University of Southern Queensland

Faculty of Engineering & Surveying

Quality Assurance For Virtual Reference Stations

A dissertation submitted by

Jamie Heit

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Spatial Science (Surveying)

Submitted: Octoboer, 2005

Abstract

The Department of Natural Resources and Mines has identified the need for Quality Assurance in all aspects of the operations of VRS. Their particular concern being with data reliability at the rover and the ability to detect any anomalies as part of a strategy to identify, minimise and manage potential risks associated with the VRS.

This thesis investigated and validated a system of monitoring and assessing the performance of RTK-VRS by developing software which outputs a summary of characteristics of the performance of VRS at a particular geographic location and time.

Quality Assurance software was created in this project to help minimise the lack of Quality Control at the rover end of the VRS system. This program deals with data collector files output from a Survey Data Recorder and creates summaries of particular characteristics relating to the accuracy of the VRS system.

This software can now be implemented by Surveyors using the VRS system in South-East Queensland. Providing them with a Quality Assurance tool to use at the Rover end of the VRS network and partially solving the problem of a lack of Quality Assurance for VRS.

Certification of Dissertation

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.

Jamie Heit

Q11222807

____

Signature

____

Date

Acknowledgments

This research was carried out under the principal supervision of Mr. Peter Gibbings. I would like to sincerely thank him for all of his guidance and sharing his wealth of experience throughout the year.

Appreciation is also due to Mr. Mathew Quarisa for his abundance of knowledge and assistance. Much gratitude must go to Matt Higgins and Garry Cislowski from the DNRM for their support.

Last, but not least, many thanks must go to my loving family and partner for their help and understanding through this period.

Jamie Heit

University of Southern Queensland

Octoboer 2005

Contents

Abstract **i**

Acknowledgments **iv**

List of Figures **ix**

Nomenclature **x**

Chapter 1 Introduction **1**

1.1 Background 1

1.2 Research Aim and Objectives 3

1.2.1 Aim 3

1.2.2 Objectives 4

1.3 Scope of Research 4

1.4 Justification 5

1.5 Research Method 6

1.6 Summary: Chapter 1 7

Chapter 2 Literature Review **8**

2.1 Introduction 8

2.2 Overview of VRS 9

2.2.1 eGPS Solutions Georgia, USA 9

2.2.2 GPSnet.dk Denmark 11

2.3 Software 11

2.3.1 DC file conversion 12

2.3.2 DC file manipulating software 13

2.3.3 Programming Languages 14

2.3.4 Programming Environments 14

2.4 Quality Assurance Procedures 15

2.4.1 QA at the Server 15

2.4.2 QA at the Base Stations 15

2.4.3 QA at the Rover 17

2.5 Summary: Chapter 2 17

Chapter 3 Methodology **19**

3.1 Introduction 19

3.2 Programming Language and Environment Choice 20

3.3 Characteristics of Software 20

3.4 Data Extraction 21

3.5 Software Coding 22

3.5.1 QC1 Data 22

3.5.2 Coordinate Data 29

3.6 Data Processing 31

3.7 Graphical User Interface Creation 35

3.8 Integration of Program Code to GUI 38

3.9 Software Validation 40

3.10 Summary: Chapter 3 41

Chapter 4 The QA System 42

4.1 Introduction 42

4.2 Revision of GUI Layout and Characteristic Definitions 43

4.3 Survey and User Requirements 45

4.4 Disclaimer 47

4.5 Summary: Chapter 4 47

Chapter 5 Validation 49

5.1 Introduction 49

5.2 Validation of dc Files 49

5.3 Summary: Chapter 5 51

Chapter 6 Discussion and Recommendations 52

6.1 Introduction 52

6.2 Discussion 53

6.3 Recommendations 54

6.3.1 Enhancements to the Program 55

6.4 Summary: Chapter 6 56

Chapter 7 Conclusion **57**

7.1 Introduction 57

7.2 Conclusions 57

7.3 Close 60

References **61**

Appendix A Project Specification **64**

Appendix B Source Code **66**

B.1 The `DC File Converter` C++ Function 66

List of Figures

1.1 Concept of VRS (*Trimble Home* 2005) 2

2.1 Coverage area of eGPS Solutions, Georgia, USA (*eGPS Coverage Area* 2005) 10

2.2 GPSnet.dk Coverage Area in Denmark (*GPSnet.dk Coverage Map Denmark* 2005) 12

2.3 The expected Horizontal accuracies of GPSnet.dk (*The idea behind GPSnet.dk is simple - VRS is ingenious* 2005) 13

3.1 Example of dc file opened with Notepad 24

3.2 The main screen which is shown after the execution of the software . . . 36

3.3 The design environment of the Windows Form Application 37

3.4 The Load dc file window 39

Nomenclature

The following abbreviations have been used throughout the text and bibliography:-

DNRM	Department of Natural Resources and Mines
GIS	Geographic Information System
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	A public digital cellular network
GUI	Graphical User Interface
MVS03	Microsoft Visual Studio.NET 2003
RTK	Real Time Kinematic
TGO	Trimble Geomatics Office
USQ	University of Southern Queensland
VRS	Virtual Reference Station

Chapter 1

Introduction

1.1 Background

Real Time Kinematic (RTK) Surveying is based on a differential use of carrier phase measurements of the GPS signals where a single reference station provides the real-time corrections of even to a centimetre level of accuracy. The integer ambiguity must be solved for the carrier signals L1 and L2. The rover receiver in these particular surveys should not be taken beyond 20-30 kilometres from the reference station (base station) to avoid losing lock on the satellites. (*Real Time Kinematic - Wikipedia, the free encyclopedia* 2005)

A Virtual Reference Station (VRS) is made up of GPS reference stations which are generally spaced anywhere from 25-70 kilometres apart. The data observed at these reference stations is sent to a central PC which is linked via computer networks. The central PC checks the integrity of the data from each reference station, rejecting outliers and correcting for any cycle slips that may have occurred.

Once the integrity of the data is check the central server corrects for any ionospheric, topospheric and ephemeris errors. The effect of these errors on any rover in the network can be modelled so systematic errors for RTK can be significantly reduced.

The rover in the field makes a phone call to the central server, sending its approximate position to the server. The central pc automatically receives this data and performs a geometric displacement to the location. It interpolates, applies corrections and generates a Virtual Reference Station for that Rover.

It then sends corrections as though they were coming from the virtual reference station, via mobile phone modem to the rover. Allowing the user to have real time coordinate data for their observations.

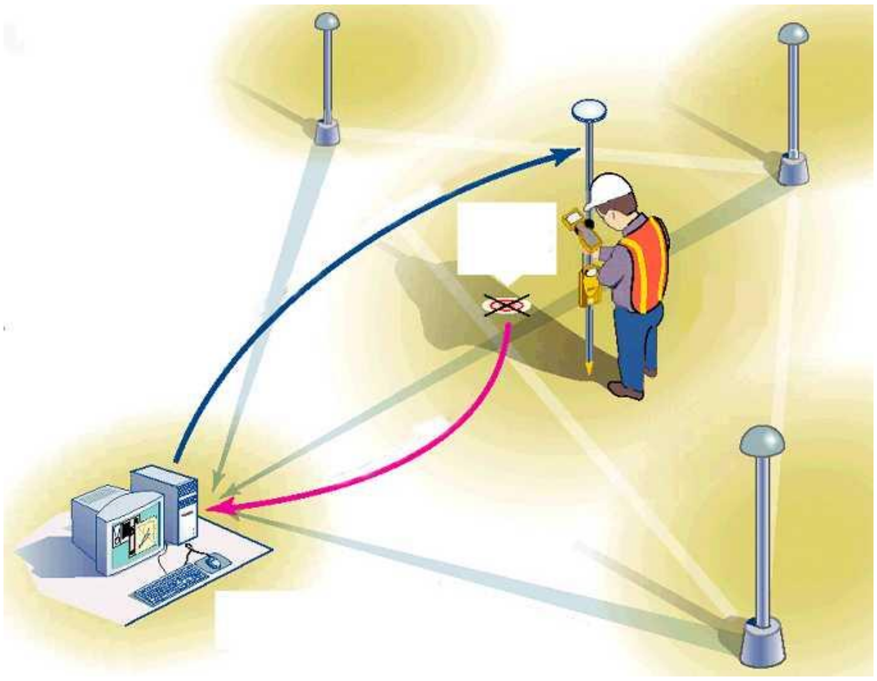


Figure 1.1: Concept of VRS (*Trimble Home* 2005)

The concept of a VRS network is a simple, but very effective one. With the regular operation of a Surveyor using GPS, the Surveyor is required to own/hire a base station, rover, radio link, and data recorder. The purpose of VRS is that it removes the need for a Surveyor in the field to setup a base station.

Background detail explaining the workings of GPS will not be discussed as this project assumes some prior knowledge of the basic workings of this system.

The Department of Natural Resources and Mines has identified the need for quality assurance (QA) in all aspects of the operation of the VRS. They are particularly concerned with data reliability at the rover and the ability to detect any anomalies in the field as part of a strategy to identify, minimise and manage potential risks associated with the VRS.

Monitoring of the VRS is necessary as a quality assurance function. If any allegation was made that the VRS was not working to its specified range of accuracy in a particular area, there could be legal consequences for the DNRM to deal with. It would be necessary to verify if there was a problem with VRS in that specific location, or if the errors had some other cause. Although VRS has some in-built assurance tools for RTK VRS surveying, this project proposes a further tool to assist with this quality assurance issue.

The issue at hand is that a Surveyor is placing trust in the coordinates which are obtained in the field via the VRS procedure. There is a potential risk that if there is a bias in the coordinate data given by VRS, the user may inadvertently end up positioning survey marks out of place.

Constant accumulation of this data will allow a larger scale area being able to be identified as giving a particular accuracy or if problems with connectivity have been encountered in the location as well as countless other benefits to the user or their company. A Surveyor would also of course make an immediate check on a known mark surrounding their survey area to let them know of any differences from the stated coordinates which may be occurring.

A Surveyors profession depends on their ability to provide a final product which is assured of being of a very high quality. The quality assurance/quality control procedures are a Surveyors way of guaranteeing this high quality outcome. Hence, this is how the creation of this project came about.

1.2 Research Aim and Objectives

1.2.1 Aim

The aim of this research is to investigate and validate (by conducting a trial) a system of monitoring and assessing the performance of RTK-VRS. A system will be developed that will allow the VRS to be monitored and assessed with a view to outputting a summary of characteristics of performance specific to a particular geographic location at a particular time.

1.2.2 Objectives

The objectives of this project are as follows:

- Research the background information regarding current QA systems for VRS and current software which manipulates survey data controller files.
- Critically evaluate existing alternatives for QA at the rover end of RTK surveying using VRS.
- Design software which will extract data from a controller file and output a summary of desired statistics.
- Test (validate) the designed software by conducting trials involving gathering data and downloading to a computer which will run the software and produce summary reports.
- Analyse results to prove the effectiveness of the designed software.

1.3 Scope of Research

This project investigated the implementation of a Quality Assurance system for RTK-VRS surveying.

The construct of this research comprised of identifying the problems, trends and needs of incorporating a Quality Assurance system as part of a strategic plan to manage the risks associated with the operation of the Virtual Reference Stations in the South-East Queensland area for the user end of VRS-RTK. The requirements of the Department of Natural Resources and Mines were incorporated in this project as they were the major sponsor. The fact the VRS infrastructure is governed by them also encourages their involvement. Their governance covers all Hardware, Software and Quality Control procedures which operate the VRS system.

The review of VRS-RTK schemes on a global scale helped to give an appreciation of the state of technology in regards to Quality Assurance systems for VRS, and which of these proved to be most effective.

1.4 Justification

The concept of VRS-RTK surveying may only seem a simple benefit, but financially, as well as the increase in productivity, it has a much greater advantage. For instance, once a base station is setup, there is a high chance that when field work commences, a Surveyor will lose sight of the base station until the job is finished. This brings problems in urban areas, and to a lesser extent rural, of the base station being removed, disturbed or even stolen. The benefit of not having to be concerned about the base station is very profitable to the surveyor. This comes about by not requiring the necessities which come at times with conventional RTK-GPS surveying, these being either paying somebody to sit at the base station for the duration of the survey, or purchasing a fence to place around the base station.

Generally, being so far away from the base station means long baselines between the rover and base station, though it is recognised that occasionally a base station may be setup beside the rover, this only occurs on the occasions in which the survey job does not cover a great area. The longer baselines lead to slightly less accurate data being captured and therefore, a final product which does not live up to the potential of its accuracy specifications.

The use of mobile phones allows the user to be much farther from the base stations than if a radio link was used as the means of communication between the rover and base. The obvious benefit from this is the fact that a far greater area may be covered by the surveyor without having to relocate the base station as would be necessary with conventional RTK.

The benefit of applying a Quality Assurance system to the rover end of a RTK-VRS surveying application is the instantaneous notification that the user will receive and the permanent record that they will have of the performance in the particular area in which they performed their survey. The benefit of the proposed software is that the user can compare the VRS data in a specific geographic location with other GPS data they may have, with the summary report intending to provide them with a log of the VRS accuracy in that area.

Its also envisaged that the DNRM will use this program in association with new safe guards built in to VRS to constantly monitor the system to provide an early warning if there is a problem with the data. This will provide the DNRM another basis of Quality Assurance in their task of ensuring high quality data is always provided to the end user.

1.5 Research Method

Specifically, the research method is divided into three stages, the first is a VRS system overview, the second is a revision of current software, and the third stage is an investigation of current QA in regards to the Server, Bases and Rover ends of VRS.

A significant problem with VRS is the lack of Quality Control, or lack of independent testing ability at the rover when used in the VRS network. Quality Assurance is well developed throughout the world in VRS in terms of the data which is sent from the central server (*Trimble Home* 2005), but this project proposes a method by which users in the field can conduct their own Quality Assurance procedure. A large number of epochs of data are gathered to provide greater confidence in the result after analysis. Once this gathered data is downloaded to a computer, the proposed software produces a summary of characteristics for the particular time and location in which the data was gathered. These summaries will allow the user to have prior knowledge of the accuracy they can be expected to get in a particular area before they leave the office.

The design for software to act as a QA system will be completed following the literature review. This design will use a data collector (.dc) file which has been downloaded to the computer as a basis for its input file and will produce a summary of average minimum number of satellites, Position Dilustion of Precion's (PDOP's), Horizontal DOP's, Vertical DOP's, initialisation times, and total GPS positions gathered in the survey for output which the user can file away for future occasions and which the DNRM can use in the larger scale issue of being able to provide users with accuracy estimates relevant to geographic location within the VRS coverage area.

1.6 Summary: Chapter 1

Monitoring of the VRS is necessary as a quality assurance function and risk management function. If any allegation was made that the VRS was not working properly in a particular area, there could be legal consequences for the providers of the VRS data. It may be necessary to verify if there was a problem with VRS in that specific location, or if the errors had some other cause. The main purpose of the software is to provide both the DNRM and end users with a warning or alert them if there is a problem.

This thesis aims to investigate and validate a system of monitoring and assessing the performance of RTK-VRS. The production of this software is hoped to partially resolve the issue of too few QA tools for VRS. The research will result in the development of software which has the capabilities to allow the real time users of VRS in the South East Queensland area when/if the coordinate data they are using is within specification. This will produce an item of QA which will perform as a considerable time-saver to many surveyors and one which will create confidence for more surveyors to embrace the real time capabilities of the VRS system and in doing so will take a massive step forward in their productivity.

A pilot study was constructed from the findings of the literature review. The outcomes of this study were used for the design and development of the Quality Assurance software needed to fulfil the requirements of this project. Once this design was completed, a process of validation occurred to purposefully induce errors and test the ability of the QA software to handle a variety of situations.

Chapter 2

Literature Review

2.1 Introduction

This chapter will review literature to establish the design and validation of a QA system which will monitor and assess the performance of the VRS for real time corrections.

The aim of this chapter is to discover and review literature which will give a general overview of various VRS establishments throughout the world, as well as literature which is relevant to current software being implemented in VRS. Finally this chapter will review an assortment of literature relating to existing Quality Assurance procedures for VRS at the central server, base stations, and rover.

A critical analysis of existing systems will provide the basis for developing a reliable method of estimating the accuracy of RTK-VRS in different locations and situations. The contemporary problems which are in place in the current VRS system with respect to QA will be discovered and reported upon. This chapter will provide the arsenal required to begin the design of the QA system software, by that, what is meant is that the needs for the QA system will be able to be perceived a great deal easier by obtaining the knowledge of what is currently in place.

2.2 Overview of VRS

This section will provide an overview of various VRS units from around the globe. This will be done to examine the infrastructure and workings behind each VRS operation. This is done to guarantee the same type of multi-based network GPS application as in South-East Queensland is in place when reviewing the QA of each VRS or CORS application

2.2.1 eGPS Solutions Georgia, USA

In Georgia, USA, eGPS solutions Inc. provides GPS users with an advanced Real Time, internet-based, GPS network. (*eGPS Solutions Internet-Based Real Time GPS Network* 2005) The eGPS solutions system runs on Trimble Navigation, Virtual Reference Station (VRS) technology and incorporates a network of fourteen permanent base stations around the central and north Georgia region. Figure 2.1 shows the coverage area in which eGPS operates.

The eGPS Solutions base station network currently serves an area of approximately 20,542 square milesroughly 35

The network software performs quality and data integrity checks continuously by relative positioning between base antennas. The network is monitored during regular business hours, 8:00am until 5:00pm, Monday through Friday.

eGPS Solutions' system reports WGS-84 datum. The antenna locations are surveyed in from HARN monuments (known as survey marks in Australia) and vertical control monuments throughout North Georgia. The values of these are presently being Bluebooked through NGS.

The maximum Positional Error Tolerance of points that eGPS solutions systematically observed using a VRS correction over a period of months and with one minute of data:

- Horizontal = 0.03'

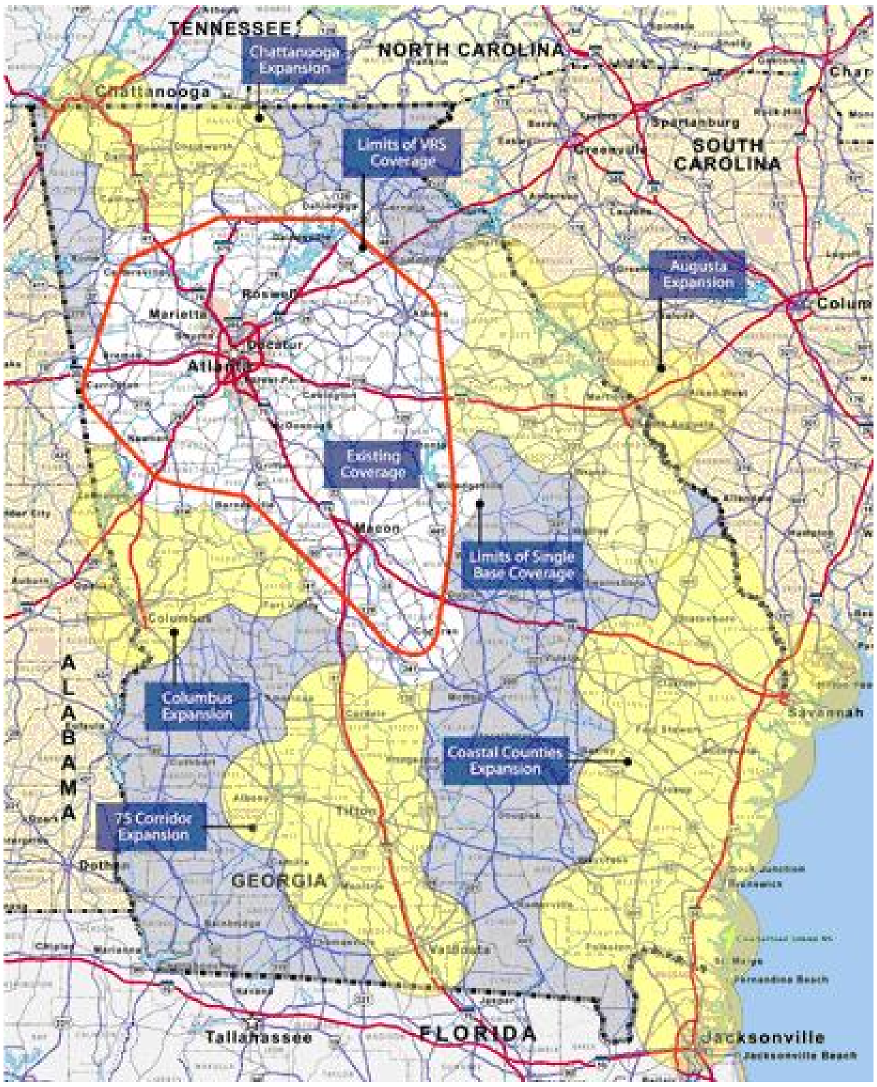


Figure 2.1: Coverage area of eGPS Solutions, Georgia, USA (*eGPS Coverage Area* 2005)

- Vertical = 0.06’

eGPS Solutions recommends that users verify their own data by observing existing survey monuments near their projects as datum verification, and that they repeat observations to model positional error tolerances for themselves. This verification would be similar to use of a calibrated baseline for EDM equipment verification. (*eGPS Data Reliability* 2005)

2.2.2 GPSnet.dk Denmark

The Trimble Center Danmark has established GPSnet.dk - an electronic Reference Net - in cooperation with the leading official and private GPS-users in Denmark.

GPSnet.dk incorporates the following characteristics:

- 100% coverage of Denmark;
- No border problems between reference stations;
- Uniform accuracy throughout Denmark;
- One phone number - to measure throughout Denmark;
- 24 hour surveillance;
- 26 Dual Frequency GPS receivers.

(*About GPSnet.dk* 2005)

Figure 2.2 shows the coverage area of GPSnet.dk throughout all of Denmark. Each of the green dots represent the reference stations. Coverage ends at 100km from the nearest reference station.

The VRS technology has significantly improved the time it takes to initialize an RTK survey. Extensive testing has shown that an average initialization time of under 1 minute is possible. 90% of all measurements are initialized under 1.5 minutes and all measurements are initialized in under 3 minutes.

The following figures give an example of the accuracies expected when using GPSnet.dk in Denmark.

2.3 Software

This section will cover the background information in regards to software which requires the use of Trimble data recorder's dc file and is currently available to the end user of

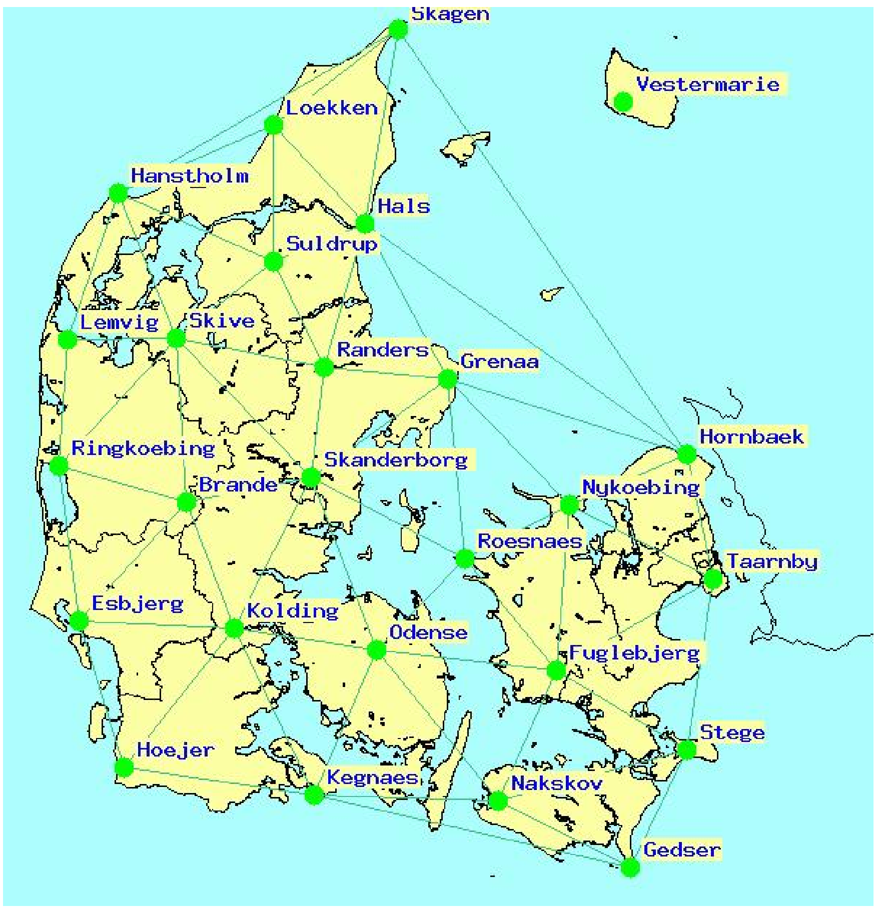


Figure 2.2: GPSnet.dk Coverage Area in Denmark (*GPSnet.dk Coverage Map Denmark* 2005)

VRS. This is done to investigate any outputs which are produced by the software and gather information on how this is done. This process is incorporated so that no long-winded unnecessary steps are done when programming is commenced for this project.

2.3.1 DC file conversion

The point of finding a conversion system for the dc file was to see if it could be converted to a text file in the format which it is much more aesthetically pleasing and legible in Trimbles' dc file editor within Trimble Geomatics Office (TGO). This would of allowed for much easier reading and simpler manipulation within software.

No programs could be found however after an extensive review of journals and information available on the world wide web. This led to the conclusion of not worrying about

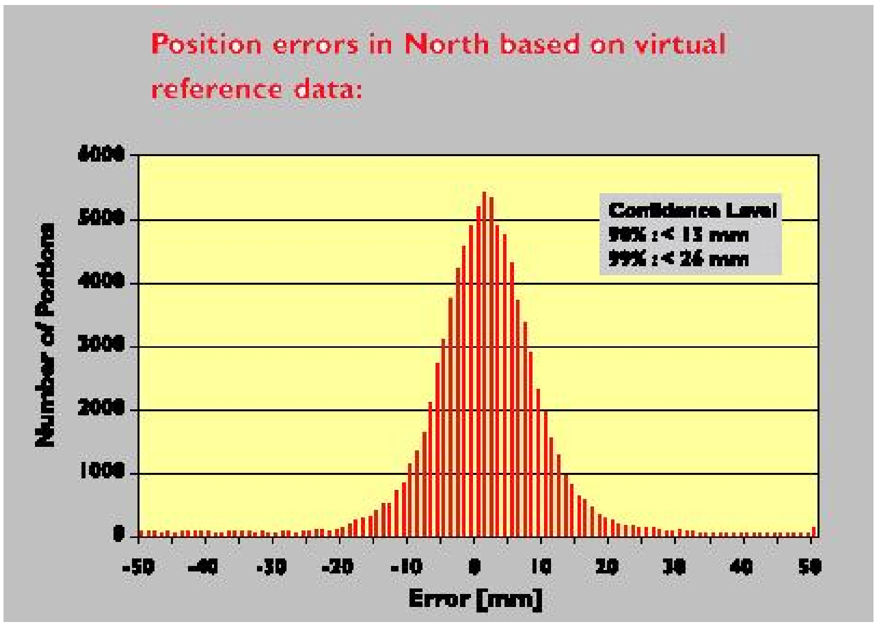


Figure 2.3: The expected Horizontal accuracies of GPSnet.dk (*The idea behind GPSnet.dk is simple - VRS is ingenious* 2005)

converting the file, but instead just finding the consistencies in the file when viewed in windows notepad. These consistencies were found and this was viewed as good enough for the software to still be implemented smoothly.

2.3.2 DC file manipulating software

Trimble Total Control Software provides the user with powerful processing and tools to enable large bodies of data to be processed extremely quickly, with extensive analysis and reporting. The software is ideal as a processing package for GPS and total station data. (*Trimble Total Control Software* 2005)

The software provides an option to analyse GPS data. The results include data for variance-covariance matrices used in the network adjustment, standard errors, and best and worst PDOP and RDOP during the measurement session. The results report can be customised to include only values of immediate interest to the user.

Data can be edited satellite by satellite, allowing the user to define satellites used, elevations and observation intervals for each line. Locations of cycle slips are shown, though the software corrects for these automatically during processing.

These processing options also helped in deciding exactly which outputs the software which is being designed should include in its summary report. The exact code used in these programs is of course unavailable for reading but the processing of files gives a good example of what can be done with survey file input.

2.3.3 Programming Languages

The development of the software within this project was to be done in any of the common programming languages known. These were C/C++, Java and Visual Basic. The pre-requisites were that it just be capable of creating a program which could take input from a file, as well as be used in the development of a Graphical User Interface (GUI).

C++ has support for both the input and output of files through a variety of classes and by the inclusion of specific header files. It also has functions to deal with the many manipulations required of the dc file to correctly extract the data. (*C++ tutorial: 6.1,Input/Output with files* 2005)

The Java programming language also deals with the input and output of files, with this being established by reading through examples and tutorials dealing with this. (*Java examples (example source code) File Input, Output Data, Input Output, Data IO Test 2* 2005)

It was discovered that Visual Basic also performs the same basic functions. With the ability of each not differing a great deal, there was little to separate the programming languages from a novice programmer's perspective.

2.3.4 Programming Environments

There was no choice as far as programming environments were concerned, with Microsoft Visual Studio.Net being available at both university and at home. The cost of purchasing another package which could perform the same functions as Microsoft Visual Studio.Net was outside the limit of spending capabilities for this research project.

The package only required the ability to handle either of the 3 programming languages previously discussed, and have the capabilities of being able to integrate any of the languages into a Graphical User Interface (GUI). The GUI being required to show the output to the user.

2.4 Quality Assurance Procedures

This section will provide an overview of current quality assurance procedures being incorporated into various VRS setups around the world. The quality assurance which is implemented at the server, bases and rover will be analysed. This is done to review which, if any, current VRS applications around the world will influence the design and operation of the QA to be incorporated into the current setup which this project is involved in.

2.4.1 QA at the Server

At the time of this project, the information on software and procedures being used in each of the researched VRS/CORS operations was very limited/non-existent. The content of these QA systems at the Server are difficult to obtain because of their expensive nature, it is known for example, that the software operating the server at the DNRM is worth in the tens of thousands of dollars.

This is the reasoning behind the lack of information on the QA procedures at the server.

2.4.2 QA at the Base Stations

Land Victoria is a section of the Victorian Government which controls the information and services about land and property anywhere in Victoria. They control a multiple reference station network called GPSnet. GPSnet spans all of Victoria, as can be seen below.

Figure 1.1 - Coverage area provided from GPSnet (Source: (*eGPS Coverage Area* 2005)

GPSnet provides position determination by differentially post-processing GPS data. They allow their users to access the files they require generally ten minutes after the end of each hour. This is so because data is stored in universal RINEX file format at five second intervals each hour.

The quality control which is in place in each of the base stations in GPSnet is a piece of software named GQC. GQC is used to assess the quality of raw GPS data in the RINEX format. It has a functional automatic mode whereby it can automatically process data generated by a GPS reference network. QGC creates summaries and sends alerts via email if quality conditions are not met.

GQC has an additional normal mode of operation to its automatic mode. The features of both modes are listed below: Normal Mode:

- Processes individual RINEX observation and navigation file to produce quality statistics.
- Quality parameters may be modified by the user.
- Output can be saved to rich text files.

Automatic Mode:

- Quality reports and summaries are created.
- Optional email error messages and warnings can be sent to alert the network manager(s) to potential quality and integrity problems.
- RINEX data can be archived.
- GQC can plot graphs of the output and save them to jpegs or bitmaps.
- GQC can create and update web pages that summarise the status of a reference network, showing plots of key quality indicators, status lights and file summaries.

This is a sample of the GQC software:

Figure 1.2 GQC software sample. Source: (*GPSnet Coverage Area Victoria* 2005)

The fundamental input required by GQC is RINEX Version 2 observation and navigation files that contain only GPS data. While the basic output of GQC is a log file containing a summary of the contents of the file and various quality indicators. When performing automatic processing, a number of log files may be produced depending on the settings in use. These log files record information regarding the operation of the program and its processes. (*GPSnet Coverage Area Victoria* 2005)

2.4.3 QA at the Rover

There was an extensive search attempting to find any QA at the Rover end of the VRS operation. This search resulted in nothing being found. Further backing the necessity for the creation of this software and justifying the topic of this research project.

The only known QA procedures at the Rover end are within the Data Recorder, with the Surveyor always being able to see on the touch screen the number of satellites, and DOP's available and the geometry of the satellites if need be.

2.5 Summary: Chapter 2

The review of literature in this chapter has helped to significantly narrow the vision of the direction of this project. Allowing a clearer path to be made in respect to the design requirements and steps involved in reaching this final design.

The review of other relevant VRS operations around the globe helped to appreciate what accuracies are thought to be relevant and to also understand the infrastructural requirements of each operation. The software relating to the manipulation of dc files helped to understand the characteristics which are generally output from software when dealing with an input of dc files. This information helped the further design of what must be included in the output of this project.

The choice between programming languages was not a difficult one considering the similarities between each. This meant that neither of the languages had an advantage, capability wise, over the other. This left the choice of C/C++ as the programming language because of previous courses studied, giving some useful background knowledge on the language. As previously mentioned the decision to use the Microsoft Visual Studio.Net programming environment was not difficult, and would of most likely been chosen even a choice was possible to purchase another.

The lack of QA at the rover end of VRS further justified the aim and objectives of this project. Allowing the design to not have any limiting parameters, and set a precedent for future work in QA software for the Rover.

The next chapter will now move on to discuss the design and methods implemented in creating the QA for VRS software. The design now has boundaries and a set of necessary characteristics which need to be included because of the findings of this chapters literature review.

Chapter 3

Methodology

3.1 Introduction

This chapter will discuss the experimental methods and design techniques used in developing software which will be used as a Quality Assurance tool for VRS. As there were very few precedents set for the design of the software, trial and error, rather than following established methods will be used in this project.

This chapter will explain in great detail the methods by which Quality Assurance software will be created and how it is planned to be validated.

Experience will be drawn from previous bodies of work to ensure that correct procedures are followed and to prevent wasting time by discovering mistakes which have already been made. Discoveries from the conclusions found in Chapter 2 will be analysed, though as previously stated, there will be only a small value from these because of the lack of experimentation and design in this specific field. The design of this software will be the most arduous task and will take up the majority of time in this project. Once the design is completed a process of validating the software will be conducted by completing a field test. 'Bad' data will be purposefully introduced to ensure that the limits of the softwares capabilities are discovered. The current VRS base station configuration will soon change. Previous stations at Beenleigh and the Landcentre will no longer be base stations in the new configuration. However, the infrastructure will remain at each of these stations. Consequently, they provide an ideal opportunity to carry out some continuous testing and monitoring of the VRS from a QA perspective. The results, further recommendations and conclusions can then be made from here.

3.2 Programming Language and Environment Choice

The initial issue which had to be analysed was which programming language would be chosen for the creation of this software. There are no set specifications from DNRM on which language they prefer, with the program only having to run on a desktop pc or laptop.

The two languages which were short-listed for use in this project were Visual Basic and C/C++. The language I decided upon was C++. This is because I have some previous experience with C and C++ from prior courses studied. These courses were CSC1100 - Foundation Programming with C and CSC2311 - Advanced Procedural Programming.

It was realised at a very early point that this software needed a Graphical User Interface (GUI) to portray the output in a simple and effective manner. This GUI needed to be created in a programming environment which can be used in the Windows operating system. This is so because of my familiarity with windows and my concern at trying to learn how to use the Linux operating system.

The decision making behind my choice of which programming environment was simple because of the access I had to Microsoft Visual Studio.NET 2003. This environment allows the user to program in Visual Basic, C/C++ and Java, thus providing the suitability I required.

3.3 Characteristics of Software

It has been established that this program will create a summary report on several characteristics which will be obtained from the Trimble Data Recorder .dc file after the user has observed and recorded enough data on a point with known or unknown coordinates. There is an option in the Trimble Data Recorder which allows the user to record QC1, QC2 and QC3 data in their .dc file. This is usually not done because of the size issues which it creates. QC means Quality Control and this section records a lot more data about the satellites and Dilution of Precision's (DOP's) than would normally be recorded. The user is only asked to record QC1 information on their .dc file for the purpose of this software, and this option can be turned off once the initial checking has been done and software has been executed. The .dc file is to be downloaded by the user to a computer with a copy of this software on it. From there the program is to be run and the summary report being the output of the execution of this software.

3.4 Data Extraction

The first problem at hand is deciding upon how to obtain the data from the .dc file which the user downloads to the pc. The .dc file is read best with the .dc file editor which is apart of Trimble Geomatics Office (TGO). The difficult decision was whether the data is read directly from the .dc file or whether it is converted to a .txt file and then read from there. If a file converter is used, then this must be able to be run as a function within the code for the software, as part of the requirements for this software is that the user only needs to run this program, and that no other manipulation of files is required on their behalf. A suitable file converter could not be found, leaving this project to extract the data directly from the dc file.

If being opened from a .txt file editor (i.e. notepad) the format of the data in a dc file becomes quite mixed and difficult to interpret which numbers represent which characteristic. The general format of the data however is always in the same order, providing an opportunity to investigate exactly the position of the QC1 data in the file and where it lies in relation to other data on the file.

This provided an opportunity to code into the software that it not store any of the read data in the file until it reads a predetermined prefix or suffix, depending of course on the location of the data in the file. For the purpose of analysis and comparison, the data in each dc file was printed out as it is seen in both the dc file editor and the text file editor. This comparison helped to locate the exact location in each file of the required data.

It was determined from the comparison and analysis that each line of QC1 data and each line of relevant GPS coordinate data had a four (4) character code at its beginning. The four character code for a line of QC1 data was "C6NM". With the code for GPS coordinate data being "08PD". The knowledge of each of these codes allowed the use of a search function throughout the whole file.

The data to be read from the file includes the following: Number of Space Vehicles (satellites), PDOP's, HDOP's, VDOP's, Easting, Northing and Reduced Level.

3.5 Software Coding

3.5.1 QC1 Data

Each C/C++ file begins with a number of header files. These header files give directives as to which functions and operations that the user can perform. For example without the header, #include <string>, the user cannot manipulate strings of data such as a line of data from the dc file. This is so because of the functions which are included in that particular header file, meaning that manipulations or enquiries of the line of data are not possible.

From this, it can be concluded that the user must ensure each necessary header file must be included at the beginning of a file used in the project. Through the help systems available in Microsoft Visual Studio.NET 2003 (MVS03) the necessary header files can be found dependant upon which functions the user requires when programming.

Moving on from the header directives, the program must next deal with the main() function. This is the first function which is executed within the code. Within the main() function many other functions can be called. These other functions can be created either before or after the main function is written, but regardless of this, the main function is always the first to be executed.

The beginning of the main function began with several variables being initialised, these variables are used to store data and they are used within some of the functions which are called within main. Their purpose is also as a counter or arrays of characters.

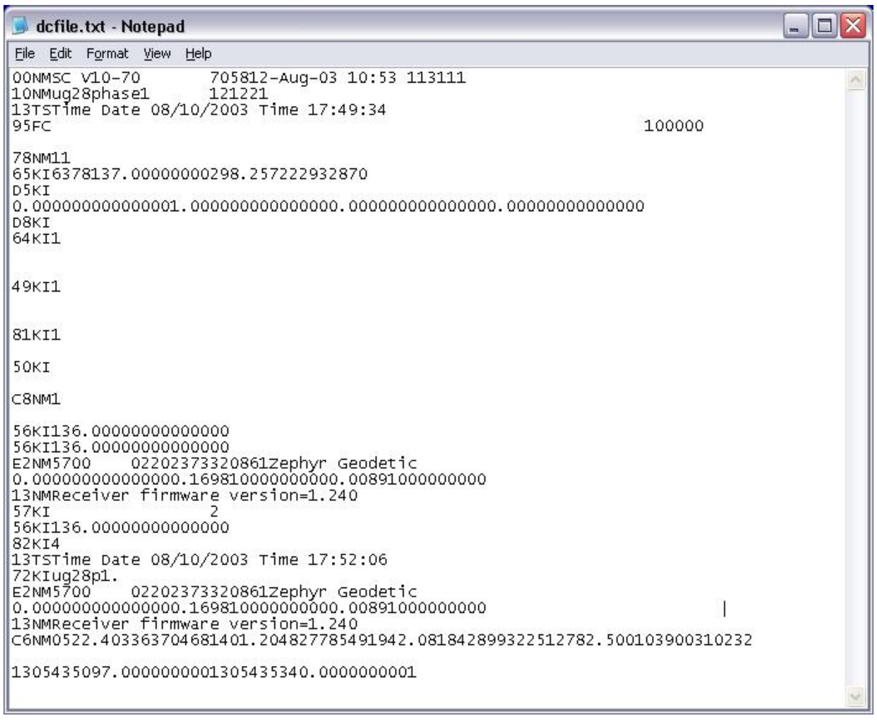
The code first checks as to whether the file chosen can be opened. If it can't, the program simply prints an error message to the screen and exits, otherwise the file is opened and is then ready for reading. The program then looks for the four character code for QC1 data (C6NM), running through each line until it reaches the end of the file. If a line in the file is found to have that code it is stored in a class along with the other relevant lines of data. If the search cannot find the QC1 data code, then it searches for one of the above mentioned Coordinate data codes. Any line of data found to have one of the coordinate data codes is then stored within coordata class, which performs operations on the line to find the data within it.

Before continuing with further methodology, it must first be explained what a class is. A class in C++ is similar to a structure in that it is like one big variable which has the ability to hold many other variables within it. It can also hold many other functions within it, allowing for many processes to be shortened by defining a class and calling it throughout the program.

For the data within the file to become usable for processing and calculations, some manipulations firstly had to be done so that they were in a legible format and showed the true values for each of the required characteristics. The data in the file came in the format shown in Figure 3.1 when opened with a text file editor.

Using this exert from a file used in the validation process, the manipulations used to get the data in their correct legible state will be described as follows. This example is for a line of QC1 data, as can be imagined, the line begins when the code C6NM is found and continues with the rest of the data in the example, it is only wrapped because of the little amount of space available here for the length of the line.

Once the program finds this code at the beginning of the line, it then begins to read in data 'after' the code. It was found that the data in the file actually matched up with



```
00NMSC V10-70       705812-Aug-03 10:53 113111
10NMug28phase1      121221
13TSTime Date 08/10/2003 Time 17:49:34
95FC                                                        100000

78NM11
65KI6378137.00000000298.257222932870
D5KI
0.000000000000001.000000000000000.000000000000000.00000000000000
D8KI
64KI1


49KI1


81KI1

50KI

C8NM1

56KI136.00000000000000
56KI136.00000000000000
E2NM5700    02202373320861Zephyr Geodetic
0.00000000000000.16981000000000.00891000000000
13NMReceiver firmware version=1.240
57KI                2
56KI136.00000000000000
82KI4
13TSTime Date 08/10/2003 Time 17:52:06
72KIug28p1.
E2NM5700    02202373320861Zephyr Geodetic
0.00000000000000.16981000000000.00891000000000
13NMReceiver firmware version=1.240
C6NM0522.403363704681401.204827785491942.081842899322512782.500103900310232

1305435097.0000000001305435340.0000000001
```

Figure 3.1: Example of dc file opened with Notepad

what was stated when opening the file with TGO's dc file editor, meaning that each decimal place, or '.' in the file was actually in the correct place, but was just attached directly to the next number and so on.

The difficulty here was to break up the numbers in the file in the correct position so that the correct number was left stored in a variable at the end. Once it was decided what needed to be done with the data in the line which began with C6NM, it was then a priority to make sure that the format was exactly the same on the other lines of QC1 data (lines with code C6NM) so that a generic solution could be made to the problem of breaking up and sorting through the numbers to give their correct values.

On further investigation, it was found that the lines of QC1 data throughout three different VRS dc files were in exactly the same format, allowing for a much easier solution to the problem than had they of been different each and every line.

Now the problem and what was required of the solution were known, I then had to sift through the MVS03 help systems to find built in functions which would allow me to manipulate the numbers in the file the way in which I knew they had to be separated.

This was what finding the functions was all about, having the ability to separate and distinguish the numbers from each other.

The function which I chose to initially separate the numbers was *char *strtok()*. This function allows the separation of blocks of data at whichever delimiters the user chooses. I chose to separate the numbers at each '.' (decimal place) and each ' ' (space). If there more than one spaces between numbers the function would sift through the line until it found the next block of data anyway, meaning the number of spaces did not matter.

The *strtok()* function was used within another function in the class created to handle lines of QC1 data. The initial line which was to be tokenised as it is called in C++ was put into a character string variable, and then the *strtok()* function was run over it. For each separation at either a decimal place or space, the block of numbers needed to be stored within a temporary variable so further manipulations could take place. A string array variable was created to handle each block of numbers which were separated from each other in the line of QC1 data.

An array is simply a variable able to hold many numbers of variables within itself at different addresses in memory. The number of variables able to be stored within the array is determined when it is declared.

To be able to store the next block of data at the next location, the array has to have counter, or variable to increment so that the next address is then available to store the next block of data, or in this case to store the next block of numbers. This can be done by running the *strtok()* function in a for loop and having the block of numbers stored after each increment in address.

A *for loop* in C++ generally comes in the form *for (i=0;i<8;i++)* {}. This all reads as follows: i=0 initializes the counter variable, i<8 can be any condition so long as it is not an infinite one, here it says while i is less than 8 and i++ says to increment the counter i while that condition holds true. The operations are then held within the {} (curly brackets). These operations are as stated earlier for this function in the class handling QC1 lines of data, which are to tokenise the line and store each block of numbers in the next address or location in the string array.

Once each block of data is separated, the next step was to find an operation which could take the last number of one block, and add it to the first number of the next block with a decimal place in between. This was needed because this is how the format of the numbers came in the line of QC1 data.

Again upon further investigation of the MVS03 help system, the option found was to use the ability to append a decimal place to the start of a block of numbers and to use the function *basic_string substr()*. This gave the ability to search through a block of numbers and move along x number of characters and then to read and therefore store y number of characters from that point. This meant that for a particular block, I could count the number of characters, decide which number/s I needed to append to the front of the next block and then use this particular function.

Before this could be done a string variable had to be created so that it could act as a temporary storage for the numbers while the operations were taking place before the correct numbers were stored in their final variables within the class created to handle the lines of QC1 data.

Once this variable was initialised it could then be used with the previously created string array which stored the blocks of numbers. This string array had seven memory addresses used. These could be accessed by simply typing that number in brackets beside the variable name (*buf[2]*), this was how each individual block of numbers was accessed in the following manipulations.

The first relative number required in each line was that of the number of minimum number of satellites for each QC1 data recording. This number is always located as a two digit number directly beside the C6NM code which identifies the QC1 line. The first digit is always a 0 after the M, so this meant that to correctly identify the number of satellites, the new string variable (bufnew) is given the value from the first block of numbers ((*buf[0]*) 0 is always the first address used in an array) and the *substr()* must be used to move along 4 characters before grabbing the next 2. (eg C6NM08 - here *substr()* would move along the first 4 characters, then grab 08 as the number of satellites, which is the correct number).

The number is then moved from the temporary storage point of bufnew to the permanent variable numsats. Before it is transferred however, the value in bufnew is currently still a string (which means it is regarded as a character, not a numerical value), so it must be transformed to an integer so that it can be used in future calculations. It is impossible in C++ to do calculations with other integers if the number you are trying to calculate with is in the form of a string or character, regardless of whether or not it still looks like 8 in both forms.

Before continuing, it is worth mentioning that in C++, an integer is regarded as a whole number, and therefore using it for satellite numbers is fine because there is never going to be a value of 8.2 or 6.4 satellites. Though it must be remembered when dealing with any form of DOP's, the program must transform these to doubles, as doubles allow a value much greater than what will be used in this field and they also allow a great number of significant decimal places. This is all relative because when it comes to calculations, each particular characteristic in the QC1 data line must be of a relevant numerical standing, otherwise no calculations are possible.

Another couple of C++ functions which are relevant to what is about to be discussed are the *unsigned int length()* function and the *basic_string append()* function. The *length()* function returns the length of the characters in a variable as a single numerical integer, allowing it to be of particular use when incorporated with the *substr()* function previously mentioned. The *append()* function allows the programmer to simply "add" or "append" a character or number to the beginning or end of a block of data, regardless of whether it is alpha or numerical. This function is used in the creation of each characteristic from QC1 which has significant figures after the decimal value.

The process used in the forming of the numerical value for PDOP's is exactly the same as that for HDOP's and VDOP's. This is so because the number of characters in each of the blocks containing their information was exactly the same.

The first block of data which contained the QC1 data code for the line and the number of satellites also contained as it's last digit, the whole number PDOP's. This is so because as was stated earlier, each line of QC1 data was broken up into blocks wherever there was a decimal place '.' or a space ' '. Therefore obviously this leaves us with the last digit of several of the blocks as the whole number, and the next block containing the numbers 'after' the decimal place, until of course the last digit, which is then the whole number for the next QC1 characteristic, be it in this case after PDOP's it is HDOP's.

The manipulations used in forming the DOP values also required the use of a temporary storage variable, so once again the string variable bufnew was used, it could be used again because the value it was storing has been passed to it's permanent address and therefore not required in the temporary bufnew anymore.

So using the *substr()* and *length()* functions it was possible to gain the whole number of each of the DOP values. The actual code read as follows: *bufnew = buf[0].substr(buf[0].length()-1,1);*

Notice the use of the *length()* function within the *substr()* function. This is done so that for the whole number of PDOP's, the length of the first number block minus 1 is given as the number of characters to move along, before grabbing the last character, which is obviously the number required.

From here, the temporary storage variable bufnew then has a "." (decimal place/full stop) placed at the end of it, signifying the break between the whole number just retrieved and the part values to follow. The variable bufnew then has the next block of numbers in the array appended to it apart from the final value, which of course is the whole number for the next number of DOP's, in this case it would be the whole number for the number HDOP's.

After this the numerical value currently stored in bufnew then has to be transformed from its current format as a string, into the required format as a double. Being a double allows the variable storing the value for either of the DOP numbers to be used in the calculations required in the production of this software.

The gathering of the values for HDOP and VDOP were done exactly the same way, with the same number of digits in each. The only difference being with each one is that number of the array being queried incremented by one in each case. This meant that the array of number blocks had moved through to array number 5 (*buf[4]*) after the manipulation of the data in the previous blocks to produce the values for each of the

DOP's. Each of the other DOP values were also transformed to the format of a double to assist in the calculations to come.

The program required the use of the *substr()* function to enable the sorting through the characters as with the previous QC1 characteristics which were obtained from the line of data. The value is then stored in the temporary string variable bufnew, from which it has to be converted from the format of a string to the format of an integer. As with the other numbers retrieved from the file, the format of the number is an integral part of it's formation by helping the ability to be used calculations. This helping the production of the final product, which are the summaries.

This then finishes the manipulations of the line of data which is found when a QC1 data code is found at its beginning. The retrieval of information from the lines of data allow for the simple calculations and viewing which will be later seen when entered into a Graphical User Interface. The extraction of the data from each line of QC1 data was also made simpler because of the consistency factor. Each line of QC1 data available to me on file from a VRS survey had the exact same format. By format it is meant that spacing and decimal places and character numbers were the same for each file viewed. Without this the task of extracting QC1 data from the file would have been a far more arduous task and one which would of most likely set the project back quite a deal of time.

3.5.2 Coordinate Data

While the program is searching through each line of the file searching for the QC1 data line code, it is also searching for the code which relates to GPS coordinate data. The code has a four character code at the beginning of its line.

Each line of coordinate data contains three very important pieces of information. These are the Easting, Northing and Reduced Level of the current GPS position. These three pieces of information differ slightly in the manner in which they have to be retrieved from the line of data, therefore requiring several methods of extracting coordinate data will be explained.

The code which I have discovered is that of ”08PD”. This code appears on the dc files which were exported with a SDR 33 format. This file was produced in the field using conventional RTK-GPS. None of the VRS dc files at my disposal were exported in the SDR 33 format. This left the assumption that these lines would be in the same format in a dc file from a RTK-VRS survey. This is a fair assumption because the lines of coordinate data which are in the standard dc file transfer do not differ between conventional RTK and RTK-VRS.

Once the four character code at the beginning of the line has been determined, the program then shifts to the class created to handle the lines of coordinate data. The program then moves through the line of data, tokenising the characters in the line as was done with the QC1 data. This being at every ”.” (decimal place) and at every ” ” (space). This left the data in blocks, these were stored in string arrays as was also the case with the QC1 data.

The first characteristic which had to be retrieved from the line of data was that of the easting of the point. The easting of a point is given in the 3rd number block in the array. This, as was the case with the operations in the QC1 data file, will be stored in a temporary string variable before its format is changed and it is stored permanently. So, after giving the temporary string the 3rd number block, the program then uses the *append()* function to add a decimal place behind it and then uses the *append()* function again to add the first 6 characters of the next number block after the decimal place. It only gathers the first 6 numbers by using the *substr()* function which was also outlined earlier. As the case was with the retrieved characteristics in from the lines of QC1 data, the easting value has to be converted from the format of a string to that of a double. The double is necessary because of the significant figures after the decimal place. Once converted, the easting value is then stored in its permanent variable location in the class designed to handle the lines of GPS coordinate data.

The characteristic next in line was that of the northing of the point. The extraction of the northing value of each point used the *substr()* and *append()* functions to create the correct number. The string variable bufnew was given the whole number of the longitude by grabbing 7 numbers from the 4th number block array. From here that whole number then had a decimal place appended to it, followed by the appending of the first 8 characters from the 5th number block array. As with every other previous characteristic, the number value had to be converted from its format as a string to that of a double for calculation purposes.

The final characteristic to be retrieved from the lines of coordinate data was that of the reduced level value of the point measured. Once again as with the previous manipulations of the line to extract data, the number given to represent the reduced level of the point needed to use the *substr()* and the *append()* functions. The height number was contained firstly within the 5th number block array and then moved to the 6th number block array. The *substr()* function was first used to move through the number block 8 places so that the next 3 numbers after that could be grabbed, i.e. the 9th and 10th characters in the number block. This of course was placed again in the temporary string variable bufnew. A decimal place was then appended to the end of the whole number value. From here the first 9 numbers of the 6th number block was appended to the figure in bufnew to form the number of significant figures after the decimal place. The reduced level value was converted from its format as a string to that of a double to assist in the calculations.

3.6 Data Processing

The aim within the unit of data Processing, or data computations, is to produce the desired outcome of a summary of the previously mentioned data. That data which is read from the file will then have to be run through a series of formulae. These formulae will help to produce some later explained output graphs and also provide the user with a simple summary report in which mostly averages of each individual characteristic will be shown. This output would then provide a historical log of how accurately RTK VRS performs under different circumstances with many variables in a particular location.

The accumulation and average of the DOP values was the first thing which needed to be calculated. These calculations were all performed back in the *main()* function which was explained earlier. The requirements for these calculations were that of having a double variable for each individual DOP each time one was read from the data class. As well there were double variables required to accumulate the values from each individual pass to the class, along with an integer variable to act as a counter within the *while()* loop.

The *while()* loop in C++ has the following form: *while(argument)* {}. The argument within the first set of brackets is generally in the form of a Boolean expression. Boolean expressions can range from: == (equal to), != (not equal to), < (less than), > (greater than), && (and), || (or), <= (less than or equal to) and finally >= (greater than or equal to). The while loop performs whatever operations are then placed within the curly brackets {} while the Boolean expression is true. To avoid the case of an infinite loop, a counter will generally be placed within the curly brackets to increment itself if it is placed within the Boolean expression.

The program has the expression *while(dcdata_list->next != NULL)* {} in the while loop, so as long as this expression was true the while loop continued. Within the while loop the functions created within the class used to handle the QC1 data were called again to move through the variables which were given the DOP values after they were manipulated in the raw file and transformed to their true form. This function would run through and grab each lines DOP values, i.e. the PDOP, HDOP and VDOP values from each line. Once this value was ascertained it was then added to the respective accumulating variable for each DOP. The counter was in place within the loop to count how many different lots of DOP values were received and added to the accumulator variable.

This was done until as the while loop I showed before stated, the next value was equal to NULL. The averages for each of the DOP values could then be calculated by incorporating another 3 variables to represent each DOP value. Each of these variables had to be created as doubles as stated earlier to help deal with the calculations. These new variables were given the value of the accumulated variable divided by the counter variable. This is obviously very simple mathematics, though it proves very effective in that in can handle in theory, hundreds of thousands of lines of QC1 data in the one file and output just three numbers to show an average of each of the DOP numbers. Having such a large file in conventional VRS RTK surveying for the user in the field is highly improbable and verging on never happening, but the ability is there for the rare cases. As is always the case, it is better to over design than to underestimate and run into trouble very early on.

The accumulation and averages of the MGA 94 Coordinates were the next processes to take place. The need was for the creation of the averages of the Easting, Northing and Reduced Level of the point. As has been stated earlier, the theory behind being able to use this as Quality Assurance/Quality Control tool is that the user will remain on the same mark, be it a known Permanent Survey Mark or simply just an unknown random mark anywhere within the VRS coverage region. This provides the opportunity to give the user a report, or feedback, on how well the VRS system is performing at that particular time, in that particular location.

With this in mind, the ability to report to the user the averages of the coordinates is very beneficial especially when considering its independent ability and assuring qualities.

The steps involved are very similar to those which were involved in the calculations of the DOP number averages. The first step was to ensure that double variables were created to temporarily store each of the 3 coordinate values as they were read in, and also for 3 accumulating variables to store the total of each of the coordinates which were read.

The difference between the workings of the *while()* loop in the calculations of the Coordinate data is that within the while loop the ability to compare each individual against the final average is required. This is needed so that the graphs of accuracy and precision can be created. To create the precision graph, a plot of each of the points residual is needed, and in order to have the ability to calculate that residual, each point must be compared to the final average. With the calculation of averages for the DOP values, once the individual value for each DOP was read in and added to the accumulating variable, it was then overwritten by the next respective value behind it.

To counteract the overwriting of the value, it was decided to store each value into a variable and store it permanently so that it could be used in comparison later. The simplest way to do this was by creating three arrays of double variables. These arrays of doubles could store a number for each coordinate value and then a counter within the while loop could then move each of them to their next memory address, i.e. move them up one position so as to not overwrite the previous value, thus allowing a mass storage of the coordinate values.

As the *while()* loop formation was stated previously, it can just be said that the argument was the same as for the QC1 data, except obviously this was from the class created to handle the lines of Coordinate data, not that of the QC1 data. Each coordinate value, as in each Easting, Northing and Reduced Level value were gathered from the class using a function previously stated. Once gathered and placed in their temporary storage variables, these were added to the double arrays and also to the accumulating variables. The counter would then increment one value to move the memory address forward one location in the arrays and also to keep a count on the number of different coordinate values passed in so as to help in the calculation of the average for each coordinate. Once the next value was equal to NULL the while loop ceased and the calculation of averages and residuals could commence.

The averages were a very simple mathematical calculation as it was for the DOP values. Each Coordinate, those being Easting, Northing and Reduced Level were given a double variable to use as the storage point for the averages. Input to these variables were, the accumulating variable divided by the counter, which incidentally had to be converted to a double from an integer to allow calculations. This also occurred in the previous DOP average calculations.

After the averages were calculated, the residuals for each coordinate had to be calculated. For this task it was required to run the array variables through a *for()* loop so that each of the memory addresses could be accessed. The variable used in the arguments for the *for()* loop was that of the counter. Within the for loop the arrays which stored each individual coordinate value for all 3 of the Easting, Northing and Reduced Level numbers had to be subtracted by the average. This number was stored in a separate double array so that the residuals could be kept and not overwritten as was the case with each individual coordinate value. The array would then move to its next position so that the next individual coordinate vale for all 3 could be calculated, and so on.

The values within these double array variables which stored the residuals will then be used in the future for the creation of the precision graphs for each point which is surveyed, be it known or unknown.

As has been shown in the previous paragraphs, the mathematical calculations are very simplistic when dealing with creating summaries for VRS RTK surveying. This is because of the need for only averages and residuals, and these are all that are required when a surveyor is trying to deal with large amounts of data. All the surveyor requires is another Quality Assurance check and this is what this software provides. The difficult processes are quite easy to see in that the manipulation of the data from the file is a very exact process and cannot be out by one character, otherwise the data becomes completely negligible.

3.7 Graphical User Interface Creation

The Graphical User Interface for this project is something which was required to be simple, yet very effective in delivering every required piece of characteristic information to the user. A graphical user interface (or GUI) is a method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text. (*Graphical User Interface - Wikipedia, the free encyclopedia* 2005)

So, in short, the GUI that was created for this project is a very simple design which shows the user all of the required information. It shows the user the summaries and at the click of a button the user can have the graphs pop up in a similar sized window.

The design for the GUI took a couple of different forms before deciding to stick with the layout which will be discussed. It was recognised early on that the user required the ability to both load a file into the interface, and also to save the summary which was created by loading a file. This saving ability was particularly necessary for its help in creating a log of VRS performance and also providing documentation for Quality Assurance purposes. The format of the program is that the user will just need to double click on the exe file (executable file) to open the software. This will just pop up the GUI in a conventional windows format window.

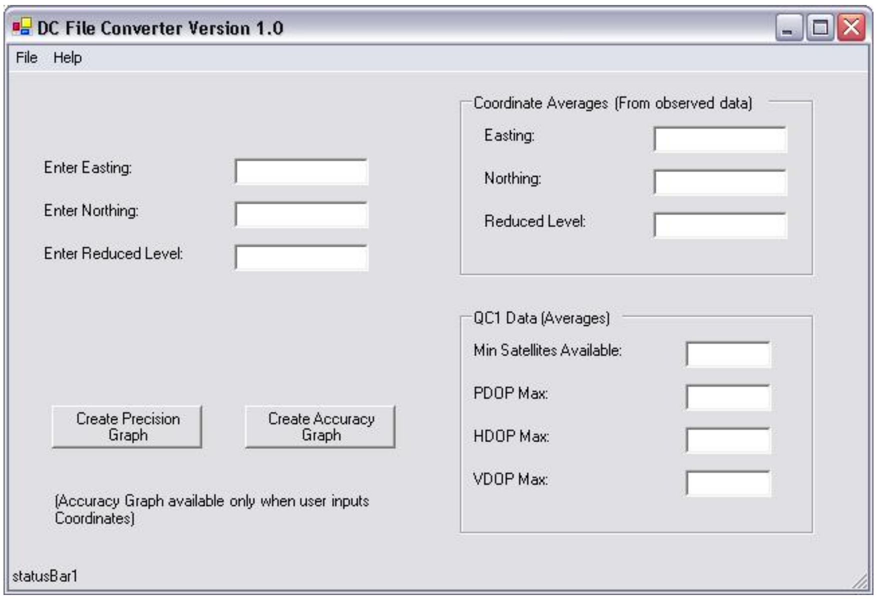


Figure 3.2: The main screen which is shown after the execution of the software

Figure wotever shows the form of the software once the exe file is run. This is the GUI which is being discussed. As can be seen, the QC1 data and coordinate data which was calculated in the previous section are shown to the user in a simple format, giving the surveyor in the field the ability to make quick, decisive decisions on whether or not the survey should continue, or if the accuracy is within the standards required.

The purpose of having such a simple interface for the user to look at and read through is that although looking pretty does help, it's effectiveness in loading quickly and showing clearly the summaries are far more valuable to a Surveyor.

The decision to group the QC1 data to one side was quite clear in its intentions and needs very little explanation. The need to show these QC1 data summaries together is for simply the ease of reading through them, showing arguably the important information in order of priority from top to bottom. The area on the left hand side of the form shows where the user can enter the coordinates of the point in which they are gathering their test data on. This can only be done if the user is performing the VRS RTK survey on an established/known mark. This would generally be in the form of a Permanent Survey Mark.

If the user does not enter into the software the coordinates of their mark, then it is assumed that they are on a random point somewhere within the VRS system coverage region. Without these coordinates, it is then impossible for the user to create an accuracy graph, though a precision graph can be created regardless of the situation. These two graphs are created by simply clicking on the buttons which are shown in the bottom left hand corner of the software layout. The placement of these buttons was just as simple as the text boxes with the ability to simply drag and drop them where it was deemed necessary.

The only other obvious characteristics of the GUI is that of the drop down menu in the top left corner. The File menu allows the user to load the file which they want a summary created for, which is what will happen every time that user executes the software. The other options are to save the summary which is currently being shown on the interface, and the last option is that of exiting the program. The drop down Help menu which is located beside the File menu will has a very simple set of instructions in regards to what must be done to ensure that the program runs smoothly and also some user requirements.

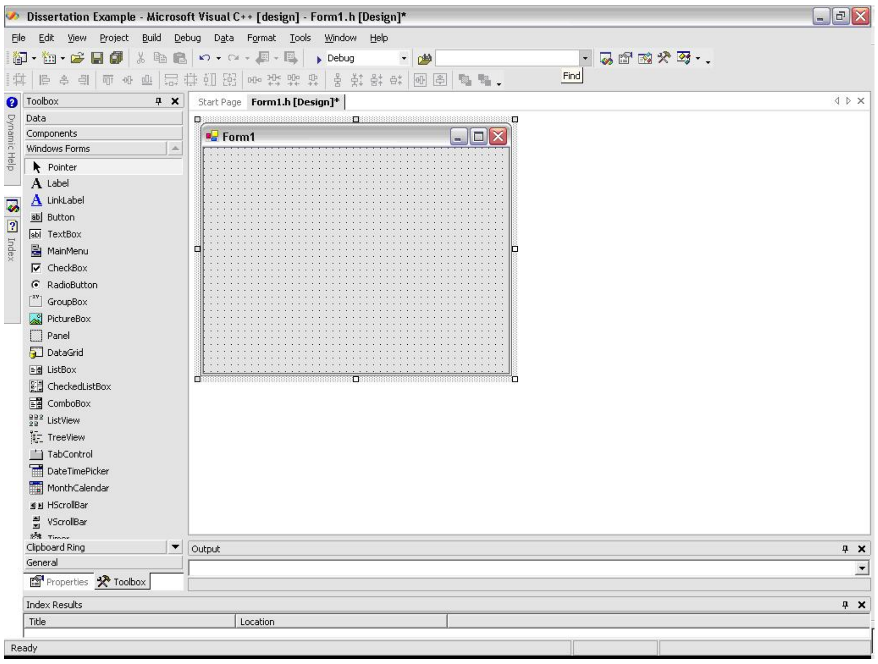


Figure 3.3: The design environment of the Windows Form Application

A GUI like the one in this project is very easy to design in MVS03. Firstly the creation of a new project must be that of a Windows Form Application. This form starts blank and can be expanded to a full size window dependant on the information required to be shown. Text boxes are simply added in the places where information will be loaded to once the programs code is integrated with the GUI. These boxes can then be given names by adding Labels to the form. These can be all grouped as the QC1 data is with a simple Group Box from the Toolbox area in the design environment. The Toolbox is where all of the textboxes, labels and buttons are taken from and generally just dragged to the windows form design window wherever they are deemed necessary. The design environment of the Windows Form Application is shown in Fig wotever. This further shows the nature of dragging the required boxes from the Toolbox on the left to the form on the right.

These steps outline how the creation of the GUI came about and how simple the steps were to create a basic design like the one made for the purposes of this project. This also leads into the next subchapter in that the program code which was previously discussed must be integrated into the GUI created in this project to enable the software to work fully on its own. This is by far a more difficult step than the design of the GUI. Once again, this is quite clearly not the most demanding looking GUI, design wise, but its effectiveness in showing the required output summaries is second to none. The notion was taken in designing this that the less superfluous information on the GUI, the less that could go wrong.

3.8 Integration of Program Code to GUI

The integration of the programs code to the GUI was done by adding the files created earlier into the project which had made the GUI. Copying the cpp files which store all the code, and the h (header) files which govern how the cpp files operate, left the GUI with the ability to handle any variables which had been created within the previous work.

The first function to be operated was that of opening a dc file. Once the load file function was opened, it was only a matter of using standard visual C++ functions to get the operation of clicking on the load menu to pop up a window allowing the user to pick a file from their hard drive. As shown in Figure 3.4.

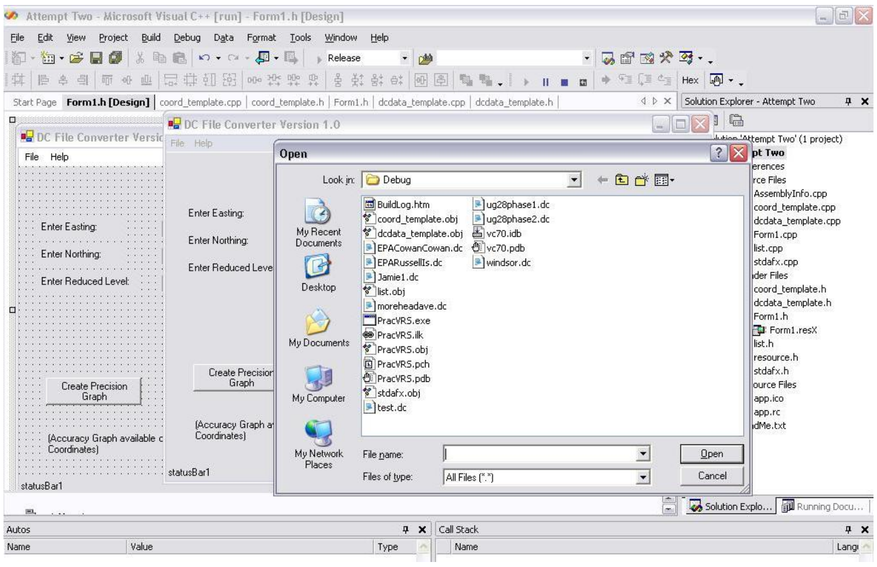


Figure 3.4: The Load dc file window

The status bar at the bottom of the GUI was given a getfilename function. This function let the name of the file being loaded appear in this section, reminding the user of which dc file they are viewing the summary of.

The save function allowed a similar process to take place, with the user having the ability to save the summary to their hard drive. The exit function consisted only of exit(1);, this left the program to exit upon execution of the exit option in the main menu.

The next operation to take place was that of printing the previously created averages to the GUI upon the choice of a file by the user. This yet again involved a simple solution with only the following required:

```
sprintf(sz, "%f", pavg);
textBox5->Text = sz;
```

The only thing to change between variables was the name where pavg is, and the textbox number. The operation to accept the users input of the coordinate values was just an exercise in assigning a variable that particular textbox, and using that value to compare against each individual coordinate value to help the creation of the accuracy graph.

The help menu was able to have a file addressed to it so that whenever it was clicked, a file with a set of parameters is popped up to remind the user of their requirements both in the field, and when using the software.

The last piece of the integration plan was to have the graphs pop up when either one of the buttons were selected. This was the most difficult part of completing the software, with requirements to set up pen colours, sizes and where the x and y axis began and finished. Upon the clicking of either of the Precision graph or Accuracy graph buttons, the functions within these were to allow a window to pop up and show the user a graph of residuals formed from calculations previously addressed. Problems occurred with the integration of the graphing funcitons to the GUI. They did not work correctly once integrated, and due to the time frame of this project, the submitted copy of this software could not show the graphs working correctly. Though this problem can be fixed and the software will be ready in time for final submission to the DNRM.

3.9 Software Validation

Ideally, a process of validation was to be done after the software was completed and was running bug free on some test .dc files. The validation process was designed to be identical to the situations which will arise for the end user in the field when this project is complete. The testing for this software was to take place in an area which is still to be decided by the DNRM, but it is known that it will be relatively central in regards to the area covered by the VRS setup.

This was the perfect world situation for this project. But as time was at a minimum toward the end, it was decided that for all intents and purposes the validation of the software would be more than sufficient by just testing the running of the program on the dc files which were provided from the DNRM.

The single requirements of the dc files which I obtained from the DNRM, was that they be from a VRS RTK survey, and not from a conventional RTK survey. There is very little difference between the two, as will be spoken about in the next chapter, but the fact that there is a difference made it very difficult to provide a piece of software which could deal with both, so for this project, conventional RTK dc files were not required. It must be stated however, that with some slight variations in the programs code, this software could quite easily deal with conventional RTK dc files.

Three VRS dc files were received from the DNRM to test the software on. These files obviously also assisted in the creation of the program, because without them there would have been no standards in which to measure against. These standards I am referring to are those of finding out the spacing and format of data in each line so as to construct the methods required in the earlier discussion of the manipulations which took place. So, in essence, these files were both the precedents and the validation files. This mattered little as the files showed that they have the same format throughout.

3.10 Summary: Chapter 3

This chapter discussed the experimental methods and design techniques which were used in establishing the software which will be used as a Quality Assurance tool for VRS.

Following this chapter, it is envisaged that the methods and details that have been discussed will allow another person to recreate the process' that were involved in developing this piece of software and how it was validated. Though it also must be concluded that besides reading through these steps, it would be well in the following students/users interest to seek references from Appendix B so as to guide them along the way. All of the files which were used within the MSV03 programming environment are repeated here and will provide a very beneficial tool to anybody looking to improve and upgrade the software.

Now the outline of the design and steps taken to write the programs code has been completed, the actual functionality of the software, its possible faults, problems involved and final product can be discussed and evaluated in Chapter 4, this will show the final GUI which has been designed and prepare the user on how to use the software.

Chapter 4

The QA System

4.1 Introduction

This chapter will discuss the attributes and workings of the Quality Assurance software which has been developed in this project. The user of course needs to be instructed in how to use the software and must be shown any limitations and be given troubleshooting tips when problems arise.

The chapter aims to provide the eventual users of this program an insight into what they can expect from the software. The user should, after reading this chapter, be able to take away the executable file created which runs this software, and be able to use it and solve most issues which may occur along the way.

The user/reader will be shown what each result in the output summary means. This will cover information on all of the QC1 data averages, coordinate information and the graphs which the user will have the ability to execute. The method in loading files and saving summaries will be covered so the user has the ability to start keeping logs/records of the VRS system performance in each survey that they perform.

4.2 Revision of GUI Layout and Characteristic Definitions

It is important that the user be familiarised with what the GUI of the software will be showing them before speaking about what each number represents and this can be used in the survey that they are performing.

The GUI for this software is only very basic, with it only producing the pop-up window which is started up execution of the file. From here the only other windows to be shown are that for loading, saving, help and the graphs. Once again, though simple, it provides the user with all the information required and is therefore very effective.

The layout, as shown earlier in Fig.wotever, can be referenced to identify with the following descriptions. The information in the group box on the right hand side of the window consists entirely of QC1 data. The QC1 data is Quality Control data which the user can reference in the future as a measure of how well the VRS system was performing in the particular area of the survey at that particular date and time.

These QC1 summaries are the backbone of the software in regards to Quality Assurance and Quality Control. These summaries will be able to provide the user with leg to stand on as far as performing a survey to the best of their abilities and as best as the conditions will let them. If for example, any of the DOP averages are too far out of range, or the minimum satellite average is too low, then the user can postpone their survey on the basis of the softwares outputs.

In the future this software will hopefully be recognised to be sufficient enough to allow the surveyor to claim that VRS system was either good enough, or the quality was not good enough, for them to use this as a basis for evidence in a court of law if required. It is realised that after only just being developed and not tested for long enough, that this is an unrealistic proposition at present, but definitely a hope for the future of the project and the software.

In the top left hand corner of the GUI, the information in regards to known point coordinates is located. As has been previously stated, the user may only enter the coordinates of the point that they are surveying on if that point is a recognised known mark. If the mark is unrecognised/random, then the user must leave these boxes blank as they will only affect the integrity of the accuracy graph if they are entered. These coordinates must be entered as MGA 94 Coordinates, this is because of the way that they are measured within the VRS dc files. So, in short, the user mustn't enter any coordinates into these boxes unless they are those of the known point in which they are performing data collection on.

In the middle of the GUI the user is given an average of the coordinates for the point in which they are observing their data. These averages are of the observations which they have performed, and are the number which the residuals will come from in the precision graph.

In the bottom left hand corner of the interface, the user has the ability to create a precision graph, accuracy graph, or both. This is all dependant upon whether the data collection is over a mark which the surveyor knows the coordinates of. The creation of the precision graph is one which can be done for either case. This is so because the precision graph will show the residuals of how far each and every point read is away from the calculated average of the coordinates. This calculated average simply came from the addition of every set of coordinate over the mark being added and divided by the number of sets. This precision graph is very useful in more showing the consistency of the VRS system, rather than its overall effectiveness in tying in with coordinate data from previous surveys.

The creation of the accuracy graph, as has already been stated, may only occur when the user has entered the coordinates of the point in which they are collecting the data from. For this reason, it is assumed that this graph will not be created as often as the precision graph. The accuracy graph will plot the residuals from each set of coordinates read over the known mark against those coordinates which the user has entered in the software window. The purpose of the accuracy graph is to give an indication to the user of how well the VRS system is performing in comparison with previously recorded and reported survey data. These residuals will give the user an idea of how far different the VRS system is to that of the National Geodetic Network. That is assuming there is a difference.

Both of these graphs provide the user with yet another log of how the VRS system performs in different areas at different times. The precision graph can be logged to keep a record of the consistency of VRS in many different locations all over the coverage region. The accuracy graph can therefore help to keep a log of any differences from previously surveyed marks and from the National Geodetic Network. These logs of the accuracy graphs will help in particular to find any bias that there might be between the VRS system and the National Geodetic Network by logging data over some known marks covered by both of these networks. If any trends are found, then this will lend itself to help in perhaps calculating a block shift, or just identifying users of the trend in differences between the two.

The other functions which the software provides are those of being able to save the current summary to a file, as well of course, as being able to load a file into it. A small help section is also available, more to clarify any queries the user may have than being an extensive help system.

With the characteristics of the software now identified, after reading through this chapter it should be possible for any user of the VRS system in South East Queensland to obtain a copy of the software and understand its outputs and the effect their values would have on their survey.

4.3 Survey and User Requirements

This section of the chapter is to clarify exactly what the user needs to do in regards to their performance in the field and also some minor things which may come about in using the software.

When performing their VRS-RTK survey in the field, there are very few requirements asked of the user to ensure that the software is given its highest chance to perform optimally. Before beginning the survey, the user should turn on in the Trimble Data Recorder the ability to record QC1 data, as well as recording their coordinates in the Easting, Northing, Reduced Level format of MGA 94 coordinates. Beside this, the only other requirement which would be advisable to meet is that of the user keeping their rover over the same mark/point for anywhere between 10-50 epochs of data. This size of data allows some trends to come out in the results. Though not desirable in size for the data analysis purist, up to 50 epochs of data will begin to give a very good indication of how the VRS system will be performing on that day in that location. The software is designed after all, to benefit the user by allowing them perform the checks relatively quickly if they have a laptop in the field, and if the software is installed in the Trimble Data Recorder, even more so.

There should be very few issues for the user when it comes to loading their file into the software. The user must first transfer their dc as normal, then after this has occurred, the user is then asked to transfer the file in the format of SDR 33 dc file. The dc file needs to be loaded in the SDR 33 format so that the MGA 94 coordinates can be loaded from this. The normal format of the dc file when transferred does not contain these coordinates, and therefore this second format is a necessity.

Before the user can begin using the software, they first must have the Microsoft .NET Framework installed. This is a 23mb installation package, it contains Microsoft.Net runtime libraries and it is a necessary requirement to run an executable (exe) file which has been created in the Microsoft Visual Studio.NET environment. (*.NET Framework Developer Center* 2005)

The software should be first run by double-clicking on the VRSQA.exe. Once the window pops up, the user may then load their file into the software by clicking on the File drop down menu in the top left hand corner of the window, before scrolling down to the load button. Once loaded, the user may then enter the coordinates of the mark in which they captured their data on if it was a known Permanent Survey Mark, otherwise the user may then click on either of the graph buttons to execute the creation of either only the precision graph, or if over a known mark, then the accuracy graph as well.

The user may also get a reminder of the fundamentals of using the software by simply clicking on the Help drop down menu bar in the top left hand corner of the window. To exit the software, the user may simply click on the File drop down menu, and then scroll down to the Exit button.

These descriptions combined with instructions which will be given to each of the users of the software, should be more than sufficient to ensure that they can process their data quickly and easily.

4.4 Disclaimer

This disclaimer is to ensure that the users of this software realise that it has been created by a USQ student for the purposes of the Department of Natural Resources and Mines. USQ in no way hold responsibility for any shortcomings or problems which may arise from the software. The purpose and workings of this software are spelled out quite clearly in this dissertation, leaving no stone unturned means there will be no unexpected surprises from using the software.

The software is also not guaranteed to be 100% correct, 100% of the time, so any errors which come about from the software are the responsibility of the user. This software is not recognised as a device which could hold a person liable, or used as a defence in a court of law, and should therefore not attempted to be.

4.5 Summary: Chapter 4

This chapter discussed the attributes and processes of the Quality Assurance software which has been developed in this project in accordance to the requirements of the Department of Natural Resources and Mines. The user is now instructed in how to use the software and has been shown the limitations and given warnings on it also.

Following this chapter, it is envisaged that the user, once given a copy of the software, should now be able to go out into the field, perform their required VRS-RTK survey over a known or unknown mark and bring it back to the computer and download it. Once downloaded, the user should then be able to open the software and load their file, creating their summary of VRS performance in their area of survey at the time in which they performed it.

Now that the description of how the software works, and steps on how to implement its capabilities have been discussed, the next chapter will look to describe how the software was validated by running through the steps on how the test VRS dc files were broken down to find the location of their characteristics and how these were confirmed.

Chapter 5

Validation

5.1 Introduction

This chapter will show how the information in the dc files was found when opened with a text file editor. The comparisons will be made between the image of a dc file opened with a dc file editor and then with a text file editor, showing the validation of the software.

This chapter aims to prove that this software was created using the correct formats and that it has the ability to correctly handle VRS dc files. The validation process will be explained thoroughly.

The chapter will show how the validation of the characteristics within the dc files was obtained, as well as explaining the differences between files.

5.2 Validation of dc Files

The problem with viewing a dc file in a text file editor is that it loses any legibility it had. This section will look at how this problem was overcome and how the lines of QC1 data and Coordinate data were deciphered.

The issue which first arose was that when programming in C++, the way the program which is being created will deal with a file, is to look at it as if it were being opened as a text file. This led to some problems when trying to establish where each attribute lay in the file.

This was overcome by viewing the file in its recommended dc file editor (a subsidiary of TGO) and printing out the output from these files. Once this was completed, the printed files were then compared to the same files viewed with a text file editor. The comparison of these files took a great deal of time, because the difference between the two is quite astounding.

After some further investigation, it was realised that each new line (besides those which were wrapped around) had a unique four character code at its beginning. These codes differed greatly, but after establishing in one of the earlier standard RTK dc files which were available before the VRS dc files that the position of the QC1 data was in the first row in the dc file editor, the four character code for that first line could be matched up in the file when opened in the text file. After checking the numbers against each other, it was found that "C6NM" was the code for a line of QC1 data in the text file editor. This process continued throughout the file and data proved to be very consistent, which was the basis around this whole software development. Without consistency in the files, it would be impossible to tell which lines of data were QC1, and which were coordinate data without analysing every single character in their line, rather than simply analysing the four character code at its beginning.

When several VRS-RTK dc files became available, it was with great relief to find that they too had the same code as the standard RTK produced dc file. The only difference between the two as far as lines of QC1 data was concerned is that of the difference in buffers and spacing between characters on the line, though this was only slight and required very little adjustment between the two.

The process to establish the position of the lines of Coordinate data within the dc files was identical to the exploration which occurred for the QC1 data lines. After checking the position of the lines of coordinate data, these were then checked to find a similar situation with the four character code.

The methods of extraction, and functions required for the manipulation of the QC1 and Coordinate data was detailed in Chapter 3. These manipulations extracted what looked like several big blocks of numbers, and made them appear as they would when opened with the dc file editor. This allowed the computations to take place, and therefore allowed the program to produce its summaries which provide the basis of the Quality Assurance to the Surveyor.

5.3 Summary: Chapter 5

This chapter explained the comparisons with the dc files as viewed by a dc file editor and by a text file editor. This gave an insight into the difficulties associated with dealing with dc file opened in a text file editor.

Following this chapter it is hoped that there is a greater understanding of the working of a dc file and the layout in which it comes. The purpose of this being that the user may understand why some of the parameters are put into place, hopefully ensuring that they abide by each of these when using the software.

Now that the validation of the software has been described, the next chapter will move into some discussions on the software and recommendations for the future of QA for VRS.

Chapter 6

Discussion and Recommendations

6.1 Introduction

This chapter will discuss some of the issues which may arise from the user about what may go wrong with the software, and also some of the possibilities of further developing the software. The recommendations section will look to the future, in regards to further research on this particular topic and the future projects that this particular study may bring about.

The aim of the chapter is to inform about some of the possible future endeavours which are now available to students as a result of this project. As well as trying to ensure that most issues users may have with the software will be answered.

A discussion on errors which may occur, how to fix some of these and some attributes which could ideally be added to this software in the future will be mentioned. The topics for future student projects are to be shown as well as the benefit each of these will bring to the DNRM.

6.2 Discussion

This project was designed with idea of having a minimalist approach. This stemmed from the manner in which the code was manipulated once gathered from the file, to the design of the GUI and its functionality. This approach was adopted, for starters, with the short length of time available for this project in mind, as well as having something which could be looked upon as more efficient and effective than pretty.

The design criteria also allowed the user to be able to understand all the operations of the software, and have something which was conducive to encouraging the surveying population to embrace it. Its simple, yet effective GUI is something which could be made to be more aesthetically pleasing in the future, but for the moment, and for the purpose of this project, it does the job.

The ability to keep an independent log/record of VRS performance is encouraging for the user. With its dual purposes of providing Quality Assurance for the independent users as well as slowly increasing their trust in the system and enhancing the likelihood of them embracing the technology, and turning to it more often as their first option, providing of course they are within the coverage region.

The software itself should provide very few problems to its users. The only operations the user needs to perform are those of loading a dc file into the interface, as well as saving the summary and entering the coordinates of the known mark they are/were positioned on, if that were the circumstance. If problems occurred with the summaries, or numbers appeared which seemed to be well outside of what would normally be expected, then the user may want to go the methods in which they collected the data and make sure that the correct survey style was chosen and that they had selected QC1 data to be recorded.

These are the only known/recognisable problems which may occur with the software. Other issues may require programming fixes or a reload of the software. If it is programming issues, then the user may in fact want to revisit the survey again, and ensure that, as mentioned earlier, the correct style was chosen.

The user should be able to load the software and .NET framework required to run it without any problems. It is also recommended that the user ensure they keep a save of the summaries in a folder common to their jobs, so that reference can be made to previous surveys and their accuracies. This will be particularly relevant dependant almost wholly on location.

6.3 Recommendations

This project had a responsibility to deliver a product which not only gave the user software which provides a piece of Quality Assurance, but also provides students in the coming years a basis on which to begin other projects. Thus, further developing more Quality Assurance and getting to better understand the way in which a VRS system performs under a multitude of conditions and over its whole coverage region.

The future for projects involving VRS are very bright, with this year's projects only skimming the surface in regards to analysing the performance and quality of results gathered from the system. As far as Quality Assurance is concerned, it is hoped that this project will provide an incentive for students in the future to continue on with the work which has been started.

One possible development which is hoped will occur in the next year or so, is that of using the software which has been developed in this project to gain an idea of the performance of the VRS system over the SEQ coverage region. This would involve a student performing an RTK-VRS survey for approximately 60 - 100 epochs of data over a known survey mark in the coverage region. From here the student would run the dc file created in the survey through the software from this project, always keeping a log of their data.

It is envisaged that these logs of data could be used in conjunction with a GIS of some description, helping to show an overlayed map of accuracies across the whole region. This map could show these accuracies with different colour shading to particular regions to represent the specific accuracy of that area.

This particular map would be a very useful tool for the DNRM. It is recommended that once this project is completed, that particular accuracy map could be issued to users of the VRS system once they register their intent to use it with the DNRM. A package handed to users could include the accuracy map, as well as still providing them with the software created in this project for their own independent checking. Thus still providing another QA tool if the user so desires to use it.

The package could be distributed on a compact disc (CD). Having a digital copy of the accuracy map just allows users to print or view this at their own will. It is still not established whether the DNRM will establish a small fee for the CD, or whether it will be bundled in the information given once the user registers for use of the VRS setup.

6.3.1 Enhancements to the Program

Several issues arose with the time constraint of this project with three of the characteristics from QC1 data which were intended to be shown. These characteristics constantly proved a bane of the softwares' existence by providing continuous errors in the development of the program. These were the average initialisation times, the total number of GPS positions captured in the survey, and the avg number of satellites. These will be made ready for the version of the software to be given to the DNRM, but unfortunately the time requirement could not be met for this dissertation.

A final problem which occured on the day before printing this project, was that of the graph functions not integrating into the GUI correctly. These graph functions were created separately and when supplied with fake data, produced the expected residual plot. However upon integration, errors occured, and due to the time at which these were integrated, a solution could not be immediately found. The problem is known, and will be fixed so that the correct integration can occur in the copy given to the DNRM. This is unfortunate as it takes away from the final product handed in with this dissertation. Though the main point is that the DNRM will still be receiving their QA software which works correctly and analyses the coordinate and QC1 data as specified throughout this text.

These would be very advantageous in a future version of the software, but without them, the software still provides enough Quality Control data to notify the user of the potential integrity of the survey they are about to undertake. Another potential improvement would be to be able to create perhaps some scatter plots, showing the known mark with each individual set of coordinates captured shown around it.

6.4 Summary: Chapter 6

This chapter discussed some of the potential shortcomings of the software, and how to deal with these. The benefits of the design of the software were mentioned, with the reasons for these clearly stated. Some recommendations for potential projects to come from this project were made, as well as providing ideas on how the software should be managed and distributed.

Following this chapter it is envisaged that some future project ideas will have been established, along with the DNRM having ideas on how the software will be packaged and distributed. The user should now be aware of the problems which may occur and any questions which they may have had should be answered. This discussion should help the user realise that there are no hidden bugs or problems and hopefully endear them to the software and its potential benefits.

Now that the discussion on some possible Frequently Asked Questions has occurred, and the recommendations for the future of QA involving VRS has been stated, the software should now be potentially implemented by the user and encouraged to be used by the DNRM. The next chapter will forward in summarising the main points from this project, and will establish whether the aims and objectives have been successfully completed along with the legacy that this project will leave.

Chapter 7

Conclusion

7.1 Introduction

This chapter will draw together and provide summaries of the information which has been gathered in this project. It will also draw some conclusions from the results of the work which has been undertaken.

The aim of the chapter is to ensure that correct final message is taken away. Summaries will be drawn from each chapter to reiterate the main points of this project as well its benefits to all parties concerned.

The chapter will step through the aims and objectives mentioned at the start of the project and discuss the success or failure of each of these. The enhancements which this software will bring to the QA of VRS will also be discussed, helping to further enhance the justification behind the conclusions which will be drawn in this chapter.

7.2 Conclusions

The aim of this research project was to investigate and validate a system of monitoring and assessing the performance of VRS-RTK. A system was to be developed that allowed the VRS to be monitored and assessed with a view to outputting a summary of characteristics of performance specific to a particular geographic location at a particular time. It can be stated that this aim was achieved in full, though as with most other things, there is room for improvement. These suggestions were made in the discussion chapter and won't be noted again here.

Chapter 1 also gave an outline as to what VRS-RTK surveying is, and how it works. The scope and justification for this project were also discussed, showing a clear reason for the necessity of this research.

1. Research the background information regarding current QA systems for VRS and current software which manipulates survey data controller files.

2. Critically evaluate existing alternatives for QA at the rover end of RTK surveying using VRS.

The first two objectives, as stated above, were successfully met throughout the course of Chapter 2. Chapter 2 reviewed literature showing the overviews and accuracies of other global VRS operations, as well as their QA procedures within the Server, Bases, and Rover. This review further enhanced the importance of this project by showing very limited QA operations at the Rover. The only QA were those characteristics which are updated in real-time on the survey data controller.

The review also covered relevant literature in regards to any software which may manipulate survey data controller files, be it in this case the Trimble dc file. The software reviewed showed the characteristics which are regarded as most important, and which have the greatest significance in regards to showing their influence on the final accuracy of a survey. These findings helped lead into the completion of the design, and hence also, the next objective.

3. Design a piece of software which will extract data from a controller file and output a summary of desired statistics.

The information gathered from the literature review was imperative in helping to narrow the field of vision in the designing of this piece of software. The literature review helped recognise the important characteristics which could be stored within a dc file. These characteristics were then used in the design of the software, allowing only the characteristics whose data affected the accuracy with their fluctuations to be included in the final output summaries. The programming language and environment was also decided in the literature review.

The logical steps in the design have all been recorded with the files for the software being included in the submitted CD. Thus allowing a future student to go back over the work conducted in this project and see how it was completed, and make further changes if so required. It is quite clear from the design methods in Chapter 3, and the explanation of the softwares capabilities in Chapter 4, that objective 3 was partially met in this research project. The problem being with the integration of graphs. Though these will be corrected in the copy submitted to the DNRM, it is dissapointing that they were not ready in time for the final submission of this project.

4. Test (validate) the designed software by conducting trials which will gather data and download to a pc which will run the software and produce summary reports.

This objective had some changes as it was found out when it came to validation time, that the effectiveness of running a previously acquired RTK-VRS survey's dc file was the same as that of physically being in the test area and acquiring the file in real-time. From this knowledge the testing was completed successfully with the provided dc files. The testing on the graphs was done individually before the attempted integration, and all of these worked correctly. Manipulating of the files also provided the desired outcome, proving that software worked as stated, with it, completing objective number 4.

5. Analyse results to prove the effectiveness of the designed software.

Analysis of the results from running the dc files through the software was also apart of the validation step. The analysis further proved the effectiveness of the software and showed that it met all design specifications and will provide a beneficial tool for the DRNM. This meaning that the final objective was also met successfully.

7.3 Close

Following the successful completion of the aim and objectives, this project will provide an invaluable tool for the Department of Natural Resources and Mines to be used in association with the new safe guards built in to VRS to constantly monitor the system.

The final outcome to be taken away from this project is most importantly the new software created. This software solves a significant problem for the DNRM and provides another basis of Quality Assurance in their task of ensuring high quality data is always provided to the end user.

References

About GPSnet.dk (2005), Trimble Center Danmark.

`http://www.gpsnet.dk/omgpsnet_int.php`

current October 2005.

C++ tutorial: 6.1,Input/Output with files (2005), The C++ Resources Network,2000.

`http://www.cplusplus.com/doc/tutorial/tut6-1.html`

current October 2005.

eGPS Coverage Area (2005), eGPS Solutions USA.

`http://www.egps.net/aboutus.htm`

current May 2005.

eGPS Data Reliability (2005), eGPS Solutions USA.

`http://www.egps.net/data.htm`

current October 2005.

eGPS Solutions Internet-Based Real Time GPS Network (2005), eGPS Solutions USA.

`http://www.egps.net/index.htm`

current May 2005.

GPSnet Coverage Area Victoria (2005), Land Victoria.

`http://www.land.vic.gov.au/land/lcnlc2.nsf/LinkView/`

`C22C2EBAF2DC90F34A256A250007E33C0887F847B6D2DA16CA256E5F0013CBB8`

current April 2005.

GPSnet.dk Coverage Map Denmark (2005), Trimble Center Danmark.

`http://www.gpsnet.dk/daekning_int.php`

current October 2005.

Graphical User Interface - Wikipedia, the free encyclopedia (2005), Wikimedia.
http://en.wikipedia.org/wiki/Graphical_User_Interface
current October 2005.

Java examples (example source code) File Input, Output Data, Input Output, Data IO Test 2 (2005), Java Source and Support.
http://www.java2s.com/ExampleCode/File-Input-Output/DataIOTest2.htm
current October 2005.

Land Victoria Home Page (2005), Land Victoria.
http://www.land.vic.gov.au/land/lcnlc2.nsf/Home+Page/Land+Channel~Home+Page
current April 2005.

.NET Framework Developer Center (2005), Microsoft Corporation.
http://msdn.microsoft.com/netframework/
current October 2005.

Real Time Kinematic - Wikipedia, the free encyclopedia (2005), Wikimedia.
http://en.wikipedia.org/wiki/Real_Time_Kinematic
current October 2005.

The idea behind GPSnet.dk is simple - VRS is ingenious (2005), Trimble Center Danmark.
http://www.gpsnet.dk/VRSteknik_int.php
current October 2005.

Trimble - GPS Glossary (2005), Trimble Navigation Limited.
http://www.trimble.com/gps/glossary.html
current August 2005.

Trimble Home (2005), Trimble Navigation Limited.
http://www.trimble.com/
current October 2005.

Trimble Total Control Software (2005), Trimble Navigation.
http://trl.trimble.com/docushare/dsweb/Get/Document-10049/

`12631B_Trimble_Total_%20Control_TN_0305_lr.pdf`

current May 2005.

Tutorials From FunctionX (2005), Function X.

`http://www.functionx.com/`

current September 2005.

Appendix A

Project Specification

University of Southern Queensland
Faculty of Engineering and Surveying

ENG 4111/4112 Research Project
PROJECT SPECIFICATION

FOR:	**Jamie Robert HEIT**
TOPIC:	Quality Assurance for Virtual Reference Stations
SUPERVISOR:	Peter Gibbings
SPONSORSHIP:	Department of Natural Resources and Mines

PROJECT AIM: This project will investigate and validate (by conducting a trial) a system of monitoring and assessing the performance of RTK VRS in specific locations. The data collected will be monitored and assessed with a view to outputting a summary of characteristics of performance to that particular area.

PROGRAMME: **Issue C, 22 October 2005**

1. Research the background information regarding current QA systems for VRS and current software which manipulates survey data controller files.

2. Critically evaluate existing alternatives for QA at the rover end of RTK surveying using VRS.

3. Design a piece of software which will extract data from a controller file and output a summary of desired statistics.

4. Test (validate) the designed software by conducting trials which will gather data and download to a pc which will run the software and produce summary reports.

5. Analyse results to prove the effectiveness of the designed software.

AGREED:______________________(Student) _____________________(Supervisor)

(Dated) ____/ ____/ ____

Appendix B

Source Code

B.1 The `DC File Converter` C++ Function

Listing B.1: dcdatatemplate.h.

```
#include <string>
#include <iostream>
#include <fstream>

using namespace std;

class dcdt {
        private:
                string code;
                int numsats, total_gps_pos, gps_start_week, gps_start_time, gps_end_week, gps_end_time;
                double p_dop, h_dop, v_dop;
                //double latitude, longitude, height;
                void eval(char *currentLine);

        public:
                bool read_data(char *buf);
                int get_num_sats(void);
                string get_qc1_code(void);
                void set_qc1_code(string theCode);
                void show_details(void);

                void get_dops(double &pdop, double &hdop, double &vdop);
};
```

Listing B.2: coordtemplate.h.

```
#ifndef _COORD_
#define _COORD_

#include <string>
#include <iostream>
#include <fstream>

using namespace std;

class coordt {
        private:
                string code;
                double c_easting, c_northing, c_reducedlevel;
                void eval(char *currentLine);

        public:
                bool read_data(char *buf);
                string get_coord_code(void);
                void set_coord_code(string theCode);
                void show_details(void);

                void get_coords(double &easting, double &northing, double &reducedlevel);
};

struct graph_data {
        double e_inter[1024];
        double n_inter[1024];
        double rl_inter[1024];
    double eavg;
        double navg;
        double rlavg;
        int count;
};

#endif
```

Listing B.3: dcdatatemplate.cpp.

```
#include <stdafx.h>
#include "dcdata_template.h"

int dcdt::get_num_sats(void){
        return numsats;
}

string dcdt::get_qc1_code(void){
        return code;
}

void dcdt::set_qc1_code(string theCode){
        code = theCode;
}

void dcdt::show_details(void){
        cout << "Type:␣" << get_qc1_code() << "␣-␣Num␣Sats:␣" << get_num_sats() << \
                "␣|␣GPS:␣TGP:␣" << total_gps_pos << "␣GSW:␣" << gps_start_week <<
                "␣GST:␣" << gps_start_time << "␣GEW:␣" << gps_end_week << "␣GET:␣" << gps_end_time <<
                "␣|␣DOPS:␣PDOP:␣" << p_dop << "␣HDOP:␣" << h_dop << "␣VDOP:␣" << v_dop << "␣|"<< endl;
}

void dcdt::get_dops(double &pdop, double &hdop, double &vdop){

        // use call by reference to pass
        // all the juju
        pdop = p_dop;
        hdop = h_dop;
        vdop = v_dop;

}
bool dcdt::read_data(char *buf){
                // now check if it is qc1 data or just crap
                if(!strncmp(buf, "C6NM", 4)){
                        cout << "(dcdt␣class)␣Found␣C6NM␣QC1␣Data...\n";
                        // now here pull apart the line and save
                        eval(buf);

                        return true; // return true so we know to new() another class
                } else {
                        cout << "-";
                        return false;
                }
}

void dcdt::eval(char *currentLine){
        string buf[8];
        string bufnew;
        char *ptr;

        int i = 0;

        ptr = strtok(currentLine,".␣");
        buf[i++] = ptr;

        for(;i<8;i++){
                ptr = strtok(NULL,".␣");
                buf[i] = ptr;
        }

        /*for(i = 0;i<8;i++){
                cout << "BUF" << i << ": " << buf[i] << endl;
        }*/

        // now start pulling out the peices
        // first get the totalnum of sats
        bufnew = buf[0].substr(4, 2);
        numsats = atoi( bufnew.c_str() );

        // get the pdop
        bufnew = buf[0].substr(buf[0].length()-1,1);
        bufnew.append(".");
        bufnew.append(buf[1].substr(0,14));
        p_dop = strtod(bufnew.c_str(), NULL);

        // get the hdop
        bufnew = buf[1].substr(buf[1].length()-1,1);
        bufnew.append(".");
        bufnew.append(buf[2].substr(0,14));
        h_dop = strtod(bufnew.c_str(), NULL);

        // get the vdop
        bufnew = buf[2].substr(buf[2].length()-1,1);
        bufnew.append(".");
        bufnew.append(buf[3].substr(0,14));
        v_dop = strtod(bufnew.c_str(), NULL);

        bufnew = buf[4].substr(15,2);
        total_gps_pos = atoi(bufnew.c_str());
        gps_start_week = atoi((buf[5].substr(0,4)).c_str());

        bufnew = buf[5].substr(buf[5].length()-6,6);
        bufnew.append(".");
        bufnew.append(buf[6].substr(0,9));
        gps_start_time = atoi(bufnew.c_str());

        gps_end_week = atoi((buf[6].substr(9,4)).c_str());

        bufnew = buf[6].substr(buf[6].length()-6,6);
        bufnew.append(".");
        bufnew.append(buf[7].substr(0,9));
        gps_end_time = atoi(bufnew.c_str());

        /*for(i = 5;i<7; i++) {
                bufnew[i] = buf[i].substr(buf[i].length()-6,6);
                bufnew[i].append(".");
                bufnew[i].append(buf[i+1].substr(0,9));
                cout << "GPS time is: " << bufnew[i] << endl;
        }*/

        cout << "\nDecoded_QC1_data\n" << endl;
        //cout << "Min. Number of Satellites: " << bufnew[0] << endl;

        set_qc1_code(buf[0].substr(0, 4));


        /*for(i = 1;i<4;i++){
                cout << "*DOP Max " << i << ": " << bufnew[i] << endl;
         }*/
}
```

Listing B.4: coordtemplate.cpp.

```
#include <stdafx.h>
#include "coord_template.h"

string coordt::get_coord_code(void){
        return code;
}

void coordt::set_coord_code(string theCode){
        code = theCode;
}

void coordt::show_details(void){
        cout << "Type:_" << get_coord_code() << "_Easting:_" << c_easting << "_|_Northing:_" <<
                c_northing << "_|_Reduced_Level:_" << c_reducedlevel << endl;
}

void coordt::get_coords(double &easting, double &northing, double &reducedlevel){

        // use call by reference to pass
        // all the coords
        easting = c_easting;
        northing = c_northing;
        reducedlevel = c_reducedlevel;

}
bool coordt::read_data(char *buf){
                // now check if it is qc1 data
                if(!strncmp(buf, "08PD", 4)){
                        cout << "(coordt_class)_Found_08PD_Coordinate_Data...\n";
                        // now here pull apart the line and save
                        eval(buf);
                        return true; // return true so we know to new() another class
                }

                else {
                        cout << "-";
                        return false;
                }
}

void coordt::eval(char *currentLine){
        string buf[32];
        string bufnew;
        char *ptr;

        int where = 0;
        int i = 0;

        cout << "(COORD)_We_are_at_mark_" << ++where << endl;
        cout << "(COORD)_Token_ise_this_cunt_" << currentLine << endl;

        ptr = strtok(currentLine,"._");
        cout << "_POInter__is_" << ptr << endl;

        do{
                buf[i++] = ptr;
                cout << "(COORD)_i_is_" << i << "_and_here_it_is_" << buf[i-1] << endl;
                ptr = strtok(NULL,"._");

        }while (ptr != NULL);

        cout << "(COORD)_We_are_really_at_mark_" << ++where << endl;

        cout << "\nDecoded_GPS_Coordinate_Data\n" << endl;
        bufnew = buf[2];
        bufnew.append(".");
        bufnew.append(buf[3].substr(0, 6));
        c_easting = strtod(bufnew.c_str(), NULL);

        cout << "Easting: " << c_easting << endl;

        //cout << "Latitude:   " << bufnewer[0] << endl;
        cout << "(COORD) We are at mark " << ++where << endl;

        cout << " BUF[3] is " << buf[3] << endl;
        bufnew = buf[3].substr(buf[3].length()-7,7);
        bufnew.append(".");
        bufnew.append(buf[4].substr(0, 8));
        c_northing = strtod(bufnew.c_str(), NULL);

        cout << "Northing: " << c_northing << endl;

        cout << "(COORD) We are at mark " << ++where << endl;

        //cout << "Longitude: " << bufnewer[1] << endl;

        bufnew = buf[4].substr(8, 3);
        bufnew.append(".");
        bufnew.append(buf[5].substr(0, 9));
        c_reducedlevel = strtod(bufnew.c_str(), NULL);

        cout << "RL: " << c_reducedlevel << endl;

        cout << "(COORD) We are at mark " << ++where << endl;

        //cout << "Height: " << bufnewer[2] << endl;

        cout << "\nDecoded Coordinate data\n" << endl;
        //cout << "Min. Number of Satellites: " << bufnew[0] << endl;
        cout << "(COORD) We are at mark " << ++where << endl;

        set_coord_code(buf[0].substr(0, 4));
        cout << "(COORD) We are at mark " << ++where << endl;
}
```

Listing B.5: list.h.

```
// linked list class

struct list {
        list *next;
        void *data;
};

list *add_list_item(list *head, void *data);
list *get_last_list_item(list *head);
```

Listing B.6: list.cpp.

```
#include <stdafx.h>
#include <iostream>
#include "list.h"

using namespace std;

// linked list functions

// add an item to a list
list *add_list_item(list *head, void *data){

        list *listptr = new list;
        listptr->next = head;
        listptr->data = data;
        head = listptr;

        return head;
}
```

Listing B.7: Form1.h.

```
#pragma once
#include <string>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <cstringt.h>
#include <sstream>
#include <atlstr.h>
#include "dcdata_template.h"
#include "list.h"
#include "coord_template.h"
#include "graph.h"

graph_data graphdt;

namespace AttemptTwo
{
        using namespace System;
        using namespace System::ComponentModel;
        using namespace System::Collections;
        using namespace System::Windows::Forms;
        using namespace System::Data;
        using namespace System::Drawing;

        CString dcf;

        /// <summary>
        /// Summary for Form1
        ///
        /// WARNING: If you change the name of this class, you will need to change the
        ///          'Resource File Name' property for the managed resource compiler tool
        ///             associated with all .resx files this class depends on.  Otherwise,
        ///             the designers will not be able to interact properly with localized
        ///             resources associated with this form.
        /// </summary>
        public __gc class Form1 : public System::Windows::Forms::Form
        {
        public:
                Form1(void)
                {
                        InitializeComponent();
                }

        protected:
                void Dispose(Boolean disposing)
                {
                        if (disposing && components)
                        {
                                components->Dispose();
                        }
                        __super::Dispose(disposing);
                }
        private: System::Windows::Forms::MainMenu *  mainMenu1;
        private: System::Windows::Forms::MenuItem *  menuItem1;
        private: System::Windows::Forms::MenuItem *  menuItem2;
        private: System::Windows::Forms::MenuItem *  menuItem3;
        private: System::Windows::Forms::MenuItem *  menuItem4;
        private: System::Windows::Forms::Label *  label1;
        private: System::Windows::Forms::Label *  label2;
        private: System::Windows::Forms::Label *  label3;
        private: System::Windows::Forms::TextBox *  textBox1;
        private: System::Windows::Forms::TextBox *  textBox2;
        private: System::Windows::Forms::GroupBox *  groupBox1;
        private: System::Windows::Forms::Label *  label4;
        private: System::Windows::Forms::Label *  label5;
        private: System::Windows::Forms::Label *  label6;
        private: System::Windows::Forms::Label *  label7;

        private: System::Windows::Forms::TextBox *  textBox4;
        private: System::Windows::Forms::TextBox *  textBox5;
        private: System::Windows::Forms::TextBox *  textBox6;
        private: System::Windows::Forms::TextBox *  textBox7;

        private: System::Windows::Forms::Button *  button1;
        private: System::Windows::Forms::Button *  button2;
        private: System::Windows::Forms::Label *  label10;
        private: System::Windows::Forms::StatusBar *  statusBar1;
        private: System::Windows::Forms::TextBox *  textBox3;
        private: System::Windows::Forms::GroupBox *  groupBox2;
        private: System::Windows::Forms::Label *  label11;
        private: System::Windows::Forms::Label *  label12;
        private: System::Windows::Forms::Label *  label13;
        private: System::Windows::Forms::TextBox *  textBox10;
        private: System::Windows::Forms::TextBox *  textBox11;
        private: System::Windows::Forms::TextBox *  textBox12;

        private:
                /// <summary>
                /// Required designer variable.
                /// </summary>
                System::ComponentModel::Container * components;

                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                void InitializeComponent(void)
                {
                        this->mainMenu1 = new System::Windows::Forms::MainMenu();
                        this->menuItem1 = new System::Windows::Forms::MenuItem();
                        this->menuItem2 = new System::Windows::Forms::MenuItem();
                        this->menuItem3 = new System::Windows::Forms::MenuItem();
                        this->menuItem4 = new System::Windows::Forms::MenuItem();
                        this->label1 = new System::Windows::Forms::Label();
                        this->label2 = new System::Windows::Forms::Label();
                        this->label3 = new System::Windows::Forms::Label();
                        this->textBox1 = new System::Windows::Forms::TextBox();
                        this->textBox2 = new System::Windows::Forms::TextBox();
                        this->groupBox1 = new System::Windows::Forms::GroupBox();
                        this->textBox7 = new System::Windows::Forms::TextBox();
                        this->textBox6 = new System::Windows::Forms::TextBox();
                        this->textBox5 = new System::Windows::Forms::TextBox();
                        this->textBox4 = new System::Windows::Forms::TextBox();
                        this->label7 = new System::Windows::Forms::Label();
                        this->label6 = new System::Windows::Forms::Label();
                        this->label5 = new System::Windows::Forms::Label();
                        this->label4 = new System::Windows::Forms::Label();
                        this->button1 = new System::Windows::Forms::Button();
                        this->button2 = new System::Windows::Forms::Button();
                        this->label10 = new System::Windows::Forms::Label();
                        this->statusBar1 = new System::Windows::Forms::StatusBar();
                        this->textBox3 = new System::Windows::Forms::TextBox();
                        this->groupBox2 = new System::Windows::Forms::GroupBox();
                        this->textBox12 = new System::Windows::Forms::TextBox();
                        this->textBox11 = new System::Windows::Forms::TextBox();
                        this->textBox10 = new System::Windows::Forms::TextBox();
                        this->label13 = new System::Windows::Forms::Label();
                        this->label12 = new System::Windows::Forms::Label();
                        this->label11 = new System::Windows::Forms::Label();
```

```
this->groupBox1->SuspendLayout();
this->groupBox2->SuspendLayout();
this->SuspendLayout();
//
// mainMenu1
//
System::Windows::Forms::MenuItem* __mcTemp__1[] = new System::Windows::Forms::MenuItem
__mcTemp__1[0] = this->menuItem1;
__mcTemp__1[1] = this->menuItem4;
this->mainMenu1->MenuItems->AddRange(__mcTemp__1);
//
// menuItem1
//
this->menuItem1->Index = 0;
System::Windows::Forms::MenuItem* __mcTemp__2[] = new System::Windows::Forms::MenuItem
__mcTemp__2[0] = this->menuItem2;
__mcTemp__2[1] = this->menuItem3;
this->menuItem1->MenuItems->AddRange(__mcTemp__2);
this->menuItem1->Text = S"File";
//
// menuItem2
//
this->menuItem2->Index = 0;
this->menuItem2->Text = S"Load␣DC␣file..";
this->menuItem2->Click += new System::EventHandler(this, menuItem2_Click);
//
// menuItem3
//
this->menuItem3->Index = 1;
this->menuItem3->Text = S"Exit";
//
// menuItem4
//
this->menuItem4->Index = 1;
this->menuItem4->Text = S"Help";
//
// label1
//
this->label1->Location = System::Drawing::Point(24, 64);
this->label1->Name = S"label1";
this->label1->TabIndex = 0;
this->label1->Text = S"Enter␣Easting:";
//
// label2
//
this->label2->Location = System::Drawing::Point(24, 96);
this->label2->Name = S"label2";
this->label2->TabIndex = 1;
this->label2->Text = S"Enter␣Northing:";
//
// label3
//
this->label3->Location = System::Drawing::Point(24, 128);
this->label3->Name = S"label3";
this->label3->Size = System::Drawing::Size(120, 23);
this->label3->TabIndex = 2;
this->label3->Text = S"Enter␣Reduced␣Level:";
//
// textBox1
//
this->textBox1->Location = System::Drawing::Point(168, 64);
this->textBox1->Name = S"textBox1";
this->textBox1->TabIndex = 3;
this->textBox1->Text = S"";
//
// textBox2
//
this->textBox2->Location = System::Drawing::Point(168, 96);
this->textBox2->Name = S"textBox2";
this->textBox2->TabIndex = 4;
this->textBox2->Text = S"";
//
// groupBox1
//
this->groupBox1->Controls->Add(this->textBox7);
this->groupBox1->Controls->Add(this->textBox6);
this->groupBox1->Controls->Add(this->textBox5);
this->groupBox1->Controls->Add(this->textBox4);
this->groupBox1->Controls->Add(this->label7);
this->groupBox1->Controls->Add(this->label6);
this->groupBox1->Controls->Add(this->label5);
this->groupBox1->Controls->Add(this->label4);
this->groupBox1->Location = System::Drawing::Point(336, 176);
this->groupBox1->Name = S"groupBox1";
this->groupBox1->Size = System::Drawing::Size(264, 168);
this->groupBox1->TabIndex = 6;
this->groupBox1->TabStop = false;
this->groupBox1->Text = S"QC1␣Data␣(Averages)";
//
// textBox7
//
this->textBox7->Location = System::Drawing::Point(168, 120);
this->textBox7->Name = S"textBox7";
this->textBox7->Size = System::Drawing::Size(64, 20);
this->textBox7->TabIndex = 9;
this->textBox7->Text = S"";
//
```

```
// textBox6
//
this->textBox6->Location = System::Drawing::Point(168, 88);
this->textBox6->Name = S"textBox6";
this->textBox6->Size = System::Drawing::Size(64, 20);
this->textBox6->TabIndex = 8;
this->textBox6->Text = S"";
//
// textBox5
//
this->textBox5->Location = System::Drawing::Point(168, 56);
this->textBox5->Name = S"textBox5";
this->textBox5->Size = System::Drawing::Size(64, 20);
this->textBox5->TabIndex = 7;
this->textBox5->Text = S"";
//
// textBox4
//
this->textBox4->Location = System::Drawing::Point(168, 24);
this->textBox4->Name = S"textBox4";
this->textBox4->Size = System::Drawing::Size(64, 20);
this->textBox4->TabIndex = 6;
this->textBox4->Text = S"";
//
// label7
//
this->label7->Location = System::Drawing::Point(8, 120);
this->label7->Name = S"label7";
this->label7->TabIndex = 3;
this->label7->Text = S"VDOP␣Max:";
//
// label6
//
this->label6->Location = System::Drawing::Point(8, 88);
this->label6->Name = S"label6";
this->label6->TabIndex = 2;
this->label6->Text = S"HDOP␣Max:";
//
// label5
//
this->label5->Location = System::Drawing::Point(8, 56);
this->label5->Name = S"label5";
this->label5->TabIndex = 1;
this->label5->Text = S"PDOP␣Max:";
//
// label4
//
this->label4->Location = System::Drawing::Point(8, 24);
this->label4->Name = S"label4";
this->label4->Size = System::Drawing::Size(128, 23);
this->label4->TabIndex = 0;
this->label4->Text = S"Min␣Satellites␣Available:";
//
// button1
//
this->button1->Location = System::Drawing::Point(32, 248);
this->button1->Name = S"button1";
this->button1->Size = System::Drawing::Size(112, 32);
this->button1->TabIndex = 7;
this->button1->Text = S"Create␣Precision␣Graph";
this->button1->Click += new System::EventHandler(this, button1_Click);
//
// button2
//
this->button2->Location = System::Drawing::Point(176, 248);
this->button2->Name = S"button2";
this->button2->Size = System::Drawing::Size(112, 32);
this->button2->TabIndex = 8;
this->button2->Text = S"Create␣Accuracy␣Graph";
//
// label10
//
this->label10->Location = System::Drawing::Point(32, 312);
this->label10->Name = S"label10";
this->label10->Size = System::Drawing::Size(264, 32);
this->label10->TabIndex = 9;
this->label10->Text = S"(Accuracy␣Graph␣available␣only␣when␣user␣inputs␣Coordinates)";
//
// statusBar1
//
this->statusBar1->Location = System::Drawing::Point(0, 363);
this->statusBar1->Name = S"statusBar1";
this->statusBar1->Size = System::Drawing::Size(640, 22);
this->statusBar1->TabIndex = 10;
this->statusBar1->Text = S"Loading:";
//
// textBox3
//
this->textBox3->Location = System::Drawing::Point(168, 128);
this->textBox3->Name = S"textBox3";
this->textBox3->TabIndex = 11;
this->textBox3->Text = S"";
//
// groupBox2
//
this->groupBox2->Controls->Add(this->textBox12);
this->groupBox2->Controls->Add(this->textBox11);
this->groupBox2->Controls->Add(this->textBox10);
```

```
                        this->groupBox2->Controls->Add(this->label13);
                        this->groupBox2->Controls->Add(this->label12);
                        this->groupBox2->Controls->Add(this->label11);
                        this->groupBox2->Location = System::Drawing::Point(336, 16);
                        this->groupBox2->Name = S"groupBox2";
                        this->groupBox2->Size = System::Drawing::Size(264, 136);
                        this->groupBox2->TabIndex = 12;
                        this->groupBox2->TabStop = false;
                        this->groupBox2->Text = S"Coordinate_Averages_(From_observed_data)";
                        //
                        // textBox12
                        //
                        this->textBox12->Location = System::Drawing::Point(144, 88);
                        this->textBox12->Name = S"textBox12";
                        this->textBox12->TabIndex = 5;
                        this->textBox12->Text = S"";
                        //
                        // textBox11
                        //
                        this->textBox11->Location = System::Drawing::Point(144, 56);
                        this->textBox11->Name = S"textBox11";
                        this->textBox11->TabIndex = 4;
                        this->textBox11->Text = S"";
                        //
                        // textBox10
                        //
                        this->textBox10->Location = System::Drawing::Point(144, 24);
                        this->textBox10->Name = S"textBox10";
                        this->textBox10->TabIndex = 3;
                        this->textBox10->Text = S"";
                        //
                        // label13
                        //
                        this->label13->Location = System::Drawing::Point(16, 88);
                        this->label13->Name = S"label13";
                        this->label13->TabIndex = 2;
                        this->label13->Text = S"Reduced_Level:";
                        //
                        // label12
                        //
                        this->label12->Location = System::Drawing::Point(16, 56);
                        this->label12->Name = S"label12";
                        this->label12->TabIndex = 1;
                        this->label12->Text = S"Northing:";
                        //
                        // label11
                        //
                        this->label11->Location = System::Drawing::Point(16, 24);
                        this->label11->Name = S"label11";
                        this->label11->TabIndex = 0;
                        this->label11->Text = S"Easting:";
                        //
                        // Form1
                        //
                        this->AutoScaleBaseSize = System::Drawing::Size(5, 13);
                        this->ClientSize = System::Drawing::Size(640, 385);
                        this->Controls->Add(this->groupBox2);
                        this->Controls->Add(this->textBox3);
                        this->Controls->Add(this->statusBar1);
                        this->Controls->Add(this->label10);
                        this->Controls->Add(this->button2);
                        this->Controls->Add(this->button1);
                        this->Controls->Add(this->groupBox1);
                        this->Controls->Add(this->textBox2);
                        this->Controls->Add(this->textBox1);
                        this->Controls->Add(this->label3);
                        this->Controls->Add(this->label2);
                        this->Controls->Add(this->label1);
                        this->Menu = this->mainMenu1;
                        this->Name = S"Form1";
                        this->Text = S"v";
                        this->groupBox1->ResumeLayout(false);
                        this->groupBox2->ResumeLayout(false);
                        this->ResumeLayout(false);

                }
        private: System::Void menuItem2_Click(System::Object *  sender, System::EventArgs *  e)
                        {
                                OpenFileDialog *openFileDialog1 = new OpenFileDialog();

                                //openFileDialog1->InitialDirectory = S"c:\\" ;
                                 openFileDialog1->Filter = "DC_Files_(*.dc)|*.txt|All_Files_(*.*)|*.*" ;
                                 openFileDialog1->FilterIndex = 2 ;
                                 openFileDialog1->RestoreDirectory = true ;

                                        if(openFileDialog1->ShowDialog() == DialogResult::OK)
                                         {

                                                dcf = openFileDialog1->FileName;
                                                statusBar1->Text = dcf;
                                                loadfile(dcf);

                                        }

                        }

private: System::Void loadfile(CString loadthis){
                        statusBar1->Text = "Loading:_" + loadthis;
                dcdt *dcdata;
                coordt *coordata;
                list *dcdata_list = new list;
                list *coordata_list = new list;
                dcdata_list->next = NULL;
                coordata_list->next = NULL;
                //dcdt *dcdtptr = &dcdata;
                CString tmpstr;

                int count = 0;
                //int counter = 0;
                //int counting = 0;
                //int county = 0;

                ifstream dcfile;

                if(loadthis){
                        dcfile.open( loadthis );
                } else {
                        MessageBox(NULL, "File_does_not_exist!", "Fatal_Error", MB_OK);
                        exit(1);
                }

                if(!dcfile){
                        MessageBox(NULL, "Could_not_open_file!", "Fatal_Error", MB_OK);
                        exit(1);

                }

                char buf[1024];
                dcdata = new dcdt;
                coordata = new coordt;

        while(!dcfile.eof()){

                // read a line of data from the file
                dcfile.getline(buf, 256);

                if(dcdata->read_data(buf)){
                        //if here, it was qc1 data, so make a new dcdt
                        //cout << "(main) CHECKING READ_DATA Code: " << dcdata->get_qc1_code() << " Numsats: "
                        dcdata_list = add_list_item(dcdata_list, (void *)dcdata);
                        dcdata = new dcdt;
                }
                else if(coordata->read_data(buf)){
                        //if here, it was coord data
                        coordata_list = add_list_item(coordata_list, (void *)coordata);
                        coordata = new coordt;
                }

        }

        // here we are just showing the data for debugging perposes

        list *listptr = dcdata_list;
        double ptmp,htmp,vtmp;
        double h_accum = 0.0, p_accum = 0.0, v_accum = 0.0;
        double e_accum = 0.0, n_accum = 0.0, rl_accum = 0.0;
        count = 0;
        TCHAR sz[1024];

    while(dcdata_list->next != NULL) {
                ((dcdt *)(dcdata_list->data))->show_details();
                ((dcdt *)(dcdata_list->data))->get_dops(ptmp,htmp,vtmp);


                // keep a total
                p_accum += ptmp;
                h_accum += htmp;
                v_accum += vtmp;
                count++;

                dcdata_list = dcdata_list->next;
        }

        double pavg = p_accum / (double)count;
        double havg = h_accum / (double)count;
        double vavg = v_accum / (double)count;

        //sprintf(sz, "%d", dc);
        //textBox4->Text = sz;


        sprintf(sz, "%f", pavg);
        textBox5->Text = sz;

        sprintf(sz, "%f", havg);
        textBox6->Text = sz;

        sprintf(sz, "%f", vavg);
        textBox7->Text = sz;


        list *listpointer = coordata_list;
        double etmp,ntmp,rltmp;
        count = 0;


        int counter = 0;
        double e_inter[1024];
        double n_inter[1024];
        double rl_inter[1024];


        while(coordata_list->next != NULL) {
                ((coordt *)(coordata_list->data))->show_details();
                ((coordt *)(coordata_list->data))->get_coords(etmp,ntmp,rltmp);

                //cout << "The Coordinates ARE IN! " << ltmp << " " << lgtmp << " " << hgtmp << endl;
                e_inter[counter] = etmp;
                n_inter[counter] = ntmp;
                rl_inter[counter] = rltmp;

                graphdt.e_inter[counter] = etmp;
                graphdt.n_inter[counter] = ntmp;
                graphdt.rl_inter[counter] = rltmp;

                // keep a total
                e_accum += etmp;
                n_accum += ntmp;
                rl_accum += rltmp;
                count++;
                counter++;

                coordata_list = coordata_list->next;
        }
        //cout << "LETS SEE TOTALAGE! lavg = " << l_accum / (double)count << " lgavg = " <<
        //      lg_accum / (double)count << " hgavg = " << hg_accum / (double)count << endl;
        //cout << "|\n" << endl;

        double eavg = e_accum / (double)count;
        double navg = n_accum / (double)count;
        double rlavg = rl_accum / (double)count;

        graphdt.eavg = eavg;
        graphdt.navg = navg;
        graphdt.rlavg = rlavg;

        for(counter = 0;counter < 6; counter++){
                //tmpstr = "Show Residuals of Latitude: " + _itoa(l_inter[counter] - lavg) + " Longitude: "
                //      + _itoa(lg_inter[counter] - lgavg) + " Height: " + _itoa(hg_inter[counter] - hgavg);
                sprintf(sz, "Residuals␣of␣Easting:␣%f\n␣Northing:␣%f\n␣Reduced␣Level:␣%f", e_inter[counter] -

                MessageBox(NULL, sz , "Information", MB_OK);

        }

        sprintf(sz, "%f", eavg);
        textBox10->Text = sz;

        sprintf(sz, "%f", navg);
        textBox11->Text = sz;

        sprintf(sz, "%f", rlavg);
        textBox12->Text = sz;

        double u_easting;
        double u_northing;
        double u_reducedlevel;

        tmpstr = textBox1->Text;
        u_easting = atof(tmpstr);

        tmpstr = textBox2->Text;
        u_northing = atof(tmpstr);

        tmpstr = textBox3->Text;
        u_reducedlevel = atof(tmpstr);

        //sprintf(sz, "User East: %f\n North: %f\n Reduced Level: %f", u_easting, u_northing, u_reducedlevel);
        //MessageBox(NULL, sz , "Information", MB_OK);

        dcfile.close();
        MessageBox(NULL, "QC1␣Data␣File␣Analysis␣Complete." , "Information", MB_OK);
        }
        private: System::Void menuItem3_Click(System::Object * sender, System::EventArgs * e)
                         {
                                 exit(0);
                         }

private: System::Void button1_Click(System::Object * sender, System::EventArgs * e)
                 {
                         Graph *thegraph = new Graph();
                         thegraph->ShowDialog();
                         //thegraph->pictureBox1->Enabled=0;


                 }

};
}
```

Listing B.8: Form1.cpp.

```
#include "stdafx.h"
#include "Form1.h"
#include <windows.h>

using namespace AttemptTwo;

int APIENTRY _tWinMain(HINSTANCE hInstance,
                       HINSTANCE hPrevInstance,
                       LPTSTR    lpCmdLine,
                       int       nCmdShow)
{
        System::Threading::Thread::CurrentThread->ApartmentState = System::Threading::ApartmentState::STA;
        Application::Run(new Form1());

        return 0;
}
```