, to ensure the safety of equipment and personnel during any surveillance mission.

The GCS was set up with one of the transceivers and a power supply. The laptop was equipped with the X-CTU software provided with the transceivers. This software is a basic terminal program with the ability to carry out range testing. A null modem adapter was connected to the other transceiver which was powered by a portable battery. The null modem adapter allows loop-back testing to be carried out on the transceivers without the need to interface with any other components.

The transceiver connected to the null modem adapter was progressively moved away from the GCS transceiver, initially in increments of 10m and then, once 100m was reached, in increments of 50m. The X-CTU software indicated 100% data packet receipt along the test path. There were no signs of this deteriorating beyond the 500m range necessary. The test was carried out in a park where there were trees and hills that would sometimes mean that the GCS was out of the line of sight of the moving transceiver. These environmental changes had no effect on the performance of the transceivers and therefore, the transceiver testing proved successful. Based on the results of this test, the

risk of telemetry downlink loss, as identified in Chapter 5, is a low probability of occurring within the 500m limit.

7.2 Image Capture System

The Image Capture System designed for carrying out surveillance over the pre-determined flight path includes a MinCam TDC-32 digital camera and electronic trigger unit. The camera is a very light 55 grams including battery. It is small in size with dimensions 65 x 53 x 21 mm. Its size and weight make it ideal for use in this project. It is capable of taking 26 shots at a maximum resolution of 640 x 480 pixels. Unfortunately this camera includes automatic shut down after 15 seconds to conserve power. This was overcome by modifying the camera so that an electronic trigger unit consisting of a PICAXE-18 microcontroller is in control of turning the camera on and taking a photo. A simple logic low on the input of the PICAXE will trigger the camera to take a photo. The electronic trigger unit has been designed and built by Terry Byrne of the faculty.

The connection of the trigger unit to the navigation HC12 is through PORTB. A logic low is sent to the trigger unit every time a photo needs to be taken. The timing of surveillance photos has been set up in software so that a photo is taken whenever a desired waypoint is reached within 5 metres. Therefore, whenever the distance to the next waypoint is less than or equal to 5 metres, a low is sent to the trigger unit and a photo is taken and saved in the MiniCam digital camera. Viewing of the surveillance photos can be carried out once the surveillance flight is complete by connecting the camera to the GCS laptop and viewing the photos using the Ulead Photo Express software that accompanies the camera.

7.2.1 Image Capture System Test

Initial testing of the camera comprised of simple logic levels being applied to the camera without the electronic trigger unit connected. Photos were taken and downloaded to the laptop for viewing. Next, the camera was connected to the trigger

unit and power applied. A low simulated by 0 volts was given as an input to the trigger unit. The trigger unit successfully triggered the camera with a delay of a second or so. Final testing of the image capture system was carried out in conjunction with the software used to send the low from the HC12 to the trigger unit. Distances of less than 5 metres were simulated and the program was run to check that the trigger unit and camera were being triggered. Testing proved successful with the only concern being the quality of the photos taken. It is anticipated that photos taken from high altitudes will not have the required resolution for effective crop surveillance.

7.3 Hardware Integration

Integration of the navigation and image capture hardware designed for the UAV is shown in Figure 15. The hardware connections for the GCS are not shown. The implementation of the design discussed in this Chapter and Chapter 6 are shown. The connections between the compass and the HC12 have been implemented using ribbon cable and connectors to reduce the size of the prototype system. The system includes a casing for protection of the circuit and a power switch for easy disconnection and reconnection of batteries during testing.

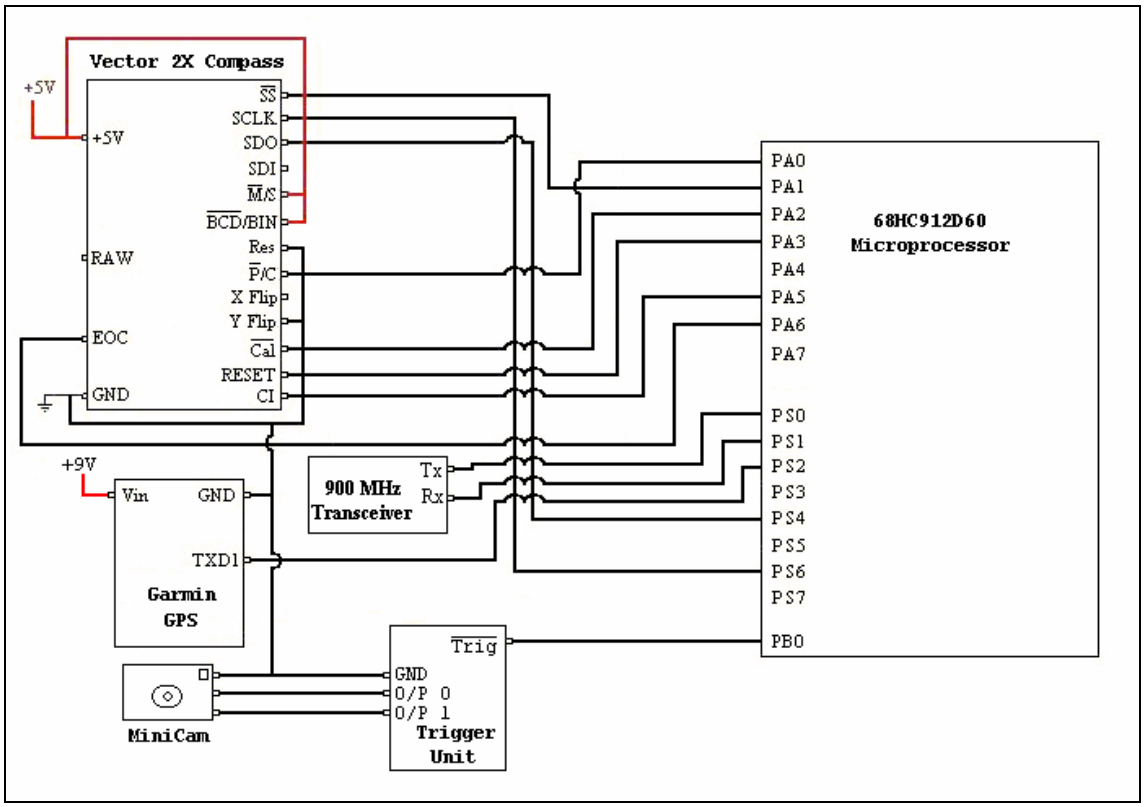


Figure 15- Integrated Hardware Wiring Diagram

Chapter 8

Software Design and Implementation

The operational and performance requirements detailed in Chapter 1 were used as a guide in determining the behaviour of the software, and as a means of qualifying and quantifying the system goals. In order to analyse the system in terms of software requirements, it was necessary to define the software structure, layout, functionality and interconnections at a high level. The programming of the individual functions was then carried out with an understanding of the specific responsibilities and interconnections of each. The final C programmed source code for the HC12 is included in Appendix G, and the Java source code for the User Interface is included in Appendix H.

8.1 Navigation Algorithm

The purpose of the Waypoint Path Planner Algorithm is to calculate the desired heading and distance to the next waypoint. The method used is the Great Circle Navigation formula described in the Navigation Strategy in Chapter 2. The inputs required for the distance formula are current position (latitude and longitude) and desired position. The current position is taken from the GPS and the desired position is taken from the user-entered waypoints. The distance calculation corresponds to an arc linking two points on a sphere. Using this distance, the bearing is calculated providing an angle measured horizontally from north to the next waypoint. The heading error is calculated by measuring the current heading from the compass and subtracting the previously calculated bearing away from it. This heading error is then scaled and transmitted to the

CAN so that it can be used by the flight control system. The flow diagram for the system is shown in Figure 16.

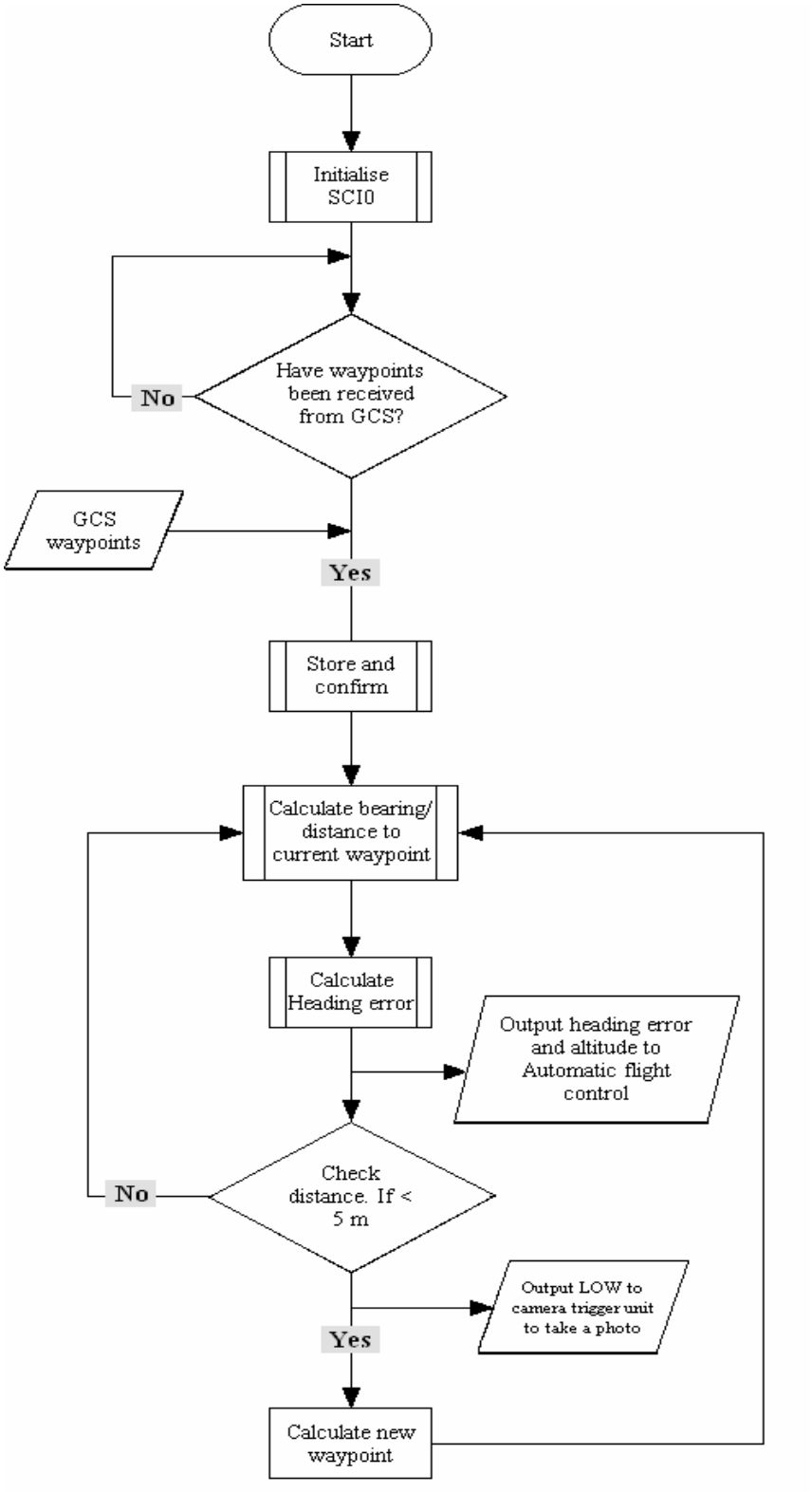


Figure 16- Navigation System Flow Diagram

8.2 GPS Message Holder

The purpose of the GPS message holder program is to read the incoming NMEA data from the GPS, and separate and extract the data required for the Waypoint Path Planner algorithm, the GCS and the flight control system. The $GPGGA message is the one that the function looks for as this holds the GPS status indication, UTC time, current latitude and longitude and antenna height (altitude) required by the system. As the GPS sends out many different messages one after another, the $GPGGA message is found by waiting for the SCI1 receive register to be full, searching for the dollar sign which precedes every message and then checking that it has the 'G' 'G' 'A' portion which will always be the third, fourth and fifth characters following on from the dollar sign. Each $GPGGA message has delimiting commas so that each section of data can be found and separated just by knowing the length of the data. Indexing through the message and saving the relevant data to a defined structure allows these variables to be used within the rest of the navigation algorithm.

The GCS waypoint reader function is a slightly modified version of the GPS message holder function. The GCS waypoint reader's purpose is to receive the data coming in from the GCS (user-entered waypoints) and save them into the applicable variables for use by the Waypoint Path Planner Algorithm. The GCS waypoint reader returns a '!' to the GCS user interface, to confirm receipt of the four waypoints.

8.3 Compass Interface

The compass interface function is designed to initialise SPI communication with the compass, and to return the current heading. This data is used for the calculation of heading error, using the desired heading from the Waypoint Path Planner Algorithm, and subtracting the current heading from the compass. These calculations are carried out four times for every GPS position and bearing calculation and as such, the resolution of the heading error is an improvement upon what would be possible with low update rates of 1 Hz.

The steps involved in SPI initialisation, compass power up, calibration and interpretation of data have been carried out in accordance with the hardware design as described in Chapter 6. Specifically, the sequence of events carried out in the SPI initialisation for the compass interface program included in Appendix G is as follows:

1. Set PORT A on the HC12 for PA0, PA1, PA2 and PA3 outputs. The rest set to inputs.
2. Set $\overline{P}/C$, $\overline{SS}$, $\overline{CAL}$ and $\overline{RESET}$ high.
3. Enable SPI, master mode, clock idle high, phase high, normal (non-bidirectional) mode.
4. SPI clock set to 1 MHz.
5. SCLK and MISO set as outputs.
6. Set $\overline{RESET}$ low. Delay 100 ms. Set $\overline{RESET}$ high. Delay 750 ms. (Power Up procedure).

The sequences of events carried out for data retrieval are:

1. Set $\overline{P}/C$ low. Delay 10 ms.
2. Wait for EOC to go low.
3. Set $\overline{P}/C$ high. Wait for EOC to go high.
4. Delay 10 ms. Lower $\overline{SS}$.
5. Then initialise transfer and interpret binary data.

The steps carried out for compass calibration are described in Chapter 6, and are implemented in software according to these steps. The heading error sent to the automatic flight control system has been scaled according to the requirements of that system. The following equation was used for scaling and implemented prior to transmission onto the CAN bus:

$$12000 + (1146/180 \, x \, heading \; error)$$

Equation 4- Heading Error Scaling

8.4 Camera Trigger Module

The camera trigger module is a simple function that sends a low to the electronic trigger unit, to enable surveillance photos be taken above waypoints during flight. The function sets PORT B's PB0 as an output. PB0 is set high normally, and then when the system is within 5 metres of a waypoint (based on distance calculations carried out in Waypoint Path Planner Algorithm), PB0 is set low, triggering the camera to take a photo. The output is then set high again until the next waypoint is reached within 5 metres.

8.5 User Interface (GUI)

As initially discussed in Chapter 3, the purpose of the user interface is to provide a familiar platform for the user to enter flight path waypoints as well as providing a visual indication of real-time UAV status during flight. The interface must be easy to use and provide adequate information without overwhelming the user. It is assumed that the user has only the most basic level of UAV technical understanding so that any status indications and/ or functions of the GUI must be easily understood.

The design of the user-interface began with an analysis of the requirements of the system, in conjunction with an evaluation of the software design tools and the integrated development environments (IDEs) available. The Java programming language was chosen due to its cross platform compatibility. After evaluating Borland JBuilder, GEL and NetBeans IDEs, NetBeans was chosen because it was found to be more user friendly than GEL, and less complex than Borland JBuilder. Once this decision was made, the functional design of the GUI was carried out. To aid in the design process, a Use Case model was developed. A Use Case model describes what the system will do at a high level with a user focus. Hence, it aids in scoping the GUI and giving the application some structure. From this model, a logic scenario diagram was developed as shown in Figure 17. The steps used to develop the 'use cases' for the GUI were:

1. Identification of 'who' is going to be using the system directly, e.g. farmer.

2. Define what this user wants to do with the system. Each of the things the user wants to do becomes a ‘use case’. This included the entering of desired waypoint flight path and up-linking these to the UAV navigation system. Passively the user wants to receive UAV status; heading and altitude, datalink loss warning, ‘no GPS capture’ warning and notification of when each waypoint has been passed. These passive requirements are included in the window design of the GUI and not in the Use Case model.

3. The most usual sequence of user/system interaction is defined. This is the basic sequence and only includes what normally happens.

4. The basic sequence is described (shown in logic scenario diagram), i.e. “the user does something…..the system does something”.

5. Once the basic sequence and descriptions were complete, the alternate paths were considered and added as extending use cases. An example of this is the inclusion of waypoint-entered data, repeated back to the user for verification. This function has been added to mitigate the risk of entering incorrect waypoints as identified in the Risk Assessment in Chapter 5.

6. Usually the next step would be to repeat this Use Case analysis for each possible user of the system. It was not necessary to repeat the cycle for this system because it is only designed to carry out UAV crop surveillance applications. Therefore, the only user considered is the farmer.

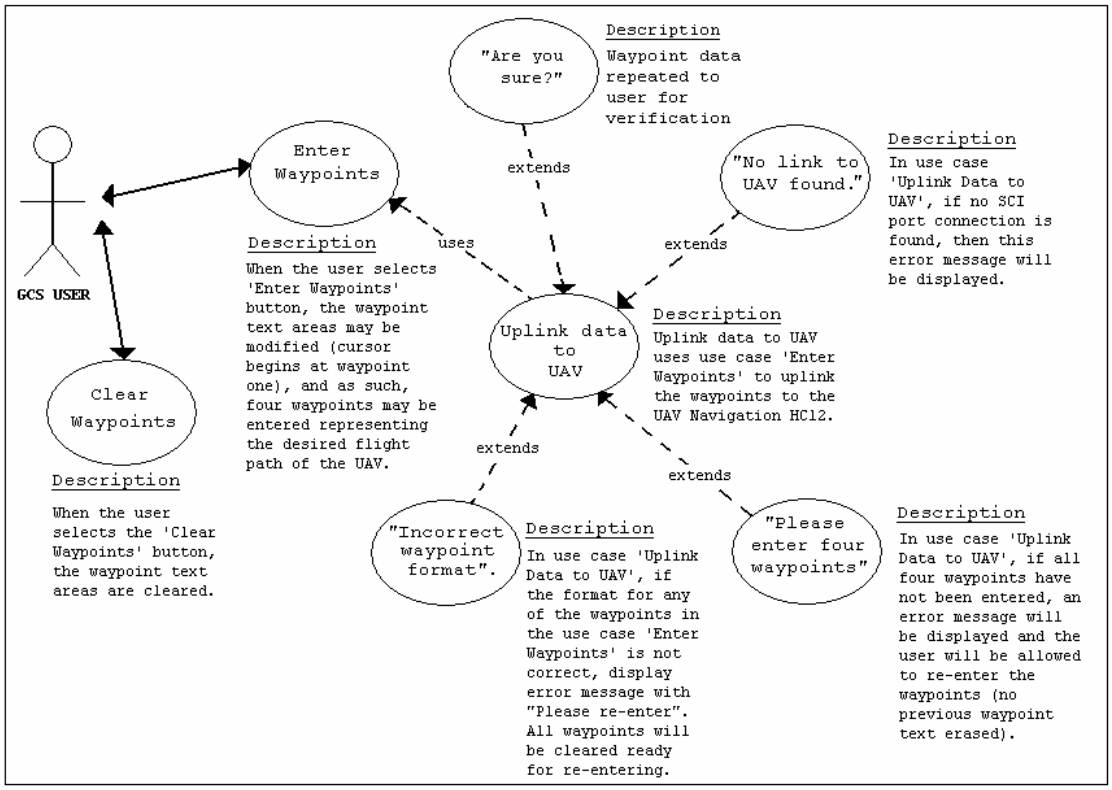


Figure 17- GCS GUI Logic Scenario Diagram

The initial GUI layout was designed by following the use cases identified in the logic scenario diagram. The passive use cases were added in this layout. The UAV status panel includes a datalink loss warning and GPS fix indication which provides important safety features. If these status' are undesirable, i.e. datalink drop out or no GPS position fix, then the UAV may be switched to manual control for safe landing. These features enhance the risk control measures outlined in the Risk Assessment for datalink failure and GPS signal loss. The GUI also includes an indication of battery voltage levels which, whilst not functional at this time, provides a layout for future inclusion of a low battery voltage detection circuit to add another safety feature to the system. At this stage the only mitigation of the risk of low power is to ensure that all batteries are sufficiently charged prior to any surveillance mission.

The compass and desired heading dials were not designed from scratch. These gauges come from freeware Java classes called ElegantJGauges and are available from Hallogram Publishing (2004). The seven segment class used for displaying the altitude was written so as to emulate the look of a LCD or seven segment display. The waypoint data panel includes all the functions outlined in the logic scenario diagram and a raw

telemetry text area has been added to the UAV status panel to aid in the testing of the final integrated Navigation and User Interface system.

The next step was to link the GUI with the navigation system to initially allow for uplink of entered waypoints and then the downlink of telemetry information. The Java SDK standard addition does not come with serial communication support. For this reason, an additional package, javax.comm (Sun Microsystems n.d), was downloaded to provide a number of Java classes that support serial communications protocol. This download comes with a number of example programs. Sun Microsystems permits reuse of this code and as such, slightly modified versions of the examples are integrated into the GUI. Once serial communications between the GUI and HC12 tested successfully (described in Section 7.6 of this Chapter), the functionality for the formatting of user-entered waypoints for transmission to the HC12 navigation algorithm was added. Following this, the program required for reception of telemetry data from the HC12 was coded. Minor revisions to the code from the initial design include the addition of a message “Waypoint Data Received” when the HC12 has successfully received and saved the user-entered waypoints. The source code written for the implementation of the GCS GUI is included in Appendix H. A captured screen view of the GUI running, after having passed the first waypoint, is shown in Figure 18.

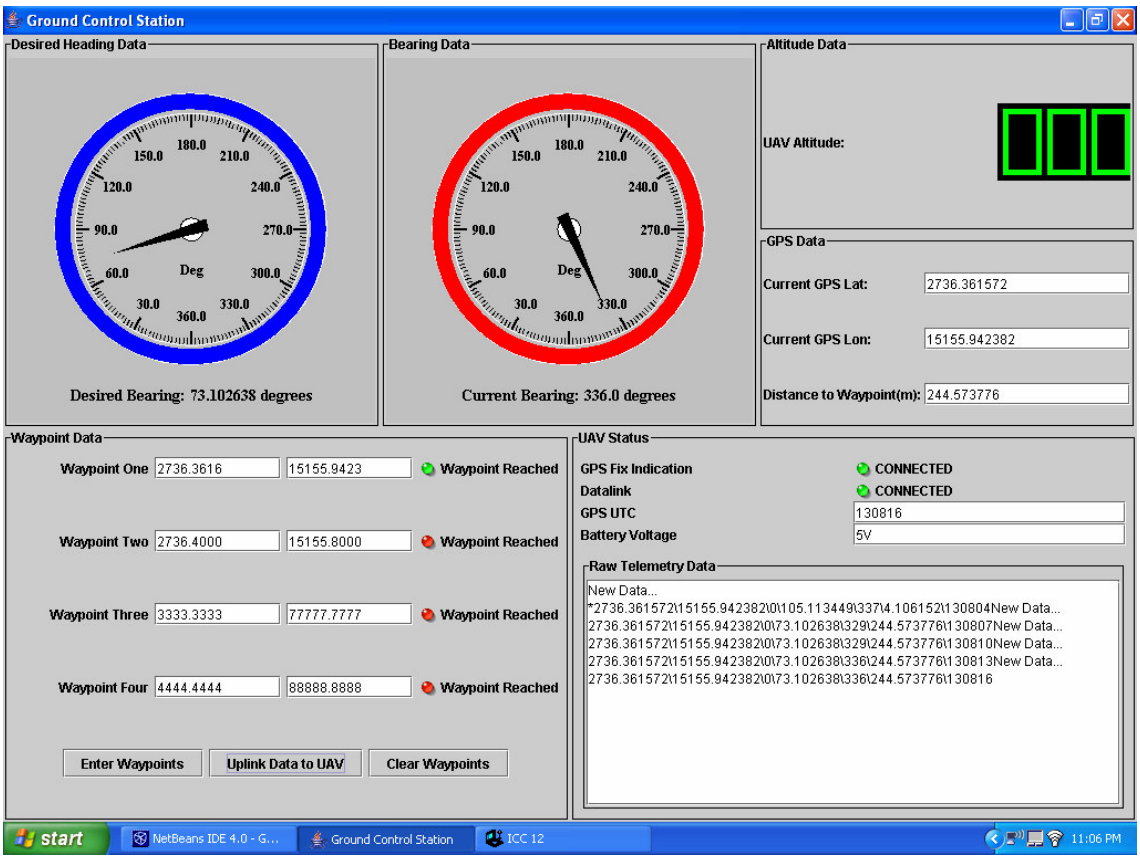


Figure 18- Ground Control Station User Interface

8.5.1 GUI Behaviour

During a typical UAV surveillance mission, the GUI behaves as follows:

- User selects 'Enter Waypoints' button to enable editing of the waypoint text fields. The user enters the desired flight path as four waypoints (latitudes and longitudes).

- The user selects 'Uplink Data to UAV'. The GUI ensures that there are enough waypoints entered and in the correct format. If not, an error message is displayed "Incorrect Waypoint Format" or "Please enter four waypoints" and the GUI allows re-entering of the waypoints.

- Once the data is entered in the correct format, the GUI displays the message "Are you sure?" along with the waypoints to be transmitted, to enable the user to change the waypoints if they have accidentally pressed 'Uplink Data to UAV' without correctly entering the waypoints. The GUI then opens the serial port and

sends the waypoint data to the HC12. The user is informed if there are any connection issues with the serial port communication and if not, a message is displayed to verify the reception of the data from the HC12, i.e. “Waypoint Received”.

- The GUI then disables all user interaction and is used as a status panel for UAV telemetry.

- The navigation HC12 sends a string containing telemetry data delimited by ‘/’. The GUI separates this string into individual elements and displays them in the appropriate area of the GUI. The information displayed as shown in Figure 18 is current heading, desired heading, altitude, current position, distance from waypoint in metres, and raw telemetry data.

- Once the first waypoint is passed (within 5 metres), the Waypoint Path Planner Algorithm sends a character to the GUI which changes the red light next to ‘Waypoint Reached’ in the waypoint data panel to green. Once all waypoints are passed, all the applicable lights will be green and the raw telemetry panel will display “All four waypoints passed” to indicate to the user that the UAV is ready for manual take-over for landing.

8.6 Inter-Processor Communications

The CAN bus described in Chapter 6 is used for communicating between the ‘Navigation HC12’ and the ‘Automatic Flight Control HC12’. It is used to send the heading error and altitude required by the Automatic flight control system to enable movement of the servos according to the desired flight path. Due to the simplistic nature of the network connected to the bus, all the filters in the CAN are set to ‘don’t care’. This is because the purpose of message filtering is to control which nodes can receive the messages that are sent. In this case, all messages are to be received by the only other node connected to the bus, i.e. the Automatic flight control system. The steps carried out in the source code for CAN bus initialisation and transmissions were:

1. Place CAN in reset during initialisation,

2. Set the CAN bit timing. In this case, 500 kbps, 16 MHz oscillations,

3. Transmit heading error in string format by passing a pointer to the data buffer to be transmitted, and then,

4. Clear the corresponding transmit buffer empty flag.

Once the data is on the CAN bus it is ready for the Automatic flight control system to read accordingly. It is imperative that the timing set up in each system corresponds to the other to ensure error free data sharing.

8.7 Validation and Testing

The testing of each individual program written in C initially involved simple compilation and running with simulated values from the Visual C++ IDE. This testing proved the functionality of the Waypoint Path Planner Algorithm as this program does not rely on an interface with any external component. The distance and bearing calculations were compared to those produced by a handheld Magellan GPS, thereby verifying the accuracy of the algorithm. The precision of the handheld Magellan was only to one decimal place, whereas the Waypoint Path Planner algorithm produces results in the order of six decimal places. The GPS itself is only accurate to within 10 to 30 metres, and so this precision is a little bit of overkill. Considering the processing power of the HC12, it was not necessary to reduce this precision. If the speed or memory size of the calculations had been a concern, this precision would have been modified.

8.7.1 Navigation

Testing and debugging the GPS Message Holder was achieved by comparing the stored data with the actual data as given in the raw messages displayed when the GPS message

was captured using an assembly language program. Using the TwinPEEKS monitor program, the raw input from the GPS was captured and saved to a word document. The GPS Message Holder was then run and the stored values printed to the screen. By ensuring that both sets of values were exactly the same, the data stored by the GPS Message Holder was verified. This testing, along with the testing of the compass interface and Camera Trigger Module, was conducted once the relevant hardware interface and testing had been carried out, to ensure the validity of the data coming from the sensors. The compass interface testing proved successful data acquisition. The camera trigger module was tested directly and photos were taken as expected. The compass and navigation algorithm code were tested together once both functions proved successful. This testing included the simulation of position data and then verification of distance and heading error calculations.

8.7.2 User Interface

The initial functionality of the GUI that required testing was the serial communications. This test used a laptop and a desktop PC with the laptop running the GCS and the PC running the serial demo program that comes with the javax.comms package. This verified that the GUI was capable of outputting correctly formatted waypoint data through the serial port. However, this test proved incomplete as the serial demo program was not set up to correctly emulate the serial communication parameters of the HC12. The HC12 requires hardware flow control to be used for incoming serial communications whereas this initial testing was conducted with no flow control. This problem was rectified by setting the flow control parameters in the GUI to RTS/CTS. Testing communications between the GUI and the HC12 showed that the HC12 could correctly receive transmissions from the GUI, but the GUI could not receive transmissions from the HC12. Eventually, it was determined that the correct flow control setting for the serial port on the GUI were to have RTS/CTS flow control on the output from the GUI, and no flow control on the inputs to the GUI. Troubleshooting this problem took many hours of testing between the laptop, PC and HC12.

Chapter 9

System Validation and Testing

Once the Navigation and User Interface hardware and software systems were fully integrated, validation and testing was carried out. The tests' success was measured in terms of the operational requirements outlined in Chapter 1. As the Navigation and User Interface system is designed to work in conjunction with an automatic flight control system for implementation into a UAV, the system was tested in a simulated environment on the ground. Two system tests were carried out. This chapter details the method, results, discussion and necessary improvements determined from the first system test, as well as the second and final system test method and results.

9.1 First System Test Method

The first integrated system testing was carried out on the USQ soccer field. A test trolley was manufactured to enable simulated UAV movement over a pre-determined user-selected path. The first system test did not include the transceivers, nor the eventual circuit casing. The soccer field co-ordinates were mapped with a handheld Magellan GPS. Although the handheld GPS and the GPS used in the system are susceptible to 10- 30m of error, the Magellan handheld allowed a known path to be followed, and readings of sensor and navigation algorithm outputs to be analysed. The distance calculations, current heading, desired heading, heading error and current position were all saved to a text file during the test to enable collation of the data collected during analysis, into more meaningful representations such as a graph. The original idea was to follow the heading error given and attempt to correct along the path

accordingly. The first system test trolley without transceivers or protective casing for the circuit is shown in Figures 19 and 20. The GPS is shown mounted on the pole to prevent inadvertent shadowing, and the camera is mounted half way down.

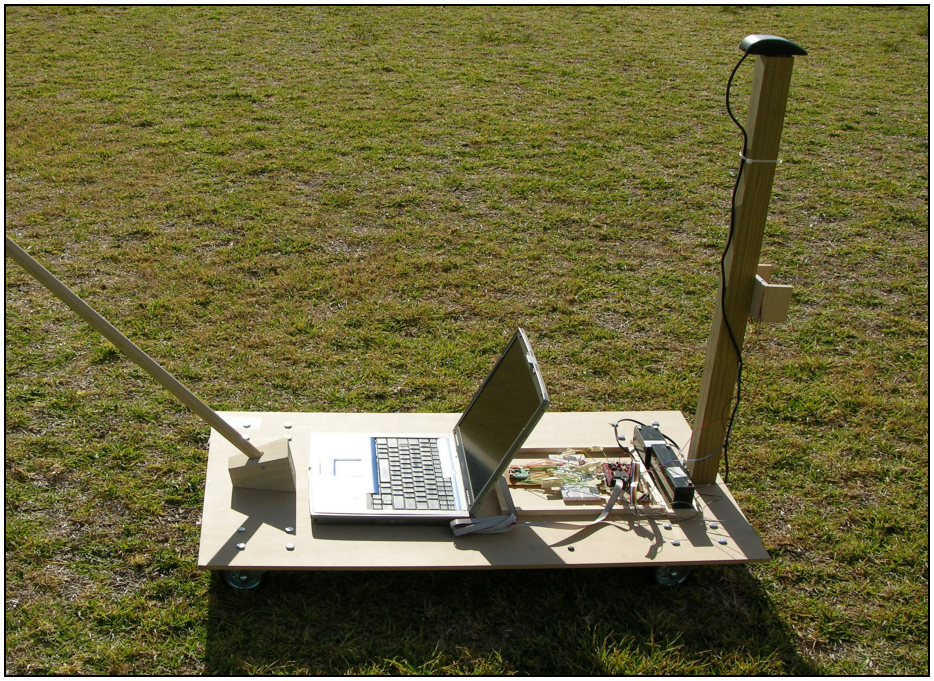


Figure 19- First System Test Trolley and Components side view

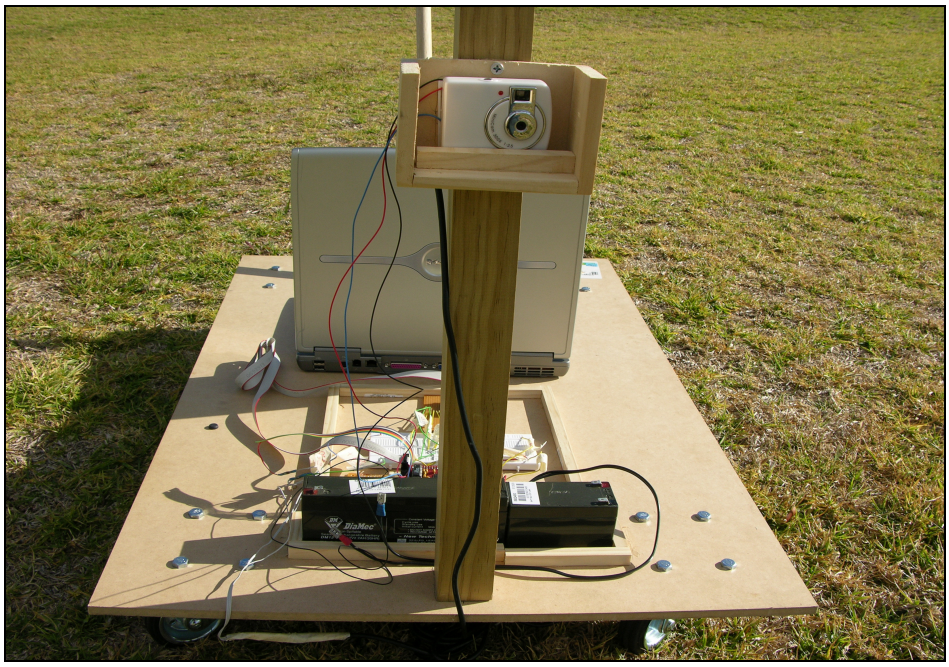


Figure 20- First System Test Trolley front view

9.2 Results

The first system test path is shown in Figure 21. The pink line shows the desired path (as designated by the handheld Magellan GPS) whilst the blue line shows the path that the saved text data indicated was actually carried out.

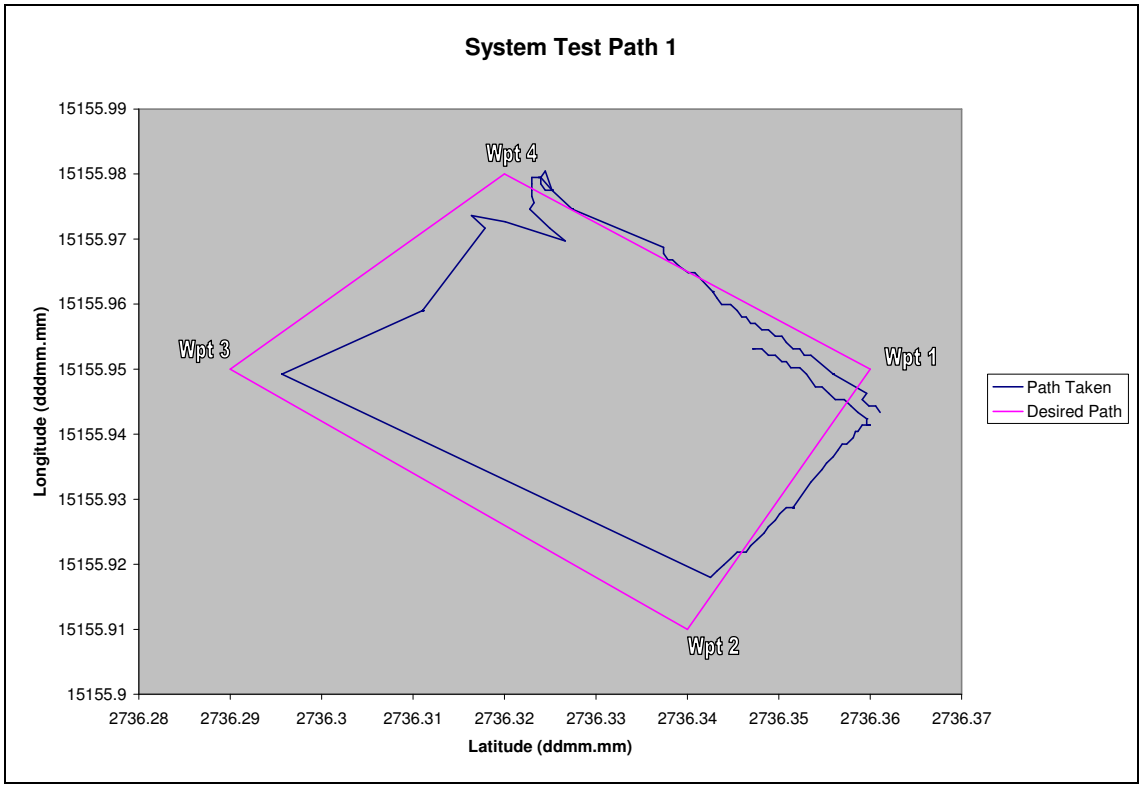


Figure 21- System Test Path 1

Unfortunately, it was evident early on in the test that the desired heading and distance to next waypoint were incorrect, and that surveillance pictures over the waypoints were not being taken along the way. Also, the calculation and subsequent printing to screen of the information was stopping (hanging) randomly along the path. By resetting the HC12 whenever this happened the test path was able to be completely followed. As a consequence, the path was followed with minor variations to check the differences in position information between the desired path and path taken.

The system outputted desired heading to the first waypoint along the whole route and was unable to change to the second, third and fourth waypoints. The reason for this was that the only way that the algorithm determines when to take a photo and when to

change to the next saved waypoint is when the distance to the current waypoint becomes less than 5m. The distance in metres being calculated by the navigation algorithm was in the order of 5000m even when the test trolley was exactly on the desired waypoint. In spite of this, the test path shows that the GPS position information and current heading from the compass were working, even if intermittently.

9.3 Discussion

Considering the inherent error of both GPS receivers as mentioned above, the desired path looks to be close to the path taken. The reasons for some of the variations on the path were that the test trolley was moved from side to side some of the way along, to check the heading error changes. Disturbingly though, the distance measurements were wrong and as such, the whole navigation algorithm was useless in terms of guidance along a pre-determined path. Also, problems with the compass 'hanging' intermittently needed to be rectified. Clearly further analysis and serious fault finding was required to determine the cause of these errors.

9.3.1 Problem Rectification

The distance calculation was again tested on its own by running the navigation algorithm function on Visual C++ and hard coding in four waypoints. The values were still found to be wrong and so redefinition of the Great Circle Distance was included. A mathematically equivalent formula to the one discussed in Chapter 2 but which is less subject to rounding errors for short distances (Williams n.d) is:

$$d = 2\,\mathrm{asin}\sqrt{\left(\sin\left(\Phi_1-\Phi_2\right)/2\right)^2+\cos\left(\Phi_1\right)\cos\left(\Phi_2\right)\left(\sin\left(\lambda_1-\lambda_2\right)/2\right)^2}$$

Equation 5- Great Circle Distance Equivalent

where $\Phi_1 = \mathrm{Lat1}, \Phi_2 = \mathrm{Lat2}, \lambda_1 = \mathrm{Lon1}$ and $\lambda_2 = \mathrm{Lon2}$.

Previous modular testing was not carried out using short distances between the waypoints and this explains why the problem was not identified earlier.

Changing the formula proved successful and the distance calculations worked once again using simulated values in the integrated development environment.

The compass hanging problem proved to be more of a mystery. After many hours of rewriting the compass interface function, it turned out that the problem lay in the set-up of the HC12. The Slave Select Output Enable (SSOE) pin on the HC12 is enabled only in the Master mode by asserting SSOE and DDS7 high. The reason these parameters were not originally set was because the compass Slave Select ($\overline{SS}$) pin was designed to connect to PA1 of the HC12 instead of the corresponding $\overline{SS}$ of the HC12, so that the interface function could have more control over timing issues. Unfortunately, by not setting the HC12 SSOE, the MODF bit of SP0SR (status register) kept getting set. The MODF bit is set automatically by SPI hardware if the MSTR control bit is set and the $\overline{SS}$ input pin becomes zero. This condition is not permitted in normal operation (Motorola Inc. 2000). Once SSOE and DDS7 were asserted high, the compass hanging problem was resolved. This fault was not evident in the original compass interface testing because the function was not run long enough to encounter a problem.

9.4 Second System Test Method

The second and final system test was carried out using the same desired path as mapped out in the first system test. This time, the Ground Control Station was set up away from the path as would be the case when used in an actual surveillance UAV application (see Figure 24). The transceivers were connected and the navigation circuit placed in a casing for protection as shown in Figures 22 and 23. An on/off switch was installed to ensure easy connection and disconnection of the batteries.

The desired waypoints, as mapped out on the soccer field, were entered into the GCS GUI. Once received by the navigation system, the trolley was guided by adjusting its bearing according to the heading error output on the GCS. This test was carried out with two people. One person sat at the GCS to relay heading error (via mobile phone) to the other person pushing and adjusting the path of the trolley/ navigation system. Once

again, the data from the test was saved in a text file for import into MS Excel to enable analysis and comparison.

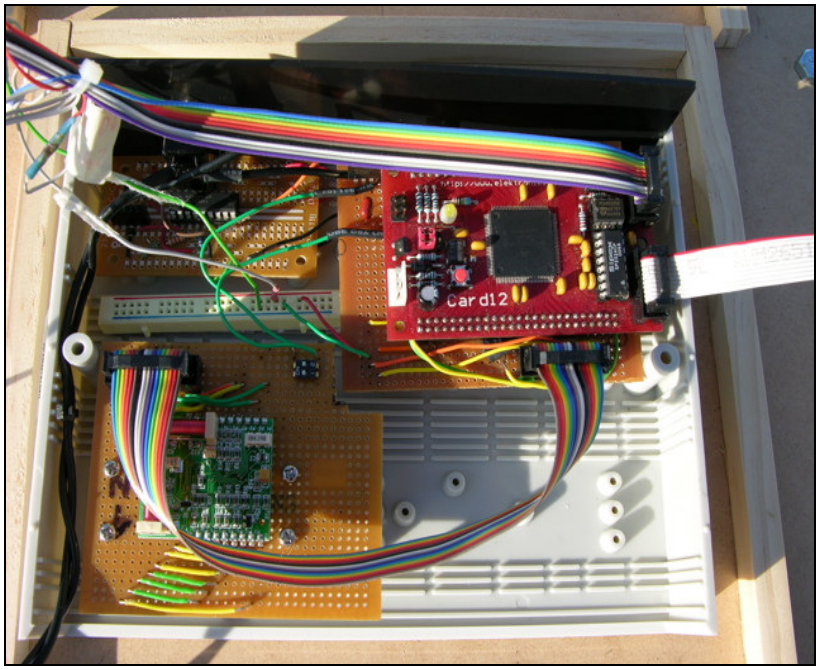


Figure 22- Navigation Circuit

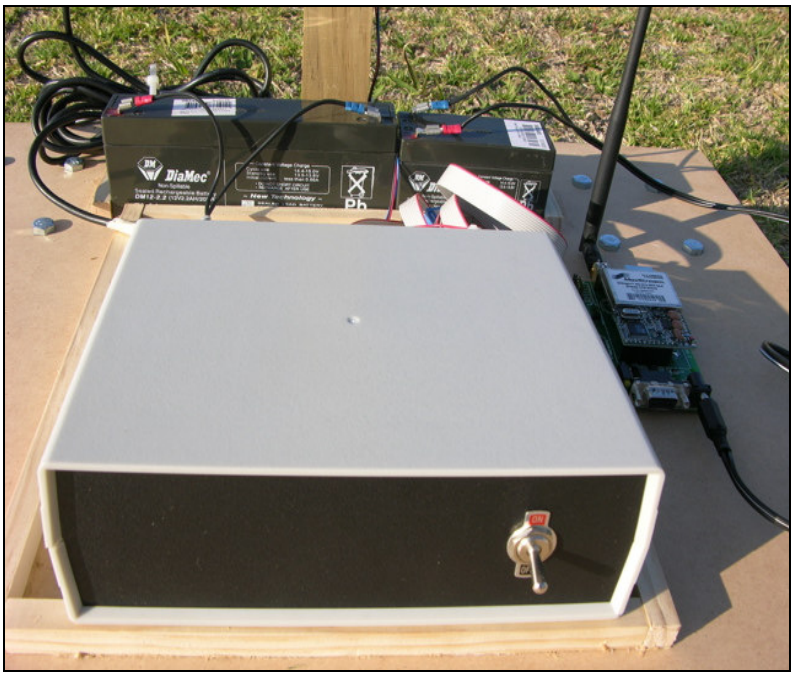


Figure 23- Navigation System (in protective housing)

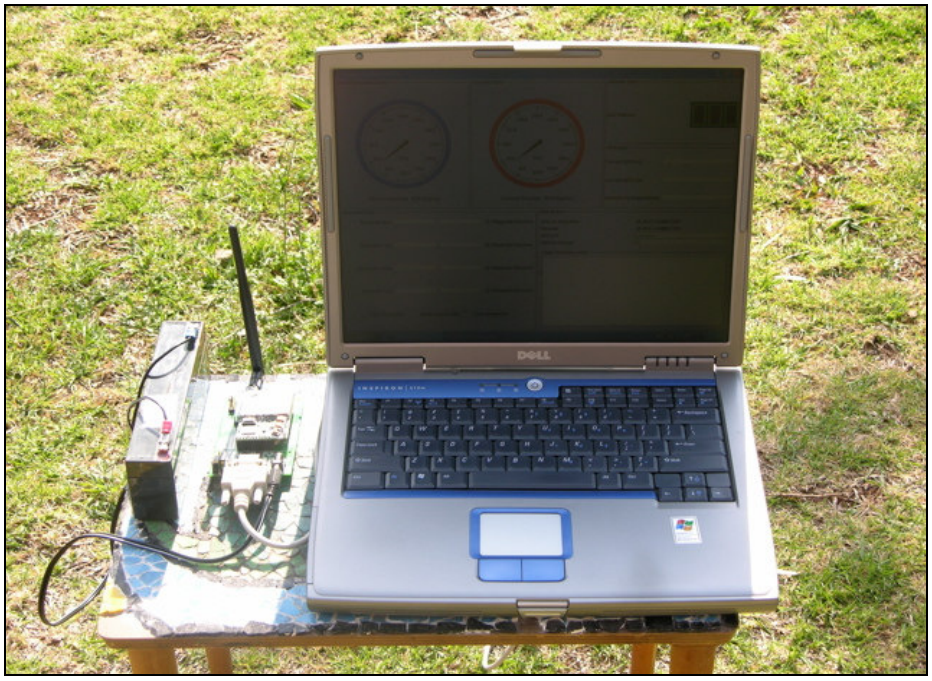


Figure 24- Second System Test Ground Control Station

9.5 Results

The results of the second and final system testing were very pleasing. The distance to the next waypoint, heading error and altitude were all in the ranges expected. Once again, the path taken was graphed against the desired path as shown in Figure 25.

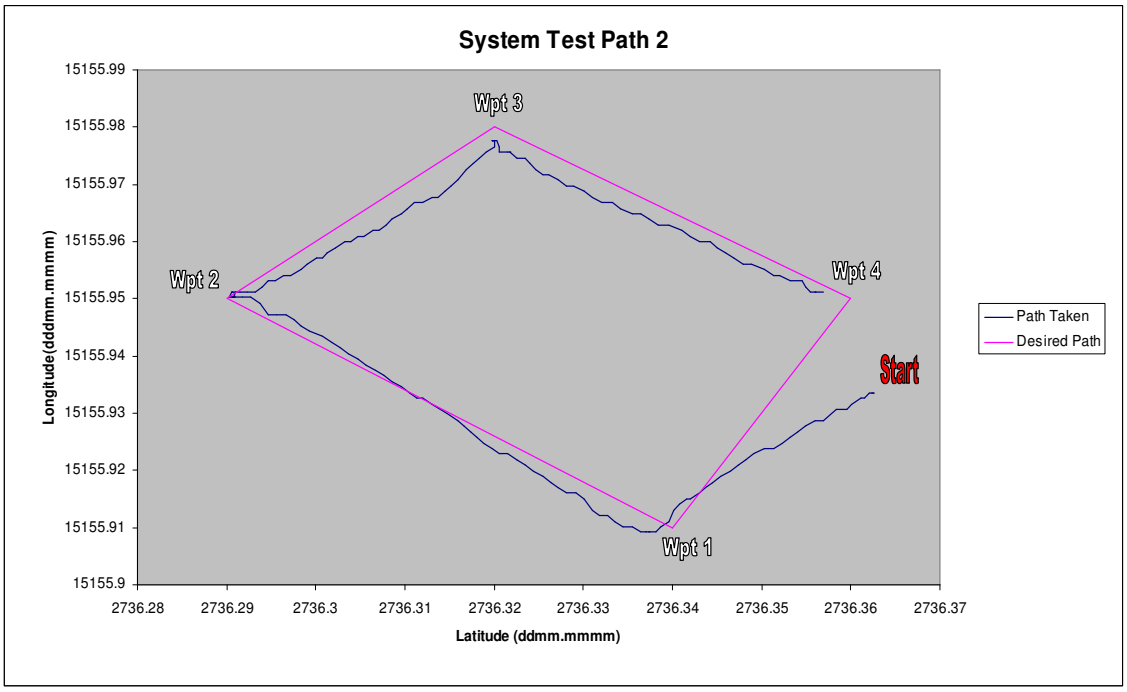


Figure 25- System Test Path 2

The path taken is within a few metres of the desired path at all times. The graph of Figure 26 represents the distance data recorded for the duration of the test path. As shown in this graph, each waypoint gets passed within 5 metres. The navigation algorithm successfully switched to the next waypoint when the trolley was within 5 metres of each desired waypoint. The camera trigger module triggered the camera once at each waypoint, and the user interface successfully displayed the telemetry data as expected.

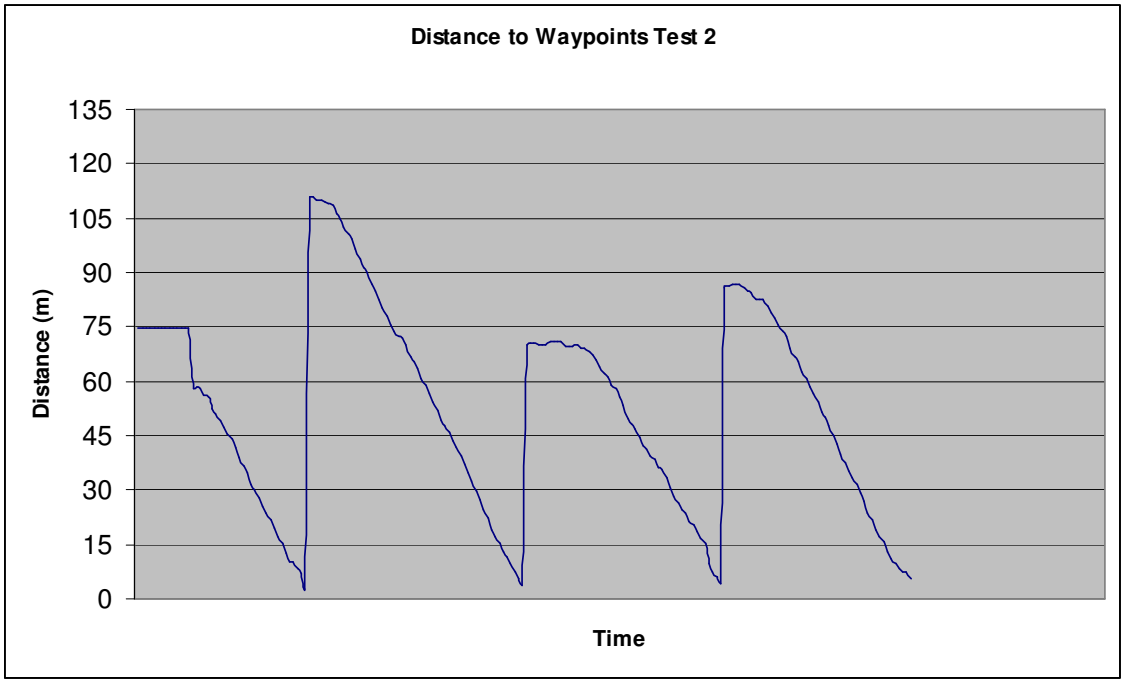


Figure 26- Distance Measurements Test 2

Chapter 10

System Performance Discussion

The integrated system tests carried out in the previous chapter have been analysed in relation to the project objectives. Importantly, the success of this project has been qualified in terms of the project specification included in Appendix A and the specific operational and performance requirements of Chapter 1.

Considering the error inherent in both the handheld GPS used for referencing and the GPS connected to the system, the results of the second and final test discussed in Chapter 9 illustrate that the system guided path is impressively accurate. Also, when you consider the fact that a real surveillance UAV would be covering a larger area than a soccer field and at much higher altitudes, the path taken as shown in the final system test of Figure 25 would be more than sufficient to ensure photos be taken of the area required. Another consideration is that a UAV Flight Control System would have a much faster and more accurate response to the heading error provided than a person does, and this would inevitably improve the response time, and consequently the accuracy of the path taken.

The success of the system can be qualified by revisiting the aim of the project and the specification details as outlined in Appendix A. As stated in the Specification, the aim of the project was to “develop a navigation algorithm, image capture system and user interface to integrate into a fully functional prototype of an autonomous Unmanned Aerial Vehicle (UAV) capable of carrying out surveillance over a pre-determined flight path.” In general terms, the aim of the project has been fulfilled without actual

integration and testing of the Navigation and User Interface system in a fully functional prototype UAV. Financial limitations imposed on this project have meant that a suitable remote control aircraft capable of the necessary payload was not available this year. Whilst the integration into a fully functional UAV has not been implemented, the simulated testing carried out has successfully proven the Navigation and User Interface system is capable of fulfilling its operational requirements. Addressing these requirements specifically, the results of the final testing prove the following:

- The system provides a user interface to allow for entering of waypoints to determine a flight path. This user interface includes error handling to mitigate the risk of incorrect waypoints entered by the user. It also provides visual indication of communication link status and GPS fix status to provide the necessary safety features required to mitigate those risks identified in the risk assessment of Chapter 5. The number of waypoints required by the system and user interface is four, allowing a square or rectangular path to be followed in accordance with the performance requirements originally identified during objective analysis.

- The system provides navigational guidance information to allow for autonomous flight over this pre-determined path. This system provides heading error and altitude to the CAN bus at a rate of 5 per second. The mitigation of incorrect heading error has been addressed in this system by minor shielding of the circuit and by including a compass calibration function.

- The system allows for capturing of image information over pre-determined points on the flight path. The system triggers the camera within five metres of each waypoint to ensure complete surveillance coverage of the area of interest.

- The system has not been integrated into a fully functional prototype UAV due to the financial limitations mentioned above.

Chapter 11

Conclusions and Recommendations

The recent commitment from the Queensland State Government to provide funding for research and development of UAVs, as well as the need for alternative means of crop surveillance, make this project a relevant step towards the University of Southern Queensland being part of a growing area of interest. As suggested by Premier Beattie in June of this year (Dept. of State Development, Trade and Innovation 2005):

> "Globally, this market is recognised to be the next revolution in aviation as information technology matures in the aerospace sector."

The navigation and user interface suites currently on the market are prohibitively expensive. This project has sought to prove that a cheap alternative is possible, without sacrificing quality.

11.1 Overall Performance

The prototype navigation, image capture system and user interface design has been achieved through the structured implementation as outlined in the development methodology in Chapter 1. In order to achieve the objectives of the project, common UAV GPS guided navigation algorithms and user interfaces were extensively researched. The details of this research and the research into the requirements for the

design and implementation of an image capture system are described in Chapters 2 and 3 of this document. Chapter 3 details the background information necessary for selection of the communications datalink chosen for downlink of telemetry, and uplink of user-entered waypoints. The literature review provided an important basis for the selection of components and methods used for the design and implementation of the whole system. Based on the literature review and the resources available, the initial high level system architecture design was established as detailed in Chapter 4. The background literature review also highlighted the need to carry out a Risk Assessment early on in the design phase. This, whilst important in all projects, was considered paramount due to the fact that aviation activities are inherently dangerous. As such, the mitigation of risks identified in Chapter 5 was a major focus throughout the design and implementation phases.

Chapters 6 and 7 give detailed descriptions of the hardware, development tools and interfacing considerations for the sensors connected to the microprocessor used to implement the Navigation and User Interface. The testing of each of the individual components is discussed as part of the staged approach planned in the design development methodology of section 1.5.

Chapter 8 gives a detailed description of the design process and implementation of the software required to carry out navigation and guidance, image capture over a pre-determined flight path, and the user interface for entering waypoints and real-time telemetry indication.

The integrated system testing is detailed in Chapter 9 with the appropriate method and results discussed. Necessary fault finding and improvements carried out in response to unforeseen problems identified in the first system test are highlighted. The second and final system testing is discussed, and the results of this test are analysed in Chapter 10 with reference to the specification, objectives and operational requirements of the system.

Overall, the system has performed well. The majority of unanticipated hurdles were overcome, with the exception of final integration of the system into a remote control aircraft. The project timeline in the Gantt chart included in Appendix I was followed within a week either side of anticipated timings, and this along with the project design

methodology ensured that the risk of breaching the schedule (as identified in the Risk Assessment) was averted.

11.2 Recommendations for Further Work

The most obvious improvement that can be made to this system is to implement it into a remote control aircraft with an Automatic Flight System. Whilst this is not technically an improvement to the system, it would certainly give the Navigation and User Interface a purpose and practicable functionality. Once implemented in a UAV, extensive testing would be required to assess the ability of the system to withstand the noisy environment created by the servos and engines. Unfortunately the simulated UAV testing carried out thus far has not been able to emulate this kind of electromagnetic interference and therefore, it is unknown whether the system will function reliably in that environment. The results of this testing would enable analysis of the current shielding, and identify the need or otherwise of further measures.

In terms of the system specifically, there is room for improvement and areas where further work could enhance the current features of the system. Some recommendations include:

- A gimballed compass to eliminate tilt error induced by rolling of the UAV.

- Reduce the current prototype circuit size by designing a printed circuit board that includes the HC12, compass and trigger unit in one.

- Design and build an anti-vibration platform for the avionics to ensure protection and durability of the components.

- Design a low battery detection circuit to interface with the low battery voltage panel on the user interface GUI for enhanced safety.

- Include real-time monitoring of images and the ability to provide updated flight path information during flight.

- Include mapping of surveillance flights on the user interface GUI by incorporating GIS type software.

The recommendations for further work listed above indicate that this project offers a platform for future students to improve and build upon. UAV technology is an exciting area of growth. This dissertation describes a cost effective navigation and user interface for any ongoing projects in this field. With the current world wide interest in civilian UAV applications and the Queensland State Government's financial commitment to their development, the Navigation and User Interface here described is a relevant contribution, as it offers a cheap alternative to current packages on the market.

References

Almy, T 2004, *'Designing with Microcontrollers- The 68HC12'*, viewed 18 April 2005, http://hc12text.com

Beasley, JS & Miller, GM 2005, '*Modern Electronic Communication*', 8th edn, Pearson Education, Inc., Upper Saddle River, New Jersey.

Civil Aviation Safety Authority 1998, '*CASR 1998*', Australian Government, viewed 12 March 2005, http://www.casa.gov.au/rules/1998casr/.

Department of Defence 2005, *'UAV (Avatar uninhabited aerial vehicle)',* Defence Material Organisation, Australian Government Canberra, viewed 06 April 2005, http://www.defence.gov.au/dmo/teamaustralia/indexc143.html.

Department of Defense & Department of Transportation 1996, *'Federal Radio Navigation Plan'*, National Technical Information Service, Springfield Virginia

Department of State Development, Trade and Innovation 2005, *'Queensland to be a World Centre for RD of Unmanned Aircraft'*, Queensland Government, viewed 09 Sep 2005, http://www.sdi.qld.gov.au/dsdweb/v3/guis/templates/content/gui_cue_cntnhtml.cfm?id=20859

Dittrich, JS 2002, '*Design and Integration of an Unmanned Aerial Vehicle Navigation System'*, School of Aerospace Engineering, Georgia Institute of Technology, viewed 28 January 2005, http://controls.ae.gatech.edu/papers/dittrich_msthesis.pdf .

Elektronikladen 2005, *'Card 12'*, viewed 12 March 2005, http://elmicro.com/en/card12.php

England, B, Garcia, A, Herlugson, K & Montano, C 2002, '*Golfing Robot- Team B Navigation*', New Mexico Technology, viewed 06 April 2005, http://www.ee.nmt.edu/~wedeward/EE382/SP02/Nav_B.pdf.

FXC Corporation 2005, *'The Airborne Guidance Unit'*, FXC Corporation, Santa Anna, California, viewed 15 March 2005, http://www.pia.com/fxc/apadsagu.htm.

Garmin Corporation 2000, '*GPS 35 LP TracPak$_{TM}$*', Olathe Kansas.

Hallogram Publishing 2004, *'ElegantJGauges'*, viewed 10 March 2005, http://www.hallogram.com/elegantjgauges/

Herwitz, SR, Johnson, LF, Dunagan, SE, Higgins, RG, Sullivan, DB, Zheng, J, Lobitz, BM, Leung, JG, Gallmeyer, BA, Aoyagi, M, Slye, RE & Brass JA, 2003, '*Imaging from an unmanned aerial vehicle: agricultural surveillance and decision support*', Computers and Electronics in Agriculture, 2004, viewed 20 March 2005, http://www.sciencedirect.com.

Honeywell Sensor Products n.d., '*Digital Compass Module HMR3000*', Solid State Electronics Center, Plymouth, MN, viewed 12 March 2005, http://www.ssec.honeywell.com/magnetic/datasheets/hmr3000.pdf.

Huang, HW 2003, *'MC68HC12: An Introduction Software and Hardware Interfacing'*, Delmar Learning, Clifton Park, NY.

Jensen, T, Apan, A, Young, F, Zeller, L & Cleminson K 2003, '*Assessing grain crop attributes using digital imagery acquired from a low-altitude remote controlled aircraft*', Spatial Sciences 2003, viewed 06 April 2005, Http://www.usq.edu.au/users/apana/JENSEN,%20TROY.pdf.

Kayton, M & Fried, WR 1997, '*Avionics Navigation Systems*', 2nd Edn, John Wiley & Sons, Inc, Canada.

Keeffe, M 2003, '*The Autonomous Scale Model Surveillance Aircraft*', Faculty of Engineering and Surveying, University of Southern Queensland.

Littleton, C 2005, *'Autonomous Unmanned Aerial Surveillance Vehicle- Autonomous Control and Flight Dynamics'*, Faculty of Engineering and Surveying, University of Southern Queenslnd.

MaxStream 2005, '*XStream$_{TM}$ OEM RF Modules*' MaxStream, Inc, Lindon, UT, viewed 11 March 2005, http://www.maxstream.net/products/xstream/module/datasheet_XStream_OEM-RF-Module.pdf.

McManus, IA 2004, *'Intelligent Agent Based Avionics for Low Cost Uninhabited Aircraft Operations in Civilian Airspace'* Cooperative Research Centre for Satellite Systems, viewed 15 March 2005, http://www.bee.qut.edu.au/projects/quav/Postgraduate/Research/Mission%20Planning/mainpage.php.

Merminod, B 1989, '*The Use of Kalman Filters in GPS Navigation*' University of New South Wales, Sydney.

Motorola Inc 2000, '*68HC912D60 Advance Information*' Rev 2.0.

Nygards, J, Skoglar, P & Ulvklo, M 2003, '*Navigation Aided Image Processing in UAV Surveillance*', Journal of Robotic Systems, 21(2), 63-72 2004, viewed 26 April 2005, http://www.interscience.wiley.com.

PNI Corporation 1998, '*Vector 2X User Manual*', Version 1.08, viewed 17 Mar 2005, http://pnicorp.com/tecnical-information/pdf/vector-2x.pdf.

Royal Australian Navy 2000, '*Aviation Safety Manual*', ABR 5147 Annex C, Ch. 4.

Standards Australia 2004, *'Risk Management'*, AS/NZS 4360:2004, viewed 18 Mar 2005, http://www.standards.com.au

Schulze, K, Abramson, A & Rogan B 2004, '*An Economical Approach to Autopilot Development and Integration*' 3rd AIAA, viewed 2 March 2005, http://pdf.aiaa.org/MUAV2004_1007/PV2004_6572.pdf.

Sheldon, T 2001, '*Tom Sheldon's Linktionary*' Big Sur Multimedia, Viewed 26 April 2005, http://www.linktionary.com/b/bandwidth.html.

Stefan, J 2000, '*Navigating with GPS*', Circuit Cellar, October 2000, Issue 123, viewed 05 February 2005, http://www.circuitcellar.com/library/print/1000/Stefan123/Stefan123.pdf.

Sun Microsystems n.d., *'Java Communications API'*, viewed 02 June 2005, http://java.sun.com/products/javacomm/downloads/

Valvano, JW 2000, *'Embedded Microcomputer Systems: Real Time Interfacing'*, Brooks-Cole, United States.

Williams, E n.d., '*Aviation Formulary V1.42*' viewed 16 March 2005, http://williams.best.vwh.net/avform.htm.

Wong, KC & Bil, C 2003, '*UAVs Over Australia*', University of Sydney, viewed 20 February 2005, http://www.aeromech.usyd.edu.au/wwwdocs/uavs_over_australia_0398.pdf.

Appendix A- Project Specification

Appendix A

University of Southern Queensland
FACULTY OF ENGINEERING AND SURVEYING

ENG4111/4112 Research Project

PROJECT SPECIFICATION

FOR: Soz Deja KNOX

TOPIC: AUTONOMOUS UNMANNED AERIAL SURVEILANCE VEHICLE- NAVIGATION AND USER INTERFACE

SUPERVISORS: Mr. Mark Phythian
LEUT Kathryn Burr, Royal Australian Navy, Res

ENROLMENT: ENG4111 – S1, D, 2005
ENG4112 – S2, D, 2005

PROJECT AIM: This project seeks to develop a navigation algorithm, image capture system and user interface to integrate into a fully functional prototype of an autonomous Unmanned Aerial Vehicle (UAV) capable of carrying out surveillance over a pre-determined flight path

PROGRAMME: Issue A, 21 March 2005

1. Research information for the design and implementation of common Unmanned Aerial Vehicle (UAV) GPS guided navigation algorithms and user interfaces.
2. Research requirements for the design and implementation of an image capture system including hardware, interfacing requirements and payload limitations.
3. Research communication alternatives to interface Ground Station (user interface) with UAV for uplink of navigation algorithm and downlink of UAV telemetry.
4. Design, develop and test individual systems, i.e navigation algorithm, image capture system and user interface.
5. Design interface for the individual systems and integrate to enable surveillance over a pre-determined path with non real-time entering of flight path and real-time telemetry downlink.
6. Construct prototype UAV (using a model aircraft) and integrate systems with 'Autonomous Control and Flight Dynamics' project being carried out by Craig Littleton.

As time permits:

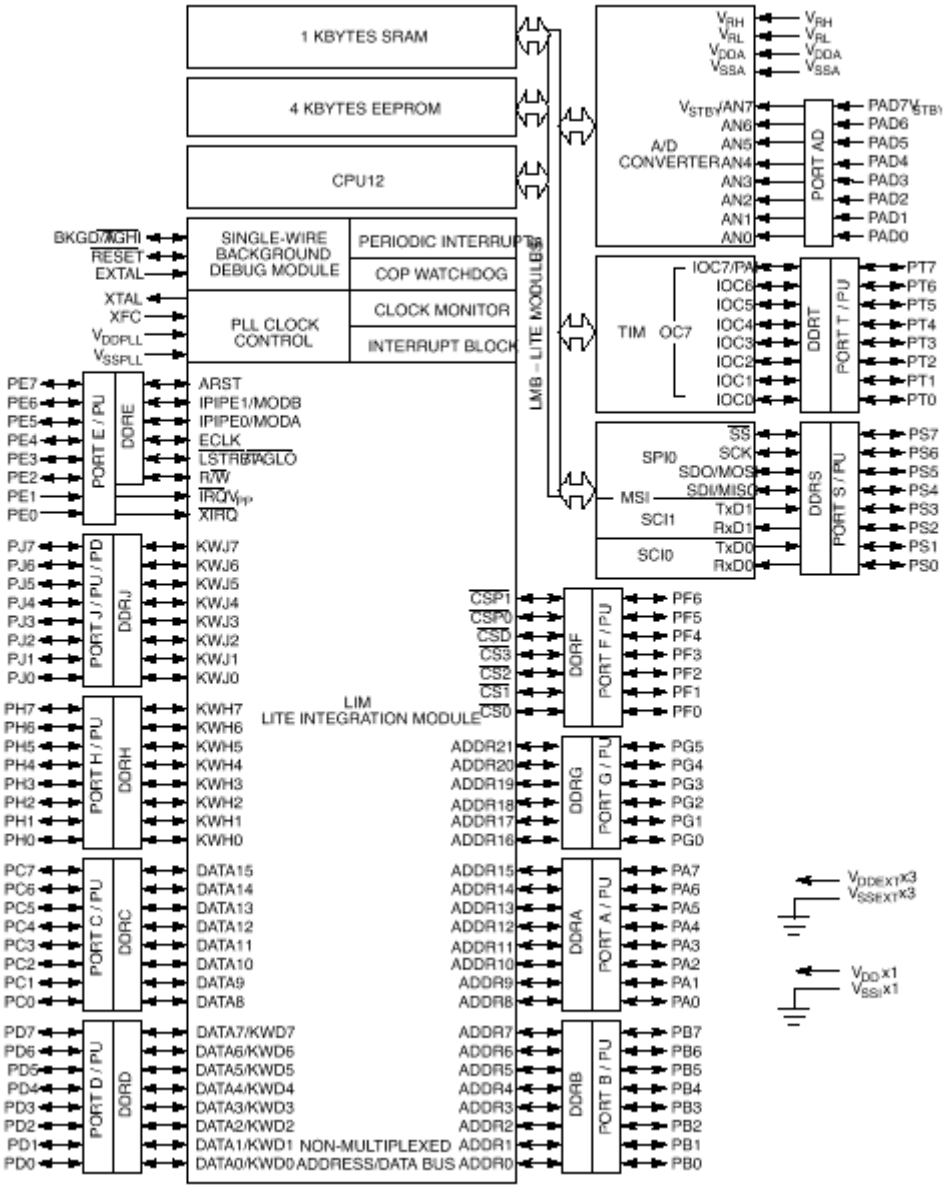
1. Design real-time monitoring of images and the ability to provide updated flight path information during flight.

AGREED:

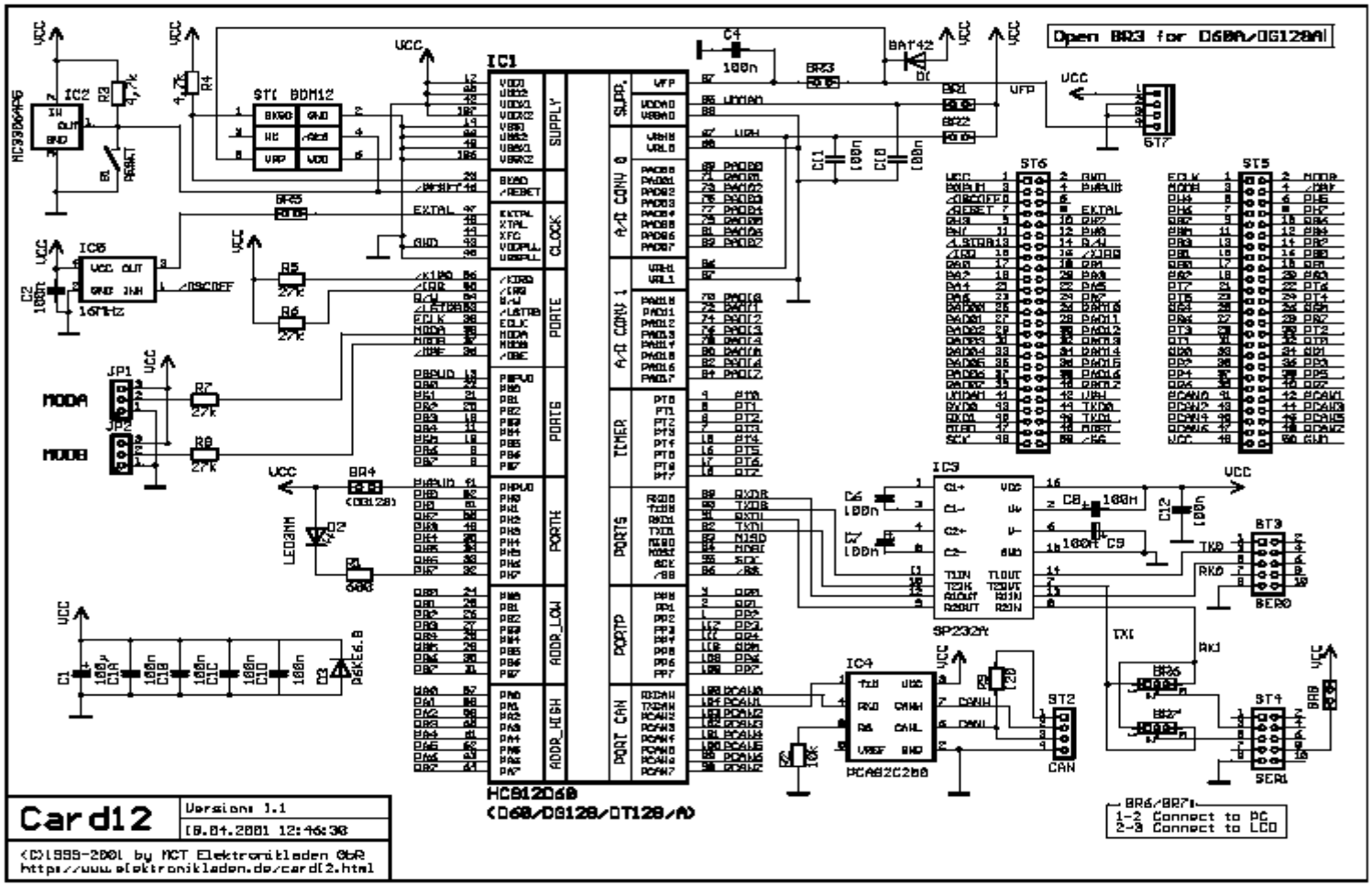
_____ (Student) _____ , _____ (Supervisors)

__ / __ /2005 __ / __ /2005 __ / __ /2005

Appendix B- 68HC(9)12D60 Block Diagram


1 KBYTES SRAM
4 KBYTES EEPROM
CPU12
SINGLE-WIRE BACKGROUND DEBUG MODULE
PERIODIC INTERRUPT
COP WATCHDOG
PLL CLOCK CONTROL
CLOCK MONITOR
INTERRUPT BLOCK
LMB – LITE MODULE BUS
A/D CONVERTER
PORT AD
TIM
OC7
DDRT
PORT T / PU
SPI0
MSI
SCI1
SCI0
DDRS
PORT S / PU
PORT E / PU
DDRE
PORT J / PU / PD
DDRJ
PORT H / PU
DDRH
PORT C / PU
DDRC
PORT D / PU
DDRD
LIM
LITE INTEGRATION MODULE
DDRF
PORT F / PU
DDRG
PORT G / PU
DDRA
PORT A / PU
DDRB
PORT B / PU
NON-MULTIPLEXED ADDRESS/DATA BUS

Appendix C- Card12 Schematic


Open BR3 for D60A/DG128A!
IC1
HCS12D60
(D60/DG128/DT128/A)
Card12
Version: 1.1
(C)1999-2001 by MCT Elektronikladen GbR
https://www.elektronikladen.de/card12.html
BR6/BR7:
1-2 Connect to PC
2-3 Connect to LCD
MODA
MODB
SP232A
MCAN2C200
BDM12

Appendix D- NMEA Transmitted Sentences GPS 35LP

4.2 NMEA Transmitted Sentences

The subsequent paragraphs define the sentences which can be transmitted on TXD1 by the GPS 35LP receivers.

4.2.1 Sentence Transmission Rate

Sentences are transmitted with respect to the user selected baud rate.

Regardless of the selected baud rate, the information transmitted by the GPS 35LP is referenced to the one-pulse-per-second output pulse immediately preceding the GPRMC sentence.

The GPS 35LP will transmit each sentence (except where noted in particular transmitted sentence descriptions) at a periodic rate based on the user selected baud rate and user selected output sentences. The sensor board will transmit the selected sentences contiguously. The contiguous transmission starts at a GPS second boundary. The length of the transmission can be determined by the following equation and tables:

$$\text{length} = \frac{\text{total characters to be transmitted}}{\text{characters transmitted per sec}}$$

Baud	characters_transmitted_per_sec
300	30
600	60
1200	120
2400	240
4800	480
9600	960
19200	1920

Sentence	max_characters
GPGGA	81
GPGSA	66
GPGSV	210
GPRMC	74
GPVTG	42
PGRMB	26
PGRME	35
PGRMT	50
PGRMV	32
PGRMF	82
LCGLL	44
LCVTG	39

The maximum number of fields allowed in a single sentence is 82 characters including delimiters. Values in the table include the sentence start delimiter character "$" and the termination delimiter <CR><LF>.

The factory set defaults will result in a once per second transmission at the NMEA specification transmission rate of 4800 baud.

4.2.2 Transmitted Time

The GPS 35LP receivers output UTC (Coordinated Universal Time) date and time of day in the transmitted sentences. Prior to the initial position fix, the date and time of day are provided by the on-board clock. After the initial position fix, the date and time of day are calculated using GPS satellite information and are synchronized with the one-pulse-per-second output.

The GPS 35LP uses information obtained from the GPS satellites to add or delete UTC leap seconds and correct the transmitted date and time of day. The transmitted date and time of day for leap second correction follow the guidelines in *"National Institute of Standards and Technology Special Publication 432 (Revised 1990)"* (for sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC, 20402, USA).

When a positive leap second is required, the second is inserted beginning at 23h 59m 60s of the last day of a month and ending at 0h 0m 0s of the first day of the following month. The minute containing the leap second is 61 seconds long. The GPS 35LP would have transmitted this information for the leap second added December 31, 1989 as follows:

Date	Time
311289	235959
311289	235960
010190	000000

If a negative leap second should be required, one second will be deleted at the end of some UTC month. The minute containing the leap second will be only 59 seconds long. In this case, the GPS 35LP will not transmit the time of day 23h 59m 59s for the day from which the leap second is removed.

4.2.3 Global Positioning System Almanac Data (ALM)

<field information> can be found in section 4.1.1.

$GPALM,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,<12>,<13>,<14>,<15>*hh<CR><LF>

Almanac sentences are not normally transmitted. Almanac transmission can be initiated by sending the sensor board a $PGRMO,GPALM,1 command. Upon receipt of this command the sensor board will transmit available almanac information on GPALM sentences. During the transmission of almanac sentences other NMEA data output will be temporarily suspended.

<field information> can be found in section 4.1.1.

4.2.4 Global Positioning System Fix Data (GGA)

$GPGGA,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,M,<10>,M,<11>,<12>*hh<CR><LF>

<1> UTC time of position fix, hhmmss format
<2> Latitude, ddmm.mmmm format (leading zeros will be transmitted)
<3> Latitude hemisphere, N or S
<4> Longitude, dddmm.mmmm format (leading zeros will be transmitted)
<5> Longitude hemisphere, E or W

<6> GPS quality indication,
0 = fix not available,
1 = Non-differential GPS fix available,
2 = Differential GPS (DGPS) fix available
6 = Estimated
<7> Number of satellites in use, 00 to 12 (leading zeros will be transmitted)
<8> Horizontal dilution of precision, 0.5 to 99.9
<9> Antenna height above/below mean sea level, -9999.9 to 99999.9 meters
<10> Geoidal height, -999.9 to 9999.9 meters
<11> Differential GPS (RTCM SC-104) data age, number of seconds since last valid RTCM transmission (null if non-DGPS)
<12> Differential Reference Station ID, 0000 to 1023 (leading zeros transmitted, null if non-DGPS)

4.2.5 GPS DOP and Active Satellites (GSA)

$GPGSA,<1>,<2>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<4>,<5>,<6>*hh<CR><LF>
<1> Mode, M = manual, A = automatic
<2> Fix type, 1 = not available, 2 = 2D, 3 = 3D
<3> PRN number, 01 to 32, of satellite used in solution, up to 12 transmitted (leading zeros will be transmitted)
<4> Position dilution of precision, 0.5 to 99.9
<5> Horizontal dilution of precision, 0.5 to 99.9
<6> Vertical dilution of precision, 0.5 to 99.9

4.2.6 GPS Satellites in View (GSV)

$GPGSV,<1>,<2>,<3>,<4>,<5>,<6>,<7>,...<4>,<5>,<6>,<7>*hh<CR><LF>
<1> Total number of GSV sentences to be transmitted
<2> Number of current GSV sentence
<3> Total number of satellites in view, 00 to 12 (leading zeros will be transmitted)
<4> Satellite PRN number, 01 to 32 (leading zeros will be transmitted)
<5> Satellite elevation, 00 to 90 degrees (leading zeros will be transmitted)
<6> Satellite azimuth, 000 to 359 degrees, true (leading zeros will be transmitted)
<7> Signal to noise ratio (C/No) 00 to 99 dB, null when not tracking (leading zeros will be transmitted)
NOTE: Items <4>,<5>,<6> and <7> repeat for each satellite in view to a maximum of four (4) satellites per sentence. Additional satellites in view information must be sent in subsequent sentences. These fields will be null if unused.

4.2.7 Recommended Minimum Specific GPS/TRANSIT Data (RMC)

$GPRMC,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,<12>*hh<CR><LF>
<1> UTC time of position fix, hhmmss format
<2> Status, A = Valid position, V = NAV receiver warning
<3> Latitude, ddmm.mmmm format (leading zeros will be transmitted)
<4> Latitude hemisphere, N or S
<5> Longitude, dddmm.mmmm format (leading zeros will be transmitted)
<6> Longitude hemisphere, E or W
<7> Speed over ground, 0000.0 to 1851.8 knots (leading zeros will be transmitted)
<8> Course over ground, 000.0 to 359.9 degrees, true (leading zeros will be transmitted)
<9> UTC date of position fix, ddmmyy format
<10> Magnetic variation, 000.0 to 180.0 degrees (leading zeros will be transmitted)
<11> Magnetic variation direction, E or W (westerly variation adds to true course)

<12> Mode indicator (only output if NMEA 2.30 active), A = Autonomous, D = Differential, E = Estimated, N = Data not valid

4.2.8 Track Made Good and Ground Speed with GPS Talker ID (VTG)

The GPVTG sentence reports track and velocity information with a checksum:

$GPVTG,<1>,T,<2>,M,<3>,N,<4>,K,<5>*hh<CR><LF>
<1> True course over ground, 000 to 359 degrees (leading zeros will be transmitted)
<2> Magnetic course over ground, 000 to 359 degrees (leading zeros will be transmitted)
<3> Speed over ground, 000.0 to 999.9 knots (leading zeros will be transmitted)
<4> Speed over ground, 0000.0 to 1851.8 kilometers per hour (leading zeros will be transmitted)
<5> Mode indicator (only output if NMEA 2.30 active), A = Autonomous, D = Differential, E = Estimated, N = Data not valid

4.2.9 Geographic Position with LORAN Talker ID (LCGLL)

The LCGLL sentence reports position information.

$LCGLL,<1>,<2>,<3>,<4>,<5>,<6>,<7><CR><LF>
<1> Latitude, ddmm.mmmm format (leading zeros will be transmitted)
<2> Latitude hemisphere, N or S
<3> Longitude, dddmm.mmmm format (leading zeros will be transmitted)
<4> Longitude hemisphere, E or W
<5> UTC time of position fix, hhmmss format
<6> Status, A = Valid position, V = NAV receiver warning
<7> Mode indicator (only output if NMEA 2.30 active), A = Autonomous, D = Differential, E = Estimated, N = Data not valid

4.2.10 Track Made Good and Ground Speed with LORAN Talker ID (LCVTG)

The LCVTG sentence reports track and velocity information.

$LCVTG,<1>,T,<2>,M,<3>,N,<4>,K,<5><CR><LF>
<1> True course over ground, 000 to 359 degrees (leading zeros will be transmitted)
<2> Magnetic course over ground, 000 to 359 degrees (leading zeros will be transmitted)
<3> Speed over ground, 000.0 to 999.9 knots (leading zeros will be transmitted)
<4> Speed over ground, 0000.0 to 1851.8 kilometers per hour (leading zeros will be transmitted)
<5> Mode indicator (only output if NMEA 2.30 active), A = Autonomous, D = Differential, E = Estimated, N = Data not valid

4.2.11 Estimated Error Information (PGRME)

The GARMIN Proprietary sentence $PGRME reports estimated position error information.

$PGRME,<1>,M,<2>,M,<3>,M*hh<CR><LF>
<1> Estimated horizontal position error (HPE), 0.0 to 999.9 meters
<2> Estimated vertical position error (VPE), 0.0 to 999.9 meters
<3> Estimated position error (EPE), 0.0 to 999.9 meters

4.2.12 GPS Fix Data Sentence (PGRMF)

The sentence $PGRMF is GARMIN Proprietary format.

$PGRMF,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,<12>,<13>,<14>,<15>*hh<CR><LF>
<1> GPS week number (0 - 1023)
<2> GPS seconds (0 - 604799)
<3> UTC date of position fix, ddmmyy format
<4> UTC time of position fix, hhmmss format
<5> GPS leap second count
<6> Latitude, ddmm.mmmm format (leading zeros will be transmitted)
<7> Latitude hemisphere, N or S
<8> Longitude, dddmm.mmmm format (leading zeros will be transmitted)
<9> Longitude hemisphere, E or W
<10> Mode, M = manual, A = automatic
<11> Fix type, 0 = no fix, 1 = 2D fix, 2 = 3D fix
<12> Speed over ground, 0 to 1851 kilometers/hour
<13> Course over ground, 0 to 359 degrees, true
<14> Position dilution of precision, 0 to 9 (rounded to nearest integer value)
<15> Time dilution of precision, 0 to 9 (rounded to nearest integer value)

4.2.13 Sensor Status Information (PGRMT)

The GARMIN Proprietary sentence $PGRMT gives information concerning the status of the sensor board. This sentence is transmitted once per minute regardless of the selected baud rate.

$PGRMT,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>*hh<CR><LF>
<1> Product, model and software version (variable length field, e.g., "GPS 25LP VER 1.10")
<2> ROM checksum test, P = pass, F = fail
<3> Receiver failure discrete, P = pass, F = fail
<4> Stored data lost, R = retained, L = lost
<5> Real time clock lost, R = retained, L = lost
<6> Oscillator drift discrete, P = pass, F = excessive drift detected
<7> Data collection discrete, C = collecting, null if not collecting
<8> Board temperature in degrees C
<9> Board configuration data, R = retained, L = lost

4.2.14 3D velocity Information (PGRMV)

The GARMIN Proprietary sentence $PGRMV reports three-dimensional velocity information.

$PGRMV,<1>,<2>,<3>*hh<CR><LF>
<1> True east velocity, -514.4 to 514.4 m/second
<2> True north velocity, -514.4 to 514.4 m/second
<3> Up velocity, -999.9 to 9999.9 m/second

4.2.15 DGPS Beacon Information (PGRMB)

The GARMIN proprietary sentence $PGRMB reports DGPS beacon information.

$PGRMB,<1>,<2>,<3>,<4>,<5>,K*<CR><LF>
<1> Beacon tune frequency, 0.0, 283.5 – 325.0 kHz in 0.5 kHz steps
<2> Beacon bit rate, 0, 25, 50, 100, or 200 bps
<3> Beacon SNR, 0 to 31
<4> Beacon data quality, 0 to 100
<5> Distance to beacon reference station in kilometers

Appendix E- GPS Communications Assembly Language Program

```
          CPU 68HC12
          PADDING OFF


PORTA     equ $00     ;Port A Data
PORTB     equ $01     ;Port B Data
DDRA      equ $02     ;Port A Data Direction
DDRB      equ $03     ;Port B Data Direction
SC0BDH    equ $C0     ;SCI 0 Baud Rate
SC0BDL    equ $C1     ;SCI 0 Baud Rate Low Byte
SC0CR1    equ $C2     ;SCI 0 Control 1
SC0CR2    equ $C3     ;SCI 0 Control 2
SC0SR1    equ $C4     ;SCI 0 Status 1
SC0SR2    equ $C5     ;SCI 0 Status 2
SC0DRH    equ $C6     ;SCI 0 Data
SC0DRL    equ $C7     ;SCI 0 Data Low Byte
SC1BDH    equ $C8     ;SCI 1 Baud Rate
SC1BDL    equ $C9     ;SCI 1 Baud Rate Low Byte
SC1CR1    equ $CA     ;SCI 1 Control 1
SC1CR2    equ $CB     ;SCI 1 Control 2
SC1SR1    equ $CC     ;SCI 1 Status 1
SC1SR2    equ $CD     ;SCI 1 Status 2
SC1DRH    equ $CE     ;SCI 1 Data
SC1DRL    equ $CF     ;SCI 1 Data Low Byte
PORTS     equ $D6     ;Port S Data
DDRS      equ $D7     ;Port S Data Direction
PURDS     equ $D9     ;
RDRF      equ $20
RAMTOP    equ  $0600     ;Top of usable RAM
RAMTOPMON equ  $07FF     ;Top of RAM used by monitor
CODE      equ  $8000

          ORG $0200

TABLE          RMB 1          ;ALWAYS LAST IN LIST

;=============================================================
; START OF MAIN PROGRAMME
     ORG  CODE

main lds       #RAMTOP   ;Stack Pointer
     clr       COPCTL    ;Disable Computer Operating
Properly Watchdog

     jsr       serial_init  ; Setup serial port
     jsr       spi_init
     ldaa      #23
     jsr       txbyte
```

```
     jsr       gps_serial_init ; Setup serial port for GPS
               comms

MAINTEST
     ldaa      #96
     jsr       txbyte
     jsr       get_gps_data
     bra       MAINTEST

;*******************************************************
;Print a space

PRINT_SPACE
     ldaa      #' '
     jsr       txbyte
     rts

;*******************************************************
; To use the serial port, one must set the baud rate. Set
; it to 19200 baud. The value here is a 16 bit divisor.

serial_init
     ldd       #26  ; Value from Baud Rate Generation Table
     std       SC0BDH
     ldaa      #$0C ; Enable transceiver
     staa      SC0CR2
     rts

;*******************************************************
; To use gps serial port, one must set the baud rate. Set
; it to 4800 baud. The value here is a 16 bit divisor.

gps_serial_init
     ldd       #104 ; Value from Baud Rate Generation Table
     std       SC1BDH
     ldaa      #$0C ; Enable transceiver
     staa      SC1CR2
     rts

;*******************************************************
; Set up the SPI to talk to the compass

spi_init
     ldaa      #$E0 ; Value from Baud Rate Generation Table
     std       DDRS
     ldaa      #$5C ; Enable transceiver
     staa      SP0CR1
     ldaa      #$00
     staa      SP0CR2
     ldaa      #$02
     staa      SP0BR
```

```
     rts

;*************************************************
; Send a carriage return and line feed to screen
write_newline
     ldaa      #$0D ;Load Carriage return ASCII
     jsr       txbyte
     ldaa      #$0A ;Load Line Feed ASCII
     jsr       txbyte
     rts

;*************************************************
get_gps_data
     clra
SCI_WAIT
     brclr     SC1SR1, RDRF, SCI_WAIT
     ldaa      SC1DRL
     JSR       txbyte
     ldaa      #96
     jsr       txbyte
     JSR       print_space
     rts

;*************************************************
; Transmit a character which is in A
txbyte
     brclr     SC0SR1, #$80 txbyte
     staa      SC0DRL
     rts

     END main
```

Appendix F- Raw Captured GPS Data

```
--- Start of recording: 28/03/2005 4:51:27 PM ---
  ` `$` `G` `P` `R` `M` `C` `,` `0` `6` `5` `1` `2` `9` `,` `A` `,` `2` `7` `3`
 `3` `.` `0` `3` `9` `0` `,` `S` `,` `1` `5` `1` `5` `7` `.` `6` `4` `2` `6` `,`
 `E` `,` `0` `0` `0` `.` `0` `,` `0` `0` `0` `.` `0` `,` `2` `8` `0` `3` `0` `5`
` `` `0` `1` `1` `.` `0` `,` `E` `*` `6` `0` `
  ` `$` `G` `P` `G` `G` `A` `,` `0` `6` `5` `1` `2` `9` `,` `2` `7` `3` `3` `.`
 `0` `3` `9` `0` `,` `S` `,` `1` `5` `1` `5` `7` `.` `6` `4` `2` `6` `,` `E` `,`
 `6` `,` `0` `3` `,` `4` `.` `8` `,` `2` `6` `0` `.` `8` `,` `M` `,` `3` `9` `.`
` `` `,` `M` `,` `,` `*` `5` `4` `
  ` `$` `G` `P` `G` `S` `A` `,` `A` `,` `2` `,` `,` `0` `7` `,` `,` `2` `4` `,`
 `,` `2` `8` `,` `,` `,` `,` `,` `,` `,` `,` `4` `.` `8` `,` `4` `.` `8` `,` `*` `1`
` `` `
  ` `$` `G` `P` `G` `S` `V` `,` `2` `,` `1` `,` `0` `6` `,` `0` `2` `,` `1` `3`
 `,` `3` `5` `0` `,` `,` `0` `7` `,` `6` `3` `,` `1` `6` `3` `,` `4` `9` `,` `0`
 `9` `,` `1` `7` `,` `2` `2` `0` `,` `,` `2` `4` `,` `4` `4` `,` `0` `8` `8` `,`
` `` `2` `*` `7` `E` `
  ` `$` `G` `P` `G` `S` `V` `,` `2` `,` `2` `,` `0` `6` `,` `2` `6` `,` `3` `0`
 `,` `2` `7` `6` `,` `4` `5` `,` `2` `8` `,` `3` `8` `,` `1` `1` `6` `,` `3` `7`
` `` `,` `,` `,` `,` `,` `,` `,` `*` `7` `9` `
  ` `$` `G` `P` `R` `M` `C` `,` `0` `6` `5` `1` `3` `0` `,` `A` `,` `2` `7` `3`
 `3` `.` `0` `3` `9` `0` `,` `S` `,` `1` `5` `1` `5` `7` `.` `6` `4` `2` `6` `,`
 `E` `,` `0` `0` `0` `.` `0` `,` `0` `0` `0` `.` `0` `,` `2` `8` `0` `3` `0` `5`
` `` `0` `1` `1` `.` `0` `,` `E` `*` `6` `8` `
  ` `$` `G` `P` `G` `G` `A` `,` `0` `6` `5` `1` `3` `0` `,` `2` `7` `3` `3` `.`
 `0` `3` `9` `0` `,` `S` `,` `1` `5` `1` `5` `7` `.` `6` `4` `2` `6` `,` `E` `,`
 `6` `,` `0` `3` `,` `4` `.` `8` `,` `2` `6` `0` `.` `8` `,` `M` `,` `3` `9` `.`
` `` `,` `M` `,` `,` `*` `5` `C` `
  ` `$` `G` `P` `G` `S` `A` `,` `A` `,` `2` `,` `,` `0` `7` `,` `,` `2` `4` `,`
 `,` `2` `8` `,` `,` `,` `,` `,` `,` `,` `,` `4` `.` `8` `,` `4` `.` `8` `,` `*` `1`
` `` `
  ` `$` `G` `P` `G` `S` `V` `,` `2` `,` `1` `,` `0` `6` `,` `0` `2` `,` `1` `3`
 `,` `3` `5` `0` `,` `,` `0` `7` `,` `6` `3` `,` `1` `6` `3` `,` `4` `8` `,` `0`
 `9` `,` `1` `7` `,` `2` `2` `0` `,` `,` `2` `4` `,` `4` `4` `,` `0` `8` `8` `,`
` `` `1` `*` `7` `C` `
  ` `$` `G` `P` `G` `S` `V` `,` `2` `,` `2` `,` `0` `6` `,` `2` `6` `,` `3` `0`
 `,` `2` `7` `6` `,` `4` `4` `,` `2` `8` `,` `3` `8` `,` `1` `1` `6` `,` `3` `6`
` `` `,` `,` `,` `,` `,` `,` `,` `,` `*` `7` `9` `
TwinPEEKs V1.6a for Card12.D60A
(C)1996-2001 by MCT Elektronikladen GbR
The makers of fine HC12/11/08 stuff!
http://www.elektronikladen.de/mct
mailto:leipzig@elektronikladen.de
;->
--- End of recording: 28/03/2005 4:53:23 PM ---
```

Appendix G- HC12 Source Code

G.1 navigationalgorithm.c

G.2 hc12.h

G.3 GPSMessageHolder.h

G.4 GPSMessageHolder.c

G.5 compass.h

G.6 compass.c

G.7 can.h

G.8 can.c

```
//--------------------------------------------------------------------------
// Author:  Soz Knox 0039911442
// Title:   Navigation Algorithm for UAV
/* Purpose: Calculates distance in metres and desired heading in degrees
            from current position to next waypoint. Also calculates heading
            error for Flight Control System (rudder control).
*/
// Last updated: 20/09/05

//--------------------------------------------------------------------------
// Included Libraries

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "hc12.h"
#include "GPSMessageHolder.h"
#include "compass.h"
#include "can.h"

//--------------------------------------------------------------------------
//Global variables
#define TRUE 1
#define FALSE 0

// Math constants

gpsData data;
double M_PI = 3.14159;
double actualHeading;
double distanceToWaypointRad;
double desiredHeadingRad;
double desiredHeadingDeg;
double distanceToWaypointNM;
double distanceToWaypointMetres;
double headingError;
double destLat1; //These are the waypoints we get from the Control Station
double destLon1;
double destLat2;
double destLon2;
double destLat3;
double destLon3;
double destLat4;
double destLon4;
double curWptLat;//Actual latitude of waypoint, UAV is heading to
double curWptLon;// Actual longitude of waypoint, UAV is heading to
int curWpt;// Between 1 and 4 depending on which waypoint UAV is up to
int craigHead;
//--------------------------------------------------------------------------
// Function prototypes

void calculateBearingAndDistance();
void checkDistance();
void sendDataToGCS();
double greatCircleDistance(double lat1, double lat2, double lon1,
                           double lon2);
double radBearing(double lat1, double lat2, double lon1, double lon2,
                  double distance);
double deg2Rad(double coOrd);
void calculateHeadingError();
void getWaypointsFromGCS();
void currentWaypoint();
void trigger();

//--------------------------------------------------------------------------

void main(void) {
    int numWpts = 4;

    //Settings for comms with GCS
    SC0CR1 = 0x00;
    SC0CR2 = 0x0C;
    SC0BDH = 0x00;
    SC0BDL = 0x34;
    DDRB = 0x01; //sets PB0 an output with a high

    PORTB = 0x01;//set PB0 high
```

```
    curWpt = 1;

    compass_power_up();//Initialises compass
    CANInit();

    compassCal();

    getWaypointsFromGCS();

    currentWaypoint();//which point is the UAV heading towards?

    //This loop will run until we pass the final waypoint
    while ( curWpt <= numWpts ) {
        //get data from the GPS
        data = acquireGpsData();
        //Work out how far we are from the waypoint and what
        //direction we need to go to get there
        calculateBearingAndDistance();
        //Work out where we are heading currently then calculate
        //heading error
        calculateHeadingError();
        //Check how far we are away from the waypoint
        //if < 5 increment the waypoint, and take a photo
        checkDistance();
        //send current telemetry data to the GCS
        sendDataToGCS();
    }
}

//------------------------------------------------------------------------
// Functions

void calculateBearingAndDistance() {
    double lat1Rad;
    double lon1Rad;
    double lat2Rad;
    double lon2Rad;

    lat1Rad = deg2Rad(data.lat); //current lat from the GPS
    lon1Rad = deg2Rad(data.lon); //current lon from the GPS
    lat2Rad = deg2Rad(curWptLat); //current waypoint lat
    lon2Rad = deg2Rad(curWptLon); //current waypoint lon

    distanceToWaypointRad = greatCircleDistance(lat1Rad, lat2Rad,
                            lon1Rad, lon2Rad);
    desiredHeadingRad = radBearing(lat1Rad, lat2Rad, lon1Rad,
                            lon2Rad, distanceToWaypointRad);
    desiredHeadingDeg = (180/M_PI) * desiredHeadingRad;
    distanceToWaypointNM = ((180*60)/M_PI)*distanceToWaypointRad;
    distanceToWaypointMetres = distanceToWaypointNM*1852;
}

//Function to calculate when to take a photo and when to start heading to
//next waypoint (increment curWpt)
void checkDistance() {
    if (distanceToWaypointMetres <= 5) {
        trigger();
        putchar('*');
        curWpt++;
        currentWaypoint();
    }
}

//Telemetry to Ground Control Station
void sendDataToGCS() {
    printf( "%f\\%f\\%f\\%f\\%f\\%f\\%s", data.lat, data.lon,
        data.antHeight, desiredHeadingDeg,actualHeading,
        distanceToWaypointMetres,data.UTC);
}

double greatCircleDistance(double lat1, double lat2, double lon1,
                           double lon2) {
    double radDist;
    double t1 = pow(sin((lat1-lat2)/2),2);
    double t2 = pow(sin((lon1-lon2)/2),2);
    radDist = 2*asin(sqrt(t1 + cos(lat1)*cos(lat2)*t2));
```

```
    return radDist;
}

double radBearing(double lat1, double lat2, double lon1, double lon2,
                  double distance) {
    double rad_Bearing;

    if (sin(lon2-lon1) < 0.0)
    {
        rad_Bearing = acos((sin(lat2)-sin(lat1)*cos(distance))/
                      (sin(distance)*cos(lat1)));
    }
    else
    {
        rad_Bearing = 2*M_PI-acos((sin(lat2)-sin(lat1)*cos(distance))/
                      (sin(distance)*cos(lat1)));
    }

    return rad_Bearing;
}

double deg2Rad(double coOrd) {
    int dd  = (int)coOrd/100;
    double ee = dd*100;
    double mm = coOrd-ee;
    double oDDD = mm/60;
    double ddDDDD = oDDD+dd;

    double coOrdRad = ddDDDD/57.2957;
    return coOrdRad;
}

void calculateHeadingError() {
    int j;
    unsigned char *s;

    for (j = 0; j<4; j++) {
        actualHeading = compass();
        headingError = desiredHeadingDeg - actualHeading;
        if (headingError < -180)
            headingError += 360;

        if (headingError > 180)
            headingError -= 360;

        //scale heading before sending to Craig
        craigHead = 12000 + (1146/180*(int)headingError);
        itoa(s, craigHead, 10);
        //put the heading onto the CAN bus
        tx_can(s);
    }
}

void getWaypointsFromGCS() {
    unsigned char inputQueue[MAX_SIZE]= {"\0"};
    unsigned char bufferString[100]= {"\0"};
    unsigned char latitudeString[10]= {"\0"};
    unsigned char longitudeString[11]= {"\0"};
    unsigned char *locptr;
    unsigned char *s;   //local buffer pointer
    unsigned char comPortChar;

    int i = 0;
    int flag = 1;
    int index = 0;
    unsigned int x = 0, k = 0;

    do
    {
        //Wait for Receive register full flag to be set
        while((SC0SR1 & 0x20) == 0);
        comPortChar = SC0DRL;

        i = 0;
        inputQueue[i]=comPortChar;
        flag = 0;
```

```
    do
    {
        //Wait for Receive register full flag to be set
        while((SC0SR1 & 0x20) == 0);

        comPortChar = SC0DRL;
        i++;
        inputQueue[i]=comPortChar;
        flag = 0;
        //Termination string for Wpt data from the GCS
    } while ((comPortChar != '!' ));

} while(flag);

//points locptr to the same location that inputQueue points to
locptr = inputQueue;

while (*(locptr+x)!= 0x2C) // not equal to a comma
{
    latitudeString[k] = *(locptr + x);
    x++;
    k++;
}

latitudeString[k] = '\0';
s = latitudeString;
destLat1 = atof(s);

locptr = inputQueue;

x+=1;
k=0;
while(*(locptr+x) != 0x2C)
{
    longitudeString[k] = *(locptr + x);
    x++;
    k++;
}

longitudeString[k] = '\0';
s = longitudeString;

destLat2 = atof(s);

locptr = inputQueue;
x+=1;
k=0;

while (*(locptr+x)!= 0x2C) // not equal to a comma
{
    latitudeString[k] = *(locptr + x);
    x++;
    k++;
}

latitudeString[k] = '\0';
s = latitudeString;
destLat3 = atof(s);

locptr = inputQueue;

x+=1;
k=0;
while(*(locptr+x) != 0x2C)
{
    longitudeString[k] = *(locptr + x);
    x++;
    k++;
}

longitudeString[k] = '\0';
s = longitudeString;

destLat4 = atof(s);

locptr = inputQueue;
x+=1;
```

```
    k=0;
    while (*(locptr+x)!= 0x2C) // not equal to a comma
    {
        latitudeString[k] = *(locptr + x);
        x++;
        k++;
    }

    latitudeString[k] = '\0';
    s = latitudeString;
    destLon1 = atof(s);

    locptr = inputQueue;

    x+=1;
    k=0;
    while(*(locptr+x) != 0x2C)
    {
        longitudeString[k] = *(locptr + x);
        x++;
        k++;
    }

    longitudeString[k] = '\0';
    s = longitudeString;

    destLon2 = atof(s);

    locptr = inputQueue;
    x+=1;
    k=0;
    while (*(locptr+x)!= 0x2C) // not equal to a comma
    {
        latitudeString[k] = *(locptr + x);
        x++;
        k++;
    }

    latitudeString[k] = '\0';
    s = latitudeString;
    destLon3 = atof(s);

    locptr = inputQueue;

    x+=1;
    k=0;
    while(*(locptr+x) != 0x2C)
    {
        longitudeString[k] = *(locptr + x);
        x++;
        k++;
    }

    longitudeString[k] = '\0';
    s = longitudeString;

    destLon4 = atof(s);
    putchar('!');
}

void currentWaypoint () {
    //switch case to decide which lat is current one
    switch (curWpt) {
    case 1:

        curWptLat = destLat1;
        curWptLon = destLon1;

        break;

    case 2:

        curWptLat = destLat2;
        curWptLon = destLon2;

        break;
```

```
    case 3:

        curWptLat = destLat3;
        curWptLon = destLon3;

        break;

    case 4:

        curWptLat = destLat4;
        curWptLon = destLon4;

        break;

    default:
        printf("Houston we have a problem!!\n");
    }
}


void trigger() {
    //send a low to take a photo
    PORTB = 0x00;
    delay(2000);

    PORTB = 0x01;
}

//---------------------------------------------------------------------------
```

```
//  filename ******* HC12.H **************
//  Header file for 6812 I/O ports
//  This example accompanies the book
//  "Embedded Microcomputer Systems: Real Time Interfacing", Brooks-Cole,
//  copyright (c) 2000,
//  Jonathan W. Valvano 5/4/99
//  msCAN port addresses added 08/09/05 by Soz Knox

#ifndef __HC12_H
#define __HC12_H    1

/* base address of register block, change this if you relocate the register
 * block. This is for 812A4, 912B32 contains a subset.
 */
#define _IO_BASE    0
#define _P(off)     *(unsigned char volatile *)(_IO_BASE + off)
#define _LP(off)    *(unsigned short volatile *)(_IO_BASE + off)
#define PORTA   _P(0x00)
#define PORTB   _P(0x01)
#define DDRA    _P(0x02)
#define DDRB    _P(0x03)
#define PORTC   _P(0x04)
#define PORTD   _P(0x05)
#define DDRC    _P(0x06)
#define DDRD    _P(0x07)
#define PORTE   _P(0x08)
#define DDRE    _P(0x09)
#define PEAR    _P(0x0A)
#define MODE    _P(0x0B)
#define PUCR    _P(0x0C)
#define RDRIV   _P(0x0D)
#define INITRM  _P(0x10)
#define INITRG  _P(0x11)
#define INITEE  _P(0x12)
#define MISC    _P(0x13)
#define RTICTL  _P(0x14)
#define RTIFLG  _P(0x15)
#define COPCTL  _P(0x16)
#define COPRST  _P(0x17)
#define ITST0   _P(0x18)
#define ITST1   _P(0x19)
#define ITST2   _P(0x1A)
#define ITST3   _P(0x1B)
#define INTCR   _P(0x1E)
#define HPRIO   _P(0x1F)
#define KWIED   _P(0x20)
#define KWIFD   _P(0x21)
#define PORTH   _P(0x24)
#define DDRH    _P(0x25)
#define KWIEH   _P(0x26)
#define KWIFH   _P(0x27)
#define PORTJ   _P(0x28)
#define DDRJ    _P(0x29)
#define KWIEJ   _P(0x2A)
#define KWIFJ   _P(0x2B)
#define KPOLJ   _P(0x2C)
#define PUPSJ   _P(0x2D)
#define PULEJ   _P(0x2E)
#define PORTF   _P(0x30)
#define PORTG   _P(0x31)
#define DDRF    _P(0x32)
#define DDRG    _P(0x33)
#define DPAGE   _P(0x34)
#define PPAGE   _P(0x35)
#define EPAGE   _P(0x36)
#define WINDEF  _P(0x37)
#define MXAR    _P(0x38)
#define CSCTL0  _P(0x3C)
#define CSCTL1  _P(0x3D)
#define CSSTR0  _P(0x3E)
#define CSSTR1  _P(0x3F)
#define LDV     _LP(0x40)
#define RDV     _LP(0x42)
#define CLKCTL  _P(0x47)
#define ATDCTL0 _P(0x60)
#define ATDCTL1 _P(0x61)
#define ATDCTL2 _P(0x62)
```

```
#define ATDCTL3 _P(0x63)
#define ATDCTL4 _P(0x64)
#define ATDCTL5 _P(0x65)
#define ATDSTAT _LP(0x66)
#define ATDTEST _LP(0x68)
#define PORTAD  _P(0x6F)
#define ADR0H   _P(0x70)
#define ADR1H   _P(0x72)
#define ADR2H   _P(0x74)
#define ADR3H   _P(0x76)
#define ADR4H   _P(0x78)
#define ADR5H   _P(0x7A)
#define ADR6H   _P(0x7C)
#define ADR7H   _P(0x7E)
#define TIOS    _P(0x80)
#define CFORC   _P(0x81)
#define OC7M    _P(0x82)
#define OC7D    _P(0x83)
#define TCNT    _LP(0x84)
#define TSCR    _P(0x86)
#define TQCR    _P(0x87)
#define TCTL1   _P(0x88)
#define TCTL2   _P(0x89)
#define TCTL3   _P(0x8A)
#define TCTL4   _P(0x8B)
#define TMSK1   _P(0x8C)
#define TMSK2   _P(0x8D)
#define TFLG1   _P(0x8E)
#define TFLG2   _P(0x8F)
#define TC0     _LP(0x90)
#define TC1     _LP(0x92)
#define TC2     _LP(0x94)
#define TC3     _LP(0x96)
#define TC4     _LP(0x98)
#define TC5     _LP(0x9A)
#define TC6     _LP(0x9C)
#define TC7     _LP(0x9E)
#define PACTL   _P(0xA0)
#define PAFLG   _P(0xA1)
#define PACNT   _LP(0xA2)
#define TIMTST  _P(0xAD)
#define PORTT   _P(0xAE)
#define DDRT    _P(0xAF)
#define SC0BD   _LP(0xC0)
#define SC0BDH  _P(0xC0)
#define SC0BDL  _P(0xC1)
#define SC0CR1  _P(0xC2)
#define SC0CR2  _P(0xC3)
#define SC0SR1  _P(0xC4)
#define SC0SR2  _P(0xC5)
#define SC0DRH  _P(0xC6)
#define SC0DRL  _P(0xC7)
#define SC1BD   _LP(0xC8)
#define SC1BDH  _P(0xC8)
#define SC1BDL  _P(0xC9)
#define SC1CR1  _P(0xCA)
#define SC1CR2  _P(0xCB)
#define SC1SR1  _P(0xCC)
#define SC1SR2  _P(0xCD)
#define SC1DRH  _P(0xCE)
#define SC1DRL  _P(0xCF)
#define SP0CR1  _P(0xD0)
#define SP0CR2  _P(0xD1)
#define SP0BR   _P(0xD2)
#define SP0SR   _P(0xD3)
#define SP0DR   _P(0xD5)
#define PORTS   _P(0xD6)
#define DDRS    _P(0xD7)
#define EEMCR   _P(0xF0)
#define EEPROT  _P(0xF1)
#define EETST   _P(0xF2)
#define EEPROG  _P(0xF3)
#define CMCR0   _P(0x0100)
#define CMCR1   _P(0x0101)
#define CBTR0   _P(0x0102)
#define CBTR1   _P(0x0103)
#define CRFLG   _P(0x0104)
```

```
#define CRIER   _P(0x0105)
#define CTFLG   _P(0x0106)
#define CTCR    _P(0x0107)
#define CIDAC   _P(0x0108)
#define CRXERR  _P(0x010E)
#define CTXERR  _P(0x010F)
#define CIDAR0  _P(0x0110)
#define CIDAR1  _P(0x0111)
#define CIDAR2  _P(0x0112)
#define CIDAR3  _P(0x0113)
#define CIDMR0  _P(0x0114)
#define CIDMR1  _P(0x0115)
#define CIDMR2  _P(0x0116)
#define CIDMR3  _P(0x0117)
#define CIDAR4  _P(0x0118)
#define CIDAR5  _P(0x0119)
#define CIDAR6  _P(0x011A)
#define CIDAR7  _P(0x011B)
#define CIDMR4  _P(0x011C)
#define CIDMR5  _P(0x011D)
#define CIDMR6  _P(0x011E)
#define CIDMR7  _P(0x011F)
#define PCTLCAN _P(0x013D)
#define PORTCAN _P(0x013E)
#define DDRCAN  _P(0x013F)
#define RxFG    _P(0x0140)
#define Tx0     _P(0x0150)
#define Tx1     _P(0x0160)
#define Tx2     _P(0x0170)

/* These values are for a 8Mhz clock
 */
typedef enum {
    BAUD38K = 13, BAUD19K = 26, BAUD14K = 35,
    BAUD9600 = 52, BAUD4800 = 104, BAUD2400 = 208,
    BAUD1200 = 417, BAUD600 = 833, BAUD300 = 2273
    } BaudRate;

void setbaud(BaudRate);

#ifndef INTR_ON
#define INTR_ON()   asm("cli")
#define INTR_OFF()  asm("sei")
#endif

#ifndef bit
#define bit(x)  (1 << (x))
#endif

#ifdef _SCI
/* SCI bits */
#define TE      bit(3)
#define RE      bit(2)
#define TDRE    bit(7)
#define TC      bit(6)
#define RDRF    bit(5)
#define T8      bit(6)
#define R8      bit(7)
#endif

#ifdef _SPI
/* SPI bits */
#define MSTR    bit(4)
#define SPE     bit(6)
#define SPIF    bit(7)
#endif

#ifdef _EEPROM
/* EEPROM */
#define EEPGM   bit(0)
#define EELAT   bit(1)
#endif

#endif
```

```
#ifndef _GPSMESSAGEHOLDER_

#include "hc12.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>


#define MAX_SIZE 100

typedef struct
        {
            char UTC[10];
            double lat;
            char NS;
            double lon;
            char EW;
            int fixIndication;
            double antHeight;
        } gpsData;

gpsData acquireGpsData();
int putchar(char x);
void SCI_Output(char data);

#endif
```

```
//---------------------------------------------------------------------------
// Author:  Soz Knox 0039911442
// Title:   GPS message Holder
/* Purpose: Reads in NMEA data from GPS and extracts required $GPGGA
            message
*/
// Last updated: 29/05/05

//---------------------------------------------------------------------------
// Included Libraries

#include "GPSMessageHolder.h"
#define STRING10 10
#define STRING7 7

gpsData GPRMC;

//---------------------------------------------------------------------------
// Functions

gpsData acquireGpsData() {
    unsigned char inputQueue[MAX_SIZE]= {"\0"};
    unsigned char bufferString[MAX_SIZE]= {"\0"};
    unsigned char latitudeString[10]= {"\0"};
    unsigned char longitudeString[11]= {"\0"};
    unsigned char fixString[2] = {"\0"};
    unsigned char dateString[7] = {"\0"};
    unsigned char antHeightString[8] = {"\0"};

    unsigned char *locptr;
    unsigned char *s;   //local buffer pointer
    unsigned char comPortChar;

    int flag = 1;

    int i = 0;
    int index = 0;
    unsigned int x = 0, k = 0;

    SC1BDH = 0x00; //set SCI baud to 4800 bps
    SC1BDL = 0x68;
    SC1CR2 = 0x0C;

    do
    {
        //Wait for Receive reg full flag to be set
        while((SC1SR1 & 0x20) == 0);
        comPortChar = SC1DRL;

        if(comPortChar == '$') //if character is '$', then store in array
        {
            i = 0;
            inputQueue[i]=comPortChar;

            do
            {
                //Wait for Receive reg full flag to be set
                while((SC1SR1 & 0x20) == 0);
                comPortChar = SC1DRL;
                i++;
                inputQueue[i]=comPortChar;
            }   while ((comPortChar != '\r') && (comPortChar != '\n' ));

            inputQueue[i+1]='\0';
            // Analyse the string to see if it is $GPGGA
            x = 0;
            locptr = inputQueue;//points locptr to inputQueue data

            while (*(locptr+x)!= 0x2C) // not equal to a comma
            {
                bufferString[x] = *(locptr + x);
                x++;
            }
            bufferString[x] = '\0';
            // now check to see if string is the $GPGGA message

            if (bufferString[3] == 'G' && bufferString[4] == 'G' &&
```

```
    bufferString[5] == 'A')
{
    locptr = inputQueue;
    //Get UTC time
    x = 7; // beginning of UTC field
    k = 0;
    while (*(locptr+x) != 0x2C)
    {
        bufferString[k] = *(locptr + x);
        x++;
        k++;
    }
    bufferString[k]='\0';
    for (index = 0; index <= k; index++)
    {
        GPRMC.UTC[index]=bufferString[index];
    }

    locptr = inputQueue;

    // get Latitude
    x+=1;
    k=0;
    while(*(locptr+x) != 0x2C)
    {
        latitudeString[k] = *(locptr + x);
        x++;
        k++;
    }

    latitudeString[k] = '\0';
    s = latitudeString;

    GPRMC.lat = atof(s);

    locptr = inputQueue;

    //get Latitude Hemisphere
    x+=1;
    k = 0;
    while (*(locptr+x) != 0x2C)
    {
        bufferString[k] = *(locptr + x);
        x++;
        k++;
    }
    GPRMC.NS = bufferString[k-1];

    locptr = inputQueue;

    //get Longitude

    x+=1;
    k=0;
    while(*(locptr+x) != 0x2C)
    {
        longitudeString[k] = *(locptr + x);
        x++;
        k++;
    }

    longitudeString[k] = '\0';
    s = longitudeString;

    GPRMC.lon = atof(s);

    locptr = inputQueue;

    // Get longitude hemisphere
    x+=1;
    k = 0;
    while (*(locptr+x) != 0x2C)
    {
        bufferString[k] = *(locptr + x);
        x++;
        k++;
    }
```

```
                    GPRMC.EW = bufferString[k-1];

                    locptr = inputQueue;

                    //Get fix quality indication
                    x+=1;
                    k=0;
                    while(*(locptr+x) != 0x2C)
                    {
                        fixString[k] = *(locptr + x);
                        x++;
                        k++;
                    }
                    fixString[k] = '\0';
                    s = fixString;

                    GPRMC.fixIndication = atoi(s);
                    locptr = inputQueue;

                    //Get height above ground
                    x=23;
                    k=0;
                    while(*(locptr+x) != 0x2C)
                    {
                        antHeightString[k] = *(locptr + x);
                        x++;
                        k++;
                    }
                    antHeightString[k] = '\0';
                    s = antHeightString;

                    GPRMC.antHeight = atof(s);

                    locptr = inputQueue;

                    flag = 0;

                }//if statement
            }//Another if statement
        } while ( flag );
        return GPRMC;
}

int putchar(char x) {
    while((SC0SR1&0x80)==0);

    SC0DRL=x;

    if(x=='\n') {
        while((SC0SR1&0x80)==0);
        SC0DRL='\r';
    }
}

//-------------------------------------------------------------------------
```

```
#ifndef _COMPASS_

#include "hc12.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

double compass();
void compass_power_up();
void delay(unsigned int ms);
void compassCal();

#endif
```

```
//--------------------------------------------------------------------------
// Author:  Soz Knox 0039911442
// Title:   Compass Interface
/* Purpose: Initialises SPI for compass readings and returns current
            Heading
*/
// Last updated: 16/08/05

//--------------------------------------------------------------------------
// Included Libraries

#include "compass.h"
//--------------------------------------------------------------------------
//Global variables

//Defining a constant to use in the delay function
const int DELAYCONST=(8000/4);

//--------------------------------------------------------------------------
// Function prototypes
void delay(unsigned int ms);

//--------------------------------------------------------------------------

void compass_power_up() {
    DDRA = 0x0F; //sets port A for: PA0-3 to outputs and the rest inputs

    PORTA = 0x0F; //sets /RESET, /CAL, /SS and /P/C high for power up

    //SPI setup in HC12
    SP0CR1 = 0x5E; //SPI enable, Master mode, clock idle HIGH, Phase Hi,
    //Data is transferred MSB first. Also. SS is high to enable an output.

    SP0CR2 = 0x00; //Not bidirectional, normal mode

    SP0BR = 0x02; //SPI clock set to 1 MHz

    DDRS = 0xD0; // SCLK, MISO outputs.

    //Power Up.
    PORTA = 0x07; //set RESET low while keeping /P/C, /SS and /CAL high.
    delay(100); //delay 100ms
    PORTA = 0x0F; //toggle RESET high again
    delay(750); //delay 750ms
}

double compass() {
    unsigned int head = 0x0000;
    unsigned int temp = 0x0000;
    unsigned char heading[2];

    //Power Up.
    PORTA = 0x07; //set RESET low keeping /P/C, /SS and /CAL high.
    delay(16); //delay 10ms
    PORTA = 0x0F; //toggle RESET high again
    delay(16); //delay 10ms

    //retrieve data
    PORTA &= 0x0E; //Set /P/C low
    delay(10); //delay 10 ms to keep /P/C low for 10ms

    while ((PORTA & 0x40) == 0x01);//wait for EOC to go low.

    PORTA |= 0x01; //Set /P/C high

    while ((PORTA & 0x40) == 0x00);//wait for EOC to go HIGH

    delay(10);

    PORTA  &= 0x0D; //lower /SS

    delay(10);

    SP0DR = 0xFF; //Initialise transfer

    //wait for register to fill
    while((SP0SR & 0x80)==0);
```

```
    heading[0] = SP0DR; //fill heading array
    delay(5);
    SP0DR = 0xFF; //initialise second transfer

    //wait for register to fill
    while((SP0SR & 0x80) == 0);

    heading[1] = SP0DR;
    if (heading[0] & 0x01) {
        head = 256 + heading[1];
    }
    else {
        head = heading[1];
    }

    PORTA |= 0x02;// raise /SS
    return (double)head;
}

//Delay function
void delay(unsigned int ms) {
    int i;
    while (ms > 0) {
        i=DELAYCONST;
        while(i > 0) {
            i--;
        }
        ms--;
    }
}

void compassCal() {

    PORTA = 0x0B; // sets CAL low and leaves the others high
    delay(1); //delay 100ms
    PORTA = 0x0F; //toggle CAL high again
    putchar('S');
    delay(10000); // to allow time to rotate the system 180 degrees
    PORTA = 0x0B; // toggle CAL low again
    delay(15); //hold for a minimum of 10ms
    PORTA = 0x0F; //toggle CAL high again
}

//---------------------------------------------------------------------------
```

```
#ifndef _CAN_

#include "hc12.h"

void CANInit(void);
void tx_can(char *ptr);

#endif
```

```
//------------------------------------------------------------------------
// Author:  Soz Knox 0039911442
// Title:   CAN inter-processor set up
/* Purpose: Sets up the CAN bus for transmission of heading error in
            format required by Autmatic Flight Control System
*/
// Last updated: 20/09/05
//------------------------------------------------------------------------
// Included Libraries
#include "can.h"
//------------------------------------------------------------------------

void tx_can(char *ptr)    //ptr to data buffer to be transmitted
{
    unsigned char test, mask, i, k;
    unsigned volatile char *pt;

    test=0;
    while(!test) {
        //depending on the transmit buffer free, the char string gets put
        //into the appropriate transmit buffer
        if(CTFLG & 0x01) {
            pt = &Tx0;
            mask = 0x01;
            test = 1;
        }
        else if(CTFLG & 0x02) {
            pt = &Tx1;
            mask = 0x02;
            test = 1;
        }
        else if(CTFLG & 0x04) {
            pt = &Tx2;
            mask = 0x04;
            test = 1;
        }
        else
            test = 0;
    }
    //assigning values to the other bit settings required in msCAN
    for(i = 0;i<4;i++)
        *pt++ = *ptr++;

    *(pt+8) = *(ptr+8);
    k = *(ptr+8);
    *(pt+9) = *(ptr+9);
    if (k) {
        for (i=0;i<k;i++)
            *pt++ = *ptr++;
    }
    //clear the transmit enable flag
    CTFLG = mask;
}

void CANInit(void)
{
     CMCR0 |=1; // set SFTRES to place CAN module in reset
     CBTR0=0xC1;
     CBTR1=0xB3; //timing for 500 kbps 16 MHz oscillation
     CTCR =0x00; //disables transmit interrupts

     //setting the identifier filters so that all messages will be received
     CIDMR0=0xFF;
     CIDMR1=0xFF;
     CIDMR2=0xFF;
     CIDMR3=0xFF;
     CIDMR4=0xFF;
     CIDMR5=0xFF;
     CIDMR6=0xFF;
     CIDMR7=0xFF;

     CMCR0 &=~1; // clear SFTRES to take CAN out of reset
     while (!(CMCR0 & 0x10)); // synch to CAN bus
}
//------------------------------------------------------------------------
```

Appendix H- User Interface Source Code

H.1 Main.java

H.2 GCSFrame.java

H.3 WaypointPanel.java

H.4 SerialConnection.java

H.5 SerialParameters.java

H.6 SerialConnectionException.java

H.7 UAVStatusPanel.java

H.8 TelemetryPane.java

H.9 SevenSegment.java

```java
/*
 * Main.java
 *
 * Created on 26 May 2005, 13:39
 */

package groundcontrolstation;

/**
 *
 * @author Soz
 */
public class Main {

    // private GCSFrame gcs;
    /**
      * @param args the command line arguments
      */
     public static void main(String[] args) {
         GCSFrame gcs = new GCSFrame();
     }

}
```

```
/*
 * GCSFrame.java
 *
 * This is the main frame that holds all the panels
 *
 * Created on 26 May 2005, 13:41
 */

package groundcontrolstation;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.io.*;
import java.util.*;

/**
 *
 * @author Soz
 */
public class GCSFrame extends JFrame{

    private WaypointPanel pane1 = new WaypointPanel( this );
    private UAVStatusPanel pane2 = new UAVStatusPanel( this );

    private TelemetryPane pane3 = new TelemetryPane();
    private JPanel panel = new JPanel();

    private int currWaypoint = 1;

    /** Creates a new instance of GCSFrame */
    public GCSFrame() {

        super();
        panel.setLayout( new GridLayout(1, 2));
        panel.add(pane1);
        panel.add(pane2);

        getContentPane().setLayout( new GridLayout(2,1));
        getContentPane().add(pane3);
        getContentPane().add(panel);

        setSize(1000,700);
        setTitle("Ground Control Station");
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    /*This method takes the data sent from the nav algorithm,
    splits it up into the individual pieces of information then
    displays it on the approprite part of the GUI
     */
    public void parseGPSData( String data ) {
        StringTokenizer str = new StringTokenizer(data, "\\");
        int i = 1;
        Float f;
        Double d;
        Integer j,x,y,z;
        String s;
        while (str.hasMoreTokens()) {
               switch(i) {
               case 1: //data.lat
                    s = str.nextToken();
                    pane3.updateLat(s);
                    i++;
                    break;
               case 2: //data.lon
                    s = str.nextToken();
                    pane3.updateLon(s);
                    i++;
                    break;
               case 3: //height
                    s = str.nextToken();
                    i++;
                    int len = s.length();
                    if (len == 1) {
                        j = new Integer(s);
                        pane3.updateHeight(0,0,j.intValue());
                    }
                    else if (len == 2) {
```

```
                        char c[] = s.toCharArray();
                        x = new Integer(c[0]);
                        y = new Integer(c[1]);
                        pane3.updateHeight(0,x.intValue(),
                              y.intValue());
                    }
                    else if (len == 3) {
                       char c[] = s.toCharArray();
                       x = new Integer(c[0]);
                       y = new Integer(c[1]);
                       z = new Integer(c[2]);
                       pane3.updateHeight(x.intValue(),y.intValue(),
                             z.intValue());
                    }
                    break;
               case 4: //desired Heading
                    d = new Double(str.nextToken());
                    double desiredHead = d.doubleValue();
                    pane3.updateDesiredHeading(desiredHead);
                    i++;
                    break;
               case 5: //actual Heading
                    d = new Double(str.nextToken());
                    double currHead = d.doubleValue();
                    pane3.updateHeading(currHead);
                    i++;
                    break;
               case 6: //distance to waypoint in metres
                    s = str.nextToken();
                    pane3.updateDist(s);
                    i++;
                    break;
               case 7:
                    s = str.nextToken();
                    pane2.updateStatus(s);
                    pane2.updateTelemetry(data);
               default:
                    return;
            }
        }
    }

    public void allSystemsGo() {
        JOptionPane.showMessageDialog(null, "Waypoints Received ",
        "Ready for Takeoff!", JOptionPane.INFORMATION_MESSAGE);
        pane2.GPSConnGood();
        pane1.waypointsStart();
    }

    public void updateHeading( double d ) {
        pane3.updateHeading(d);
    }

    public void waypointReached() {
        pane1.waypointReached(1);
        currWaypoint++;
    }
}
```

```
/*
 * WaypointPanel.java
 *
 * This panel contains all the fields etc required connect to
 * the HC12, enter 4 waypoints then uplink them to the HC12.
 *
 * Created on 26 May 2005, 13:43
 */

package groundcontrolstation;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;

/**
 *
 * @author Soz
 */
public class WaypointPanel extends JPanel{
    private GCSFrame parent;

    private JLabel label1 = new JLabel( "Waypoint One");
    private JLabel label2 = new JLabel("Waypoint Two");
    private JLabel label3 = new JLabel("Waypoint Three");
    private JLabel label4 = new JLabel("Waypoint Four");

    private JTextField field1 = new JTextField(10);
    private JTextField field2 = new JTextField(10);
    private JTextField field3 = new JTextField(10);
    private JTextField field4 = new JTextField(10);

    private JTextField field5 = new JTextField(10);
    private JTextField field6 = new JTextField(10);
    private JTextField field7 = new JTextField(10);
    private JTextField field8 = new JTextField(10);

    private JLabel label5 = new JLabel("Waypoint Reached");
    private JLabel label6 = new JLabel("Waypoint Reached");
    private JLabel label7 = new JLabel("Waypoint Reached");
    private JLabel label8 = new JLabel("Waypoint Reached");

    private JPanel pane2 = new JPanel();
    private JPanel pane3 = new JPanel();
    private JPanel pane4 = new JPanel();
    private JPanel pane5 = new JPanel();
    private JPanel pane6 = new JPanel();

    private JButton button1 = new JButton("Enter Waypoints");
    private JButton button2 = new JButton("Uplink Data to UAV");
    private JButton button3 = new JButton("Clear Waypoints");

    private String dataFromHC12;
    private SerialConnection connection;

    private Icon grayIcon = new ImageIcon(
            "Images/little-led-gray.gif");
    private Icon greenIcon = new ImageIcon(
            "Images/little-led-green.gif");
    private Icon redIcon = new ImageIcon(
            "Images/little-led-red.gif");


    /** Creates a new instance of WaypointPanel */
    public WaypointPanel(GCSFrame p) {
        initComponents();
        parent = p;
        dataFromHC12 = "";
    }

    public void initComponents() {
        field1.setEnabled(false);
        field2.setEnabled(false);
        field3.setEnabled(false);
        field4.setEnabled(false);
        field5.setEnabled(false);
        field6.setEnabled(false);
        field7.setEnabled(false);
```

```
        field8.setEnabled(false);
        field1.setText("2736.3400");
        field2.setText("2736.2900");
        field3.setText("2736.3200");
        field4.setText("2736.3600");
        field5.setText("15155.9100");
        field6.setText("15155.9500");
        field7.setText("15155.9800");
        field8.setText("15155.9500");

        pane2.setLayout(new FlowLayout( FlowLayout.RIGHT));
        pane2.add(label1);
        pane2.add(field1);
        pane2.add(field5);
        pane2.add(label5);
        pane2.setPreferredSize( new Dimension(500, 20));

        pane3.setLayout(new FlowLayout( FlowLayout.RIGHT));
        pane3.add(label2);
        pane3.add(field2);
        pane3.add(field6);
        pane3.add(label6);
        pane3.setPreferredSize( new Dimension(500, 20));

        pane4.setLayout(new FlowLayout( FlowLayout.RIGHT));
        pane4.add(label3);
        pane4.add(field3);
        pane4.add(field7);
        pane4.add(label7);
        pane4.setPreferredSize( new Dimension(500, 20));

        pane5.setLayout(new FlowLayout( FlowLayout.RIGHT));
        pane5.add(label4);
        pane5.add(field4);
        pane5.add(field8);
        pane5.add(label8);
        pane5.setPreferredSize( new Dimension(500, 20));

        button1.addActionListener(new EditListener());
        button2.addActionListener(new UplinkListener());
        button3.addActionListener(new ClearListener());
        pane6.add(button1);
        pane6.add(button2);
        pane6.add(button3);
        pane6.setPreferredSize( new Dimension(500,20));

        setLayout( new BoxLayout( this,  BoxLayout.Y_AXIS));
        this.add(pane2);
        this.add(pane3);
        this.add(pane4);
        this.add(pane5);
        this.add(pane6);

        setBorder( new TitledBorder(
                new LineBorder(Color.BLACK), "Waypoint Data"));
        label5.setIcon(grayIcon);
        label6.setIcon(grayIcon);
        label7.setIcon(grayIcon);
        label8.setIcon(grayIcon);
    }

    public void openComms( String waypointData ){
        connection = new SerialConnection(parent,
                new SerialParameters());
        try {
                connection.openConnection();
            } catch (SerialConnectionException e2) {
                JOptionPane.showMessageDialog(null, "alert",
                        "No link to UAV found",
                        JOptionPane.ERROR_MESSAGE);
                return;
            }
        connection.sendData(waypointData);
    }

    //Check that the data entered by the user is in the correct form,
    //and there is enough data to define 4 waypoints
    public String getWaypointData() {
        String data = "x";
        if ( checkLength(field1.getText())
```

```
        || checkLength(field2.getText())
        || checkLength(field3.getText())
        || checkLength(field4.getText())
        || checkLength(field5.getText())
        || checkLength(field6.getText())
        || checkLength(field7.getText())
        || checkLength(field8.getText()) ) {
            JOptionPane.showMessageDialog(null,
                    "Please enter four waypoints", "alert",
                    JOptionPane.ERROR_MESSAGE);
            return data;
        }
        if ( checkFormatLat(field1.getText())
        && checkFormatLat(field2.getText())
        && checkFormatLat(field3.getText())
        && checkFormatLat(field4.getText())
        && checkFormatLon(field5.getText())
        && checkFormatLon(field6.getText())
        && checkFormatLon(field7.getText())
        && checkFormatLon(field8.getText()) ) {
            data = field1.getText() + "," + field2.getText() +
                    "," + field3.getText() + "," + field4.getText() +
                    "," + field5.getText() + "," + field6.getText() +
                    "," + field7.getText() + "," + field8.getText() +
                    "!";
        }
        return data;
    }

    public boolean checkLength(String s) {
        if ( s.compareTo("") == 0 )
            return true;
        else
            return false;
    }

    public boolean checkFormatLat(String s) {
        if (s.length() != 9)
            return false;
        Character c = new Character(s.charAt(4));
        if (c.compareTo(new Character('.')) != 0)
            return false;
        return true;
    }

    public boolean checkFormatLon(String s) {
        if (s.length() != 10)
            return false;
        Character c = new Character(s.charAt(5));
        if (c.compareTo(new Character('.')) != 0)
            return false;
        return true;
    }

    public void waypointReached( int i ) {
        switch(i) {
            case 1:
               label5.setIcon(greenIcon);
               break;
            case 2:
                label6.setIcon(greenIcon);
               break;
            case 3:
                label7.setIcon(greenIcon);
               break;
            case 4:
                label8.setIcon(greenIcon);
               break;
            default:
                break;
        }
    }

    public void waypointsStart() {
        label5.setIcon(redIcon);
        label6.setIcon(redIcon);
        label7.setIcon(redIcon);
        label8.setIcon(redIcon);
    }
```

```
    public class ClearListener implements ActionListener {
        public void actionPerformed( ActionEvent e){
            field1.setText("");
            field2.setText("");
            field3.setText("");
            field4.setText("");
            field5.setText("");
            field6.setText("");
            field7.setText("");
            field8.setText("");
        }
    }

     public class UplinkListener implements ActionListener {
        public void actionPerformed( ActionEvent e){
            String s = getWaypointData();
            String temp = "";
            if (s.compareTo("x") == 0) {
                JOptionPane.showMessageDialog(null, "Wrong format",
                        "alert", JOptionPane.ERROR_MESSAGE);
            }
            else {
               temp = field1.getText() + "/" + field2.getText() + "\n"
                       + field3.getText() + "/" + field4.getText() +
                       "\n" + field5.getText() + "/" +
                       field6.getText() + "\n" +
                       field7.getText() + "/" + field8.getText();
            }
               int n = JOptionPane.showConfirmDialog(null,
                       "Do you wish to uplink these waypoints: \n" +
                       temp, "choose one", JOptionPane.YES_NO_OPTION);
               if ( n == JOptionPane.YES_OPTION ) {
                    openComms(s);
               }
        }
    }

    public class EditListener implements ActionListener {
        public void actionPerformed( ActionEvent e){
            field1.setEnabled(true);
            field2.setEnabled(true);
            field3.setEnabled(true);
            field4.setEnabled(true);
            field5.setEnabled(true);
            field6.setEnabled(true);
            field7.setEnabled(true);
            field8.setEnabled(true);
        }
    }
}
```

```
package groundcontrolstation;
/* @(#)SerialConnection.java    1.6 98/07/17 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license
 * appear on all copies of the software; and ii) Licensee does not
 * utilize the software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN
 * AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS
 * BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line
 * control of aircraft, air traffic, aircraft navigation or
 * aircraft communications; or in the design, construction,
 * operation or maintenance of any nuclear facility. Licensee
 * represents and warrants that it will not use or redistribute
 * the Software for such purposes.
 */

import javax.comm.*;
import java.io.*;
import java.awt.TextArea;
import java.awt.event.*;
import java.util.TooManyListenersException;

/**
A class that handles the details of a serial connection. Reads from
one
TextArea and writes to a second TextArea.
Holds the state of the connection.
*/
public class SerialConnection implements SerialPortEventListener,
                                         CommPortOwnershipListener {
    private GCSFrame parent;

    //private TextArea messageAreaOut;
   // private TextArea messageAreaIn;
    private SerialParameters parameters;
    private OutputStream os;
    private InputStream is;
   // private KeyHandler keyHandler;

    private CommPortIdentifier portId;
    private SerialPort sPort;

    private boolean open;
    private boolean dataReceived;
    private String data;
    /**
    Creates a SerialConnection object and initilizes variables
    passed in as params.

    @param parent A SerialDemo object.
    @param parameters A SerialParameters object.
    @param messageAreaOut The TextArea that messages that are to
    be sent out of the serial port are entered into.
    @param messageAreaIn The TextArea that messages comming into
    the serial port are displayed on.
    */
    public SerialConnection(GCSFrame parent,
            SerialParameters parameters) {
        this.parent = parent;
        this.parameters = parameters;
        /*this.messageAreaOut = messageAreaOut;
        this.messageAreaIn = messageAreaIn;*/
        open = false;
```

```
        dataReceived = false;
        data = "";
    }

    /**
    Attempts to open a serial connection and streams using the
    parameters in the SerialParameters object. If it is unsuccesful
    at any step it returns the port to a closed state, throws a
    <code>SerialConnectionException</code>, and returns.

    Gives a timeout of 30 seconds on the portOpen to allow other
    applications to reliquish the port if have it open and no longer
    need it.
    */
    public void openConnection() throws SerialConnectionException {

        // Obtain a CommPortIdentifier object for the port you want
        //to open.
        try {
            portId =
                 CommPortIdentifier.
                     getPortIdentifier(parameters.getPortName());
        } catch (NoSuchPortException e) {
            throw new SerialConnectionException(e.getMessage());
        }

        // Open the port represented by the CommPortIdentifier object.
        // Give the open call a relatively long timeout of 30 seconds
        // to allow a different application to reliquish the port if
        // the user wants to.
        try {
            sPort = (SerialPort)portId.open("SerialDemo", 30000);
        } catch (PortInUseException e) {
            throw new SerialConnectionException(e.getMessage());
        }

        // Set the parameters of the connection. If they won't set,
        // close the port before throwing an exception.
        try {
            setConnectionParameters();
        } catch (SerialConnectionException e) {
            sPort.close();
            throw e;
        }

        // Open the input and output streams for the connection.
        // If they won't open, close the port before throwing an
        // exception.
        try {
            os = sPort.getOutputStream();
            is = sPort.getInputStream();
        } catch (IOException e) {
            sPort.close();
            throw new SerialConnectionException(
                    "Error opening i/o streams");
        }

        // Create a new KeyHandler to respond to key strokes in the
        // messageAreaOut. Add the KeyHandler as a keyListener to the
        // messageAreaOut.
        //keyHandler = new KeyHandler(os);
        //messageAreaOut.addKeyListener(keyHandler);

        // Add this object as an event listener for the serial port.
        try {
            sPort.addEventListener(this);
        } catch (TooManyListenersException e) {
            sPort.close();
            throw new SerialConnectionException("" +
                    "too many listeners added");
        }

        // Set notifyOnDataAvailable to true to allow event driven
        // input.
        sPort.notifyOnDataAvailable(true);

        // Set notifyOnBreakInterrup to allow event driven break
        // handling.
        sPort.notifyOnBreakInterrupt(true);
```

```
        // Set receive timeout to allow breaking out of polling
        // loop during input handling.
        try {
            sPort.enableReceiveTimeout(30);
        } catch (UnsupportedCommOperationException e) {
        }

        // Add ownership listener to allow ownership event handling.
        portId.addPortOwnershipListener(this);

        open = true;
    }

    /**
    Sets the connection parameters to the setting in the parameters
    object. If set fails return the parameters object to origional
    settings and throw exception.
    */
    public void setConnectionParameters()
            throws SerialConnectionException {

        // Save state of parameters before trying a set.
        int oldBaudRate = sPort.getBaudRate();
        int oldDatabits = sPort.getDataBits();
        int oldStopbits = sPort.getStopBits();
        int oldParity   = sPort.getParity();
        int oldFlowControl = sPort.getFlowControlMode();

        // Set connection parameters, if set fails return parameters
        // object to original state.
        try {
            sPort.setSerialPortParams(parameters.getBaudRate(),
                                      parameters.getDatabits(),
                                      parameters.getStopbits(),
                                      parameters.getParity());
        } catch (UnsupportedCommOperationException e) {
            parameters.setBaudRate(oldBaudRate);
            parameters.setDatabits(oldDatabits);
            parameters.setStopbits(oldStopbits);
            parameters.setParity(oldParity);
            throw new SerialConnectionException(
                    "Unsupported parameter");
        }

        // Set flow control.
        try {
            sPort.setFlowControlMode(parameters.getFlowControlIn()
                                     | parameters.getFlowControlOut());
        } catch (UnsupportedCommOperationException e) {
            throw new SerialConnectionException(
                    "Unsupported flow control");
        }
    }

    /**
    Close the port and clean up associated elements.
    */
    public void closeConnection() {
        // If port is alread closed just return.
        if (!open) {
            return;
        }

        // Remove the key listener.
        //messageAreaOut.removeKeyListener(keyHandler);

        // Check to make sure sPort has reference to avoid a NPE.
        if (sPort != null) {
            try {
                // close the i/o streams.
                os.close();
                is.close();
            } catch (IOException e) {
                System.err.println(e);
            }

            // Close the port.
            sPort.close();

            // Remove the ownership listener.
```

```
            portId.removePortOwnershipListener(this);
        }

        open = false;
    }

    /**
    Send a one second break signal.
    */
    public void sendBreak() {
        sPort.sendBreak(1000);
    }

    /**
    Reports the open status of the port.
    @return true if port is open, false if port is closed.
    */
    public boolean isOpen() {
        return open;
    }

    /**
    Handles SerialPortEvents. The two types of SerialPortEvents
    that this program is registered to listen for are DATA_AVAILABLE
    and BI. During DATA_AVAILABLE the port buffer is read until it is
    drained, when no more data is availble and 30ms has passed the
method
    returns. When a BI event occurs the words BREAK RECEIVED are
written
    to the messageAreaIn.
    */

    public void serialEvent(SerialPortEvent e) {
        // Create a StringBuffer and int to receive input data.
        StringBuffer inputBuffer = new StringBuffer();
        int newData = 0;

        // Determine type of event.
        switch (e.getEventType()) {

            // Read data until -1 is returned. If \r is received
            // substitute \n for correct newline handling.
            case SerialPortEvent.DATA_AVAILABLE:

                    while (newData != -1) {
                        try {
                            newData = is.read();
                            if (newData == -1) {
                                break;
                            }
                            if ('\r' == (char)newData) {
                                inputBuffer.append('\n');
                            } else {
                                inputBuffer.append((char)newData);
                            }
                        } catch (IOException ex) {
                            System.err.println(ex);
                            return;
                        }
                    }
                    String s = new String(inputBuffer);
                    char c = s.charAt(0);
                    if (s.compareTo("!") == 0 ) {
                        parent.allSystemsGo();
                    }
                    else if (c == '*') {
                        parent.waypointReached();
                        parent.parseGPSData(s);
                    }
                    else
                        parent.parseGPSData(s);

                // Append received data to messageAreaIn.
                //messageAreaIn.append(new String(inputBuffer));

                break;

            // If break event append BREAK RECEIVED message.
            case SerialPortEvent.BI:
                //messageAreaIn.append("\n--- BREAK RECEIVED ---\n");
```

```
            }

        }

        /**
        Handles ownership events. If a PORT_OWNERSHIP_REQUESTED event is
        received a dialog box is created asking the user if they are
        willing to give up the port. No action is taken on other types
        of ownership events.
        */
        public void ownershipChange(int type) {
        }

        public void sendData(String s) {
                //char newCharacter = evt.getKeyChar();
                for (int i = 0; i<s.length();i++){
                    try {
                        os.write((int)s.charAt(i));
                    } catch (IOException e) {
                        System.err.println(
                                "OutputStream write error: " + e);
                    }
                }
        }

        public boolean hasDataBeenReceived() {
            return dataReceived;
        }

        public void setDataReceived(boolean x){
            dataReceived = x;
        }

        public String getData() {
            return data;
        }

    }
```

```
/* @(#)SerialParameters.java    1.5 98/07/17 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license
 * appear on all copies of the software; and ii) Licensee does not
 * utilize the software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN
 * AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS
 * BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line
 * control of aircraft, air traffic, aircraft navigation or
 * aircraft communications; or in the design, construction,
 * operation or maintenance of any nuclear facility. Licensee
 * represents and warrants that it will not use or redistribute
 * the Software for such purposes.
 */

package groundcontrolstation;
import javax.comm.*;

/**
A class that stores parameters for serial ports.
*/
public class SerialParameters {

    private String portName;
    private int baudRate;
    private int flowControlIn;
    private int flowControlOut;
    private int databits;
    private int stopbits;
    private int parity;

    /**
    Default constructer. Sets parameters to no port, 9600 baud,
    no flow control, 8 data bits, 1 stop bit, no parity.
    */
    public SerialParameters () {
        this("COM1",
             9600,
             SerialPort.FLOWCONTROL_RTSCTS_IN,
             SerialPort.FLOWCONTROL_NONE,
             SerialPort.DATABITS_8,
             SerialPort.STOPBITS_1,
             SerialPort.PARITY_NONE );

    }

    /**
    Paramaterized constructer.

    @param portName The name of the port.
    @param baudRate The baud rate.
    @param flowControlIn Type of flow control for receiving.
    @param flowControlOut Type of flow control for sending.
    @param databits The number of data bits.
    @param stopbits The number of stop bits.
    @param parity The type of parity.
    */
    public SerialParameters(String portName,
                            int baudRate,
                            int flowControlIn,
                            int flowControlOut,
                            int databits,
                            int stopbits,
                            int parity) {
```

```
        this.portName = portName;
        this.baudRate = baudRate;
        this.flowControlIn = flowControlIn;
        this.flowControlOut = flowControlOut;
        this.databits = databits;
        this.stopbits = stopbits;
        this.parity = parity;
    }

    /**
    Sets port name.
    @param portName New port name.
    */
    public void setPortName(String portName) {
        this.portName = portName;
    }

    /**
    Gets port name.
    @return Current port name.
    */
    public String getPortName() {
        return portName;
    }

    /**
    Sets baud rate.
    @param baudRate New baud rate.
    */
    public void setBaudRate(int baudRate) {
        this.baudRate = baudRate;
    }

    /**
    Sets baud rate.
    @param baudRate New baud rate.
    */
    public void setBaudRate(String baudRate) {
        this.baudRate = Integer.parseInt(baudRate);
    }

    /**
    Gets baud rate as an <code>int</code>.
    @return Current baud rate.
    */
    public int getBaudRate() {
        return baudRate;
    }

    /**
    Gets baud rate as a <code>String</code>.
    @return Current baud rate.
    */
    public String getBaudRateString() {
        return Integer.toString(baudRate);
    }

    /**
    Sets flow control for reading.
    @param flowControlIn New flow control for reading type.
    */
    public void setFlowControlIn(int flowControlIn) {
        this.flowControlIn = flowControlIn;
    }

    /**
    Sets flow control for reading.
    @param flowControlIn New flow control for reading type.
    */
    public void setFlowControlIn(String flowControlIn) {
        this.flowControlIn = stringToFlow(flowControlIn);
    }

    /**
    Gets flow control for reading as an <code>int</code>.
    @return Current flow control type.
    */
    public int getFlowControlIn() {
        return flowControlIn;
```

```
}

/**
Gets flow control for reading as a <code>String</code>.
@return Current flow control type.
*/
public String getFlowControlInString() {
    return flowToString(flowControlIn);
}

/**
Sets flow control for writing.
@param flowControlIn New flow control for writing type.
*/
public void setFlowControlOut(int flowControlOut) {
    this.flowControlOut = flowControlOut;
}

/**
Sets flow control for writing.
@param flowControlIn New flow control for writing type.
*/
public void setFlowControlOut(String flowControlOut) {
    this.flowControlOut = stringToFlow(flowControlOut);
}

/**
Gets flow control for writing as an <code>int</code>.
@return Current flow control type.
*/
public int getFlowControlOut() {
    return flowControlOut;
}

/**
Gets flow control for writing as a <code>String</code>.
@return Current flow control type.
*/
public String getFlowControlOutString() {
    return flowToString(flowControlOut);
}

/**
Sets data bits.
@param databits New data bits setting.
*/
public void setDatabits(int databits) {
    this.databits = databits;
}

/**
Sets data bits.
@param databits New data bits setting.
*/
public void setDatabits(String databits) {
    if (databits.equals("5")) {
        this.databits = SerialPort.DATABITS_5;
    }
    if (databits.equals("6")) {
        this.databits = SerialPort.DATABITS_6;
    }
    if (databits.equals("7")) {
        this.databits = SerialPort.DATABITS_7;
    }
    if (databits.equals("8")) {
        this.databits = SerialPort.DATABITS_8;
    }
}

/**
Gets data bits as an <code>int</code>.
@return Current data bits setting.
*/
public int getDatabits() {
    return databits;
}

/**
Gets data bits as a <code>String</code>.
@return Current data bits setting.
```

```
*/
public String getDatabitsString() {
    switch(databits) {
        case SerialPort.DATABITS_5:
            return "5";
        case SerialPort.DATABITS_6:
            return "6";
        case SerialPort.DATABITS_7:
            return "7";
        case SerialPort.DATABITS_8:
            return "8";
        default:
            return "8";
    }
}

/**
Sets stop bits.
@param stopbits New stop bits setting.
*/
public void setStopbits(int stopbits) {
    this.stopbits = stopbits;
}

/**
Sets stop bits.
@param stopbits New stop bits setting.
*/
public void setStopbits(String stopbits) {
    if (stopbits.equals("1")) {
        this.stopbits = SerialPort.STOPBITS_1;
    }
    if (stopbits.equals("1.5")) {
        this.stopbits = SerialPort.STOPBITS_1_5;
    }
    if (stopbits.equals("2")) {
        this.stopbits = SerialPort.STOPBITS_2;
    }
}

/**
Gets stop bits setting as an <code>int</code>.
@return Current stop bits setting.
*/
public int getStopbits() {
    return stopbits;
}

/**
Gets stop bits setting as a <code>String</code>.
@return Current stop bits setting.
*/
public String getStopbitsString() {
    switch(stopbits) {
        case SerialPort.STOPBITS_1:
            return "1";
        case SerialPort.STOPBITS_1_5:
            return "1.5";
        case SerialPort.STOPBITS_2:
            return "2";
        default:
            return "1";
    }
}

/**
Sets parity setting.
@param parity New parity setting.
*/
public void setParity(int parity) {
    this.parity = parity;
}

/**
Sets parity setting.
@param parity New parity setting.
*/
public void setParity(String parity) {
    if (parity.equals("None")) {
        this.parity = SerialPort.PARITY_NONE;
```

```
        }
        if (parity.equals("Even")) {
            this.parity = SerialPort.PARITY_EVEN;
        }
        if (parity.equals("Odd")) {
            this.parity = SerialPort.PARITY_ODD;
        }
    }

    /**
    Gets parity setting as an <code>int</code>.
    @return Current parity setting.
    */
    public int getParity() {
        return parity;
    }

    /**
    Gets parity setting as a <code>String</code>.
    @return Current parity setting.
    */
    public String getParityString() {
        switch(parity) {
            case SerialPort.PARITY_NONE:
                return "None";
            case SerialPort.PARITY_EVEN:
                return "Even";
            case SerialPort.PARITY_ODD:
                return "Odd";
            default:
                return "None";
        }
    }

    /**
    Converts a <code>String</code> describing a flow control type
    to an <code>int</code> type defined in <code>SerialPort</code>.
    @param flowControl A <code>string</code> describing a flow
    control type.
    @return An <code>int</code> describing a flow control type.
    */
    private int stringToFlow(String flowControl) {
        if (flowControl.equals("None")) {
            return SerialPort.FLOWCONTROL_NONE;
        }
        if (flowControl.equals("Xon/Xoff Out")) {
            return SerialPort.FLOWCONTROL_XONXOFF_OUT;
        }
        if (flowControl.equals("Xon/Xoff In")) {
            return SerialPort.FLOWCONTROL_XONXOFF_IN;
        }
        if (flowControl.equals("RTS/CTS In")) {
            return SerialPort.FLOWCONTROL_RTSCTS_IN;
        }
        if (flowControl.equals("RTS/CTS Out")) {
            return SerialPort.FLOWCONTROL_RTSCTS_OUT;
        }
        return SerialPort.FLOWCONTROL_NONE;
    }

    /**
    Converts an <code>int</code> describing a flow control type
    to a <code>String</code> describing a flow control type.
    @param flowControl An <code>int</code> describing a flow
    control type.
    @return A <code>String</code> describing a flow control type.
    */
    String flowToString(int flowControl) {
        switch(flowControl) {
            case SerialPort.FLOWCONTROL_NONE:
                return "None";
            case SerialPort.FLOWCONTROL_XONXOFF_OUT:
                return "Xon/Xoff Out";
            case SerialPort.FLOWCONTROL_XONXOFF_IN:
                return "Xon/Xoff In";
            case SerialPort.FLOWCONTROL_RTSCTS_IN:
                return "RTS/CTS In";
            case SerialPort.FLOWCONTROL_RTSCTS_OUT:
                return "RTS/CTS Out";
            default:
```

```
                return "None";
            }
        }
    }
```

```
package groundcontrolstation;
/* @(#)SerialConnectionException.java   1.3 98/06/04 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license
 * appear on all copies of the software; and ii) Licensee does not
 * utilize the software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN
 * AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS
 * BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line
 * control of aircraft, air traffic, aircraft navigation or
 * aircraft communications; or in the design, construction,
 * operation or maintenance of any nuclear facility. Licensee
 * represents and warrants that it will not use or redistribute
 * the Software for such purposes.
 */

public class SerialConnectionException extends Exception {

    /**
     * Constructs a <code>SerialConnectionException</code>
     * with the specified detail message.
     *
     * @param   s   the detail message.
     */
    public SerialConnectionException(String str) {
        super(str);
    }

    /**
     * Constructs a <code>SerialConnectionException</code>
     * with no detail message.
     */
    public SerialConnectionException() {
        super();
    }
}
```

```
/*
 * UAVStatusPanel.java
 *
 * This panel displays some of the raw telemetry data, as well
 * as a status of the connection to the UAV
 *
 * Created on 26 May 2005, 15:36
 */

package groundcontrolstation;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.awt.*;
/**
 *
 * @author Soz
 */
public class UAVStatusPanel extends JPanel{
    private GCSFrame parent;

    private JLabel label1 = new JLabel( "GPS Fix Indication");
    private JLabel label2 = new JLabel( "Datalink");
    private JLabel label3 = new JLabel( "GPS UTC");
    private JLabel label4 = new JLabel( "Battery Voltage");
    private JLabel status1 = new JLabel( "NOT CONNECTED");
    private JLabel status2 = new JLabel( "NOT CONNECTED");
    private JTextField status3 = new JTextField( "Unknown");
    private JTextField status4 = new JTextField( "5V" );
    private JTextArea details = new JTextArea("", 10, 20);
    private JScrollPane scroller = new JScrollPane(details);

    private JPanel pane3 = new JPanel();
    private JPanel pane4 = new JPanel();
    private Icon grayIcon = new ImageIcon(
            "Images/little-led-gray.gif");
    private Icon greenIcon = new ImageIcon(
            "Images/little-led-green.gif");
    private Icon redIcon = new ImageIcon(
            "Images/little-led-red.gif");

    /** Creates a new instance of UAVStatusPanel */
    public UAVStatusPanel( GCSFrame p ) {
        parent = p;
        initComponents();
    }

    public void initComponents() {
        status1.setIcon(grayIcon);
        status2.setIcon(grayIcon);

        pane3.setLayout(new GridLayout(4, 2));
        pane3.add(label1);
        pane3.add(status1);
        pane3.add(label2);
        pane3.add(status2);
        pane3.add(label3);
        pane3.add(status3);
        pane3.add(label4);
        pane3.add(status4);

        pane4.setLayout( new BorderLayout() );
        pane4.add( scroller, BorderLayout.CENTER );
        pane4.setPreferredSize( new Dimension(300, 400 ));
        pane4.setBorder( new LineBorder(Color.BLACK));

        GridBagLayout g = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;
        c.gridx = 0;
        c.gridy = 0;
        c.weightx = 0;
        c.weighty = 0;
        c.gridheight = GridBagConstraints.RELATIVE;
        c.gridwidth = GridBagConstraints.REMAINDER;
        c.insets = new Insets(5,5,5,5);
        setLayout( g );
        g.setConstraints(pane3, c);
        add(pane3);
```

```
        pane4.setBorder( new TitledBorder(
                new LineBorder(Color.BLACK), "Raw Telemetry Data"));
        c.fill = GridBagConstraints.BOTH;
        c.gridy = 1;
        c.weightx = 1;
        c.weighty = 1;
        c.gridheight = GridBagConstraints.REMAINDER;
        c.gridwidth = GridBagConstraints.REMAINDER;
        g.setConstraints(pane4, c);
        add(pane4);

        setBorder( new TitledBorder(
                new LineBorder(Color.BLACK), "UAV Status"));

    }

    public void updateTelemetry( String data ){
        details.append( "New Data...\n" + data );
    }

    public void GPSConnGood() {
        status1.setIcon(greenIcon);
        status2.setIcon(greenIcon);
        status1.setText("CONNECTED");
        status2.setText("CONNECTED");
    }

    public void updateStatus( String s ) {
        status3.setText(s);
    }
}
```

```
/*
 * TelemetryPane.java
 *
 * This panel contains the displays which allow the user on the
 * ground to view telemetry data from the UAV
 *
 * Created on 5 August 2005, 14:33
 */

package groundcontrolstation;

import javax.swing.*;
import javax.swing.event.*;
import com.elegantj.gauges.needle.*;
import com.elegantj.gauges.dial.*;
import java.awt.*;
import javax.swing.border.*;
import java.util.*;


/**
 *
 * @author Soz
 */
public class TelemetryPane extends JPanel {
    private JPanel velPane = new JPanel();
    private JPanel altPane = new JPanel();
    private JPanel compassPane = new JPanel();
    private JPanel GPSPane = new JPanel();

    private NumericDialGauge desiredBearing;
    private JLabel desiredBearingValue;

    private NumericDialGauge compass;
    private JLabel compassValue;
    private JLabel gpsLatLabel = new JLabel("Current GPS Lat: ");
    private JLabel gpsLonLabel = new JLabel("Current GPS Lon: ");
    private JLabel distLabel = new JLabel(
            "Distance to Waypoint(m): ");

    private JTextField gpslatfield = new JTextField(10);
    private JTextField gpslonfield = new JTextField(10);
    private JTextField distField = new JTextField(10);


    private SevenSegment s1;
    private SevenSegment s2;
    private SevenSegment s3;

    /** Creates a new instance of TelemetryPane */
    public TelemetryPane() {
        initComponents();
        GridBagConstraints c = new GridBagConstraints();
        GridBagLayout g = new GridBagLayout();
        velPane.setLayout(g);
        c.fill = GridBagConstraints.BOTH;
        c.gridheight = GridBagConstraints.REMAINDER;
        c.gridwidth = GridBagConstraints.REMAINDER;
        c.weighty = 1;
        c.weightx = 1;
        g.setConstraints(desiredBearing,c);
        velPane.add(desiredBearing);
        velPane.setBorder( new TitledBorder(
                new LineBorder(Color.BLACK), "Desired Heading Data"));
        Vector zoneValues = new Vector();
        Vector v1 = new Vector();

        v1.addElement(new com.elegantj.gauges.MeterZone(
                "Safe Zone",0.0,120.0,Color.blue,30,2,15));
        v1.addElement(new Double(0));
        v1.addElement(new Double(100));
        zoneValues.addElement(v1);

        Vector v2 = new Vector();

        v2.addElement(new com.elegantj.gauges.MeterZone(
                "Warning Zone",120.0,240.0,Color.blue,30,2,15));
        v2.addElement(new Double(0));
        v2.addElement(new Double(100));
        zoneValues.addElement(v2);
```

```
        Vector v3 = new Vector();
        v3.addElement(new com.elegantj.gauges.MeterZone(
                "Alarm Zone",240.0,360.0,Color.blue,30,2,15));
        v3.addElement(new Double(0));
        v3.addElement(new Double(100));
        zoneValues.addElement(v3);

        desiredBearing.setNewZoneValues(zoneValues);

        initAltPane();
        initCompassPane();

        setLayout( new GridLayout(1, 3));
        add(velPane);
        add(compassPane);
        add(altPane);

    }

    public void initComponents() {
        desiredBearingValue = new JLabel();
        String caption = "UAV Desired Heading";
        int size = 240; // dial's diameter, default is 320
        boolean clockWise = false;
        double min = 0; // dial's minimum value, default is 0
        double max = 360; // dial's maximum value, default is 100
        double minAngle = 0; // dial's minimum angle, default is 0
        double maxAngle = 360; // dial's minimum angle, default is 360
        desiredBearing = new NumericDialGauge(caption, size, min, max,
                minAngle, maxAngle, clockWise);
        desiredBearing.setValue(50);
        desiredBearing.setUnit("Deg");
        desiredBearing.setDisplayBoxVisible(false);
        desiredBearing.setHeaderText("");
        desiredBearing.setFooterText("Desired Bearing: "+
                Double.toString(desiredBearing.getValue())+
                " degrees");
        desiredBearing.setNeedleType(
                com.elegantj.gauges.dial.
                NumericDialGauge.ARROW_NEEDLE);

        compassValue = new JLabel();
        String caption2 = "UAV Bearing";
        max = 360; // dial's maximum value, defalut is 100
        compass = new NumericDialGauge(caption2, size, min, max,
                minAngle, maxAngle, clockWise);
        compass.setValue(50);
        compass.setUnit("Deg");
        compass.setHeaderText("");
        compass.setDisplayBoxVisible(false);
        compass.setFooterText("Current Bearing: "+Double.toString(
                compass.getValue())+" degrees");
        compass.setNeedleType(
                com.elegantj.gauges.dial.NumericDialGauge.
                ARROW_NEEDLE);
    }

    public void initAltPane() {
        GridBagConstraints c1 = new GridBagConstraints();
        GridBagLayout g1 = new GridBagLayout();
        GPSPane.setLayout(g1);
        c1.fill = GridBagConstraints.NONE;
        c1.anchor = GridBagConstraints.WEST;
        c1.gridx = 0;
        c1.gridy = 0;
        c1.weighty = 0.5;
        c1.weightx = 0;
        g1.setConstraints(gpsLatLabel,c1);
        GPSPane.add(gpsLatLabel);

        c1.fill = GridBagConstraints.HORIZONTAL;
        c1.gridx = 1;
        c1.gridy = 0;
        c1.weightx = 1;
        c1.weighty = 0.5;
        g1.setConstraints(gpslatfield,c1);
        GPSPane.add(gpslatfield);

        c1.fill = GridBagConstraints.NONE;
```

```
        c1.gridx = 0;
        c1.gridy = 1;
        c1.weighty = 0.5;
        c1.weightx = 0;
        g1.setConstraints(gpsLonLabel,c1);
        GPSPane.add(gpsLonLabel);

        c1.fill = GridBagConstraints.HORIZONTAL;
        c1.gridx = 1;
        c1.gridy = 1;
        c1.weightx = 1;
        c1.weighty = 0.5;
        g1.setConstraints(gpslonfield,c1);
        GPSPane.add(gpslonfield);

        c1.fill = GridBagConstraints.NONE;
        c1.gridx = 0;
        c1.gridy = 2;
        c1.weighty = 0.5;
        c1.weightx = 0;
        g1.setConstraints(distLabel,c1);
        GPSPane.add(distLabel);

        c1.fill = GridBagConstraints.HORIZONTAL;
        c1.gridx = 1;
        c1.gridy = 2;
        c1.weightx = 1;
        c1.weighty = 0.5;
        g1.setConstraints(distField,c1);
        GPSPane.add(distField);

        JPanel p = new JPanel();

        s1 = new SevenSegment(0);
        s2 = new SevenSegment(0);
        s3 = new SevenSegment(0);
        s1.setPreferredSize(new Dimension(40, 70));
        s2.setPreferredSize(new Dimension(40, 70));
        s3.setPreferredSize(new Dimension(40, 70));

        JLabel altLabel = new JLabel("UAV Altitude: ");

        GridBagConstraints c = new GridBagConstraints();
        GridBagLayout g = new GridBagLayout();
        p.setLayout(g);
        c.fill = GridBagConstraints.HORIZONTAL;
        c.gridheight = GridBagConstraints.RELATIVE;
        c.gridx = 0;
        c.weighty = 0;
        c.weightx = 1;
        g.setConstraints(altLabel,c);
        p.add(altLabel);

        c.fill = GridBagConstraints.NONE;
        c.gridheight = GridBagConstraints.RELATIVE;
        c.gridx = 1;
        c.weightx = 0;
        c.weighty = 0.5;
        g.setConstraints(s1,c);
        p.add(s1);

        c.gridheight = GridBagConstraints.REMAINDER;
        c.gridx = 2;
        g.setConstraints(s2,c);
        p.add(s2);

        c.gridheight = GridBagConstraints.REMAINDER;
        c.gridx = 3;
        g.setConstraints(s3,c);
        p.add(s3);

        p.setBorder( new TitledBorder( new LineBorder(Color.BLACK),
                "Altitude Data"));
        GPSPane.setBorder( new TitledBorder(
                new LineBorder(Color.BLACK), "GPS Data"));

        altPane.setLayout(new GridLayout(2,1));
        altPane.add(p);
        altPane.add(GPSPane);
    }
```

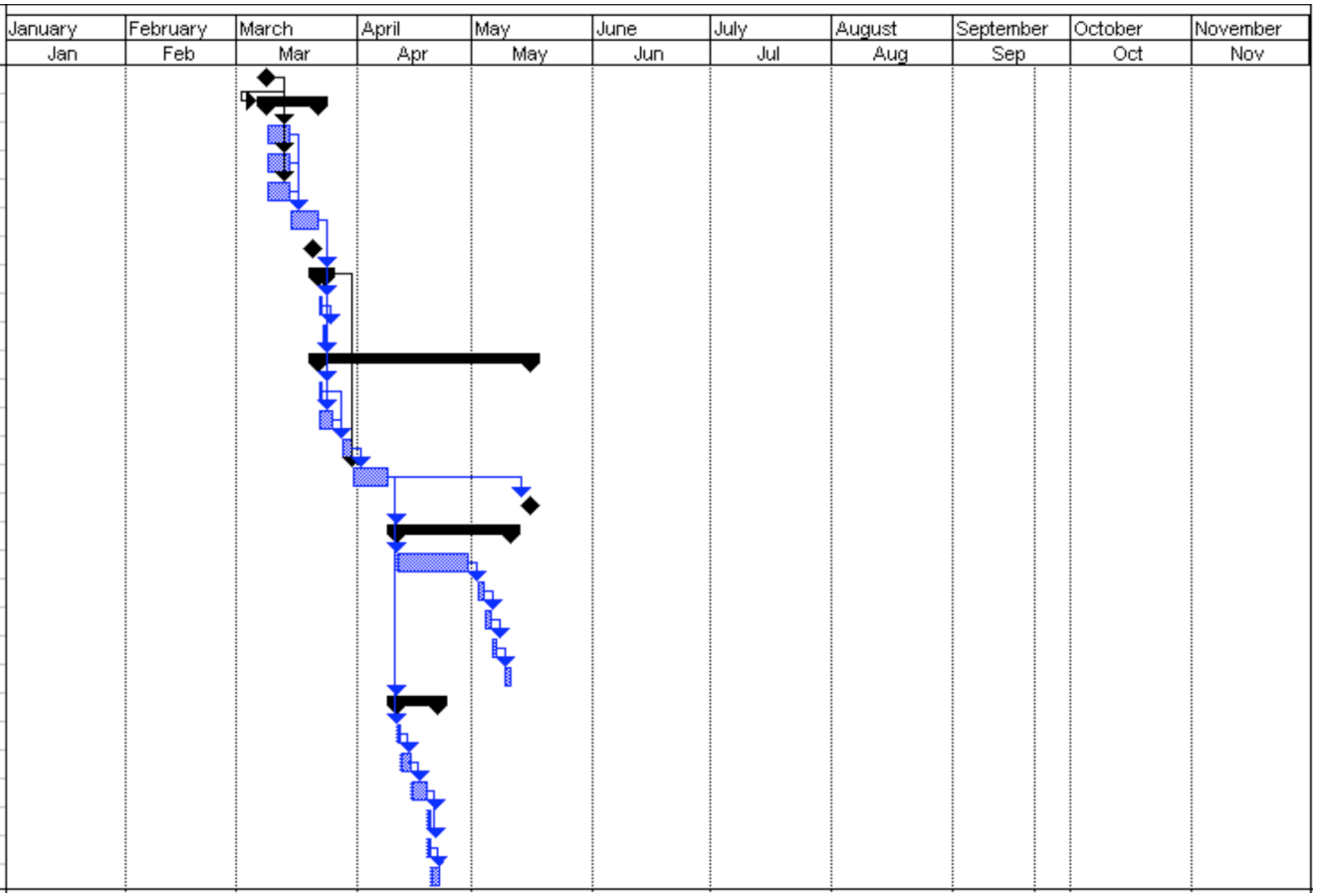
```
    public void initCompassPane() {
        GridBagConstraints c = new GridBagConstraints();
        GridBagLayout g = new GridBagLayout();
        compassPane.setLayout(g);
        c.fill = GridBagConstraints.BOTH;
        c.gridheight = GridBagConstraints.REMAINDER;
        c.gridwidth = GridBagConstraints.REMAINDER;
        c.weighty = 1;
        c.weightx = 1;
        g.setConstraints(compass,c);
        compassPane.add(compass);
        compassPane.setBorder( new TitledBorder(
                new LineBorder(Color.BLACK), "Bearing Data"));
        Vector zoneValues = new Vector();
        Vector v1 = new Vector();

        v1.addElement(new com.elegantj.gauges.MeterZone(
                "Safe Zone",0.0,120.0,Color.red,30,2,15));
        v1.addElement(new Double(0));
        v1.addElement(new Double(100));
        zoneValues.addElement(v1);

        Vector v2 = new Vector();

        v2.addElement(new com.elegantj.gauges.MeterZone(
                "Warning Zone",120.0,240.0,Color.red,30,2,15));
        v2.addElement(new Double(0));
        v2.addElement(new Double(100));
        zoneValues.addElement(v2);

        Vector v3 = new Vector();
        v3.addElement(new com.elegantj.gauges.MeterZone(
                "Alarm Zone",240.0,360.0,Color.red,30,2,15));
        v3.addElement(new Double(0));
        v3.addElement(new Double(100));
        zoneValues.addElement(v3);

        compass.setNewZoneValues(zoneValues);
    }

    public void updateHeading( double newHeading ){
        compass.setValue(newHeading);
        compass.setFooterText("Current Bearing: "+
                Double.toString(compass.getValue())+
                " degrees");
        compass.resetDial();
    }

    public void updateDesiredHeading( double newHeading ){
        desiredBearing.setValue(newHeading);
        desiredBearing.setFooterText("Desired Bearing: "+
                Double.toString(desiredBearing.getValue())
                +" degrees");
    }

    public void updateHeight(int h1, int h2, int h3) {
        s1.updateDisplay(h1);
        s2.updateDisplay(h2);
        s3.updateDisplay(h3);
    }

    public void updateLat( String s ) {
        gpslatfield.setText(s);
    }

    public void updateLon( String s ) {
        gpslonfield.setText(s);
    }

    public void updateDist( String s ) {
        distField.setText(s);
    }
}
```

```
/*
 * SevenSegment.java
 * This class is used to provide an seven segment style display
 * which displays the altitude of the UAV
 * Created on 10 August 2005, 14:33
 */

package groundcontrolstation;

import javax.swing.*;
import java.awt.*;

public class SevenSegment extends JPanel {

        Color fg1 = Color.black;
        Color fg2 = Color.green;
        int number;

        public SevenSegment( int x ) {
            number = x;
        }

        public void paint(Graphics g) {
            Graphics2D g2 = (Graphics2D)g;
            g2.setPaint(fg1);
            g2.fill(new Rectangle(45, 70));

            switch(number) {
                case 0:
                    g2.setPaint(fg2);
                    //first front down stroke
                    g2.fill(new Rectangle(5, 5, 5, 30));
                    //second front down stroke
                    g2.fill(new Rectangle(5, 35, 5, 25));
                    //first back down stroke
                    g2.fill(new Rectangle(35, 5, 5, 30));
                    //second back down stroke
                    g2.fill(new Rectangle(35, 35, 5, 30));
                    //top cross stroke
                    g2.fill(new Rectangle(5, 5, 30, 5));
                    //bottom cross stroke
                    g2.fill(new Rectangle(5, 60, 30, 5));

                    break;

                case 1:
                    g2.setPaint(fg2);
                    g2.fill(new Rectangle(35, 5, 5, 30));
                    g2.fill(new Rectangle(35, 35, 5, 30));
                    break;

                case 2:
                    g2.setPaint(fg2);
                    g2.fill(new Rectangle(5, 5, 30, 5));
                    g2.fill(new Rectangle(5, 60, 35, 5));
                    g2.fill(new Rectangle(35, 5, 5, 30));
                    g2.fill(new Rectangle(5, 35, 5, 25));
                    //middle cross stroke
                    g2.fill(new Rectangle(5, 35, 35, 5));
                    break;

                case 3:
                    g2.setPaint(fg2);
                    g2.fill(new Rectangle(35, 5, 5, 30));
                    g2.fill(new Rectangle(35, 35, 5, 30));
                    //top cross stroke
                    g2.fill(new Rectangle(5, 5, 30, 5));
                    //bottom cross stroke
                    g2.fill(new Rectangle(5, 60, 35, 5));
                    g2.fill(new Rectangle(5, 35, 30, 5));
                    break;

                case 4:
                    g2.setPaint(fg2);
                    g2.fill(new Rectangle(35, 5, 5, 30));
                    g2.fill(new Rectangle(35, 35, 5, 30));
                    g2.fill(new Rectangle(5, 35, 30, 5));
                    g2.fill(new Rectangle(5, 5, 5, 30));
                    break;
```

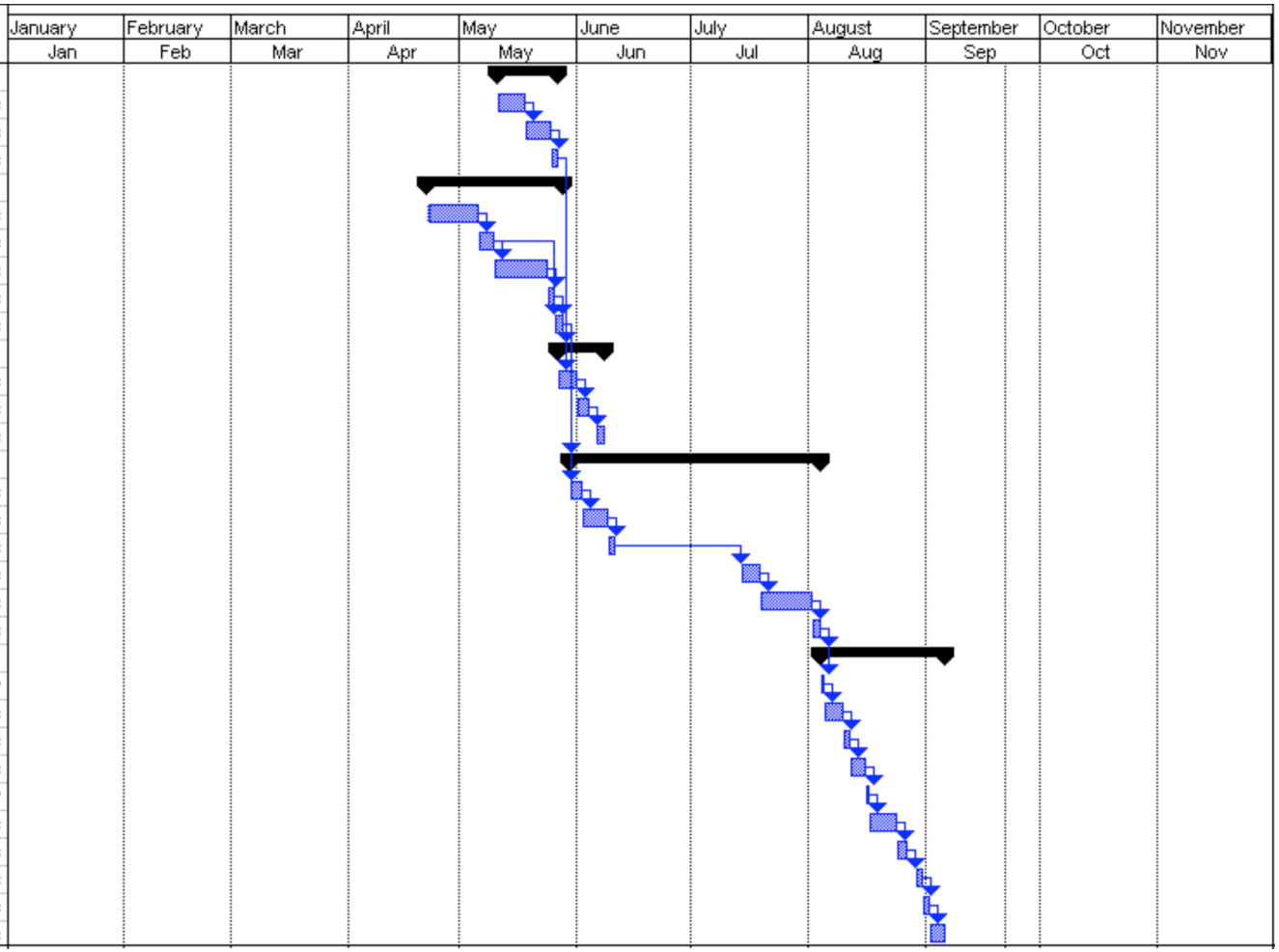
```
                case 5:
                    g2.setPaint(fg2);
                    //top cross stroke
                    g2.fill(new Rectangle(5, 5, 35, 5));
                    //bottom cross stroke
                    g2.fill(new Rectangle(5, 60, 35, 5));
                    g2.fill(new Rectangle(5, 35, 35, 5));
                    //first front down stroke
                    g2.fill(new Rectangle(5, 5, 5, 30));
                    //second back down stroke
                    g2.fill(new Rectangle(35, 35, 5, 30));
                    break;

                case 6:
                    g2.setPaint(fg2);
                    //top cross stroke
                    g2.fill(new Rectangle(5, 5, 35, 5));
                    //bottom cross stroke
                    g2.fill(new Rectangle(5, 60, 35, 5));
                    g2.fill(new Rectangle(5, 35, 35, 5));
                    //first front down stroke
                    g2.fill(new Rectangle(5, 5, 5, 30));
                    //second back down stroke
                    g2.fill(new Rectangle(35, 35, 5, 30));
                    //second front down stroke
                    g2.fill(new Rectangle(5, 35, 5, 30));
                    break;

                case 7:
                    g2.setPaint(fg2);
                    g2.fill(new Rectangle(35, 5, 5, 30));
                    g2.fill(new Rectangle(35, 35, 5, 30));
                    //top cross stroke
                    g2.fill(new Rectangle(5, 5, 30, 5));
                    break;

                case 8:
                    g2.setPaint(fg2);
                    //first front down stroke
                    g2.fill(new Rectangle(5, 5, 5, 30));
                    //second front down stroke
                    g2.fill(new Rectangle(5, 35, 5, 30));
                    //first back down stroke
                    g2.fill(new Rectangle(35, 5, 5, 30));
                    //second back down stroke
                    g2.fill(new Rectangle(35, 35, 5, 30));
                    //top cross stroke
                    g2.fill(new Rectangle(5, 5, 30, 5));
                    //bottom cross stroke
                    g2.fill(new Rectangle(5, 60, 35, 5));
                    g2.fill(new Rectangle(5, 35, 30, 5));
                    break;

                case 9:
                    g2.setPaint(fg2);
                    //first front down stroke
                    g2.fill(new Rectangle(5, 5, 5, 30));
                    //first back down stroke
                    g2.fill(new Rectangle(35, 5, 5, 30));
                    //second back down stroke
                    g2.fill(new Rectangle(35, 35, 5, 30));
                    //top cross stroke
                    g2.fill(new Rectangle(5, 5, 30, 5));
                    g2.fill(new Rectangle(5, 35, 30, 5));
                    break;

                case 10:
                    break;
                }
        }

        public void updateDisplay( int x ) {
                number = x;
                this.repaint();
        }
}
```

Appendix I- Gantt Chart

ID	Task Name	Duration
1	Topic Allocation	0 days
2	**System Requirements**	**9 days**
3	Research UAV navigation	4 days
4	Research UAV communications link	4 days
5	Reaearch UAV User Interfaces	4 days
6	System Design	5 days
7	Project Specification Due	0 days
8	**Hardware Requirements**	**2 days**
9	Analyse hardware requirements	1 day
10	Acquire/ purchase Hardware	1 day
11	**Software Requirements**	**39 days**
12	Determine GCS software requirement	1 day
13	Determine Navigational software requ	4 days
14	Determine software interface requirei	3 days
15	Write Project Appreciation	7 days
16	Project Appreciation Due	0 days
17	**Hardware Design**	**22 days**
18	Prototype interface (sensors)	15 days
19	Prototype interface (camera unit)	2 days
20	Prototype interface transmitter and re	2 days
21	Develop interface circuit diagram	1 day
22	Develop hardware test plan	2 days
23	**Software Design**	**9 days**
24	Develop use cases for GCS	1 day
25	Prototype GUI	3 days
26	Develop class structures for GCS	2 days
27	Construct logical scenario diagrams-	1 day
28	Construct logical scenario diagrams-	1 day
29	Develop software test plan	2 days



ID	Task Name	Duration
30	**Build Hardware**	**12 days**
31	Construct navigation HC12 module	5 days
32	Construct GPS and compass modules	5 days
33	Construct the Camera and Transmitter	2 days
34	**Build Software**	**26 days**
35	Code GCS	10 days
36	Develop unit tests for GCS	2 days
37	Code Navigation software	10 days
38	Develop unit tests for Navigation	2 days
39	Execute unit tests	2 days
40	**Test Hardware**	**8 days**
41	Execute hardware tests	3 days
42	Rework hardware design	3 days
43	Retest	2 days
44	**Test Software**	**25 days**
45	Execute software tests - GCS	3 days
46	Rework software design - GCS	5 days
47	Retest GCS	2 days
48	Execute software tests - navigation	3 days
49	Rework software desgin - navigation	10 days
50	Retest navigation software	2 days
51	**Integration and Test**	**23 days**
52	Load tested software onto hardware	1 day
53	Develop integration test procedures	3 days
54	Execute test procedures	2 days
55	Rework integration if required	2 days
56	Retest integration	1 day
57	Integrate with external control system	5 days
58	Develop system test procedures	3 days
59	Execute test procedures	2 days
60	Rework UAV integration if required	2 days
61	Retest UAV integration	2 days



ID	Task Name	Duration
62	**Documentation**	**95 days**
63	Progress Assessment	0 days
64	Develop Extended Abstract	1 day
65	Extended Abstract Due	0 days
66	Develop Draft Dissertation	15 days
67	Partial Draft Dissertation Due	0 days
68	Develop conference presentation	5 days
69	Conference Presentation	1 day
70	Develop final dissertation	15 days
71	Project Dissertation Due	0 days