Title: , Autonomous Helicopter Navigation System, System Level, Ground Control Station Design Document

*“A Project”*

Prepared by Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

,

Checked by Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

,

Approved by Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

, Student Manager 2010

Authorised for use by Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr Luis Mejias, Project Coordinator

**QUT Avionics**

Queensland University of Technology

CRCSS-EESE, GPO Box 2434

Gardens Point Campus

Brisbane, Australia, 4001.

Telephone (+61 7) 3864 1772

Facsimile (+61 7) 3864 1517

e-mail luis.mejias@qut.edu.au

web <http://code.google.com/p/ahns10/>

This document is Copyright 2010 by the QUT. The content of this document, except that information which is in the public domain, is the proprietary property of the QUT and shall not be disclosed or reproduced in part or in whole other than for the purpose for which it has been prepared without the express permission of the QUT

**Revision Record**

|  |  |  |  |
| --- | --- | --- | --- |
| Document Issue/Revision Status | **Description of Change** | **Date** | **Approved** |
|  | Initial Issue |  |  |

**Distribution List**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Affiliation** | **Distribution Date** | **Approved** |
| Avionics Lab File Archive | QUT Avionics |  |  |

**Foreword**

As part of the Autonomous Helicopter Navigation System (AHNS) project for 2010 ground control station (GCS) software for a Linux based desktop computer was developed. The purpose of the GCS in a systems context is to provide an interface for sensor data monitoring and mission flight control. This purpose originates from the project’s fifth High Level Objective.

The systems engineering methodology required specification of system requirements from the customer and also their derivation during preliminary design. Three requirements from the customer, treated as baseline, specify WiFi link communications and control of the aircraft flight mode. Four derived system requirements specifying data handling and display. All requirements were used in the design process to ensure each GUI widget had a specific purpose. The GCS was also designed with threading to ensure responsive GUI behaviour and to ensure efficient telemetry.

Following design the implementation of the main GUI class and threading was undertaken. Concerns in this implementation stage included cross-thread resource sharing, data duplication and GUI update rate. The solution to cross-thread resource sharing was to use the Qt signal/slot mechanism. This was used to ensure centralised data access and GCS update rate control. In cases where systems testing led to the requirement for implementation changes, such as the display of the RC commands and mode, the derived requirements needed to be reviewed. The completion of the design, implementation and initial systems testing stages signifies the readiness for AHNS 2010 GCS acceptance testing.

**Table of Contents**

Paragraph Page No.

[1 Introduction 8](#_Toc275080427)

[1.1 Scope 8](#_Toc275080428)

[1.2 Background 8](#_Toc275080429)

[2 Reference Documents 9](#_Toc275080430)

[2.1 QUT Avionics Documents 9](#_Toc275080431)

[2.2 Non-QUT Documents 9](#_Toc275080432)

[3 High Level Objectives, System Requirements and Acceptance Testing 10](#_Toc275080433)

[4 Design 13](#_Toc275080434)

[4.1 Thread Architecture 13](#_Toc275080435)

[4.2 GUI Design 14](#_Toc275080436)

[4.2.1 Artificial Horizon 15](#_Toc275080437)

[4.2.2 System Status 15](#_Toc275080438)

[4.2.3 Communications 15](#_Toc275080439)

[4.2.4 Received Data Console 16](#_Toc275080440)

[4.2.5 Transmitted Data Console 16](#_Toc275080441)

[4.2.6 Data Plotter 17](#_Toc275080442)

[4.2.7 Blackfin Camera Feed 17](#_Toc275080443)

[4.2.8 Control Parameters 18](#_Toc275080444)

[4.2.9 Control Gains 18](#_Toc275080445)

[4.2.10 Flight Control 19](#_Toc275080446)

[5 Implementation 20](#_Toc275080447)

[5.1 Main Window Implementation 20](#_Toc275080448)

[5.2 Thread Implementation 21](#_Toc275080449)

[5.2.1 Telemetry Thread Implementation 22](#_Toc275080450)

[5.2.2 Vicon Thread Implementation 23](#_Toc275080451)

[5.3 Widget Implementation 23](#_Toc275080452)

[5.4 Iteration and Tools in Implementation 24](#_Toc275080453)

[6 Conclusions 25](#_Toc275080454)

[7 Recommendations 26](#_Toc275080455)

**List of Figures**

Figure Page No.

[Figure 1 – Initial GCS Architecture 13](#_Toc275080456)

[Figure 2 - Final GCS Architecture 14](#_Toc275080457)

[Figure 3 – OpenGL Artificial Horizon Widget 15](#_Toc275080458)

[Figure 4 - System Status Widget 15](#_Toc275080459)

[Figure 5 - Communications Widget, Telemetry Tab 16](#_Toc275080460)

[Figure 6 - Communications Widget, Vicon Tab 16](#_Toc275080461)

[Figure 7 - Received Console Widget 16](#_Toc275080462)

[Figure 8 - Transmitted Console Widget 17](#_Toc275080463)

[Figure 9 - Data Plotter Widgets 17](#_Toc275080464)

[Figure 10 - Parameter Control Widget 18](#_Toc275080465)

[Figure 11 - Gains Control Widget 18](#_Toc275080466)

[Figure 12 - Flight Control Widget, Control Tab 19](#_Toc275080467)

[Figure 13 - Flight Control Widget, State Tab 19](#_Toc275080468)

[Figure 14 - Member Objects of the gcsMainWindow Class 20](#_Toc275080469)

[Figure 15 - GCS Thread Inheritance 21](#_Toc275080470)

[Figure 16 - Non-GUI Widget QObjects 21](#_Toc275080471)

[Figure 17 - UDP Datagram Format 22](#_Toc275080472)

[Figure 18 - Inheritance of Widgets from QWidget 23](#_Toc275080473)

[Figure 19 - Inheritance from QGLWidget 24](#_Toc275080474)

**List of Tables**

Table Page No.

[Table 1 - GCS System Requirements 10](#_Toc275080475)

[Table 2 - GCS System Requirement Acceptance Tests 11](#_Toc275080476)

**Definitions**

|  |  |
| --- | --- |
| AHNS | Autonomous Helicopter Navigation System |
| QUT | Queensland University of Technology |
| HLO | High Level Objective |
| GCS | Ground Control Station |
| GUI | Graphical User Interface |
| HMI | Human Machine Interface |
| UI | User Interface |
| RC | Radio Control |
| AT | Acceptance Test |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Introduction

The Autonomous Helicopter Navigation System (AHNS) project for 2010 entailed the design, development and use of GNU/Linux based ground control station (GCS) software. As part of the systems design, the GCS provides one of the two interfaces between system operators and the quadcopter controller (the second interface being the pilot’s radio control (RC) link). The use of the GCS to present flight data from onboard sensors for monitoring and for mission control means the systems engineering processes of design, implementation and testing are all critically important.

## Scope

This document considers the design and implementation of the GCS. It does not consider the details of network code design or implementation as this is external code, as was the use of OpenGL in several of the Qt widgets.

## Background

AHNS 2009 left a considerable amount of legacy C++ OpenGL and Qt code. Learning OpenGL fell outside of the scope of AHNS 2010 due to a smaller group size but where possible code was reused with minimal modification. The legacy Qt threading and network UDP code was judged unsuitable after testing and code inspection. In 2010 the use of WIFI dictated a GCS capable of multiple simulations threads, rather than sequential execution. It was also noted that UDP does not required application level packet synchronisation; a timestamp sufficies in detecting delayed and duplicate packets.

# Reference Documents

## QUT Avionics Documents

|  |  |  |
| --- | --- | --- |
| RD/1 | AHNS-2010-SY-HL-001 | AHNS, High Level Objectives of |
| RD/2 | AHNS-2010-SY-SR-001 | AHNS, System Requirements of |
| RD/3 | Heliconnect10 | Molloy, T. Heliconnect10. Description of the messages that can be sent in heliconnect10 udp packets. 2010.  Available: <http://code.google.com/p/heliconnect10/wiki/packetmessages> (accessed October 16 2010) |

## Non-QUT Documents

|  |  |  |
| --- | --- | --- |
| RD/4 | Qwt | Rathmann ,U. Qwt - Qt Widgets for Technical Applications. 2010.  Available:  [<http://qwt.sourceforge.net/>](http://code.google.com/p/heliconnect10/wiki/packetmessages) (accessed October 16 2010) |
|  |  |  |

In the event of any conflict between this document and any RD referenced herein, such conflict shall be notified to Dr Luis Mejias.

In the following text, RD/x identifies referenced documents, where "x" denotes the actual document.

# High Level Objectives, System Requirements and Acceptance Testing

Delivery of a GCS was flagged in initial customer discussions and finalised in the fifth project high level objective. From the high level objective several system requirements were created.

[HLO-5] Ground Control Station from [RD/1] established that a ground control station that supports appropriate command and system setting inputs and data display and logging should be developed. The design should be derived from previous AHNS developments and enable future ground station developments.

The system requirements shown in Table 1 are divided into baseline and derived requirements. Baseline requirements were specified by the customer and from previous AHNS experience whilst derived requirements were developed after a period of preliminary systems design.

Table - GCS System Requirements

|  |  |  |
| --- | --- | --- |
| SR-B-02 | The GCS shall enable autopilot flight mode switching between manual, stability augmented flight, and autonomous station keeping. | AT-02 |
| SR-B-08 | The autopilot system gain and reference parameters shall be updatable in flight using an 802.11g WLAN uplink from the GCS. | AT-08 |
| SR-B-09 | The airborne system shall transmit telemetry data including state data to the GCS using 802.11g WLAN. | AT-09 |
| SR-D-07 | The airborne system shall transmit all actuator inputs, including radio control inputs, to the GCS. | AT-17 |
| SR-D-08 | The GCS shall log all telemetry and uplink data communications. | AT-18 |
| SR-D-09 | Aircraft state data and control inputs received shall be displayable on the GCS along with appropriate time references. | AT-19 |
| SR-D-10 | The GCS shall provide display of avionics system health monitoring including telemetry, uplink, radio control link and battery level status read-outs. | AT-20 |

The requirements were written to avoid subjectivity in judging if they had been met however the difficulty in testing a GUI based software system was still a concern. Concern came from the inability to use automated testing tools due to limited development time. This method of testing could have ensured stable code operation by profiling resource usage, checking for improper threading and suggesting areas of optimisation or code review. The other concern with GUI testing is the subjective nature of operator expectations. To alleviate the concerns it was necessary to define the specific methods that would be used for conformance testing with the customer at the same stage as system requirement definition. The acceptance tests are listed in the table of system requirements and described in [RD/2] and Table 2.

Table - GCS System Requirement Acceptance Tests

|  |  |  |
| --- | --- | --- |
| AT-02 | Inspection | The GCS will switch between the three modes while the platform in on the ground and while in the air. The operator will ensure that the onboard processor has received the commands and activate the corresponding mode. |
| AT-08 | Inspection | System gain and reference parameters will be updated from the GCS to the onboard processor. The operator will inspect the platform in-flight to ensure that the uploaded data has been modified. |
| AT-09 | Testing Log Data | Logged telemetry data, including state data, will be inspected to ensure that the onboard processor is sending the correct information, and that is being received by the GCS. |

|  |  |  |
| --- | --- | --- |
| AT-17 | Testing Log Data | After a flight test while the platform is transmitting information to the GCS, the log data will be analysed to ensure that all actuator inputs are received. |
| AT-18 | Testing Log Data | After a flight test while the platform is transmitting information to the GCS, the log data will be analysed to ensure that the telemetry and uplink data communications are received. |
| AT-19 | Inspection | During the flight test, the transmitted aircraft state data and control inputs will be inspected on the GSC for accuracy. |
| AT-20 | Inspection | During the flight test, the transmitted avionics system health monitoring including telemetry, uplink, radio control link and battery level status will be inspected on the GSC for accuracy. |

# Design

2009 demonstrated the development of both a HMI and autopilot GUI. It was noted that the HMI was important in providing real time data visualisation whilst the autopilot GUI was needed to update and run the flight control. Splitting control and data display obviously adds additional and unnecessary interfaces. To meet the 2010 HLO and system requirements it was decided to merge the functionality of both 2009 GUIs into a single GCS.

## Thread Architecture

The GCS architecture is based on the philosophy that time-critical or background data processing should be performed in a separate thread to that running the GUI. In Qt this was suggested to be the case as both the GUI and network objects require event loop processing for user events and datagram receive events. Being able to create multiple event loops using threading therefore removes the possibility of the GUI becoming slow and unresponsive.

Figure 1 shows the initial architecture of the GCS based on two threads. The telemetry thread was originally envisaged to receive, transmit, log and display all airborne system data via a UDP WiFi communication link. Following initial implementation however the existence of the GUI objects in a separate thread meant the telemetry thread could realistically only be run in the background to check for data that had been received or required transmission. The GCS thread therefore handles user input events and data display and logging.



Figure – Initial GCS Architecture

The final GCS architecture of reflects several more iterations in implementation and testing. Three threads are deemed necessary to enable the GCS to display, access and forward data from onboard and Vicon sensors. The Vicon thread is necessary to use the proprietary C++ dynamic library to access object position and pose information without locking the GUI. Vicon data is accessed through the GCS rather than the flight computer because the on board software is not coded in C++. The computer running the GCS is also often more powerful and can be connected to the server using Ethernet rather than WiFi when latency is a concern.



Figure - Final GCS Architecture

## GUI Design

The use of the GCS in every flight test necessitates efficient GUI layout. Layout design was judged to be an individual’s preference and would also change depending on the stage of flight. For example once telemetry was engaged its settings would not need to be modified. The approach of designing a static GUI layout in 2009 led to a considerable amount of time being dedicated to designing an operator efficient layout. A static layout also means GUI layout modifications require code changes, a condition that increases the risk of untested code being used after system integration. GUI design was therefore not based on layout but on making design choices to enable layout customisation.

With knowledge of the Qt GUI framework the means of giving the user control of the GUI layout is to create dockable, atomic GUI units termed *widgets*. Each widget can be dragged and docked in different locations inside or outside the main window.

The atomic nature of the widgets enables code reuse and unit testing; both considered industry best practise in software design. The purpose and design of each individual widget is associated with a specific system requirement.

### Artificial Horizon

The artificial horizon was designed in 2009 for the now defunct HMI GUI. Its aim was to provide an OpenGL visualisation of the current aircraft altitude and pitch and roll attitude. After testing it was found that altitudes over 10 meters could not be displayed. The redesign abstracts it as a standalone widget and disables the altitude display. Its use continues as it contributes to the array of state data displayed in fulfilment of SR-D-09.

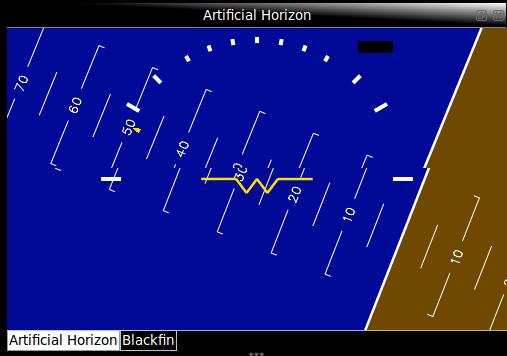


Figure – OpenGL Artificial Horizon Widget

### System Status

The system status widget is the primary method of displaying onboard system health and status information as required by SR-D-10 and SR-D-07. It is designed to display the RC link status, battery level and commanded RC inputs. The commanded RC pulse information is displayed in the form of percentage of range 1000 to 2000 micro-seconds.

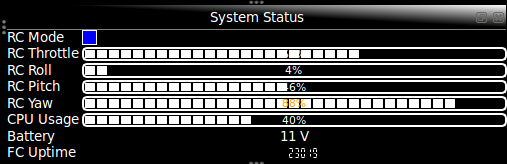


Figure - System Status Widget

### Communications

The communications widget is designed to enable configuration of the network connections to both the flight computer and Vicon. Settings for each connection include server IP addresses and port. Buttons provide thread start, stop and restart control whilst a timer for each thread monitors uptime and connection status. This enables fulfilment of SR-B-08 and SR-B-09.

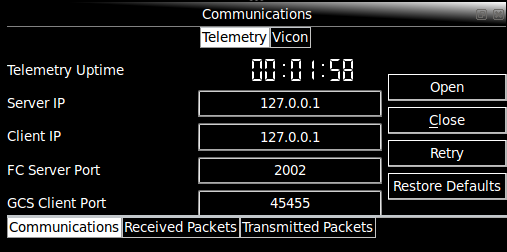


Figure - Communications Widget, Telemetry Tab

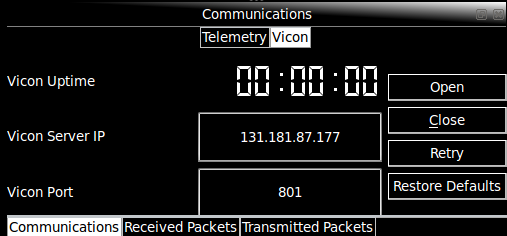


Figure - Communications Widget, Vicon Tab

### Received Data Console

The received data console is designed to provide information on the packets being received from the flight computer. It displays timestamps and details of the packets received. Summary information provided includes packet rate and packet discard count. The function to enable all packets to be shown enables completion of SR-D-08 whilst the time reference display of packets meets SR-D-09.

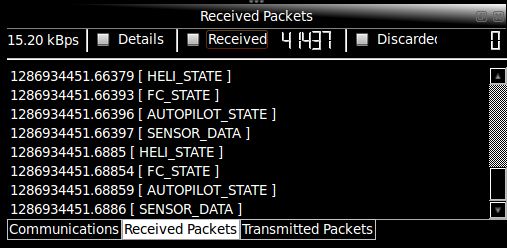


Figure - Received Console Widget

### Transmitted Data Console

The transmitted data console is the complement to the received data console. It displays the status of all transmitted data packets in fulfilment of SR-D-08. It includes a transmission count and rate.

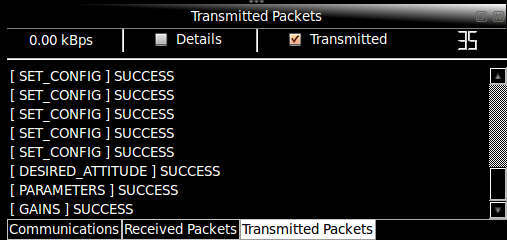


Figure - Transmitted Console Widget

### Data Plotter

The data plotting widget was included in the GCS design based on feedback from AHNS 2009. It provides real-time data plotting of data from the sensors, autopilot, flight computer and Vicon to fulfil SR-D-09. The widget does not manage its own data, instead it accesses a centralised, shared array of data. This enables multiple data plotters to run with minimal memory overhead.

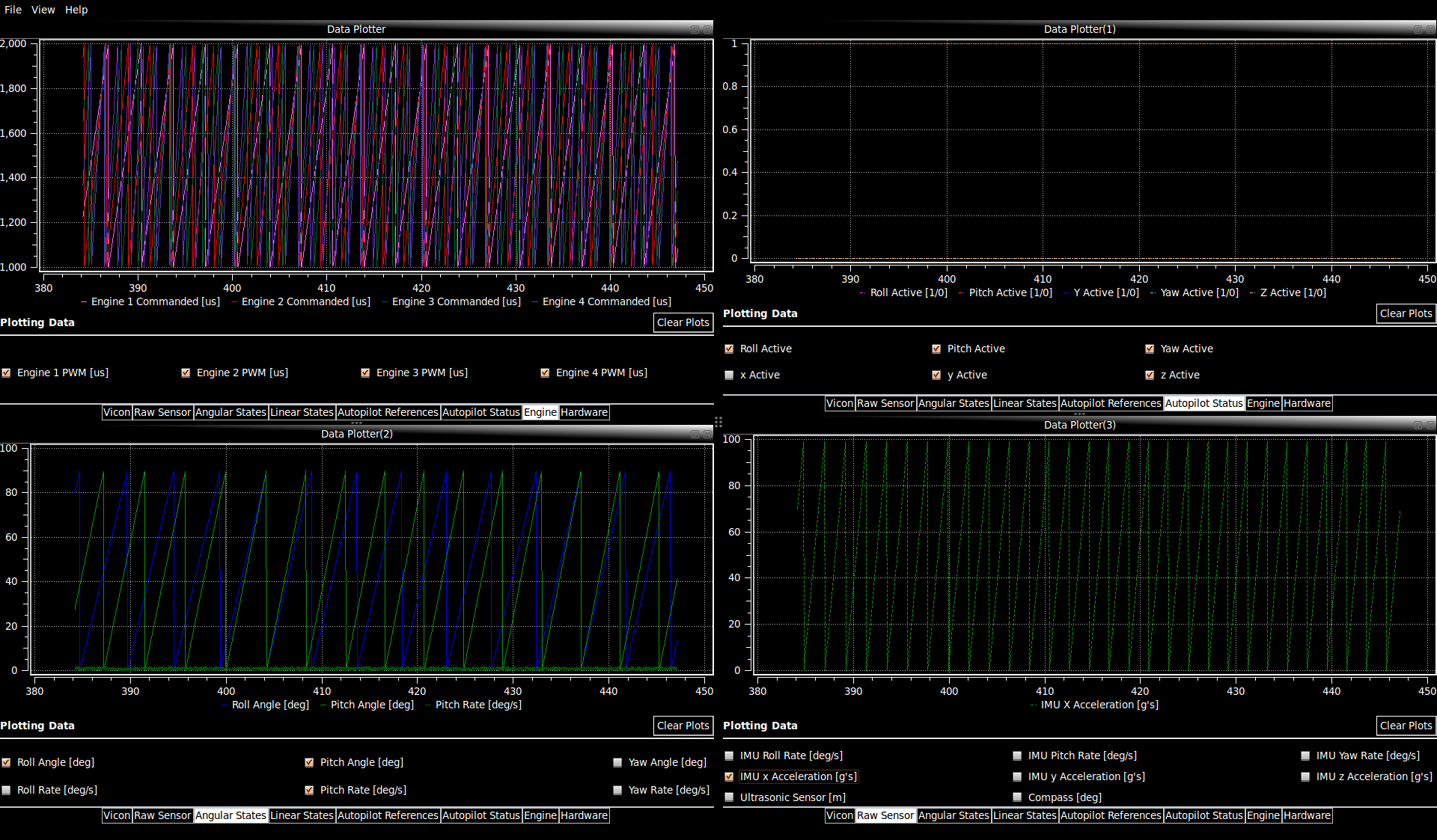


Figure - Data Plotter Widgets

### Blackfin Camera Feed

The Blackfin Camera widget was as part of the State Estimation subsystem. It demonstrates the ability for multiple team members to contribute widgets without concern for other aspects of the GCS. It enables connection and configuration of the camera and OpenGL display of a live video feed through the camera’s 802.11 WiFi interface.

### Control Parameters

The purpose of the parameter control widget is to enable the operator to set, save and load the trims and bounds on all active control loops without modifying the onboard flight computer code. This directly aims to meet SR-B-08. During some control loop implementation stages it is nonsensical to bound the control loop outputs, in these cases the parameters can be used to transmit other required parameters to the flight computer.

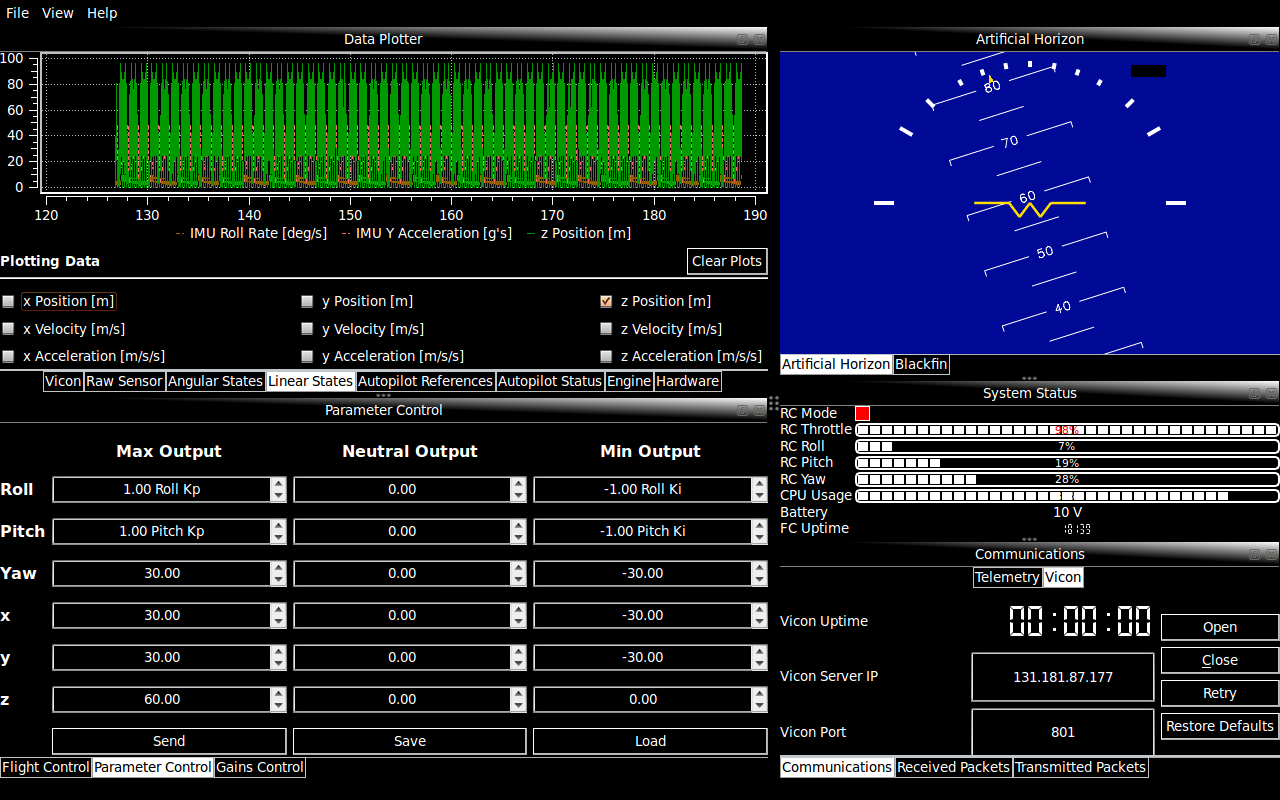


Figure - Parameter Control Widget

### Control Gains

SR-B-08 requires the ability to modify gains from the GCS. The control gains widget is designed to provide the GCS operator with a means of setting, saving and loading the PID control gains on all airborne control loops.

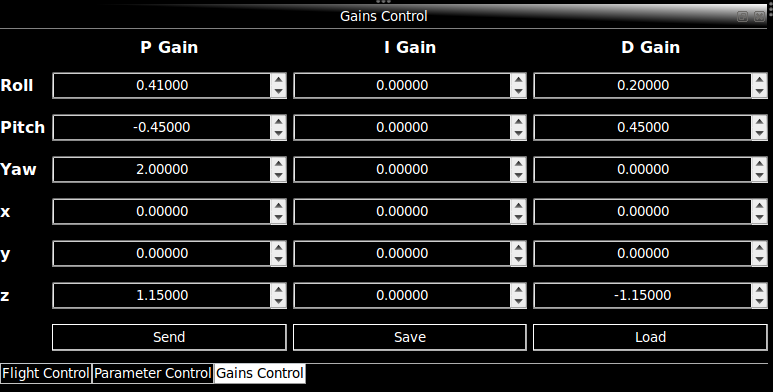


Figure - Gains Control Widget

### Flight Control

The flight control widget is designed to meet SR-B-02. It is designed to enable the operator to set and control the active control loops and report current settings. The risk of uncontrolled flight is also countered through inclusion of a manual ‘*KILL*’ button to force the MCU and all engines into failsafe. The inclusion of Vicon in the state estimation system required that there be a means of online switching between Vicon and onboard sensors.

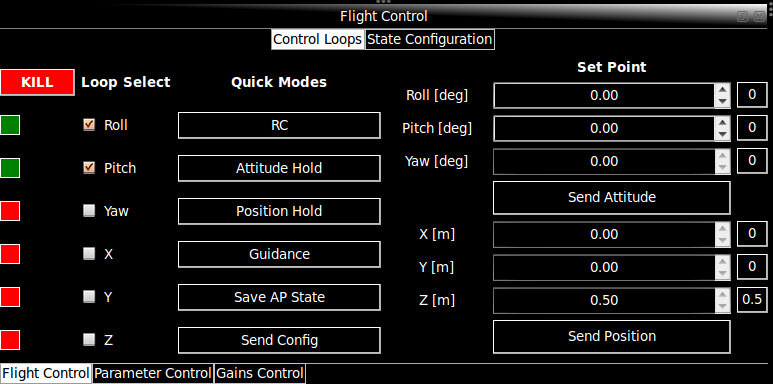


Figure - Flight Control Widget, Control Tab

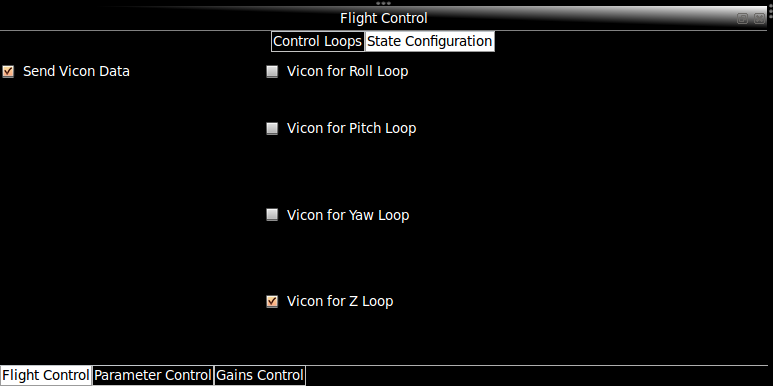


Figure - Flight Control Widget, State Tab

# Implementation

The use of threads, dockable widgets, UDP network code and OpenGL led to considerable challenges in C++ Qt development. To ensure the final implementation was such that the system requirements were met several iterations of design and implementation were required.

## Main Window Implementation

The GCS is based on the gcsMainWindow class derived from the QMainWindow class. This class runs the main event loop called from the main function of the GCS process. The class acts as the parent of all the graphical widgets and non-graphic objects used in the GCS. All inter-thread and inter-object signal/slot connections are made within this parent class. Figure 14 is a diagram of the members of the class, including the object variable names and classes. The window provides a stylised visual canvas but once each widget is placed in QDockWidget objects they can be moved freely on screen.



Figure - Member Objects of the gcsMainWindow Class

## Thread Implementation

The threads implemented in the GCS are derived objects of the class QThread as shown in . Not shown in the parent process, referred hereafter to as the GCS thread. The telemetry and Vicon threads are required to provide a continuous supply of information that the GCS thread is required to display and log. The use of QThreads enabled use of a Qt signal/slot mechanism to provide non-blocking, cross thread resource sharing. A mutex or semaphore based system was not deemed appropriate as the GUI would be unresponsive as a telemetry or Vicon data transmit or receive operation was completed.

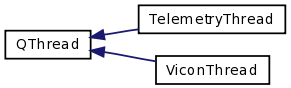


Figure - GCS Thread Inheritance

The disadvantage of the signal slot mechanism is obviously the need for a pass by value function call. Pass-by-value signals are only used for cross-thread signals and with simple structure data types to reduce overhead. In the GCS thread a centralised data storage object was also created to given all widgets centralised access to the cross-thread telemetry and Vicon data without multiple slot function calls (Figure 16).

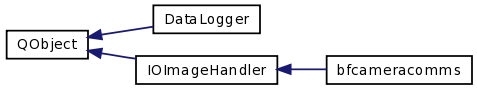


Figure - Non-GUI Widget QObjects

The QThreads are started and stopped through emission of *ConnectionStart*, *ConnectionClose* or *ConnectionRetry* signals from the corresponding buttons in the GCS Communications widget. The signals are handled by slots in the gcsMainWindow. The slots pass the server and client information required for network connection to the thread object constructors. This implementation enables both the main window and the communications widget to monitor connection status by simply checking the thread’s existence.

### Telemetry Thread Implementation

Besides the initial creation of the QUdpSocket, the thread’s execution is entirely event driven. That is, the thread’s operation is limited to running its event loop which operates on a first-in-first-out queue of signals. This is avoids manual polling of the socket for received data as the *readyRead* signal is automatically added to the event loop queue. When executed the received signal calls the *DataPending* slot to extract the packet type and data. The slot then emits a signal that contains the received data to the GCS thread for data logging and display.

Data transmission is also achieved using the cross-thread Qt signal/slot system. Data structures that require transmission are emitted with signals from widgets in the GCS thread. The signals are connected to unique slots in the telemetry thread which function to pack the data into raw byte form within QByteArray objects. The raw data is then packed with a message header into a datagram to be transmitted using *writeDatagram* method of the QUdpSocket. The packet format is shown in Figure 17 with command types specified in the common files from heliconnect10 [RD/3].

|  |  |  |
| --- | --- | --- |
| ***Header*** | | ***Message*** |
| Time Stamp (32bit timeval) | Command Type (uint32\_t) | Data Bytes (Packed struct) |

Figure - UDP Datagram Format

UDP is not connection oriented but the airborne and GCS telemetry code does have application level checks for reliable data transmission. To initially confirm the GCS has communication with the server, the socket sends a packet with the type COMMAND\_OPEN and no data. If communication is established the server will respond in kind with a message of type COMMAND\_ACK and data of COMMAND\_OPEN. The telemetry thread queues timer events that periodically check for replies and resend the packet. After not connecting for a period time the thread notifies the GCS thread of its failure to connect and terminates. For critical autopilot packets the same send, acknowledge and retry implementation is followed. Instead of terminating the thread on failure however a failed message is displayed in the transmit console GUI widget and the thread continues normal operation. Vicon data packets requiring transmission are not implemented with any form of reliability above standard UDP due to their high update rate.

### Vicon Thread Implementation

The Vicon thread does not rely on Qt network objects and it is not event driven. Proprietary code is used to create a client object using the server network address provided and create a connection. If connection is successful the thread will run until stopped. Conversely if the client fails to connect the thread is terminated and the gcsMainWindow object is notified. During operation the Vicon server is queried and data is received using vendor functions. The queries are time intensive and if threading is not implemented correctly these cause the GCS thread to become unresponsive. Once a query yields valid Vicon state data these are emitted to the data logger and GCS thread objects using Qt signal/slots.

## Widget Implementation

Implementation of the individual widgets is a relatively straight forward task of coding GUI object events. Besides the obvious button click events, other objects such as the text edit boxes in the ParameterControl and GainsControl class trigger signal emits. The challenge for all widgets is placing them in individual QDockWidget objects to enable rearrangement. To do this every widget needs to inherent from QWidget or QGLWidget as shown in Figure 18 and Figure 19. The objects are also created as members of the gcsMainWindow class as shown in Figure 14. Finally QDockWidget objects are created and added to the dockable areas of the main window. In the case of the DataPlotter widgets, where multiple instances of the class can exist, multiple QDockWidgets are required.

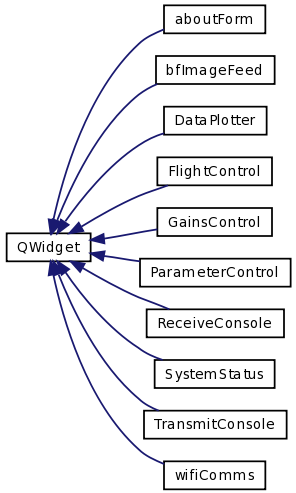


Figure - Inheritance of Widgets from QWidget

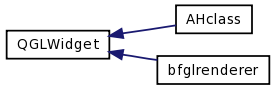


Figure - Inheritance from QGLWidget

To achieve multiple DataPlotter objects the gcsMainWindow class includes two QLinkedList objects. One is used to track the DataPlotter widgets and the other their QDockWidget objects. Tracking the objects is important to ensure dynamic creation and deletion of the objects. The use of the third party, open-source graphing widgets from Qwt [RD/4] posses no hurdles as data is accessed from the centralised DataLogger object. Note that creating new DataPlotter widgets does not lead to data duplication.

## Iteration and Tools in Implementation

The design of the GCS is such that if implemented correctly all the HLOs and SRs should be met. During software integration testing however code inefficiencies and poor performance not concerning the SRs were noticed. This included increased processor load due to excessive number-to-string conversions in the transmitted and received consoles and redrawing OpenGL scenes and replotting data with each received packet.

The iterative nature of systems engineering when applied to software development meant new implementation methods were sought. To overcome the number-to-string conversions options are included to show only that data that is necessary. Likewise update performance of the Artificial Horizon and plotting widgets is controlled by a 25Hz timer in the gcsMainWindow window. The GCS event loop therefore can reduce the update rate to significantly less than the telemetry rate.

Systems integration testing has also resulted in code reimplementation. The System Status widget is an example of flight test results determining code level changes. Pulses original displayed in the widget were the individual engine commands and the RC Link was set by a throttle failsafe value. After an incident involving a flipped quadcopter it was deemed necessary for the GCS operator to have oversight of the RC pilot’s commanded inputs and control mode. The GUI design remained essentially unchanged with the exception of the revised data to better meet the SRs of data display.

The iterative software implementation process required the use of many tools including the Qt Creator for code development and Qt documentation, Doxygen for code documentation and review, SVN for version control and tcpdump for network debugging. The process yielded a stable GCS under Ubuntu with OpenGL, Vicon and Qwt code contributing to instabilities under Mac OSX.

# Conclusions

This document has described the design and implementation of the GCS to meet the associated High Level Objective and the seven System Requirements. The use of a Qt C++ framework has enabled design and implementation of a user customisable GUI. Each GUI object’s purpose and implementation is traceable to a system requirement or test scenario demonstrating the adherence to systems engineering and successive refinement.

# Recommendations

It is recommended that the final GUI design described undergoes formal acceptance testing. In the event the AHNS team and/or customer do not agree the test criteria are met the outcomes of the tests will be incorporated into future document revisions.