MEASUREMENT-BASED

GEOSPATIAL DATA MANAGEMENT

by

Michael Bremner
BScE. Geodesy and Geomatics Engineering, University of New Brunswick, 2015

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science in Engineering

In the Graduate Academic Unit of Geodesy and Geomatics Engineering

Supervisor:	Marcelo Santos, PhD, Geodesy and Geomatics Engineering
Examining Board:	Maciej Bazanowski, PhD, Geodesy and Geomatics Engineering
	Suprio Ray, PhD, Computer Science

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

June 2019

Abstract

Most land surveying firms use ad hoc data management systems that inhibit the reuse of existing data and make it difficult to identify redundant work. This causes systematic inefficiencies in land survey operations. This project aims to provide a solution to this inefficiency by researching data management methods for land survey data.

Measurement-Based GIS (MBGIS) is selected as the avenue for this research because of previous literature discussing the potential benefits of its use as a land survey data management system. Unfortunately, existing implementations have issues with processing times and flexibility of data integration. Further research requires a MBGIS prototype to be implemented. A conceptual design of a full data management system is created to provide context for the design and implementation of the prototype.

The prototype is developed as a geodetic library and a suite of data integration tests. Each of the tests integrates a series of discrete datasets into a network using a different data integration strategy. The processing times, precision of the resulting networks, and flexibility of the strategies are compared. Two of the strategies provide very similar, positive results. It appears that a MBGIS performing rigorous geodetic reductions of measurements and using one of the two successful integration strategies could be used as a data management system and could improve the efficiency of land survey operations.

Acknowledgements

Throughout this project I have received a great deal of support and assistance. I would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and Mitacs for their support of this research. I would like to thank Julia Ramirez, from Planet Hatch, for her assistance with managing this project.

I would like to thank my supervisor Dr. Santos for encouraging and guiding me through this project. I would also like to thank the entire Faculty and Staff of the Geodesy and Geomatics Engineering Department, the contributions I have received in the forms of advice, feedback, technical expertise, and administrative support have been too numerous to detail.

Several individuals in the Computer Science Department have also provided me with invaluable advice through this project. I would Like to thank Dr. Evans, Rick Wightman, and John Peterson for their help with designing, implementing and optimizing the software required for this project.

Finally, I would like to thank my friends and family for their support through this process. A thesis can be a long and lonely task for those without a strong support system; thankfully, for me it was only long… really, really, long.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Land surveyors collect vast sums of geospatial data. Collection and processing of geospatial data is the bulk of the work performed by most land survey firms. Typically, data is collected for a specific project, it is used only for that project then is archived indefinitely. The data management procedures at most land survey firms inhibit the reutilization of previously collected data by storing it in small discrete datasets that are difficult to search through. These data management procedures make it difficult to identify redundant work, causing systematic inefficiency (Bartosh, 2012).

This thesis explores options for improving data management of land survey data. If discrete datasets of survey data could be integrated into queryable databases, the efficiency of land survey operations could be improved (McKay, 2018). One tool for storing, querying, and manipulating geospatial data in databases is GIS, unfortunately some design choices made early in the evolution of GIS systems have made it unsuitable for managing land survey data (Bartosh, 2012).

Goodchild (2002) proposed a Measurement-Based GIS (MBGIS) that, theoretically, would be much better suited for managing land survey data. Bartosh (2012) proposed that a MBGIS implementation could be used as a data management system for land survey data. Bartosh was unable to achieve his desired results, due to limitations of the

MBGIS implementation he was testing. A MBGIS implementation suitable for management of land survey data does not appear to exist. In order to further explore the possibilities of MBGIS as a land survey data management tool, a prototype MBGIS is implemented.

In order to provide a context for the design and development of the prototype, conceptual designs are created for an entire measurement-based data management system. The full data management system would consist of several modules: a web server for storing and managing access to the data, a mobile client for data collection, a desktop application for data manipulation, application plugins for accessing integrated data in other surveying software, and a geodetic library for performing calculations.

The prototype implemented in this project is limited to the geodetic library and a suite of data integration tests. The geodetic library and data integration tests are designed to explore and address some of the issues found in previous MBGIS implementations that make the management of land survey data problematic. These issues include the processing time of data integrations, the flexibility of data integrations, and the geodetic reduction of field measurements. Functions in the geodetic library address the reduction of field measurements and provide several different data integration strategies. The data integration tests compare these data integration strategies to explore the issues of processing time and flexibility.

Five data integration strategies are tested. The first uses the same method as previous MBGIS implementations. This method is expected to provide the most rigorous results and the longest processing times. The networks generated by integrating using this method are used as the baseline for determining the precision of the four alternative integration strategies. Each of the alternative integration strategies uses constraints derived from previously integrated data to process later integrations. These alternative strategies are expected to provide significant processing time reductions. The processing times, precision of the resulting networks, and flexibility will be considered when comparing the alternative strategies to each other and the standard method.

This thesis is divided into chapters. The second chapter starts with a review of the current state of land survey data management. It then discusses the issues that make it difficult for land surveyors to use conventional GIS to manage their geospatial data. The concept of MBGIS and some previous implementations of it are also discussed. Because a significant portion of the project is software development, a discussion of software engineering processes is also provided. Chapter three discusses the objectives of this project.

Chapter four explores the development and testing of the prototype MBGIS. It begins with a discussion of the development tools that were researched for this project. The

conceptual designs for the entire system and each of the modules are then reviewed. Detailed documentation of the implementation of the geodetic library is provided, focusing on design decision and geodetic techniques. Finally, a discussion is provided on the design and implementation of both the systems integrity tests and the data integration tests. Chapter five provides the results of the data integration tests and Chapter six provides an analysis of the results.

2. Background

The goal of this thesis is to research potential improvements to the data management procedures used in the Land Surveying Industry. Bartosh (2012) and McKay (2018) have both proposed improvements to the current methods of data management. The systems proposed by these two papers have many similarities, with the largest difference being scale.

Bartosh (2012) proposes that land survey data could be better managed in a Measurement-Based GIS (MBGIS), a system that stores measurements and uses least squares estimation to integrate data and manage uncertainty. McKay (2018) proposes a National Spatial Data Infrastructure that is also measurement-based and uses least squares estimation for data integration. Although MBGIS is never mentioned in McKay (2018) the system proposed would seem to qualify as a MBGIS. The similarity between these systems seem reasonable as the primary data recorded by land surveyors is measurements and least squares is a rigorous method of integrating measurements.

This project will further consider MBGIS as a method of data management for land survey data. This background section will consider four topics: current methods of data management, usage of GIS by the land surveying industry, the Measurement-Based GIS concept, and software development processes.

2.1 Land Surveying Data Management

Collection and processing of geospatial data is the bulk of the work performed at most land survey firms. While land survey firms may work on a variety of projects like legal boundary projects, construction projects, and mapping projects, nearly all projects involve the collection of geospatial data. Over time, the survey activities performed by a given firm inevitably accumulate vast sums of geospatial data.

Typically, data is collected for a specific project, used to produce survey plans or other deliverables for said project, then archived. For legal and liability reasons, the collected data and derived products are often retained for extended periods of time, if not indefinitely. Most land surveying firms create ad-hoc data management procedures for archiving paper documents and digital data in discrete datasets indexed by project numbers (Bartosh, 2012). Data archived in these discrete datasets is often difficult to search, making it difficult to identify redundant work. These ad-hoc data management procedures often cause systematic inefficiency for the land survey firm employing them by inhibiting the utility of the archived data (Bartosh, 2012).

GIS is a tool for managing geospatial data. It can contain and manage very large sets of data and provides many tools for the analysis, extraction and manipulation of the data. If land survey data was managed in a centralized GIS database, it could allow authorized users to search and extract data from previous projects to improve the

efficiency of later projects. Some of the benefits that could be achieved include: improved project planning and quotes, reduced time searching for physical evidence, reduced redundant work, and immediate detection of some blunders (McKay, 2018).

2.2 GIS as a Land Surveying Tool

GIS systems were initially developed in the 1960's (Goodchild, 2002). Since this time the GIS has undergone numerous developments and has become an invaluable tool to many users. Despite both GIS and Land Surveying both being industries heavily related to geospatial data, Land Surveyors have largely declined to use GIS (Bartosh, 2012; Jeffress, 2005). Articles are published on potential uses of GIS in Land Surveying trade publications on a regular basis, demonstrating an interest in the Land Surveying community to explore the potential GIS has as a surveying tool (Bartosh, 2012; Jeffress, 2005).

Unfortunately, there are several arguments for surveyors not to trust GIS. Conventional GIS systems are coordinated based and do not retain the raw measurements, while Land Surveyors consider the measurements to be the primary information and consider coordinates to be secondary information (Belussi & Migliorini, 2012; Vranic, Jurakic, & Matijevic, 2014). Error management of geospatial data is a major component of the Land Surveyor's responsibility, and conventional GIS systems have no capacity for error propagation or uncertainty tracking (Bartosh, 2012; Goodchild, 2002). Finally,

Land Surveyors are professionals and have a professional responsibility to ensure the quality of their work, work which often caries significant liability. Most GIS implementation are so complex that they require a specially-trained GIS professional to operate (Savvaidis & Stergioudis, 2012). The training investment required to allow Land Surveyors to utilize GIS at the level that their professional responsibilities require presents a significant barrier to entry.

2.3 Measurement-Based GIS

Goodchild (2002) laid out a conceptual design for a Measurement-Based GIS (MBGIS) system. This conceptual design was created to address issues faced by the GIS community, caused by ignoring the uncertainties inherent in all geospatial data. MBGIS is distinct from the commonly used Coordinate-Based GIS (CBGIS) systems in that it retains the original measurements and prioritizes them above the coordinates, allowing for error analysis and integration of new data. Goodchild (2002) asserts that MBGIS systems are superior to CBGIS systems and that the dominance of CBGIS systems severely limits the value of GIS to users. The MBGIS model was further refined by Leung, Jiang-Hong, & Goodchild (2004) who proposed using least squares estimation for data integration and error propagation.

ESRI (Redlands, CA) has since made attempts to implement some MBGIS concepts in their existing ArcGIS software, creating the Survey Analyst module, then the Parcel

Fabric module. Both these modules were created with a measurement-based cadastre in mind. Unfortunately, neither of these modules has been well received by reviewers (Bartosh, 2012; Navratil, Franz, & Pontikakis, 2004)

Navratil et al. (2004) evaluated ESRI's Survey Analyst module. They manufactured a network of 7000 observations based on 870 points from a sub-set of an actual property map. Survey Analyst delegates the actual adjustment to an external software package and in this test an adjustment package name Geosi was used. The test required an adjustment of the 7000 observations and 2610 parameters and took 36 hours to complete on a single core 2.4 GHz computer with 1GB of RAM. The team concluded that while Survey Analyst was a step towards MBGIS, the implementation had several issues, including an unacceptable processing time. They believed that due to the frequent re-adjustments that would be required of a MBGIS, the system would need to be able to handle adjustments of at least 47,000 observations and 22,000 parameters (numbers that appear to be derived from the full set of the property map used) in a small fraction of the time that was required for the test as performed.

Bartosh (2012) proposed that the MBGIS concept could be applied to solve the data management problems experienced by the land survey Industry. He anticipated that a MBGIS based data management system would reduce redundant work and improve the overall efficiency of land surveying operations. He evaluated ESRI's Parcel Fabric

module against the concept of a MBGIS and tested its performance as a data management tool for land surveyors. Bartosh used points from another property map and metes and bounds descriptions of the parcels as observations for his tests. Bartosh performed three tests, expanding the network with each test. The third test failed to adjust because the observations were inconsistent, and the system was not flexible enough to handle the inconsistencies. He did not include the number of points or observations in his publication, but from the figures it appears that these numbers were considerably smaller than those used in Navratil et al. (2004). The processing times were also not published. Bartosh concluded that while the Parcel Fabric module had several elements of a MBGIS, it did not fulfil Goodchild's definition of a MBGIS and that there were several issues preventing it from becoming a suitable data management tool for land surveyors.

It is notable that neither of these tests used actual survey measurements. Navratil et al. (2004) manufactured observations by calculating inverses between points. Bartosh (2012) used metes and bounds descriptions of existing parcels. In both cases the observations would have already be reduced to the mapping plane. A MBGIS suitable for land survey data management would need to be able to reduce field measurements itself, it appears that in order to perform further research into MBGIS's suitability as a land survey data management system, new software will need to be developed.

2.4 Software Development Processes

Software systems have evolved drastically since the early days of electronic computing. As the capabilities of these systems has grown so has their complexity and the complexity of their development (Orso, 2014). In order to keep pace with the continually increasing complexity of software systems, developers have developed numerous tools to increase the productivity of developers and ensure the stability of the systems. These tools include high level programing languages, integrated development environments, automated build and test suites, and formal software development processes (Orso, 2014).

Software is engineered in similar fashion to how physical objects are engineered. In Canada, Software Engineers are recognized and accredited by the Canadian Engineering Accreditation Board just as all other engineering disciplines are (Engineers Canada, 2017). Although it is an engineering discipline, there are some drastic differences between software engineering and more established engineering disciplines like civil engineering. Some differences are the result of the intangible nature of software products, while other differences come from the relative youth of the software engineering discipline (Orso, 2014). Due to both factors, clients often only have vague notion of what they need and have difficulty describing even that much (Orso, 2014). However, when presented with the product, they are typically very capable of describing why it does not meet their needs (Orso, 2014). Additionally,

design patterns are not as well established in software engineering as they are in other disciplines, leading to design choices that are often revised as the system is assembled (Orso, 2014). These, and other, factors combine to create an environment where requirements can change at any time during a project (Orso, 2014).

The first software development processes were modeled on other project management and engineering processes. These processes typically resembled the waterfall model (Figure 2.1) where each phase was completed before work on the next phase began (Orso, 2014). The system designs created were often very detailed, as designs are in other engineering disciplines. The overhead costs of creating these detailed plans were often lost when requirements changed, which caused many projects to go over budget, and many more to be cancelled after significant expenditure (Orso, 2014).

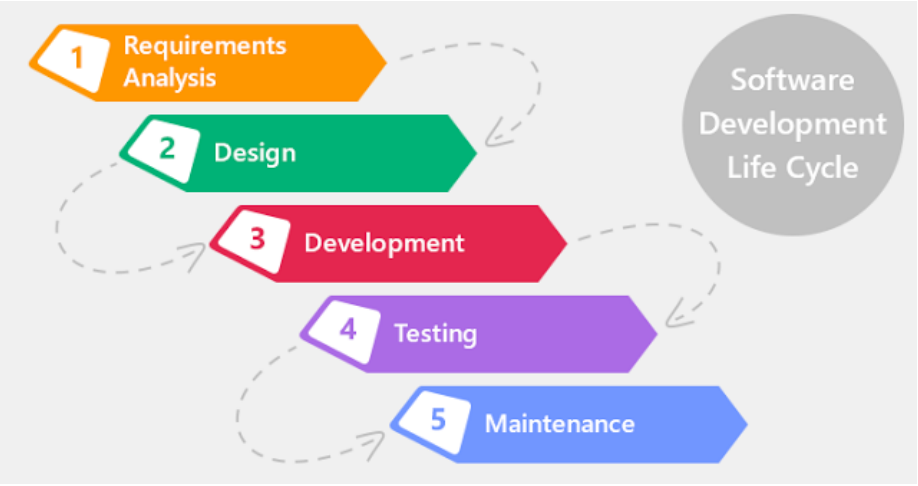

Figure 2.1: Waterfall Development Model (Mallepally, 2016)

The waterfall approach has since given way to lower risk development processes. Agile processes such as Extreme Programing and SCRUM are now popular methods for software development (Orso, 2014). These agile processes have drastically reduced planning overhead and focus on incremental development (Orso, 2014). Producing small reusable code components early and often, these processes allow developers to demonstrate prototype products to clients and allow clients to critique and request changes without endangering the project. Designing code in ways to ensure it is reusable when requirements change takes skill and experience, therefore agile methods may not be the best choice for a project run by novice developers.

The Rational Unified Process (RUP) is a specific case of the Unified Software Development Process (Kruchten, 2000). It lies somewhere in the middle ground between the rigid and high overhead waterfall methods and the lightweight agile methods. RUP was developed by the Rational Software Company who marketed the process to other software development entities (Kruchten, 2000). It was developed in conjunction with and by the same personnel as the Unified Modeling Language (UML) which is used extensively in almost every modern software development process (Jacobson, Grady, & Rumbaugh, 1999; Rumbaugh, Jacobson, & Booch, 2005).

RUP is marketed as use case driven, architecture centric, and iterative. It advocates the creation of an extensive use-case model which is used in the creation of architecture

models, development of components, and design of unit and system tests (Kruchten, 2000). The architecture models are used to compartmentalize component development and promote reusability of code (Kruchten, 2000). The process consists of four phases (Figure 2.2): Inception, Elaboration, Construction, and Transition and contains iterations within some phases (Kruchten, 2000). It also consists of 6 engineering workflows and 3 support workflows (Kruchten, 2000). Work flows are performed approximately in parallel with varying degrees within each phase.

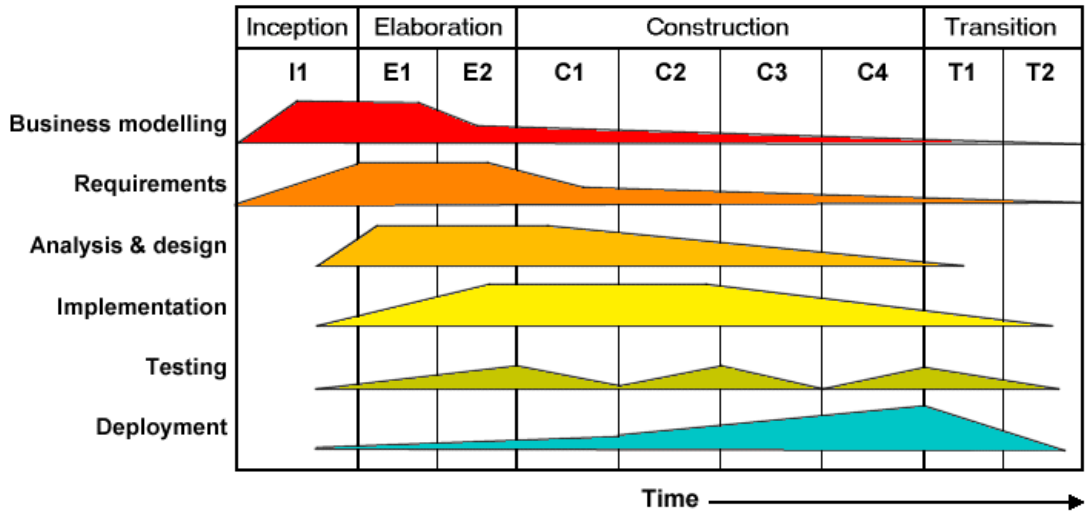

Figure 2.2: Rational Unified Process Structure (Mallepally, 2016)

RUP was developed to facilitate the development of large software systems with high technical complexity and high organizational complexity (Kruchten, 2000). The process generates a large variety of documents, models and code samples, termed artifacts. Many of these artifacts are intended to facilitate development, while some are intended to accompany the final product (also an artifact) when it is delivered to the

client (Kruchten, 2000). RUP advises users to streamline the process whenever possible. Users are advised that during the adoption the process artifacts and workflows should be assessed in the context of the project or adopting organization and those deemed unnecessary should be removed (Kruchten, 2000).

2.5 Conclusions

The collection of geospatial data is a major component of the activities carried out by land survey firms and over time the quantity of data they accumulate can be vast. Unfortunately, the ad hoc data management procedures used by many firms limit the utility of the accumulated data. There is significant room for improvement in land survey data management procedures.

Despite strong arguments against the use of GIS, the land surveyor community still shows interest in it as a potential tool. This indicates that there is some understanding of how the GIS could improve land survey data management. There is reason to believe that if the arguments against GIS were properly addressed, and its value demonstrated, a portion of the land surveyor community would embrace it enthusiastically.

MBGIS addresses two of the three main arguments against adoption of GIS by land surveyors. MBGIS was conceived as a tool for GIS professionals, but has not seen

widespread adoption. Repurposed, MBGIS may be an excellent basis for a survey data management system. Unfortunately, a MBGIS implementation suitable for this role is not currently available. In order to perform further research a new MBGIS implementation will need to be prototyped.

The third argument against adoption of GIS by land surveyors could be addressed in the implementation of the system. A data management tool designed for Land Surveyors does not need full GIS functionality. Combining core MBGIS functionality with tools and interfaces similar to existing land surveying software will ease the adoption process, allowing Land Surveyors to seamlessly transition and enjoy the benefits of GIS technologies.

In order to develop this prototype, the Rational Unified Process is used. The technical complexity of this project is within the range that RUP was intended for, but the organizational complexity is far less than what is expected. Therefore, several artifacts and all of the support workflows are omitted, while several of the engineering workflows have reduced scope.

3. Research Objectives

The main goal of this project is to improve the data management of land survey data. Bartosh (2012) suggests that this goal could be accomplished utilizing an implementation of a Measurement-Based GIS (MBGIS). Unfortunately, a suitable implementation of MBGIS does not appear to exist. Therefore, to further explore this assertion, a prototype MBGIS must be designed, implemented and tested.

This project develops a Java library, containing datatypes and geodetic functions, that is used to prototype a MBGIS as a suite of data integration tests. In order to provide a context for the design and development of the geodetic library, conceptual designs of the entire data management system and some of the modules is developed, several high-risk components are also researched and prototyped.

Three main issues with previous MBGIS implementations were identified in section 2.3: processing times, flexibility of adjustment methods, and reductions of field measurements. These issues are addressed in the design of the overall system, geodetic library and data integration tests.

Geodetic techniques for processing of measurements and least squares estimation are researched. These geodetic processing techniques are implemented in the geodetic

library. The standard adjustment method and several alternative adjustment strategies are developed and implemented so that they can be tested and compared in the data integration tests. Suites of integrity tests is also developed and implemented as part of the geodetic library implementation. These suites contain unit and integration tests (not to be confused with the academic tests focusing on data integration) to ensure the correct behavior of the datatypes and functions within a specific package and the packages it depends on.

Once the geodetic library is complete, the suite of academic tests is developed. These academic tests perform data integration of actual land survey measurements, imported from ".fbk" measurement files. Tests are developed for each of the data integration strategies implemented in the geodetic library. Each of the alternative adjustment strategies are compared to the standard method to determine its precision.

Once all the academic tests are run they are compared against each other on metrics of precision and processing time. The flexibility of the alternative strategies is difficult to quantify, so this is compared qualitatively. Finally, the quantitative and qualitative metrics of the academic tests are analyzed so that conclusion can be drawn on the suitability of the system and each of the integration strategies as a land survey data management tool.

4. Design and Implementation

Research of geodetic techniques, software development and academic testing are all important components of this project, but the bulk of the work required is software development. Therefore, software development techniques, specifically the Rational Unified Process (RUP), forms the basis for the planning and management of this project.

RUP was developed to allow for the development of highly complex systems with many stakeholders by large teams of developers (Kruchten, 2000). The complexity of the systems is referred to as technical complexity and the complexity of the team-stakeholder system is referred to as organizational complexity. The system developed by this project has reasonably high technical complexity, but low organizational complexity. Therefore, some components of RUP are omitted, and others are reduced, as recommended by the process documentation (Kruchten, 2000).

After an analysis of the project objectives and the RUP process documentation, the project plan displayed in Figure 4.1 was developed. The project will use the four phases defined by RUP: inception, elaboration, construction, and transition, some of which will contain iterations.

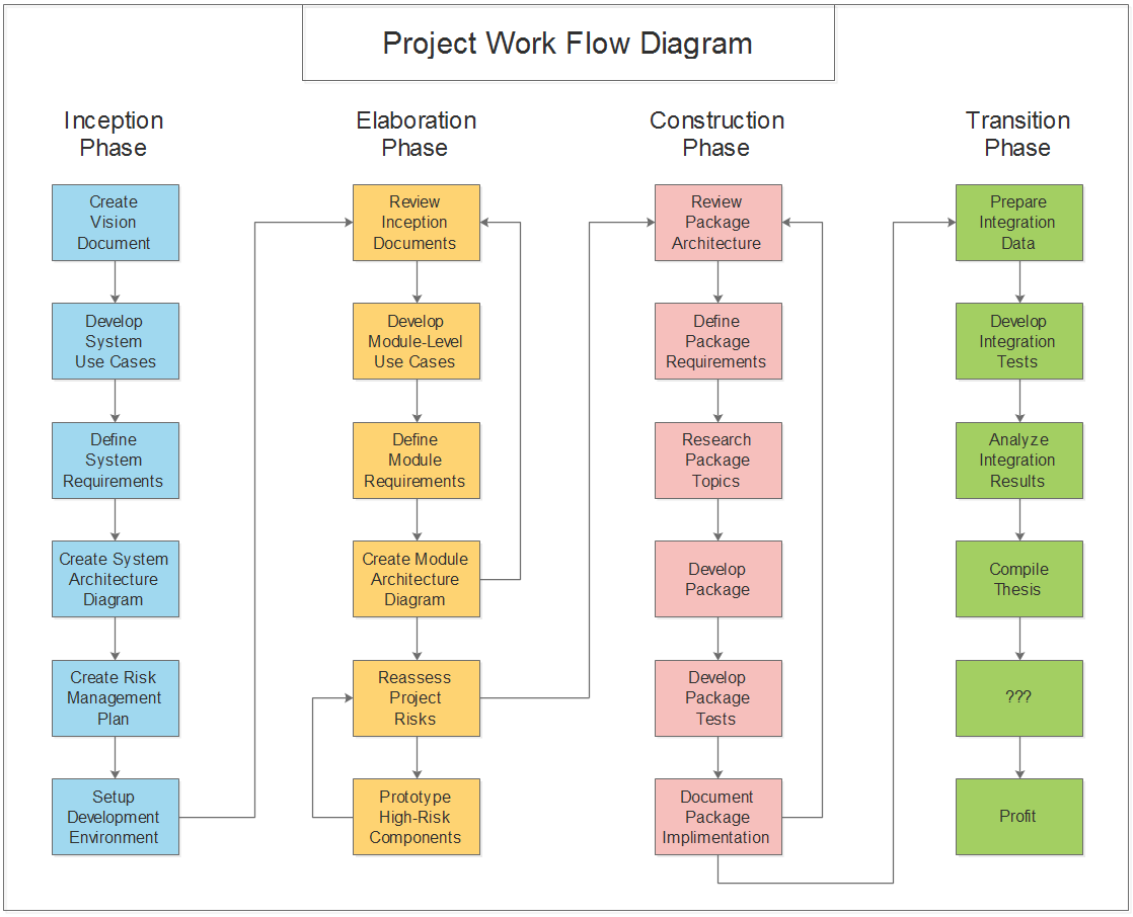

Figure 4.1: Project Workflow Summary

No iteration take place during the inception phase. The objectives for this phase are the development of system level use-cases, system requirements, system architecture diagrams, risk management plans and the setup of the development environment. System use cases are developed and used to define system requirements. Use-cases and system requirements are used to develop a system architecture diagram. Project risks are identified and, and a risk management plan is generated to address the identified risk. Finally, the development environment is configured by researching, testing and selecting development tools.

The objectives of the elaboration phase are the development of module level architecture diagrams and prototypes of certain high-risk components. Both objectives require iterations. For certain modules use-cases and / or module requirements are developed. Module architectures are modeled based on the module level use cases and requirements. Once all necessary module level architecture diagrams are created, project risks are reassessed, and prototypes are developed to explore high risk components, iteratively until all high-risk components have been addressed.

The construction phase objectives are the design, development and integrity testing of all geodetic library packages. This phase is performed in iterations, developing one package at a time. Package requirements are determined after reviewing the geodetic library architecture diagram. For most packages, some research of geodetic techniques is required in order to design and implement the package. During research, if any numerical examples or similar software packages are found they are saved in order to generate unit tests during the development of the package. A regression test suite is created from the unit tests to aide in the detection of faults introduced during later maintenance. For packages that rely on previously developed packages, integration tests are also added to the package test suite. Once the development and testing of the package is complete documentation is updated to reflect the design choices made and geodetic methods used to develop the package. The geodetic library architecture diagram is also updated to reflect the actual implementation of the package.

The objectives of the Transition phase are to perform the academic data integration tests, analyze the results, and complete the documentation of the project. The field measurement test data is prepared for integration testing. The standard least squares integration test is developed and performed first, then each of the alternative integration strategies are developed, performed, and compared to the standard method. Finally, the results are analyzed and documented.

4.1 Development Environment

The final step in the inception phase is the development environment step. During this step, technologies are researched and tested to determine their usefulness to the project. Testing is also performed to ensure understanding of the usage and limitations of the technologies. Once testing is complete, tools are selected, and the environment is configured. Subsequently, the development environment should remain as static as possible as changes to the environment could cause disruptions in the project.

There are many software tools required to develop a software system and many optional components that can add functionality or improve efficiency. These tools and components combine to create the software environment. Aspects of the development environment can include the programming language, development tools, runtime environments, databases, and libraries. The tools and components explored for the design and implementation of this project will be discussed in the following sections.

4.1.1 Software Languages

Many programming languages have more than one version. This is the case for both the languages used in this project. This section discusses the editions of Java and C explored in this project.

Java Standard Edition

Java is a programing language, and it is also a computing platform (Oracle, 2014a). A Java Runtime Environment (JRE) comes preloaded by the manufacturer on most computers, regardless of the operating system. Java source code is compiled to Java bytecode and run in this Java Runtime Environment (JRE). The JRE consists of a Java Virtual Machine (JVM), as well as Java core classes and supporting libraries (Oracle, 2014a). This dependency on the JRE allows Java the platform independence that it is known for.

The common usage of the term Java refers to Java Standard Edition (Java SE) (Oracle, 2012). Java SE is intended for the development of stand-alone applications and web applets (Oracle, 2012). Java SE applications run in the JRE of the machine calling the application, while applets run in the JRE of the client machine. Java SE applications can be developed using only a text editor and the Java Development Kit (JDK) provided by Oracle (Oracle, 2017).

Java Enterprise Edition

Java Enterprise Edition (Java EE) is an extension of Java SE that is focused towards large-scale network applications (Oracle, 2012). Java EE provides several supporting libraries in addition to those provided in Java SE, with most of these libraries relating to server-side network services (Heffelfinger, 2014). Java EE applications typically cannot be run as stand-alone applications and must be run inside of a Java EE compliant application server (Heffelfinger, 2014). These application servers are themselves standalone applications that provide the environment for the Java EE application and provide access to the applications to clients across the internet (Heffelfinger, 2014). Development of Java EE applications requires a text editor and Oracle's JDK, similar to Java SE development, and also requires an application server (Heffelfinger, 2014).

Android Java

Android applications are written using the Java language just as Java SE and Java EE applications are, however the source code is compiled to Dalvik bytecode and run on a proprietary virtual machine (Friesen, 2013). Google provides the tools to compile to Dalvik byte code and additional support libraries focused towards the development of mobile applications in the Android SDK. Meanwhile, the virtual machine is a component of the Android Operating System and is present on every Android device. Development of Android applications requires Oracle's JDK and an extensive list of additional tools, all of which are provided with the Android Studio IDE (Android, 2017).

C using the Java Native Interface

There are two advantages of using native code within a Java application. First, native code may execute faster than Java code; and second it allows for the use of existing native libraries that may not be available in Java (Liu, 2013). Native code can replace Java code in long running processes, however this should be carefully considered as the gains are often minimal and there is an overhead cost when switching between Java and Native code (Liu, 2013). Utilizing native code also adds additional complexity to the application as both a native interface and a Java interface must be implemented.

The Java Native Interface (JNI) provides Java applications with access to native code (C, C++, Fortran, etc.) (Husaini, 1997). JNI is a component of the JDK provided by Oracle and is accessible within each of the Java flavors discussed previously. However, C code is not platform independent and must be built differently depending on the operating system and chipset of the computer that will be accessing it (Liu, 2013). Many tasks are performed during the building of C code, including preprocessing, compiling, assembly, and linking; the set of tools that performs this process is called a toolchain (Liu, 2013).

Within a Windows environment the C code must be built to a Dynamic Link Library (DLL) file. This project utilizes Microsoft's Visual C++ (VC++) toolchain to build the DLL file used in the Windows environment components. VC++ was selected primarily because it is the default native toolchain of the Gradle Build Tool (Gradle, 2017).

C using the Native Development Kit

Android's Native Development Kit (NDK) is an optional component of the Android SDK (Android, 2017). It is built specifically for utilizing native code within android applications. Some of the NDK is built utilizing the JNI, however the NDK utilizes the CMAKE build tool and a toolchain named Clang which is not supported by the Gradle build tool (Android, 2017a; Gradle, 2017).

The Android Operating System is built on a Linux kernel, therefore native libraries are built to Shared Object (SO) files, as they are for all Linux environments (Liu, 2013). Additionally, the NDK is built utilizing an older version of the JNI, and several JNI method calls have changed between the JNI version in the NDK and the JDK, making the creation of a cross-platform native source code base complicated.

4.1.2 Integrated Development Environments

Applications can be developed in Java, C, and many other languages by using a simple text editor and a set of command line tools. However, software development is typically performed within an Integrated Development Environment (IDE) (Orso, 2014). An IDE is an application that is designed to assist and expedite the software development process by providing tools such as intelligent code editors, version control systems, build tools, test suites, and debuggers, all in one package.

IntelliJ IDEA

IntelliJ IDEA is a Java IDE produced by JetBrains s.r.o. that facilitates the development of Java SE, Java EE, and other applications (JetBrains, 2017). It provides all the tools mentioned previously and several others, however it delegates the build process to one of the bundled external build tools Maven, Gradle and SBT (JetBrains, 2017). Gradle would be build tool of choice, due to familiarity, however Maven is the default build tool for Java EE applications and the configuration required to utilize Gradle is exhaustive, outweighing the benefits of such a change.

Android Studio

Android Studio is the standard IDE, provided free by Google, for developing Android applications (Android, 2017). It is built on the IntelliJ IDEA platform with reduced functionality, as it is specifically for creating Android applications. Among the reductions, is the limitation of using only the Gradle build tool (Silver & Vieira, 2015).

Although Android Studio is only intended for the development of Android Applications, Gradle can be reconfigured to allow for the development of Java SE applications and libraries (Silver & Vieira, 2015). Gradle can also be configured to build native code using the VC++ toolchain, allowing a Java SE application and the accompanying native library to be built together (Gradle, 2017). Gradle is also able to trigger the CMAKE build tool to build native code for an Android application, but only within an Android project.

4.1.3 Runtime Environments

As discussed previously, Java SE code is run within a JRE that is provided by the manufacturer of the machine. The JRE is ubiquitous and needs no further discussion, however the runtime environments for Java EE and Android Java code do warrant further consideration.

Glassfish Web Server

Glassfish is an open source application server sponsored by Oracle (Heffelfinger, 2014). Glassfish is the reference implementation for Java EE, meaning that it is the first application server to implement Java EE API's as they are released (Heffelfinger, 2014). Glassfish 5.0 is used in this project, which is the reference implementation of Java EE 8.

Glassfish is itself a Java SE application that runs in the JRE of the host machine. Web applications are built to Web ARchive (WAR) files by the IDE, then are provided to Glassfish (Heffelfinger, 2014). WAR files are deployed within a Web Container that manages the life cycle of the application and provides the runtime environment (Pilgrim, 2013).

Glassfish can deploy one or more applications simultaneously within the web container, as instructed by the user (Pilgrim, 2013). Glassfish manages a set of HTTP listeners and sets of connection pools to the network and databases (Heffelfinger,

2014). It supports regular HTTP and secure HTTPS connections. When it receives an HTTP request, Glassfish can check the authentication parameters of the request against lists of authorized users and assign or reject connections as appropriate (Heffelfinger, 2014). When the web application requires access to a database, Glassfish provides the database connection and the required authentication parameters (Heffelfinger, 2014).

Android Platform

Android is an open source software project supported by Google (Android, 2017b). It provides a operating system designed for a variety of mobile devices, based on a Linux kernel (Android, 2017b). Android is built as a software stack where lower layers are configured by device manufacturers and higher layers can run without concern for implementation at lower layers (Android, 2017b). Android applications run within the virtual machine with the support of the application framework that provides the API's necessary for the application to access the environment (Android, 2017c).

Within the Android Operating System, the virtual machine is a core component and most Android applications are expected to run within it (Android, 2017c). Android 1.0 to 4.4 utilized the Dalvik virtual machine, which was replaced by the Android Runtime virtual machine in Android 5.0 (Android, 2017c). The Android Runtime virtual machine still utilizes Dalvik byte code, however compatibility issues do arise occasionally (Android, 2017c).

The Android environment provides applications with private storage (Android, 2017c). Files written to this private storage cannot be accessed by other applications, or directly by the user (Android, 2017c). This is the default location for file storage, and is commonly used for internal configurations, preferences and databases (Android, 2017c). Data can also be stored and accessed outside of the application's private storage but User permission is required. There is a trade-off to storing data outside the private storage as data stored outside can be accessed by any application, but all data in private storage is lost if the application is ever uninstalled (Android, 2017c).

Android applications have a minimum SDK setting, which refers to the oldest version of Android that that the application can be run on (Android, 2017). This minimum SDK setting determines, or is determined by, the application framework components that the application can access. Due to the layered nature of the Android platform, applications only need to be concerned with the hardware configuration of the device to confirm whether or not specific components exist (Android, 2017c). Typically, if a device contains the necessary hardware components and meets the requirements of the minimum SDK, it will run the application.

4.1.4 Databases

Relational database systems are the industry standard tool for the long-term persistence of data (Keith & Schincarlol, 2006). As this project is primarily concerned

with data management, a database system is all but required. Numerous relational database implementations are available, though most have the similar core functionality and interfaces due to standardization (Keith & Schincarlol, 2006). Thanks to abstraction in the Java Persistence API (JPA) the vendor specific components of various implementations are hidden, and implementations become largely interchangeable (Keith & Schincarlol, 2006).

Java DB

Java DB was selected as the relational database system for this project by default, because it comes packaged and preconfigured with Glassfish (Heffelfinger, 2014). The apparent interchangeability of database systems operating under the JPA makes the time and effort required to research, select, and configure another database system unwarranted.

Java DB is Oracle's release of the Apache Derby open source project (Oracle, 2014). Java DB, and Apache Derby, is developed entirely in Java SE and runs in the Java Virtual Machine (Oracle, 2014). As a relational database, Java DB utilizes Standard Query Language (SQL) for inputs and outputs, specifically the "entry level SQL-92" standard plus some standardized and vendor specific higher-level features (Oracle, 2014). Java DB is supported by both the Java Database Connectivity (JDBC) and Java Persistence APIs (Heffelfinger, 2014).

Java DB also supports several security methods including database encryption, authorized users, table ownership, and read/write permissions, however none of these are enabled in the default configuration (Oracle, 2014). During the creation of a database an encryption method and a boot password can be added, and from that point on the database can only be opened by providing the boot password (Oracle, 2014). However, this provides minimal security as once the database is open it can be accessed by others without requiring the password (Oracle, 2014).

The username provided when creating the database becomes the administrator and becomes the only user able to create other users and grant read/write permissions (Oracle, 2014). Once a new user has been granted write permission they can create new tables and receive ownership (Oracle, 2014). Said user is then able to grant access of owned tables to other users as necessary, although the administrator can access any tables regardless of ownership (Oracle, 2014).

SQLite

SQLite is a library that provides a light-weight self-contained SQL database engine (SQLite, 2017). It is run within other applications and frameworks and cannot be run as a separate service (SQLite, 2017). It can be optimized for low memory environments such as mobile device, although this optimization may sacrifice some higher

functionality (SQLite, 2017). SQLite stores the entire database in a single file that is transferable across operating systems and system architectures (SQLite, 2017).

The SQLite library is embedded within Android and the application framework contains APIs to provide applications with access to SQLite tools (Android, 2017c). Android applications can create one or more databases within their private storage and can control access to these private databases (Android, 2017c).

4.1.5 Software Components

Efforts spent reinventing the wheel should be avoided whenever possible; a statement as true in software development as it is in automobile design. Software engineering principals strongly advocate both the development of reusable code and the use of existing code at every opportunity (Orso, 2014). With this in mind, this project utilizes several existing libraries, APIs, and architectures. The section will discuss the most prominent of these existing components.

RESTful Web Service

REpresentational State Transfer (REST) is a set of constraint placed on the design of a web service to promote the scalability and maintainability of the service (Sandoval, 2009). The general implementation of these constraints is said to be RESTful Architecture, while a web service that implements these constraints is said to be a

RESTful web service (Sandoval, 2009). The essence of these constraints is that the client-server interactions should be stateless, server resources should be identified by unique addresses, data should be reduced to a representation before being transmitted, the system should be layered and should support data caches (Sandoval, 2009).

Jersey is a Java support library and is the reference implementation of the Java API for RESTful web services (JAX-RS) (Sandoval, 2009). Jersey allows developers to quickly implement a RESTful web service by adding specific annotations to basic Java code (Mehta, 2014). These annotations are read in at compile time and are used to configure the web application (Sandoval, 2009). The web application module of this project uses JAX-RS 2.0 to expose functions to the web for use by the client modules. The majority of client-server communications are facilitated by the RESTful web service components of the web application.

WebSockets

WebSockets are a relatively new communication method introduced to the web in HTML5 (Pilgrim, 2013). They allow for sustained full-duplex connections, meaning full two-way communication between the participants for as long as required (Pilgrim, 2013). In standard client-server interactions, the connection must be initiated by the client, then the response sent by the server (Kalali & Mehta, 2013). The server has no

capacity to send new information to the client, unprompted, in standard interactions. Several technologies have been developed to solve this problem, including Long Polling; Publish-Subscribe; WebHooks; and Server Sent Events, however none of these solutions offer the full-duplex communication of WebSockets (Kalali & Mehta, 2013; Mehta, 2014; Pilgrim, 2013).

This project uses WebSockets to facilitate the communication between the desktop client and the server for the data administrator functionality. These functions require significant back and forth communication and may exceed the timeout duration of standard HTTP connections. The Java EE implementation of WebSockets is javax.websocket which allows developers implement both WebSocket Clients and Servers (Pilgrim, 2013). This library is used within on the server side, however as the desktop client is developed as a Java SE application a third-party library, Java-WebSocket, is used.

JSON Binding

As stated above, data must be reduced to representations for transmission in a RESTful architecture. This project uses JavaScript Object Notation (JSON) format for representing data objects. JSON is a lightweight data exchange format that was introduced to replace XML when the highly-structured nature of XML is not required (Kalali & Mehta, 2013). JSON is less verbose than XML, requiring less data packets sent

and less processing on both ends, to transmit the same information (Kalali & Mehta, 2013). The JSON representation of an object, or group of objects, is a string with specific formatting and key pairs to describe the information (Sandoval, 2009).

Binding refers to the use of tools to automatically convert Java data objects to a representation and vice-versa. The web application module of this project uses the Java API for XML Binding (JAXB) for JSON binding. JAXB is another support library provided in Java EE and, despite the name, it is capable of JSON binding in addition to XML binding (Heffelfinger, 2014). Neither the desktop client or the mobile client has access to this library however, as is its another Java EE library. Both the desktop and mobile client use a third-party library, Jackson, for JSON binding.

Object Relational Mapping

Within Java, datatypes are created as Classes and data is stored as Objects that are instances of said Classes (Keith & Schincarlol, 2006). The Object-Oriented model used by Java, and many other programing languages, is very powerful but can become very complex (Keith & Schincarlol, 2006). Relational databases store data in tables, they also store the table definitions in tables; one could say it is tables all the way down (Keith & Schincarlol, 2006). The difference between the Object model and Relational model is called the Impedance Mismatch and the methods of relating the two are called Object-

Relational Mapping (ORM) (Keith & Schincarlol, 2006). Difficulties with ORM arise because both models have features that have no parallel in the other model.

Several software vendors independently developed ORM tools that allow developers to define an ORM, then provide automatic conversions between the models, prior to Java EE setting standards (Keith & Schincarlol, 2006). The standards eventually created for Java EE are called the Java Persistence API, and several vendor-specific ORM libraries now conform with it (Keith & Schincarlol, 2006). The library chosen for the server module of this project is the EclipseLink Library. None of the light-weight mobile application ORM packages that were explored were able to handle class inheritance so no ORM is used for the mobile application.

GNSS Processing

GNSS technology and GNSS processing is a complex subject and is the focus of a considerable sum research. The capacity to process GNSS observations would be a considerable boon to this project, however the resources required to develop that capacity from the ground up would be prohibitive. As such, this project utilizes an GNSS processing library named RTKLIB. RTKLIB is an open-source C library with extensive GNSS processing options (Takasu, 2013). In order to utilize RTKLIB the system relies on the JNI and NDK tools described in section 4.1.1.

4.2 Conceptual System Design

During the inception phase of the project several artifacts are developed, including: a set of system-level use cases, a set of system requirements, and a system architecture diagram. Later, during the elaboration phase, several more artifacts are developed, including: sets of module-level use cases, sets of module requirements, and module architecture diagrams. These artifacts comprise the system-level and module-level conceptual designs.

The system-level use cases are developed around 5 actors: a legal surveyor, an engineering surveyor, a drafter, a land surveyor, and a data administrator; each a potential user. Approximately 20 use cases are developed, varying in length from a few steps to over 50 steps, to describe the expected interactions between the actor and the system. From these use cases, a system requirements artifact is developed containing 16 requirements for the system, stating that the system shall:

- Contain data in a central repository
- Be accessible from desktop computers
- Be accessible from mobile devices
- Authenticate clients
- Interface with surveying instruments
- Record surveying measurements even with no connection to central repository

- Keep an event log
- Calculate geometries from surveying measurements
- Calculate geometries from cogo
- Transform between time varying coordinate systems
- Integrate surveying measurements from different equipment
- Integrate surveying measurements from different epochs
- Extract integrated geometries
- Link external document / image files to jobs
- Recalculate from event log
- Read write to common file formats

From these system requirements, a system architecture (Figure 4.2) is developed.

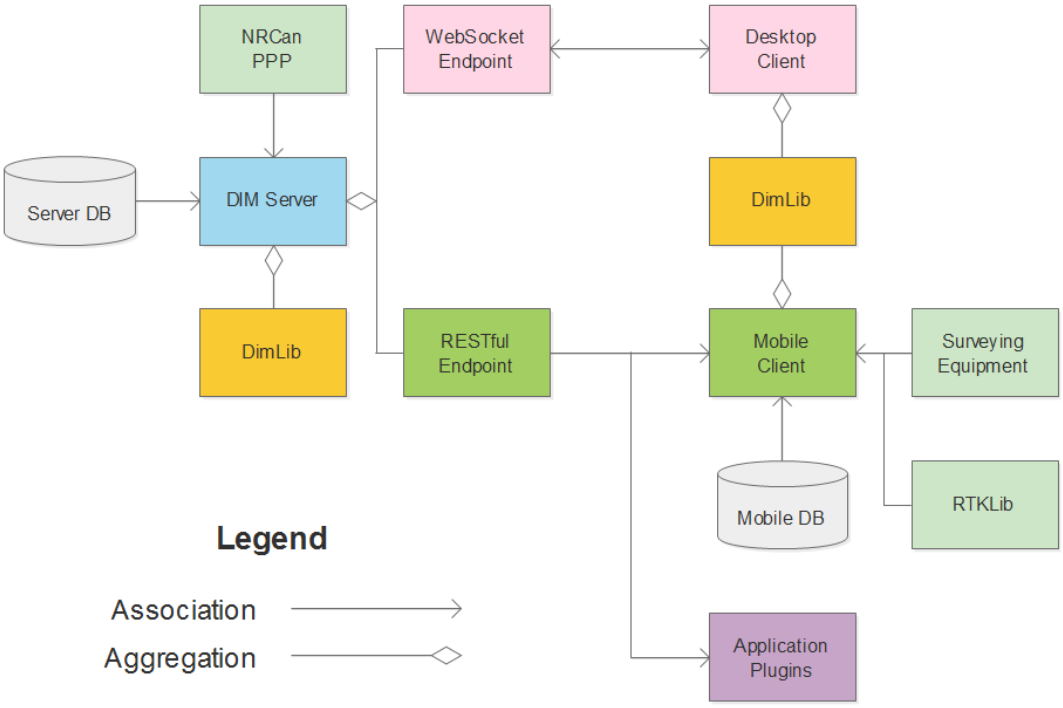

Figure 4.2: Preliminary System Architecture

Within this system architecture is 5 modules: a server, a mobile client, a desktop client, a set of application plugins, and a geodetic library. The server manages the central database and provides endpoints for the clients and application plugins to access services. The mobile client provides access for field surveyors, it is a heavy client that contains its own database and interfaces with survey equipment. The desktop client provides access for the data administrator and land surveyor to manage the server and database. The application plugins provide access for drafters to extract data from the server into CAD or GIS applications. The server, mobile client, and desktop client all depend on the geodetic library to define data types and perform geodetic calculations.

4.2.1 Server Design

No module-level use cases are developed for the server module as the system-level use cases already cover all the use cases that were identified for the server. From the system-level use cases, the module requirements are determined as the server shall:

- Manage the central database
- Communicate with clients using REST
- Communicate with clients using WebSockets
- Authenticate clients
- Send and receive geodetic library objects to clients via JSON
- Send and receive PDF / Image files with clients
- Process PPP data

The server module architecture (Figure 4.3) is then created to satisfy the module requirements.

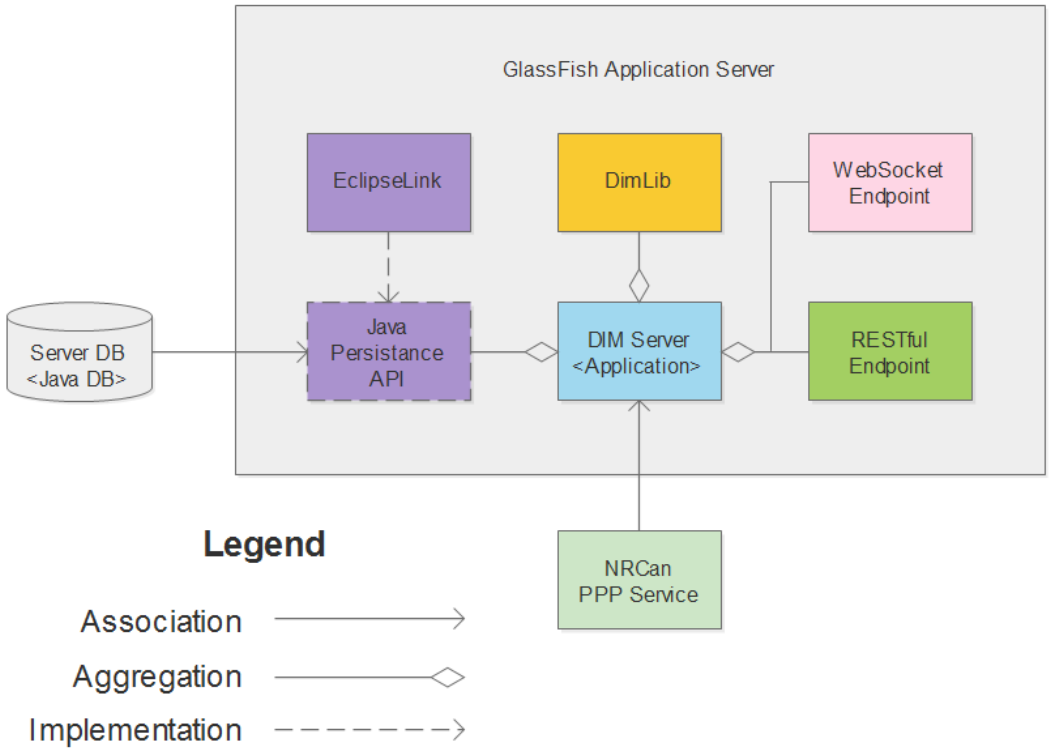

Figure 4.3: Preliminary Server Architecture

The server run inside of the Glassfish Application Server discussed in section 4.1.3. It uses the Java Persistence API (JPA) to interface with the Java DB service that comes bundled with Glassfish. JPA manages persistence and retrieval of the geodetic library datatypes provided to it by the server. The server provides WebSocket and RESTful end points, both requiring HTTPS connections and basic HTTP authentication headers. Authorization values are stored in a separate scheme in the database. Processing of PPP is provided by accessing the NRCan PPP service.

4.2.2 Mobile Client Design

Once again, the system-level use cases sufficiently cover the use module use cases, so no further use cases are developed. Module requirements are defined such that the mobile client shall:

- Run on Android operating system mobile devices
- Store geodetic library datatypes in a local database
- Communicate with the server's RESTful endpoint
- Send and receive geodetic library datatypes to server via JSON
- Communicate with surveying equipment using Bluetooth
- Record survey measurements
- Display geometry objects on a map screen
- Access device's GPS sensors
- Access internet for atmospheric and temporal data
- Provide access to geodetic library transformation functionality
- Provide access to geodetic library measurement processing functionality
- Provide access to geodetic library Cogo functionality
- Provide access to geodetic library event log functionality
- Provide access to geodetic library Least Squares Functionality
- Provide in-textbox Cogo Functionality

From these module requirements a mobile client architecture (Figure 4.4) is developed.

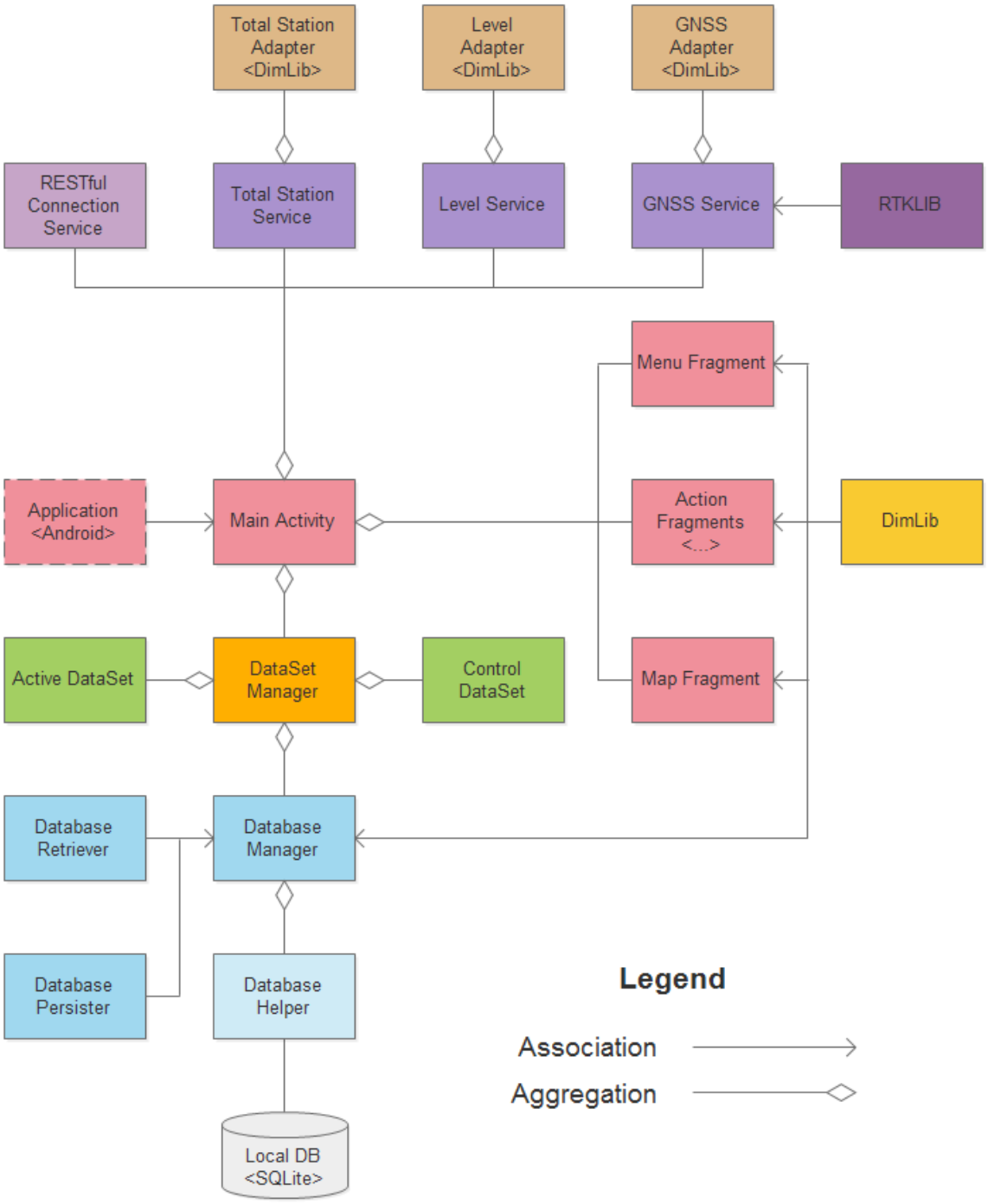

Figure 4.4: Preliminary Mobile Client Architecture

Android applications are developed as one or more Activity Classes that are linked by an Application Class hidden in the Android Framework. Older versions of Android required an Activity Class for each User Interface (UI), but more recent versions of Android allow developers to use Fragment Classes to display UIs. This is now the preferred method of development as fragments provide greater flexibility (Android, 2017). Therefore, the module uses a single MainActivity Class which can host a variety of fragments. Numerous fragments are required to display UI's to provide users access to geodetic library functions.

Several layers of abstraction are placed between the MainActivity and the local database because no Android compatible Object Relational Mapping (ORM) package was found that could handle Java class inheritance. A DataSetManager holds an active dataset and optionally a control dataset. These datasets are data structures that hold lists of linked geodetic library datatypes. The DataSetManager calls on the DatabaseManager to persist or retrieve DataSets. The DatabasePersister and DatabaseRetriever Classes contain what amount to stored procedures for persisting or retrieving individual datatype objects. The DataSetManager uses these classes to manage the individual data objects while it manages the ordering and linking of objects to assemble the DataSet. The DataBaseHelper manages table definitions and is a requirement of Android's interface with SQLite.

Android Services are background processes that can be run on separate threads. These are used for all external connections to the server or survey instruments so that connections will not be disrupted when the UI changes. Separate services are created for each of the survey equipment types, and each service relies on an Adapter Class provided by the geodetic library to translate manufacturer specific communication strings. The GNSS service also relies on RTKLIB to process GNSS measurements.

4.2.3 Desktop Client and Application Plugin Design

The desktop client and application plugin modules are both far beyond the scope of this project and did not receive the same level of consideration as the other modules. Both were considered as requirements for the system level design, but neither received use case development or architecture diagrams. Module-level requirements were developed for the desktop client module and it shall:

- Run on Windows or Linux operating system desktop computers
- Communicate with the server's WebSocket endpoint
- Send and receive geodetic library datatypes to server via JSON
- Display geometry objects on a map screen
- Provide access to geodetic library functionality
- Provide access to database authorization tables
- Provide access to network integration functionality

4.2.4 Geodetic Library Design

As the Geodetic Library is the focus of this project it received the most attention and the highest fidelity design. The module-level use cases are centered around 3 actors, each another module: the server, the mobile client, and the desktop client. 16 use cases are developed to describe expected interactions between the geodetic library and the actors. From the use cases it is determined that the geodetic library shall:

- Contain data objects to describe survey instruments
- Contain data objects to describe measurements
- Contain data objects to describe geometric objects
- Support layers for geometric objects
- Support a code / description / attribute system
- link numerous points into super-points
- Perform geodetic corrections and reduction of measurements
- Propagate uncertainty from measurements to points
- Support a variety of standard geodetic reference systems
- Generate local coordinate reference systems
- Transform points between time-varying cartesian coordinate systems
- Transform coordinates between cartesian, geodetic and projection systems
- Perform coordinate geometry functions
- Perform 1D, 2D and 3D least squares adjustments in projection space

- Support a variety of adjustment constraints
- Perform 2D similarity transformations
- Store all events in an event log
- Reprocess data from the event log
- Read and Write coordinate files
- Read and Write measurement files

An architecture diagram is developed from the module requirements, but the diagram is large and complex. The module-level architecture diagram (Figure 4.5) is organized into package-level class diagrams which are each displayed with the appropriate package discussions in in section 4.3.

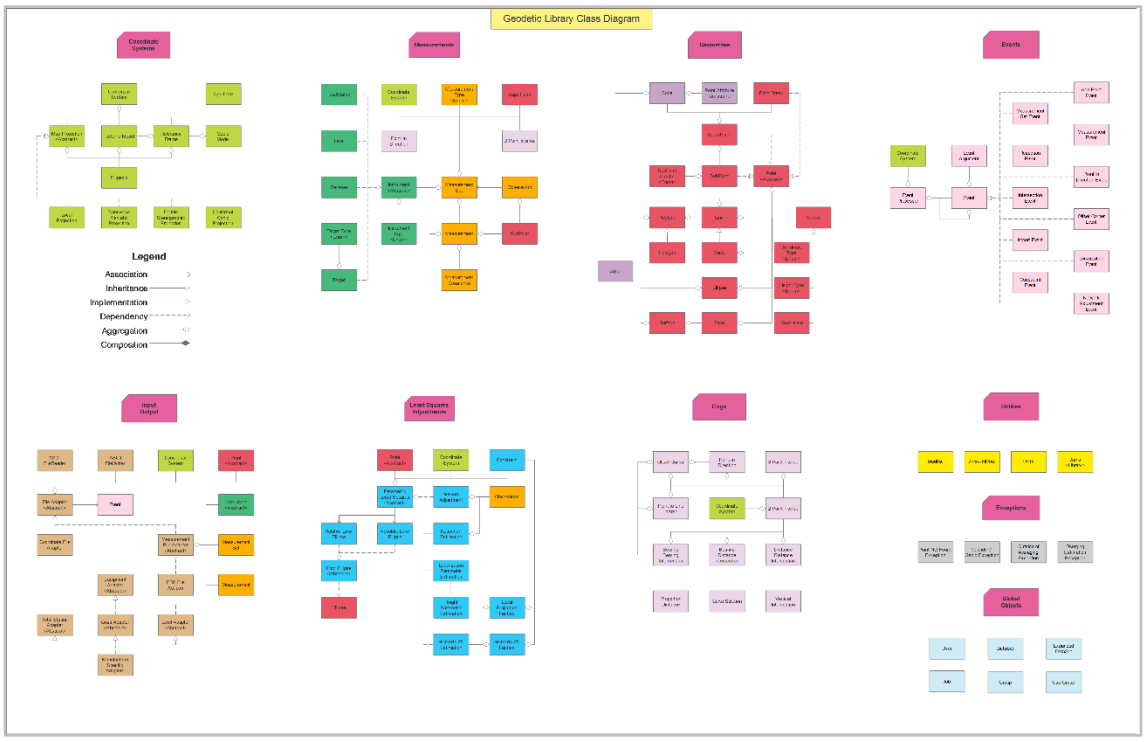

Figure 4.5: Overall Library Architecture Diagram

4.3 Geodetic Library Implementation

The Geodetic Library is the foundation of the software system. It provides the data type definitions and processing tools used by each of the other components of the system. Implementation of the geodetic library is necessary to performing the integration testing and completing the research objectives of the project. Due to its size and the importance of its role, providing support to the rest of the system, this library required the most rigorous design and its implementation is the longest phase of the project.

The library was designed as several packages with varying degrees of interdependence. Each package consists of a set of Java classes with related purposes or functions. Packages can be loosely classified as actor packages or data type packages, although in the object-oriented programming model that Java enforces the distinction can be nebulous. The primary responsibility of actor packages is processing of data. Many of the classes in these packages contain numerous methods, performing complex logic operations or extensive calculations. The primary responsibility of data type packages is the retention of data. Classes in data types packages tend to be shorter, with most methods relating to the organization and transfer of data. Actor packages therefore receive more attention in this section, with rigorous discussion of the logic and calculations performed. Discussion of data type packages is mostly limited to purpose and design considerations.

One class level diagram was created for the entire Geodetic Library (Figure 4.5). Unfortunately, it is illegible when displayed in its entirety as it requires a 34” x 22” page size. It is organized as a set of package-level class diagrams so that these class diagrams can be extracted from the full architecture diagram and displayed in each of the following sub-sections. The legend (Figure 4.6) is extracted from the full architecture diagram and applies to each of the extracted class diagrams. Classes within the diagrams are color-coded by the package they belong to.

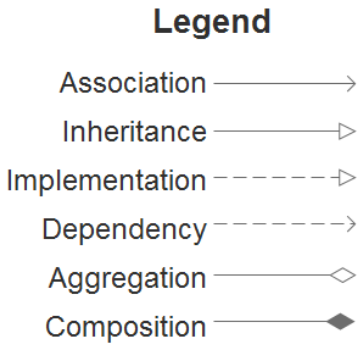

Figure 4.6: Class Diagram Legend

The original architecture diagram was created during the elaboration phase and was used as a reference for the implementation of the library. However, during the implementation of nearly every package, deviations from the original design were required. The deviations from the original design were used to update the architecture diagram during each development cycle. Therefore, the class diagrams displayed in the subsequent sub-sections reflect the library as it was implemented.

4.3.1 Geometries and Descriptors Packages

The Geometries Package is a data types package. It was the first package implemented as it is critical to the functionality of many of the other packages. The Geometries Package (Red) and the Descriptors Package (Purple) are displayed in the same class diagram (Figure 4.7) due to the high interdependence between them.

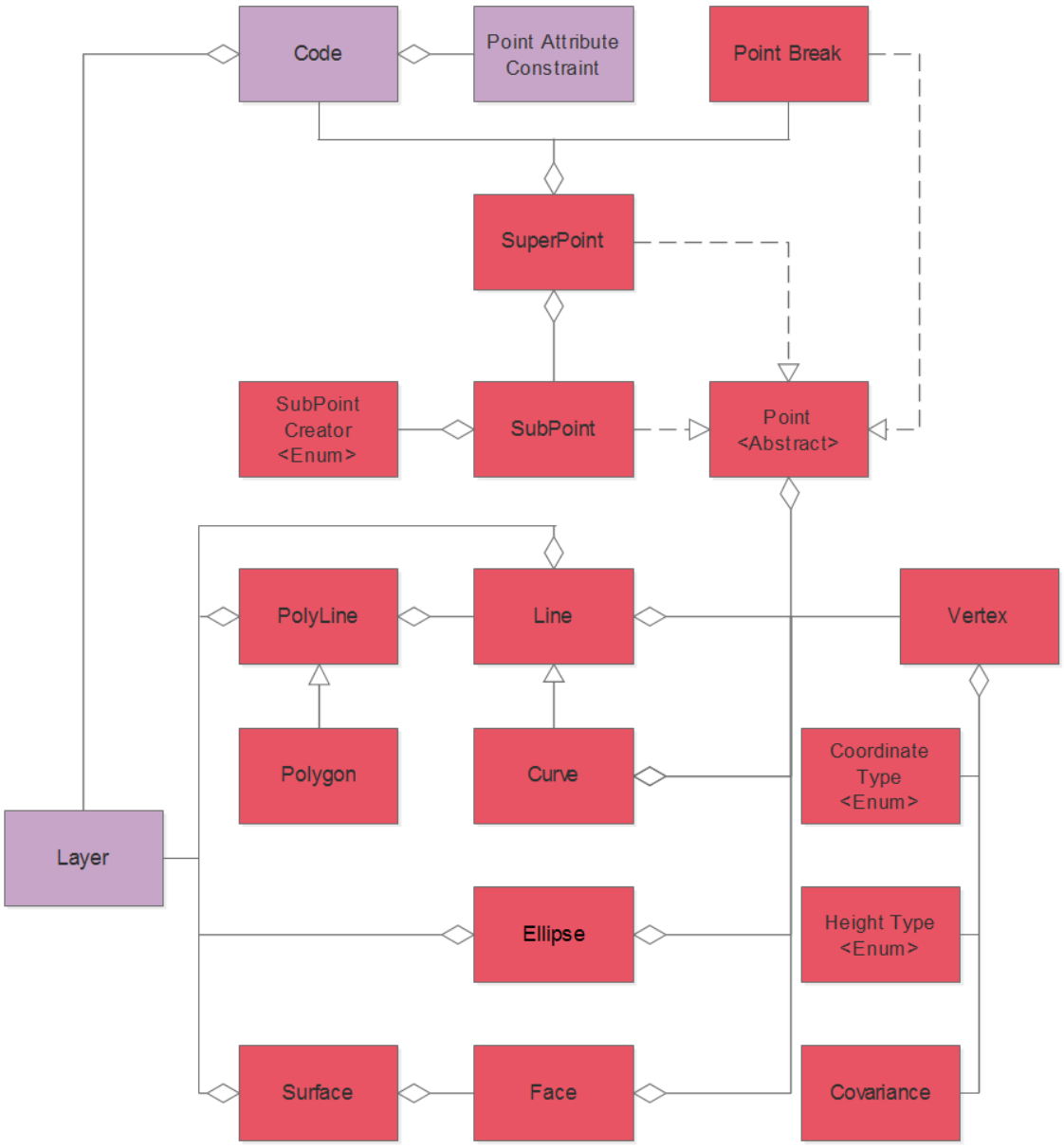

Figure 4.7: Class Diagram of the Geometries and Descriptor Packages

The foundation of the geometries package is the Vertex Class. The Vertex represents a set of coordinates and contains descriptors for the types of coordinates type, height type and epoch of said coordinates. All other geometry objects contain one or more vertices. The Point Class contains one vertex and the Super Point, Sub Point, and Point Break Classes inherit this single vertex from the Point Class. An ellipse object also contains one vertex. A line object contains two vertices while curve and face objects contain three vertices. Polyline and polygon objects are composed of line and curve objects and only contain vertices indirectly, just as surface objects do, which are composed of face objects.

A vertex created for one geometry object can also be passed to another geometry object, linking the coordinates of the two objects. For example, a vertex is created for a super point by taking an observation of a post at a property corner. The same vertex is then used in the creation of a line object representing the property boundary between the lots. Later, a least squares adjustment is performed, changing the coordinates of the super point. Because the super point and line share the vertex, the coordinates of the end of the line are also changed.

The relationships between the Point, Super Point, Sub Point, and Break Point Classes may appear convoluted in the class diagram, however these relationships are necessary to describe the complexity of some real-world situations. A super point

describes a location in the real-world and as such is the only point implementation that contains descriptors. A sub point describes a single estimate of a location, where an estimate is obtained from a measurement or calculation. A super point must contain one or more sub points. Coordinates of a super point are obtained by averaging the coordinates of it's sub points, using a hierarchy of creation methods of the sub points.

A point breaks describes the coordinates of a super point up to a specific epoch. A new super point should contain zero point breaks and may never obtain one. However, if the physical marker of a super point is disturbed (as they often are near construction projects) then a point break can be added to separate the coordinates prior to the disturbance from the coordinates after the disturbance. The point break is created with the averaged coordinates of all sub point prior to the disturbance, while the super point receives the post disturbance coordinates. When the coordinates are requested from a super point, said super point compares the epoch of the request against it's list of point breaks and returns the coordinates valid at that epoch.

Super Points also contain a code object and up to three attribute strings. The code object describes the point type and contains a list of required attributes. When a point is created with a given code, all required attributes must also be added to the point. Code objects and all non-point geometry objects also contain a layer object, which is used for display and querying purposes.

4.3.2 Coordinate Systems Package

The Coordinate System package is the most fundamental of the actor packages. Its responsibilities include performing the transformation of coordinates between Cartesian, Geodetic and Projection systems, performing transformations between reference frames and across epochs, and providing geodetic corrections for measurements. The class diagram of the coordinate system package (Figure 4.8) appears simple, however the simple class structure of this package obscures the complexity within many of the classes. As such, an in-depth discussion of each class is required.

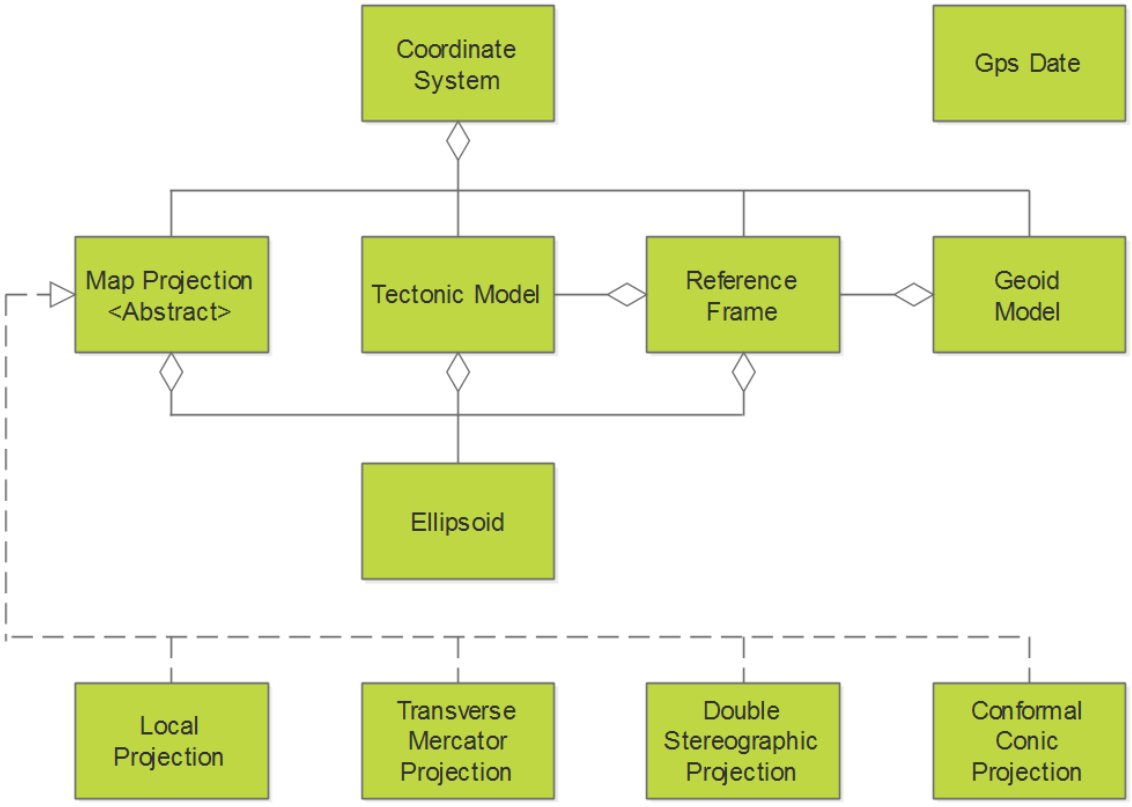

Figure 4.8: Class Diagram of Coordinate Systems Package

Coordinate System Class

The Coordinate System Class is the primary interface between this package and all other packages. It contains the usage logic of all the classes beneath it in this package. Each of the responsibilities of this package are fulfilled by the methods this class exposes to the rest of the library and the applications utilizing the library.

Gps Date Class

The Gps Date Class shows no connections to other classes on the class diagram, not because of a lack of usage, but rather an over abundance. Usage of the Gps Date Class is so ubiquitous that displaying the connections would clutter the class diagrams of nearly every package.

Java contains both a Date Class and a Calendar Class for the management of time. Unfortunately, these classes to not work together well and both have some odd behaviors and shortcomings. The Gps Date Class exists to simplify the tracking of time. It utilizes both of these classes, encompassing the logic to translate between them, and adds additional functionality, providing conversions between time systems and accounting for leap seconds. Leap seconds are corrected for, utilizing a list of the leap seconds announced by the International Earth Rotation Service (IERS), provided by the Internet Engineering Task Force (IETF, 2016).

The Gps Date Class provides transformations between several time systems (Figure 4.9), including: Universal Coordinated Time (UTC), International Atomic Time (TAI), Global Positioning System (GPS) time, Network Time Protocol (NTP), and Unix time. Input time values are converted to GPS time millisecond stamp for storage, as GPS time is unaffected by leap seconds therefore provides unique, continuous time values.

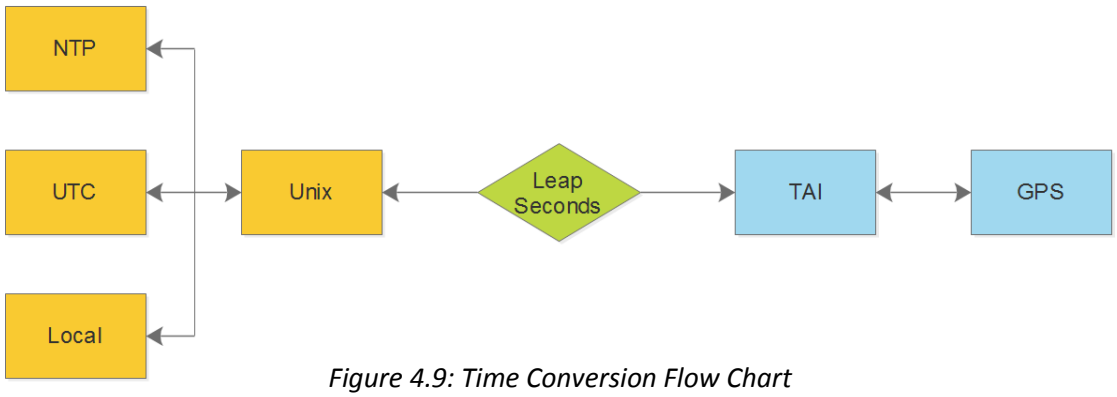

Figure 4.9: Time Conversion Flow Chart

Conversion from GPS time to TAI time uses a false time stamp and a constant value:

$$False = GPS - 315964819000\ msec \tag{4.1}$$

then the Java Date Class is used to convert from the false timestamp to a human time and date string. Conversion from TAI time to Unix again uses the false time stamp:

$$Unix = false - (leaps * 1000)\ msec \tag{4.2}$$

where leaps = the cumulative number of leap seconds at the time of interest.

Conversion from Unix time to NTP time again uses a constant value:

$$NTP = Unix + 2208988800000\ msec \tag{4.3}$$

Finally, conversions between Unix time, UTC time, and local time are provided by the Java Date Class.

Ellipsoid Class

The Ellipsoid Class stores four values that define the ellipsoid object. From these four values, all other values required for ellipsoidal computations are calculated. The values for the GRS80 and WGS84 ellipsoids, obtained from Hofmann-Wellenhof & Mortiz (2005), are hardcoded so that either of these ellipsoids can be instantiated with a single parameter. Other ellipsoids can be created by providing the defining values.

The primary responsibilities of this class include the transformation of coordinate and covariance matrices between cartesian (x, y, z) and geodetic (φ, λ, h) systems. The transformation of geodetic coordinates to cartesian is performed using the following equations from Torge & Muller (2012):

$$x = (N + h)\cos(\varphi)\cos(\lambda) \tag{4.4}$$

$$y = (N + h)\cos(\varphi)\sin(\lambda) \tag{4.5}$$

$$z = \left((1 - e^2)\,N + h\right)\sin(\varphi) \tag{4.6}$$

where N is the radius of curvature of the prime vertical section and e is the first eccentricity of the ellipsoid. The inverse transformation is performed using the following formulas from Torge & Muller (2012):

$$\lambda = \arctan\left(\frac{y}{x}\right) \tag{4.7}$$

$$\varphi = \arctan\left(\frac{z}{\sqrt{x^2+y^2}}\right)\left(1 - e^2\,\frac{N}{N+h}\right)^{-1} \tag{4.8}$$

$$h = \frac{\sqrt{x^2+y^2}}{\cos\varphi} - N \tag{4.9}$$

where λ is calculated once, then φ, h, and N are calculated iteratively, until the difference between successive calculations falls below a certain threshold.

The transformation of covariance matrices from geodetic is performed using the following equations reformatted from Thomson, Krakiwsky, & Steeves (1977):

$$c_{\varphi,\lambda,h} = \begin{bmatrix} \sigma_{\varphi}^2 & \sigma_{\varphi\lambda} & \sigma_{\varphi h} \\ \sigma_{\varphi\lambda} & \sigma_{\lambda}^2 & \sigma_{\lambda h} \\ \sigma_{\varphi h} & \sigma_{\lambda h} & \sigma_{h}^2 \end{bmatrix} \tag{4.10}$$

$$c_{x,y,z} = \begin{bmatrix} \sigma_{x}^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{y}^2 & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{z}^2 \end{bmatrix} \tag{4.11}$$

$$J = \begin{bmatrix} \frac{\partial x}{\partial \varphi} & \frac{\partial x}{\partial \lambda} & \frac{\partial x}{\partial h} \\ \frac{\partial y}{\partial \varphi} & \frac{\partial y}{\partial \lambda} & \frac{\partial y}{\partial h} \\ \frac{\partial z}{\partial \varphi} & \frac{\partial z}{\partial \lambda} & \frac{\partial z}{\partial h} \end{bmatrix} \tag{4.12}$$

$$c_{x,y,z} = J\, c_{\varphi,\lambda,h}\, J^t \tag{4.13}$$

and the reverse transformation is performed:

$$c_{\varphi,\lambda,h} = J^{-1}\, c_{x,y,z}\, (J^{-1})^t \tag{4.14}$$

The other responsibilities of the ellipsoid class include the reduction of ground measurements to the ellipsoid and the calculation of normal gravity values. These responsibilities are performed using the formulas found in Table 4.1

Table 4.1: Ellipsoid Class Formula Sources

Value	Source	Listing
First Eccentricity	(Hooijberg, 2008)	Table 19
Second Eccentricity	(Hooijberg, 2008)	Table 19
First Flattening	(Hooijberg, 2008)	Table 19
Second Flattening	(Hooijberg, 2008)	Table 19
Radius of Curvature of the Meridian Section	(Torge & Muller, 2012)	Eq 4.13
Radius of Curvature of the Prime Vertical Section	(Torge & Muller, 2012)	Eq 4.15
Surface Normal Unit	(Torge & Muller, 2012)	Eq 4.26b
Gaussian Radius of Curvature	(Krakiwsky, 1973)	Pg 133
Euler Radius of Curvature	(Thomson, Krakiwsky, & Adams, 1978)	Eq 3.4
Radius of Rectifying Sphere	(Stem, 1989)	Pg 33
Meridian Arc Length	(Stem, 1989)	Pg 33
Astronomic to Geodetic Azimuth Correction	(Thomson, Krakiwsky, & Adams, 1978)	Eqs 3.15 – 3.18
Astronomic to Geodetic Zenith Angle Correction	(Thomson, Krakiwsky, & Adams, 1978)	Eq 3.14
Geodetic Distance	(Thomson, Krakiwsky, & Adams, 1978)	Eqs 3.19 – 3.20
Normal Gravity at Equator	(Hofmann-Wellenhof & Mortiz, 2005)	Eq 2.186
Normal Gravity at Pole	(Hofmann-Wellenhof & Mortiz, 2005)	Eq 2.186
Normal Gravity on Ellipsoid	(Hofmann-Wellenhof & Mortiz, 2005)	Eq 2.146
Normal Gravity above Ellipsoid	(Hofmann-Wellenhof & Mortiz, 2005)	Eq 2.215

Reference Frame Class

The primary responsibilities of the of the Reference Frame Class are to indicate the reference frame that coordinates refer to, and to provide transformations to other reference frames. The first responsibility is fulfilled simply by the existence of a reference frame object within the coordinate system object. To perform its second responsibility, this class stores 15 values, being 7 transformation parameters, 7 velocities of the transformation parameters, and a transformation epoch. The transformation parameter for the ITRF2000, ITRF2005, ITRF2008, ITRF2014, NAD83(CSRS), and WGS84(G1674) reference frames are hardcoded so that any of these reference frames can be instantiated by providing a single parameter. Additional reference frames may be created by providing additional transformation parameters.

The hardcoded values for each of these frames are the transformation parameters to ITRF2008 and can be found in Table 4.2. The transformation parameters for NAD83 caused some issues due to a discrepancy between what ITRF considers the definition of ITRF96 and what NRCan considers the definition of ITRF96. The discrepancy was identified while attempting to replicate the transformation parameter found in Craymer (2006) and the missing values between ITRF97 and ITRF96 were found in Craymer (1998). The correct parameters were calculated by summing the transformation parameters between ITRF2008 & ITRF97 (ITRF, 2010), ITRF97 & ITRF96 (Craymer, 1998), and ITRF96 & NAD83 (Craymer, 1998) at epoch 1997.

Table 4.2: Transformation Parameters to ITRF2008

Frame	tx (mm)	ty (mm)	tz (mm)	s (ppb)	rx (mas)	ry (mas)	rz (mas)	Epoch (t_0)
Source	$\dot{t}x$ (mm/y)	$\dot{t}y$ (mm/y)	$\dot{t}z$ (mm/y)	$\dot{s}$ (ppb/y)	$\dot{r}x$ (mas/y)	$\dot{r}y$ (mas/y)	$\dot{r}z$ (mas/y)	
ITRF2014	1.6	1.9	2.4	-0.02	0	0	0	2010
(ITRF, 2016)	0	0	-0.1	0.03	0	0	0	
ITRF2008	0	0	0	0	0	0	0	2000
	0	0	0	0	0	0	0	
ITRF2005	2	0.9	4.7	-0.94	0	0	0	2000
(ITRF, 2010)	-0.3	0	0	0	0	0	0	
ITRF2000	1.9	1.7	10.5	-1.34	0	0	0	2000
(ITRF, 2010)	-0.1	-0.1	1.8	-0.08	0	0	0	
NAD83	-993.4	1903.3	526.5	-1.715	25.915	9.426	11.599	1997
(ITRF, 2010) (Craymer, 1998)	-0.8	0.6	1.3	0.102	0.0662	-0.7573	-0.0516	
WGS84 G1674	0	0	0	0	0	0	0	2000
(ITRF, 2013)	0	0	0	0	0	0	0	

When a transformation is called between two frames, transformation parameters are calculated for the epoch of interest (t), in both frames, using the function (ITRF, 2010):

$$\begin{bmatrix} S \\ T_x \\ T_y \\ T_z \\ R_x \\ R_y \\ R_z \end{bmatrix} = \begin{bmatrix} s \\ t_x \\ t_y \\ t_z \\ r_x \\ r_y \\ r_z \end{bmatrix} + (t - t_0) \cdot \begin{bmatrix} \dot{s} \\ \dot{t}_x \\ \dot{t}_y \\ \dot{t}_z \\ \dot{r}_x \\ \dot{r}_y \\ \dot{r}_z \end{bmatrix} \tag{4.15}$$

The transformation parameters are then combined by subtracting the transformation parameters of the new reference frame (j) from the parameters of the old reference frame (i):

$$\begin{bmatrix} S \\ T_x \\ T_y \\ T_z \\ R_x \\ R_y \\ R_z \end{bmatrix} = \begin{bmatrix} S \\ T_x \\ T_y \\ T_z \\ R_x \\ R_y \\ R_z \end{bmatrix}_i - \begin{bmatrix} S \\ T_x \\ T_y \\ T_z \\ R_x \\ R_y \\ R_z \end{bmatrix}_j \tag{4.16}$$

The cartesian coordinates in the new reference frame are then calculated using a Helmert's transformation (ITRF, 2010):

$$\begin{bmatrix} x_j \\ y_j \\ z_j \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} + (1 + S) \cdot \begin{bmatrix} 1 & R_z & -R_y \\ -R_z & 1 & R_x \\ R_y & -R_x & 1 \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \tag{4.17}$$

The Reference Frame Class also contains an embedded tectonic model to calculate the motion of the frame across epochs. The only frame that currently uses this feature is NAD83 which uses an embedded NNR-NUVEL-1A model.

Tectonic Model Class

The sole responsibility of the Tectonic Model Class is the transformation of cartesian coordinates from one epoch to another, within a given reference frame. In order to perform this task, the Tectonic Model Class stores three values, rotation about x, rotation about y and rotation about z. The rotation values for each plate in the NNR-NUVEL-1a and ITRF2008 tectonic models are hardcoded so that a new tectonic model

object can be generated with two parameters, model and plate. The values in Table 4.3 are for NNR-NUVEL-1A model and are converted to milli-arc seconds per year from the values in McCarthy (1996). Values for the ITRF2008 model in table 4.4 are obtained from Altamimi, Metivier, & Collilieux (2012).

Table 4.3: NNR-NUVEL-1a Plate Rotation Values

Plate	**Ωx (mas/y)**	**Ωy (mas/y)**	**Ωz (mas/y)**
Africa	0.184	-0.639	0.809
Antarctica	-0.169	-0.351	0.764
Arabia	1.379	-0.107	1.394
Australia	1.617	1.057	1.296
Caribbean	-0.037	-0.698	0.326
Cocos	-2.150	-4.456	2.253
Eurasia	-0.202	-0.494	0.650
India	1.376	0.008	1.401
Juan De Fuca	1.073	1.776	-1.200
Nazca	-0.316	-1.769	1.982
North America	0.053	-0.742	-0.032
Pacific	-0.311	0.998	-2.056
Philippine	2.081	-1.477	-1.995
Rivera	-1.937	-6.386	2.485
Scotia	-0.085	-0.549	-0.262
South America	-0.214	-0.312	-0.179

Table 4.4: ITRF2008 Plate Rotation Values

Plate	Ωx (mas/y)	Ωy (mas/y)	Ωz (mas/y)
Amurian	-0.190	-0.442	0.915
Antarctica	-0.252	-0.302	0.643
Arabia	1.202	-0.054	1.485
Australia	1.504	1.172	1.228
Caribbean	0.049	-1.088	0.664
Eurasia	-0.083	-0.534	0.750
India	1.232	0.303	1.540
Nazca	-0.330	-1.551	1.625
North America	0.035	-0.662	-0.100
Pacific	-0.411	1.036	-2.166
South America	-0.243	-0.311	-0.154
Nubia	0.095	-0.598	0.723
Somalia	-0.080	-0.745	0.897
Sundaland	0.047	-1.000	0.975

Cartesian coordinates are then transformed from the previous epoch (t_0), to the time of interest (t), using another Helmert's transformation (Soler & Marshall, 2003):

$$\begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} = \begin{bmatrix} x_{t_0} \\ y_{t_0} \\ z_{t_0} \end{bmatrix} + (t - t_0) \cdot \begin{bmatrix} 0 & \Omega_z & -\Omega_y \\ -\Omega_z & 0 & \Omega_x \\ \Omega_y & -\Omega_x & 0 \end{bmatrix} \cdot \begin{bmatrix} x_{t_0} \\ y_{t_0} \\ z_{t_0} \end{bmatrix} \quad (4.18)$$

Geoid Model Class

The primary function of the Geoid Model Class is to provide geoid undulation and deflection of vertical values. In order to perform this function, the Geoid Model Class must retrieve values from a geoid undulation file. It is able to extract values from Canadian geoid (.byn) files, American geoid (.bin) files, and the unnamed EGM08 geoid undulation files.

When instantiated a geoid model object must be provided the location of the geoid file. If a set of extents is also provided, then the object will immediately read the geoid file and will retain all geoid undulation values within the extents. If no extents are provided, then the object will wait to read the file until a request is made for geoid values. When a request is made, the object calculates extents 1° in each direction from the point of interest and reads the file. With each subsequent request, the object determines if the new point of interest is within the current extents, if it is not, then the object calculates new extents and reads the file again.

When reading the file, the object retains all metadata in the file header. With the BYN format one piece of included meta data is the reference frame. The geoid model object creates its own object of the Reference Frame Class corresponding with the reference frame dictated in the meta data. If the reference frame object contained within the geoid model object does not match the reference frame of the coordinate system

object making a request for geoid values, the geoid undulation is transformed from the geoid's reference frame to the coordinate system's reference frame.

Interpolation of geoid undulation uses the same bi-quadratic interpolation method adapted from Vanicek, Zhang, & Ong (1990) that NRCan uses in their GPS-H software (Véronneau, 2017). The nine closest points surrounding the point of interest are selected (Figure 4.10)

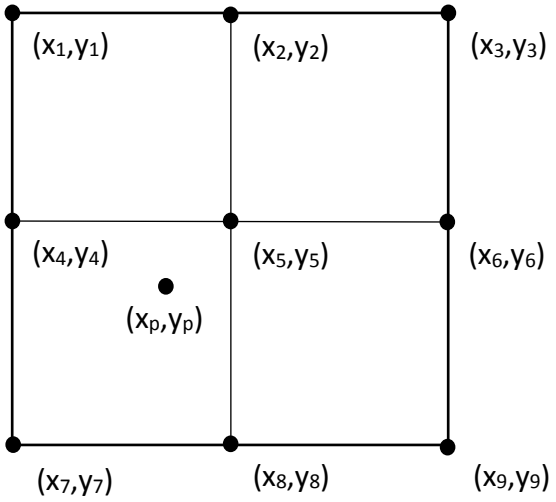

Figure 4.10: Interpolation Grid

x and y are calculated for all 10 points using the formulas:

$$x = R\,(\lambda - \lambda_5)cos\varphi_5 \tag{4.19}$$

$$y = R\,(\varphi - \varphi_5) \tag{4.20}$$

Where R is the Gaussian radius of curvature.

Each point on this surface can be described using the geoid undulation (N), x & y coordinates and 9 coefficients (C_i):

$$N = C_1x^2y^2 + C_2x^2y + C_3xy^2 + C_4x^2 + C_5y^2 + C_6xy + C_7x + C_8y + C_9 \quad (4.21)$$

Creating equations in the form of (4.21) for each of the grid points, we then define the following matrices:

$$L^t = [N_1 \quad N_2 \quad N_3 \quad N_4 \quad N_5 \quad N_6 \quad N_7 \quad N_8 \quad N_9] \quad (4.22)$$

$$X^t = [C_1 \quad C_2 \quad C_3 \quad C_4 \quad C_5 \quad C_6 \quad C_7 \quad C_8 \quad C_9] \quad (4.23)$$

$$A = \begin{bmatrix} x_1^2y_1^2 & x_1^2y_1 & x_1y_1^2 & x_1^2 & y_1^2 & x_1y_1 & x_1 & y_1 & 1 \\ x_2^2y_2^2 & x_2^2y_2 & x_2y_2^2 & x_2^2 & y_2^2 & x_2y_2 & x_2 & y_2 & 1 \\ x_3^2y_3^2 & x_3^2y_3 & x_3y_3^2 & x_3^2 & y_3^2 & x_3y_3 & x_3 & y_3 & 1 \\ x_4^2y_4^2 & x_4^2y_4 & x_4y_4^2 & x_4^2 & y_4^2 & x_4y_4 & x_4 & y_4 & 1 \\ x_5^2y_5^2 & x_5^2y_5 & x_5y_5^2 & x_5^2 & y_5^2 & x_5y_5 & x_5 & y_5 & 1 \\ x_6^2y_6^2 & x_6^2y_6 & x_6y_6^2 & x_6^2 & y_6^2 & x_6y_6 & x_6 & y_6 & 1 \\ x_7^2y_7^2 & x_7^2y_7 & x_7y_7^2 & x_7^2 & y_7^2 & x_7y_7 & x_7 & y_7 & 1 \\ x_8^2y_8^2 & x_8^2y_8 & x_8y_8^2 & x_8^2 & y_8^2 & x_8y_8 & x_8 & y_8 & 1 \\ x_9^2y_9^2 & x_9^2y_9 & x_9y_9^2 & x_9^2 & y_9^2 & x_9y_9 & x_9 & y_9 & 1 \end{bmatrix} \quad (4.24)$$

Where:

$$L = A\,X \quad (4.25)$$

The coefficients are then solved:

$$\mathrm{X} = A^{-1}\,L \quad (4.26)$$

The geoid undulation at P is solved using Eq (4.21) and deflections of vertical are calculated by the partial derivatives of Eq (4.21):

$$\xi = -\frac{\partial N}{\partial y} \quad (4.27)$$

$$\eta = -\frac{\partial N}{\partial x} \quad (4.28)$$

Finally, a small correction is applied to the deflection of vertical in northing to correct for the curvature of the normal plumbline at heights above the geoid, from Hofmann-Wellenhof & Mortiz (2005):

$$\delta\xi = -\frac{0.17}{206264.8}\left(\frac{h}{1000}\right) sin2\varphi \tag{4.29}$$

Map Projection Class

As seen in Figure 4.8, the Map Projection Class is annotated as abstract. In Java, an abstract class cannot be instantiated and is essentially a placeholder for other classes that implement the abstract class. In this case, the map projection object held by the coordinate system object becomes an empty socket that any of the map projection implementations can be placed into.

An abstract class may contain a list of abstract methods. These abstract methods are not implemented in the abstract class and are not capable of performing any actions. The contract between the abstract class and its implementations requires that all abstract methods listed in the abstract object be implemented in the implementing class. The Map Projection Class has six abstract methods:

- calcProjectedCoordinatesFromGeodetic(Matrix geodeticCoordinates)
- calcGeodeticCoordinatesFromProjected(Matrix projectionCoordinates)
- calcMeridianConvergence(Matrix geodeticCoordinates)
- calcProjectionScaleFactor(Matrix geodeticCoordinates)
- calcArcToChordCorrection(Matrix geodeticCoordsFrom, Matrix geodeticCoordsTo)

- calcGeodeticToProjectionJacobianMatrix(Matrix geodeticCoordinates)

The primary responsibilities of the map projection class are the transformation of coordinates, covariance, and observation between geodetic and projection space. The Map Projection Class implements these transformations expecting that any given map projection implementation will provide functions to replace the abstract methods.

The transformation of coordinates between geodetic ($\varphi, \lambda, \mathrm{h}$) and projection ($\mathrm{N,E,H}$) systems are implementation specific and will be discussed with the implementations. The transformation of covariance matrices and observations are performed by the Map Projection Class, relying on values from the specific implementation.

The reduction of geodetic azimuths to grid azimuth is performed (Thomson, Krakiwsky, & Adams, 1978):

$$t_{ij} = \alpha_{ij} - \gamma_i - (T - t)_{ij} \tag{4.30}$$

where t_{ij} = grid azimuth, α_{ij} = geodetic azimuth, Υ_i = meridian convergence, and $(T - t)_{ij}$ = arc to chord correction. The meridian convergence and arc to chord correction values are supplied by the specific implementation of the Map Projection Class.

The reduction of geodetic distances to grid distances is performed (Thomson, Krakiwsky, & Adams, 1978):

$$\ell_{ij} = \bar{k}_{ij}\, s_{ij} \tag{4.31}$$

$$\bar{k}_{ij} = \frac{k_i + k_j + 4k_m}{6} \tag{4.32}$$

where ℓ = grid distance, s = geodetic distance, k = projection scale factor, and k_m = scale factor at the midway point between i and j. the projection scale factor at each point is supplied by the specific map projection implementation.

Transformation of covariance matrices from geodetic to projection space is performed (Thomson, Krakiwsky, & Steeves, 1977):

$$c_{\varphi,\lambda} = \begin{bmatrix} \sigma_\varphi^2 & \sigma_{\varphi\lambda} \\ \sigma_{\varphi\lambda} & \sigma_\lambda^2 \end{bmatrix} \tag{4.33}$$

$$c_{N,E} = \begin{bmatrix} \sigma_N^2 & \sigma_{NE} \\ \sigma_{NE} & \sigma_E^2 \end{bmatrix} \tag{4.34}$$

$$c_{N,E} = J\, c_{\varphi,\lambda}\, J^t \tag{4.35}$$

and the reverse transformation is performed:

$$c_{\varphi,\lambda} = J^{-1}\, c_{N,E}\, (J^{-1})^t \tag{4.36}$$

where J is supplied by the specific map projection, but always follows the form:

$$J = \begin{bmatrix} \frac{\partial N}{\partial \varphi} & \frac{\partial N}{\partial \lambda} \\ \frac{\partial E}{\partial \varphi} & \frac{\partial E}{\partial \lambda} \end{bmatrix} \tag{4.37}$$

Transverse Mercator Projection Class

The Transverse Mercator Projection Class is the first implementation of the Map Projection Class. The class is sufficiently generalized as to allow for creation of UTM, 3TM and custom TM (as used in the state plane system) projections which are among the most widely used projections in North America.

The first attempt to implement the transformation from geodetic coordinates to projection followed the procedure detailed in Chapter 6 of Krakiwsky (1973), however when testing against NRCan's TRX software, a small discrepancy was detected of ~2mm towards the edges a UTM zone. The current implementation matches TRX coordinates to the millimeter by following the procedure detailed in section 3.2 of Stem (1989):

$$N = S - S_0 + N_0 + \frac{R\,t\,L^2}{2} + \frac{R\,t\,L^4}{24}(5 - t^2 + 9\eta^2 + 4\eta^4) + \frac{R\,t\,L^6}{720}(61 - 58t^2 + t^4 + 270\eta^2 - 330t^2\eta^2) \quad (4.38)$$

$$E = E_0 + R\,L + \frac{R\,L^3}{6}(1 - t^2 + \eta^2) + \frac{R\,L^5}{120}(5 - 18t^2 + t^4 + 14\eta^2 - 58t^2\eta^2) + \frac{R\,L^7}{5040}(61 - 479t^2 + 179t^4 - t^6) \quad (4.39)$$

$$t = tan\varphi \quad (4.40)$$

$$\eta = \sqrt{e'^2 cos^2\varphi} \tag{4.41}$$

$$L = (\lambda - \lambda_0) cos\varphi \tag{4.42}$$

where S = meridian arc length, R = radius of the rectifying sphere, and e' = second eccentricity. All of which are provided by the ellipsoid object.

The inverse transformation is an iterative process which is also detailed in section 3.2 of Stem (1989), however for the sake of brevity these equasions are omitted here.

Meridian convergence, projection scale factor, and arc to chord correction are calculated using formulas from Krakiwsky (1973):

$$\gamma = \Delta\lambda sin\varphi(1 + \frac{\Delta\lambda^2 cos^2\varphi}{3}(1 + 3\eta^2 + 2\eta^4) + \frac{\Delta\lambda^4 cos^4\varphi}{15}(2 - t^2) \tag{4.43}$$

$$k = k_0(1 + \frac{\Delta\lambda^2 cos^2\varphi}{2}(1 + \eta^2) + \frac{\Delta\lambda^4 cos^4\varphi}{24}(5 - 4t^2)\ ) \tag{4.44}$$

$$(T - t) = \frac{(N'_2 - N'_1)(E'_2 + 2E'_1)}{6R_m^2}\left(1 - \frac{(E'_2 + 2E'_1)^2}{27R_m^2}\right) \tag{4.45}$$

where $N' = N - N_0$, $E' = E - E_0$, and R_m is the Gaussian radius of curvature at mid latitude.

An arc to chord correction dependent on geodetic coordinates would have been preferable as the abstract method in the Map Projection Class requires geodetic coordinates. Unfortunately, such an equation was not found. Consequently, this implementation of arc to chord accepts geodetic coordinates, then transforms them to projection coordinates, before calculating the arc to chord correction.

Lambert Conformal Conic Projection Class

The Lambert Conformal Conic Project Class is the second implementation of the Map Projection Class. It allows for transformations to both 1 and 2 standard parallel Lambert Conformal Conic Projections, though only the 2 standard parallel methods have been tested. This Map Projection Class was implemented to support the numerous state plane zones that utilize 2 standard parallel LCC projections.

Implementation of the transformation from geodetic to projection coordinates is a hybrid between the procedures found in Krakiwsky (1973) and Stem (1989). The initial attempt followed Chapter 7 of Krakiwsky (1973) however the equations (7.32 & 7.33) were found to erroneous and were replaced by the corresponding equations from Stem (1989). The equations as implemented:

$$N = N_0 + R_0 - R\cos\gamma \tag{4.46}$$

$$E = E_0 + R\sin\gamma \tag{4.47}$$

$$\gamma = \ell\,\lambda \tag{4.48}$$

$$R = K\,e^{-\ell q} \tag{4.49}$$

$$q = \ln\left(\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\left(\frac{1-e\sin\varphi}{1+e\sin\varphi}\right)^{e/2}\right) \tag{4.50}$$

$$K = \frac{N_1\cos\varphi_1}{\ell\,e^{-\ell\,q_1}} \tag{4.51}$$

$$\ell = \frac{\ln N_1 - \ln N_2 + \ln\cos\varphi_1 - \ln\cos\varphi_2}{q_2 - q_1} \tag{4.52}$$

where N_i is the radius of curvature on the prime vertical, K and ℓ are zone constants calculated from the 1st and 2nd standard parallels when the projection is instantiated.

The inverse transformation to longitude and isometric latitude (q) is straightforward (Krakiwsky, 1973):

$$\theta = \frac{E - E_0}{R_0 - (N - N_0)} \tag{4.53}$$

$$R = \left(\frac{R_0 - (N - N_0)}{\cos\theta}\right)\left(\frac{1}{\ell}\right) \tag{4.54}$$

$$\lambda = \left(\frac{E - E_0}{R_0 - (N - N_0)}\right) \tag{4.55}$$

$$q = \left(\frac{\ln K - \ln R}{\ell}\right) \tag{4.56}$$

The calculation of latitude from isometric latitude is an iterative function and can be found in Section 4.1 of Krakiwsky (1973).

Meridian convergence is calculated during the direct transformation as Eq (4.48). Scale factor is calculated (Krakiwsky, 1973):

$$k = \frac{K\,\ell\,e^{-\ell q}}{N_p} \tag{4.57}$$

Arc to Chord correction as a function of geodetic position from Vincenty (1986):

$$\varphi_3 = \varphi_1 + \frac{(\varphi_2 - \varphi_1)}{3} \tag{4.58}$$

$$(T - t) = (\ell - \sin\varphi_3)\frac{(\lambda_1 - \lambda_2)}{2} \tag{4.59}$$

Double Stereographic Projection Class

The Double Stereographic Projection Class is the third implementation of the Map Projection Class. It is implemented to support the official coordinate systems of New Brunswick and Prince Edward Island. Double stereographic projection is a two step process of projecting the ellipsoid to a sphere, producing spherical coordinates (χ,Λ), then projecting the sphere to a plane.

The procedure for projecting from the ellipsoid to the sphere, from Chapter 2 of Thomson, Mepham, & Steeves (1977):

$$\chi = 2(\arctan\{c_2[\tan\left(\frac{\pi}{4}+\frac{\varphi}{2}\right)\left(\frac{1-e\sin\varphi}{1+e\sin\varphi}\right)^{e/2}]^{c_1}\}-\frac{\pi}{4}) \tag{4.60}$$

$$\Lambda = c_1\,\lambda \tag{4.61}$$

$$\chi_0 = \frac{\varphi_0}{c_1} \tag{4.62}$$

$$\Lambda_0 = \lambda_0 c_1 \tag{4.63}$$

$$c_1 = \sqrt{1+\frac{e^2}{1-e^2}cos^4\varphi_0} \tag{4.64}$$

$$c_2 = \tan\left(\frac{\pi}{4}+\frac{\chi_0}{2}\right)[\tan\left(\frac{\pi}{4}+\frac{\varphi_0}{2}\right)\left(\frac{1-e\sin\varphi_0}{1+e\sin\varphi_0}\right)^{e/2}]^{-c_1} \tag{4.65}$$

where c_1 and c_2 are zone constants calculated when the projection is instantiated.

The procedure for projecting from the sphere to the plane, from Chapter 3 of Thomson, Mepham, & Steeves (1977):

$$N = N_0 + 2\ k_0\ R\ \frac{\sin\chi\cos\chi_0 - \cos\chi\sin\chi_0\cos\Delta\Lambda}{1+\sin\chi\sin\chi_0 + \cos\chi\cos\chi_0\cos\Delta\Lambda} \tag{4.66}$$

$$E = E_0 + 2\ k_0\ R\ \frac{\cos\chi\sin\Delta\Lambda}{1+\sin\chi\sin\chi_0 + \cos\chi\cos\chi_0\cos\Delta\Lambda} \tag{4.67}$$

$$\Delta\Lambda = \Lambda - \Lambda_0 \tag{4.68}$$

where R is the Gaussian radius of curvature at the origin of the projection.

The inverse transformation is also a two step process, transforming from the plane to the sphere, then from the sphere to the ellipsoid. The transformation from plane to sphere, from Chapter 3 of Thomson, Mepham, & Steeves (1977):

$$\chi = \arcsin(\sin\chi_0\cos\delta + \cos\chi_0\sin\delta\sin\beta) \tag{4.69}$$

$$\Delta\Lambda = \arcsin(\frac{\cos\beta\sin\delta}{\cos\chi}) \tag{4.70}$$

$$\Lambda = \Lambda_0 + \Delta\Lambda \tag{4.71}$$

$$s = \sqrt{N^2 + E^2} \tag{4.72}$$

$$\sin\beta = \frac{N}{s} \tag{4.73}$$

$$\cos\beta = \frac{E}{s} \tag{4.74}$$

$$\delta = 2\arctan(\frac{s}{2R}) \tag{4.75}$$

The transformation from the sphere to the ellipsoid is another iterative process that can be found in Chapter 2 of Thomson, Mepham, & Steeves (1977).

There is no meridian convergence introduced during the transformation from ellipsoid to sphere (Thomson, Mepham, & Steeves, 1977). The meridian convergence of the transformation from sphere to the plane, from Thomson, Mepham, & Steeves (1977):

$$\gamma = \arctan\left(\frac{\sin\Delta\Lambda\,(\sin\chi+\sin\chi_0)}{\cos\chi\cos\chi_0+(1+\sin\chi\sin\chi_0)\cos\Delta\Lambda}\right) \tag{4.76}$$

The scale factor introduced by the transformation from the ellipsoid to the sphere is negligible, but non-zero (Thomson, Mepham, & Steeves, 1977). As such, it is included in a combined scale factor equation, however it is dominated by the scale factor of the second transformation. From Chapter 4 of Thomson, Mepham, & Steeves (1977):

$$k = \frac{2\,k_0\,R\,c_1\cos\chi}{N\cos\varphi(1+\sin\chi\sin\chi_0+\cos\chi\cos\chi_0\cos\Delta\Lambda)} \tag{4.77}$$

The arc-to-chord correction of the transformation from the ellipsoid to the sphere is negligible, and is safely neglected (Thomson, Mepham, & Steeves, 1977). The arc to chord correction of the second transformation, as a function of spherical position, from Chapter 4 of Thomson, Mepham, & Steeves (1977) was rearranged to:

$$(T-t) = \frac{\pi}{2} - \arctan\left(\frac{(\cos\beta_j\cos\beta_i+\sin\beta_j\sin\beta_i)+\cot\left(\frac{\delta_j}{2}\right)\cot\left(\frac{\delta_i}{2}\right)}{\sin\beta_j\cos\beta_i-\cos\beta_j\sin\beta_i}\right) \tag{4.78}$$

$$\delta = \arccos(\sin\chi\sin\chi_0 + \cos\chi_0\cos\chi\cos\Delta\Lambda) \tag{4.79}$$

$$\sin\beta = \frac{\sin\chi\cos\chi_0-\sin\chi_0\cos\chi\cos\Delta\Lambda}{\sin\delta} \tag{4.80}$$

$$\cos\beta = \frac{\cos\chi\sin\Delta\Lambda}{\sin\delta} \tag{4.81}$$

Local Projection Class

The fourth and final implementation of the Map Projection Class is the Local Projection Class. This Projection Class allows the creation of a coordinate system with user defined coordinates. This implementation is based off the Double Stereographic Projection, with an addition rotation parameter used to define the transformation from the sphere to the plane. Parameters for a bilinear surface are also added to allow for user defined height coordinates.

The transformation from the ellipsoid to the sphere is identical to the same transformation in the Double Stereographic Projection Class above. The transformation from spherical coordinates to plane coordinates has been modified by adding a rotation parameter (θ) using the trigonometric sum-difference rule (Bremner & Santos, 2019):

$$N = N_0 + 2\,k_0\,R\,\frac{\sin\chi\cos\chi_0\cos\theta - \cos\chi\sin\chi_0\cos\Delta\Lambda\cos\theta - \cos\chi\sin\Delta\Lambda\sin\theta}{1+\sin\chi\sin\chi_0+\cos\chi\cos\chi_0\cos\Delta\Lambda} \tag{4.82}$$

$$E = E_0 + 2\,k_0\,R\,\frac{\cos\chi\sin\Delta\Lambda\cos\theta+\sin\chi\cos\chi_0\sin\theta-\cos\chi\sin\chi_0\cos\Delta\Lambda\sin\theta}{1+\sin\chi\sin\chi_0+\cos\chi\cos\chi_0\cos\Delta\Lambda} \tag{4.83}$$

The inverse transformation is the same as described above except that Eqs (4.73 & 4.74) are replaced by (Bremner & Santos, 2019):

$$\sin\beta = \frac{N}{s}\cos\theta - \frac{E}{s}\sin\theta \tag{4.84}$$

$$\cos\beta = \frac{E}{s}\cos\theta + \frac{N}{s}\sin\theta \tag{4.85}$$

The meridian convergence formula also has the rotation parameter added (Bremner & Santos, 2019):

$$\gamma = \arctan\left(\frac{\sin\Delta\Lambda\,(\sin\chi+\sin\chi_0)}{\cos\chi\cos\chi_0+(1+\sin\chi\sin\chi_0)\cos\Delta\Lambda}\right) - \theta \tag{4.86}$$

The scale factor and arc to chord corrections are identical to those in the double stereographic projection implementation (Bremner & Santos, 2019).

Three additional parameters are retained by this Class in order to define a plane surface that allows for user defined height coordinates. The first is a false geoid undulation value (N') that relates the orthometric height at the origin to the user defined elevation at the origin. The other 2 parameters are the slopes of the plane in the directions of latitude and longitude (ξ', η'). If a geoid model is not selected, then these three parameters are used in the place of geoid undulation and deflections of the vertical for all positions. If a geoid model is also used, then combined values (N_c, ξ_c, η_c) are calculated from false geoid values and the geoid values at the origin and the position of interest:

$$N_c = N' + N_p - N_o \tag{4.87}$$

$$\xi_c = \xi' + \xi_p \tag{4.88}$$

$$\eta_c = \eta' + \eta_p \tag{4.89}$$

4.3.3 Coordinate Geometry Package

The Coordinate Geometry (Cogo) package is an actor package that is responsible for the calculation of coordinates from measurements or constraints; and the reverse calculations. The classes in this package are highly interdependent (Figure 4.11). Each class in this package is dependent on the Coordinate System Class for the calculation of geodetic corrections. All classes in this package, other than the Two-Point Inverse Class, utilize the Two-Point Inverse Class, typically, to assist with iterative calculations.

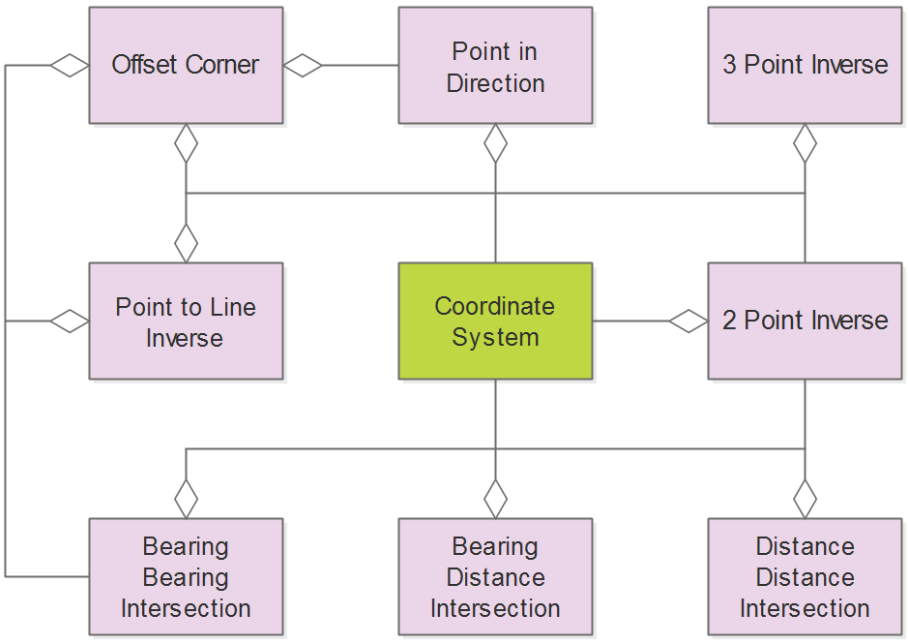

Figure 4.11: Class Diagram of the Coordinate Geometry Package

The Two-Point Inverse Class requires two sets of projection coordinates and a coordinate system object as input. Grid inverse values are calculated from the differences in projection coordinates provided:

$$\ell = \sqrt{\Delta N^2 + \Delta E^2} \tag{4.90}$$

$$t = \text{atan}(\frac{\Delta E}{\Delta \text{N}}) \tag{4.91}$$

$$\psi = \text{acos}(\frac{\Delta H}{r}) \tag{4.92}$$

Where ℓ is grid distance, t is grid azimuth, ψ is reduced zenith angle, and r is the ground slope distance. Geodetic, astronomic, and ground values are calculated by applying corrections provided by the coordinate system object to the grid values.

The Point in Direction Class, like all classes in this package requires a coordinate system object to provide geodetic corrections. It also requires the projection coordinates of the original point, plus distance, direction, and height arguments. Additional arguments are also required to define the argument types because the following types are supported:

- Distance:
    - Slope Distance (r)
    - Horizontal Distance (ϖ)
- Azimuth:
    - Astronomic Azimuth (α)
    - Grid Azimuth (t)
- Height:
    - Astronomic Zenith Angle (Z)
    - Reduced Zenith Angle (ψ)
    - Delta Elevation (ΔH)

Once a set of arguments is provided, approximate solution coordinates are calculated:

$$N_j = N_i + r_{ij} \cos \alpha_{ij}^E \tag{4.93}$$

$$E_j = E_i + r_{ij} \sin \alpha_{ij}^E \tag{4.94}$$

$$H_j = H_i + r_{ij} \cos Z_{ij} \tag{4.95}$$

Where r is ground slope distance, α^E is geodetic azimuth, and Z is astronomic zenith angle. The coordinates of the original point and the approximate solution point are then provided to a two-point inverse object to so that geodetic corrections can be calculated. These corrections are then applied to the input arguments and new approximate solution coordinates are calculated:

$$N_j = N_i + \ell_{ij} \cos t_{ij} \tag{4.96}$$

$$E_j = E_i + \ell_{ij} \sin t_{ij} \tag{4.97}$$

$$H_j = H_i + r_{ij} \cos \psi_{ij} \tag{4.98}$$

Where ℓ is grid distance, t is grid azimuth, and ψ is reduced zenith angle. This process continues iterating until the difference in successive solution coordinates falls below a predefined, sub-millimeter threshold. The solution coordinates and reduced measurements are then provided as results.

The intersection classes are more restricted in their argument types, only accepting horizontal distance, grid azimuth, and reduced zenith angle or delta elevation arguments.

The Bearing-Bearing Intersection Class requires two sets of projection coordinates, two grid azimuths, and a height argument as input. It calculates the slope and intercept of each of the two lines in projection space:

$$\beta = \tan(\frac{\pi}{2} - t) \tag{4.99}$$

$$\gamma = N - \beta\, E \tag{4.100}$$

Where β is slope, t is grid azimuth, and γ is the intercept. A set of horizontal projection coordinates that falls on the intersection of the two lines is calculated:

$$\begin{bmatrix} N \\ E \end{bmatrix} = \begin{bmatrix} 1 & -\beta_1 \\ 1 & -\beta_2 \end{bmatrix}^{-1} \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} \tag{4.101}$$

This calculation is not iterative, as only grid azimuths are accepted. If the height argument is a delta elevation, then the elevation of the solution coordinates is calculated by adding the delta elevation to the elevation of the first set of input coordinates and the process is complete. If the height argument is a reduced zenith angle then an iterative process begins:

$$\Delta H = r\cos(\psi) \tag{4.102}$$

$$r = r(\varpi, \Delta H) \tag{4.103}$$

Where ΔH is delta elevation, r is slope distance, ψ is reduced zenith angle, and ϖ is horizontal distance. Once the change in elevation between successive solutions falls below a predefined threshold the process stops, and the solutions coordinates are returned.

The Bearing-Distance Intersection Class also requires two sets of projection coordinates, one grid azimuth, one horizontal distance, and one height argument as input, as well as an argument specifying which coordinate set should be used for the calculation of solution elevation. Because the horizontal distance is a ground value, geodetic corrections must be applied to obtain a grid distance. The solution of horizontal coordinates is then an iterative process as the distance corrections are dependent on the coordinates of the solution. The slope and intercept of the azimuth are solved using equations (4.99 & 4.100). Two Easting values for the approximate solution coordinates are calculated by finding the roots of:

$$0 = (\beta^2 + 1)\, E_s^2 + 2(\gamma\beta - \beta N_D - E_D)E_s + (\gamma^2 + N_D^2 + E_D^2 - 2\gamma\beta - \ell^2) \quad (4.104)$$

Where E_s is the Easting of the solution, N_D and E_D are the coordinates the distance is from, and ℓ is the approximate Grid Distance. Northing for each of the solution eastings are calculated:

$$N = \beta E + \gamma \quad (4.105)$$

Solution coordinates are provided to a two-point inverse object which assists with the calculation of approximate grid distance for the next iteration. The process continues until the difference between successive sets of horizontal coordinates falls below the predetermined threshold. The elevations of the solutions are then calculated the same as they were for bearing-bearing intersection.

The Distance-Distance Intersection class requires two sets of projection coordinates, two horizontal distances, and one height argument. It is also an iterative process to calculate horizontal coordinates. Two sets of approximate solution coordinates are calculated (McRae, 2018):

$$N = \frac{N_2+N_1}{2} + \frac{(N_2-N_1)(\ell_1^2-\ell_2^2)}{2d^2} \pm \frac{(E_2-E_1)K}{2d^2} \quad (4.106)$$

$$E = \frac{E_2+E_1}{2} + \frac{(E_2-E_1)(\ell_1^2-\ell_2^2)}{2d^2} \pm \frac{-(N_2-N_1)K}{2d^2} \quad (4.107)$$

$$d = \sqrt{(N_2 - N_1)^2 + (E_2 - E_1)^2} \quad (4.108)$$

$$K = \sqrt{((\ell_1 + \ell_2)^2 - d^2)(d^2 - (\ell_1 - \ell_2)^2)} \quad (4.109)$$

Where the two solutions result from alternating the ±, and ℓ is approximate Grid Distance. Once again, new grid distance approximations are calculated using the approximate solution coordinates, and new solution coordinates are calculated until the difference between solutions falls below the threshold. Solution elevations are calculated just as they are for the other intersections.

The Three-Point Inverse Class uses two two-point inverse objects to solve the forward and backward inverses, then subtracts the backward azimuths from the forward azimuths to obtain angles. The Point to Line Inverse Class uses a bearing-bearing intersection object to calculate an intersection of the line and a perpendicular line passing through the point of interest, then uses a two-point inverse object to calculate the inverse values.

The Offset-Corner Class requires one set of projection coordinates, two azimuths, two offset distances, and a coordinate system object. It uses a point in direction object to calculate positions perpendicular to the specified azimuths and the specified offsets. It then calculates the solution position by performing a bearing-bearing intersection on the offset positions.

4.3.4 Observation and Equipment Packages

The observation and equipment packages are both data type packages. The primary purpose of the Equipment Package is the storage of variance information for error propagation by the observation package. Because of this dependency, the Observation Package (Yellow) and the Descriptors Package (Green) are displayed in the same class diagram (Figure 4.12).

All classes in the Equipment Package implement the abstract class Instrument. The Total Station, Level, and Receiver classes represent the instrument at the occupation position, while the Receiver and Target classes represent the instrument used at the recorded position. Each of these classes store meta data about the instrument being used. Some of this meta data is used in the processing of the measurements, such as offsets and expected variances. Other meta data is simply for record keeping such as instrument name and serial number.

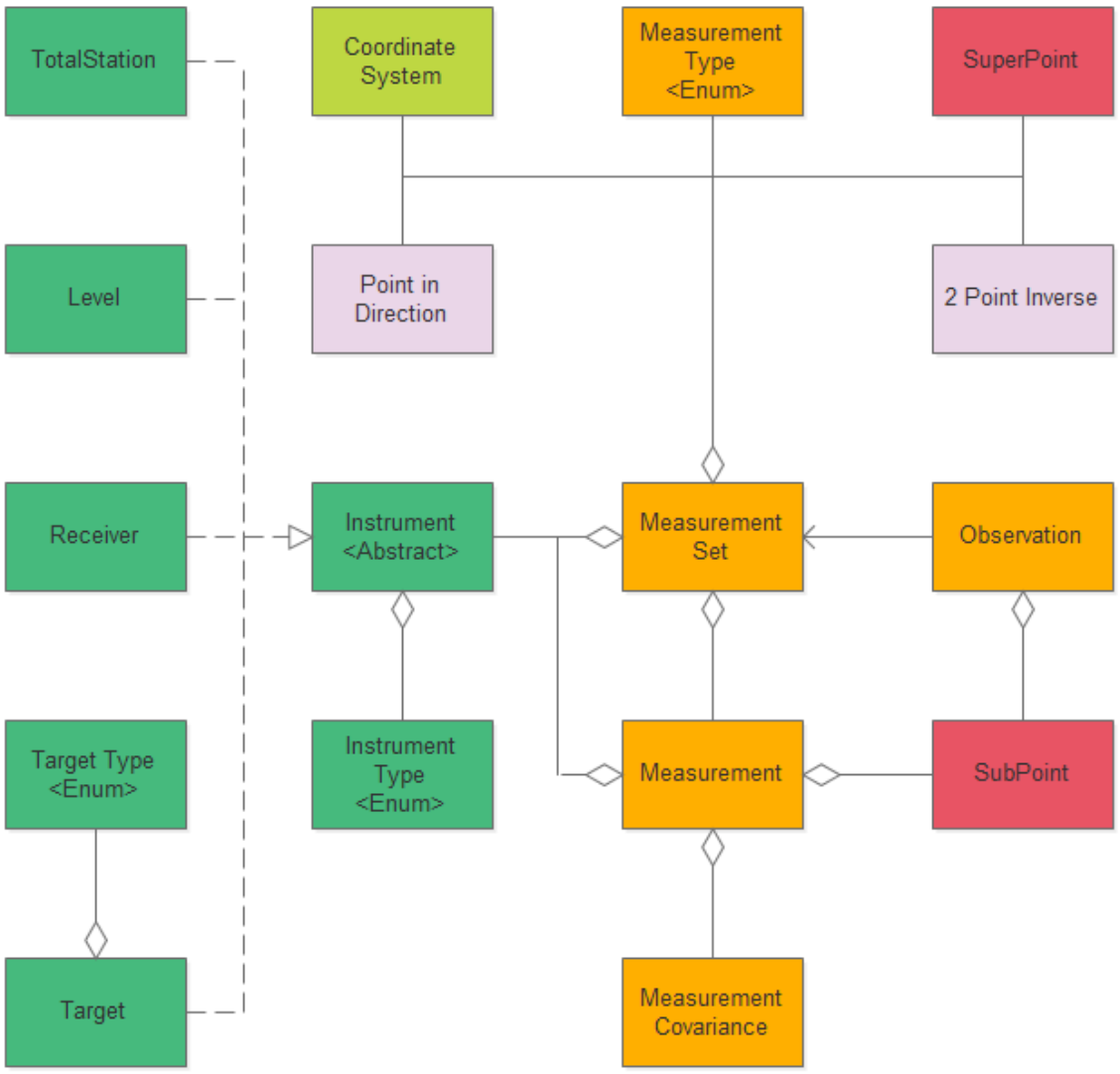

Figure 4.12: Class Diagram of Observation and Equipment Packages

The primary responsibilities of the Observation Package are the storage and processing of measurements. The Measurement Set Class holds a list of measurements that were recorded as a set and performs the processing of these measurements. The Measurements Set Class is dependent on several classes from other packages to assist with this processing of measurements.

The structure of measurements and measurement sets is dependent on the measurement type. The measurement types currently supported are:

- Total Station
- Level
- RTK
- Autonomous GNSS
- Static GNSS
- PPP

For total station measurements, a measurement set typically represents one occupation of the total station. Each record of the total station, containing a slope distance; azimuth; and zenith angle, is represented by a measurement object, which is attached to the measurement set object representing the occupation. No measurement covariance is attached to the measurement as the variance are functions of the total station and the measurements and can be calculated as required.

For level measurements, a measurement set also represents one occupation of the level, although the occupation is not associated with a specific position. A measurement represents a single record of the level, containing an elevation difference and an optional horizontal distance. Once again, no measurement covariance is a created.

A measurement set represents a single rover occupation for RTK and autonomous GNSS measurements, while each measurement represents one epoch of the occupation. Each measurement contains the (x,y,z) coordinates of the rover position, measured at the specific epoch, and each measurement has a measurement covariance object attached, containing the variance information associated with the measurement.

A measurement represents a single receiver occupation for static GNSS and PPP measurements. Each measurement also has a measurement covariance object attached to store the variances of the measurement. A measurement set object is created, but only so that the usage of objects within this package is consistent across measurement types.

For all measurement types, the measurement set can process the measurements to produce a list of points or a list of observations. Typically, the list of points would be generated immediately upon completion of the measurement set, while the observation list would be generated at a later date, when a least squares estimation is to be performed. The processing procedure generally requires calculating several intermediate values: averaged measurements, mark to mark measurements, and grid measurements.

The averaging performed on total station measurements is intended to be between the face 1 and face 2 measurements to targets; however, all measurements to a single target are averaged, regardless of quantity or face. Level, RTK, and Autonomous GNSS measurements, to a single target, within a single measurement set, are also averaged. No averaging is performed on PPP and Static measurements.

The calculation of Mark to Mark measurements, for total station measurements, corrects for atmospheric PPM, prism constant, height of instrument, and height of target. For level measurements, a correction is applied for the curvature of the earth. Both the differential GNSS reductions apply corrections for the height of instrument and the height of target, then difference the coordinates of both positions to obtain the cartesian interstation vector. The single receiver GNSS reductions only apply a correction for height of target.

The calculation of grid measurements from total station measurements relies on the Two Point Inverse and Point in Direction classes from the coordinate geometry package to perform the iterative calculations. No reduction to grid measurements is performed for level measurements. All GNSS Measurements transform the coordinates to projection coordinates. The single receiver GNSS solutions treat the projection coordinates of the receiver as the grid measurement, whereas the differential GNSS solutions difference the coordinates of the rover and base to obtain the projection

interstation vector. In rare cases, the reduction of GNSS measurements to grid measurements is not performed. In these cases, the cartesian coordinates of the receiver are used as the measurement, and the cartesian interstation vector is used for the differential GNSS solutions.

The first time the processing is performed is typically to calculate points from the measurements, subsequently a list of observations may be calculated for use in a least squares estimation. The intermediate values are not stored long-term, and must be calculated again from the raw measurements when the observations are required.

As stated above, each measurement typically contains multiple observables. Each observation object represents a single observable. Therefore, when the observation list is calculated it is typically longer than the measurement list. When an observation list is requested, an argument must be provided specifying the number of dimensions of the least squares estimation. Total station and GNSS measurements will return one observation per dimension of the estimation, while level measurements will only return one observation for the 1D and 3D cases. Error propagation is also performed during the calculation of observations, and each observation contains the variance associated with its value. The Observation Class also contains fields for the adjusted value and variance of the observation, however these values are calculated and added by the least squares estimation.

4.3.5 Event Package

The Event Package is classified as a data type package although the Event Processor Class is one of the primary actors of the library. The primary responsibility of this package is to provide an immutable log of inputs. However, existing events can also be reprocessed as required, by the Event Processor Class (Fig 4.13).

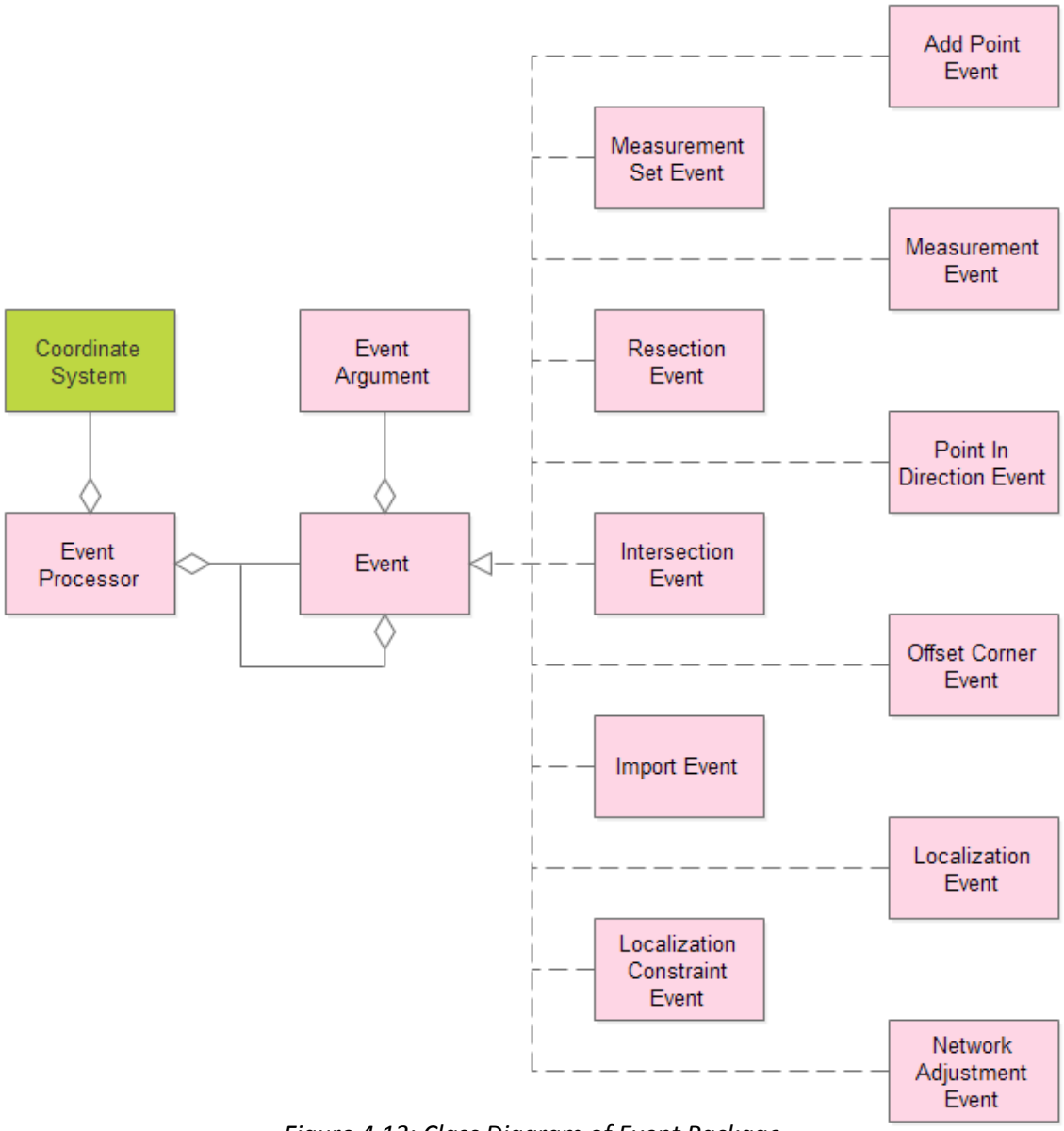

Figure 4.13: Class Diagram of Event Package

The purpose of this software system is the management of land surveying data. In many jurisdictions land surveying data can be considered legal evidence, if certain requirements are met. One of the requirements, in many cases, is the rigorous documentation of data, including erroneous data. The event package was designed to facilitate this logging, by creating records of inputs, including inputs from users, instruments and imported files. Events are designed to be viewed in event logs within applications, and in event report documents generated by the applications.

Event Classes have mandatory fields including: creation date, epoch, points created, and relationships to other events. Each Event Class also has a list of Event Arguments, which are key pairs of data, with the key describing the data and the value recording the data. Possible argument types include: point numbers, units, coordinates, measurement values, and descriptors. Each type of event has its own set of required arguments.

The intention is that once an event is created, it should not be modified, but this is outside of the control of the library and must be implemented in the applications utilizing the library. Events can be “replaced” by linking a new event of the same type to the event. In this case the Event Processor will ignore the “replaced” event and processing the new event in its place. The “replaced” event is not removed however, it can still be viewed in the event log, and will be included in the event report document.

The Event Processor Class is designed to parse a list of events and perform the calculations described by the events. This can either be the first processing of the data or a reprocessing. To ensure that duplication of data does not occur as a result of reprocessing, each event retains a link to all data objects generated as a result of the event. The Event Processor is not capable of generating new data objects, nor changing the linkage between them. An actor relying on the Event Processor to process data must create blank points, measurements, etc. in addition to the events and link the data objects to each other. The Event Processor does not perform calculations itself. It parses events and event arguments then performs the logic to select the appropriate actors and options. It then activates theses other components and collects the results of the calculations, which it applies to the appropriate data objects.

Special cases of event include Import, Localization, and Network Adjustment Events. Import events may refer to a different coordinate system object from the working coordinate system. Any events linked as sub events of the import event are then processed using the import coordinate system by the Event Processor. The resulting points are then transformed to the working coordinate system for storage. Localization and Network Adjustments are not processed by the Event Processor. If these events are encountered in a processing list, they are simply skipped.

4.3.6 Input Output Package

The function of the Input Output is the translation of data between the internal data structures used by this system and the data used by other systems. This function is performed primarily by a set of adapters. All non-abstract adapters are implementations of either the File Adapter or Equipment Adapter abstract classes (Fig 4.14) and are classified by this distinction.

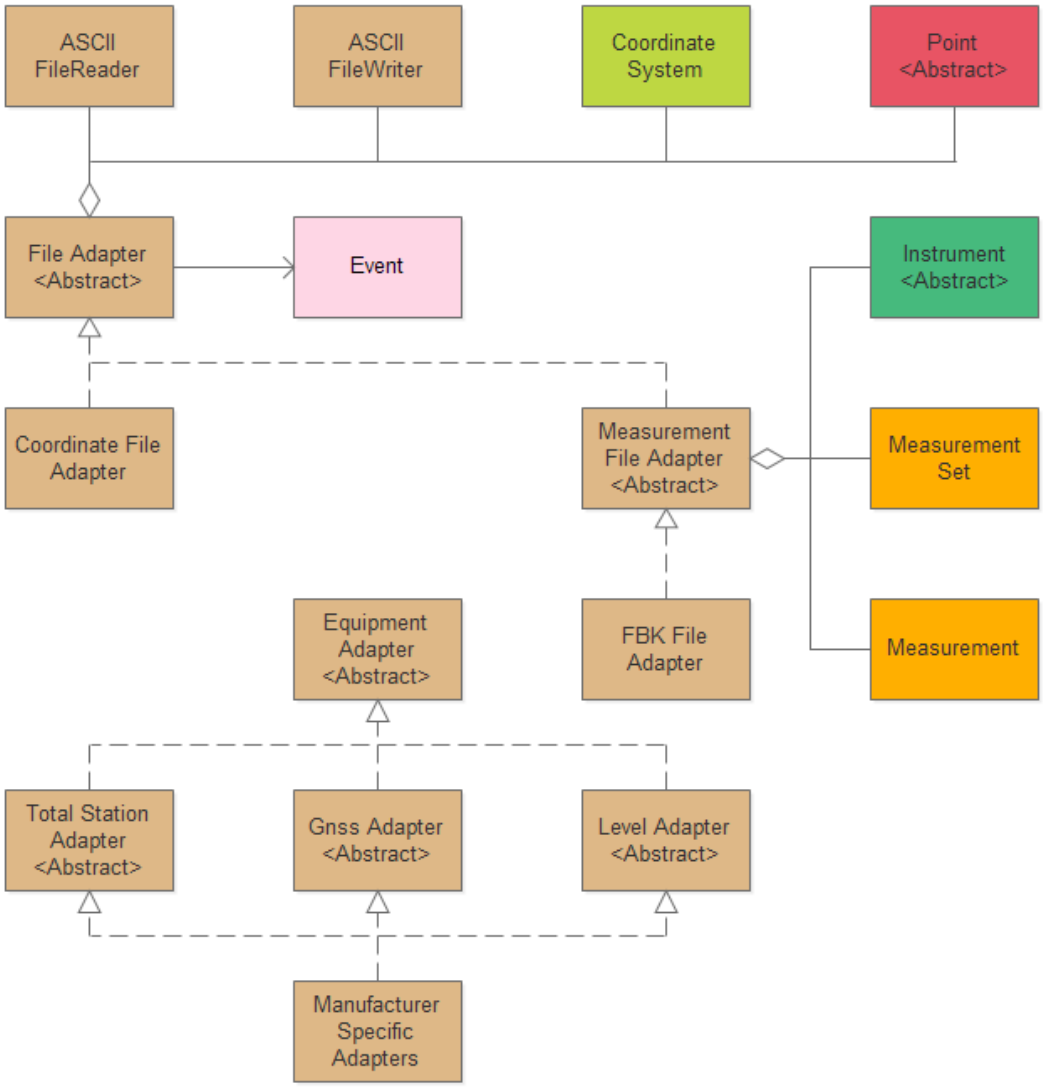

Figure 4.14: Class Diagram of the Input Output Package

File adapters exist to import and export data from specific file formats. File adapters are further broken down into coordinate file and measurement file adapters. At this time, the only implementation of the Measurement File Adapter is the FBK file Adapter, however this level of abstraction was added to allow the implementation of adapters for other measurement file formats, such as RW5 and LandXML, at a later date.

The Coordinate File Adapter is primarily concerned with the importing and exporting of points, though it does generate events when points are imported. Conversely measurement file adapters are primarily concerned with the importing and exporting of events, thought points are generated on import.

The Ascii File Reader and Ascii File Writer Classes read or write text files. The file reader converts the file to a list of string arrays which are then parsed by the appropriate adapter. The file writer accepts a list of string arrays from an adapter and prints it to an ascii file.

All file adapters strictly produce events and empty data objects on import. The processing of the imported data is delegated to the Event Processor class, which delegates processing of data, and populates the empty data objects produced by the file adapters.

The Coordinate File Adapter requires that list of fields, a delimiter, and units are provided, so that it knows how to parse import data, and how to format export data. Acceptable fields include: Point number, projection coordinates, geodetic coordinates, cartesian coordinates, standard deviations, code, and attributes. Any ascii character can theoretically be used as a delimiter, however comma, space, or tab are typically used. The coordinate file adapter also requires a coordinate system and an epoch, so that coordinates can be transformed.

As previously stated, the Coordinate File Adapter creates points and events on import, but delegates population of coordinates to the Event Processor. The entire import action is encapsulated within a single import event which retains the name of the file imported, the format used, and coordinate system selected for the data. The import event contains a list of sub events, each of the Add Point Class. Each of these retain the coordinate values as read from the file, along with the coordinate type, units, and epoch, so that the event processor can convert to the coordinate format of the job file.

On export, the Coordinate File Adapter transforms coordinates to the system, type, units and epoch specified. The point values corresponding with the specified fields are condensed to arrays of strings, which are provided to the Ascii File Writer Class. No events are generated when coordinates are exported, as no changes are made to the stored data.

Measurement File Adapters do not require specification of fields or delimiters as these are implicit in the selection of file format. Units must be specified for export but are determined by the file on import. The coordinate system still needs to be specified, but the epoch does not. On import the epoch is determined from the file, while on export the epoch is determined by the events.

The FBK file format is an older measurement file format and is primarily suitable for total station measurements. This makes extracting information from FBK files difficult as it lacks some command types that would help define data. For example, in the basic FBK format, GNSS measurements are represented by the add point command, which makes the command ambiguous. FBK files exported by Leica software add comment lines to fill in data missing from the file format, such as adding lines defining the GNSS reference station when GNSS measurements start. These comments cannot be relied upon as they are not part of the FBK format and may not be included in some files.

To balance the correct interpretation of data with the need to support the most basic FBK files several processing options are provided to the FBK File Adapter. For the GNSS measurement case, an option is provided to assume add point commands are GNSS measurements: always, never, or only after a reference station comment is found. Reference station can also be defined by the user or read from comment lines.

Many other comment lines are added by Leica, many of which are ignored, however some the comments like atmospheric settings, prism definition, and set up type can be interpreted or printed by the FBK File adapter depending on the processing options selected. Additional comment lines have also been added to FBK File Adapter to support leveling measurements within an FBK file.

On import, the FBK File Adapter Class also generates an import event, stating the file, format and coordinate system of the import. The import event contains a list of sub events which can include add point events and measurement set events for each of the supported measurement types. Again, the adapter creates empty point and measurement objects and links them with the event objects, so that the values can be added by the event processor.

On export, the FBK File Adapter Class reads a list of add point and measurement set events and formats a list of string arrays which are provided to an Ascii File writer for printing. Again, no events are generated on export as no internal data is changed.

The equipment adapters are not implemented at this stage, as their implementation is tightly coupled with the implementation of the equipment communication services within the Mobile Client Module. The equipment adapters will be implemented in conjunction with the services in the future.

4.3.7 Least Squares Estimation Package

The least squares package is the final actor package and capstone of the library. Each of the different adjustment types are implementations of the Parametric Least Squares Estimation Class (Fig 4.15). The generic estimation class utilizes the standard least parametric squares equations, while each implementation utilizes its own set of observation models.

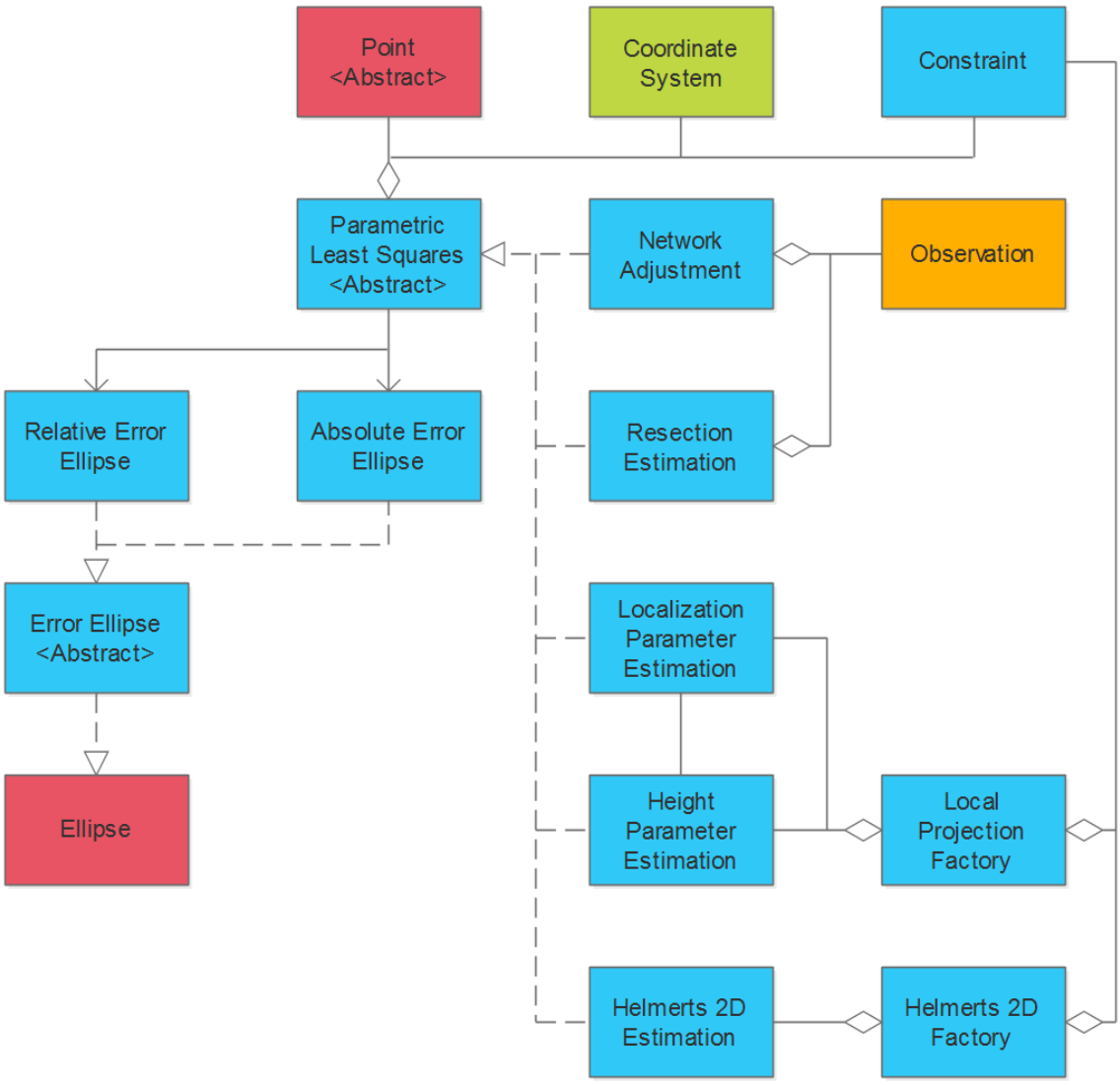

Figure 4.15: Class Diagram of the Least Squares Estimation Package

Parametric Least Squares Estimation Class

The term least squares refers to the criteria of minimum residuals (Ogundare, 2012). A least squares estimation is mathematical tool for the rigorous estimation of values from a set of observations, typically where the set contains redundant observations (Ogundare, 2012). A least squares estimation performed directly on the observations is called a conditional estimation and will produce a set of residuals that can be applied to the observations to obtain a set of adjusted observations with zero misclosure (Ogundare, 2012). A least squares estimation that utilizes to the observations to estimate derived quantities (parameters) is called a parametric estimation, although adjusted observations can also be obtained from this type of estimation (Ogundare, 2012). In this package, only the Parametric Least Squares Estimation is implemented.

The Parametric Least Squares Estimation Class is an Abstract Class that can be implemented to provide a specific type of parametric least squares estimation. It contains four abstract methods that must be implemented by classes implementing this class:

- buildInitialMatricies()
- buildMisclosureMatrix()
- buildDesignMatrix()
- buildConstraintMatrix()

Each implementation of the buildInitialMatrices() method is responsible for parsing the observation and constraint lists to build the observation, parameter, and weight matrices required for the specific estimation type. From Ogundare (2012) the general forms of these matrices are:

$$\ell = [o_1 \quad \dots \quad o_n]^T \tag{4.110}$$

$$x_0 = [p_1 \quad \dots \quad p_n]^T \tag{4.111}$$

$$P = \begin{bmatrix} \sigma_{o_1}^2 & \cdots & \sigma_{o_1 o_n} \\ \vdots & \ddots & \vdots \\ \sigma_{o_1 o_n} & \cdots & \sigma_{o_n}^2 \end{bmatrix}^{-1} \tag{4.112}$$

Where ℓ is the observation matrix, o_n is the individual observation (or constraint), x_o is the initial parameter matrix, p_n is the individual parameter, P is the weight matrix, $\sigma_{o_n}^2$ is the variance of an observation, and $\sigma_{o_1 o_n}$ is the covariance between two observations. Implementations of this method are also responsible for indexing the parameters so that their position in the parameter matrix can be retrieved.

Implementations of the buildMisclosureMatrix() method must parse the observation and constraint lists, and utilize the current parameter matrix, to construct a misclosure matrix using the observation models specific to the implementation. The general misclosure formula, from Ogundare (2012):

$$\omega_i = f_i(x_n) - \ell_i \tag{4.113}$$

$$\omega = [\omega_1 \quad \dots \quad \omega_i]^T \tag{4.114}$$

Where ω_i is the misclosure, f_i is the observation equation, x_n is the current iteration of parameters, and ω is the misclosure matrix. Misclosures must be placed in the misclosure matrix in the same position as the modeled observation in the observation matrix.

Implementations of the buildDesignMatrix() must parse the observation and constraint lists, and utilize the current parameter matrix, to construct a design matrix in the general form of (Ogundare, 2012):

$$A = \begin{bmatrix} \frac{\partial o_1}{\partial p_1} & \cdots & \frac{\partial o_1}{\partial p_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial o_n}{\partial p_1} & \cdots & \frac{\partial o_n}{\partial p_n} \end{bmatrix} \tag{4.115}$$

Where A is the design matrix, and $\frac{\partial o_n}{\partial p_n}$ is the partial derivatives of the observations by the parameters.

The buildConstraintMatrix() method constructs the constraint matrix G. It is only truly implemented in the Network Adjustment Class, while all other implementations throw an exception stating that constraint matrices are not supported for this type of estimation. As such there is no general form, only the one specific implementation which will be discussed with the network adjustment implementation.

The solve() method is implemented within this abstract class, and each implementing class utilizes this method to perform its estimation. This method first determines if the constraint Matrix (*G*) should be included in the adjustment. The solve method then calls the implementation specific methods to construct the required matrices. An iterative process begins where the corrections to the parameters δ are calculated and applied until the values of δ fall below a set threshold. The process from Ogundare (2012):

$$N = A^T P A \tag{4.116}$$

$$u = A^T P w \tag{4.117}$$

$$\delta = -N^{-1} u \tag{4.118}$$

$$x_n = x_{n-1} + \delta_n \tag{4.119}$$

Where *N* and u are coefficient matrices, x_n is the current estimation of parameters, and δ is the correction to the parameters. For each iteration, new misclosure and design matrices are created by the implementation specific methods, using the current estimation of parameters.

If a constraint matrix is to be included, then the *N* coefficient matrix is calculated as:

$$N^{-1} = (A^T P A + G G^T)^{-1} A^T P A (A^T P A + G G^T)^{-1} \tag{4.120}$$

Once the iterative process is complete several final values are calculated including: degrees of freedom, residuals, adjusted observations, a-posteriori variance factor, and adjusted covariance matrices, using formulas from Ogundare (2012):

$$df = n_o - n_p \tag{4.121}$$

$$v = A\delta + \omega \tag{4.122}$$

$$\hat{\ell} = \ell + v \tag{4.123}$$

$$s_0^2 = \frac{v^T P v}{df} \tag{4.124}$$

$$C_{\hat{x}} = s_0^2 \, N^{-1} \tag{4.125}$$

$$C_{\hat{\ell}} = s_0^2 \, (A \, N^{-1} \, A^T) \tag{4.126}$$

Where *df* is the degrees of freedom, n_o and n_p are the number of observation equations and parameters respectively, *v* is the residuals, $\hat{\ell}$ is the adjusted observations, s_0^2 is the a-posteriori variance factor, $C_{\hat{x}}$ is the covariance matrix of adjusted parameters, and $C_{\hat{\ell}}$ is the covariance matrix of adjusted observations.

Three test methods have been implemented to assess the quality of the estimation and observations: Goodness of Fit Test, Global Test, and Local Test. The Goodness of Fit Test and Local Tests both rely on standardized residuals calculated (Ogundare, 2012):

$$\widehat{v_i} = \frac{v_i}{\sigma_{v_i}} \tag{4.127}$$

$$\sigma_{v_i}^2 = \frac{s_0^2}{\sigma_0^2} \, \sigma_{\ell_i}^2 - \sigma_{\hat{\ell}_i}^2 \tag{4.128}$$

Where $\widehat{v_i}$ is the standardized residual, $\sigma_{v_i}$is the standard deviation of the residual, $\sigma_{\ell_i}^2$ is the a-priori standard deviation of the observation, and $\sigma_{\hat{\ell}_i}^2$ is the standard deviation of the adjusted observation.

The Goodness of Fit Test follows the procedure of the Chi Squared Goodness of Fit Test of the Residuals of Vanicek & Krakiwsky (1986). A histogram with $\sqrt{n_o}$ bins is formed from the standardized residuals, and the actual number in each bin is compared to the expected number in a normal distribution:

$$0 < \sum_{i=1}^{n} \frac{(a_i - e_i)^2}{e_i} < \xi_{\chi^2_{n-2},1-\alpha} \tag{4.129}$$

Where *n* is the number of bins, a_i and e_I are the actual and expected values for each bin, and $\xi_{\chi^2_{n-2},1-\alpha}$ is the Chi Squared Inverse Function for a given significance level α.

The Global Test from Ogundare (2012) is equivalent to the Test of the Quadratic Form of the Residuals from Vanicek & Krakiwsky (1986). This test compares the a-priori variance factor (σ_0^2) to the a-posteriori variance factor (s_0^2) to determine if the calculated uncertainties of the observations were accurate:

$$\frac{df\ s_0^2}{\xi_{\chi^2_{df},1-\frac{\alpha}{2}}} < \sigma_0^2 < \frac{df\ s_0^2}{\xi_{\chi^2_{df},\frac{\alpha}{2}}} \tag{4.130}$$

It is a two-tailed test where a failure on the high side likely indicates that uncertainties were inaccurately estimated, and a failure on the low side likely indicates the presence of blunders in the observations (Ogundare, 2012).

The Local Tests are In-context Local Tests from Ogundare (2012). These tests are nearly equivalent to the Test of a Residual Outlier from Vanicek and Krakiwsky (1986), except that rather than performing a symmetrical two-tailed test, a one-tailed test is performed on the absolute value of the standardized residual:

$$|\widehat{v_i}| < \xi_{\tau,1-\alpha_0} \tag{4.131}$$

$$\alpha_0 = 1 - (1 - \alpha)^{1/n_o} \tag{4.132}$$

Where $\xi_{\tau,1-\alpha_0}$is the Tau Inverse Function, and α_0 is the significance factor, scaled to perform in-context testing (Ogundare, 2012). A failure of the local test indicates that the observation is an outlier and often indicates a blunder in the observation (Ogundare, 2012).

Constraint Class

As stated previously, parametric least squares estimations estimate a set of derived quantities (parameters) from a set of observations. Relative observations like angles, distances, and vectors provide internal network geometry, but do not define the network externally (Ogundare, 2012). For survey networks, constraints on parameters

are required to provide external network geometry, otherwise the N coefficient matrix becomes singular when the estimation is run, and the estimation fails.

The Constraint Class provides a means to add constraints to an estimation. The Parametric Least Squares Estimation Class accepts a list of constraint objects as a parameter and most implementations require this constraint list. Differing estimation implementations require and accept different types of constraints.

Constraint types implemented include:

- Coordinate
- Scale
- Rotation
- Direction
- Distance
- FreeNetwork
- PointPair

Coordinate constraints can be for any or all of the three dimensions and can be fixed or weighted. Fixed coordinates are removed from the parameter and design matrices so not to be modified from the estimation. Weighted coordinates are treated as additional observation equations in the observation and design matrices and are subject to modification from the estimation. Direction, distance, scale and rotation

constraints can also be fixed or weighted, although direction and distance constraints are treated as additional observation equations regardless.

Free Network Constraints are only accepted by the Network Adjustment Class and triggers the creation of a constraint matrix that fixes the centroid of the included points and creates no net rotation and no net scale change conditions. Point pair constraints are not accepted by any of the estimation implementations, but are instead required by the factory classes which generate sets of coordinate constraints from them.

Helmerts 2D Estimation Class

The Helmerts 2D Estimation Class estimates a set of transformation parameters for a 2D Helmerts Transformation (k, θ, ΔN, ΔE). It does not accept standard observations as input and works entirely on a set of constraints. Coordinate constraints are required, and rotation and scale constraints are optional.

The buildInitalMatrices() method parses the constraint list and adds the target coordinates to the observation matrix (Eq 4.110). It also averages the Northings and Eastings of the current coordinates to find the origin for the rotation and scale. If fixed rotation or scale constraints are found, it adds these values to a fix parameter list that is not included in the parameter or design matrices, otherwise rotation (θ) and scale (k)

are added to the parameter list along with ΔN and ΔE (Eq 4.111). Finally, it inverses the variance value in each constraint and adds them to a diagonal weight matrix (Eq 4.112). The buildMisclosureMatrix() method parses the constraint list and calculates misclosures using Eq (4.113), placing each value in the misclosure matrix (Eq 4.114). The observation equations(*f*(*x*)) for this estimation are:

$$N_j = k(N_i \cos\theta - E_i \sin\theta) + \Delta N \tag{4.133}$$

$$E_j = k(N_i \sin\theta + E_i \cos\theta) + \Delta E \tag{4.134}$$

$$k = k \tag{4.135}$$

$$\theta = \theta \tag{4.136}$$

The buildDesignMatrix() method calculates the partial derivatives of each of the observation equations and places them in the design matrix (Eq 4.115). The partial derivatives of the observations are:

$$\frac{\partial N}{\partial \Delta N} = 1 \tag{4.137}$$

$$\frac{\partial N}{\partial k} = N_i \cos\theta - E_i \sin\theta \tag{4.138}$$

$$\frac{\partial N}{\partial \theta} = k(-N_i \sin\theta - E_i \cos\theta) \tag{4.139}$$

$$\frac{\partial E}{\partial \Delta E} = 1 \tag{4.140}$$

$$\frac{\partial E}{\partial k} = N_i \sin\theta + E_i \cos\theta \tag{4.141}$$

$$\frac{\partial E}{\partial \theta} = k(N_i \cos\theta - E_i \sin\theta) \tag{4.142}$$

$$\frac{\partial k}{\partial k} = 1 \tag{4.143}$$

$$\frac{\partial \theta}{\partial \theta} = 1 \tag{4.144}$$

With all other partial derivatives resulting in 0.

The estimation test methods provide no meaningful results for this estimation. The Goodness of Fit Test cannot be run because observations are not utilized. The Local Tests cannot be run for the same reason. The Global Test can be performed but provides no meaningful results as the variances of the coordinate constraints are typically set to 1 and the variance of scale and rotation constraints are selected to specify weights in the estimation.

Localization Parameter Estimation Class

The Local Parameter Estimation Class estimates the four horizontal parameters necessary to construct a local projection object (k_0, θ, N_0, E_0). Like the Helmerts 2D Estimation Class, it works entirely off of a set of constraints, requiring a set of coordinate constraints and accepting optional scale and rotation constraints.

Again, the buildInitalMatrices() method parses the constraint list and adds the target coordinates to the observation matrix (Eq 4.110). However, this implementation requires the current coordinates in geodetic form. It averages the latitudes and

longitudes of the current coordinates to find the geodetic origin of the projection. Rotation and scale constraints are treated the same as they are in the Helmerts estimation, with fixed parameters being sorted to a separate parameter list, or weighted parameters being added to the parameter matrix (Eq 4.111). The parameters, regardless of which list they are sorted into, are rotation (θ), origin scale factor (k_0), Northing of origin (N_0), and Easting of origin (E_0). The weight matrix (Eq 4.112) is again constructed by inversing the variance of the constraints and adding them to the diagonal elements of the matrix.

The observation equations (*f*(*x*)) used by the buildMisclosureMatrix() method to build the misclosure matrix (Eq 4.114) are:

$$N = N_0 + 2\ k_0\ R\ \frac{\sin\chi\cos\chi_0\cos\theta - \cos\chi\sin\chi_0\cos\Delta\Lambda\cos\theta - \cos\chi\sin\Delta\Lambda\sin\theta}{1 + \sin\chi\sin\chi_0 + \cos\chi\cos\chi_0\cos\Delta\Lambda} \quad (4.145)$$

$$E = E_0 + 2\ k_0\ R\ \frac{\cos\chi\sin\Delta\Lambda\cos\theta + \sin\chi\cos\chi_0\sin\theta - \cos\chi\sin\chi_0\cos\Delta\Lambda\sin\theta}{1 + \sin\chi\sin\chi_0 + \cos\chi\cos\chi_0\cos\Delta\Lambda} \quad (4.146)$$

$$k_0 = k_0 \quad (4.147)$$

$$\theta = \theta \quad (4.148)$$

Where χ and Λ are spherical latitudes and longitudes, defined in the discussion of the Double Stereographic Projection in section 4.3.3.

The partial derivatives of the observation equations, used to build the design matrix (Eq. 4.115), are:

$$\frac{\partial N}{\partial N_0} = 1 \tag{4.149}$$

$$\frac{\partial N}{\partial k_0} = 2\,R\,\frac{\sin\chi\cos\chi_0\cos\theta - \cos\chi\sin\chi_0\cos\Delta\Lambda\cos\theta - \cos\chi\sin\Delta\Lambda\sin\theta}{1+\sin\chi\sin\chi_0+\cos\chi\cos\chi_0\cos\Delta\Lambda} \tag{4.150}$$

$$\frac{\partial N}{\partial \theta} = 2\,k_0\,R\,\frac{-\sin\chi\cos\chi_0\sin\theta+\cos\chi\sin\chi_0\cos\Delta\Lambda\sin\theta-\cos\chi\sin\Delta\Lambda\cos\theta}{1+\sin\chi\sin\chi_0+\cos\chi\cos\chi_0\cos\Delta\Lambda} \tag{4.151}$$

$$\frac{\partial E}{\partial E_0} = 1 \tag{4.152}$$

$$\frac{\partial E}{\partial k_0} = 2\,R\,\frac{\cos\chi\sin\Delta\Lambda\cos\theta+\sin\chi\cos\chi_0\sin\theta-\cos\chi\sin\chi_0\cos\Delta\Lambda\sin\theta}{1+\sin\chi\sin\chi_0+\cos\chi\cos\chi_0\cos\Delta\Lambda} \tag{4.153}$$

$$\frac{\partial E}{\partial \theta} = 2\,k_0\,R\,\frac{-\cos\chi\sin\Delta\Lambda\sin\theta+\sin\chi\cos\chi_0\cos\theta-\cos\chi\sin\chi_0\cos\Delta\Lambda\cos\theta}{1+\sin\chi\sin\chi_0+\cos\chi\cos\chi_0\cos\Delta\Lambda} \tag{4.154}$$

$$\frac{\partial k_0}{\partial k_0} = 1 \tag{4.155}$$

$$\frac{\partial \theta}{\partial \theta} = 1 \tag{4.156}$$

Where all other derivatives result in 0. As with the Helmerts estimation, the estimation test methods provide no meaningful results for this estimation.

Localization Height Parameter Estimation Class

The Localization Height Parameter Estimation Class estimates the three vertical parameters that can be used in the construction of a local projection object (N', ξ', η'). It requires, and only accepts elevation coordinate constraints.

The buildIntialMatrices() method parses the list of elevation coordinate constraints and adds the target elevations to the observation matrix (Eq 4.110). The three parameters pseudo undulation (N'), pseudo deflection of vertical northing (ξ'), and pseudo

deflection of vertical easting (η') are added to the parameter matrix (Eq 4.111) by this method. The diagonal elements of the weight matrix (Eq 4.112) are also populated with the inverse of the variances of the constraints.

The only observation equation ($f(x)$) used to build the misclosure matrix (Eq 4.114) is:

$$H = \mathrm{h} - (\xi' \, \Delta\varphi \, r + \, \eta' \, \Delta\lambda \, r + N' + \, N_p - N_o) \tag{4.157}$$

Where r is the radius, p is the point of interest, o is the origin, and:

$$\Delta\varphi = \, \varphi_p - \varphi_o \tag{4.158}$$

$$\Delta\lambda = \, \lambda_p - \lambda_o \tag{4.159}$$

The design matrix (Eq 4.115) is build using the partial differentials:

$$\frac{\partial H}{\partial N'} = -1 \tag{4.160}$$

$$\frac{\partial H}{\partial \xi'} = -\Delta\varphi \, r \tag{4.161}$$

$$\frac{\partial N}{\partial \eta'} = -\Delta\lambda \, r \tag{4.162}$$

Once again, the Goodness of Fit and Local Tests cannot be run on this type of estimation due to a lack of observations, and again, the Global Test provides no meaningful results as the variances of constraints are not propagated from measurements.

Resection Estimation Class

The Resection Estimation Class estimates a set of coordinates (N, E, Z) for a total station occupation point. It acts on a list of observations, derived from total station measurements, to a set of known points. 2D observation (azimuth (α) and grid distance (t)) are required, while 3D observations (delta elevation (ΔZ)) are optional. It is the only estimation implementation that does not accept constraints.

The buildInitialMatrices() method parses the observation list and adds the values to the observation matrix (Eq 4.110). For most observations the inverse of the variance is added to the weight matrix (Eq 4.112), unless the observation is marked as suspect, in which case the weight is divided by 9. The coordinates of the occupation point are added to the parameter matrix (Eq 4.111). Northing (N_o) and easting (E_o) coordinates are added in all cases, but elevation (Z_0) is only added if the estimation is set to 3D. A rotation value (θ) is also added to the parameter list so that the orientation of the occupation station can be estimated. The buildMisclosureMatrix() method uses Eq (4.113) to populate the misclosure matrix (Eq 4.114). The observation equations ($f(x)$) used are:

$$t = \arctan\left(\frac{E_t - E_o}{N_t - N_o}\right) + \theta \tag{4.163}$$

$$\ell = \sqrt{(N_t - N_o)^2 + (E_t - E_o)^2} \tag{4.164}$$

$$\Delta Z = Z_t - Z_0 \tag{4.165}$$

The buildDesignMatrix() method populates the design matrix (Eq 4.115) with the partial derivatives of the observation equations:

$$\frac{\partial t}{\partial N_o} = \frac{E_t - E_o}{(N_t - N_o)^2 \left(\frac{(E_t - E_o)^2}{(N_t - N_o)^2} + 1\right)} \tag{4.166}$$

$$\frac{\partial t}{\partial E_o} = \frac{-1}{(N_t - N_o) \left(\frac{(E_t - E_o)^2}{(N_t - N_o)^2} + 1\right)} \tag{4.167}$$

$$\frac{\partial t}{\partial \theta} = 1 \tag{4.168}$$

$$\frac{\partial \ell}{\partial N_o} = \frac{N_o - N_t}{\sqrt{(N_t - E_0)^2 + (E_t - E_0)^2}} \tag{4.169}$$

$$\frac{\partial \ell}{\partial E_o} = \frac{E_o - E_t}{\sqrt{(N_t - E_0)^2 + (E_t - E_0)^2}} \tag{4.170}$$

$$\frac{\partial \Delta Z}{\partial Z_o} = -1 \tag{4.171}$$

Any of the estimation health tests can be performed on the resection estimation. However, the results should be suspect in all cases, as a resection estimation typically contains too few observations for the results of statistical analysis to be trustworthy.

Network Adjustment Class

The Network Adjustment Class is the largest and most complex implementation of the Parametric Least Squares Estimation Class. It supports every type of observation discussed in section 4.3.4 and many of the constraints discussed earlier in this section. It is the primary focus of the estimation tests and is the only estimation that produces

error ellipse objects. A network adjustment simultaneously estimates the coordinates (N, E, Z) of all points included in the estimation. This estimation can be 1D, 2D or 3D and can integrate measurements from Total station, Level, and GNSS instruments.

Several options for integrating discrete networks have been implemented. Options include: simultaneous estimation of all observations from both networks, estimation of one network using weighted or fixed points from the existing network, Free-Network adjustment of a network and transformation to match and existing network, and reduction of existing network to a series of pseudo-observation vectors to be included in the estimation of the subsequent network.

The buildInitialMatrices() method first parses the list of observations then parses the list of constraints, adding the values in each to the observation matrix (Eq. 4.110). The diagonal elements of the weight matrix (Eq 4.112) are populated with the inverse of the variance for observations and weighted constraints, except observations marked as suspect, where weights are reduced by an arbitrary factor of 9. For each observation and constraint, each associated point is compared with the existing parameter lists, if a point is not found in either the standard parameter list or the fixed parameter list then it is added to the parameter list and its coordinates are added to the parameter matrix (Eq 4.111). When a fixed coordinate constraint is encountered a search is also performed for point in question in the parameter lists. If the point is found in the

standard parameter list, it is removed and added to the fixed parameter list, while its coordinates are removed from the parameter matrix. Rotation parameters (θ) are also added to the parameter matrix for each total station occupation encountered.

The misclosure matrix (Eq 4.114) is built using both the observation and constraint lists. Misclosures are calculated using Eq (4.113) and the observation equations (*f*(*x*)):

$$t = \arctan\left(\frac{E_t - E_o}{N_t - N_o}\right) + \theta \tag{4.172}$$

$$\ell = \sqrt{(N_t - N_o)^2 + (E_t - E_o)^2} \tag{4.173}$$

$$\Delta N = N_t - N_0 \tag{4.174}$$

$$\Delta E = E_t - E_0 \tag{4.175}$$

$$\Delta Z = Z_t - Z_0 \tag{4.176}$$

$$\mathrm{N} = \mathrm{N}_t \tag{4.177}$$

$$\mathrm{E} = E_t \tag{4.178}$$

$$\mathrm{Z} = Z_t \tag{4.179}$$

The design matrix (Eq 4.115) is also built from both the observation and constrain list. It is populated with the following partial derivatives:

$$\frac{\partial t}{\partial N_o} = \frac{E_t - E_o}{(N_t - N_o)^2\left(\frac{(E_t - E_o)^2}{(N_t - N_o)^2} + 1\right)} \tag{4.180}$$

$$\frac{\partial t}{\partial E_o} = \frac{-1}{(N_t - N_o)\left(\frac{(E_t - E_o)^2}{(N_t - N_o)^2} + 1\right)} \tag{4.181}$$

$$\frac{\partial t}{\partial N_t} = \frac{E_o - E_t}{(N_t - N_o)^2 \left(\frac{(E_t - E_o)^2}{(N_t - N_o)^2} + 1\right)} \tag{4.182}$$

$$\frac{\partial t}{\partial E_t} = \frac{1}{(N_t - N_o) \left(\frac{(E_t - E_o)^2}{(N_t - N_o)^2} + 1\right)} \tag{4.183}$$

$$\frac{\partial t}{\partial \theta} = 1 \tag{4.184}$$

$$\frac{\partial \ell}{\partial N_o} = \frac{N_o - N_t}{\sqrt{(N_t - E_0)^2 + (E_t - E_0)^2}} \tag{4.185}$$

$$\frac{\partial \ell}{\partial E_o} = \frac{E_o - E_t}{\sqrt{(N_t - E_0)^2 + (E_t - E_0)^2}} \tag{4.186}$$

$$\frac{\partial \ell}{\partial N_t} = \frac{N_t - N_o}{\sqrt{(N_t - E_0)^2 + (E_t - E_0)^2}} \tag{4.187}$$

$$\frac{\partial \ell}{\partial E_t} = \frac{E_t - E_o}{\sqrt{(N_t - E_0)^2 + (E_t - E_0)^2}} \tag{4.188}$$

$$\frac{\partial \Delta Z}{\partial Z_o} = -1 \tag{4.189}$$

$$\frac{\partial \Delta Z}{\partial Z_o} = -1 \tag{4.190}$$

$$\frac{\partial \Delta Z}{\partial Z_o} = -1 \tag{4.191}$$

$$\frac{\partial \Delta N}{\partial N_t} = 1 \tag{4.192}$$

$$\frac{\partial \Delta E}{\partial E_t} = 1 \tag{4.193}$$

$$\frac{\partial \Delta Z}{\partial Z_t} = 1 \tag{4.194}$$

$$\frac{\partial N}{\partial N_t} = 1 \tag{4.195}$$

$$\frac{\partial E}{\partial E_t} = 1 \tag{4.196}$$

$$\frac{\partial Z}{\partial Z_t} = 1 \tag{4.197}$$

This is the only estimation implementation that truly implements the buildConstraintMatrix() method. It is only used for performing a Free-Network adjustment, constructing the constraint matrix (G) with the following constraints:

$$\sum_{i=1}^{n} \delta N_i = 0 \tag{4.198}$$

$$\sum_{i=1}^{n} \delta E_i = 0 \tag{4.199}$$

$$\sum_{i=1}^{n} \delta Z_i = 0 \tag{4.200}$$

$$\sum_{i=1}^{n} (N_i\, \delta E_i - E_i\, \delta N_i) = 0 \tag{4.201}$$

$$\sum_{i=1}^{n} (N_i\, \delta N_i + E_i\, \delta E_i) = 0 \tag{4.202}$$

Where Eqs (4.198, 4.199 & 4.200) constrain the coordinates of the centroid of the network, Eq (4.201) adds a no net rotation constraint, and Eq (4.202) adds a no scale change constraint. For a 3D estimation the constraint matrix takes the form:

$$G = \begin{bmatrix} 1 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 1 \\ -E_1 & N_1 & 0 & \cdots & -E_n & N_n & 0 \\ N_1 & E_1 & 0 & \cdots & N_n & E_n & 0 \end{bmatrix} \tag{4.203}$$

After an estimation, The Goodness of Fit Test and Global Test can be used to asses the health of the estimation and Local Tests can be used to identify possible outliers. When outlier observations are detected, they can either be deactivated or flagged as suspect and the estimation can be run again. Inactive observations are not used in subsequent estimations, while suspect observations have their weight reduced by a factor of 9.

To visualize, and help with analysis of, the results of the network adjustment, absolute and relative error ellipse objects can be generated. Error ellipses contain uncertainty data for two dimensions. Each error ellipse contains a maximum uncertainty value (semi- major axis (a)), a minimum uncertainty value (semi-minor axis (b)), and a direction of the maximum value (β). The direction of the minimum value is also defined as it is always orthogonal to the maximum value (Ogundare, 2012).

An absolute error ellipse contains the uncertainty data for a single position, while a relative error ellipse contains uncertainty data about the spatial relationship between two positions. Error ellipses are calculated from a 2 x 2 covariance matrix describing the variances of the two parameters and the covariance between them:

$$C = \begin{bmatrix} \sigma_N^2 & \sigma_{NE} \\ \sigma_{NE} & \sigma_E^2 \end{bmatrix} \tag{4.204}$$

The ellipse parameters are then calculated:

$$a = \sqrt{\lambda_1} \tag{4.205}$$

$$b = \sqrt{\lambda_2} \tag{4.206}$$

$$\beta = \arctan(\sigma_{NE} \,/\, (\lambda_1 - \sigma_e^2)) \tag{4.207}$$

$$\lambda_1 = (\sigma_N^2 + \sigma_E^2 + z)/2 \tag{4.208}$$

$$\lambda_2 = (\sigma_N^2 + \sigma_E^2 - z)/2 \tag{4.209}$$

$$z = \sqrt{(\sigma_N^2 - \sigma_E^2)^2 + 4\,\sigma_{NE}^2} \tag{4.210}$$

For absolute error ellipses the covariance values are extracted directly from the adjusted covariance matrix (Eq 4.125) to form the local covariance matrix (Eq 4.204). For relative error ellipses the covariances of both positions are extracted from the adjusted covariance matrix to form a 4 x 4 relative covariance matrix, then are propagated to form a local covariance matrix:

$$C_r = \begin{bmatrix} \sigma_{N_1}^2 & \sigma_{N_1E_1} & \sigma_{N_1N_2} & \sigma_{N_1E_2} \\ \sigma_{N_1E_1} & \sigma_{E_1}^2 & \sigma_{E_1N_2} & \sigma_{E_1E_2} \\ \sigma_{N_1N_2} & \sigma_{E_1N_2} & \sigma_{N_2}^2 & \sigma_{N_2E_2} \\ \sigma_{N_1E_2} & \sigma_{E_1E_2} & \sigma_{N_2E_2} & \sigma_{E_2}^2 \end{bmatrix} \tag{4.211}$$

$$C = B\, C_r\, B^T \tag{4.212}$$

$$B = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \tag{4.213}$$

Once the local covariance matrix is formed, the relative error ellipse parameters are calculated using Eqs (4.205 – 4.210).

Sets of pseudo-observations, between points of interest, can also be extracted post-estimation. The values of these pseudo-observations are the coordinate deltas between the points. Sets of pseudo-observations then form interstation vectors. The variances of these pseudo-observations are calculated by forming a relative covariance matrix (Eq 4.211) and propagating it to a local covariance matrix (Eq 4.212). For 3D estimations the relative covariance matrix is extended to 6 x 6 by including elevation variances and covariances, and the propagation matrix (Eq 4.213) is extended to 3 x 6.

Error Ellipses

The Error Ellipse Class extends the Ellipse Class from the geometry package, but is an abstract class, with two implementations: Absolute Error Ellipse and Relative Error Ellipse. Error Ellipse is placed in the middle here so that it can provide methods for calculating confidence regions to its implementations.

In order to calculate these confidence regions, degrees of freedom (*df*) and confidence interval (1-α) are required. Therefore, the Error Ellipse Class retains the degrees of freedom value from the estimation that spawned it, and confidence interval is a required parameter when the confidence region is requested. Confidence regions are calculated using the Fisher Distribution:

$$a_c = \sqrt{2\ \lambda_1\ \xi_{F_{2,df},1-\alpha}} \tag{4.214}$$

$$b_c = \sqrt{2\ \lambda_2\ \xi_{F_{2,df},1-\alpha}} \tag{4.215}$$

Where $\xi_{F_{2,df},1-\alpha}$ is the Fisher Inverse Function with the first degrees of freedom fixed at 2, and 1 – α is the confidence interval. The direction of the semi-major confidence interval is the same as the direction of the semi-major axis (β).

The two implementations differ in the number of points attached and the method in which the vertex is obtained. Absolute error ellipses use the vertex of their point, while relative error ellipses create a vertex by averaging the coordinates of their points.

Factories

Two factory classes have been created to manage the logic of preparing estimations and utilizing the results. The Helmerts 2D Factory manages the usage of the Helmerts 2D Estimation Class and can perform the resulting transformation on a list of vertices. The Local Projection Factory manages usage of both the Localization Parameter Estimation and the Localization Height Parameter Estimation Classes and constructs a new local projection object from the results.

Both of these factories act on lists of constraints, however they primarily utilize PointPair constraints. PointPair constraints can be horizontal only, vertical only or standard. Both factories parse the list of constraints provided to them and generate lists of horizontal and vertical coordinates constraints from the PointPair constraints. When scale or rotation constraints are encountered, by either factory class, they are passed through to the relevant estimation unaltered. Both factories count the numbers of horizontal and vertical point-pairs while parsing the constraint lists and allow these numbers to dictate the procedure followed.

The Helmerts Transformation Factory requires a minimum of two horizontal point-pairs to perform an estimation. If only one horizontal point-pair is detected, then the horizontal shift between the points is calculated by subtraction then the scale is set to 1 and rotation is set to 0, unless dictated by constraints. If no point pairs, horizontal or

vertical, are detected, the factory throws an error. The vertical shift is calculated as the simple mean of the of the shifts between vertical point pairs regardless of the number of vertical point-pairs.

The Local Projection Factory also requires a minimum of two horizontal point-pairs to perform the horizontal parameter estimation. When only one horizontal point-pair is provided the geodetic and projection origin coordinates are taken directly from the point-pair, then the scale and rotation are set to 1 and 0 respectively, unless overridden by constraints. If no horizontal point-pairs are provided, then an error is thrown.

Three vertical point pairs are required for the Local Projection Factory to perform the height parameter estimation. If two vertical point pairs are provided then the factory calculates the pseudo-undulation (N') for both points and averages them, while the pseudo-deflections of vertical (ξ' & η') are set to 0. If one vertical point-pair is provided, then the pseudo-undulation of that point is used. Finally, if no vertical point pairs are provided then the actual geoid undulation (N) of the origin point is used.

After all estimations are performed and parameters are collected the two factories have different strategies for sharing the result. Because transformations are only expected to be performed once, the Helmerts 2D Factory retains the transformation

parameters and performs the transformation itself. Conversely, a local projection is expected to persist and be utilized by many other processes, so the Local Projection Factory creates and returns a local projection object.

4.4 Tests

Tests are an important part of software development. They are also an important part of academic research. As this project is using software development as a means to perform academic research, two separate types of tests are required. The terminology of test becomes ambiguous as the term “integration test” has a specific meaning within software development but would also be the best description of the tests required for the academic research component of the project. To alleviate this ambiguity, the term “academic test” is used to describe the tests designed to satisfy the academic research, although in many of the artifacts developed early in the project “integration test” was used to describe these academic tests. The term “integrity test” is used to describe the tests performed as part of the software development process.

4.4.1 Integrity Tests

Three types of tests fall into the integrity test category: unit tests, integration tests, and regression tests. The difference between unit and integration tests is the scope of the code tested. Unit tests refer to tests performed to validate a single component and integration tests refer to tests performed to validate the cooperation between

components. Regression tests refer to tests performed to ensure the continued validity of previously tested components.

Integrity tests are developed in each iteration of the construction phase. During the initial iteration, nearly all tests developed are unit test. As the number of iterations increases the dependency between components increases and the prevalence of integration tests correspondingly increases. These tests are largely developed in parallel with the components they test. Once the relevant components are developed and tests are performed successfully, the tests are moved to regression test suites. These regression test suites can be run after any maintenance is performed on the relevant components to detect unforeseen effects of any changes during maintenance.

In this project regression test suites are assembled for each of the packages discussed in section 4.3 and contain 131 tests. Each of these tests was developed in conjunction with the components they test, to verify the expected behavior during development. Several of these tests use values obtained from numerical examples in texts. Several other tests, particularly cogo package tests, compare forward and inverse calculations to validate operations. A few tests, particularly in the least squares package tests, do not have known correct values and simply use values from previous tests to detect any changes. Many tests use values obtained from other software packages to validate results.

The coordinates system package tests use results from several other geodetic software tools for validation including: NRCan's TRX, NRCan's GPS-H, NRCan's INDIR, NGS's DEFLEC12A, NGS's GEOID12A, NGS's NCAT, and UNAVCO's Plate Motion Calculator.

The least squares package test uses results from Autodesk's Civil 3D least squares adjustment tool to verify the adjustment of a network, although the results (Table 4.5) are imperfect. It is believed that the differences are caused by the different geodetic reductions and observation weighting schemes performed by the packages rather than by any errors, as the differences would be expected to be considerably larger if errors were occurring.

Table 4.5: Civil 3D Adjustment Comparison

Mean ΔN (m)	**Mean ΔE (m)**	**RMS ΔN (m)**	**RMS ΔE (m)**	**Max ΔN (m)**	**Max ΔE (m)**
-0.0005	0.0004	0.0008	0.0011	0.0013	0.0023

4.4.2 Academic Tests

Once development and integrity testing of the geodetic library is complete, a set of data integration tests is run to compare data integration strategies. For each strategy, a series of tests is performed by integrating discrete data sets into a network one at a time. Each series is run incrementally to simulate what would be the state of the network in the database of the full system after each integration. The state of the network and a summary of the integration is exported to a file after each integration so that each network state can be compared across the different integration strategies.

Five series are run using five strategies: (*i*) All Observations, (*ii*) Fixed Points, (*iii*) Weighted Point, (*iv*) Free Network, and (*v*) Pseudo Observations. Each of the series starts with the same first adjustment, using one fixed point and two weighted points as constraints. For each subsequent integration the constraints are determined by the strategy being tested. The order that datasets are integrated in is the same for each series and all estimations are limited to two dimensions.

The All Observations strategy (*i*) mimic the strategy used by existing MBGIS implementations. It combines all observations, from all previously integrated datasets, into a set and performs an estimation of all points in the network. This is expected to be the most rigorous method of integration and is used as the precision baseline for each of the alternative strategies. This is also expected to be the most computationally expensive method. The cost of this method will limit the potential size of the network as the processing times will quickly become unmanageable as the network grows.

The Fixed Points strategy (*ii*) uses previously adjusted points to apply fixed point constraints to the current adjustment. This strategy prevents any change in coordinates of existing points, which prevents any discontinuities in the integrated data, but may also degrade the accuracy of the network. Any error in the position of points in their initial adjustment will not be reduced in future integrations and will be propagated throughout later datasets, potentially biasing or distorting them.

The Weighted Point strategy (*iii*) uses any previously adjusted points as weighted point constraints for the current adjustment. This strategy will allow some movement of previously adjusted points, allowing subsequent adjustments to refine the position of control points, and reducing the distortions caused by errors in initial control point position. This movement will also cause discontinuities in previously adjusted datasets as the positions of control points are updated and positions of other points are not.

The Free Network strategy (*iv*) adjusts each dataset independently using free network constraints, then performs a 2D Helmert's Transformation to bring the dataset into coincidence with the existing network. This strategy will eliminate distortion of new datasets but will still have issues with discontinuities and may cause problems with relative GNSS measurements. The free-network constraints use a no-net-rotation constraint to constrain the dataset. Total station measurements sometimes need to be rotated during estimation and in a total station only dataset it causes no problem. In a dataset containing both total station and GNSS measurements the no-net-rotation constraint may rotate GNSS measurements to compensate for the rotation caused by the total station measurements. Even if the estimation does not rotate the GNSS measurements, the Helmert's Transformation will almost certainly rotate them. Rotating relative GNSS measurements does not fit with the measurement properties. In the best-case scenario rotation reduces the measurements from distance and orientation to distance only and in most situation is likely to cause distortions.

The Pseudo Observation strategy (*v*) maintains a sub-network of control points and extracts vectors and relative variances between the control points after each estimation, then uses these pseudo-observations to constrain the subsequent dataset. It is expected that this strategy will provide a reasonable balance between the attributes of the other strategies. The accuracy of this strategy is expected to be similar to that of the Weighted Point strategy. Issues with inconsistencies would still exist, although it is expected that these will be reduced as all control points in the network would be affected, not only the control points observed in the new dataset. The Pseudo Observation strategy will also be subject to processing time restrictions just as the All Observations method is. The size of the control network will grow much slower than the full network, so the processing times are expected to increase slower, however eventually a similar processing time restriction will be reached.

Once all test series are processed, the results are summarized by comparing the network generated after each integration of the All Observations series against the equivalent network generated in each of the series. For each point in each All Observation network, the equivalent point is found in the alternative strategy network and the coordinates of the point are differenced. For each alternative network, six values are determined: Mean Δ Northing, Mean Δ Easting, RMS of Δ Northings, RMS of Δ Eastings, Maximum Δ Northing, Maximum Δ Easting. Tables are generated for each of the alternative strategies, listing these values for each state of the network.

4.4.3 Academic Test Data

The data sample used to perform the academic tests is a collection of 80 discrete datasets containing total station and RTK measurements. The datasets are mostly organized into daily datasets spanning approximately 4 months. The area covered by this data is very small, only about 2 hectares. Two of the datasets represent control network surveys with extremely high redundancy. The rest of the data sets have low internal redundancy, although the redundancy across datasets is very high.

This data was collected in the process of normal land surveying activities and was not intended for this purpose. The data was exported from the native data collector format to an ascii measurement file format (.fbk) which appears to have introduced some errors. Because of these factors, some preparation and cleaning of the data is required.

The first several datasets used points from a previous survey as control points, until the first control network survey was performed. These points used a combination of numbers and letters as point names. The native data collector format appears to support alphanumeric point names, but the ascii measurement file format does not. In the export process, the data collector assigned numeric point names to the control points but assigned different names to points in each dataset. The first step in cleaning the datasets was to rename these control points in the measurement files as to have consistent control point names through all datasets.

As the data was not collected for this purpose, no consideration was given to least squares estimation during data collection. Most redundant measurements to control points used new point names for existing control points and had descriptions like “Check Control”, “Check”, “Check #”. The second step in clearing was identifying and renaming these points so that the system would recognize the redundancy.

The ascii measurement file format was originally intended only for total station measurements. The data collector uses an annotation to include the GPS base station and then uses the add point command to include rover measurements. Unfortunately, the data collector neglects to include the base station height in the export. The base station height was recorded in the daily field notes so an additional annotation, recording base station height, was manually added to each dataset (Figure 4.16).

```
 8  NEZ 1 -150.0000 0.0000 611.8600
 9  NEZ 2 -0.0600 0.0000 610.9700
10  NEZ 3 0.1710 -99.9400 613.2500
11  NEZ 4 -45.8900 -89.9200 614.1410
12  NEZ 5 -149.9200 -130.0000 614.7900
13  NEZ 6 -45.8500 -208.9900 616.2000
14
15  !GPS Reference station: NEZ 5 -149.9200 -130.0000 614.7900
16  !GPS Reference height: 0.759
17
18  !Instrument Model: CS15
19
20  !GPS Survey 5.61
21  !Antenna height: 2.000
22  NEZ 4 -45.8877 -89.9350 614.1329 "11"
```

Figure 4.16: Example of GPS Annotations in Measurement Files

In some cases, more than one dataset was collected on a single day. Some of these were to separate the control network survey from the other data collected in the same day. In one case a second dataset was recorded due to corruption of the first dataset. In every case this led to incomplete data in at least one of the datasets. These incomplete datasets were removed from the sample.

In a few cases, the measurement file recorded total station occupations, but did not record any backsight point or direction. This causes ambiguity in the position of the foresight points as no direction can be determined. In every case the entire record of the total station occupation was removed from the dataset to prevent this ambiguity.

In many cases, the measurement file recorded a total station occupation defined by resection from control points. In a few of these cases no checks to control points were performed after the resection was complete. The geodetic library considers the resection a different measurement set from any measurements taken after the resection. This causes an issue where the system is unable to determine a backsight direction for the second measurement set. This is a short coming of the system and should be corrected in the future, but at this time these occupations were removed from the dataset to prevent ambiguity.

Two datasets were removed from the sample due to their size and lack of redundancy. These datasets each contained over 2000 RTK measurements of the ground to determine topography at the beginning and end of the project. It was determined that these datasets would drastically increase the processing time of the All Observation method and added nothing of value due to said lack of redundancy.

Free Network adjustments were performed on each dataset individually, to detect blunders. Most of the blunders detected were due to errors in the manual point name changes performed during the cleaning process. These name-change blunders were corrected without needing to remove any data. The system was consistently unable to adjust one of the datasets and the issue could not be identified, so the dataset was removed from the sample.

The first control network survey was originally the 14^{th} dataset and several of the control points had been observed with RTK prior to this dataset. This dataset was moved to be the first dataset, so that the coordinates of control points would not be fixed at their RTK determined coordinates by the Fixed Point strategy, as this would likely cause major distortions of the network.

5. Results

Five test series were run. Each test series produced a network of 5630 points, so there is too much data to display point by point comparisons. As described in section 4.4.2, summaries were created by using the All Observations series as a baseline for each of the other series. Tabulations of the summaries can be found in Appendix A. The legend shown in Figure 5.1 displays the colors of the test series displayed in Figures 5.2 to 5.8.

Figure 5.1: Result Graph Legend

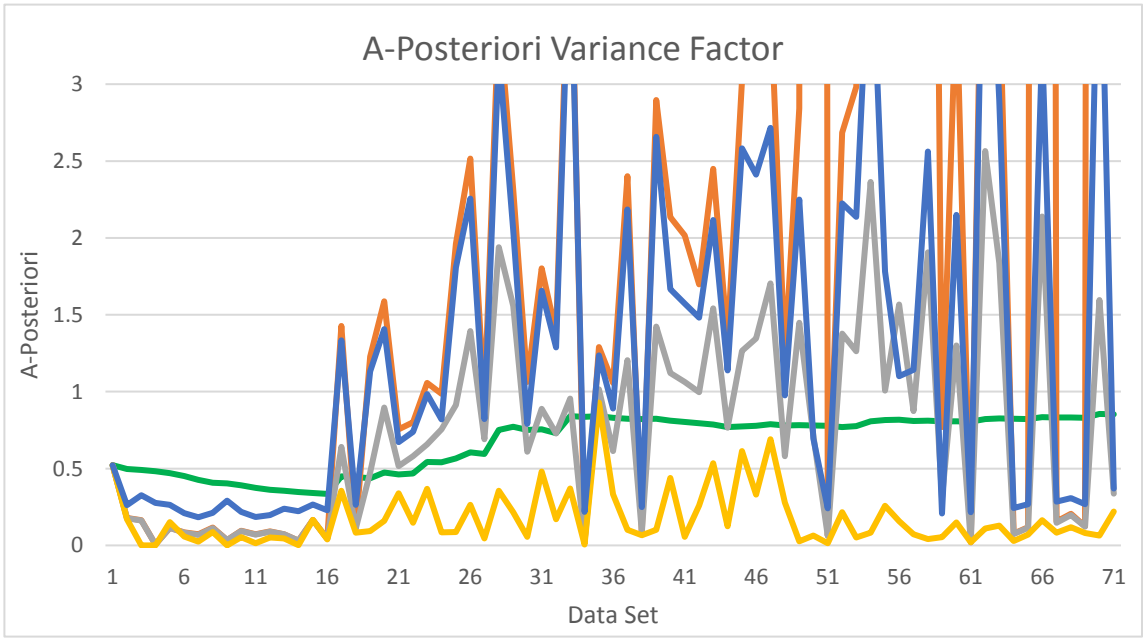

Figure 5.2: A-Posteriori Variance Factors of All Series

As Figure 5.2 displays, the a-posteriori variance factors produced by the All Observation test series are relatively stable. The first data set, containing the first network survey, produced a low a-posteriori of only 0.522. The next several data sets, consisting mostly of RTK measurements with little redundancy, see the a-posteriori drop slightly as more measurements are added without increasing redundancy or residuals. The a-posteriori begins climbing again at data set 16, when total station measurements and redundancy begin increasing. Around data set 32, containing the second network survey, the a-posteriori levels off around 0.84 and holds for the remainder of the series.

The Free Network series generally produces the lowest a-posteriori, as would be expected. The purpose of this method is to produce the Best Linear Unbiased Estimate (BLUE) which, by definition, should minimize the a-posteriori (Ogundare, 2012).

Each of the other three series has higher variability of a-posteriori, particularly the Fixed Point series. This is also to be expected. The uncertainties of the total station observations are low. The Weighted Point and Pseudo Observation series produce very low uncertainty of coordinates, and in the Fixed Point series the uncertainty of coordinates is effectively zero. Under these conditions, small discrepancies due to random effects can produce high a-posteriori values. Unfortunately, as shown in Eq 4.113, the a-posteriori directly affects the estimated uncertainties, so this variation in a-posteriori has a large influence on the estimated uncertainty of coordinates.

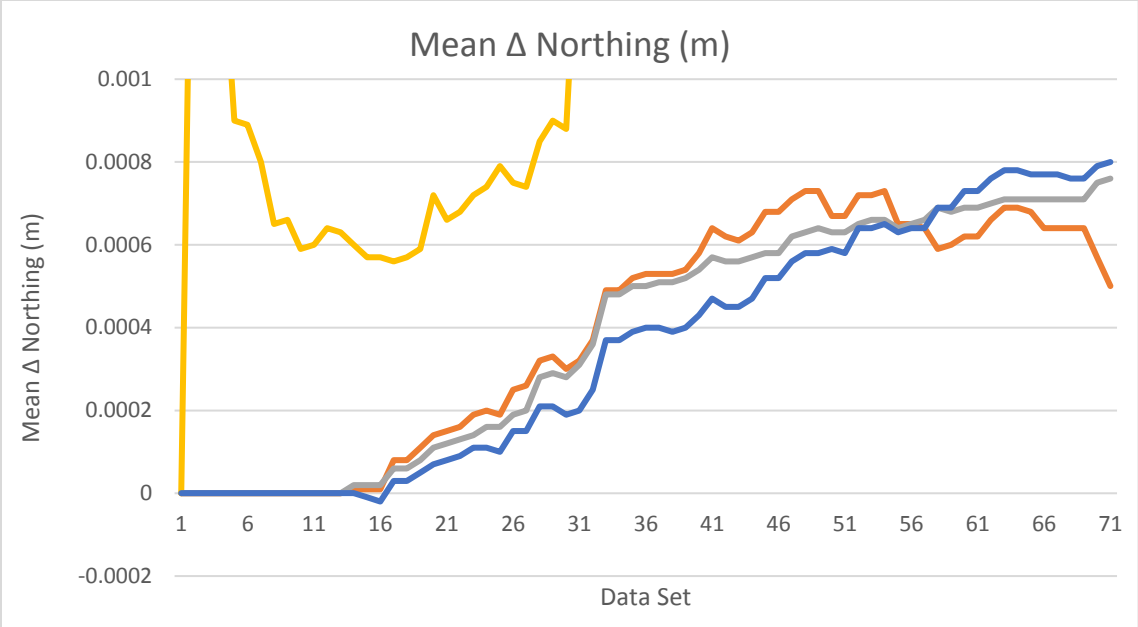

Figure 5.3: Mean Δ Northing between Test and Baseline Series

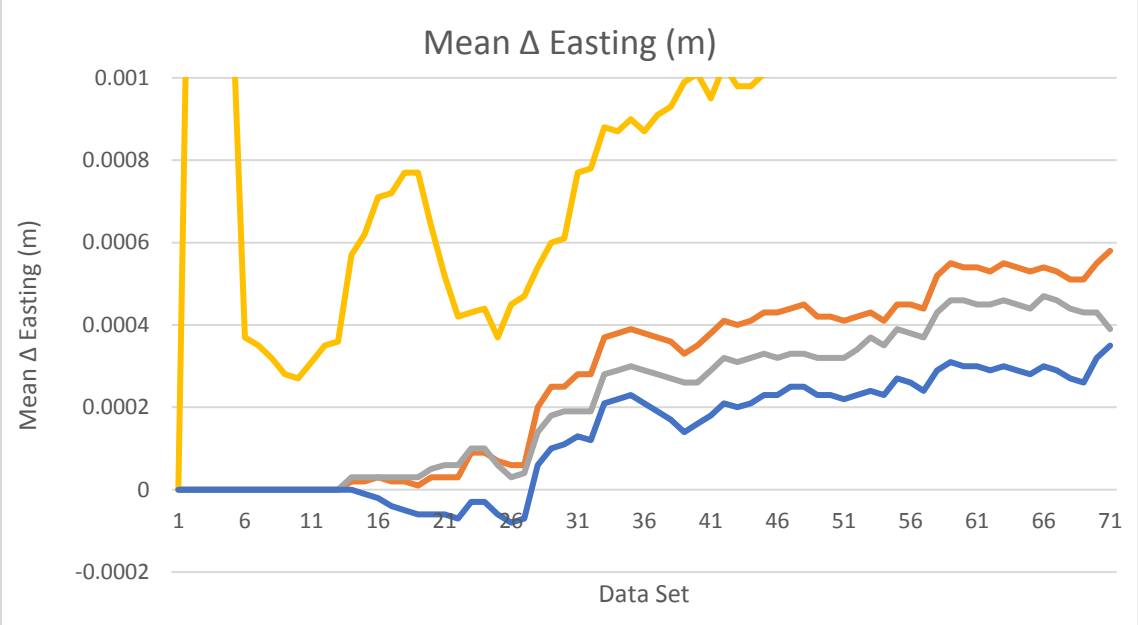

Figure 5.4: Mean Δ Easting between Test and Baseline Series

Figures 5.3 and 5.4 display the mean Δ of the respective coordinate between the All Observation series and each of the other series. These values are meant to check the bias of the entire network and could be either positive or negative. In this case the biases happen to be positive for throughout each series except for a brief period in the Pseudo Observation series. The network biases calculated are sub-millimeter, in each dimension, for three of the series. The Free Network series exceeds the millimeter level immediately then displays a significant amount of variation over the series.

Figures 5.5 and 5.6 display the RMS of the respective coordinate Δ's between the All Observation series and each of the other series. These values provide an overall check on the difference of coordinates as the squaring of the Δ's prevent them from cancelling.

The Free Network series again produces large values immediately, then displays large variations. Each of the other three series initially produce small values that slowly accumulate. The Weighted Point produces the best results for most of the series, but both it and the Pseudo Observation series produce maximum values of approximately 1.5 mm in each dimension. The Fixed Point accumulates only slightly faster than the Weighted Point and Pseudo observation series, producing maximum values of 2.28 mm in Northing and 1.66 mm in easting.

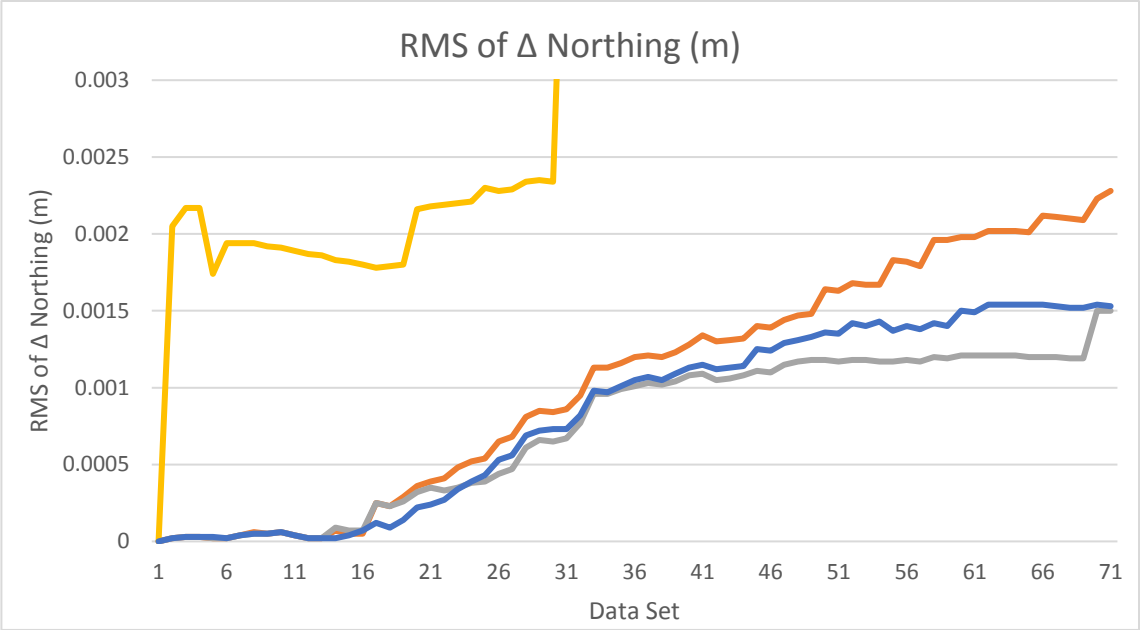

Figure 5.6: RMS of Δ Northing between Test and Baseline Series

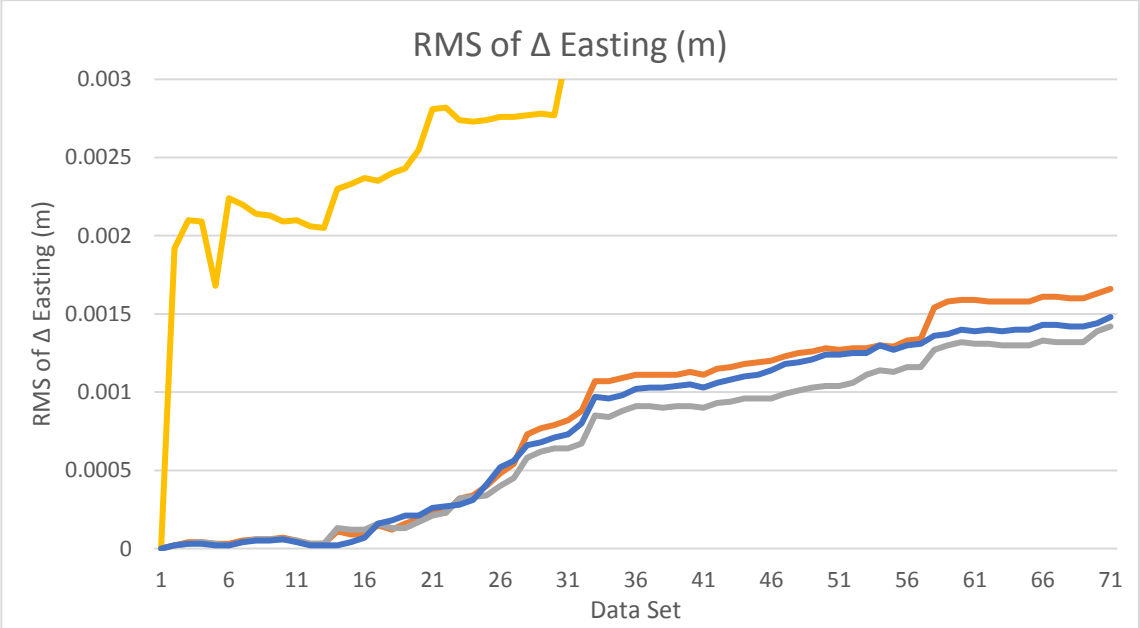

Figure 5.5: RMS of Δ Easting between Test and Baseline Series

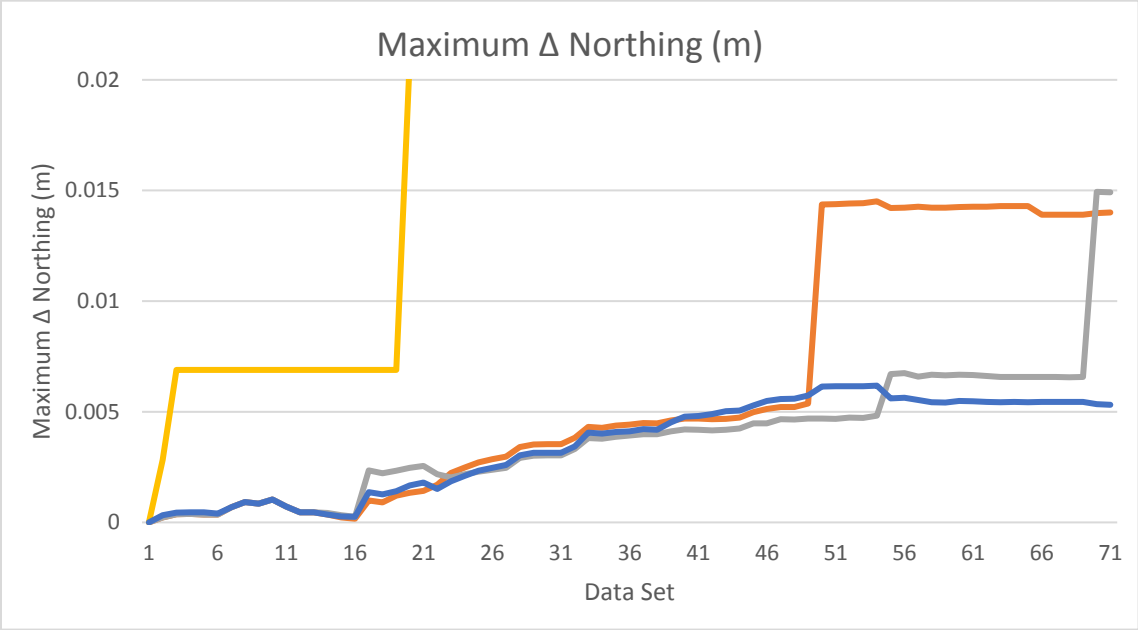

Figure 5.7: Maximum Δ Northing between Test and Baseline Series

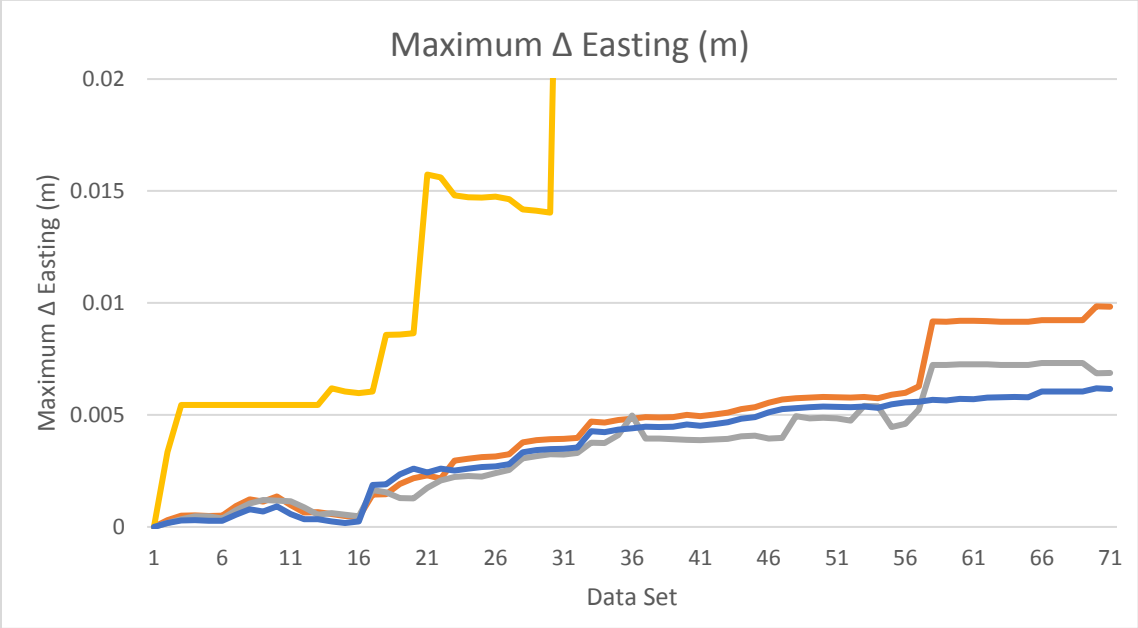

Figure 5.8: Maximum Δ Easting between Test and Baseline Series

Figures 5.7 and 5.8 display the maximum Δ of the respective coordinate between the All Observation series and each of the other series. These are to identify the worst differences between each network and the baseline network. Once again, the Free Network series displays more variability than the other series. The other three series appear to display similar gradual accumulations again, until the Fixed Point and Weighted Point series experience immediate jumps. The Pseudo Observation series displays the most consistency with values in both dimensions capped around 6mm. All other series display maximum values reaching into the centimeter range.

The processing times for each test series is displayed in Table 5.1. The difference in processing times between the All Observations method and the alternative methods is vast. While the system required more than 90 hours to process the All Observations series, it required less than 2 minutes to process each of the other series. The final test in the All Observations series integrated all 71 data sets, had 13788 observation and 11471 parameters, and took 13650 seconds (~4 hours). The computer running the tests was equipped with a 3.2 GHz octo-core processor, 32 GBs of RAM, and Windows 7.

Table 5.1: Test Series Processing Times

Test Series	**Processing Time (s)**
All Observations	326413
Fixed Points	59
Weighted Points	65
Free Network	85
Pseudo Observations	67

6. Discussion

The objective of this project is to test Bartosh's (2012) hypothesis that a Measurement-Based GIS (MBGIS) implementation could be used to improve the data management of land survey data. Navratil (2004) and Bartosh (2012) have previously tested MGBIS implementations using data from property maps. Both these studies found issues with the implementations they were testing, particularly the processing times and flexibility of the estimation. A third issue was inferred from these studies, being the reduction of measurements to the mapping plain.

This project prototypes a new implementation of MBGIS as a suite of data integration tests. A geodetic library and several adjustment strategies are implemented and tested in order to explore the issues identified in existing MBGIS implementations. The library implementation addresses the issue of measurement reductions, while the processing time and flexibility issues are explored by the comparison of the adjustment strategies.

The results of the data integration tests show that the processing times can be drastically reduced by integrating data sets one at a time, but this time savings comes with a precision penalty. The final adjustment of the All Observation method took nearly 4 hours to process, while the entire series of each alternative strategy was processed in less than 2 minutes.

The All Observations method is used as the precision baseline for the alternative strategies as it the most rigorous method of network adjustment (Lugoe, 1985). This method can be quite flexible, depending on the available options for constraints and weight modifiers. The only drawback to this method is the processing time, which does pose a problem for MBGIS as the size of the network grows.

The Free Network strategy provided the worst results of the alternative strategies. The precision metrics were so poor that the results graphs had to be rescaled to ignore the results, because without this rescaling the differences between the other strategies are nearly imperceptible. This strategy also requires the longest time to process, although the processing times are not so bad as to disqualify this strategy on this factor alone. This method is arguably the most flexible of the alternative strategies, providing the best a-posterior variance factor in almost all cases. Due to the poor precision of the networks produced by this strategy it will be largely omitted from further discussion.

The Fixed Point strategy required the least time to process, requiring only 59 seconds to process the entire series. This strategy created networks with similar mean delta and RMS of delta values to those created by the Weighted Point and Pseudo Observation strategies, although it typically had the largest values of the three. The maximum delta values for this strategy saw larger jumps than the other strategies. Unsurprisingly, the largest issue with this strategy is flexibility. As the coordinates of

control points are fixed in the adjustment, the existing network becomes rigid and the new dataset is forced to distort. This causes high observation residuals and high a-posteriori variance factors.

The Weighted Point strategy provided the best RMS of delta values and good mean delta values. The maximum delta values were reasonably good until sudden jumps occurred, although these jumps were less than those observed for the Fixed Point strategy. The processing of this strategy takes only 10% longer than the Fixed Point strategy at 65 seconds for the full series. This strategy is reasonably flexible, allowing for the coordinates of control points to change slightly to accommodate new observations, although this change causes small discontinuities between control points as only points observed in the current dataset are adjusted.

The Pseudo Observation strategy provides the best mean delta values for most of the series, although the northing value drifts above Fixed Point and Weighted Point strategies towards the end. The RMS of delta values and maximum delta values are consistent with the other strategies through out the series, although this is the only strategy that does not experience any sudden jumps in maximum deltas. Processing of this strategy takes only 13% longer than the Fixed Point strategy at 67 seconds, although it is suspected that the processing times of this strategy would increase faster than the other strategies for larger networks. This strategy also seems fairly flexible,

allowing the coordinates of control points to change and propagating this change through out the control sub-network.

Comparing the alternative integration strategies, it seems apparent that the Free Network strategy is not an optimal strategy. The networks generated by this method differ from the networks generated by the All Observation method by considerably more than the other strategies. The Fixed Point strategy also appears to be a sub-optimal strategy. Although the network results of the Fixed Point strategy are nearly as good as the other two strategies, and the processing time is better, the issues with flexibility would be problematic for long-term data management.

The comparison between the Weighted Point and Pseudo Observation strategies is more difficult to make. The precision of these two strategies seems nearly equal, as do the processing times for the datasets tested. It is expected that the processing times for the Pseudo Observations strategy would increase faster than the Weighted Point strategy for much larger networks. It is also expected that that the Pseudo Observation strategy would maintain the control sub-network better than the Weighted Point strategy, resulting in better precision, for much larger networks. A larger collection of test data is required in order to test these expected effects and make a better comparison between these strategies.

The results of testing the prototype MBGIS appear to support the assertion that a MBGIS could be used to manage land survey data. It was able to integrate discrete sets of unreduced survey measurement data into a single network of survey points using several different strategies. The implementation of the geodetic library addresses issues with reduction of measurements, and the varied adjustment strategies addresses the issues with processing time and flexibility, although none of the strategies provides a perfect data integration method.

There are limitations on the validity of the tests due to the small spatial area, short time period, and measure types of the test data. The test data only covers an area of approximately 2 hectares, and all observations were taken within a four-month period. The data management system would be expected to manage networks covering much larger areas, and with much larger differences between measurement epochs. Large increases to the area or difference in measurement epochs would be expected to cause further loss of precision. The geodetic reductions performed by the library should constrain this loss of precision, but more test data would be required to verify this. The test data also only contains total station and RTK measurements. Many of the total station measurements were taken without regard for elevation, so the integration tests need to be limited to 2D estimations. Better test data would include proper height records for total station data, leveling measurements and PPP measurements.

The prototype also has limitations that would be addressed by further development of the system. The integrated networks created by the prototype are not very useful as they are processed in system memory then written to ascii files. Implementation of the server module would allow the data to be queried and would provide much more utility. The prototype was not able to process raw data. Several data cleaning steps were required before the data could be integrated. Implementation of the mobile client module and specification of data collection standards could address most of the issues encountered in section 4.4.3. The mobile client could ensure that all required data was collected and could perform some verification of the data to detect blunders. Data collected with integration in mind would follow data collection standards that address issues with point naming, so the system could detect redundancy.

The results indicate that a measurement-based data management system as proposed by Bartosh (2012) is possible, although trade-off between processing time and precision is required for networks with large numbers of points and observations. A fully developed system would likely provide several benefits to land survey firms that choose to use it, particularly in reducing redundant work (Bartosh, 2012). Unfortunately, a data management system on the scale proposed by McKay (2018) is probably not possible with these data integration methods. An integrated National Spatial Data Infrastructure would likely require an integration method capable of handling massive datasets with no loss of precision.

7. Conclusion

Land survey firm often have issues with data management. The ad hoc data management procedures they use often make it difficult to reuse previously collected data or identify redundant work. The primary interest of this thesis was to explore options that could improve the data management systems employed by land surveyors. Towards this end, academic literature was reviewed and Bartosh's (2012) proposal to use a Measurement-Based GIS (MBGIS) to manage land survey data was selected for further study. Previous studies of MBGIS had found issues with processing times and flexibility of existing implementations. Because a suitable MBGIS implementation was not available, a prototype MBGIS was implemented. A suite of data integration tests was developed, and the results were analyzed on metrics of processing times; precision; and flexibility, in order to draw conclusion on the ability of MBGIS to act as a land survey data management system.

The prototype MBGIS that was implemented was limited to the suite of data integration tests and the geodetic library that the tests depended on. However, in order to design the geodetic library within the context of a full MBGIS based data management system, conceptual designs were created for the full system. This conceptual design included several modules: a web server, a mobile client, a desktop client, application plugins, and the geodetic library. The web server was designed to

manage a central database, where all datasets could be integrated, and provide this integrated data to the various clients. The mobile client was designed to collect datasets, verify the data, and send it to the server to be integrated. The desktop client and application plugins were designed to allow access to the integration functions and integrated data. Finally, the geodetic library was designed to provide the datatypes and functions that each of the other modules needed to perform their operations.

Implementation of the geodetic library was the largest component of this project. Numerous datatypes were created, and numerous geodetic functions were implemented in order to ensure the correct reduction of field measurements to the mapping plane. Functions were also developed to perform geodetic coordinate geometry calculations, record keeping, import and export of raw and processed data, and a variety of least squares estimations. The data integration tests relied heavily on the geodetic library. Field measurements were imported and processed using library functions. Data integrations were performed using the least squares functions of the library and results were exported to coordinate files using library functions.

The data integration tests focused on comparing the results of several different data integration strategies. The first strategy was termed “All Observations” and represents the method used by existing MBGIS implementations. It collected the observations from all included datasets and performed a least squares estimation using this large set

of observations. The second strategy was termed “Fixed Points” and used the previously estimated coordinates of control points as fixed values to constrain subsequent estimations. The third, “Weighted Point” strategy, uses the previously estimated coordinates of control points as weighted parameters to constrain subsequent estimations. The fourth strategy “Free Network”, uses free network constraints (no net rotation, and no net translation) to estimate datasets individually, then uses a 2D similarity transformation to best-fit the dataset to the existing control points. The final strategy “Pseudo Observations”, extracts vectors and relative variances between control points to constrain subsequent estimations.

The All Observations strategy is the most rigorous, so the networks generated by it were used as the baseline for comparison of the precision of the alternative strategies. As expected, processing times for the All Observations strategy were prohibitively long. All the alternative integration strategies saw a drastic reduction in processing times.

The processing times, precision, and flexibility of the alternative strategies were compared to determine their suitability as methods of data integration for a measurement-based data management system. The Free Network strategy performed the worst, with the lowest precision and longest processing time. The Fixed Point strategy was also deemed unsuitable, although it had reasonable precision and the fastest processing time, due to its inflexibility. The performance of the Weighted Point

and Pseudo Observation were similar when integrating this sample of test data. A larger sample of data encompassing a larger geographic area is likely needed to differentiate these strategies.

The results indicate that the processing time and flexibility issues facing existing MBGIS implementations could be mitigated by using alternative data integration strategies, although a precision penalty would be incurred. The test data used in this study only encompassed a small area, so the precision penalty may grow for larger test areas, although this loss of precision should be constrained by rigorous geodetic reductions.

A MBGIS system, using rigorous geodetic processing and the alternative data integration strategies discussed, could be a viable data management system for land surveyors. Usage of such a system would likely improve the efficiency of survey operations by allowing surveyors to identify redundant work and reuse existing data.

Future works could include further development of the designed system and further testing, using samples of data with larger spatial extents and more variety of measurement types. However, a more interesting avenue for future research would be the development of new data integration methods, focusing on reduced processing times and increased flexibility, while maintaining the precision of the All Observation strategy.

References

Altamimi, Z., Metivier, L., & Collilieux, X. (2012). ITRF2008 plate motion model. *Journal of Geophysical Research: Solid Earth, 117*(7), 1-14.

Android. (2017). *Meet Android Studio*. Retrieved from developer.android.com: https://developer.android.com/studio/intro/index.html

Android. (2017a). *NDK Revision History*. Retrieved from developer.android.com: https://developer.android.com/ndk/downloads/revision_history.html

Android. (2017b). *About the Android Open Source Project*. Retrieved from source.android.com: https://source.android.com/

Android. (2017c). *API Guides*. Retrieved from developer.android.com: https://developer.android.com/guide/index.html

Bartosh, C. (2012). Integrating Land Surveying Data into Measurement-Based GIS: An Assesment of the Challenges and Practical Solutions for Surveyors in Texas. *Masters Thesis*. University of Southern California.

Belussi, A., & Migliorini, S. (2012). A Framework for Integrating Multi-Accuracy Spatial Data in Geographical Applications. *GeoInformatica*, 523-561.

Bremner, M., & Santos, M. (2019). A Local Projection for Integrating Geodetic and Terrestrial Coordinate Systems. *Survey Review*.

Craymer. (1998). *ITRF-NAD83 TRANSFORMATION & DEFINITION OF NAD83(CSRS).* Retrieved from www.naref.org: http://www.naref.org/transf/nad83itrf.txt

Craymer, M. R. (2006). The evolution of NAD83 in Canada. *Geomatica, 60*(2), 151-164.

Engineers Canada. (2017). *Accredited Engineering Programs in Canada*. Retrieved from engineerscanada.ca: https://engineerscanada.ca/accreditation/accredited-programs

Friesen. (2013). *Learn Java for Android Development: Second Edition.* Berkley: Apress.

Goodchild, M. (2002). Measurement-Based GIS. In P. F. Fisher, *Spatial Data Quality* (pp. 5-17). London: Taylor & Francis.

Gradle. (2017). *Chapter 72. Building Native Software*. Retrieved from docs.gradle.org: https://docs.gradle.org/3.3/userguide/native_software.html

Heffelfinger, D. R. (2014). *Java EE 7 with Glashfish 4 Application Server.* Birmingham: Packt Publishing Ltd.

Hofmann-Wellenhof, B., & Mortiz, H. (2005). *Physical Geodesy.* Graz, Austria: SpringerWeinNewYork.

Hooijberg, M. (2008). *Geometrical Geodesy.* Berlin: Springer-Verlag Berlin Heidelberg.

Husaini, S. F. (1997). Using The Java Native Interface. *Crossroads*, pp. 18-23.

IETF. (2016). *leap-seconds.list.* Retrieved from www.ietf.org: https://www.ietf.org/timezones/data/leap-seconds.list

ITRF. (2010). *Transformation parameters from ITRF2008 to past ITRFs.* Retrieved from itrf.ensg.ign.fr: http://itrf.ensg.ign.fr/doc_ITRF/Transfo-ITRF2008_ITRFs.txt

ITRF. (2013). *ITRS and WGS84.* Retrieved from itrf.ensg.ign.fr: ftp://itrf.ensg.ign.fr/pub/itrf/WGS84.TXT

ITRF. (2016). *Transformation parameters from ITRF2014 to past ITRFs.* Retrieved from itrf.ensg.ign.fr: http://itrf.ensg.ign.fr/doc_ITRF/Transfo-ITRF2014_ITRFs.txt

Jacobson, I., Grady, B., & Rumbaugh, J. (1999). *The Unified Software Development Process.* Reading, Massachusetts: Addison Wesley Longman.

Jeffress, G. (2005). GIS and Surveying: Future Opportunities. *Surveying and Land Information Science*, 169-177.

JetBrains. (2017). *ItelliJ IDEA*. Retrieved from www.jetbrains.com: https://www.jetbrains.com/idea/

Kalali, M., & Mehta, B. (2013). *Developing RESTful Services with JAX_RS 2.0, Websockets, and JSON.* Birmingham: Packt Publishing Ltd.

Keith, M., & Schincarlol, M. (2006). *Pro EJB 3: Java Persistence API.* Berkeley: Apress.

Krakiwsky, E. J. (1973). *Conformal Map Projection in Geodesy.* Fredericton: University of New Brunswick.

Kruchten, P. (2000). *The Rational Unified Process An Introduction Second Edition.* Upper Saddle River, New Jersey: Pearson Education.

Leung, Y., Jiang-Hong, M., & Goodchild, M. (2004). A General Framework for Error Analysis in Measurement-Based GIS: Part 1: The Basic Measurement-Error Model and Related Concepts. *Journal of Geographical Systems*, 325-354.

Liu, F. (2013). *Android Native Development Kit Cookbook.* Birmingham: Packt Publishing Ltd.

Lugoe, F. N. (1985). *Rigorous Densification of Horzontal Geodetic Networks.* Fredericton, NB: University of New Brunswick.

Mallepally, H. (2016). *Agile Methodology: History*. Retrieved from agilemethod.net: http://www.agilemethod.net/p/waterfall.html

McCarthy, D. D. (1996). *IERS Conventions (1992).* France: Central Bureau of IERS.

McKay, D. K. (2018). Proposal for a National Spatial Data Infrastructure of Surveying Data. *Surveying and Land Information Science*, 21-31.

McRae, G. (2018). *The Intersection of Two Circles*. Retrieved from 2000clicks.com: http://2000clicks.com/mathhelp/GeometryConicSectionCircleIntersection.aspx

Mehta, B. (2014). *RESTful java patterns and best practices : learn best practices to efficiently build scalable, reliable, and maintainable high performance RESTful services.* Birmingham: Packt Publishing Ltd.

Navratil, G., Franz, M., & Pontikakis, E. (2004). Measurement Based GIS Revisited. *7th AGILE Conference of Geographic Information Science*, (pp. 771-775). Heraklion, Greece.

Ogundare, J. (2012). *Data Analysis And Least Squares Estimation.* Vancouver: BCIT.

Oracle. (2012). *Your First Cup.* Retrieved from docs.oracle.com: http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

Oracle. (2014). *Java DB Developer's Guide.* Retrieved from docs.oracle.com: http://docs.oracle.com/javadb/10.8.3.0/devguide/index.html

Oracle. (2014a). *What is Java and why do I need it?* Retrieved from www.java.com: https://www.java.com/en/download/faq/whatis_java.xml

Oracle. (2017). *Java SE Downloads*. Retrieved from www.oracle.com: http://www.oracle.com/technetwork/java/javase/downloads/index.html

Orso, A. (2014). Software Development Process. Udacity. Retrieved 2017, from https://www.udacity.com/course/software-development-process--ud805

Pilgrim, P. A. (2013). *Java EE 7 Developer Handbook.* Birmingham: Packt Publishing Ltd.

Rumbaugh, J., Jacobson, I., & Booch, G. (2005). *The Unified Modeling Language Reference Manual, Second Edition.* Boston: Pearson Education.

Sandoval, J. (2009). *RESTful Java Web Services.* Birmingham: Packt Publishing Ltd.

Savvaidis, P., & Stergioudis, A. (2012). From Desktop GIS to Web-Based Cloud GIS: The Globalization of Geospatial Data Management. *Modern Technologies, Education and Professional Practice in Geodesy and Related Fields.* Sofia.

Silver, J., & Vieira, M. (2015). Gradle for Android and Java. Udacity. Retrieved 2017, from https://www.udacity.com/course/gradle-for-android-and-java--ud867

Soler, T., & Marshall, J. (2003). A note on frame transformations with applications to geodetic datums. *GPS Solutions, 7*(1), 148-149.

SQLite. (2017). *About SQLite*. Retrieved from www.sqlite.org: https://www.sqlite.org/about.html

Stem, J. E. (1989). *State Plane Coordiante System of 1983.* Rockville MD: National Geodetic Information Center.

Takasu, T. (2013). *RTKLIB ver. 2.4.2 Manual.* Retrieved from www.rtklib.com: http://www.rtklib.com/prog/manual_2.4.2.pdf

Thomson, D. B., Krakiwsky, E. J., & Adams, J. R. (1978). *A Manual for Geodetic Position Computations in the Maritime Provinces.* Fredericton: University of New Brunswick.

Thomson, D. B., Krakiwsky, E. J., & Steeves, R. R. (1977). *A Manual for Geodetic Coordinate Transformations in the Maritime Provinces.* Fredericton: University of New Brunswick.

Thomson, D. B., Mepham, M. P., & Steeves, R. R. (1977). *The Stereographic Double Projection.* Fredericton: University of New Brunswick.

Torge, W., & Muller, J. (2012). *Geodesy: 4th Edition.* Gottinggen, Germany: De Gruyer.

Vanicek, p., & Krakiwsky, E. (1986). *Geodesy the Concepts.* North - Holland.

Vanicek, P., Zhang, C., & Ong, P. (1990). *Computation of a File of Geoidal Heights Using Molodenskij's Truncation Method .* Fredericton: University of New Brunswick.

Véronneau, M. (2017). Personal Communication.

Vranic, S., Jurakic, G., & Matijevic, H. (2014). Modelling And Disemination of Land Surveying Data. *6th International Conference on Engineering Surveying*, (pp. 149-154). Prague, Czech Republic.

Appendix A: Data Integration Summaries

Tables A.1 to A.5 displays summaries of the processing results and comparison to the baseline method after each integration. The All Observation strategy (Table A.1) provides does not display comparisons to the baseline method, as it is the baseline method.

Table A.1: All Observations Integration Summaries

Data Set	A-post	Mean ΔN (m)	Mean ΔE (m)	RMS ΔN (m)	RMS ΔE (m)	Max ΔN (m)	Max ΔE (m)
DataSet-00	0.522						
DataSet-01	0.497						
DataSet-02	0.491						
DataSet-03	0.482						
DataSet-04	0.470						
DataSet-05	0.451						
DataSet-06	0.427						
DataSet-07	0.408						
DataSet-08	0.403						
DataSet-09	0.390						
DataSet-10	0.375						
DataSet-11	0.362						
DataSet-12	0.355						
DataSet-13	0.347						
DataSet-14	0.341						
DataSet-15	0.335						
DataSet-16	0.450						
DataSet-17	0.445						
DataSet-18	0.437						
DataSet-19	0.474						
DataSet-20	0.461						
DataSet-21	0.468						
DataSet-22	0.542						
DataSet-23	0.540						
DataSet-24	0.565						
DataSet-25	0.605						
DataSet-26	0.595						
DataSet-27	0.750						
DataSet-28	0.772						
DataSet-29	0.751						
DataSet-30	0.755						
DataSet-31	0.731						
DataSet-32	0.840						
DataSet-33	0.837						
DataSet-34	0.841						
DataSet-35	0.829						

DataSet-36	0.824						
DataSet-37	0.822						
DataSet-38	0.823						
DataSet-39	0.811						
DataSet-40	0.803						
DataSet-41	0.794						
DataSet-42	0.787						
DataSet-43	0.770						
DataSet-44	0.773						
DataSet-45	0.777						
DataSet-46	0.788						
DataSet-47	0.779						
DataSet-48	0.782						
DataSet-49	0.779						
DataSet-50	0.778						
DataSet-51	0.769						
DataSet-52	0.775						
DataSet-53	0.807						
DataSet-54	0.815						
DataSet-55	0.817						
DataSet-56	0.810						
DataSet-57	0.811						
DataSet-58	0.807						
DataSet-59	0.808						
DataSet-60	0.806						
DataSet-61	0.821						
DataSet-62	0.825						
DataSet-63	0.824						
DataSet-64	0.822						
DataSet-65	0.834						
DataSet-66	0.833						
DataSet-67	0.832						
DataSet-68	0.830						
DataSet-69	0.855						
DataSet-70	0.852						

Table A.2: Fixed Point Integration Summaries

Data Set	A-post	Mean ΔN (m)	Mean ΔE (m)	RMS ΔN (m)	RMS ΔE (m)	Max ΔN (m)	Max ΔE (m)
DataSet-00	0.522	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
DataSet-01	0.179	0.00000	0.00000	0.00002	0.00002	0.00022	0.00030
DataSet-02	0.164	0.00000	0.00000	0.00003	0.00004	0.00037	0.00050
DataSet-03	0.006	0.00000	0.00000	0.00003	0.00004	0.00038	0.00052
DataSet-04	0.114	0.00000	0.00000	0.00002	0.00003	0.00036	0.00049
DataSet-05	0.084	0.00000	0.00000	0.00002	0.00003	0.00036	0.00050
DataSet-06	0.072	0.00000	0.00000	0.00004	0.00005	0.00069	0.00093
DataSet-07	0.115	0.00000	0.00000	0.00006	0.00006	0.00092	0.00123
DataSet-08	0.034	0.00000	0.00000	0.00005	0.00006	0.00085	0.00113
DataSet-09	0.096	0.00000	0.00000	0.00006	0.00007	0.00103	0.00136
DataSet-10	0.073	0.00000	0.00000	0.00004	0.00005	0.00072	0.00098
DataSet-11	0.091	0.00000	0.00000	0.00002	0.00003	0.00045	0.00064
DataSet-12	0.073	0.00000	0.00000	0.00002	0.00003	0.00046	0.00066
DataSet-13	0.031	0.00001	0.00002	0.00007	0.00011	0.00035	0.00058
DataSet-14	0.167	0.00001	0.00002	0.00005	0.00009	0.00023	0.00049
DataSet-15	0.044	0.00001	0.00003	0.00005	0.00009	0.00016	0.00043
DataSet-16	1.427	0.00008	0.00002	0.00025	0.00015	0.00099	0.00144
DataSet-17	0.112	0.00008	0.00002	0.00023	0.00012	0.00090	0.00146
DataSet-18	1.227	0.00011	0.00001	0.00029	0.00016	0.00121	0.00191
DataSet-19	1.588	0.00014	0.00003	0.00036	0.00019	0.00134	0.00218
DataSet-20	0.755	0.00015	0.00003	0.00039	0.00024	0.00143	0.00230
DataSet-21	0.801	0.00016	0.00003	0.00041	0.00023	0.00170	0.00216
DataSet-22	1.057	0.00019	0.00009	0.00048	0.00032	0.00224	0.00296
DataSet-23	0.987	0.00020	0.00009	0.00052	0.00034	0.00248	0.00305
DataSet-24	1.963	0.00019	0.00007	0.00054	0.00040	0.00271	0.00312
DataSet-25	2.516	0.00025	0.00006	0.00065	0.00048	0.00285	0.00314
DataSet-26	0.930	0.00026	0.00006	0.00068	0.00054	0.00297	0.00325
DataSet-27	3.466	0.00032	0.00020	0.00081	0.00073	0.00341	0.00377
DataSet-28	2.316	0.00033	0.00025	0.00085	0.00077	0.00352	0.00387
DataSet-29	1.063	0.00030	0.00025	0.00084	0.00079	0.00353	0.00392
DataSet-30	1.802	0.00032	0.00028	0.00086	0.00082	0.00353	0.00393
DataSet-31	1.380	0.00037	0.00028	0.00095	0.00088	0.00382	0.00398
DataSet-32	4.231	0.00049	0.00037	0.00113	0.00107	0.00431	0.00470
DataSet-33	0.023	0.00049	0.00038	0.00113	0.00107	0.00428	0.00466
DataSet-34	1.290	0.00052	0.00039	0.00116	0.00109	0.00437	0.00478
DataSet-35	1.057	0.00053	0.00038	0.00120	0.00111	0.00442	0.00483

DataSet-36	2.401	0.00053	0.00037	0.00121	0.00111	0.00449	0.00490
DataSet-37	0.086	0.00053	0.00036	0.00120	0.00111	0.00448	0.00489
DataSet-38	2.896	0.00054	0.00033	0.00123	0.00111	0.00461	0.00491
DataSet-39	2.134	0.00058	0.00035	0.00128	0.00113	0.00470	0.00500
DataSet-40	2.017	0.00064	0.00038	0.00134	0.00111	0.00469	0.00495
DataSet-41	1.698	0.00062	0.00041	0.00130	0.00115	0.00466	0.00502
DataSet-42	2.450	0.00061	0.00040	0.00131	0.00116	0.00468	0.00510
DataSet-43	1.336	0.00063	0.00041	0.00132	0.00118	0.00474	0.00526
DataSet-44	3.023	0.00068	0.00043	0.00140	0.00119	0.00498	0.00534
DataSet-45	3.119	0.00068	0.00043	0.00139	0.00120	0.00512	0.00555
DataSet-46	3.558	0.00071	0.00044	0.00144	0.00123	0.00521	0.00569
DataSet-47	1.218	0.00073	0.00045	0.00147	0.00125	0.00522	0.00574
DataSet-48	2.842	0.00073	0.00042	0.00148	0.00126	0.00537	0.00578
DataSet-49	81.127	0.00067	0.00042	0.00164	0.00128	0.01437	0.00581
DataSet-50	0.065	0.00067	0.00041	0.00163	0.00127	0.01439	0.00579
DataSet-51	2.684	0.00072	0.00042	0.00168	0.00128	0.01441	0.00577
DataSet-52	2.983	0.00072	0.00043	0.00167	0.00128	0.01442	0.00581
DataSet-53	5.343	0.00073	0.00041	0.00167	0.00130	0.01451	0.00575
DataSet-54	72.391	0.00065	0.00045	0.00183	0.00129	0.01421	0.00591
DataSet-55	47.753	0.00065	0.00045	0.00182	0.00133	0.01423	0.00599
DataSet-56	9.081	0.00064	0.00044	0.00179	0.00134	0.01427	0.00627
DataSet-57	11.011	0.00059	0.00052	0.00196	0.00154	0.01422	0.00917
DataSet-58	0.769	0.00060	0.00055	0.00196	0.00158	0.01423	0.00916
DataSet-59	3.587	0.00062	0.00054	0.00198	0.00159	0.01425	0.00920
DataSet-60	0.020	0.00062	0.00054	0.00198	0.00159	0.01426	0.00920
DataSet-61	5.430	0.00066	0.00053	0.00202	0.00158	0.01427	0.00919
DataSet-62	3.680	0.00069	0.00055	0.00202	0.00158	0.01430	0.00916
DataSet-63	0.076	0.00069	0.00054	0.00202	0.00158	0.01430	0.00916
DataSet-64	0.119	0.00068	0.00053	0.00201	0.00158	0.01430	0.00916
DataSet-65	49.168	0.00064	0.00054	0.00212	0.00161	0.01390	0.00923
DataSet-66	0.155	0.00064	0.00053	0.00211	0.00161	0.01391	0.00923
DataSet-67	0.206	0.00064	0.00051	0.00210	0.00160	0.01391	0.00923
DataSet-68	0.125	0.00064	0.00051	0.00209	0.00160	0.01390	0.00923
DataSet-69	65.465	0.00057	0.00055	0.00223	0.00163	0.01398	0.00985
DataSet-70	6.753	0.00050	0.00058	0.00228	0.00166	0.01400	0.00983

Table A.3: Weighted Point Integration Summaries

Data Set	A-post	Mean ΔN (m)	Mean ΔE (m)	RMS ΔN (m)	RMS ΔE (m)	Max ΔN (m)	Max ΔE (m)
DataSet-00	0.522	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
DataSet-01	0.177	0.00000	0.00000	0.00002	0.00002	0.00022	0.00017
DataSet-02	0.161	0.00000	0.00000	0.00003	0.00003	0.00037	0.00035
DataSet-03	0.006	0.00000	0.00000	0.00003	0.00004	0.00038	0.00048
DataSet-04	0.114	0.00000	0.00000	0.00002	0.00003	0.00036	0.00048
DataSet-05	0.083	0.00000	0.00000	0.00002	0.00002	0.00036	0.00038
DataSet-06	0.068	0.00000	0.00000	0.00004	0.00004	0.00069	0.00075
DataSet-07	0.112	0.00000	0.00000	0.00005	0.00006	0.00092	0.00104
DataSet-08	0.034	0.00000	0.00000	0.00005	0.00006	0.00085	0.00120
DataSet-09	0.093	0.00000	0.00000	0.00006	0.00006	0.00103	0.00117
DataSet-10	0.071	0.00000	0.00000	0.00004	0.00005	0.00072	0.00115
DataSet-11	0.089	0.00000	0.00000	0.00002	0.00003	0.00045	0.00087
DataSet-12	0.072	0.00000	0.00000	0.00002	0.00003	0.00046	0.00055
DataSet-13	0.030	0.00002	0.00003	0.00009	0.00013	0.00042	0.00062
DataSet-14	0.164	0.00002	0.00003	0.00007	0.00012	0.00032	0.00055
DataSet-15	0.043	0.00002	0.00003	0.00007	0.00012	0.00025	0.00048
DataSet-16	0.640	0.00006	0.00003	0.00025	0.00016	0.00235	0.00165
DataSet-17	0.109	0.00006	0.00003	0.00023	0.00013	0.00222	0.00155
DataSet-18	0.473	0.00008	0.00003	0.00026	0.00013	0.00233	0.00129
DataSet-19	0.897	0.00011	0.00005	0.00032	0.00017	0.00247	0.00128
DataSet-20	0.515	0.00012	0.00006	0.00035	0.00021	0.00255	0.00174
DataSet-21	0.581	0.00013	0.00006	0.00033	0.00023	0.00218	0.00207
DataSet-22	0.657	0.00014	0.00010	0.00035	0.00032	0.00203	0.00223
DataSet-23	0.754	0.00016	0.00010	0.00038	0.00033	0.00217	0.00228
DataSet-24	0.913	0.00016	0.00006	0.00039	0.00034	0.00229	0.00224
DataSet-25	1.395	0.00019	0.00003	0.00044	0.00040	0.00238	0.00241
DataSet-26	0.691	0.00020	0.00004	0.00047	0.00045	0.00247	0.00255
DataSet-27	1.940	0.00028	0.00014	0.00061	0.00058	0.00291	0.00306
DataSet-28	1.558	0.00029	0.00018	0.00066	0.00062	0.00302	0.00316
DataSet-29	0.610	0.00028	0.00019	0.00065	0.00064	0.00303	0.00324
DataSet-30	0.888	0.00031	0.00019	0.00067	0.00064	0.00303	0.00323
DataSet-31	0.727	0.00036	0.00019	0.00077	0.00067	0.00332	0.00331
DataSet-32	0.955	0.00048	0.00028	0.00096	0.00085	0.00381	0.00376
DataSet-33	0.022	0.00048	0.00029	0.00096	0.00084	0.00378	0.00375
DataSet-34	1.016	0.00050	0.00030	0.00099	0.00088	0.00387	0.00410
DataSet-35	0.614	0.00050	0.00029	0.00101	0.00091	0.00392	0.00497

DataSet-36	1.204	0.00051	0.00028	0.00103	0.00091	0.00399	0.00394
DataSet-37	0.084	0.00051	0.00027	0.00102	0.00090	0.00398	0.00394
DataSet-38	1.424	0.00052	0.00026	0.00104	0.00091	0.00411	0.00392
DataSet-39	1.122	0.00054	0.00026	0.00108	0.00091	0.00420	0.00389
DataSet-40	1.064	0.00057	0.00029	0.00109	0.00090	0.00419	0.00388
DataSet-41	0.996	0.00056	0.00032	0.00105	0.00093	0.00416	0.00391
DataSet-42	1.541	0.00056	0.00031	0.00106	0.00094	0.00418	0.00393
DataSet-43	0.765	0.00057	0.00032	0.00108	0.00096	0.00424	0.00404
DataSet-44	1.264	0.00058	0.00033	0.00111	0.00096	0.00448	0.00407
DataSet-45	1.346	0.00058	0.00032	0.00110	0.00096	0.00448	0.00395
DataSet-46	1.704	0.00062	0.00033	0.00115	0.00099	0.00466	0.00398
DataSet-47	0.580	0.00063	0.00033	0.00117	0.00101	0.00465	0.00495
DataSet-48	1.448	0.00064	0.00032	0.00118	0.00103	0.00469	0.00484
DataSet-49	0.713	0.00063	0.00032	0.00118	0.00104	0.00470	0.00487
DataSet-50	0.061	0.00063	0.00032	0.00117	0.00104	0.00468	0.00485
DataSet-51	1.378	0.00065	0.00034	0.00118	0.00106	0.00473	0.00475
DataSet-52	1.262	0.00066	0.00037	0.00118	0.00111	0.00472	0.00541
DataSet-53	2.363	0.00066	0.00035	0.00117	0.00114	0.00482	0.00539
DataSet-54	1.007	0.00064	0.00039	0.00117	0.00113	0.00671	0.00446
DataSet-55	1.567	0.00065	0.00038	0.00118	0.00116	0.00674	0.00461
DataSet-56	0.874	0.00066	0.00037	0.00117	0.00116	0.00659	0.00525
DataSet-57	1.906	0.00069	0.00043	0.00120	0.00127	0.00667	0.00723
DataSet-58	0.307	0.00068	0.00046	0.00119	0.00130	0.00665	0.00723
DataSet-59	1.300	0.00069	0.00046	0.00121	0.00132	0.00667	0.00726
DataSet-60	0.020	0.00069	0.00045	0.00121	0.00131	0.00666	0.00726
DataSet-61	2.565	0.00070	0.00045	0.00121	0.00131	0.00661	0.00726
DataSet-62	1.838	0.00071	0.00046	0.00121	0.00130	0.00657	0.00724
DataSet-63	0.074	0.00071	0.00045	0.00121	0.00130	0.00657	0.00724
DataSet-64	0.114	0.00071	0.00044	0.00120	0.00130	0.00657	0.00724
DataSet-65	2.140	0.00071	0.00047	0.00120	0.00133	0.00657	0.00732
DataSet-66	0.148	0.00071	0.00046	0.00120	0.00132	0.00657	0.00732
DataSet-67	0.197	0.00071	0.00044	0.00119	0.00132	0.00656	0.00732
DataSet-68	0.122	0.00071	0.00043	0.00119	0.00132	0.00657	0.00732
DataSet-69	1.596	0.00075	0.00043	0.00150	0.00139	0.01494	0.00686
DataSet-70	0.336	0.00076	0.00039	0.00150	0.00142	0.01492	0.00687

Table A.4: Free Network Integration Summaries

Data Set	A-post	Mean ΔN (m)	Mean ΔE (m)	RMS ΔN (m)	RMS ΔE (m)	Max ΔN (m)	Max ΔE (m)
DataSet-00	0.522	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
DataSet-01	0.170	0.00200	0.00187	0.00205	0.00192	0.00283	0.00335
DataSet-02	0.000	0.00140	0.00196	0.00217	0.00210	0.00689	0.00545
DataSet-03	0.000	0.00139	0.00195	0.00217	0.00209	0.00689	0.00545
DataSet-04	0.152	0.00090	0.00123	0.00174	0.00168	0.00689	0.00545
DataSet-05	0.058	0.00089	0.00037	0.00194	0.00224	0.00689	0.00545
DataSet-06	0.025	0.00080	0.00035	0.00194	0.00220	0.00689	0.00545
DataSet-07	0.086	0.00065	0.00032	0.00194	0.00214	0.00689	0.00545
DataSet-08	0.000	0.00066	0.00028	0.00192	0.00213	0.00689	0.00545
DataSet-09	0.054	0.00059	0.00027	0.00191	0.00209	0.00689	0.00545
DataSet-10	0.013	0.00060	0.00031	0.00189	0.00210	0.00689	0.00545
DataSet-11	0.052	0.00064	0.00035	0.00187	0.00206	0.00689	0.00545
DataSet-12	0.045	0.00063	0.00036	0.00186	0.00205	0.00689	0.00545
DataSet-13	0.002	0.00060	0.00057	0.00183	0.00230	0.00689	0.00619
DataSet-14	0.166	0.00057	0.00062	0.00182	0.00233	0.00689	0.00605
DataSet-15	0.038	0.00057	0.00071	0.00180	0.00237	0.00689	0.00598
DataSet-16	0.355	0.00056	0.00072	0.00178	0.00235	0.00689	0.00605
DataSet-17	0.082	0.00057	0.00077	0.00179	0.00240	0.00689	0.00858
DataSet-18	0.093	0.00059	0.00077	0.00180	0.00243	0.00689	0.00859
DataSet-19	0.160	0.00072	0.00064	0.00216	0.00255	0.02092	0.00865
DataSet-20	0.339	0.00066	0.00052	0.00218	0.00281	0.02088	0.01573
DataSet-21	0.148	0.00068	0.00042	0.00219	0.00282	0.02117	0.01560
DataSet-22	0.368	0.00072	0.00043	0.00220	0.00274	0.02106	0.01481
DataSet-23	0.084	0.00074	0.00044	0.00221	0.00273	0.02134	0.01472
DataSet-24	0.087	0.00079	0.00037	0.00230	0.00274	0.02133	0.01471
DataSet-25	0.263	0.00075	0.00045	0.00228	0.00276	0.02133	0.01475
DataSet-26	0.045	0.00074	0.00047	0.00229	0.00276	0.02146	0.01464
DataSet-27	0.356	0.00085	0.00054	0.00234	0.00277	0.02198	0.01418
DataSet-28	0.215	0.00090	0.00060	0.00235	0.00278	0.02217	0.01412
DataSet-29	0.055	0.00088	0.00061	0.00234	0.00277	0.02217	0.01404
DataSet-30	0.480	0.00148	0.00077	0.00488	0.00321	0.08330	0.04341
DataSet-31	0.170	0.00157	0.00078	0.00490	0.00319	0.08299	0.04359
DataSet-32	0.370	0.00169	0.00088	0.00497	0.00323	0.08277	0.04356
DataSet-33	0.006	0.00169	0.00087	0.00496	0.00323	0.08280	0.04353
DataSet-34	0.929	0.00169	0.00090	0.00492	0.00321	0.08265	0.04363
DataSet-35	0.333	0.00169	0.00087	0.00492	0.00319	0.08257	0.04370

DataSet-36	0.101	0.00174	0.00091	0.00495	0.00323	0.08261	0.04375
DataSet-37	0.066	0.00169	0.00093	0.00493	0.00323	0.08262	0.04373
DataSet-38	0.102	0.00158	0.00099	0.00491	0.00332	0.08259	0.04375
DataSet-39	0.439	0.00164	0.00101	0.00495	0.00330	0.08253	0.04382
DataSet-40	0.056	0.00162	0.00095	0.00490	0.00331	0.08257	0.04376
DataSet-41	0.259	0.00158	0.00103	0.00485	0.00335	0.08263	0.04378
DataSet-42	0.535	0.00167	0.00098	0.00501	0.00339	0.08260	0.04380
DataSet-43	0.124	0.00168	0.00098	0.00499	0.00338	0.08257	0.04387
DataSet-44	0.613	0.00167	0.00101	0.00494	0.00336	0.08253	0.04386
DataSet-45	0.330	0.00166	0.00102	0.00490	0.00335	0.08254	0.04392
DataSet-46	0.691	0.00169	0.00104	0.00490	0.00336	0.08244	0.04393
DataSet-47	0.278	0.00169	0.00107	0.00489	0.00337	0.08244	0.04399
DataSet-48	0.027	0.00177	0.00114	0.00498	0.00352	0.08236	0.04403
DataSet-49	0.065	0.00172	0.00114	0.00497	0.00351	0.08234	0.04405
DataSet-50	0.013	0.00168	0.00115	0.00498	0.00350	0.08237	0.04403
DataSet-51	0.216	0.00166	0.00120	0.00493	0.00352	0.08240	0.04399
DataSet-52	0.051	0.00166	0.00122	0.00491	0.00352	0.08243	0.04398
DataSet-53	0.082	0.00165	0.00122	0.00488	0.00351	0.08253	0.04386
DataSet-54	0.258	0.00162	0.00121	0.00484	0.00355	0.08265	0.04383
DataSet-55	0.157	0.00177	0.00115	0.00496	0.00358	0.08261	0.04387
DataSet-56	0.073	0.00151	0.00112	0.00529	0.00366	0.08268	0.04382
DataSet-57	0.042	0.00151	0.00113	0.00529	0.00366	0.08262	0.04380
DataSet-58	0.053	0.00150	0.00117	0.00527	0.00368	0.08265	0.04378
DataSet-59	0.149	0.00149	0.00118	0.00524	0.00369	0.08260	0.04382
DataSet-60	0.020	0.00148	0.00117	0.00523	0.00368	0.08261	0.04381
DataSet-61	0.109	0.00139	0.00131	0.00522	0.00390	0.08270	0.04384
DataSet-62	0.129	0.00135	0.00122	0.00524	0.00462	0.08276	0.04753
DataSet-63	0.028	0.00137	0.00122	0.00525	0.00462	0.08276	0.04754
DataSet-64	0.070	0.00135	0.00126	0.00523	0.00464	0.08276	0.04754
DataSet-65	0.164	0.00139	0.00130	0.00527	0.00463	0.08279	0.04753
DataSet-66	0.082	0.00137	0.00132	0.00526	0.00465	0.08280	0.04754
DataSet-67	0.119	0.00152	0.00130	0.00539	0.00466	0.08280	0.04752
DataSet-68	0.081	0.00154	0.00130	0.00540	0.00466	0.08279	0.04752
DataSet-69	0.064	0.00158	0.00130	0.00537	0.00463	0.08282	0.04742
DataSet-70	0.221	0.00153	0.00135	0.00535	0.00465	0.08284	0.04740

Table A.5: Pseudo Observation Integration Summaries

Data Set	A-post	Mean ΔN (m)	Mean ΔE (m)	RMS ΔN (m)	RMS ΔE (m)	Max ΔN (m)	Max ΔE (m)
DataSet-00	0.522	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
DataSet-01	0.262	0.00000	0.00000	0.00002	0.00002	0.00033	0.00017
DataSet-02	0.327	0.00000	0.00000	0.00003	0.00003	0.00044	0.00029
DataSet-03	0.277	0.00000	0.00000	0.00003	0.00003	0.00045	0.00030
DataSet-04	0.264	0.00000	0.00000	0.00003	0.00002	0.00045	0.00028
DataSet-05	0.210	0.00000	0.00000	0.00002	0.00002	0.00040	0.00027
DataSet-06	0.183	0.00000	0.00000	0.00004	0.00004	0.00069	0.00055
DataSet-07	0.211	0.00000	0.00000	0.00005	0.00005	0.00092	0.00079
DataSet-08	0.290	0.00000	0.00000	0.00005	0.00005	0.00085	0.00069
DataSet-09	0.219	0.00000	0.00000	0.00006	0.00006	0.00103	0.00092
DataSet-10	0.184	0.00000	0.00000	0.00004	0.00004	0.00072	0.00057
DataSet-11	0.198	0.00000	0.00000	0.00002	0.00002	0.00045	0.00034
DataSet-12	0.238	0.00000	0.00000	0.00002	0.00002	0.00046	0.00034
DataSet-13	0.223	0.00000	0.00000	0.00002	0.00002	0.00035	0.00025
DataSet-14	0.266	-0.00001	-0.00001	0.00004	0.00004	0.00027	0.00018
DataSet-15	0.229	-0.00002	-0.00002	0.00007	0.00007	0.00025	0.00025
DataSet-16	1.334	0.00003	-0.00004	0.00012	0.00016	0.00136	0.00188
DataSet-17	0.263	0.00003	-0.00005	0.00009	0.00018	0.00127	0.00190
DataSet-18	1.131	0.00005	-0.00006	0.00014	0.00021	0.00141	0.00235
DataSet-19	1.407	0.00007	-0.00006	0.00022	0.00021	0.00167	0.00261
DataSet-20	0.671	0.00008	-0.00006	0.00024	0.00026	0.00180	0.00243
DataSet-21	0.738	0.00009	-0.00007	0.00027	0.00027	0.00151	0.00260
DataSet-22	0.984	0.00011	-0.00003	0.00034	0.00028	0.00186	0.00252
DataSet-23	0.819	0.00011	-0.00003	0.00039	0.00031	0.00210	0.00261
DataSet-24	1.808	0.00010	-0.00006	0.00043	0.00041	0.00233	0.00268
DataSet-25	2.256	0.00015	-0.00008	0.00053	0.00052	0.00247	0.00270
DataSet-26	0.822	0.00015	-0.00007	0.00056	0.00056	0.00259	0.00281
DataSet-27	3.203	0.00021	0.00006	0.00069	0.00066	0.00303	0.00333
DataSet-28	2.052	0.00021	0.00010	0.00072	0.00068	0.00314	0.00343
DataSet-29	0.791	0.00019	0.00011	0.00073	0.00071	0.00315	0.00348
DataSet-30	1.657	0.00020	0.00013	0.00073	0.00073	0.00315	0.00349
DataSet-31	1.288	0.00025	0.00012	0.00082	0.00080	0.00344	0.00354
DataSet-32	4.041	0.00037	0.00021	0.00098	0.00097	0.00406	0.00427
DataSet-33	0.217	0.00037	0.00022	0.00097	0.00096	0.00402	0.00423
DataSet-34	1.235	0.00039	0.00023	0.00101	0.00098	0.00408	0.00435
DataSet-35	0.890	0.00040	0.00021	0.00105	0.00102	0.00411	0.00440

DataSet-36	2.186	0.00040	0.00019	0.00107	0.00103	0.00421	0.00447
DataSet-37	0.250	0.00039	0.00017	0.00105	0.00103	0.00419	0.00446
DataSet-38	2.658	0.00040	0.00014	0.00109	0.00104	0.00452	0.00448
DataSet-39	1.666	0.00043	0.00016	0.00113	0.00105	0.00478	0.00457
DataSet-40	1.573	0.00047	0.00018	0.00115	0.00103	0.00481	0.00452
DataSet-41	1.481	0.00045	0.00021	0.00112	0.00106	0.00489	0.00459
DataSet-42	2.116	0.00045	0.00020	0.00113	0.00108	0.00503	0.00467
DataSet-43	1.138	0.00047	0.00021	0.00114	0.00110	0.00505	0.00483
DataSet-44	2.583	0.00052	0.00023	0.00125	0.00111	0.00528	0.00491
DataSet-45	2.411	0.00052	0.00023	0.00124	0.00114	0.00549	0.00512
DataSet-46	2.716	0.00056	0.00025	0.00129	0.00118	0.00558	0.00526
DataSet-47	0.976	0.00058	0.00025	0.00131	0.00119	0.00559	0.00531
DataSet-48	2.249	0.00058	0.00023	0.00133	0.00121	0.00574	0.00535
DataSet-49	0.695	0.00059	0.00023	0.00136	0.00124	0.00614	0.00538
DataSet-50	0.240	0.00058	0.00022	0.00135	0.00124	0.00615	0.00536
DataSet-51	2.225	0.00064	0.00023	0.00142	0.00125	0.00615	0.00534
DataSet-52	2.137	0.00064	0.00024	0.00140	0.00125	0.00615	0.00538
DataSet-53	4.112	0.00065	0.00023	0.00143	0.00130	0.00618	0.00532
DataSet-54	1.781	0.00063	0.00027	0.00137	0.00127	0.00560	0.00548
DataSet-55	1.101	0.00064	0.00026	0.00140	0.00130	0.00564	0.00556
DataSet-56	1.142	0.00064	0.00024	0.00138	0.00131	0.00553	0.00559
DataSet-57	2.562	0.00069	0.00029	0.00142	0.00136	0.00543	0.00568
DataSet-58	0.207	0.00069	0.00031	0.00140	0.00137	0.00541	0.00565
DataSet-59	2.149	0.00073	0.00030	0.00150	0.00140	0.00549	0.00572
DataSet-60	0.216	0.00073	0.00030	0.00149	0.00139	0.00547	0.00571
DataSet-61	4.822	0.00076	0.00029	0.00154	0.00140	0.00545	0.00578
DataSet-62	2.893	0.00078	0.00030	0.00154	0.00139	0.00543	0.00579
DataSet-63	0.244	0.00078	0.00029	0.00154	0.00140	0.00544	0.00580
DataSet-64	0.267	0.00077	0.00028	0.00154	0.00140	0.00543	0.00579
DataSet-65	3.251	0.00077	0.00030	0.00154	0.00143	0.00545	0.00605
DataSet-66	0.285	0.00077	0.00029	0.00153	0.00143	0.00544	0.00604
DataSet-67	0.307	0.00076	0.00027	0.00152	0.00142	0.00544	0.00604
DataSet-68	0.268	0.00076	0.00026	0.00152	0.00142	0.00545	0.00605
DataSet-69	4.432	0.00079	0.00032	0.00154	0.00144	0.00535	0.00619
DataSet-70	0.370	0.00080	0.00035	0.00153	0.00148	0.00531	0.00616

Curriculum Vitae

Full Name: Michael Terence Bremner

Birth: Sechelt, British Columbia, October 25th 1983

Address: 216 Saint John St, Fredericton, NB

Universities attended:

British Columbia Institute of Technology, 2010 - 2012

Diploma of Geomatics Engineering Technology

University of New Brunswick, 2012 - 2015

Bachelor of Science in Engineering

University of New Brunswick, 2016 - 2019

Master of Science in Engineering (In Progress)

Publications:

Bremner, M., & Santos, M. (2019). A Local Projection for Integrating Geodetic and Terrestrial Coordinate Systems. *Survey Review*.

Bremner, M., Keirstead, G., Muggah, J., Wallace, B., Chateauvert, A., Yamazaki, G. (2016) Determining the Bathymetry of the Mactaquac Headpond and Saint John River. Mactaquac Aquatic Ecosystem Study Report Series 2016-032.