DESIGN AND IMPLEMENTATION OF AUTOMATED MAPPING SYSTEM

- WITH EMPHASIS ON IMAGE HANDLING -

BY

JAE-HONG YOM

B.Sc. in Civil Engineering (Yonsei University, Seoul Korea) 1981
M. Eng. in Photogrammetry (Yonsei University, Seoul Korea) 1983
Post Graduate Diploma in Land Information System (ITC, The Netherlands) 1989

A Thesis for the Degree of Doctor of Philosophy (Ph.D.)

of the Faculty of Science at the University of Glasgow

Department of Geography & Topographic Science

January 2000

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

ABSTRACT

This thesis is concerned with the design and the implementation of an automated mapping system. Automation of the mapping process is a major concern in the geomatics discipline as the role of geospatial information is becoming more important in the 'information society'. Automation has been successful in some tasks of the mapping process, whereas it is faced with difficulties in other more complex tasks.

In all of these automation efforts, software development is inevitably involved. In software development, software design should be carried out prior to its implementation but at present, software design is not widely practiced by research organisations in the geomatics discipline. Research into systematic design of software for the geomatics discipline will prove beneficial for both the research organisations and the end users in the long run.

An Object Oriented software design using the Unified Modeling Language (UML) is presented in this thesis for some of the subsystems of the automated mapping system, namely:

- the image acquisition subsystem;
- the positioning subsystem; and
- the image point referencing subsystem.

For each of these subsystems, the domain knowledge and the software implementation aspects were investigated and analysed. Based on the results of this analysis, the software design for the subsystems was produced using UML. The design was then implemented in C++ programming language and tested with practical data.

This study concludes that the software design of this study enhances the implementation effort. For example, the same classes which had already been implemented in the positioning subsystem were reused in the image point referencing subsystem with only little changes.

It is also emphasized in this study that the Object Oriented design using UML should be used by research organisations of the geomatics discipline. The software design of the automated mapping system presented in this thesis will lay a foundation for further development of software which could be effectively used for geospatial research.

PREFACE

ACKNOWLEDGEMENT

I would like to offer my most sincere thanks to my supervisor Dr. J. E. Drummond for her constant support and encouragement throughout my study. Her advice and guidance were essential for the completion of this thesis.

I am also grateful to Mr. D. A. Tait, who has given me valuable guidance as second supervisor. Thanks and gratitude are also due to Mr. I. Gordon for his lectures on GPS and assistance with GPS related subjects, Dr. D. Forrest, Professor G. Petrie and Dr. R. Poet for their interest and assistance during the study.

I am also grateful to Dr. K. P. Schwarz of the Department of Geomatics Engineering of the Calgary University for his kindness and guidance offered during my study period there. My appreciation extends to Dr. N. El-Sheimy and all the members of the research group.

I would also like to thank Dr. G. Mader for making his KARS GPS program available.

My colleagues, Alias Abdul-Rahman, Matt Duckham and Hasin Rammali are also gratefully acknowledged for useful discussions and also for sharing their experiences and most of all, going through the journey together.

Thanks also goes to Professor J. Briggs and Mr. Iain McNeil for their assistance with the Glasgow University scholarship application.

Financial support for this study has been granted by the University of Glasgow, the British Council and the Air Korea Co. (now part of Korea Airport Service Co.). Their financial support is greatly appreciated.

The coordinated help of the International Office of Glasgow University and the British Council Offices at Glasgow and at Seoul is also gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Objective of Research

It is natural for expectations of the level of automation in mapping systems to grow as technology evolves and new possibilities emerge. Furthermore, as with several other disciplines, the mapping or geomatic(s) discipline (namely cartography, geodesy, photogrammetry, remote sensing and surveying) is obliged to cooperate with others. Examples of other disciplines with which geomatics cooperates include various sub-disciplines of computer science such as computer vision, digital image processing and artificial intelligence.

These expectations and obligations require rapid implementation of new technologies and concepts into working mapping systems, if end users are to benefit as soon as possible. However rapid implementation depends almost entirely upon such systems' **software design**. For a system to be augmented with the latest technologies, it is crucial for any system to have a software design which is flexible in terms of extendibility, compatibility and reusability and also to be reliable in terms of meeting the requirements of the intended users.

1.1.1 The Mapping Process

Before the introduction of computers, the production of a map was a very labour intensive task. One of the major difficulties involved in this process was the wide variety of activities involved. The production process was a sequence of tasks including surveying, aerial photography and photo processing, stereo compilation, cartographic design and editing.

It is important to understand the mapping process in the traditional environment and its inherent problems to be able to understand why automation has become an important issue in the mapping discipline.

For example, in the traditional environment:

- Surveying ground control points involves much physical activity. Surveyors may climb mountains daily for several months or travel in a desert for hundreds of miles.
- Good planning and experience are also very important. An experienced surveyor will develop an intuition for the landscape and the optimum place for a control point and the route to get there from just looking at the map.
- The surveyor needs mathematical skills. To compute the coordinates of the selected

points that have been visited, rigorous adjustment of the observations of angles and distances is required. The surveyor must be able to identify a badly observed point, and judge the influence of this point on the overall accuracy.

- The knowledge required for an aerial photographer is quite different from a surveyor and includes photographic science, aircraft and their navigation.
- In contrast to the outdoor activities of surveying and aerial photography, the stereoplotter operator works indoors using very complex instruments. These operators are highly skilled and require a lot of training. For example the operator's judgment influences the overall accuracy of what shows up in the completed map. Or given a general guideline that all buildings should be recorded, the operator has to decide for example if a wall around a house, a small hut, or a telephone box should be collected. Usually this detailed judgment comes with experience of many different mapping projects.
- Turning to the cartographer, who also works indoors (although sometimes verification of the raw map produced by stereo compilation involves the cartographer going outdoors to the mapped area), it is essential for the cartographer to possess a skill for graphic presentation and expression. It is the responsibility of the cartographer to refine the coarse information content in the raw map provided by the stereoplotter operator to one that is standardized and coherent. This involves symbolisation, annotation and the more complex task of generalisation (a process of skilled decision making regarding which information to retain in conflicting or potentially confusing situations).

1.1.2 An Automated Mapping System

With the many problems that were inherent in traditional map production, it is obvious that automation could benefit the process to a great degree. Some of the automation technologies which have been applied to the production of maps are reviewed in later paragraphs.

In an ideal automated mapping system, it is possible to imagine that an unmanned robot aeroplane equipped with necessary sensors will be controlled from a base station and the acquired images with position data will be transmitted back to the base station in real time. Subsequently spatial objects will be extracted automatically from the received images, and these will populate the GIS database.

An **automated mapping system** is, for the purpose of this study, defined as a composition of subsystems which apply the concepts and the techniques described below. The system is composed of the image acquisition subsystem, positioning subsystem, image point

referencing subsystem, object extraction subsystem and the visualisation subsystem.

For this research, a practical level of automation is assumed where human interaction is still required at many stages.

Image acquisition

The image acquisition subsystem of the automated mapping system, involves a human operated platform, such as a fixed wing airplane or a helicopter. The imaging sensors are a key component of the imaging subsystem. Some examples of imaging sensors are CCD digital cameras, video cameras, linear array scanners, laser profiling scanners and multi-spectral scanners.

Photographic processing is a task that will become unnecessary when digital images are acquired directly. It is expected that CCD cameras will be widely used for capturing digital images directly [Ohlhof et al.,1994][Peipe, 1994]. At present though, because of currently available technology, it is still a practice in the mapping community to use film based cameras and to scan the film digitally.

Positioning

In an automated system these imaging sensors (eg CCD cameras) are synchronized with the positioning sensors to compute the attitude and the position at the instant of the image capture. Some of the most popular positioning sensors in use today are GPS (Global Positioning System) receivers and IMU (Inertial Measurement Units). The GPS receivers provide the positional coordinates of the exposure stations whereas the IMU provides the attitudes of the images at the instant of exposure. The observations from these sensors are synchronised and integrated through the Kalman filtering process. This process will compute the integrated result in an optimal sense.

Another role of the positioning sensor in an automated mapping system is the navigation of the platform vehicle. The real time capability of a differential GPS will make control and monitoring of the flight simple and this will result in the improved coordination of activities between the pilot and the photographer.

A further manual task that is removed with the introduction of the positioning sensors is the need for ground control point surveying. This is very beneficial in terms of time and cost because, in a traditional mapping project, the cost of ground control point surveying can take up to more than 50% of the whole project cost. At present though the positioning sensors of

an automated mapping system have not completely removed the need for ground control. The problem now is more of a practicality and cost effectiveness rather than technical feasibility. For the automated system to be able to achieve the same accuracy and reliability as a traditional mapping system, the automated mapping system would have to use a CCD camera of a large sensor size, e.g. close to the 23 cm x 23 cm of the traditional aerial camera, and a high quality IMU. This is impractical in terms of the cost involved, although much progress has been seen in recent years in the enhancement of low or mid quality sensors through research into algorithmic development and the integration of different types of sensors [Ackermann, 1995][Schwarz, 1998][Axelsson, 1999].

Image Point Referencing

In Geomatics the term triangulation appears in many contexts. In land surveying it is the term used for control extension and exploits the geometric properties of two-dimensional triangles [Wolf et al., 1997 (pp. 249-274)]. In photogrammetry the term aerial triangulation is also a means for control extension, but exploits the projective geometry of the camera [Kraus et al., 1993 (pp. 247-295)]. In some digital photogrammetric systems the term is loosely used to describe relative and absolute orientation as well as aerial triangulation [GDE-Systems, 1997].

However in this work, the term 'image point referencing' has been selected to refer to the task of locating the ground position of any points seen on the image by using their observed image coordinates and the values obtained from the positional sensors.

In traditional mapping systems, the main reason for the aerial triangulation process is to reduce the cost of ground control point surveying by replacing it with computed coordinates from image observations. During the aerial triangulation process, the operator identifies the surveyed ground control point in the images in which they appear and measures the image coordinates of these points. These measurements and the surveyed coordinates are then fed into an aerial triangulation program to compute the coordinates of the perspective points as well as the attitudes of the observed and other connected images at their instants of exposure. These (i.e. the image coordinates, the perspective centre coordinates and image attitudes) are then used to compute ground coordinates of any points on an image.

In the image point referencing process of an automated mapping system, the perspective point coordinates and the attitudes are not computed indirectly, but are observed directly using the positional sensors. In theory, the image point referencing process would not involve any ground control point surveying.

Aside from the fact that the perspective coordinates and the attitudes are directly observed, the setting up of the observation equations and their adjustment is very similar to the traditional aerial triangulation task.

Image Matching

Digital image matching is described as a process which automatically establishes the correspondence between primitives extracted from two or more digital images depicting at least partly the same scene [Heipke, 1996].

Image matching techniques are used to automate the process of determining the coordinates of an image point. In the interior orientation process, they are used to determine the pixel coordinates of the fiducial points. In the aerial triangulation process image matching is used to measure the pixel coordinates of targeted ground control points. Image matching can also be used to compute the relative orientation of stereo pairs. In this process interest operators (a mask of functions, or kernel, used to process the radiometric values of pixels in a region of an image) are used to locate points of interest, such as road corner points, in one image. Then a matching technique is applied to locate the corresponding conjugate points from other photographs which include the same scene.

Image matching is especially appropriate for the generation of a digital terrain model from stereo aerial images. The manual task of creating a digital terrain model is very simplistic in nature but tedious and error prone, making the effort of automation relatively easy but producing much benefit as its result.

Spatial object extraction

The automation of stereo compilation is still a challenge to many researchers in both the photogrammetric field and the computer vision field. Object recognition techniques are used by the photogrammetrist to capture the semantic information at a certain location and populate the GIS database, which is identical to the task of the human stereoplotter operator. The automation effort in this field involves research into image segmentation, feature extraction from images and grouping extracted features such as points and edges. Detection and interpretation of simple features such as road centrelines has been successful, but other spatial objects, buildings in particular, are still being researched [Roux et al., 1994][Gruen et al., 1996][Boichis et al., 1998][Haala et al., 1998].

Visualisation

Schroeder et al. [1998 (pp. 1-15)] define visualisation as "... the process of exploring,

transforming, and viewing data as images (or other sensory forms) to gain understanding and insight into the data .". The concept of automation and the involvement of computer processing are inherent in this definition of visualisation. The activity of visualisation resembles closely the activity of a cartographer, except that a cartographer deals mainly with geospatial data.

The end product of a visualisation process in a mapping system might be a three dimensional perspective view of a landscape processed from the digital terrain model and a scanned aerial image of that area. The images of the mountains could also be marked with contour lines, streets could be labelled with street names and commercial buildings could be coloured red. In other words, the traditional cartographic process of symbolising information on paper maps is incorporated into the visualisation process as symbolising information on three dimensional image views.

1.1.3 The Need for Software Design in an Automated Mapping System

Integrated Approach to Cope with Increased Complexity

The issue of software design in automated mapping is recent because it is only with the latest developments in technology that considerations of software design have emerged. For example, if a photogrammetrist is interested only in the study of error propagation from traditional aerial triangulation using ground control and standard film based photographs, which is a self-contained environment, it would probably be more efficient if the software was written in a high level procedural language such as FORTRAN or C.

In this sort of study the design of the software need not be given much consideration. But if other positioning systems and sensors are to be integrated with the photogrammetric tasks, such as GPS receivers, IMU and laser scanners, for example, to carry out automatic orientation of images, or to automatically detect and extract building features in the images and place them in a GIS database, the problem becomes much more complex.

This becomes an integrated problem with different types of knowledge domains and considerable coordination effort required. The complexity involved is not only of a mathematical nature. The programming task is complex too. It needs, for example, a software design approach to come up with an efficient solution.

Efficiency and Preservation of Information

Although different types of knowledge base are necessary for the various tasks of a mapping system, there are some common key elements which carry on to the next task. For example a point, such as a footpath intersection, surveyed with GPS is imaged on different images and eventually ends up in the GIS database. The information collected during a GPS surveying session must be preserved, and processed with the imaged points in different images and later the point is identified as a footpath intersection at the object recognition stage. This set of information should be preserved in the finalized version of the GIS database. This would not only make it an efficient way of processing but also an important function enabling the tracing of error sources in a GIS database. In an unintegrated approach, only the result of the processed coordinates would be in the final GIS database, with all the intermediate information being lost.

1.1.4 Software Design Research in Geomatics Community

In the Geomatics community, depending on the type of problem faced, some fields have been quick to adopt the software design approach to solve intrinsic problems whereas others have not. The fields of GIS and cartography started their research in applying software design at an early stage, in contrast to photogrammetry and surveying .

In the field of GIS, the modeling of spatial objects is a very widely studied subject and this now can lay a foundation for a software system handling the geospatial data, produced by the surveying and photogrammetric processes. The modeling and software design aspects have been of much interest in GIS. For example, Pilouk [Pilouk, 1997] did an extensive study on integrated modeling for 3D GIS and attempted the integration of spatial objects sharing common aspects in one 3D model, using the object oriented approach for its logical design and implementation. Molenaar [Molenaar, 1998] introduced a theoretical framework covering different aspects of spatial object modeling. Commercial GIS systems, such as LaserScan LAMPS2 or SmallWorld also incorporated such approaches.

Likewise, the field of cartography was involved comparatively early in cooperation with a sub-discipline of computer science, i.e artificial intelligence. As an example: cartographic expert systems have been one of the major cartographic research fields which have required the application of good software design [Keller, 1995]. Cartographic expert systems were mainly applied to generalization. Generalization, an important task in cartography, is defined by Peng [Peng, 1996] as a transformation process with the objectives of transforming a

database to a lower resolution and as a way of presenting a legible graphic view of a database. Peng used Object Oriented design to resolve conflicts in generalization.

However in surveying and photogrammetry, despite much interest in the automation of various tasks, software design was not as much an issue as the algorithmic aspects of a process.

In surveying for example, the focus of research for the past decade has been the increase of accuracy and reliability of GPS observation [Remondi, 1984][Lachapelle, 1990][Goad et al., 1997].

In photogrammetry, the focus of research has been the automation of a simple manual task, such as selecting an interest point from an image [Forstner et al.,1987], for example a corner of a building or path intersection and locating the conjugate point from a stereo pair [Schenk et al., 1991][Gulch, 1995][Ackermann et al., 1997]. All this was carried out, of course, with the highest possible geometric accuracy as a major consideration.

The reason behind the fact that the software design aspect did not play an important role in surveying and photogrammetry may be because, although the problems in surveying and photogrammetry are complex in their algorithmic aspects, the types of data and the software implementation aspects are relatively simple. There are not many data types involved and there is not much interaction between the data types. Most tasks involve the sequential processing of a relatively few types of data.

Recently though, the effort to automate stereo compilation, which involves the interpretation of images of ground objects, has brought about coordination of efforts in photogrammetry, computer vision and artificial intelligence [McKeown, 1989][Schenk et al., 1992]. What is a simple task for a stereoplotter operator, such as following the line of a building or a road on an image and labelling these as a building, road, etc. amounts to a very complex task for a computer to emulate, in terms of positional and semantic accuracy and efficiency.

1.1.5 Problem in Software Design for Automated Mapping System

A major problem in the software design of any system is the modeling of the domain knowledge [Booch, 1994 (pp. 3-26)]. This is usually done at the requirement analysis phase of the system's life cycle. In many simple situations, such as automating an accounting system for a small scale business, the activities and the domain knowledge involved in accounting are transferred to the software engineer through interviews and investigations. If the domain is complex and wide in variety, the task is more difficult.

The design of an automated mapping system is difficult in that it involves many tasks, as well as highly theoretical knowledge. In such a case, either the software engineer will have to study all the tasks involved in the map production process or a mapping expert will have to acquire the knowledge and skills of a software engineer. In either case, it would be a difficult task.

This study has been undertaken to produce a software design for an automated mapping system from a Geomatics point of view. It is the objective of this research that the software design will lay a foundation for the development of automated mapping software. It is expected that this effort will benefit both the Geomatics researchers who wish to program and implement their ideas and the software engineer who wishes to acquire Geomatics domain knowledge. It is also expected that through refinement of the design and the addition of more classes and functions to the design, the common starting point of the development of an automated mapping system will be the software design that has emerged from this study, thereby enabling maximum software reuse and compatibility which will in turn provide quick implementation of new technologies for mapping purposes.

1.2 Scope of Research

This research focuses on the software design and implementation of an automated mapping system. Object Orientation methodology is applied in the design process and so the conceptual building of a set of common classes for the automated mapping system is of major interest. The domain analysis phase of the object orientation, i.e. details of the processes and algorithms used in the mapping field, is extensively covered from a software development perspective and then its findings are developed into a software design. Finally the design is implemented and tested (in part).

A comparison of the mapping process in the manual environment and what the author defines as the automated mapping environment is made in Table 1.1.

The various subsystems of the automated mapping system mentioned in subsection 1.1.2 will perform the mapping processes mentioned on the first row of the table.

The Positioning Subsystem will reduce human interaction significantly in the Ground Control Surveying process. A surveyor would still be needed to install the GPS receivers and initiate the observations.

Table 1.1 Comparison of Manual and Automated Mapping Environment

Mapping Process	Manual environment	Automated environment
Ground Control Surveying	Coordinates of points which appear in the images are surveyed using surveying instruments. The distance, angles between these points are observed to compute the position of each point in a selected coordinate system.	Ground Control Surveying would not be necessary in most cases. The position of image exposure stations and the attitudes of images would be computed from the integrated sensor system. However some ground points might still be necessary to increase reliability for quality control.
Image Acquisition	An area of interest which will be mapped, is selected and after making the appropriate plans, images are captured. For aerial photography, flying height, speed and the approximate coordinates of waypoints are important factors during the planning stage, and constant monitoring of the airplane's position during flight is essential. The most important factor however is the weather, especially clouds, which will decide whether a flight should be made at all.	Imagery is acquired directly using a digital sensor, such as a CCD camera. These sensors are synchronized with positioning sensors such as the GPS receivers and/or Inertial Measurement Unit and/or Laser Scanner. The position, attitude of images captured by these sensors can be computed using the synchronising features.
Image Point Referencing	Using the ground control surveyed points, ground coordinates of tie points are computed from image observation. The geometry of the camera constants, and the exposure positions and the attitudes of a stereo pair are used in the computation.	The exterior orientation of images which are available from the preceding process, ground coordinates of image objects or any points on the image are computed. This process would be very similar to the manual aerial triangulation process.
Stereo Compilation	Through the aerial triangulation process all the geometry at the instant of exposure is accurately computed. This is reconstructed on a stereo plotter. The operator, usually a skilled person, is now able to draw any feature on the image onto a map at the given coordinate system, producing a map. This process involves the interpretation of the objects on the image and recording the interpretation at its spatial location accurately	The features in the images are used to emulate the interpretation process of a human brain. The points, lines or area features are extracted through interest operators (or edge detection algorithms, segmentation methods, texture manipulation, grouping, labeling etc.) The interpreted object would then be recorded in an intermediate format before going into a target geospatial database.
Cartographic Editing	Whereas stereo compilation is concerned about recording, cartographic editing is concerned with presentation and the communication of information to the end user. The cartographer uses the 'raw map' of the stereo compilation and symbolises this in a standardised and coherent way.	The data from the geospatial database will be used to relay to and analyse information for the end user. The manipulation or modeling and the visualisation of the data content of the geospatial database is done through this module. This module would however be physically separated from the Mapping System and reside in a user's GIS

The Positioning Subsystem of the automated mapping system will be used either in real-time or post processing of the observed data. The Image Acquisition process will be performed through the collaboration of both the Image Acquisition Subsystem and the Positioning Subsystem.

The Image Point Referencing process will be performed by the Image Point Referencing Subsystem which will use image matching techniques to replace the human operator's action of observing and recording the coordinates of a point on the image. This image matching technique would be able to automate the image point referencing process completely, in the case of aerial triangulation.

The Stereo Compilation process should be automated by the Spatial Extraction Subsystem. But as mentioned earlier, automation for this process will involve a certain level of human interaction for some time during the near future.

Cartographic Editing is a process where much change has taken place recently and changes are further anticipated with regards to its role in the creation of GeoSpatial information. This is because automation has brought about drastic changes in the presentation and the dissemination of information. These changes can be understood easily if one thinks of the change from a traditional map on paper to an image of a three dimensional perspective view available on the Internet. In the automated mapping system, the process of presenting geospatial information will be performed by the Visualisation Subsystem.

Automation efforts will continue to recreate the results produced by a cartographer. At the same time there will also be other efforts to produce computer oriented maps which can be produced at a less cost and with less time, even though lacking in the overall quality. It can be expected that the end users will prefer to use these computer oriented maps because they can be easily adjusted to suit the user's specific purpose with less time and cost, even though they are not as comprehensive and as complete as traditional maps. The versatility of the computer oriented maps implies that it should be possible for the user to decide what information should be included in the produced maps. This should be taken into account in the design of the Visualisation Subsystem. The Visualisation Subsystem should provide the basic tools (i.e classes) which can be utilised by a specific application field of GIS, such as cadastral administration, utilities management or environmental monitoring.

This research will analyze in detail the Image Acquisition, Positioning and Image Point Referencing process of an automated mapping system, define the requirements for those processes and present an object oriented design for them. The design will be implemented and tested for a camera calibration application and for GPS data processing.

1.3 Outline of Thesis

Following this initial introductory chapter, this thesis is divided into six further chapters.

Chapter 2 is an introduction to the concepts of the Object Oriented method and the Unified Modeling Language (UML). Key concepts of abstraction, encapsulation, inheritance and polymorphism will be explained with examples from photogrammetry. Object Oriented design will then be explained followed by an introduction to the Unified Modeling Language.

In Chapter 3, the analysis and design of the Image Acquistion Subsystem will be described. Calibration of the imaging sensor, controlling the image capturing process and flight planning will be explained using UML diagrams. Important classes will be identified and Class Diagrams of these classes will be shown.

Chapter 4 addresses the analysis and design of the Positioning Subsystem. In the first section of this chapter, various classes used in GPS data processing will be designed. The second section deals with IMU data processing. The chapter concludes with a section on Kalman filtering.

Chapter 5 addresses the design of the Image Point Referencing Subsystem. Classes which will be responsible for the formation of the collinearity condition will be designed followed by a class to solve the normal equations. Computational optimisation schemes in terms of memory storage and speed are also introduced. A class for matrix formation and computation is explained in the final section of this chapter.

Chapter 6 addresses the implementation of the design for camera calibration application and GPS surveying. For the bundle adjustment implementation, a camera calibration test is carried out using a control grid plate and also the implemented program was tested on a set of terrestrial images taken with a digital camera. For GPS surveying the implemented program was tested for relative positioning using phase data.

Chapter 7 concludes with a summary of the findings and some recommendations.

2. AN INTRODUCTION TO OBJECT ORIENTED SOFTWARE DESIGN

The Object Oriented software design approach is now a proven and accepted methodology in the software engineering discipline. Its concepts and benefits can be found widely [Goldberg et al., 1983][Cox, 1986][Rumbaugh et al., 1991].

But to summarise, the key concepts of Object Orientation are abstraction, inheritance, encapsulation, and polymorphism [Stroustrup, 1995][Microsoft, 1997]. The benefit of using these concepts is the increase of efficiency in software design and implementation achieved by enhancing reusability, compatibility and extendibility [Meyer, 1988].

The abstraction concept in Object Orientation, for example enables the close resemblance of the model to real physical phenomena. For example in Object Orientation, a camera is named and modeled as a camera and all the characteristics of a camera such as the principal point, the film size and shutter speed are included in the camera class as its components. "Class" is a key Object Oriented concept, see section 2.1. In contrast to the Object Oriented method, in a procedural method, the camera is modeled as a set of functions or subroutines from an early stage of the development, such as CheckCameraStatus(), GetShutterSpeed() or GetPrincipalDistance(). The closer resemblance to reality of an Object Oriented model, makes the transition from reality to computer programs an easier task compared to the procedural method. When one uses a procedural method to deal with complex software development, one can easily get confused and make errors using various data types and arguments.

In a traditional structured programming design, one would begin by trying to identify what the system is going to do, i.e. what are the tasks and the processes that the program is going to carry out. Flowcharts will be laid out then refined and broken down into many hierarchies to identify each detailed step which will then be implemented as functions. However, in an Object Oriented design the first step is the identification of the candidate classes. The objects and the active entities in the problem domain are first identified. The attributes and the behaviour of the identified class are assigned and then their relationships and interactions with other classes are defined.

The Object Oriented method leaves the details of each function or subroutine to the final stage. In the initial stage its chief aim is to identify the main objects in the system, the interaction between these objects and the attributes of the objects. It proceeds gradually from

a high level of abstraction to the lower level of abstraction. This enables the development of complex software.

In this chapter, the key concepts of the Object Oriented method will be introduced, with examples as to how they are applied in the photogrammetric process. Then a software design tool, the Unified Modeling Language (UML) will be introduced.

Examples will be given in C++ codes. For readers unfamiliar with the C++ programming language, review texts are recommended such as "The C++ Programming Language" [Stroustrup, 1995] for those who have some experience with C language, and "First Course in C++ : A Gentle Introduction" [Harman et al., 1997] for those who are not familiar with C or C++.

2.1 Key Concepts in the Object Oriented Method

Meyer [Meyer, 1988] defines Object Oriented design to be the construction of software systems as **structured collections** of abstract data type implementations. The abstract data type implementations refers to **classes**. The word **collection** emphasizes class as being an independent unit. This means a class should not be specific to a system but should have its own behaviour and characteristics, leading to reusability of the class in other systems. The word **structured** emphasizes the relationships that must exist between different classes.

The instantiation of a class is called an **object**.

A simple example of a class is **CCamera** (in C++ programming notation, classes are frequently named with a 'C' in the beginning of the class name to show that it is a class). An object of this class could be called 'MyCamera' or even 'RC30' or 'UMK10', to refer to the actual camera that was used in a photogrammetric project.

The design activity begins with identifying the classes involved, their behaviour and characteristics and the relationship or the interaction between them.

In the actual execution of the program, objects of each class will interact with each other to accomplish the purpose of the system.

Classes and objects employed in software development lead to the application of some important concepts within the Object Oriented method, namely: abstraction, inheritance, polymorphism and encapsulation.

2.1.1 Abstraction

Abstraction is one of the most important facilities of the Object Oriented method. It is a facility which enables us to model the way our mind envisages and deals with the reality.

Booch [Booch, 1994 (pp. 27-80)] defines abstraction as the process which denotes the essential characteristics of an object which distinguish it from all other kinds of objects and thus provides crisply defined conceptual boundaries, relative to the perspective of the viewer.

The process of abstraction results in the identification of classes in the problem domain.

For example, an area in a city may have one way streets, two way streets, a hospital, a university, apartment dwellings, traffic lights, a river, a hill and various other phenomena. According to the perspective of the viewer, this could be abstracted in various ways.

A gas supplying company would not be interested in the river and the hill but would need the information regarding the buildings which use their gas service. They would also need information about the roads under which their gas pipes are buried. It would then be sensible to have in their software design a CBuilding class and a CRoadNetwork class. It is easy in this case to realise that hospitals, universities and apartments share common essential characteristics which differentiate them from a road or a hill. The identification of the CBuilding class can be regarded as an abstraction of the various structures which use their gas services. Hospitals, universities and apartments would be the instances of the CBuilding class.

In photogrammetry, coordinates of an image point, perspective point and the control point are used to form the collinearity equations in the bundle adjustment. At the initial stage of system development it is difficult to understand how ground control points, images of the ground control points, and the perspective points will be utilised for the adjustment, and then how the image points will be associated with the image rotation. But even at this stage it can **generally** be concluded that these different types of points will **somehow** be applied in the adjustment in association with the image rotation. It is certain that points are important in the adjustment process and there are some characteristics of a point that would differentiate it from an image or a camera.

For example, all points share the common characteristics of having a set of coordinate values in a coordinate system and are unique in space. It would be logical then, in the Object Orientation sense, that an abstraction of the point class is included in the design, for example as CGenericPoint. The details of how image points are different from control points need not

be investigated at the initial stage. Those can be determined later. For the initial stage, one can just start with the point class and define its properties and behaviours.

Abstraction makes complexities manageable. It is a process of investigating various objects in the problem domain and classifying the objects by common features and roles, and then defining such a collection of objects as belonging to a class. Making complex things into simple blocks is a major advantage in software design.

2.1.2 Encapsulation

Encapsulation is the feature which makes reusability and compatibility of programs easy.

Encapsulation, which can also be called data-hiding, means that similar functions and the data which are used by these functions are grouped together into one place, a class, and are accessed in a controlled way. This is necessary to limit the opportunity for modules to affect one another. The example described below, relating to an image class called CMappingImage, demonstrates the encapsulation feature in C++.

In C++, encapsulation of a class is realized through the use of the keywords 'private:' and 'public:'. The members of a class which are placed under the private keyword are prohibited for use by an external source. These members can only be used within the class. Usually those functions which will be used by other classes are placed in the public part.

An image class is a typical class used in photogrammetry. Some of the important attributes of an image are rotation angles, perspective point and image point list. An image class called CMappingImage can be defined as follows in C++. (The symbol '//' signifies that what follows after this symbol is a comment to explain the program code)

```
class CMappingImage
{
CMappingImage();
~CMappingImage();

private:
double Omega, Kappa, Phi; //rotation angles
CPerspectivePoint thePerspectivePoint;
CMatrix RotationMatrix;
CPointList theImagePoints;
void SetRotationMatrix();

public:
void SetRotationAngles(double omega, double phi, double kappa);
CMatrix& GetRotationMatrix();
void AddImagePoint(CImagePoint* theImagePoint);
void DeleteImagePoint(CImagePoint* theImagePoint);
};
```

From the above example, the rotation angles and the perspective point, the rotation matrix, and the image points are declared in the private part. This means that they can only be accessed by the functions of the same class, i.e. member functions. However the functions in the public part such as SetRotationAngles, GetRotationMatrix, AddImagePoint and DeleteImagePoints can be used by other functions. It is through these public member functions that classes interact with each other.

The mechanism to make some data members of a class inaccessible to other classes is a way of implementing encapsulation. This will make a particular class modular and independent.

A subject related to encapsulation is that of coupling and cohesion. **Coupling** is the measure of the strength of association from one module to another. In Object Orientation this can be translated to the interaction between different classes. In Object Orientation it is desirable that classes have weak coupling. This would result in stronger modularity and reduced complexity. In C++ weak coupling would mean that a member function of a class interacts with very few other classes. **Cohesion** is the degree of connectivity among the elements of a single class. High cohesion implies good encapsulation.

For example, adjustment in photogrammetry usually involves various matrices which are formed by observations [Mikhail, 1976 (pp. 3-8)]. The observations can be coordinates of image points, pseudo-ranges of a GPS receiver or acceleration rate of an IMU. It would be advantageous if an adjustment class in photogrammetry, example CAdjustment, has only objects of CMatrix class as its data members, instead of objects of CPoints or CGPSObservations. The next CAdjustment class defined in C++ code will demonstrate this.

```
class CAdjustment
{
CAdjustment();
~CAdjustment();

private:
CImagePoints imagePoints;
CPerspectivePoints perspectivePoints;
CControlPoints controlPoints;
CGPSObservations gpsObs;
CMatrix ObservationMatrix, WeightMatrix, ResidualMatrix, NormalMatrix;

public:
void SetGPSObservationMatrix();
void SetImagetObservationMatrix();
void FormNormalMatrix();
void Solve();
}
```

If the adjustment class has objects of point class, or objects of GPS observation class as its data members forming the relevant ObservationMatrix, as in the above code example, this would mean there is interaction with increased numbers of other classes. This implies high coupling. This also means that the adjustment class must exist with those classes. Under such a situation, the member functions of the adjustment class would also need to interact with the data members of other classes. This implies low cohesion. Encapsulation is weakened.

But if the adjustment class has only matrix related members then the result is the desired case of weaker coupling and higher cohesion; weaker coupling because it interacts with only the CMatrix class and higher cohesion because the class works with fewer classes involved, only the CMatrix objects. The next example shows the case where only matrix objects are involved.

```
class CAdjustment
{
CAdjustment();
~CAdjustment();

private:
CMatrix ObservationMatrix, WeightMatrix, ResidualMatrix, NormalMatrix;

public:
void SetObservationMatrix(CMatrix theObservationMatrix);
void FormNormalMatrix();
void Solve();
}
```

The second example shows a stronger encapsulation than the first example, because it is more self sufficient or independent than the first example. This implies that the CAdjustment code can be reused in many situations without any change, in contrast to the first example where it is applicable only for adjustment of image observations and GPS observations.

To increase the strength of encapsulation, it is necessary to identify, during the design stage, the possible environments in which the class will be used. Encapsulation is the feature which makes the reusability and compatibility of programs easy.

2.1.3 Inheritance

Inheritance is easily demonstrated by using the point class again. A point can be described as an object having a set of coordinates and related accuracy in a coordinate system. It usually has a unique number or a name to identify itself. These properties are common to all points and can be defined as the base class mentioned above, CGenericPoint. A CImagePoint,

CControlPoint or CPerspectivePoint also share these properties but have further different properties characteristic only of themselves. These classes can be derived from the base class. The derived classes inherit the properties of the base class. Looking for a base class is also a part of the abstraction process and inheritance is a result of such an abstraction process. In C++, inheritance is declared by using the colon in the class definition. For example, 'CImagePoint : CGenericPoint' means that the class CImagePoint is derived from the CGenericPoint class.

In the example below, CGenericPoint is the base class representing all types of points. It has PointName and PointCoordinate as its data members in the private domain and AssignPointName member function in the public domain.

```
class CGenericPoint
{
public:
CGenericPoint();
~CGenericPoint();
void AssignPointName(CString aName);

private:
CString PointName;
CCoordinate PointCoordinate; //embedded object, aggregation of an object in another class
}
```

Also note that the PointCoordinate data member is an object of the CCoordinate class. The CCoordinate class for example could have been defined as follows.

```
class CCoordinate
{
public:
CCoordinate();
~CCoordinate();
void TransformCartesianToGeodetic();
void TransformGeodeticToCartesian();

private:
double X, Y, Z;
double dx, dy, dz;
double phi, lambda, height;
double dphi, dlambda, dheight;
}
```

The CImagePoint class, defined below, is derived from the CGenericPoint (see above). This means that the PointName and PointCoordinate data members as well as the AssignPointName(CString aName) member function of the base class are inherited in the

CImagePoint, in addition to its own data member AdjustedResidual and GetAdjustedResidual() member functions.

```
class CImagePoint : CGenericPoint //CImagePoint is derived from the CGenericPoint
{
public:
CImagePoint();
~CImagePoint();
double GetAdjustedResidual();
private:
double AdjustedResidual;
}
```

The benefit of inheritance from the above example might seem insignificant. But imagine an image class, called for example CVisionImage, used in the Computer Vision research field, and within this class a hundred or more functions for image display, image processing, image analysis, etc., are already implemented. These functions could be available for use by a photogrammetrist for object recognition research by just declaring as follows:

```
include "VisionImage.h"

class myImage : CVisionImage
{
public:
myImage();
~myImage();
MyOwnFunction();
...
...
}
```

MyOwnFunction() in myImage class is now able to use all the functions implemented in the CVisionImage class because myImage class has inherited them. Inheritance is the key feature which makes software easily extendible.

Another benefit of inheritance is that it makes the program legible. If all the same functions and data members of a base class have to be included in every derived class, the program would be very long and complicated and it would be difficult to understand the code.

2.1.4 Polymorphism

In the Object Oriented method, letting the system take appropriate actions depending on the type of data is called polymorphism. This leads to efficient maintenance of the software.

As a mathematical example, a ray of light traveling from the ground control point passing through the perspective point and then to the image point is modeled by the collinearity

equations. The coordinates of each point through which the ray passes are used to form the collinearity equations. The adjustment to obtain camera calibration data or a tie point's coordinates involves the formation of the observation matrix using the collinearity equations.

The application of polymorphism in the formation of the observation matrix can be demonstrated. This uses the knowledge that the position of coordinate values in the observation matrix will be decided by the type of point and the fact that different types of points are derived from a base class called CGenericPoint.

First, the different points from all types through which the ray of light passes will be added to a list of points. Then the virtual function (the function declared in the base class which will also be declared at the derived classes) for forming the observation row will be declared at the base class, CGenericPoint, and finally the same function will be defined in the derived point classes, but each implemented to act according to its own type.

In C++ programming, a ray of light can be modeled as a list composed of pointers to the objects of the point class, CGenericPoint. This feature can be illustrated by a template-based collection class, as follows.

```
typedef CTypedPtrList <CPtrList,CGenericPoint*> CRay;
```

The above statement declares CRay as a pointer list and the type of pointers are pointers to objects of the CGenericPoint class. It is derived from a parameterised class (i.e. a class with parameters) CTypedPtrlList which takes CPtrList and CGenericPoint as its arguments.

Now points through which the ray of light passes can be added to the CRay.

In the example below, aRay object of the CRay class is instantiated in the first line. An object of the CImagePoint is also created as anImagePoint. The aRay is defined in the stack and the anImagePoint is defined as a pointer pointing to the address at a dynamically allocated memory in the heap. Note that although CRay is defined as a list of CGenericPoint, it will accept its derived objects, anImagePoint of the CImagePoint class and other objects of CPerspectivePoint and CControlPoint. The aRay object now has a anImagePoint, aPerspectivePoint and aControlPoint as its members, indicating these are the points through which the ray of light passes.

```
CRay aRay;
CImagePoint *anImagePoint = new CImagePoint;
aRay.AddTail(anImagePoint);
CPerspectivePoint *aPerspectivePoint = new CPerspectivePoint;
```

```
aRay.AddTail(aPerspectivePoint);
CControlPoint *aControlPoint = new CControlPoint;
aRay.AddTail(aControlPoint);
```

The next task is to declare a virtual function FormObservationRow() in the base class CGenericPoint. Then for each derived class of CImagePoint, CControlPoint and CPerspectivePoint, define an overloaded function (i.e. the function that is declared in the derived class that has the same name of the virtual class of the base class) FormObservationRow() which assigns the coordinate values to the appropriate vectors and matrices at the correct places according to the type of point. An example of the code implementation in C++ could be:

```
class CGenericPoint
{
public:
CGenericPoint();
~CGenericPoint();
void AssignPointName(CString aName);
virtual void FormObservationRow(CMatrix &ObservationMatrix); //the virtual function

private:
CString PointName;
Coordinate PointCoordinate;
}
```

The virtual keyword declares that FormObservationRow() function will be defined differently by a derived class, such as a CImagePoint as shown below.

```
class CImagePoint : CGenericPoint
{
public:
CImagePoint();
~CImagePoint();
double GetAdjustedResidual();
void FormObservationRow(CMatrix &ObservationMatrix); //image point overloaded function

private:
double AdjustedResidual;
}
```

The FormObservationRow function of the image point will now react as defined by the CImagePoint class and not as was defined in the CGenericPoint. It will position its coordinate values at the place in the observation matrix which would be correct for an image point. The same applies to other derived classes such as the CControlPoint or the CPerspectivePoint.

Finally, the points from the CRay list are processed one by one to form a complete row of the observation equation.

```
CGenericPoint *aPoint;
CRay *aRay;
POSITION pos = aRay->GetHeadPosition(); //put the list iterator at the beginning of the list
while(pos){
   aPoint = aRay->GetNext(pos); //get the pointer in the aRay list
   aPoint->FormObservationRow(CMatrix &ObservationMatrix); //the overloaded function
}
```

The program just executes aPoint->FormObservationRow(CMatrix &ObservationMatrix) without any consideration as to whether the point is an image point or a control point. The system will decide which version of the FormObservationRow function to execute: the image point version; perspective version; or, the control point version. In a procedural programming language such as the C language, this would have to be implemented by using a set of 'if ~ then' statements or 'switch ~ case' statements. The problem with this is not the difficulty in writing a few more lines, it is that the programmer has to be aware of this fact. If he or she makes any changes in any versions of the FormObservationRow function, all occurrences of the function in the program should be found and repaired. In contrast, when using polymorphism as in the above example, only the overloaded function needs to be changed. The procedural method is very error prone and changes would be difficult to locate if there are many such cases and especially if the programmer has been replaced by a new member in a team.

In the above example, the function FormObservationRow took on different forms depending on the type of point, whether it was an image point, a perspective point or a control point. The function was polymorphic. Letting the system take appropriate actions depending on the type of data is called polymorphism and leads to efficient maintenance of the software.

2.2 Object Oriented Design

The facilities provided by an Object Oriented system mentioned as above, namely abstraction, encapsulation, inheritance and polymorphism, improve error prone programming situations. The programs using such facilities would also be much more easily understood, which means that maintenance is more efficient. The Object Oriented design aims to apply and incorporate abstraction, inheritance, encapsulation and polymorphism into a program while meeting the requirement of the system.

2.2.1 Design Goals

The goal of the Object Oriented design is two-fold: to fulfil the user requirement accurately as represented in the external view; and to produce a software which is easy to implement and maintain as represented in the internal view [Meyer, 1988]. These views are considered below.

First is the need to satisfy the needs of the end user. This is the **external view**. The produced design must fulfil the requirement of the user accurately. The end product, a computerised system, must serve the end user's need quickly and efficiently. For example a stereoplotter operator using an automated mapping system to collect data on buildings would not be interested in abstraction or inheritance. But he or she would be interested in how many times the mouse button would need to be clicked to complete his or her task, or whether the system gives out a quantified report on the reliability of the recognition process. Good software must be designed to meet such user's requirements.

Second, the software engineer is considered. This is the **internal view**. The design must be made in such a way that besides meeting the user's requirements, it must be easy to implement and maintain. For example, changes to the design, due to changes in the requirements or other circumstances, should be relatively simple. Modules of software should be reusable in similar situations with minimum effort. New members to the development team should not take too long to understand the existing design. These, as others, are some of the goals of the design from a software engineering aspect.

2.2.2 Unified Modeling Language

In the computer science discipline, various Object Oriented programming languages have been developed over the years as well as different approaches to analysis and design methodology, using Object Oriented concepts.

UML is, as the name suggests, a modeling language for the Object Oriented methodology. It began as a response to the Object Management Group's (OMG) request for a standard in the Object Oriented methodology. OMG [OMG, 1999] states that UML "is defined as a language for specifying, visualising, constructing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems". The OMG is an international organization supported by more than 800 various members of the computer science discipline. It was founded in 1989 to promote the theory and practice of Object Oriented technology. Later the OMG accepted the UML as the standard [Douglass, 1999]. It

is stated by some experts that UML will become in time a core skill for software engineers [Pooley et al., 1999].

The UML has been adopted as the design tool for this study, because it has been accepted as the standard by the main standardisation body for Object Oriented matters, the OMG.

Features of UML include the Object Model; Use Cases and Scenarios; and, Behavioural Modeling with Statecharts. These features are used in the analysis and design stages of the system to capture domain characteristics accurately and represent them in a standard way. This will serve to link and communicate information to later stages of the implementation, testing and maintenance of the developed system. UML uses various types of diagrams to document and relay views of a model, including:

- Class Diagram
- Use Case Diagram
- Behaviour Diagram
    - Statechart Diagram
    - Activity Diagram
    - Interaction Diagram
        - Sequence Diagram
        - Collaboration Diagram
- Implementation Diagrams
    - Component Diagram
    - Deployment Diagram

The following paragraphs will give examples of the basic diagrams in the UML, namely the Class Diagram and the Use Case Diagram.

The **Class Diagram** is the basic diagram that shows the classes in the system domain. This is shown in Figure 2.1. A Class Diagram represents the class as a rectangular box. The box has three compartments. The top compartment labels the name of the class and the second shows the data members. The third shows the member functions or the operations of the class. Templates or parameterised classes, have a dotted rectangle on the right corner to show the arguments the template is taking. Relationships between classes are shown as lines joining the two classes. An arrowline with the closed arrowhead represents an inheritance

relationship with the arrowhead at the base class. A line with a diamondhead represents aggregation, with the head at the containing class. A dotted line with the open arrowhead shows a dependency relationship.

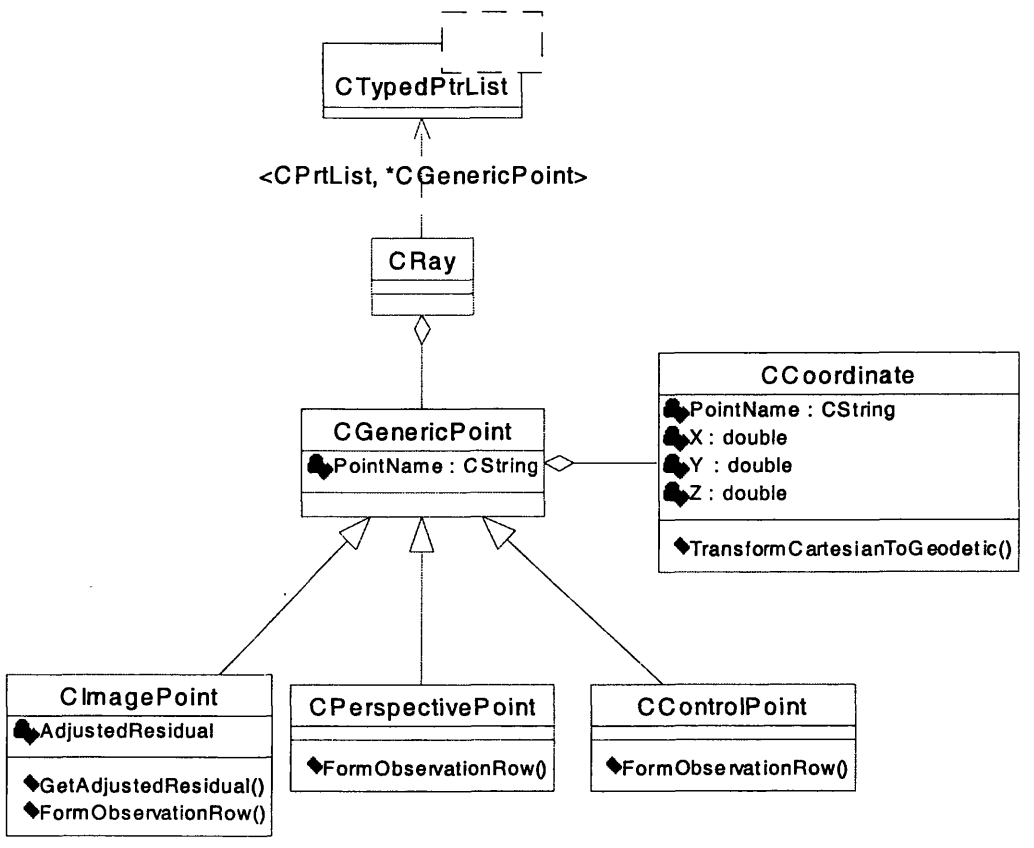

Figure 2.1 Class Diagram Example

There are many kinds of dependency, but the one shown on Figure 2.1 is a bind dependency. It explains that the CRay class is a specialization of the template class, i.e. derived from CPtrList class, but because the template actually binds the arguments rather than derives the child class, it uses this special symbol to show such a relationship.

As can be seen from the diagram above, this is a very compact and efficient way of documenting what was explained verbally in the first half of this chapter!

Another key feature of the UML is the application of a **Use Case Diagram** to define what the system would do from a user's point of view. The Use Case Diagrams show how an actor, i.e. object outside the context of the system, would use the system without showing the design structure of the functions of the system itself. Use Case Diagrams are important to communicate the key requirements of the system as well as defining the boundaries of the

system to be developed. They are also useful in guiding the direction of the development and to check if development is on track for meeting the user's requirements. The Use Case Diagram of the automated mapping system at its top level is shown in Figure 2.2.

Actors are the figures who use the system and each elliptical object is a Use Case, i.e. a case of how the system would be used. A line depicts a relationship of 'uses' or 'extends' or even 'is-a' as in the inheritance relationship. In the figure, there are five actors, i.e. the pilot, the imaging sensor specialist, the positioning sensor specialist, the geospatial database specialist and the GIS visualisation specialist. For example, the pilot uses the automated mapping system to monitor the flight. So Monitor Flight is a Use Case of the system.

By way of further explanation, in the case of the positioning sensor specialist, he or she will use the automated mapping system to tag the image with position and rotation information. The Tag Image Use Case will in turn use other Use Case modules which will process GPS data and IMU data, namely the Process GPS Data Use Case and the Process INS Data Use Case.

A Sequence Diagram models the dynamic aspects of the system and emphasizes the time ordering of messages that are exchanged between objects. In a Sequence Diagram (for example Figure 2.3), objects are represented by rectangles and these are arranged as a row on top of the page. Messages between two objects are represented by a horizontal arrow linking the two objects. The vertical positions of these arrows shows the time ordering of messages.

The Sequence Diagram of Figure 2.3 shows the sequence of events between various objects when data files are read during the initial phase of camera calibration. These data files are the sensor data file, image data file and ground control data file. When the file names are keyed in by the user, the data from these files will populate the attributes of objects with corresponding values. These objects are then grouped to form the bundles of rays.

Further details of UML are described in OMG's publication "OMG Unified Modeling Language Specification" [OMG, 1999], but they have not been found essential to this work. The application of UML in analysis and design does not necessarily imply that a better design will be produced compared to when it is not applied. It is a method of documenting what has been perceived by the designer as a result of observation and investigation.

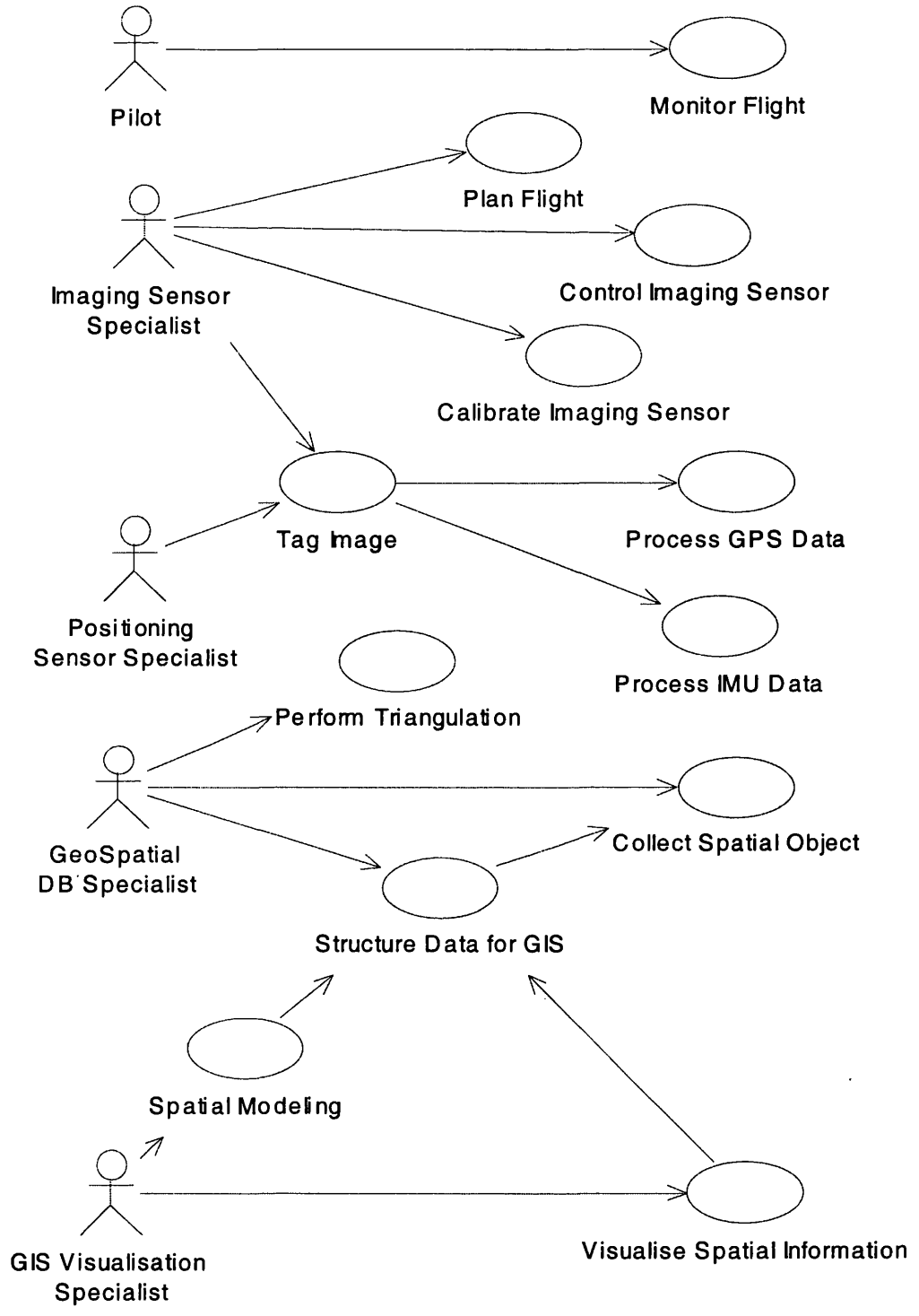

Figure 2.2 Use Case Diagram Example

The result of UML could be used as a useful means of discussing and confirming what has been perceived. This will of course improve the understanding of the problem and probably lead to a better design. But in all cases, the initial phase of being able to perceive the problem clearly will be a prerequisite to a good design. It is important to note that software design, or any design, is not a trivial activity and using a methodology or a tool will help the situation but it will not solve the problem completely.

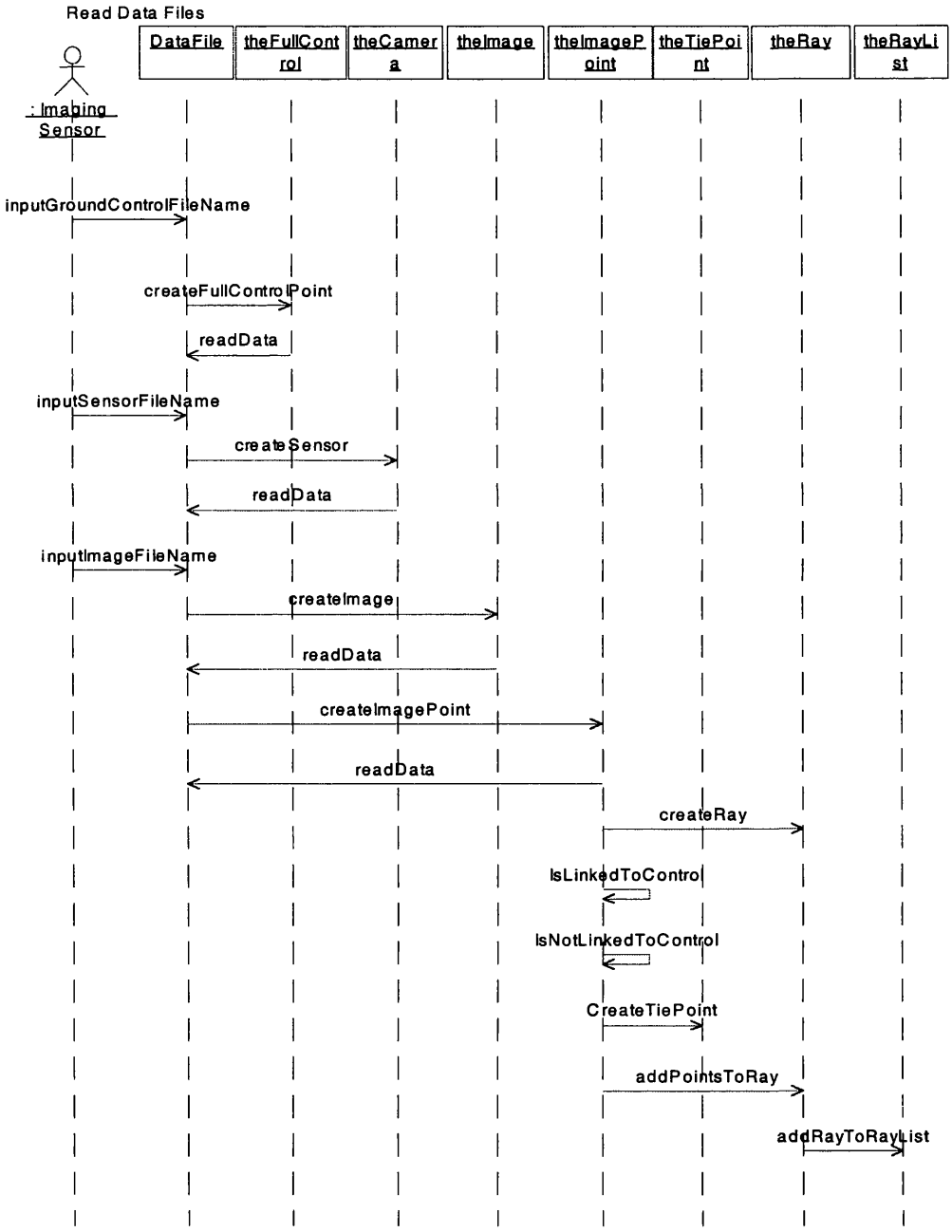

Figure 2.3 Sequence Diagram of Read Data File

3. ANALYSIS AND DESIGN OF THE IMAGE ACQUISITION SUBSYSTEM

In this chapter the analysis and the design of the image acquisition subsystem will be described.

In the **analysis stage**, the main roles and the processes of the image acquisition subsystem will be explained. The main roles of the image acquisition subsystem are classified as:

- calibration of the imaging sensor;
- control of the image capturing process; and,
- flight planning.

A Use Case Diagram of the image acquisition subsystem is produced as the result of the analysis. After investigation of each role, candidate classes are identified. The identified classes are then further explained with regards to their purpose and some background information will be provided in the photogrammetric sense. Sequence Diagrams will show the actions that are followed in the processes of selected Use Cases.

In the **design stage**, the Use Case Diagrams and the Sequence Diagrams are then used to produce the Class Diagrams which show classes with all the relevant data members and member functions as well as their relationships with other classes. The design of an automated mapping system involves disciplines including photogrammetry and geodesy, and thus some choices had to be made regarding terminology. In photogrammetry and many other disciplines, 'coordinate system' usually refers to the reference system to which the values of the coordinates of points are referenced, such as 'photo coordinate system' and 'pixel coordinate system'. But in geodesy it is conventional to use 'coordinate frame' for such purpose. 'Coordinate system' in geodesy refers to the coordinate frame as well as the physical theories and their approximations that are used to define the coordinate axes [Jekeli, 1998]. For consistency purposes, the term 'coordinate frame' will be used in this thesis to refer to what would be called 'coordinate system' in photogrammetry. Therefore, terms such as 'photo coordinate frame', 'pixel coordinate frame' and 'local coordinate frame' will be used.

The description of the roles and the processes of the image acquisition subsystem given above can be summarised by the Use Case Diagram as shown below (Figure 3.1). This diagram shows how different actors (i.e. human operators, imaging sensors and GPS receivers) would interact with the image acquisition subsystem. It also shows that the system

would be used to calibrate image sensors, control imaging sensor and to plan and monitor the flight.

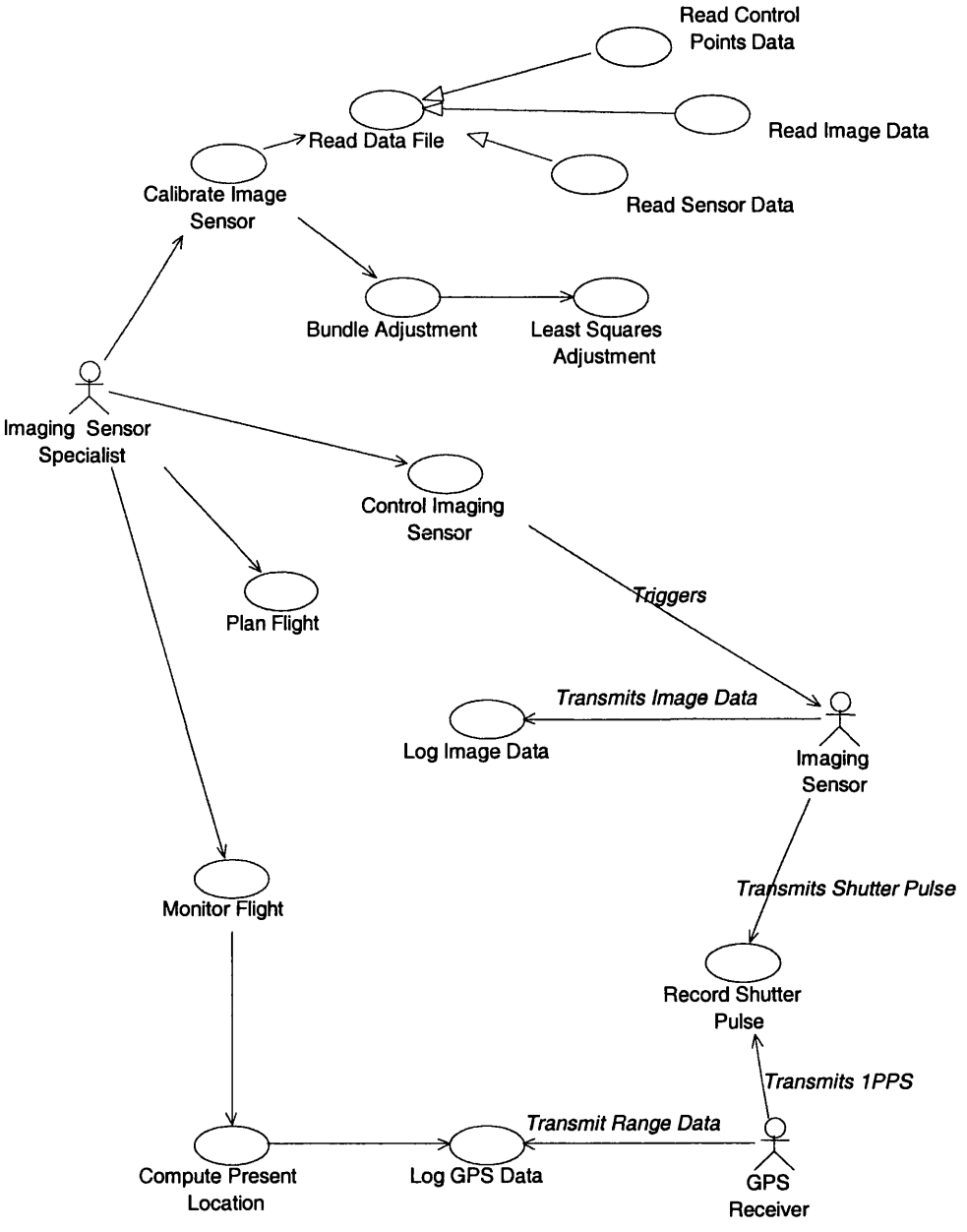

Figure 3.1 Use Case Diagram of the Image Acquisition Subsystem

3.1 Calibration of the Imaging Sensor

In the image point referencing process, which is essential in the mapping process, some quantified values regarding the camera and the image at the instant of exposure are

necessary. These values regarding the camera are called interior orientation parameters and the values regarding the image are called exterior orientation parameters.

The quantities to be determined in interior orientation (interior orientation parameters) are the principal distance (also known as camera constant), the principal point offset from its initial approximate position (usually symbolised as xp and yp) and additional parameters (usually representing lens distortion coefficients and film distortion coefficients, i.e. uniform linear scale changes and non-orthogonality of the x-y coordinate axes). The determination of these parameters is called camera calibration. These camera calibration data are then used to transform all observed coordinates of image points in an arbitrary coordinate frame (usually pixel coordinate frame) to the photo coordinate frame (definition of the different types of points and coordinate frames will be covered in the later part of this chapter). The coordinates in the photo coordinate frame are then transformed to the ground coordinate frame using the exterior orientation parameters.

Exterior orientation parameters are the three angles of rotation of the image and the ground coordinates of the perspective point in a Cartesian coordinate frame. Exterior orientation will be further explained in Chapter 5, where image point referencing will be discussed in detail.

Although there are several established methods of camera calibration, treatment will be confined to the self-calibration method in this research.

The unknown parameters of camera calibration are determined through the least squares adjustment process. Accurate control point coordinates are first acquired through surveying or some other methods which will produce observations of high accuracy. Images of the control points are then acquired and the corresponding image points are observed to get their image coordinates. The image coordinates, control point coordinates and the unknown parameters are used in setting up a set of equations, the collinearity equations in this case. The unknown parameters are then solved with the constraint that the squares of the residual (the difference between the observed and the computed value) are minimised. Usually in traditional film based cameras, coordinates of the fiducial marks on the films are also provided as part of the calibration data. When scanned digital images of the films are to be used, the fiducial marks also have to be observed to get their pixel coordinates. These will establish a set of transformation parameters from the pixel coordinate frame to the fiducial coordinate frame. All pixel coordinates will first be transformed to the fiducial coordinate frame before being transformed to the photo coordinate frame.

For most digital camera systems however usually no fiducial marks are present and the calibration data are not provided. In this case the principal point coordinates are the distances

from the centre of the CCD sensor chip. This distance and other parameters are determined through the camera calibration process. Pixel coordinates are then transformed to the photo coordinate frame.

Lens distortion is characterised by its radial components and its tangential components. The radial component of the distortion of the lens increases with the increase in the radial distance from the principal point and it is assumed to be symmetric about the principal point.

The radial components of lens distortion are characterised by the following equations [Moniwa, 1972]:

$$dr_x = \left(x - x_p\right)\left(k_1 r^2 + k_2 r^4 + k_3 r^6\right) \qquad (3\text{-}1)$$

$$dr_y = \left(y - y_p\right)\left(k_1 r^2 + k_2 r^4 + k_3 r^6\right) \qquad (3\text{-}2)$$

where, x_p, y_p are the coordinates of the principal point, dr_x, dr_y are the x and y components of the symmetric radial distortion at the radial distance r and k_1, k_2, k_3 are the unknown coefficients of the polynomial.

The tangential components of lens distortion are characterised by the following equations [Moniwa, 1972]:

$$dp_x = p_1\left\{r^2 + 2\left(x - x_p\right)^2\right\} + 2p_2\left(x - x_p\right)\left(y - y_p\right) \qquad (3\text{-}3)$$

$$dp_y = p_2\left\{r^2 + 2\left(x - x_p\right)^2\right\} + 2p_1\left(x - x_p\right)\left(y - y_p\right) \qquad (3\text{-}4)$$

where, dp_x, dp_y are the x and y components of the tangential lens distortion, p_1, p_2 are the unknown coefficients, x_p, y_p are the coordinates of the principal point, x, y are the image coordinates of a point and r is the radial distance from the principal point.

The film distortion is usually modeled by the affinity equation as follows [Moniwa, 1972]:

$$dq_x = A\left(y - y_p\right) \qquad (3\text{-}5)$$

$$dq_y = B\left(y - y_p\right) \qquad (3\text{-}6)$$

where, A, B are the parameters defining scale change and non-orthogonality of coordinate axes. During the camera calibration process the unknown lens distortion parameters, namely

k_1, k_2, k_3, p_1, p_2, A, B are determined as well as the coordinates of the principal point x_p, y_p and the principal distance.

The actual determination of these values involves a process known as bundle adjustment in photogrammetry. During bundle adjustment, all parameters, image coordinate observations and ground control points coordinates are used to model a bundle of rays. The mathematical condition is modelled by the collinearity equations. Bundle adjustment will be explained in detail in Chapter 5, which describes the image point referencing subsystem.

The camera calibration process begins by reading data files. This sequence of actions are as depicted in the Sequence Diagram of Figure 2.3 of Chapter 2. The bundles of rays which were formed through reading the data files and going through the sequence of events of Figure 2.3 are then used in the least squares adjustment of the image point observations to determine the camera calibration data.

The observation matrix, the residual matrix and the weight matrix are formed. These matrices will then be used to form the normal matrix and the constant vector which will be solved for the unknown parameters. The interaction between objects in this process is shown in the Sequence Diagram of Figure 3.2

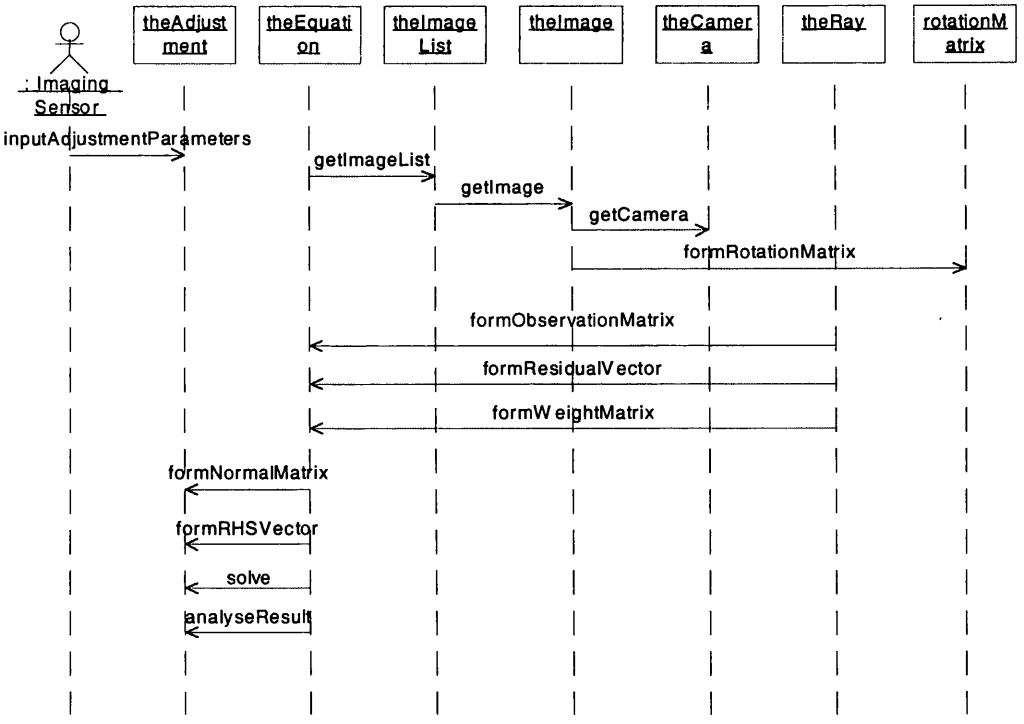

Figure 3.2 Sequence Diagram of Bundle Adjustment

3.2 Control of Image Capturing

The image capturing process can be considered to be the control of the imaging sensor and the recording of the time of shutter release.

Control of the imaging sensor involves setting up of camera and capturing of images.

The recording of the time is carried out by capturing the pulse from the shutter at the time of its release and then relating this to a given time frame. Usually external events such as the shutter pulse are captured in the computer clock time frame. These are then transformed to the GPS time frame. This can be done using the 1 Pulse-Per-Second (PPS) signal of the GPS receiver which is also captured by the computer. These 1 PPS signals will serve as reference data for the transformation. The final outcome of the time recording process is the GPS time for the shutter pulses.

3.2.1 Control of Imaging Sensor

In most film based cameras, the image capture environment, i.e. shutter release interval, the aperture, the focal length and others, are preset manually using the hardware instrumentation of the camera. In a digital camera, this should be done using a computer which is linked to the camera. Most digital camera manufacturers provide a software library to do this.

The development of the software module of an automated mapping system to set the capture environment usually would involve the utilisation of such a software library.

Some of the functions that should be present in the module to control the camera are as follows.

- Manipulation of camera properties (e.g. "initialise", "set", or "get current status")
- Capturing of images from multiple camera sources and directing each image file to specific directories in the storage media
- The capability to temporarily pause the preset image capturing motion as well as the capability to resume image capture either manually or by software. (It should be possible to trigger a temporary pause or resumption of image capture by the preset times, camera location or heading, e.g. when the airplane is in an east-west direction and within the area of interest.)
- Quick review of captured images by the imaging specialist and marking faulty images. (It should be possible for the imaging specialist to make a quick decision whether to take any complementing images while the plane is still within the vicinity of the region of interest.)

- Concurrent error reporting. (Early warnings should be provided by the system regarding the available disk space, and any problems with cameras or their power supply.)

3.2.2 Exposure Time Recording

Exposure time recording is done using the shutter pulse and a clock system. Many digital cameras are equipped with an interface to a flash which is used to synchronise the bursting of the flash light with the release of the shutter. This facility can be used to record the exposure time. The pulse from the flash interface is sent to a computer or a GPS receiver. Some GPS receivers have an interface which will record the camera pulse directly and generate a file of consecutive events with corresponding GPS time.

The pulse could also be sent to a computer which will record the time in the computer clock time frame. The computer clock should be transformed to the GPS time frame because the position of the moving camera will be given in the GPS time frame. Some GPS receivers generate very accurate 1 PPS signals. This can be used to correct the computer time as well as to transform the exposure time of the image from computer clock time frame to the GPS time frame.

This recorded time of the image exposure in GPS time is necessary to interpolate the position and the attitude of each image from the positioning sensors of GPS and IMU. These positioning sensors provide positional and attitude data at different time rates. The following diagram (Figure 3.3) depicts an example of the different time scales at which each sensor acquires its data. The position and the attitude should be computed for the instant of the dotted line.

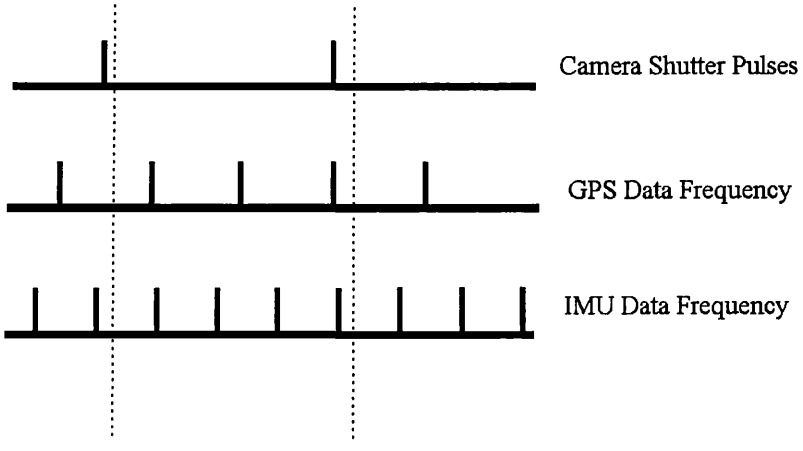

Figure 3.3 Time Scales of Different Sensors

Transferring of electronic pulses involves special hardware interfaces between different devices, such as camera, GPS receiver and computer. (In the positioning subsystem of Chapter 4, IMU data produced in the IMU time frame are also transformed to the GPS time frame).

It should be possible to tag each image with the time of exposure either automatically or from a created file. If the time tagging facilities are present, each image file produced should contain the time of exposure in its header. This could be included automatically at the moment of exposure or could be post processed after the flight.

As most image files are represented in standardised formats, such as BMP, TIFF, JPEG or PNG, the inclusion of extra information such as exposure time, perspective point or attitude to the image, should require some coordination with the standardisation organisations.

The Sequence Diagram for the process of controlling the camera and recording time is shown in Figure 3.4. The diagram is comparatively simple because it is assumed that a software library provided by the camera manufacturer is to be used. Most of the interactions would involve sending messages to this software library. The library will then send the appropriate messages to the camera or receive messages from the GPS receiver, the 1 PPS signal in this case.

3.3 Flight Planning

Flight planning is a comparatively simple task which has two purposes. One is to assist the imaging sensor specialist in planning the flight. The other is to assist the pilot who will be navigating the flight according to this plan. A good reference in flight planning can be found in the Manual of Photogrammetry produced by the American Society of Photogrammetry and Remote Sensing [Combs, 1980].

The major components of this module are the background map, the flight detail and the current location from the GPS receiver.

The background map is a scanned image of the area. The planning involves drawing planned flight lines on top of this map with the direction of flight shown with an arrowhead. The flying height, flying speed, and the time and the duration of flight are also included in the planning database. The planned forward and side overlap of images are also provided in the flight plan. All these will later be used for informing the pilot who will control the flight, and also for consideration of the images that were actually taken. The Sequence Diagram for flight planning is shown in Figure 3.5.

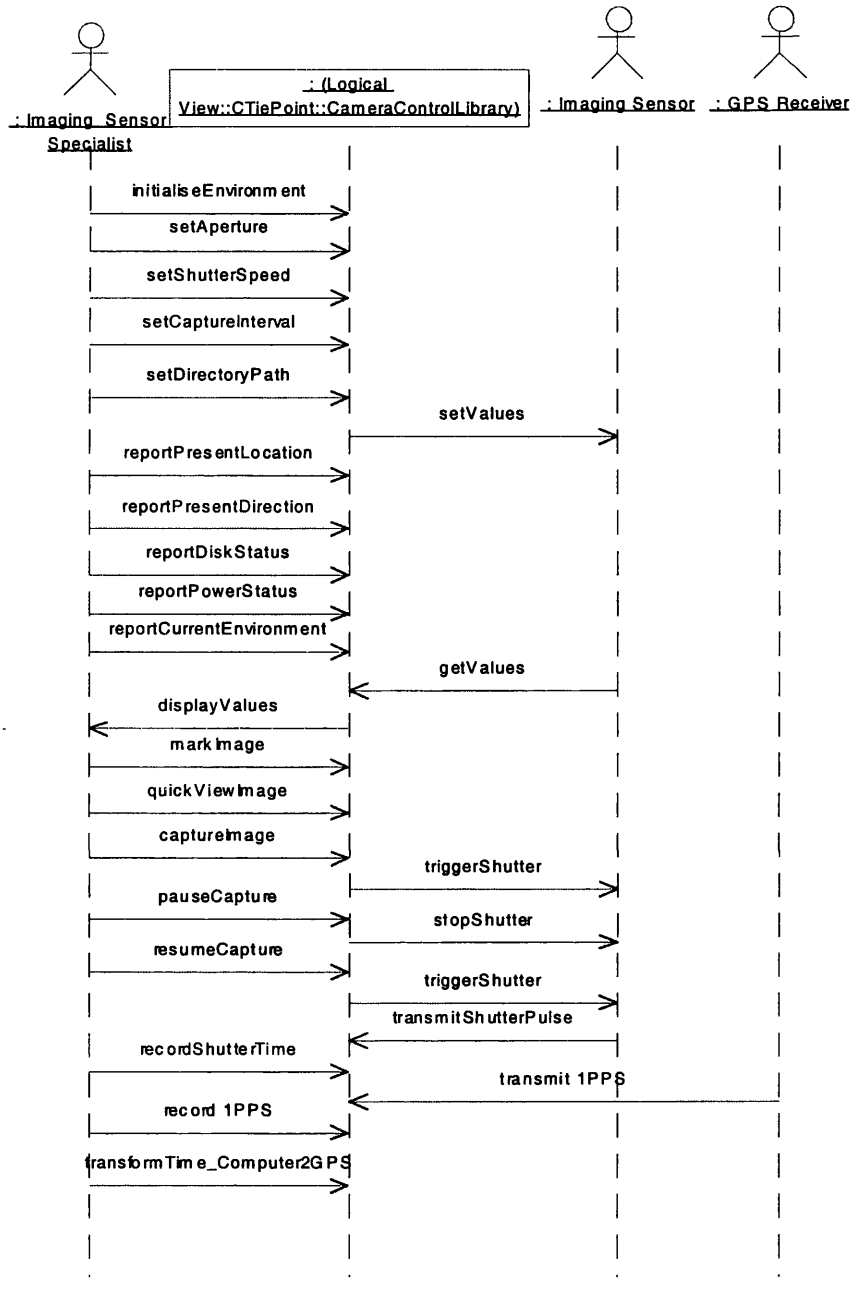

Figure 3.4 Sequence Diagram of Camera Control and Time Recording

When the program is initiated a general and small scale map will be displayed on the monitor. The user then selects and area of interest from this map and the area will be zoomed to a working scale. The user then draws flight lines on to this map and keys in the appropriate flight attributes such as flying height and speed. During the monitoring stage, the present location of the aeroplane will be computed from the GPS receivers which will continuously update the image which shows the original plan in its background. The imaging

sensor specialist will use this to inform the pilot and continuously monitor the flight in the planned direction.

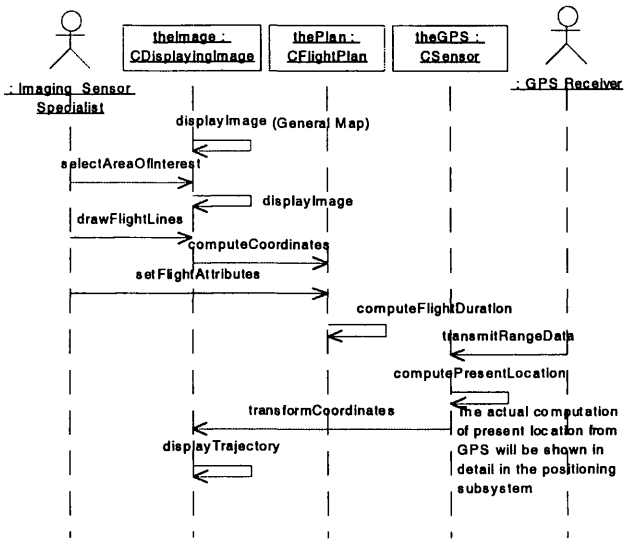

Figure 3.5 Sequence Diagram of Flight Planning

3.4 Important Objects in the Imaging Subsystem

The key elements of the Imaging Subsystem described above are the imaging sensor, the image, the points and the coordinate frames. These most probably will later be identified as classes together with many other related classes. In this section, an overview and the status of these important objects are provided. This information will be used in the design of the related classes.

3.4.1 The Imaging Sensor

Basically sensors can be seen as devices which convert a form of energy to an electrical signal. In a CCD camera the photo-site performs this transformation from the incident light to a number representing the light intensity at certain location of the scene. In a GPS receiver, the signals transmitted from the GPS satellites are transformed to pseudo-ranges and phases. In an IMU, the inertia created by movement is transformed to change in rotation and acceleration.

An imaging sensor is here defined as a type of sensor which produces images of the 3D scenes which include the topographic features of the earth. There are many types of imaging

sensors which are used for mapping purpose. Some, which are currently used or are being researched are listed below.

- Linear CCD Array camera
- Matrix camera
    - Traditional frame camera
    - Matrix CCD camera
- Active Pulse sensors
    - Imaging radar system
    - Laser Scanner

All these sensors use different mathematical models and have different processing methods. But they all share the common characteristic that they produce an image data set. This image data set is the result of a transformation by the sensor of the 3D scene to an image representing the scene. It is not attempted to model all these sensors in this research, and only the traditional metric camera and matrix type CCD camera will be modeled.

It is anticipated, however, that addition of a new sensor to the mapping system will involve software development tasks which will be simpler by using the inheritance and the encapsulation facilities of the object oriented design. For example a new sensor class could be derived from the generic sensor base class.

A class diagram of the imaging sensor showing only the class names and their relationships with other classes is shown in Figure 3.6.

Current Problems with the Digital CCD Camera in Photogrammetry

The traditional film based camera is still widely used in the mapping community. Images are reproduced on films and then transformed into a digital image using a digital scanner. The main reasons for this are the problem of digital cameras with regard to accuracy and the speed of image capture [Beyer, 1992][King, 1995][Mills et al., 1996].

The geometric accuracy of the digital camera is inferior to the film based camera mainly because of the size of the imaging area, the sensor size. A normal aerial film is about 23 x 23 cm. A typical CCD sensor of a digital camera could have 1548 x 1032 pixels with each pixel's dimension being 4.85 microns x 4.85 microns. This would mean that the sensor size is about 7.5 x 5 mm. The poor geometry of rays due to the small imaging area results in a decreased performance in accuracy. To produce a digital CCD sensor with size similar to that

of an aerial film introduces many problems such as irregularity in the flatness of the sensor plane, difficulty in management of the sensors, i.e. locating and exchanging faulty photo-site chips, and the speed of image capture, which is a major problem.

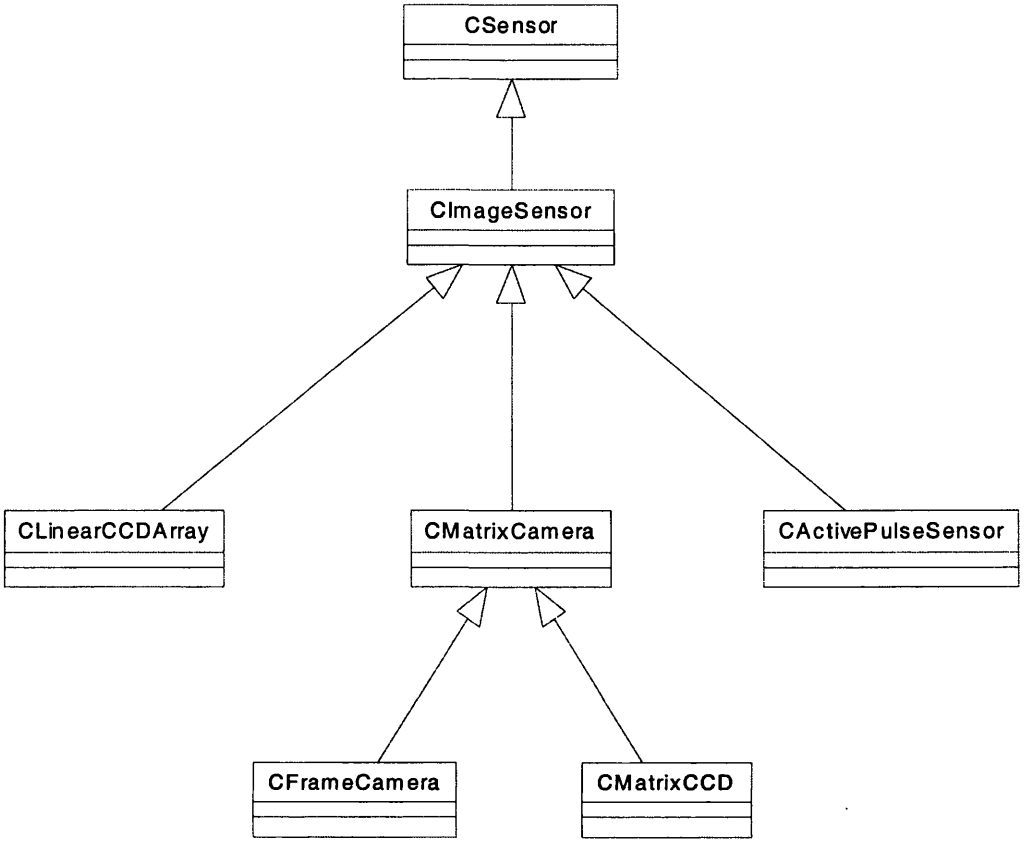

Figure 3.6 Class Diagram of Imaging Sensor

For a digital camera to capture two consecutive images, the photo-sites have to be refreshed and the quantized values transferred to a recording media. This is a time consuming task with the current technology. This limits the frame rate, i.e. the time necessary between two consecutive image exposures, effecting the flying speed of the aeroplane and the overall cost of the image capturing process.

However, where frame rate is not a major concern, and the distance of the object is relatively close, such as with most non-topographic photogrammetry, the problems mentioned above have little effect and the digital camera is suitable in applications such as industrial monitoring, architectural and historical monument preservation and restoration, robot vision and medical imaging [Sheffer et. al, 1989][Waldhausl, 1992][Streilein, 1995][Peipe, 1995].

3.4.2 The Image Class

Images are produced by an imaging sensor. An image is composed of grids of data with a spatial component and its corresponding thematic component. The thematic component can be light intensity as in a camera or height computed from travel distance of sensor signals as in the radar and laser sensors.

If the grid is two dimensional, the image is a two dimensional image. If the grid is three dimensional, it is a three dimensional image. Each grid point or pixel in the case of an image from a digital camera will have a set of coordinates in the grid coordinate frame as its spatial component and a corresponding thematic value.

Two types of two dimensional images have been considered, i.e. CMappingImage and CDisplayingImage. Although both refer to one image of a scene taken by an imaging sensor, they differ in their responsibilities and roles. CMappingImage is mostly concerned with the geometric aspects of the image and is mostly used at the initial stage of photogrammetric data processing. CDisplayingImage is used mostly to display, zoom and pan images. It is also used for changing image formats. It is most likely that CDisplayImage will use the end product of CMappingImage in the visualisation subsystem.

Images captured by the imaging sensor can have a reference point and an attitude. In the case of a frame camera the reference point is the perspective point and the attitude is the attitude of the camera at instant of capture.

In the case of a laser profiler, it is difficult to determine the optimum design for the signal of the laser sensor.

In the first case, the signal itself could be defined as an image, in which case the image will be a single point with X,Y,Z and attitude and the travelled distance. This means it could be defined as a one dimensional image. This will be advantageous for the process of calibrating the image acquisition subsystem, because the computation of the reference point and the attitude is common for all images. This implies that a function for the computation of camera attitude computation can be defined at the generic class of image and it will be applicable for camera as well as the laser profiling scanner.

In the second case, the constructed two dimensional image from the laser signals could be defined as an image and the signals as a list of embedded objects. This will be advantageous in the subsequent process of extracting spatial objects from various types of images, because most of the images at this stage are two dimensional. However, the member function for computing the reference point and attitude of an image class is repeated in another class, the

signal class. This means that if a change occurs in the computation of the reference point and the attitude, then the code in both the image class as well as the signal class should be updated. This inconsistency is something that the object oriented method aims to minimise.

In this research the first case will be selected for purposes of consistency, but more research into the design for incorporating various sensors is needed to come up with an ideal solution. A class diagram of the image class and its relationship with other classes is shown in Figure 3.7.

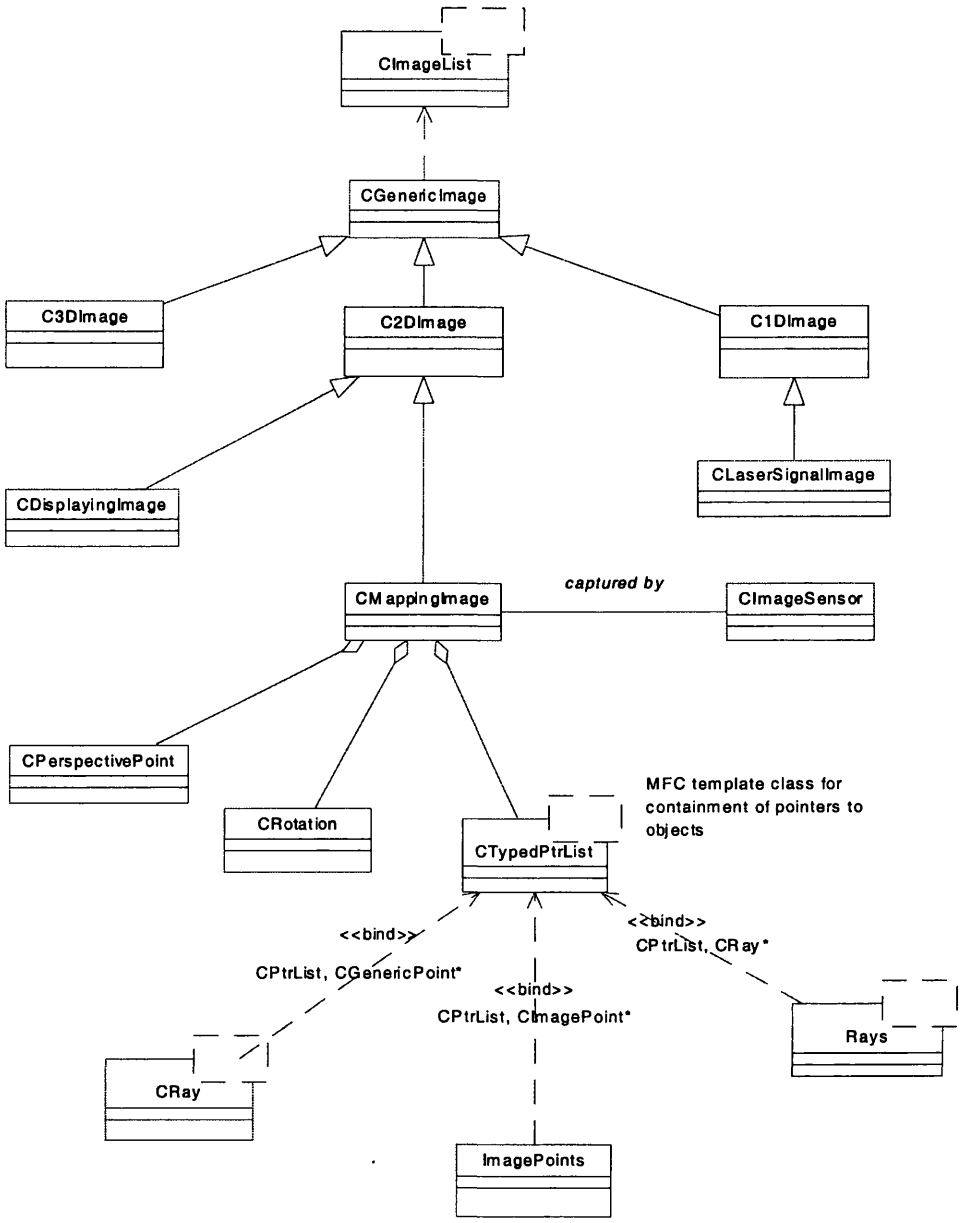

Figure 3.7 Class Diagram of the Image Class

3.4.3 The Point Class

A point is defined by its coordinates in a coordinate frame. The coordinates can be transformed between coordinate frames with the transformation parameters. The coordinates are usually computed values from observations and these coordinates usually have a set of associated numbers representing their accuracies. Also points are usually given a unique identifier to distinguish one from another.

Some examples of points used in the mapping systems are image points, control point, tie points, and perspective points as shown in Figure 3.8.

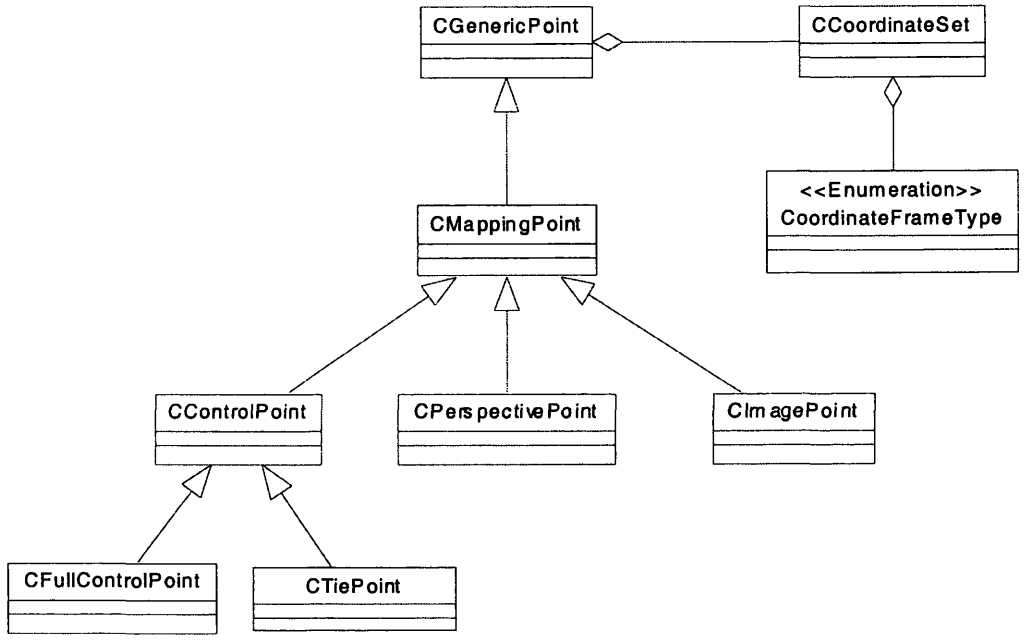

Figure 3.8 Class Diagram of Point Class

3.4.4 Coordinate Frames

The transformation between different coordinate frames is a common problem encountered in many mapping applications. The most important functions involved in coordinate transformation are scaling, rotation and translation.

Although the coordinate frame itself is important in understanding many of the theories in mapping applications, in software implementation, the transformation between different coordinate frames is of great concern. The transformation itself is incorporated into various mapping processes. For example, in the bundle adjustment of photogrammetric observations, the rotation matrix is formed and its elements are used in the forming of the collinearity

equations. In IMU data processing, rotation matrices are also formed to transform observations from a body coordinate frame to a local coordinate frame, or from an earth-centred-earth-fixed coordinate frame to a body coordinate frame.

A separate class of coordinate frame will not be defined in this design, as the transformation itself is realised in other classes in the form of observation equations and rotation matrices. Instead a coordinate frame type will be associated with the coordinate set to signify its coordinate frame type.

The pixel coordinate frame, shown in Figure 3.9, is a 2 dimensional coordinate frame usually defined with it origins on the top left corner of the image and its x axis extending right and the y axis extending down. The photo coordinate frame is a 3 dimensional right hand Cartesian coordinate frame and has its origin at the perspective point.

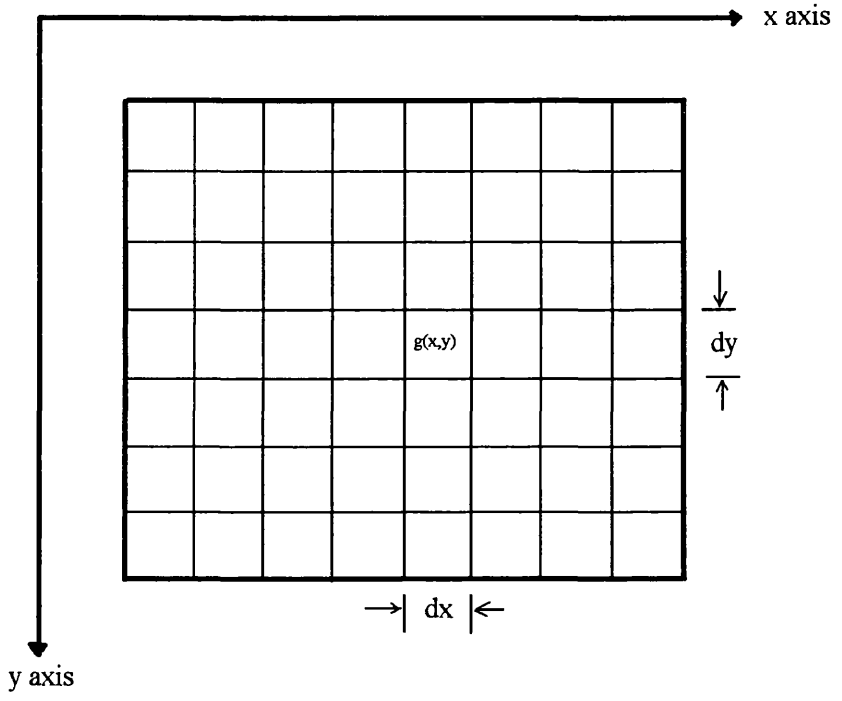

Figure 3.9 Pixel Coordinate Frame

The photo coordinate frame as shown in Figure 3.10, has its origin at the perspective point. The x-axis is usually taken to coincide with flight direction and the y-axis is perpendicular to the x-axis in the same horizontal plane. The principal point is the point on an image plane where an image would be formed if the image plane was perpendicular to a direct axial ray coming through the centre of the lens. The principal distance is the perpendicular distance

from the perspective point of the lens to the image plane. The fiducial centre is the point on an image plane where lines from opposite fiducial marks intersect.

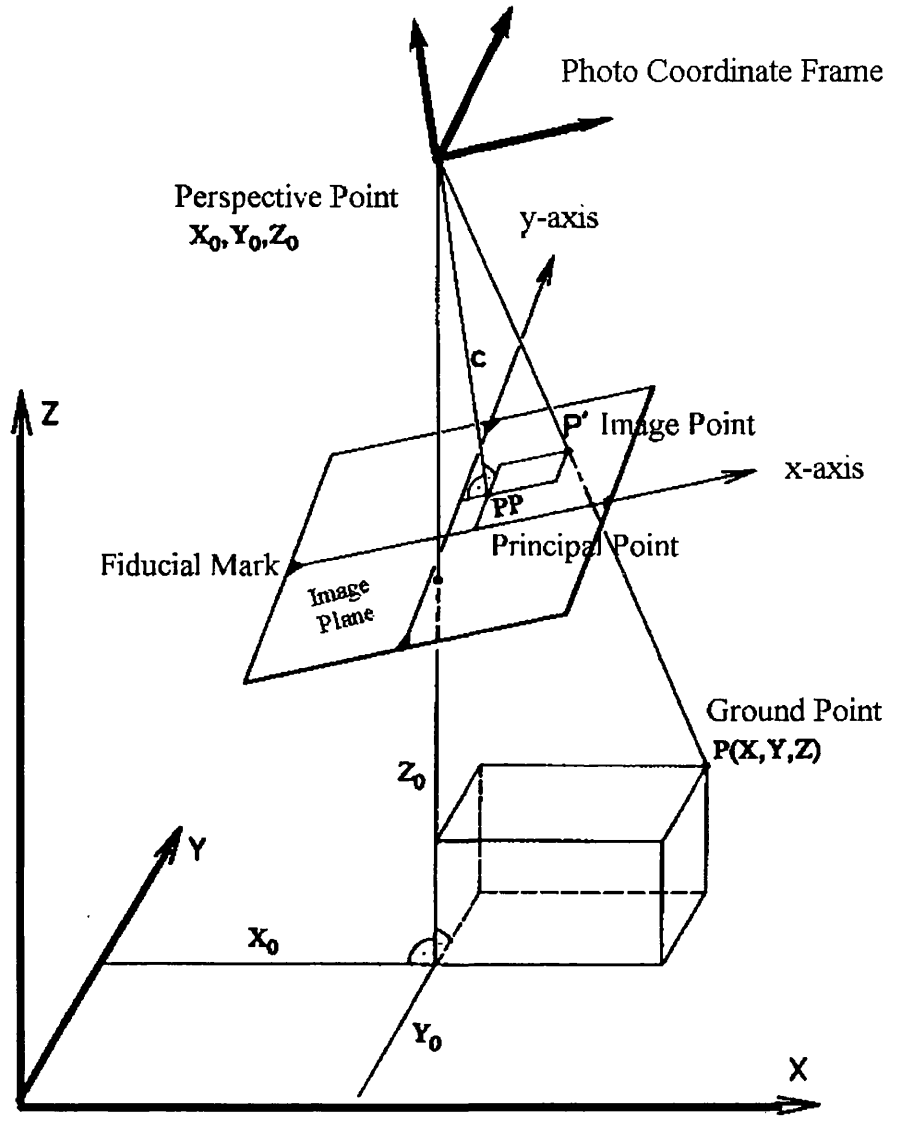

Figure 3.10 Photo Coordinate Frame

The following coordinate frames are mostly used in the computation of rotation and position from IMU observations [Schwarz et al., 1994].

The inertial coordinate frame, or i-frame, is a frame which does not rotate or accelerate (Figure 3.11). The origin is at the mass centre of the earth; the x-axis points towards the mean vernal equinox; the z-axis points towards the north celestial pole; and, the y-axis completes a right-handed system. The International Earth Rotation Service (IERS) continually establishes the inertial system through geodetic observations of quasars.

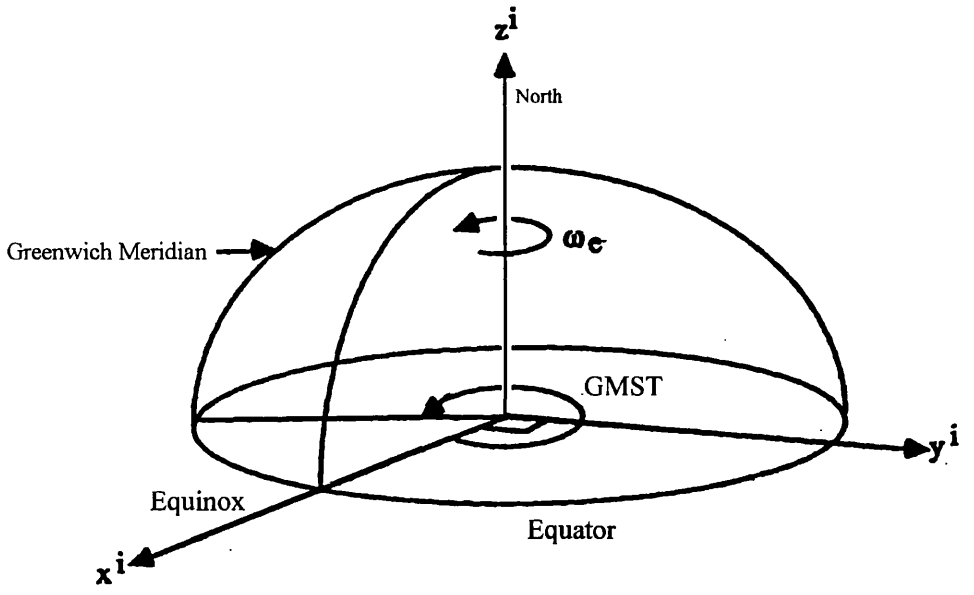

Figure 3.11 Inertial Coordinate Frame

The Earth-Centred-Earth-Fixed (ECEF) coordinate frame, or e-frame, also has its origin at the mass centre of the earth as in i-frame, as shown in Figure 3.12. The x-axis points towards the Greenwich meridian; the y-axis makes a 90^0 with the x-axis in the equatorial plane; and, the z-axis is taken as the axis of rotation of the reference ellipsoid. The e-frame is also the frame that is used by GPS.

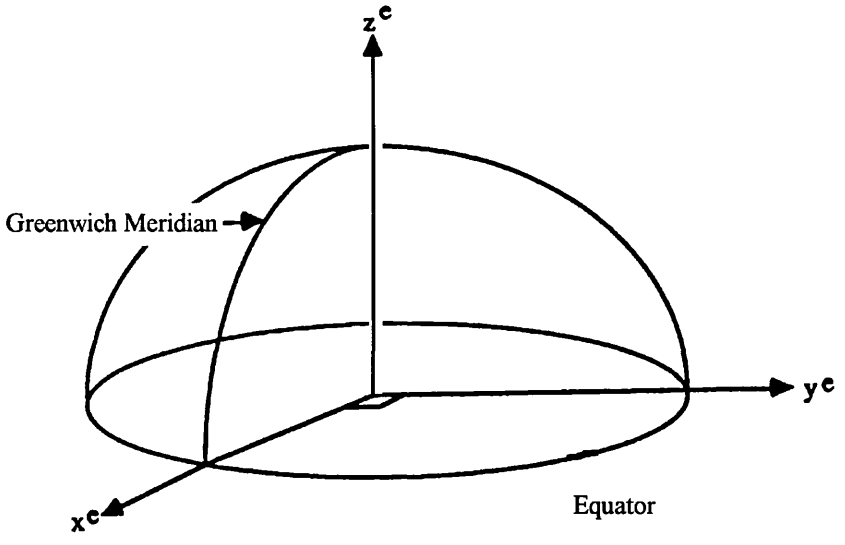

Figure 3.12 Earth-Centred-Earth-Fixed (ECEF) Coordinate Frame

The local coordinate frame, or l-frame is shown in Figure 3.13. It has its origin at the topocentre; the y-axis or the North axis points north, tangent to the geodetic meridian; the x-

axis or the East axis points to the East; and, the z-axis or the Up axis points upwards along the ellipsoidal normal.

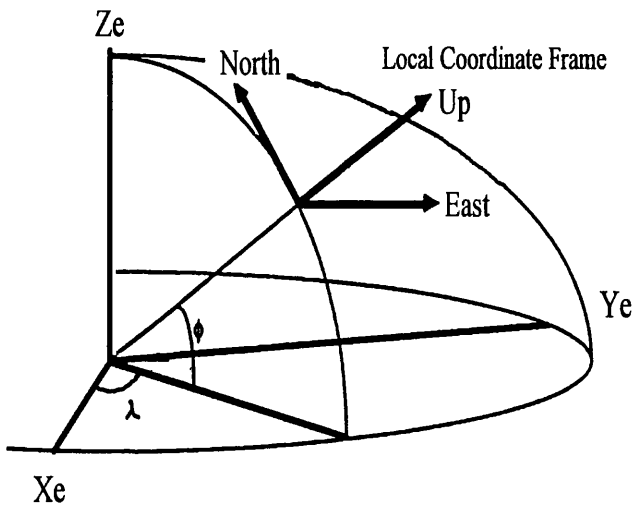

Figure 3.13 Local Coordinate Frame

The wander coordinate frame, or w-frame, is used in geographic areas of high latitudes to complement the l-frame (Figure 3.14). The l-frame needs always to orient itself to the north, which involves large rotations in areas of high latitude. The wander coordinate frame resolves this by setting the y-axis at an arbitrary direction. The angle between the y-axis and the north axis is called the wander angle.

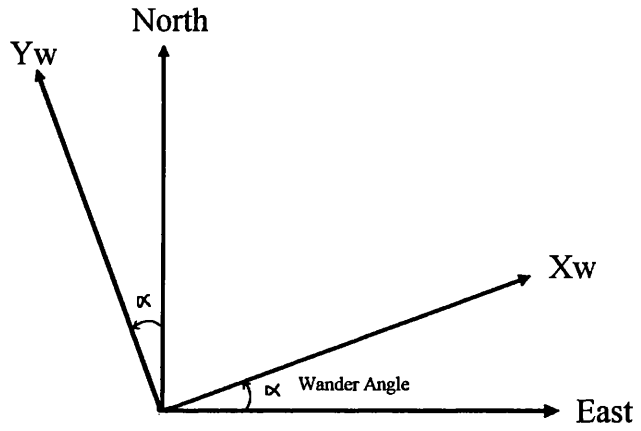

Figure 3.14 Wander Coordinate Frame

The body coordinate frame, or b-frame, is shown in Figure 3.15. It is the coordinate frame to which the IMU observations are referenced. It has its origin at the centre of the IMU, the exact location of which should be given by a specification of the manufacturer. This position is later needed when offset distances between different sensors are surveyed. The x-axis points towards the right side of the IMU; the y-axis points towards the front of the IMU; and the z-axis points upwards.

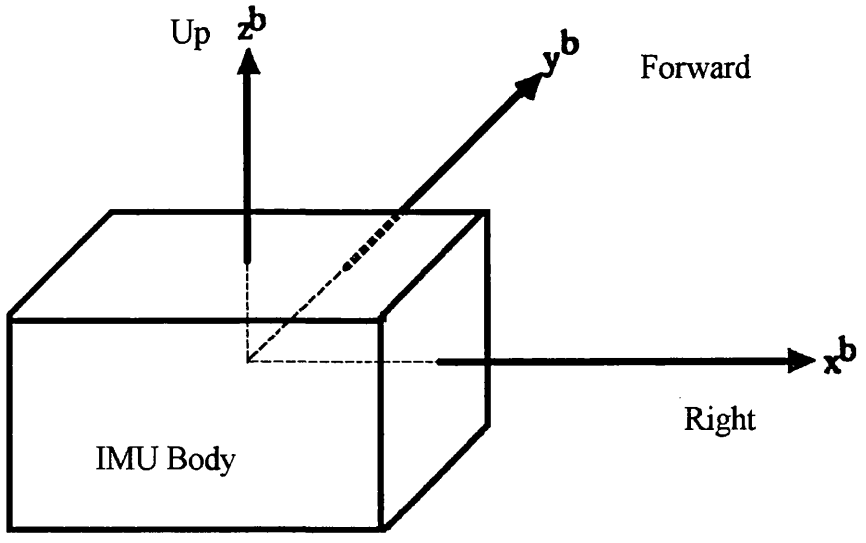

Figure 3.15 Body Coordinate Frame

3.5 Class Diagrams of the Imaging Subsystem

In this section the empty classes identified in the previous sections will be populated with the appropriate data members and member functions. This building of classes lays the foundation for the implementation stage, where objects based on these classes will be used to perform tasks such as reading data form files, the bundle adjustment and flight planning.

3.5.1 The CMatrixCamera Class

The matrix sensor class and its relation with other classes are described in Figure 3.16. It is derived from the base class CImageSensor which has a virtual function ReadDataFile(). The ReadDataFile() function will read data from files and populate the object with appropriate initial values.

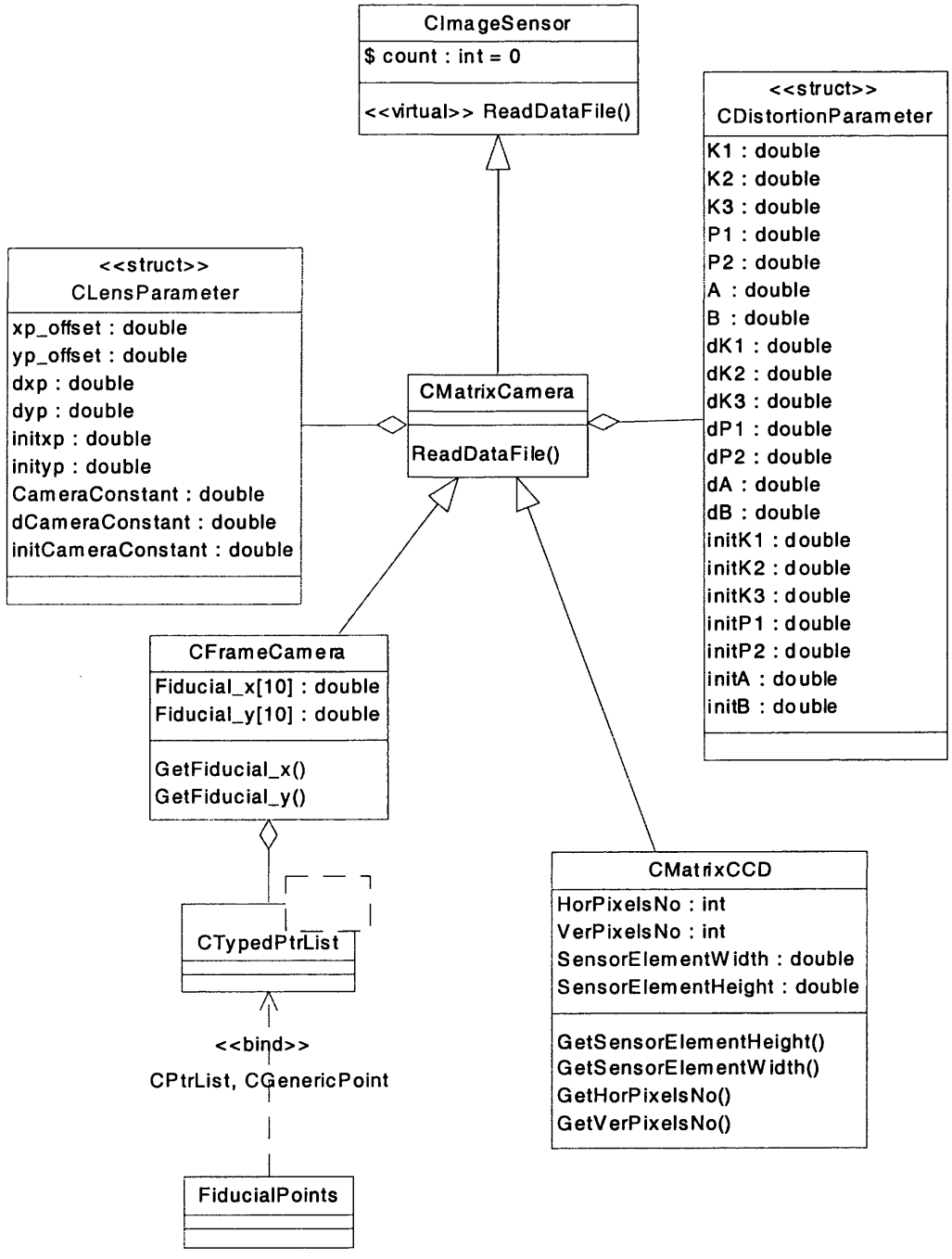

Figure 3.16 The CMatrixCamera Class

The CMatrixCamera class has two aggregated 'structs' ('struct' is a keyword used in C and C++ to define a collection of elements of arbitrary data types) as its data members, CLensParameter and the CDistortionParameters. The CLensParameter struct contains the values of the principal point offset (xp_offset and yp_offset) and the principal distance or the

camera constant(CameraConstant). Expected or known errors associated with these values are shown with a 'd' in front such as 'dxp' and 'dyp'. The initial values begin with 'init' such as 'initxp' and 'inityp'. The CDistortionParameter contains the lens distortion parameters (K1, K2, K3, P1 and P2) and the film shrinkage distortion parameters (A and B).

The frame camera and digital CCD matrix camera classes are derived from the CMatrixCamera class. The frame camera class, CFrameCamera, has fiducial points as its data members. The values of the coordinates of these fiducial points are those that are supplied by the manufacturer and are inherent to the camera. Their units are usually in millimetres. It uses a template class called CTypedPtrList which takes CPtrList and pointer to the objects as its arguments. CTypedPtrList is a template class provided by the MFC(Microsoft Foundation Class) library [Horton, 1998].

3.5.2 The CMappingImage Class

The CMappingImage Class is derived from C2DImage which is in turn derived from the CGenericImage class (refer to Figure 3.7). The following figure below, Figure 3.17, is centred on the CMapping Image which is the main image class to be used in the image acquisition subsystem.

The AffineParameter data member of the CMappingImage is used to contain the parameters of the affine transformation. These parameters transform pixel coordinates of the fiducial points of scanned images to photo coordinates. The FiducialImagePoints of the CMappingImage is used in conjunction with the FiducialPoints of the CFrameCamera to compute the affine transformation parameters. The CRotation class holds the primary, secondary and tertiary rotation angle of the axes which are usually known as omega, phi and kappa in photogrammetry. These are used to form the rotation matrix of the image. The CMatrix class is a very versatile class which will be extensively used in the adjustment processes to be followed after the objects are populated and appropriate matrices formed.

The CMappingImage also holds ImagePoints and Rays as its data members. Both of these are derived from the template class CTypedPtrList. The ImagePoints class is a list of pointers to objects of CImagePoint, and the Rays class is a list of pointers to CRay objects. CRay class itself is also a list of pointers to objects of CGenericPoint.

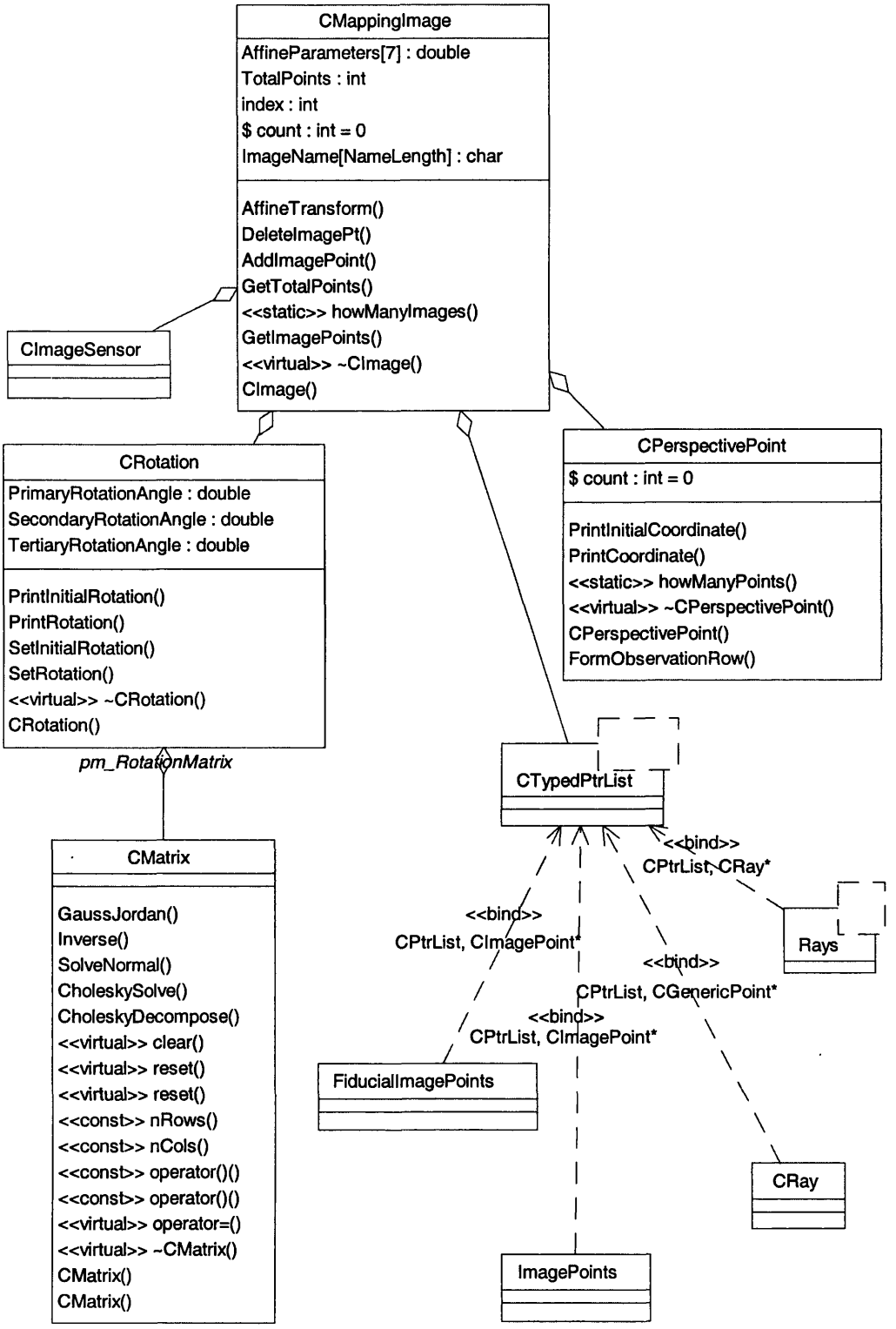

Figure 3.17 The CMappingImage Class

3.5.3 The Point Classes

The main classes of points used in the Image Acquisition subsystem are the perspective point, the control point and the image point. These classes are derived from the CGenericPoint. Most of the general functions of a point are defined in this base class (See Figure 3.18).

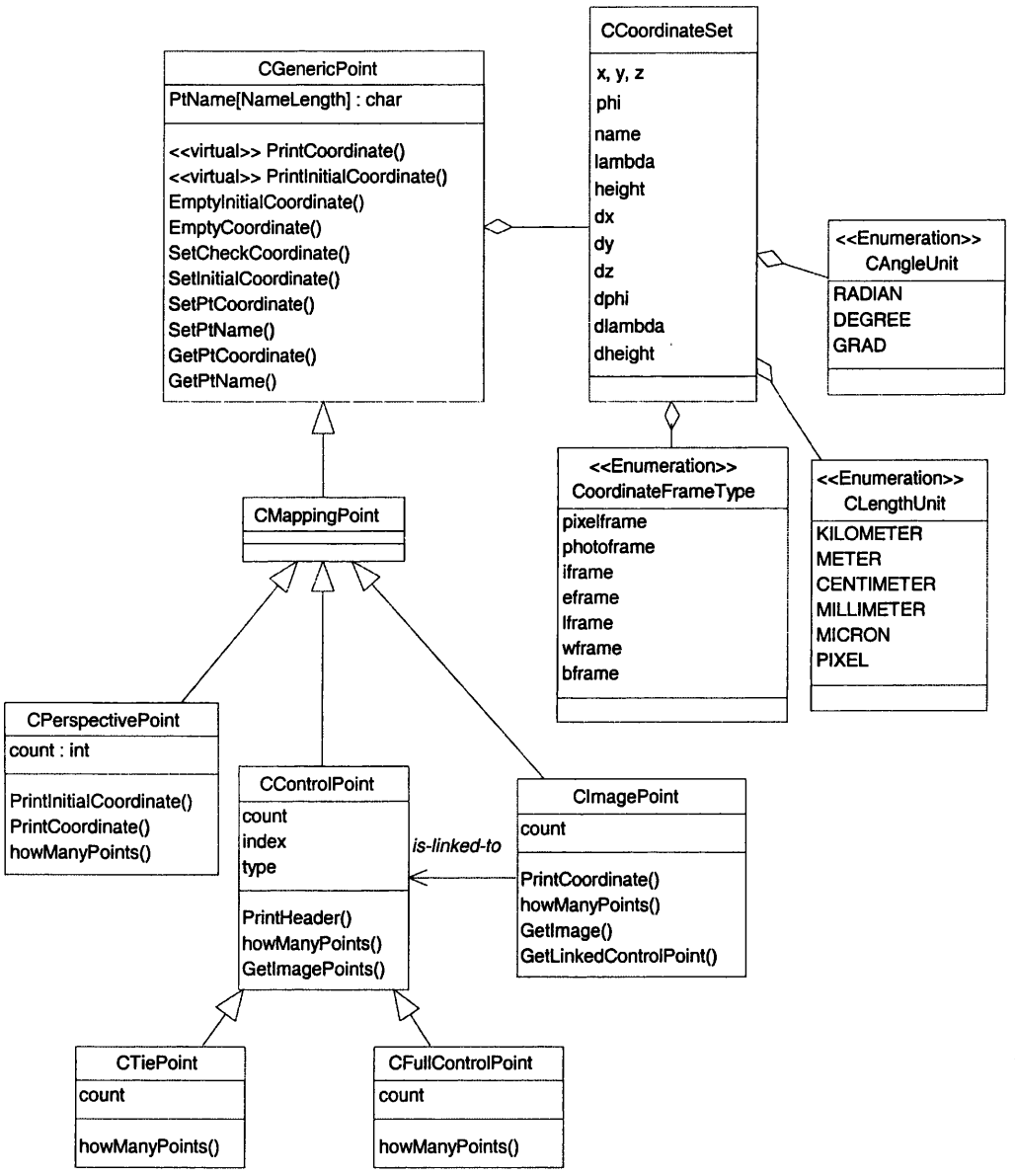

Figure 3.18 The Point Classes

Although not shown in the diagram, to avoid confusion, two coordinate sets are defined in the CGenericPoint class, an initial coordinate set and a working coordinate set. An initial coordinate set will remain unchanged throughout the adjustment whereas the working

coordinate set will be updated constantly. The reason for the many functions with the same name is because they are overloaded functions which will take different arguments. For example, SetCoordinate (x, y) will only update x and y coordinates, where as SetCoordinate (x, y, z) will update all x, y and z coordinates.

The coordinate set is an aggregated class of the point class. It has coordinate values of the Cartesian coordinate frame (x, y, z) as well as the geodetic coordinate frame (phi, lambda and height). It also has objects of the enumeration which will specify the coordinate frame, the length unit and the angle unit of the coordinates.

An image point is linked to a control point, forming a ray, and this can be accessed by the GetLinkedControlPoint() function. An image point also has the GetImage() function to return the image that contains it.

A control point may have a list of image points, and this can be accessed by the GetImagePoints() function. CTiePoint and CFullControlPoint are derived from the CControlPoint. The role of FormObservationRow() functions of these points are as explained in Chapter 2 regarding polymorphism.

3.6 Summary of Chapter 3

In this chapter, the processes involved in the acquisition of images have been analysed. Concepts used by domain experts (i.e. photographers and photogrammetrists) have been investigated. As the result of the investigation, a Use Case Diagram of the Image Acquisition Subsystem was produced (Figure 3.1). Some the important Use Cases of this subsystem that were identified include Calibrate Image Sensor, Control Imaging Sensor, Plan Flight, Monitor Flight, Log Image Data and Record Shutter Pulse. These Use Cases give a good overall view the various situations that this subsystem will be used for.

Based on this Use Case Diagram, further analysis for each Use Case was carried out. For essential Use Cases, corresponding Sequence Diagrams were produced to show the sequence of actions and messages that were exchanged between different objects. For example, the Sequence Diagram of Bundle Adjustment (Figure 3.2) illustrated how messages were passed between different objects of image, camera and ray of points to form various matrices which were necessary for the bundle adjustment.

These Sequence Diagrams were then used to identify important classes and their roles. Some of the most important classes, which are very likely to be reused in many applications of the

automated mapping system were designed in this chapter. They are the image sensor class, the image class and the point class.

The image sensor class was designed with the consideration that various types of image sensors exist today and that more will be produced in the future. To accommodate this, three types of image sensor classes were derived from the base class CImageSensor, namely, the CLinearCCDArray class, the CMatrixCamera class and the CActivePulseSensor class. The CMatrixCamera was then reclassified into CFrameCamera and the CMatrixCCD camera. Other new types of image sensors can be derived from the CImageSensor in the future.

The image class was also designed with the fact that the automated mapping system will have to deal with various types of images in the future. A very polymorphic CGenericImage is at the highest level of abstraction. It is the base class of all forms of images. This generic image is classified into C1DImage, C2DImage and C3DImage classes. The normal images that we use such as the aerial image would be an object of the C2DImage class, whereas the output data from the CActivePulseSensor would belong to the C1DImage class. A three dimensional perspective view generated from stereo images would be an example of the C3DImage class. However for this thesis, the CMappingImage class is fully developed. The CMappingImage is derived from the C2DImage class and contains data members and member functions to perform the camera calibration and the bundle adjustment.

The point class too has a CGenericPoint class which has CCoordinateSet as its data member. The CMappingPoint is derived from this CGenericPoint base class and is used to represent all points used for mapping purposes. CControlPoint, CPerspectivePoint and the CImagePoint classes are derived from the CMappingPoint. Some important member functions identified for the CImagePoint class are the GetLinkedControlPoint() function and the GetImage() function. These functions will be useful in forming the collinearity equations during bundle adjustment. Another point of detail design of the point class is the member function 'count'. It is a static data type, which means that it has a global nature and is not confined to the containing class. This global characteristic can be used to trace the number of points there are at any instant of computation. This can be implemented by increasing the 'count' variable by one when a new object of the point class is created and then reducing it by one when it is destroyed. This feature is also implemented in the CMappingImage class to trace the number of images created.

Usually there are many images and many points involved in mapping projects. The objects of these classes are managed through a pointer list class which holds pointers to the objects of a

class. The MFC (Microsoft Foundation Library) was used for this purpose. For example a list of image points can be declared as follows.

```
#include <afxtempl.h>

typedef CTypedPtrList <CPtrList,CImagePoint*> ImagePointList;
ImagePointList    MyImagePoints;
```

The first line includes the header file, which will enable access of the functions provided by the MFC. The second line defines ImagePointList as a list of pointers to the CImagePoint objects and the third line declares MyImagePoints as being a type of such an ImagePointList.

The classes designed in detail for the Image Acquisition Subsystem are summarised in the table below:

Table 3.1 Classes Designed for Image Acquisition Subsystem

Imaging sensor related classes	Image related classes	Point related classes
CImageSensor	CMappingImage	CGenericPoint
CMatrixCamera	CRotation	CImagePoint
CMatrixCCD	CRay	CPerspectivePoint
CFrameCamera		CContolPoint
CLensParameter		CTiePoint
CDistortionParameter		CFullControlPoint
		CCoordinateSet
		CCoordinateFrame
		CAngleUnit
		CLengthUnit

4. ANALYSIS AND DESIGN OF THE POSITIONING SUBSYSTEM

The main role of the positioning subsystem is to tag each image with the attitude, or the rotations, of the images and the position of the perspective centre of the image at the instant of image capture. For a laser scanner sensor, the positioning subsystem will compute the attitude and the position of the sensor for each pulse. The attitude of the images is obtained by using the data from the IMU. The position of the perspective centre of the image sensor is obtained by using GPS receivers. Kalman filtering and smoothing of data are applied in kinematic phase processing and also in GPS/IMU data integration, in order to get the optimal estimates of the unknown parameters as well as to accommodate measurement updates from different sensors.

In this chapter, GPS data processing methods and IMU data processing methods will be presented in two sections. Each of these two sections will begin with an overview of the basic theory and processing methods. Use Cases will then be discussed followed by Sequence Diagrams of these Use Cases. A third section will be devoted to an explanation of Kalman filtering.

As in Chapter 3, important objects will be identified initially and later filled with data members and member functions.

Static GPS data processing is used to establish the coordinates of the base station. These coordinates will be used in the kinematic data processing to process the phase data in the computation of the coordinates of the moving receiver.

IMU data processing begins with the alignment of the IMU. The raw data output of an IMU is a series of angular increments (of roll, pitch and azimuth) and acceleration increments with respect to the previous state. Therefore, to compute the position, velocity and the acceleration at a certain point of the trajectory, the data stream must be referenced to an initial state where the rotation angles, position and the velocity are known. The computation of this initial state is known as alignment. The alignment of a strapdown IMU (i.e. an IMU which allows its gyros and accelerometers to rotate with the body of the vehicle, because they are strapped to the vehicle) determines the initial Euler angles or the attitude matrix of the system from the IMU sensor output, approximate coordinates and the observed velocities. The alignment process is divided into two parts : the coarse and the fine alignment [Wong, 1988]. Fine alignment is the process of accurately determining the initial state using the Kalman filtering method and the known velocity. For a stationary state where the known

velocity is zero, this is known as zero velocity update. The coarse alignment computes the initial approximation values for the fine alignment process, using known values of the earth rotation rate and the magnitude of the gravity vector and approximate coordinates.

The term 'mechanisation' is frequently used by many geodesists and scientists who are involved in dealing with inertial navigation systems. Draper et al. [1960 (pp. 24)] uses it in the following context: "...problem is thus solvable by the mechanization of appropriate coordinate axes, that is, by the construction of physical objects which are designed to simulate Cartesian coordinate frames..." and Lapucha [1990 (pp. 34)] defines it as "Mechanization is the process of transforming the specific force measured by accelerometers and angular rates measured by gyros into the navigation parameters velocity, position and attitude.". Britting [1971] uses the term 'mechanization equations' throughout his book to refer to the equations involved in the processing of various types of IMU..

In this thesis, the established usage of the term 'mechanisation of IMU data' will be followed, to refer to the computation of the attitudes, position and the velocity in a selected coordinate frame using the raw data outputs of rotation and acceleration increments in a coordinate frame.

Kalman filtering is an estimation process used widely in kinematic positioning applications. The Kalman filter processes measurement data, usually in the form of digital outputs from sensors, using the linear system model and extra information regarding the model, namely:

- the external measurements of the parameters of the model;
- the statistical model of the system and the measurement errors; and,
- initial conditions.

The linear system model is a mathematical model where the observables are related to the parameters. Usually in GPS and IMU data processing, the system model is a dynamic model with position coordinates, velocity and acceleration as its state vector. State vector refers to the vector of parameters by which the system can be fully described.

If the state vector is estimated for some time in the future, the process is called prediction. If the estimate is made using the current measurement point, it is called filtering. If the estimate is made for a span of time using available measurements, as in post processing, the process is called smoothing. Further explanation about Kalman filtering will be given in section three of this chapter.

For Kalman filtering and other data processing in this study, post processing of data is assumed where the programs interact with the data files collected from the positioning sensors. Time synchronisation is also assumed and realised through an 'event file' which is a text data file containing:

- records of the image capture time;
- the GPS observation time; and
- the IMU observation time in the GPS time frame.

A Use Case Diagram (Figure 4.1) depicts the Positioning Subsystem. Its main functions are called Process GPS Data and Process IMU Data.

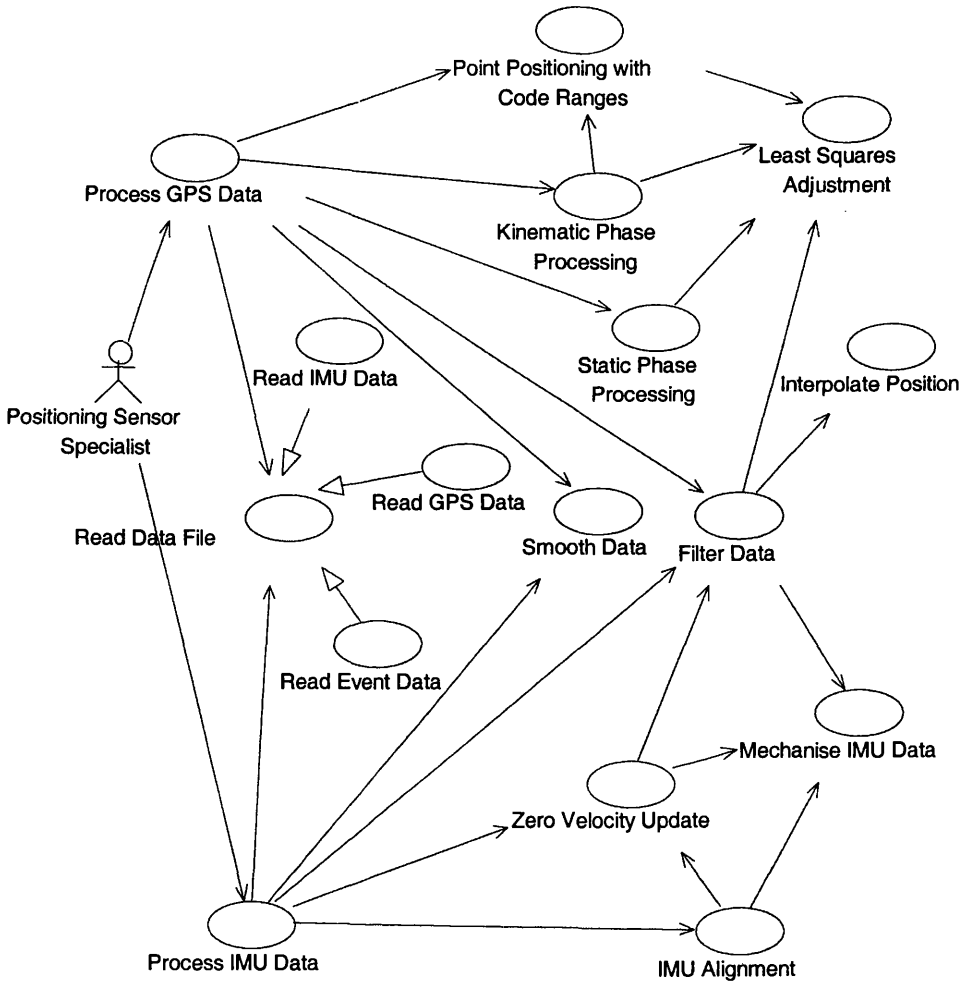

Figure 4.1 Use Case Diagram of the Positioning Subsystem

Process GPS Data uses Static Phase Processing and Kinematic Phase Processing which each in turn uses Least Squares Adjustment. The Kinematic Phase Processing also uses the Point Positioning with Code Ranges for its initial approximation. Filter Data and Smooth Data refer to the Kalman filtering of kinematic data processing.

Process IMU Data uses IMU Alignment and Zero Velocity Update as well as the Filter Data and the Smooth Data Use Cases. Mechanisation of IMU Data is used by the Zero Velocity Update, IMU Alignment and the Filter Data Use Cases.

4.1 Processing of GPS Data

GPS data can be processed in various ways depending on the application requirement. Various terms are used by GPS domain experts to refer to the various methods of data processing which handles different data types. Here are some phrases which might be confusing to someone who is newly exposed to the GPS technology [Hofmann-Wellenhof et al., 1994].

- "To a surveyor static surveying refers to static relative positioning by carrier phases"
- "Relative positioning usually refers to the case of using carrier phases observations, whereas in the case of using code observations the term 'differential' is usually used, as in 'Differential GPS'"
- "The accuracy of differential GPS or kinematic point positioning is at the meter level whereas the accuracy of kinematic relative positioning is at the centimeter level" [Hofmann-Wellenhof et al., 1994]

For consistency of terminology, only the terms described in Figure 4.2 will be used in this study to describe various aspects of a GPS survey. The term 'relative positioning' will be used to mean only the type of Positioning Scheme. It will not therefore automatically mean the usage of phase observations. Also the term 'differential' will not be used for referring to using code observations. The term 'differencing' or 'difference' will only be used at the computation level as in single difference, double difference or triple difference.

In Figure 4.2, all GPS survey projects are defined to have the following aspects:

- Positioning Scheme

  The Positioning Scheme can be Point Positioning or Relative Positioning.

- Observed Data Type

  The Observed Data Type can be Code Ranges or Phase Measurements

- Data Collection Mode and

  The Data Collection Mode can be Static or Kinematic.

- Data Processing Time.

  The Data Processing Time can be processed in Real Time or Post Processed.

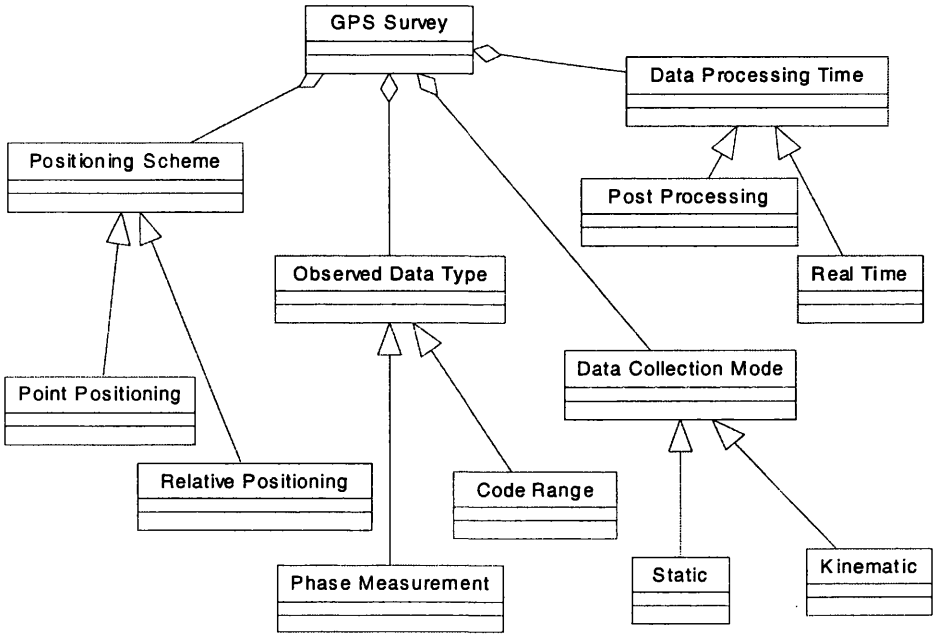

Figure 4.2 Abstraction of GPS Surveying

For example, the GPS observation for flight monitoring described in Chapter 3 would be a relative positioning scheme which collects code range data in a kinematic mode and which will be processed in real time. An actual project would consist of various combinations of the above classifications. For example, code range could also be used for initial approximation in the post processing of phase measurement data collected in the kinematic mode, and real time processing could be carried out as well as post processing.

The next section will describe GPS data processing and various objects involved in this process.

4.1.1 Analysis of GPS Data Processing

The GPS satellites or space vehicles (SV), which move along designated orbits, transmit signals continuously in a known format. The ephemeris data which describe orbits are also

transmitted as part of the signals. The position of the SV at the moment of signal transmission can be computed using this ephemeris data.

The important components of the signal which are used for positioning are the carrier component and the code component. The code component is the signal which has been modulated to the carrier signal.

These signals are captured by GPS receivers, in static or kinematic mode. The carrier components of the signals are characterised by their frequencies and wavelengths. The L1 carrier has a frequency of 1575.42 MHz. The wavelength of the L1 carrier which can be computed by dividing the frequency with the speed of light (299,792,458 metre/sec), is about 19 cm. The frequency of the L2 carrier is 1227.60 MHz and its wavelength is about 24.4 cm. Two kinds of codes signals are modulated to the carrier waves: C/A code and P code. The C/A code and P1 code are carried by the L1 carrier and the P2 code is in L2 carrier.

The GPS was designed so that civilian users could be denied full use of the system when this was deemed necessary for military purposes. This is achieved by two methods named Selective Availability (SA) and Anti-Spoofing (A-S). SA degrades the positioning accuracy of the C/A code by dithering the satellite clock and manipulating the ephemerides. A-S encrypts the P-code or makes it unavailable. The resulting encrypted P-code is called Y-Code.

The data produced by the GPS receiver, as a result of capturing the signals and the internal signal processing of the receiver, are the pseudoranges (or code ranges) and phase measurements. Code ranges are distances (in metres) between the receiver and the signal emitting SV. This is obtained by multiplying the speed of light by the measured travel time of the signal. Phase measurements are the fractional phase offset (in cycles) between the received carrier signals (i.e. L1 and L2) and the reference signal of the receiver.

Time data generated by various clocks play an important role in the processing of GPS data. The most accurate time is the one which is maintained by the GPS control segment (GPS Time). SVs also have oscillators which maintain their own time frame (SV Time). The receiver has a clock too, which records the time the signal is received (Receiver Time).

The captured code ranges and the phase measurements at the receiver are all related to a time frame. To simplify computation regarding time, for example to check if two observations are synchronous, the civilian representation of time as year, month, day, hour, minute and seconds is transformed to one number representing seconds of the GPS Week.

Errors due to time delay caused by the refraction of the signal in the troposphere and the ionosphere are also computed to achieve high positional accuracy.

The next figure (Figure 4.3) shows the relationship between SV, signal measurement and orbit. The CClockTime class is a class which deals with the transformation of civilian time format to seconds of GPS week. The CPosition class is defined as an object with a coordinate and time. Therefore it has an object of CMappingPoint (refer to the section on Point class in Chapter 3) as one of its data members as well as an object of the CClockTime class.

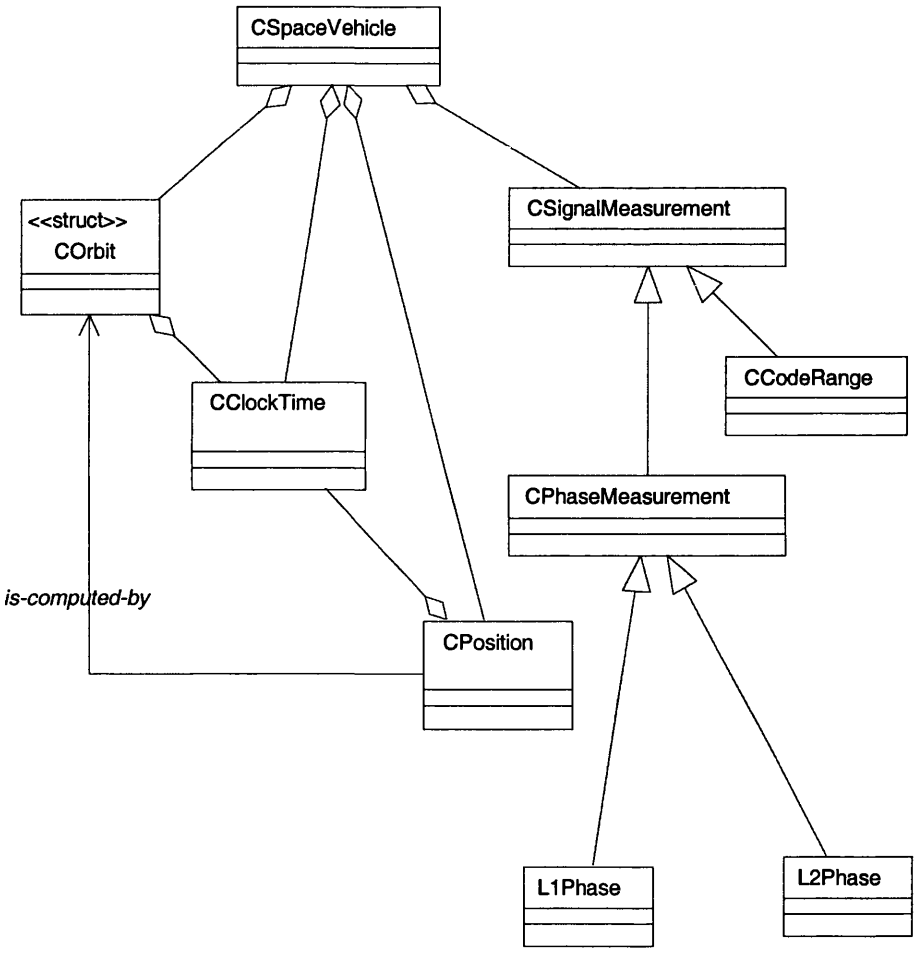

Figure 4.3 Space Vehicle and Related Classes

A GPS observation is defined as a record of all the measured signals from all visible SVs at a moment in time. It should be noted that for each epoch observation, the number of instances

of the CSpaceVehicle is equal to the number of visible GPS satellites at the time and place of observation. Therefore each GPS epoch observation has a list of objects of the CSpaceVehicle class and each CSpaceVehicle object in turn has a CSignalMeasurement which can be the code measurement, the phase measurement or both of these measurements. The phase measurement can also be either L1 measurement, L2 measurement or both phase measurements. The measured signals are contained in the CSignalMeasurement class of Figure 4.3. The CSpaceVehicle class and the CSignalMeasurement is connected by the diamond-head line indicating the aggregation relationship.

The interval for these observations is preset before the survey, for example to collect a record after every second or after every five seconds. For relative positioning the interval must be set to be equal for the relative and the base station, so that each record of a relative station will have a matching observation record of the same epoch at the base station. For each GPS survey, each receiver collects these observations in a single data file, known as an observation file.The ephemeris data are collected as a separate navigation file and a meteorological file is also produced as a separate file to provide information on atmospheric data such as pressure, temperature and humidity which are used for correcting errors due to these influences. Figure 4.4 shows the class diagram of the observation class.

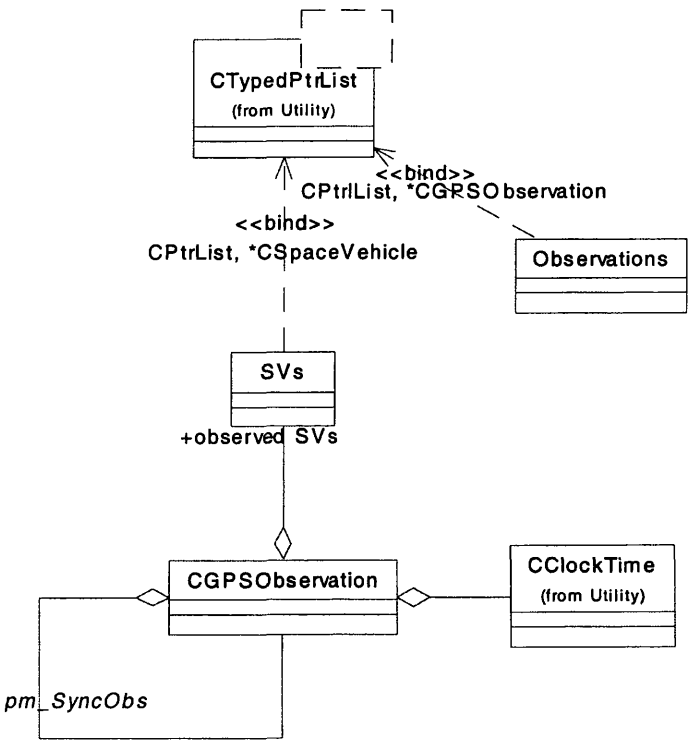

Figure 4.4 Class Diagram of CGPSObservation

The "Observations" class is shown as a list which is produced as a result of binding the argument CPtrList class and the pointer of the CGPSObservation class, *CGPSObservation. The CGPSObservation class has an object of the CClockTime to indicate the observation time. The CGPSObservation class also has a list of CSpaceVehicle objects as its data member, i.e. SVs. SVs are the GPS satellites for which signals have been recorded in this single observation. An object named pm_SyncObs, is another CGPSObservation data member which is the synchronised pair of this observation. The aggregation line which points to itself indicates that the CGPSObservation class has another object of CGPSObservation as its data member.

The signals are captured by an antenna connected to the receiver and the data files are saved in a computer disk connected to the receiver. The actual position computed using the GPS signals would be the phase centre of the antenna. However, the position of a ground marker or a point in space, such as the perspective centre, is usually the object of interest rather than the position of the antenna or the receiver. A new class called CStation is created to refer to this point of interest related to the receiver (Figure 4.5). Although in its strictest sense it is accurate to put the point coordinates in the antenna class because the initially processed coordinates refer to this point, the term 'station' is commonly employed by domain users to refer to the point of interest where the receiver has been set up. CStation is defined as the point of interest which holds the receiver and much of the role of the receiver is delegated to the CStation class. The static reference station, named CReferenceStation class, has a position object whereas the kinematic station, named CRoverStation class, has a list of position objects which depicts the trajectory of its receiver. Both of these classes are derived from the CStation class. For multi-antenna systems, though not treated in this thesis, it would be possible to have two sets of CPosition lists declared in the CRoverStaton class and name them for example as EpochPositions1 and EpochPositions2.

Events which occur during the observation period, such as shutter release, cycle slip or any other external events other than the signal recording, are kept recorded in the CEvent class.

The computation is done mostly in the e-frame. However the antenna height compensation and the presentation of most end results in surveying requires transformation to the l-frame. Transformations to and from the e-frame to geodetic coordinates of latitude, longitude and height are also necessary in many parts of the processing. These functions should be added to the CMappingPoint class.

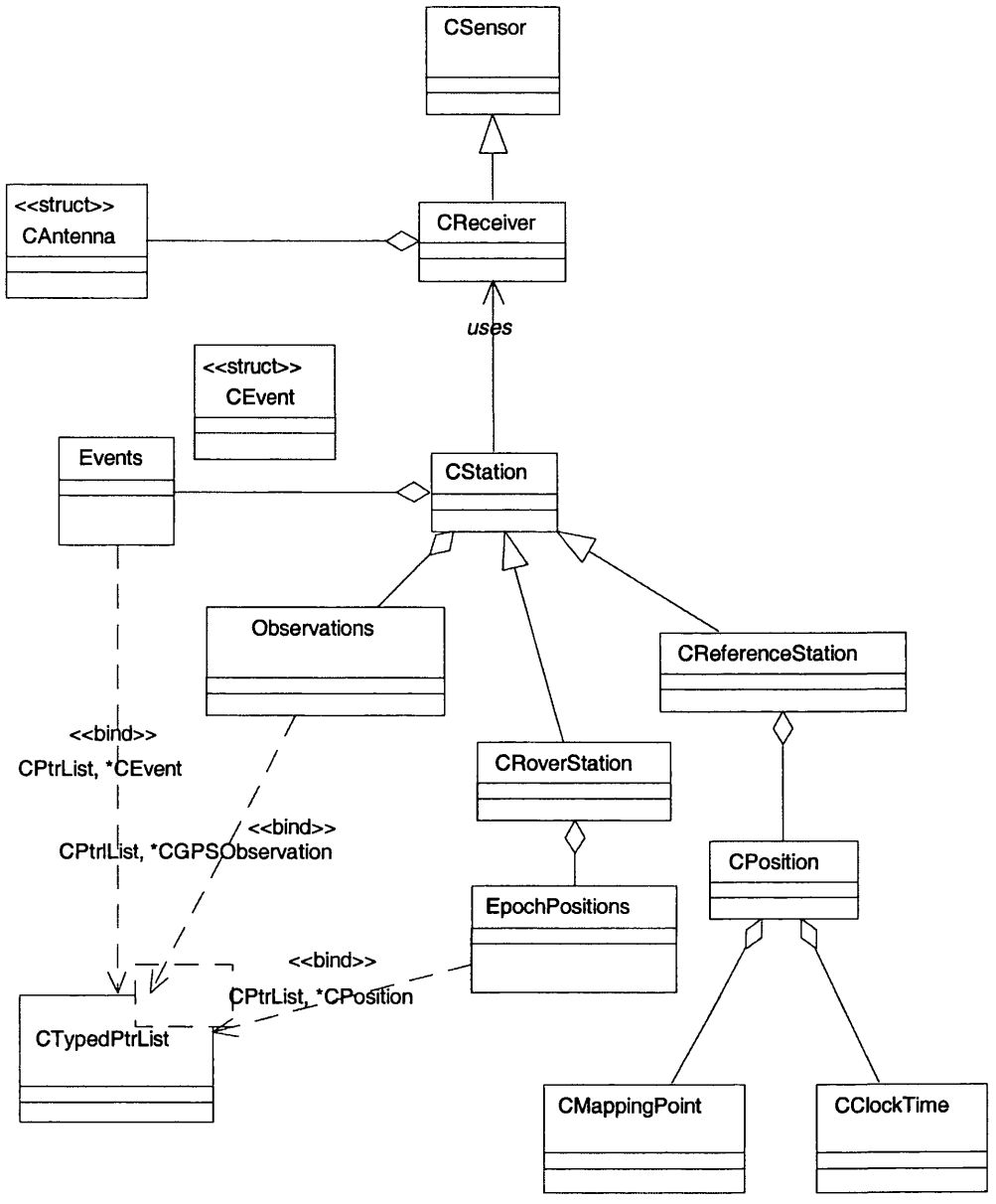

Figure 4.5 Class Diagram of CStation

Code Range Data and Phase Measurement Data

The main differences between the code range data and the phase measurement data are the level of accuracy and the processing method. In general, the accuracy of code ranges is at the metre level, whereas the accuracy of phases measurements is at the millimetre level. [Hofmann-Wellenhof et al., 1994] . The processing method differs greatly because of the different types of unknown parameters involved in the mathematical models. In the

following, the basic equations used in GPS data processing will be introduced. All basic equations for GPS data processing in this thesis have been referenced from the book by Hofmann-Wellenhof et al. [1994].

The observation equation for the code range point positioning is given as follows.

$$R_i^j(t) = \rho_i^j(t) + c\left(\delta^j(t) - \delta_i(t)\right) \qquad (4\text{-}1)$$

where, $R_i^j(t)$ is the measured code range between station i and SV j at an epoch t, $\rho_i^j(t)$ is the geometric distance between the station and the SV at an epoch t, δ_i and δ^j are the clock errors of the receiver clock and the SV clock respectively, and c is the speed of light. Also, the geometric distance can be expressed as follows.

$$\rho_i^j(t) = \sqrt{\left(X^j(t) - X_i(t)\right)^2 + \left(Y^j(t) - Y_i(t)\right)^2 + \left(Z^j(t) - Z_i(t)\right)^2} \qquad (4\text{-}2)$$

where, X, Y and Z with subscript i are the coordinates of the station i in the e-frame and X,Y and Z with superscript j are the coordinates of the SV j also in the e-frame. The SV position coordinates and the SV clock error can be computed directly from the transmitted data. Therefore the unknown parameters are the X,Y,Z coordinates of the station position and the receiver clock error, δ_i. If the number of SVs that are observed simultaneously is more than 4, the station coordinates can be computed with only a single epoch observation. Therefore real-time processing of code ranges is relatively simple.

For the case of phase measurement, the mathematical model is given by the following equation.

$$\Phi_i^j(t) = \frac{1}{\lambda}\rho_i^j(t) + N_i^j + f^j \Delta\delta_i^j(t) \qquad (4\text{-}3)$$

where, $\Phi_i^j(t)$ is the measured phase at epoch t, λ is the wavelength of the carrier signal, N_i^j is an unknown integer number, known as integer ambiguity, representing the initial number of cycles between the SV and the station when the receiver first locked on to the SV signal, f^j is the frequency of the SV signal and $\Delta\delta_i^j(t)$ is the term representing both the receiver and the SV clock error. The integer ambiguity introduces many complexities to the solution of the phase observations. Because it is constant in time it needs to be solved only once initially in static mode. In static mode the number of unknown parameter is 5 for each

phase observation equation. With each SV however a new integer ambiguity is introduced. For example in static mode, if 4 SVs are tracked for 5 epochs, the total number of unknown parameters is 12 (i.e. 3 station coordinates, 4 integer ambiguities and 5 receiver clock errors). The number of observation equations is 20 (4 observation equations for each of the 5 epochs). Therefore in this case the redundant observations can be solved by the least squares method to get the station coordinates. It can be seen from the above example that with a short period of initial static processing to solve for the integer ambiguity (known as initialisation), phase measurement data can also be used for real time processing in kinematic mode. This implies station positional accuracies reaching centimetre level, compared to the metre level when using code range data. Unfortunately in actual circumstances, it is unlikely that the integer ambiguities of each SV remain constant throughout the data collecting period, especially in urban areas or when the receiver is in kinematic mode. When the signals are interrupted or blocked, cycle slips occur and this brings changes to the integer ambiguities. This requires repeating the initialisation process. However, through much recent research into integer ambiguities [Schwarz et al., 1989][Hatch, 1990][Abidin, 1993] it is now possible to resolve the ambiguity while in kinematic mode. This technique is known as 'On-The-Fly' (OTF) ambiguity resolution. This will be discussed in detail in the later section on kinematic phase processing.

Forming Difference Equations in Relative Positioning Scheme

Relative positioning uses two or more receivers to get synchronous epoch data in order to increase the accuracy of the computation. The observation equations formed in data processing as the result of relative positioning are difference equations.

In relative positioning one receiver is placed on a known point and the synchronous data from the unknown and the known stations are differenced. Because they are synchronous the first difference of observations (single difference) between two stations for the same SV, removes the satellite clock error. A further difference between two single differenced observations for the same station vector (double difference) will remove the receiver clock error. Another difference of the double differenced observations between two different epochs (triple difference) will remove the integer ambiguity. The most commonly used double difference equation is as follows.

$$\lambda\Phi_{AB}^{jk}(t) = \rho_{AB}^{jk}(t) + \lambda N_{AB}^{jk} \qquad (4\text{-}4)$$

where, λ is the wavelength of the carrier phase, $\Phi_{AB}^{jk}(t)$ is the double difference phase observation for SV j,k observed from stations A and B at epoch t. $\rho_{AB}^{jk}(t)$ and N_{AB}^{jk} are the double difference range and integer ambiguity for SV j,k observed from stations A and B at epoch t.

It should be noted that when performing least squares adjustment with differenced observation equations, correlation between parameters should be considered and this should be used to form weight matrices in the actual adjustment.

Reading GPS RINEX Data File

In post processing, the observation data file collected by the receiver is read by the GPS data processing program for subsequent processing. Some sort of format for the data file must be fixed for this to be possible.

RINEX is an acronym for Receiver Independent Exchange Format. It has been developed by the Astronomical Institute of the University of Berne for the easy exchange of the GPS data between GPS receivers of different manufacturers [Gurtner et al.,1990].

Four different types of ASCII data files are defined in the RINEX format: Observation Data File; Navigation Message; Meteorological Data File; and, the GLONASS Navigation Message File. Each file type consists of a header section and a data section. The header section contains global information for the entire file and the data section contains the actual observation values. Each file is a set of data collected from one site and one data collection session. The following analysis will deal only with the reading of navigation and observation data files.

Figure 4.6 show the Sequence Diagram of reading a RINEX navigation data file. During post processing, when the operator inputs the file name the data file is read header first. An Almanac object is created and the data are read in from the data file. These almanac data will be added to the orbit as a data member to the orbit later, after the orbit data have been read from the file. The orbit data which follows the header will be read into the orbit object and this will be added to a list of orbits. Later the orbit data of an SV which is the closest in time to the observation epoch will be selected and used to compute the SV position.

The observation data header contains information about the survey as a whole. It contains data regarding the station such as the station name and number, the observer and the agency,

the type of receiver and antenna used. There are also lines for antenna eccentricities and approximate position of the station in the observation data header.

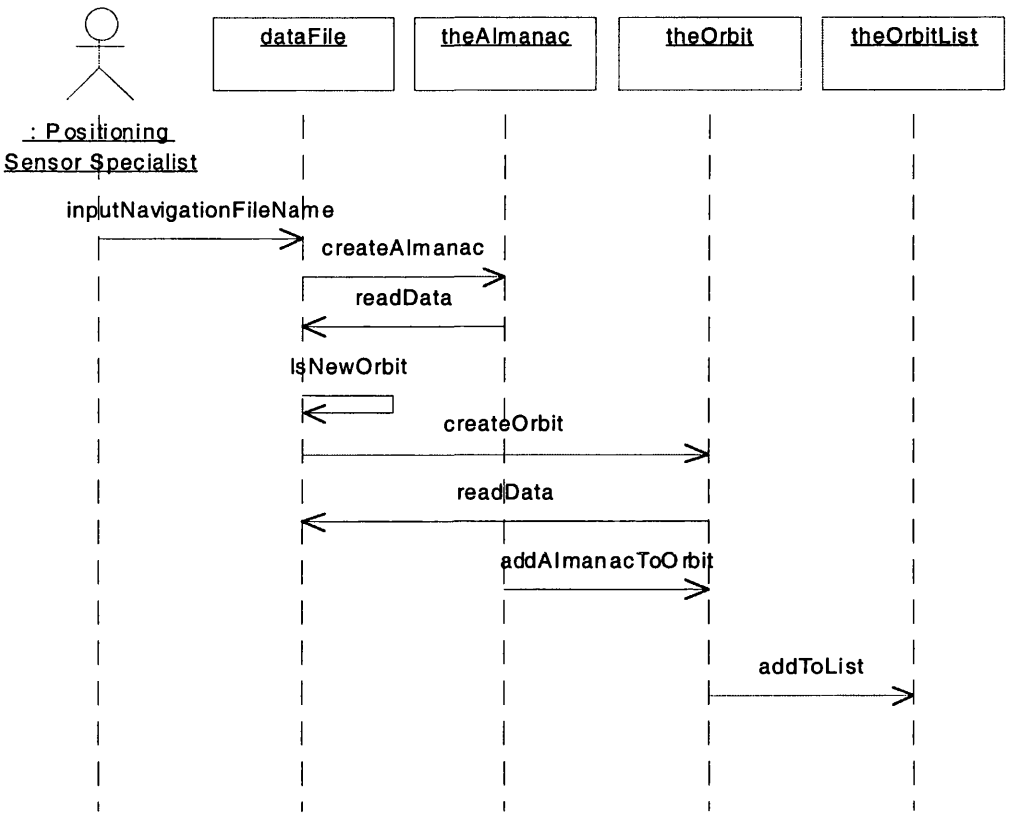

Figure 4.6 Sequence Diagram of Read Navigation Data

Types of observation (i.e. L1, L2, C1, P1, P2, D1, D2...) are listed and the interval of observation, time of first and last observation are also in the observation data header. As each line of the data header is processed, each relevant object (i.e. CStation, CReceiver, CAntenna, CPosition, CClockTime) is created and populated with the data from the header part of the observation file (Figure 4.7). An empty EventList is created which will be filled with events such as shutter release event or a cycle slip event as they are encountered in the processing of the GPS observations.

What follows after the header are records of GPS observations (Figure 4.8). Each GPS observation has a list of SVs and each SV has a list of values each representing pseudoranges in metres, phase in cycles and Doppler measurements in Hertz. This list of values is abstracted as an object of the signal measurement class. For each SV an orbit closest in time is selected from the orbit list and added to the SV as its data member. This orbit list has been created from the navigation data file. The data relevant to the signal measurement is read

from the data file and then the signal measurement is added to the SV as its data member. Finally the SV is added to the list of SVs, which is a data member of the CGPSObservation. This process is carried out with each GPS observation record.

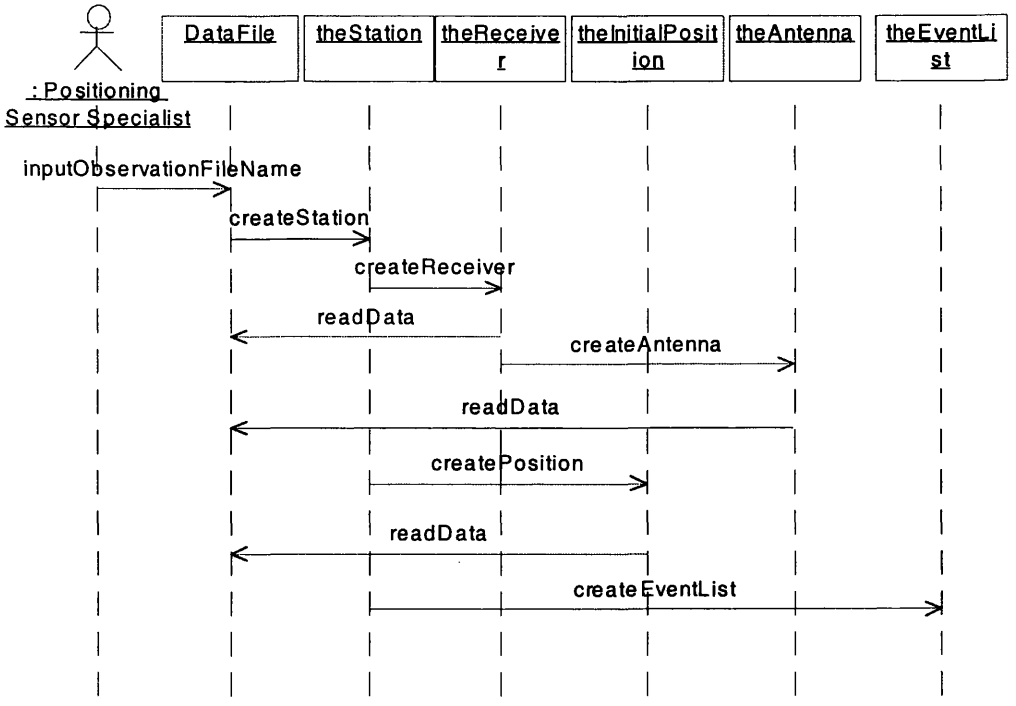

Figure 4.7 Sequence Diagram of Read Observation Data Header

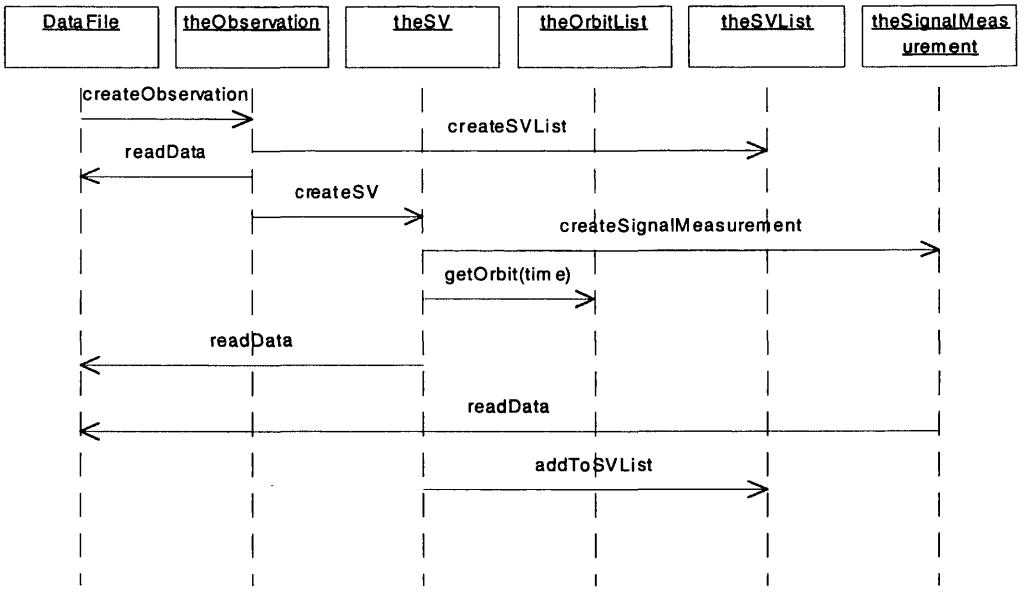

Figure 4.8 Sequence Diagram of Read Epoch Observation

Point Positioning with Code Ranges

Point positioning with code ranges refers to the computation of the coordinates of the station using the code ranges of a single GPS observation. Therefore it would suitable for this function to be a member of the CGPSObservation class, and to call it, for example, ComputeSingleCodePos. The function would take an initial approximation for the unknown parameters of the station coordinates (x, y, z) and the receiver clock error (dt). Then the position could easily be computed, for example by using the following C++ code.

```
CPosition ApproximatePosition;
CGPSObservation RovObs;
ApproximatePosition = RovObs.ComputeSingleCodePos(0.,0.,0.,0.);
```

After the execution of this command line, the object 'ApproximatePosition' has the computed values of the station coordinates. In the example code above, all the initial approximations of the station coordinates and the receiver clock offset are set to zero.

The process of point positioning with code ranges is examined in the Sequence Diagram as shown in Figure 4.9. The SVs in the SV list are accessed consecutively and the position for each SV is computed. The travel time of the signal and the correct SV time are computed prior to the computation of the SV position. Then for each SV the linearised version of observation equation 4-1 is formed. The least squares adjustment is then carried out with the final set of observation equations to get the station coordinates.

Kinematic Phase Processing

In this section post processing of kinematic phase data using the relative positioning scheme is described. Depending on the initialisation techniques carried out to resolve the integer ambiguity there are various processing strategies. This section will deal with single epoch ambiguity resolution [Corbett, 1994]. The actual ambiguity resolution is performed by a least squares ambiguity searching method introduced by Hatch [Hatch, 1989][Hatch, 1990]. The advantage of the single epoch ambiguity resolution is its insensitivity to cycle slips because ambiguity is fixed for every observation. This also means that errors due to incorrect ambiguity resolution are isolated to the single epoch.

However the disadvantage is the requirement for an increased number of visible SVs to achieve a high redundancy in observations for increased reliability.

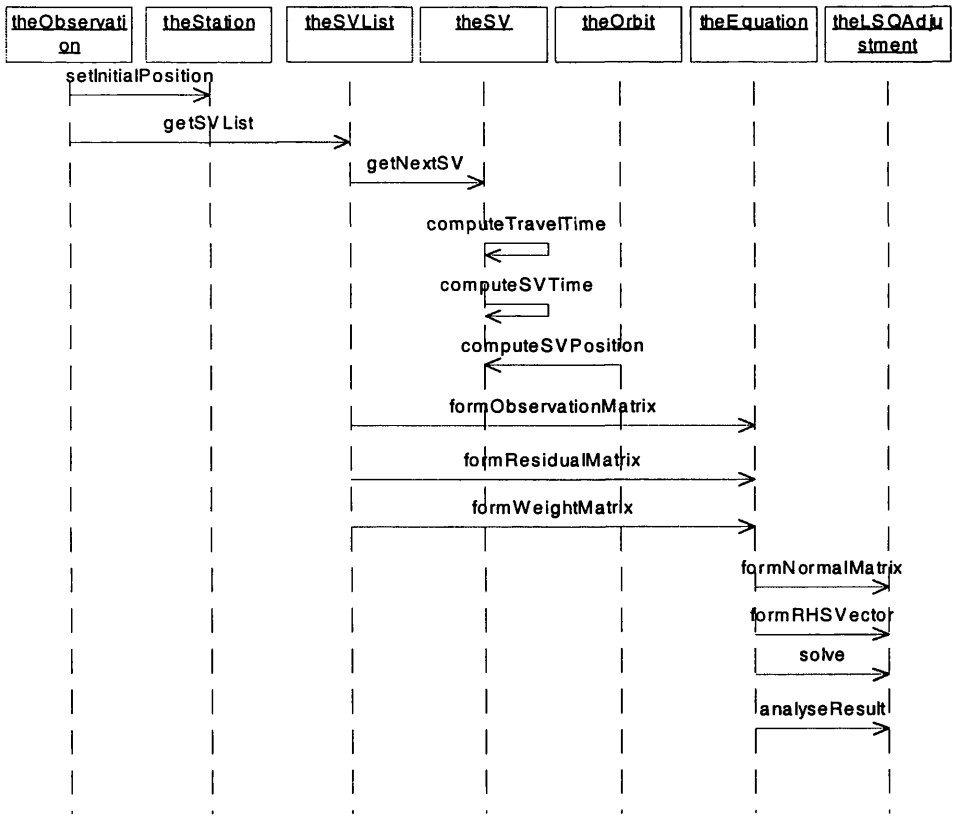

Figure 4.9 Sequence Diagram of Point Positioning with Code Ranges

For a relative positioning scheme, double difference equations between synchronised observations must be formed. The formation of double difference observations begins by the synchronisation of the two observation data files, where the file pointers of both files are set to coincide with the earliest observations of either the reference station data file or the rover station data file. Then approximate positions are computed using a point positioning scheme with code ranges. At this point the rover observation is included as a data member of the reference observation. See Figure 4.10. The Sequence Diagram of Figure 4.10 shows the sequence of actions and the interaction of objects in the formation of double differences from two synchronised phase observations.

To form the double difference equations, SVs present in both the reference and the rover observations are selected. SVs below the valid elevation angles are filtered out. The remaining SVs are added to a list of SVs called SynchronousSVs. A reference SV is selected from the SynchronousSVs and then single differences are formed by differencing the phase values of the reference observation and the rover observation. Finally double differences are formed by differencing the single differenced phase values from the single difference with

the reference SV. These double differences are added to the list of double difference objects called DoubleDiffs, a data member of CGPSObservation.

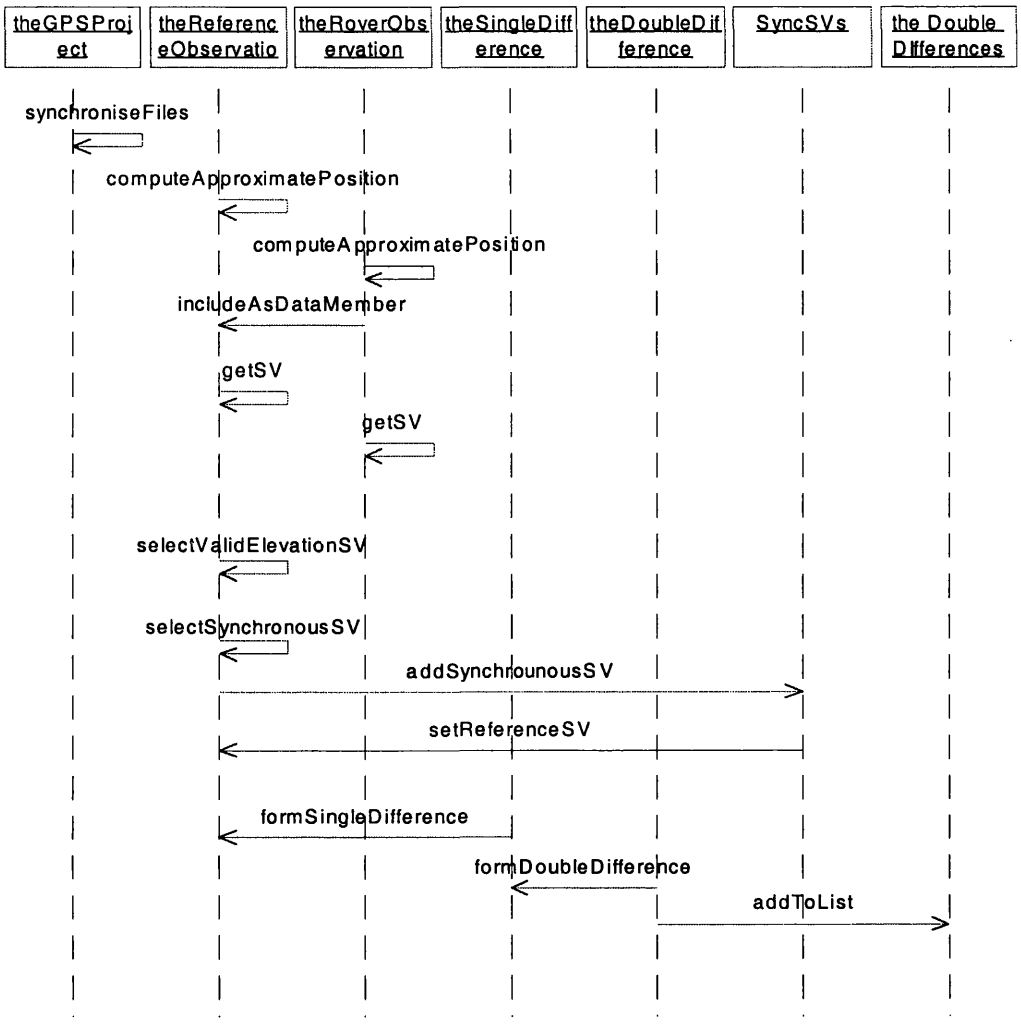

Figure 4.10 Sequence Diagram of Form Double Difference

Having formed the double differences, it is now possible to attempt to resolve the integer ambiguities. An overview of the integer ambiguity resolution is shown in Figure 4.11. Primary double differences are selected from the double differences. Primary double differences are double differences which are used to compute a trial position with a candidate integer ambiguity set. The rest of the double differences are called secondary double differences. The reason for adopting the concept of primary and secondary double differences is to speed up the computation time by reducing the search space. Only three

double differences are selected as primary double difference, regardless of the total number of double differences.

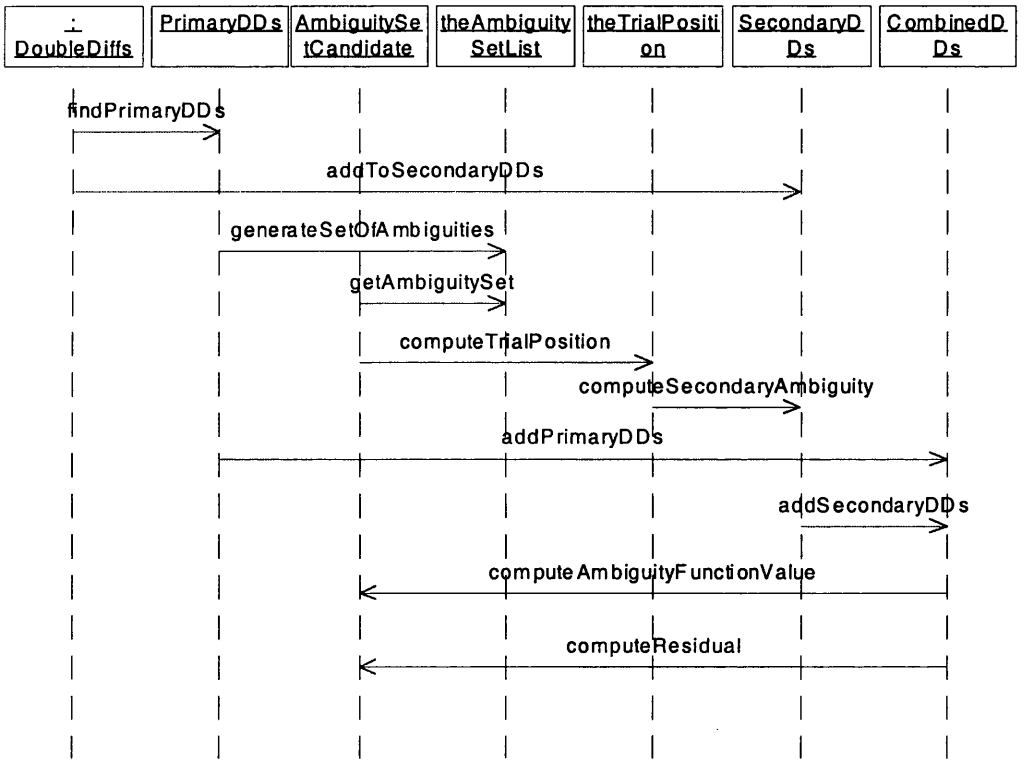

Figure 4.11 Sequence Diagram of an Overview of Resolving the Integer Ambiguities

If the range of the integer ambiguity search space was for example ± 10 cycles, the total integer ambiguity combinations to be tested would be 21^3. The trial positions computed from the candidate integer ambiguity set is then used to compute the integer ambiguities of the secondary double differences. Each double difference now has an integer ambiguity associated with itself. These double differences are added to a new list double differences, called CombinedDDs.

A technique to further reduce the computation time is introduced by computing the Ambiguity Function Value of the candidate integer ambiguity set. The Ambiguity Function Value can be used to reject a candidate integer ambiguity set if it is below a selected criteria. Only those sets which get through this initial test will go through to the next more rigorous test.

The final test is a least squares adjustment of the double differenced observation equations formed with the double differences of the CombinedDDs. The candidate integer ambiguity set with the minimum unit variance should be the correct integer set. As mentioned above the

first task in beginning the integer ambiguity search is the selection of the primary SVs and the primary double differences as shown in Figure 4.12.

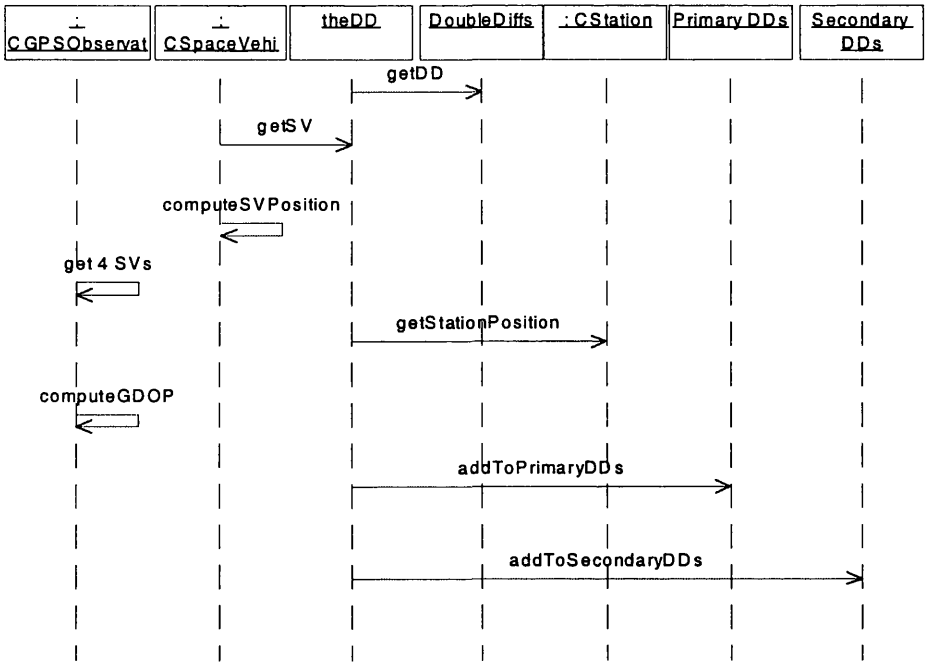

Figure 4.12 Sequence Diagram of Selecting Primary Double Differences

This is done by computing the GDOP (Geometric Dilution Of Precision). The GDOP is an index of measure reflecting the SV geometry on the position and the time estimate. It is given by the following equation.

$$GDOP = \sqrt{\frac{\sigma_E^2 + \sigma_N^2 + \sigma_U^2 + \sigma_t^2}{\sigma_0^2}} \qquad (4\text{-}5)$$

where, $\sigma_E, \sigma_N, \sigma_U$ σ_t are the standard deviation in the horizontal and vertical position and the receiver clock. σ_0 is the standard deviation of the observation.

A set of ambiguities candidates is generated from the primary double differences by the sequences of actions shown in Figure 4.13.

These candidates, which are the search space, are computed using the initial approximate position of the rover station.

In the search for the correct ambiguity set, computations of the direction cosine from the station to the SV are constantly carried out. The x, y and z components of the direction

cosine are the linear terms of the Taylor series expansion of ρ, the distance from the station to the SV, shown in the following equation.

$$\rho_i^j(t) = \rho_{i0}^j(t) - \frac{X^j(t) - X_{i0}}{\rho_{i0}^j(t)}\Delta X_i - \frac{Y^j(t) - Y_{i0}}{\rho_{i0}^j(t)}\Delta Y_i - \frac{Z^j(t) - Z_{i0}}{\rho_{i0}^j(t)}\Delta Z_i \quad (4\text{-}6)$$

$\rho_{i0}^j(t)$ is the approximate distance between the station i and SV j, at epoch t, $X^j(t)$, $Y^j(t)$, $Z^j(t)$ are the coordinates of the SV j, at epoch t, and X_{i0}, Y_{i0}, Z_{i0} are approximate coordinates of the station. $\Delta X_i, \Delta Y_i, \Delta Z_i$ are the unknown parameters to be solved, which are the correction to the initial approximation.

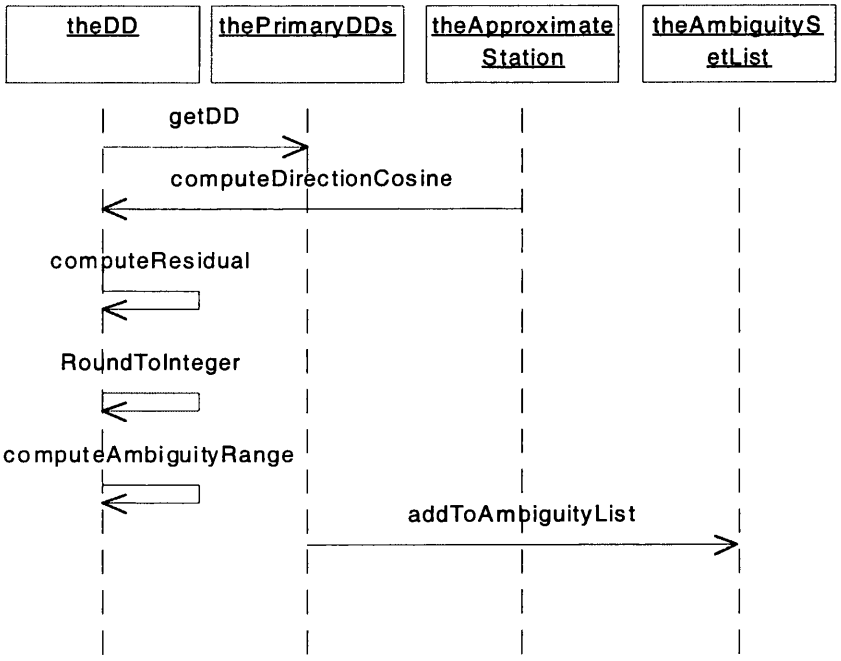

Figure 4.13 Sequence Diagram of Creating Ambiguity Set List

The observation equation of a linearised double difference equation, given below, looks similar to the equation (4-6) . The coefficients derived from the direction cosines (i.e. the coefficients of each unknown parameter) are used to build up the observation equation. It would be convenient to have a member function in the double difference class to compute these direction cosine related terms. Another difference is the introduction of the term on integer ambiguity:

$$l_{AB}^{jk}(t) = a_{XB}^{jk} \cdot \Delta X_B + a_{YB}^{jk} \cdot \Delta Y_B + a_{ZB}^{jk} \cdot \Delta Z_B + \lambda \cdot N_{AB}^{jk} \quad (4\text{-}7)$$

where,

$$a_{XB}^{jk} = -\frac{X^k(t) - X_{B0}}{\rho_{B0}^k(t)} + \frac{X^j(t) - X_{B0}}{\rho_{B0}^j(t)},$$

$$a_{YB}^{jk} = -\frac{Y^k(t) - Y_{B0}}{\rho_{B0}^k(t)} + \frac{Y^j(t) - Y_{B0}}{\rho_{B0}^j(t)},$$

$$a_{ZB}^{jk} = -\frac{Z^k(t) - Z_{B0}}{\rho_{B0}^k(t)} + \frac{Z^j(t) - Z_{B0}}{\rho_{B0}^j(t)}$$

and $\Delta X_B, \Delta Y_B, \Delta Z_B$ are the correction parameters to the approximate rover station coordinates. $l_{AB}^{jk}(t)$ is the residual, λ is the wavelength of the carrier and N_{AB}^{jk} is the unknown integer ambiguity.

The residual, also known as the misclosure, is the difference between the observed value and the computed value of the mathematical model. The residual of the double difference observation equation is given as:

$$l_{AB}^{jk}(t) = \lambda \cdot \Phi_{AB}^{jk}(t) - \rho_{B0}^k(t) + \rho_{B0}^j(t) + \rho_A^k(t) - \rho_A^j(t) \quad (4\text{-}8)$$

Direction cosines and the residuals are computed for each double difference using the approximate coordinates of the rover station. The approximate coordinates are used to compute a set of real valued numbers representing the ambiguities. A set of integer ambiguities is formed from this real valued set of ambiguities by rounding the computed value to the nearest integer. From this set a series of integers (for example -5 to +5) are added to the rounded value to form the list of candidate ambiguity sets.

From the candidate ambiguity sets, each candidate is retrieved and then tested for the probability of it being the correct ambiguity set. This begins with the computation of the trial position using the double difference data of the primary double differences and the selected candidate ambiguity set (Figure 4.14).

In this process, a standard least squares adjustment of the double difference observations is carried out for the solution of the trial position. One thing to note here is that the observation matrix and the weight matrix remain the same for all candidates. It is only the residual

vector which is changed with the change of the value of the integer ambiguity. This means that the normal matrix need be set up only once and also that only the right hand side vector of the normal equation need be set up.

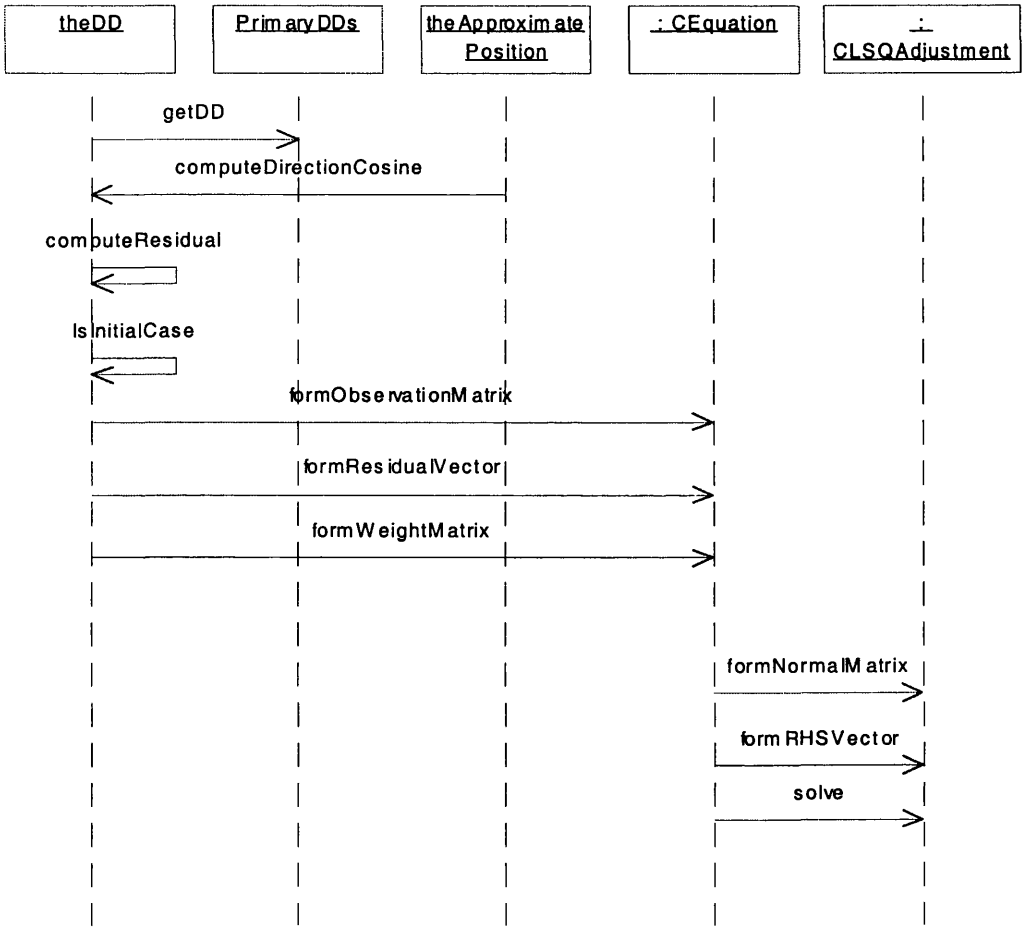

Figure 4.14 Sequence Diagram of Computing Trial Position from Primary DDs

With the trial position computed using the double differences of the primary double differences and the candidate integer ambiguity set, the integer ambiguities for the secondary double differences are derived (Figure 4.15).

The directional cosines and the residuals of the secondary double differences are computed first and then rounded to the nearest integer. Both the primary and secondary double differences are added to a list of double differences called Combined DDs. This list now contains the double differences with each double difference's integer ambiguity fixed to a value.

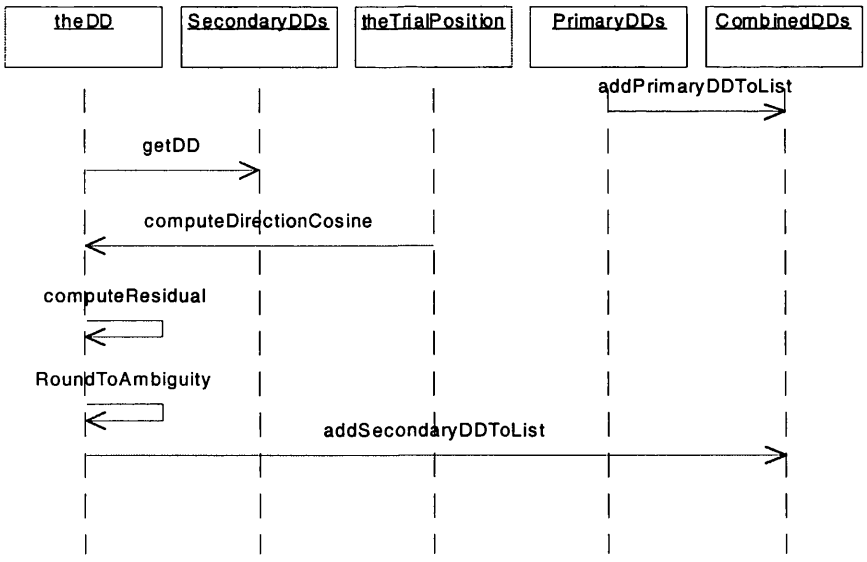

Figure 4.15 Sequence Diagram of Computing Secondary Ambiguities from Trial Position

The final stage is the evaluation of this set of ambiguity sets by performing a standard least squares adjustment with the observation equations formed from the double differences of the CombinedDDs (Figure 4.16).

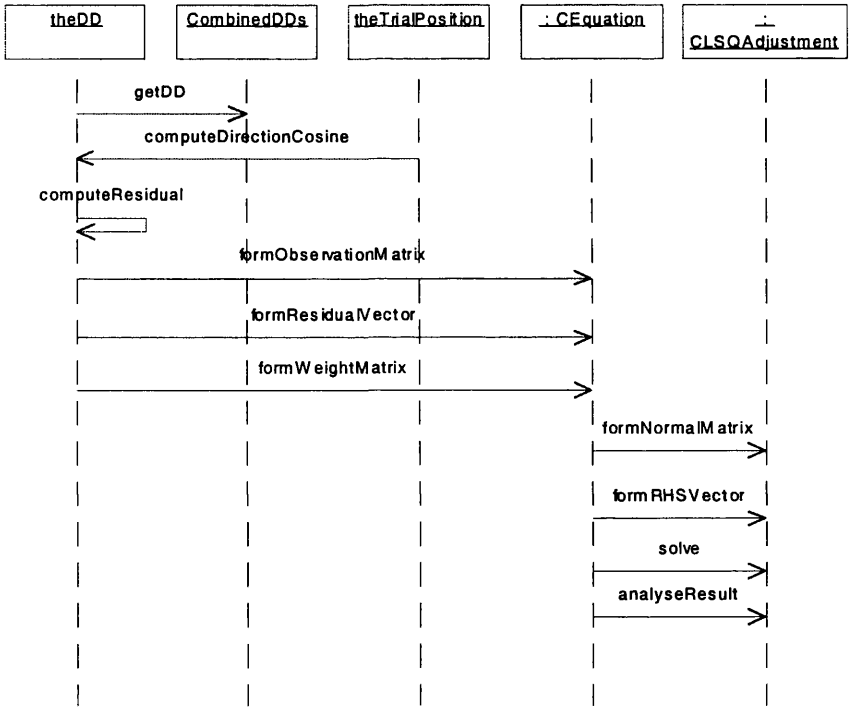

Figure 4.16 Sequence Diagram of Computation of Residual with Combined DDs

A very similar procedure of computing the trial position with the primary double differences is pursued except that CombinedDDs are used and the result is analysed to compute the *a posteriori* residual value of the finally adjusted value. This value is retained for all the candidates. The candidate with minimum *a posteriori* residual value is selected as the correct ambiguity set.

Because of the huge number of candidates, it is necessary to reduce the computation time where possible. For this purpose, the ambiguity function value can be computed to reject the candidate whose ambiguity function value falls below a selected criterion. This process is shown in Figure 4.17. The ambiguity function value can be said to represent the size of the residual from a scale of zero to one [Counselman et al., 1981][Mader, 1990].

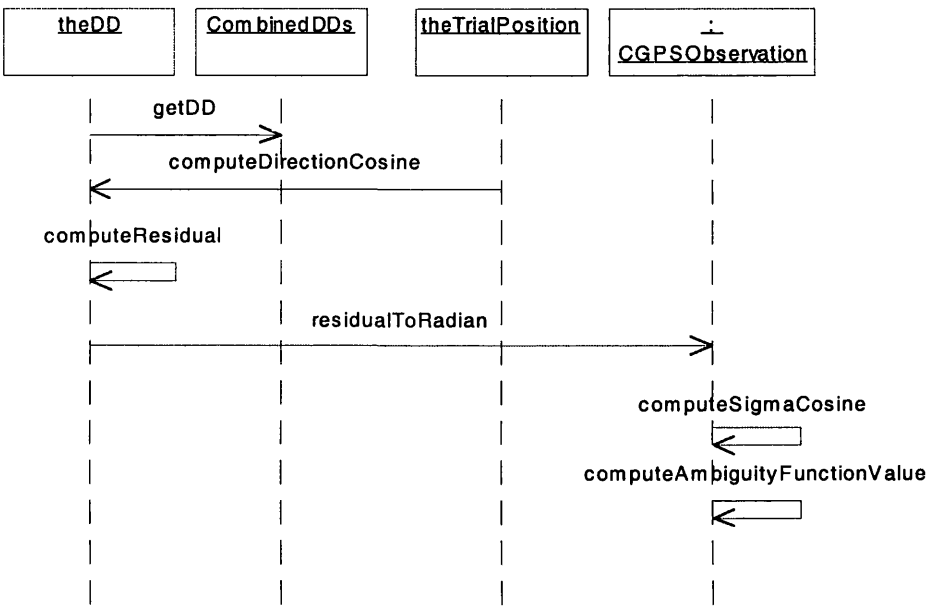

Figure 4.17 Sequence Diagram for Computation of Ambiguity Function Value

The integer ambiguity candidate whose value is closest to 1 is likely to have the minimum residual. The computation of ambiguity function value takes less computation time than the formal computation of residuals by the least squares adjustment. So candidates with relatively large residuals can be filtered out by the computation of the ambiguity function value. However the least squares method is more rigorous and provides the statistical information which is not possible to acquire with the ambiguity function value computation.

4.1.2 Design of Classes of the GPS Data Processing

The CSpaceVehicle class with the CSignalMeasurement struct are shown in Figure 4.18. Some changes have been introduced from the initially proposed structure of Figure 4.3.

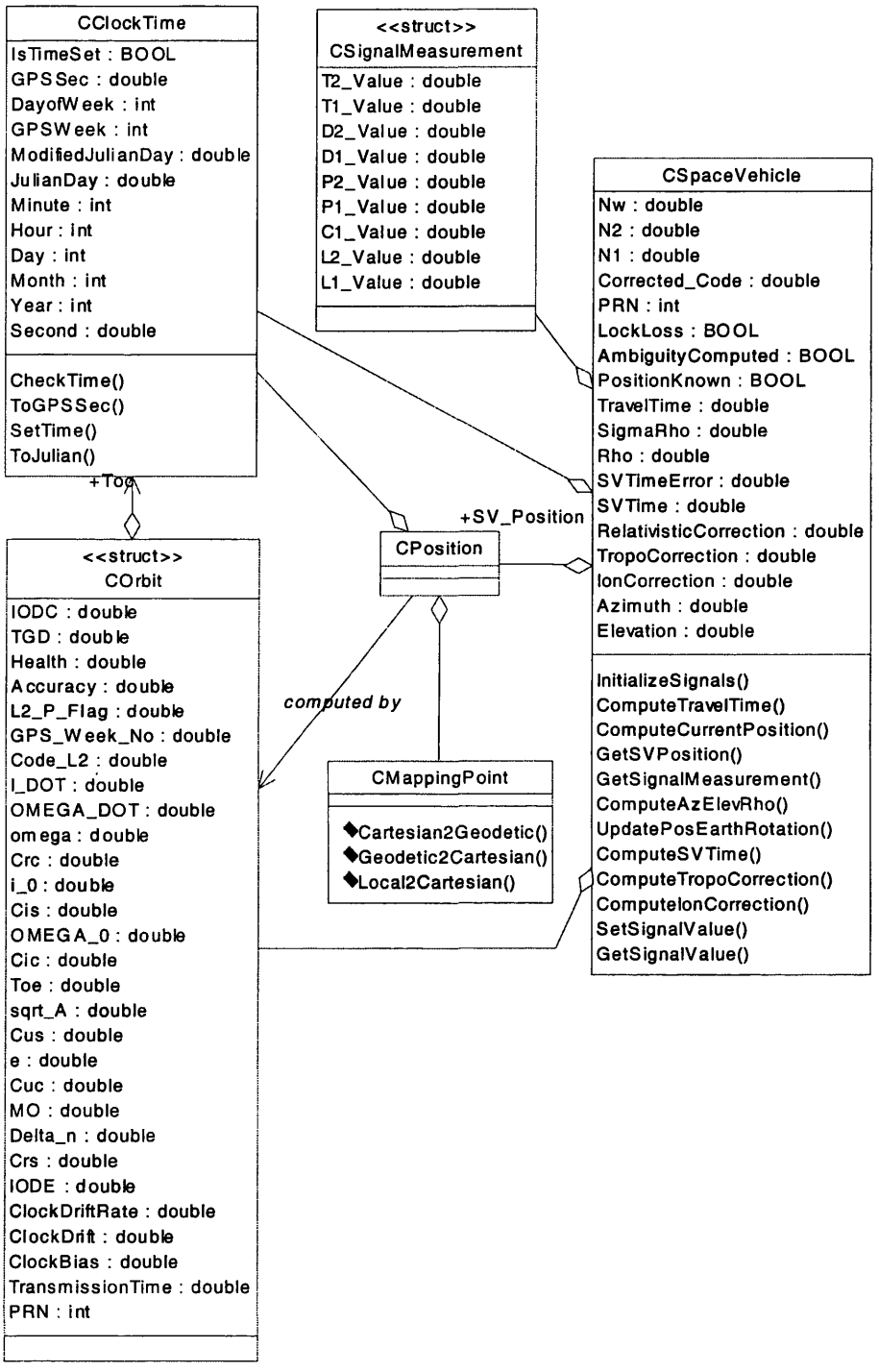

Figure 4.18 Detail Class Diagram of CSpaceVehicle and CSignalMeasurement

It was later found that classifying the signal measurement into code and phase and also classifying the phase into L1 and L2 were not necessary. Therefore to simplify the design, CSignalMeasurement has been changed to a struct with all the signals as data members. The values of the signals that are not collected will be set to zero.

The other change is in the CPosition class. It was proposed that the CPosition has CMappingPoint and CClockTime. This has been changed to an inheritance relationship where the CPosition inherits the properties of CMappingPoint which has the time property. So instead of regarding position as an object which has a point and time, it is regarded as a point in time. The transformation functions to and from geodetic and Cartesian e-frame are added to the CMappingPoint class.

Some of the important functions in the CSpaceVehicle class are ComputeCurrentPosition and SetSignalMeasurement and GetSignalMeasurement. The ComputeCurrentPosition computes the current position of the SV using the data member orbit object. The other member functions in the CSpaceVehicle class support this computation of current position and compensate for the error due to tropospheric and ionospheric refraction and the earth rotation.

The main role of the CClockTime class is to transform the civilian time of year, month, day, hour, minute and second into the GPS second of the week. This is carried out by the ToGPSSec function.

Figure 4.19 shows the detail Class Diagram of the CStation class. The main role of the CStation class is to hold the observation file and provide epoch observations to the CObservation class. It also holds the objects of the CPosition class to represent the initial approximation position and the adjusted position.

Most of its data members are read in from the header part of the RINEX observation data and its member functions are the initialisation of data members and the transformation of positions to and from the antenna and the station. The CStation class uses a receiver to compute its position. The receiver has an antenna attached to it. The antenna has a phase centre with the offset and the correction factor data.

The CStation class also has a list of objects of the CEvent class. An event is defined by EventType integer, which represents cycle slip, shutter release, lock loss or any other event the user wishes to define.

Two classes are derived from the CStation class, namely the CRoverStation and the CReferenceStation. The only difference is that the CRoverStation class holds a list of objects

of the CPosition class for the adjusted positions and the CReferenceStation has only one object of the CPosition class.

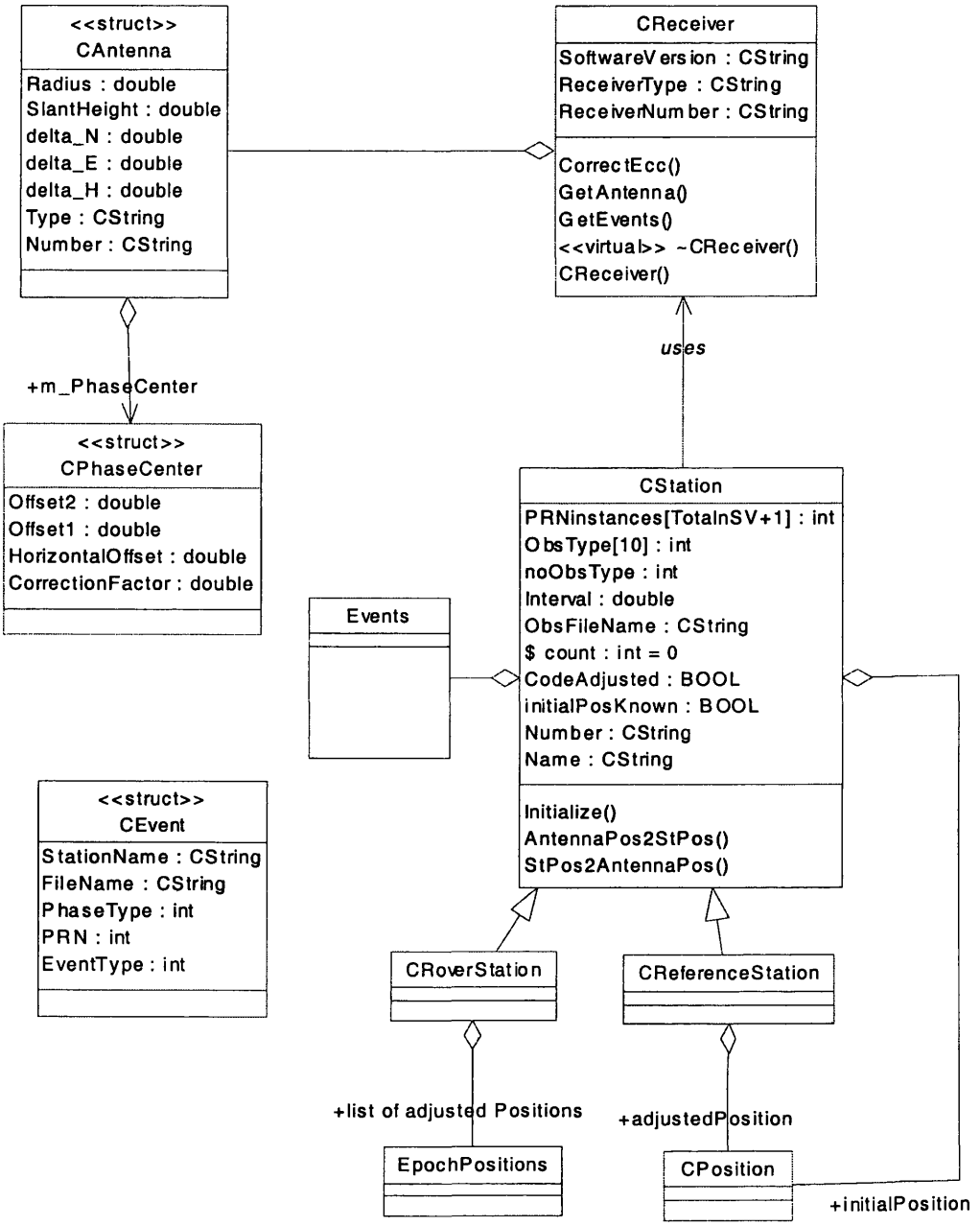

Figure 4.19 Detail Class Diagram of CStation

Figure 4.20 shows the detail Class Diagram of the CObservation class. It is the most important class in the domain of GPS data processing, as most of the computation is carried out by this class, especially in a single epoch ambiguity resolution processing. Most of the

functions in the CObservation class are named as described in the Kinematic Phase Processing of the previous section on the analysis of the GPS data processing.

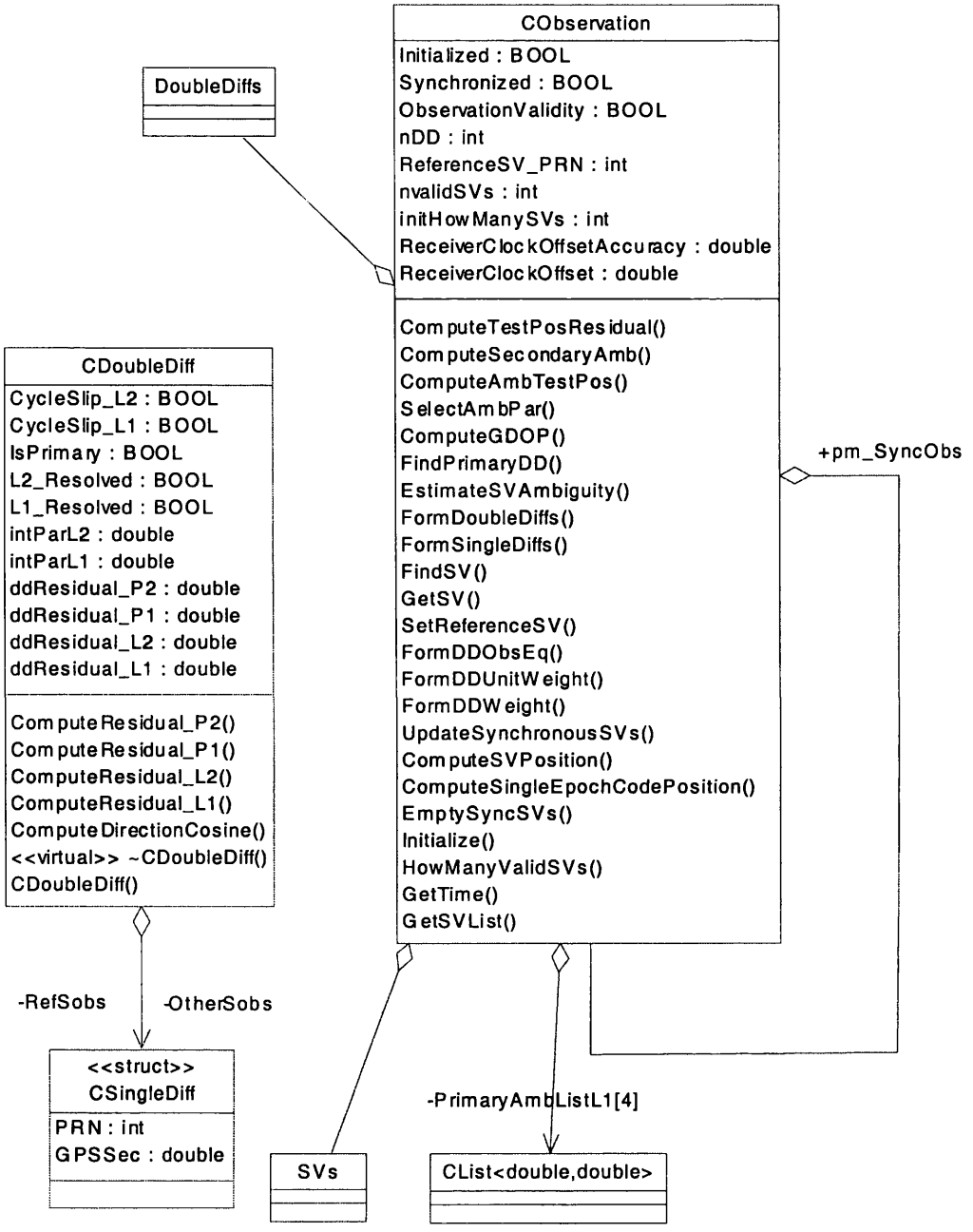

Figure 4.20 Detail Class Diagram of CObservation

The CObservation has a list of SVs, a list of objects of the CDoubleDiff class and a list of double variables representing the integer ambiguities of the primary SVs. The list of double differences are formed by the FormDoubleDiffs function. This function will in turn initiate

the FormSingleDiffs function to form the single differences prior to computing the double difference values. The CDoubleDifference class has data members to show its state as to whether cycle slip has occurred, whether the integer ambiguity has been resolved and whether the double difference is a primary double difference. The computed integer ambiguity is stored as intParL1 or intParL2. The computed residuals are also stored as its data members.

The PrimaryAmbListL1[4], holds the candidate integer ambiguity set of the primary double differences. It is formed by the EstimateSVAmbiguity function. PrimaryAmbListL1[1] will hold all the candidate integer ambiguity of the first primary double difference, PrimaryAmbListL1[2] for the second primary double difference and PrimaryAmbListL1[3] for the third primary double difference.

Each of the candidate values will then fill the intParL1 variable of the CDoubleDiff class for the subsequent processing of searching for the most likely integer ambiguity set.

4.2 Processing of IMU Data

Britting specifies that "...An inertial navigation system utilizes the inertial properties of sensors mounted aboard the vehicle to execute the navigation function. The system accomplishes this task through appropriate processing of the data obtained from force and inertial angular velocity measurements..." [Britting, 1971 (pp. 1-10)].

He also specifies that all inertial navigation systems must perform the following basic functions:

- Provide measurements in a selected reference frame;
- Measure specific force;
- Have knowledge of the gravitational field; and,
- Time integrate the specific force data to obtain velocity and position information.

The reference frame is obtained through the use of a set of gyros. Three gyros are used to obtain the rotations of a three dimensional Cartesian coordinate frame. Three accelerometers are used to measure the specific force vector. However the measured specific force vector also includes forces of the local gravitational field. Therefore the gravitational field should be taken into account when the measured specific force data are integrated to obtain the velocity and the position.

As in previous cases of adjustment of image observations and GPS observations, in IMU observations too, a mathematical model is used to relate the observations values from the sensors and the parameters that are sought. In dynamic systems, the Kalman filtering method is currently widely deployed to estimate the state vector from the mathematical model optimally.

In this section, the IMU data processing methods and the Kalman filtering methods will be analysed and designed using the Object Oriented approach. The designed methods will deal only with strapdown IMUs, as these are widely used for mapping purposes at present, due to their relatively low cost.

4.2.1 Overview of IMU Data Processing

In the Object Oriented design for IMU data processing the CIMU class, representing the IMU, aggregates a list of objects of the class CIMUEpochMeasurement in the class CIMUMeasurements (Figure 4.21).

The important components of an IMU measurement are the time of the measurement, the specific force vector and the angular rotation rate vector. It should be noted that the observed values are increment data from the previous state.

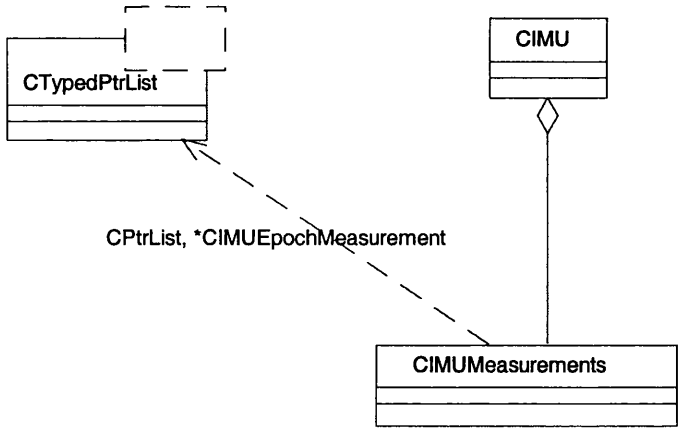

Figure 4.21 Class Diagram of CIMUMeasurements

The Sequence Diagram of Figure 4.22 shows the sequence of actions and interactions between different objects when a data file is accessed.

An IMU object reads the initial data such as its calibration data regarding the gyro drift, accelerometer bias and a description of the location of centre of gravity. As each data record is read, an object of the CIMUEpochMeasurement is created. Then the measurement time,

the gyro data output and the accelerometer data output are read; these values will populate the Angle Data object and the Force Data object. The populated CIMUEpochMeasurement is then added to the list, CIMUMeasurements. In post processing, each of the CIMUEpochMeasurements will be accessed sequentially to be processed.

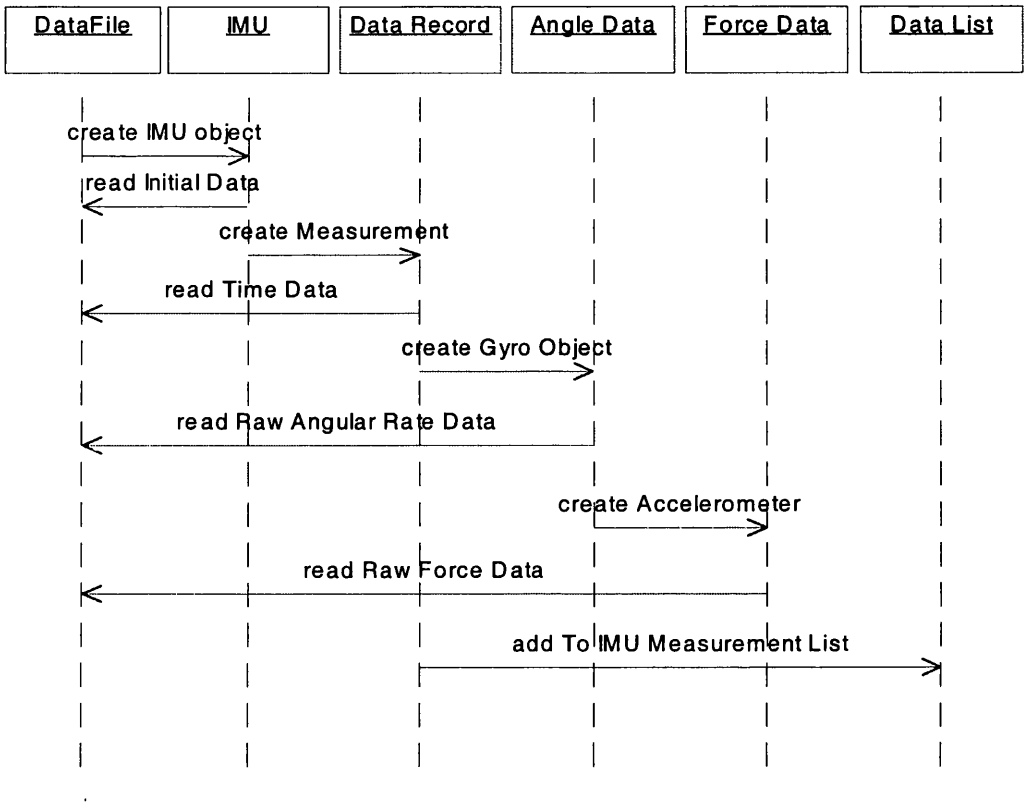

Figure 4.22 Sequence Diagram of Reading IMU Data File

Figure 4.23 shows the class diagram of CIMUEpochMeasurement. The CIMUEpochMeasurement has three aggregated classes, CIMUTime, CAccelerometer and the CGyro.

The CIMUTime is derived from the CClockTime class, which was defined in the GPS data processing section. This is a helpful feature because the local time frame of the IMU is normally related to the GPS time frame when the IMU observations are processed and updated by GPS observations during the Kalman filtering process.

The CAccelerometer holds the measurements produced by the accelerometers and processes these observations. The CGravity and CCoriolis computes the gravity force and the Coriolis force. These forces will be removed from the raw measurement to get the force from the inertial acceleration, from which velocity and position can be directly obtained.

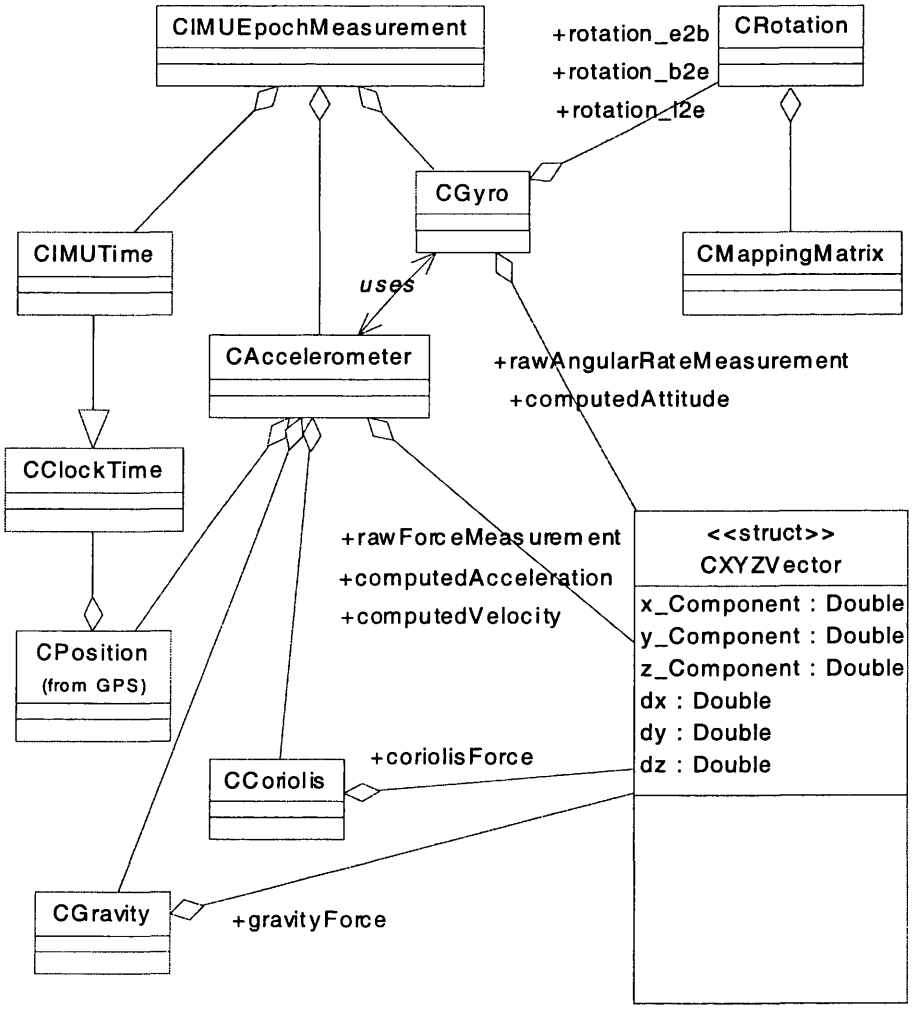

Figure 4.23 General Class Diagram of CIMUEpochMeasurement

The CGyro holds rotation matrices which are computed at the epoch of measurement. Various rotation matrices are computed. The notations rotation_e2b, rotation_b2e and rotation_l2e shown in the figure, refer to the rotation matrices which transform force measurements and angular measurements from e-frame to b-frame, b-frame to e-frame and l-frame to e-frame respectively.

Many of the measurements and computed values have three components, i.e. one component for each of the x,y and z directions. A CXYZVector struct is defined to hold these components as one unit. The CGyro class has at its data members rawAngularRateMeasurement and computedAttitude which are both objects of the CXYZVector. The CAccelerometer class also has objects of the CXYZVector, i.e. rawForceMeasurement, computedAcceleration and computedVelcity. The CCoriolis and the

CGravity class each have CXYZVector objects to hold the Coriolis force and the Gravity force.

Another point to note is that, during computation, the CAccelerometer class needs to access the rotation matrices of the CGyro class, and the CGyro class needs to access the position of CAccelerometer. These imply that each class needs to access private data members of the other class. This can be implemented in C++ by using the 'friend' keyword. This association is shown in the diagram above as the 'uses' relationship with arrows pointing to each other.

4.2.2 Mechanisation of IMU Data

The mechanisation of IMU data refers to the process of each CIMU record of observation where the raw observation data of angular rate and force in the b-frame are transformed to position, velocity and attitude in the e-frame. The usage of the term mechanisation has already been explained at the beginning of Chapter 4.

The Sequence Diagram of the mechanisation process is shown in Figure 4.24 (All basic equations used for IMU data processing in this chapter have been referenced from Schwarz et al. [1994]).

Initialisation of the mechanisation process is carried out as shown in Figure 4.25. Initialisation is necessary at the beginning of each sample of observations in the coarse alignment process which will be explained in the following subsection 4.2.3.

For each record the time interval is computed by differencing the time data of the previous and the current observation. The rotation data or the angular rate measurement is refined by removing the drift error. The force data are also likewise refined by removing the bias error. The drift error and the bias error are provided from the calibration data, which would be stored in the CIMU object.

$$\omega_{ib}^{b} = \omega raw_{ib}^{b} - driftVector \qquad (4\text{-}9)$$

$$f_{ib}^{b} = fraw_{ib}^{b} - biasVector \qquad (4\text{-}10)$$

where, ω_{ib}^{b} and f_{ib}^{b} refer to the refined angular rate measurement and the refined force measurement of the b-frame relative to the i-frame quantified to real valued numbers in the b-frame. Angular rates and force measurements are changed to angles and velocity by

integration with respect to time. The next process needs to transform the values in the b-frame to the e-frame with the rotation matrix, R_b^e .

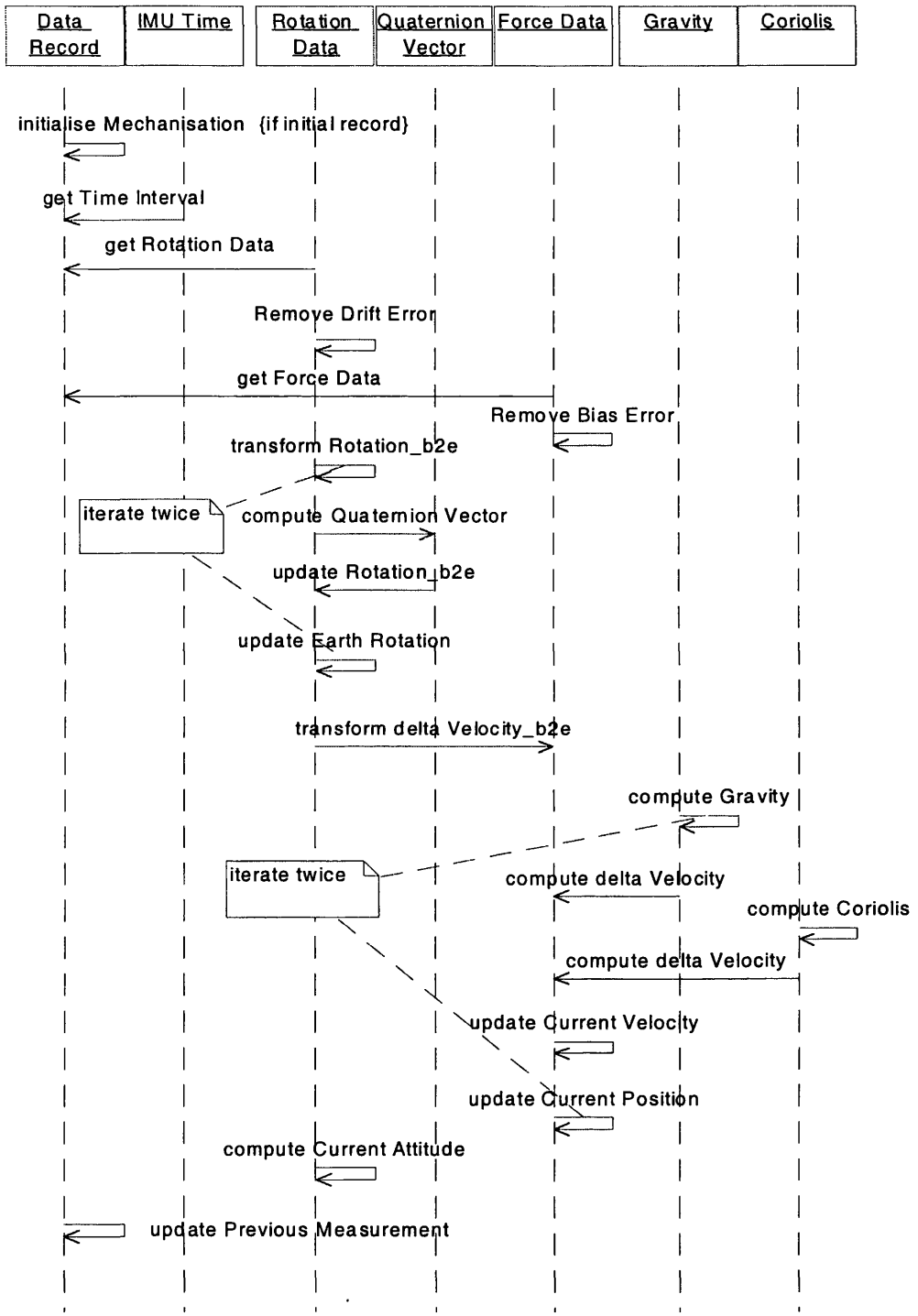

Figure 4.24 Sequence Diagram of Mechanisation of an Epoch Measurement

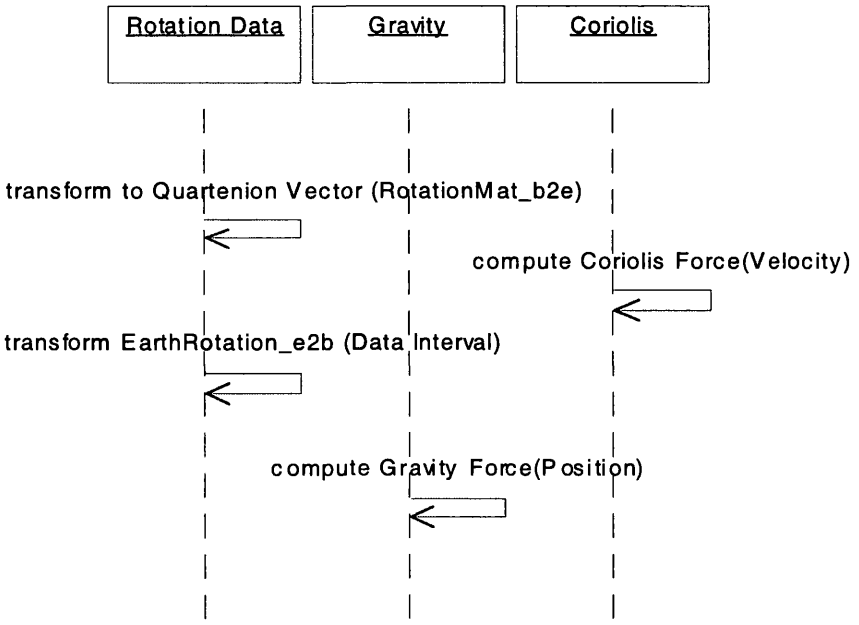

Figure 4.25 Sequence Diagram of Initialise Mechanisation

The rotation matrix R_b^e is updated from its initial state by the observed values of the increase in the rotational angles. These increment values of the b-frame relative to the e-frame can be computed by removing the earth rotation, a geodetic constant. But the earth rotation which is given in the e-frame needs to be transformed to the b-frame first by applying the rotation matrix R_e^b. This matrix is obtained by transposing the R_b^e matrix. So this process is iterated twice to converge closer to the true value.

With the computed rotation matrix, R_b^e the veloctiy data is transformed from the b-frame to the e-frame. Now that the initial raw force data has been transformed to velocity in the e-frame, the next task is to remove the gravity and the Coriolis force. The Coriolis force is given by the equation below.

$$Coriolis = \begin{bmatrix} -2 \cdot \omega_e \cdot v_y \\ 2 \cdot \omega_e \cdot v_x \\ 0 \end{bmatrix} \quad (4\text{-}11)$$

where, ω_e is the earth roation rate and v_x, v_y are velocity directional components. The normal gravity vector in the e-frame, γ_e is given by the equation below and the coefficients are as defined by Wei et al. [1990].

$$\gamma_e = \frac{a_1}{r}\begin{pmatrix} \left\{c_1 + c_2 \cdot t^2 + c_3 \cdot t^4 + c_4 \cdot t^6\right\} \cdot x_e \\ \left\{c_1 + c_2 \cdot t^2 + c_3 \cdot t^4 + c_4 \cdot t^6\right\} \cdot y_e \\ \left\{d_1 + d_2 \cdot t^2 + d_3 \cdot t^4 + d_4 \cdot t^6\right\} \cdot z_e \end{pmatrix} \quad (4\text{-}12)$$

The centripetal acceleration is added to the normal gravity vector to compute the effect of the gravity. The centripetal acceleration is given by the equation below.

$$\text{Centripetal Acceleration} = \begin{bmatrix} \omega_e^2 \cdot x_e \\ \omega_e^2 \cdot y_e \\ 0 \end{bmatrix} \quad (4\text{-}13)$$

where, ω_e is the earth rotation rate and x_e, y_e are the coordinate values in the e-frame. The remaining velocity is used to compute the position. But the computation of gravity requires positional data. Therefore as in the case of computing the rotation matrix R_b^e, the procedure is iterated twice.

The attitude is computed by using the coordinates of the current position. First the Cartesian coordinates are transformed to the Geodetic coordinates of latitude and longitude. The rotation matrix, R_l^e which transforms from l-frame to e-frame, is computed using these latitude and longitude values. Then the rotation matrix, R_b^l is computed by the following equation.

$$R_b^l = R_e^l \times R_b^e \quad (4\text{-}14)$$

Roll, pitch and yaw are derived from the R_b^l matrix as follows.

$$\left.\begin{aligned} roll &= \tan^{-1}\left(-\frac{r_{31}}{r_{33}}\right) \\ pitch &= \sin^{-1}\left(r_{32}\right) \\ yaw &= \tan^{-1}\left(-\frac{r_{12}}{r_{22}}\right) \end{aligned}\right\} \quad (4\text{-}15)$$

where, r_{ij} refers to the element of the the R_b^l matrix at row i and column j.

The mechanisation of a single record of observation data produces increments in the position and velocity during the time interval. These increments are added to the previous values to

obtain the current position and the current velocity. The new position is then used to compute the current attitude.

Finally, the values of the current values for position, velocity and attitude are set as the previous values. The increment values next computed will be added to these previous values to get new current values.

4.2.3 Alignment of IMU

Alignment in this thesis is considered only with respect to a stationary strapdown IMU. The alignment process determines the initial roll, pitch and yaw from the data observations of a stationary strapdown IMU. This process is divided into two parts: the coarse alignment and the fine alignment. Coarse alignment computes the approximate attitude from the data outputs and the positional data. Fine alignment uses the output from the coarse alignment and improves the attitude using the Kalman filtering method where zero velocity is used in the measurement update. The Sequence Diagram of Initial Alignment is shown in Figure 4.26. An alignment time is initially set to determine the number of observations that will be required to determine the attitude of the IMU.

Next, a sample size is set to divide the amount of data in the alignment time into chunks of data records. Following this, for each chunk (or sample), the attitude is computed. This is done in two separate steps. The first step computes the azimuth using the average angular rate observations of the sample.

Ideally, in a stationary state, this should be the same as the earth rotation rate. The average angular rate in the b-frame is first transformed to the l-frame then the azimuth is obtained by the following equation.

$$azimuth = -\tan^{-1}\left(\frac{\left[\omega_{ie}^{l}\right]_x}{\left[\omega_{ie}^{l}\right]_y}\right) \tag{4-16}$$

where, $\left[\omega_{ie}^{l}\right]_x, \left[\omega_{ie}^{l}\right]_y$ are the x, y directional components of the angular rate in the l-frame. The transformation from b-frame to l-frame requires the rotation matrix R_b^l, which in turn needs to hold the values of roll and pitch. But because these have not been computed yet, an initial approximation is first used for the roll and pitch.

In the second step, the roll and pitch are computed using the computed azimuth. The computed set of roll, pitch and azimuth values are then used to update the rotation matrix R_b^l. This iteration is continued for the coarse alignment time.

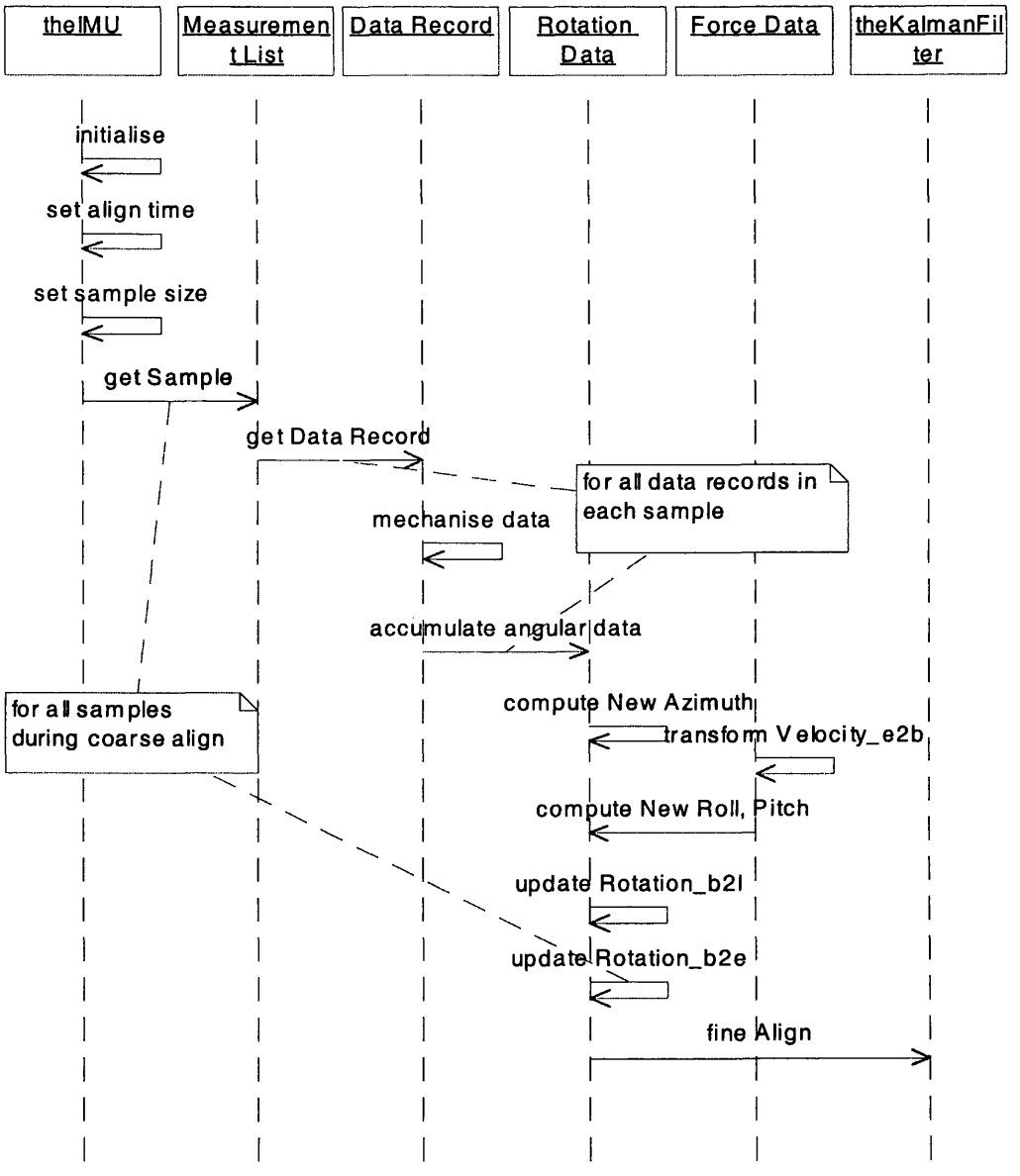

Figure 4.26 Sequence Diagram of Initial Alignment

The computation of roll and pitch is performed using the following equations.

$$\delta roll = -\sin^{-1}\left(\frac{[v^b]_x}{[T \cdot |g|]}\right) \qquad (4\text{-}17)$$

$$\delta pitch = \sin^{-}\left(\frac{[v^b]_y}{[T \cdot |g|]}\right) \quad (4\text{-}18)$$

where, $\delta roll, \delta pitch$ are the corrections to the initial approximation, $[v^b]_x, [v^b]_y$ are the x, y directional components of the velocity in the b-frame, T is the time interval for the sample and $|g|$ is the magnitude of gravity. After the completion of the coarse alignment, fine alignment is carried out.

4.2.4 Detail Class Diagrams of IMU Data Processing

In the design of the classes of IMU data processing, the CIMU class has been designated with roles which are global in nature for the whole processing time and the CIMUEpochMeasurement class is mostly involved with the mechanisation of each observation.

The CIMU class and other related classes are shown in Figure 4.27. The CIMU class itself holds data about the IMU characteristics, such as its calibration data and scale factors. Its member functions include computing the scaling factors. An object of the CIMUEpochMeasurement, named prevMeasurement, is aggregated in this class to hold the result of a processed data record. This is used in the mechanisation of each epoch measurement.

Three important classes have been added to the initial design, i.e. the CCoarseAlignment class, CFineAlignment class and the CTrajectoryEstimation class.

As the name suggests, each is responsible for the tasks of coarse alignment, fine alignment and the estimation of the trajectory. The CAttitude class is a data member of both the CoarseAlignment and the CFineAlignment class as they each computes the initial attitude of the IMU.

The result of trajectory estimation is, however, a list of positions, velocities and attitudes. A new class called CTrajectory has been created to hold the time of estimation and the estimated position, velocity and the attitude (Figure 4.28). The CTrajectoryEstimation class will store the computed list of times, position, velocities and attitudes in the Trajectories class.

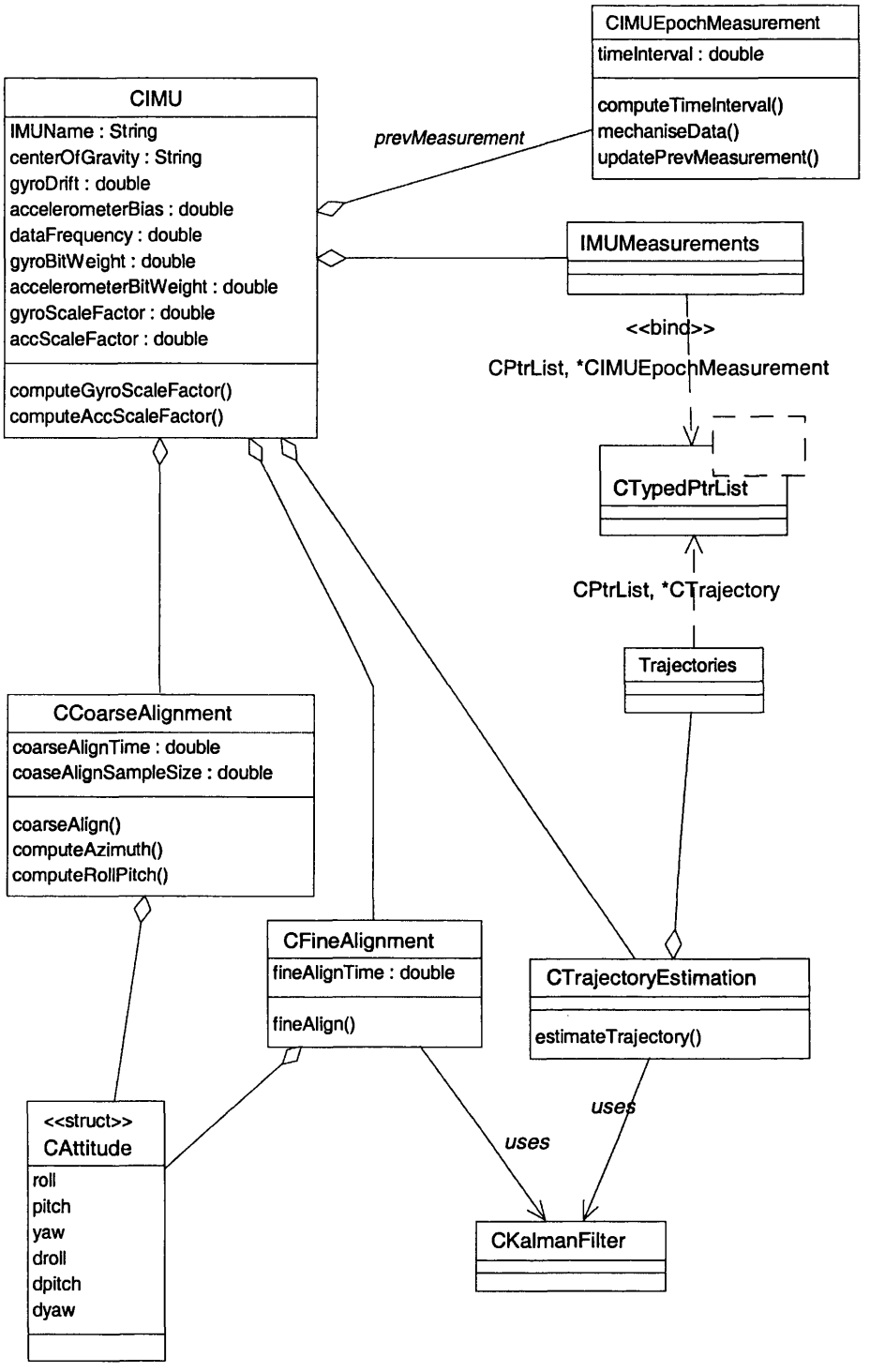

Figure 4.27 Class Diagram of CIMU

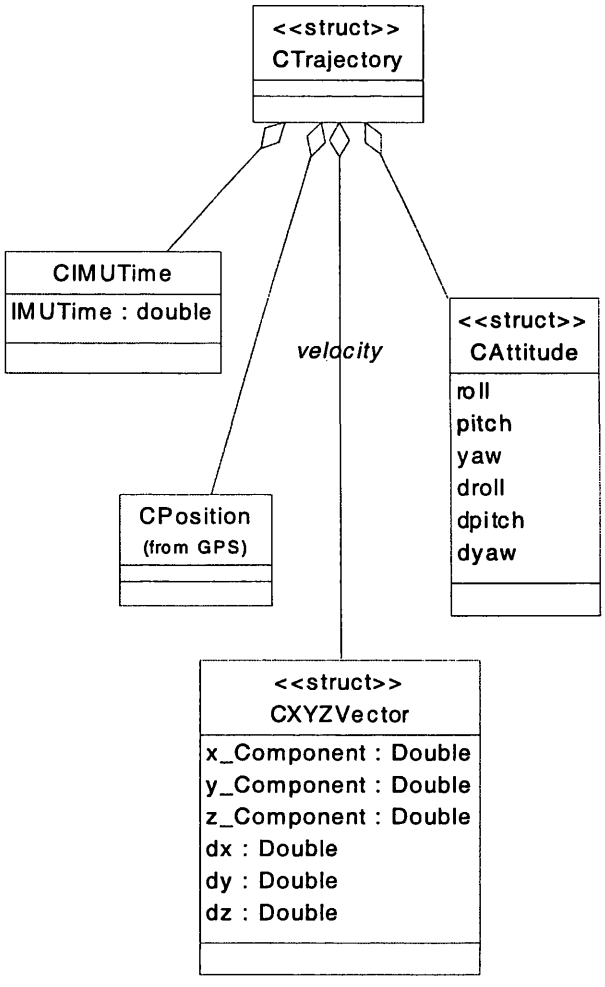

Figure 4.28 Class Diagram of CTrajectory

The Figure 4.29 shows the Class Diagram of CIMUEpochMeasurement class and other related classes. The main role of this class is to mechanise each data measurement. It has three main aggregated classes: CIMUTime, CAccelerometer and CGyro. The CIMUTime class is the time in the IMU time frame for the current measurement. It is derived from the CClockTime class, which was used in GPS data processing. Member functions related to IMU time handling should be included in this class.

The CAccelerometer handles all functions related to force measurement and processes leading to the current velocity and position. This includes computation of the gravity and the Coriolis force. The member object 'currentPosition' is used to hold positional data. The CPosition class which has been used in GPS data processing has useful functions such as transformation of geodetic to Cartesian coordinates. This is another good example of resusability of a software component through design of the software.

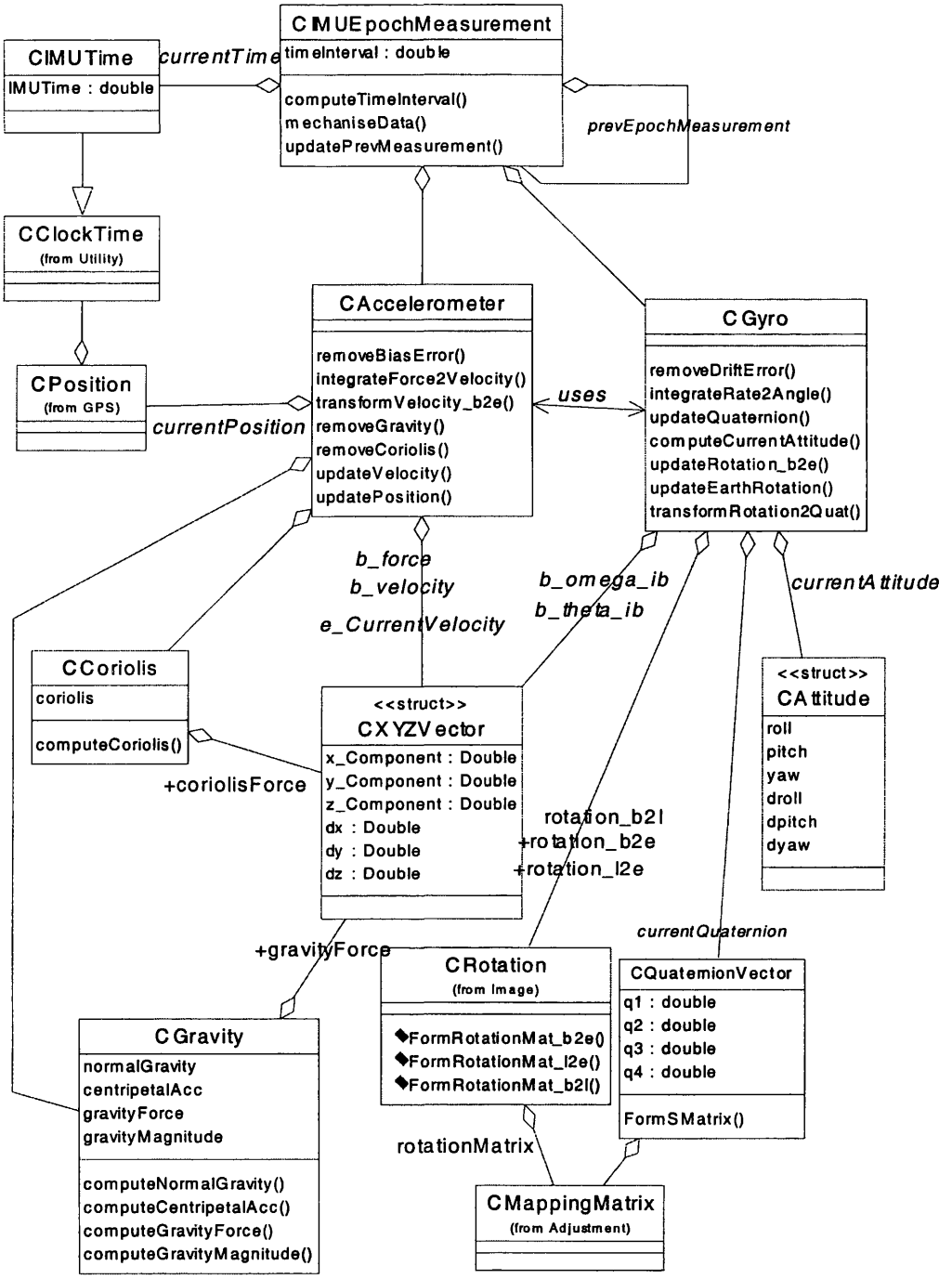

Figure 4.29 Detailed Class Diagram of CIMUEpochMeasurement

The raw force data, the integrated velocity increment and the current velocity in the e-frame are instantiated from the CXYZVector class.

The CGyro class handles all functions related to the angular rate data and the computation of rotational transformation matrices. The CQuaternionVector class is used in the updating of the rotation matrix R_b^e and the CAttitude class is used to hold the roll, pitch and yaw data.

4.3 Kalman Filtering

Kalman filtering is an estimation process widely used in kinematic and dynamic positioning problems. It has been covered in numerous textbooks since its original inception in 1960, [Kalman, 1960] [Gelb, 1974] [Anderson et al., 1979] [Maybeck, 1979] [Maybeck, 1982]. In this section an Object Oriented design of the basic Kalman filter class will be made.

4.3.1 The INS Error Model

A 15 error state INS model in the e-frame is introduced in this section which was developed at the University of Calgary [Schwarz et al., 1994].

The rate of change of an error model is described by the following linear differential equation.

$$\dot{x} = F \cdot x + w \tag{4-19}$$

where, $\dot{x}$ is the rate of change of the error model, F is the dynamics matrix, x is the error state vector and w is the system noise.

The error state vector for the INS error model is given by the next equation, where, each row is a vector with x,y and z components.

$$x = \begin{bmatrix} \delta position \\ \delta velocity \\ \delta misalignment \\ \delta drift \\ \delta bias \end{bmatrix} \tag{4-20}$$

The derivation of the dynamics matrix is important because it is used in the formation of the transition matrix which is a part of the Kalman filtering process. The formation of the transition matrix from the dynamics matrix is given by the equation below.

$$\Phi_{k-1} = e^{-F\Delta t} = I + F\Delta t + \frac{(F\Delta t)^2}{2!} \tag{4-21}$$

where, Φ_{k-1} is the transition matrix describing change from instance k-1 to k and I is an identity matrix. Usually only the first two terms on the right hand side of the above equation are used in the computation of the transition matrix.

The dynamics matrix is derived from the following set of error equations [Skaloud, 1995][Schwarz et al., 1994].

$$\dot{\delta velocity} = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ N_{21} & N_{22} & N_{23} \\ N_{31} & N_{32} & N_{33} \end{bmatrix} \delta position + \begin{bmatrix} 0 & 2\omega_e & 0 \\ -2\omega_e & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \delta velocity + \begin{bmatrix} 0 & f_z^e & -f_y^e \\ -f_z^e & 0 & f_x^e \\ f_y^e & -f_x^e & 0 \end{bmatrix} \delta misalignment + R_b^e \cdot \delta bias + noise \quad (4\text{-}22)$$

$$\dot{\delta position} = \delta velocity + noise \quad (4\text{-}23)$$

$$\dot{\delta misalignment} = \begin{bmatrix} 0 & \omega_e & 0 \\ -\omega_e & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \delta misalignment + R_b^e \cdot \delta drift + noise \quad (4\text{-}24)$$

$$\dot{\delta drift} = noise \quad (4\text{-}25)$$

$$\dot{\delta bias} = noise \quad (4\text{-}26)$$

where, f_x^e, f_y^e and f_z^e are specific force components in e-frame and N_{11} to N_{33} represent the influence of the normal gravity error.

4.3.2 The Kalman Filter Class

The Kalman filtering process can be classified into the two different subprocesses of propagating (or estimating) and updating. The Kalman filtering process is shown in the Figure 4.30 Sequence Diagram.

An initialisation of the state vector (X vector), the state covariance matrix (P matrix) and the spectral density matrix of the system noise (Q matrix) is followed by the iteration of the propagation process and the measurement update process.

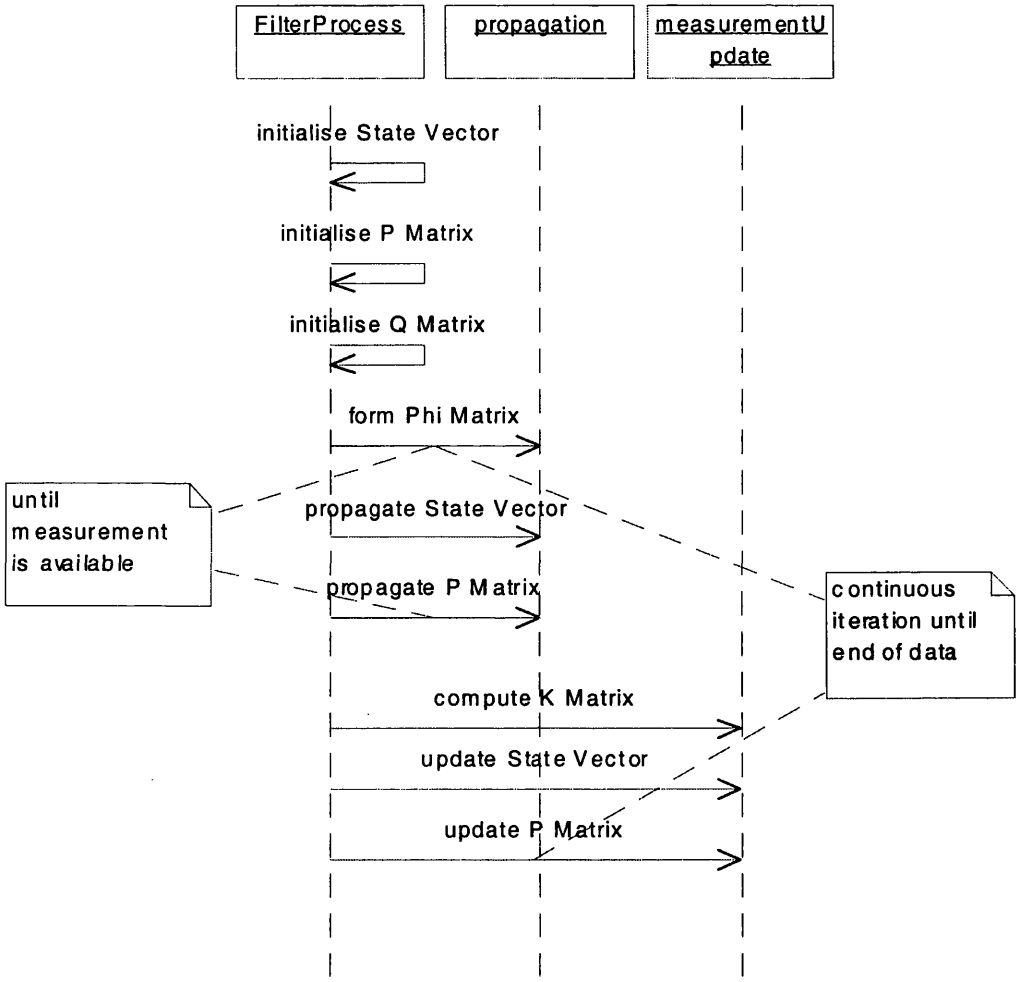

Figure 4.30 Sequence Diagram of Kalman Filtering

In the propagation process, the transition matrix (Phi matrix) is formed to be used in the propagation of the state vector, given by the following equation.

$$x_k^- = \Phi_{k-1} x_{k-1}^+ \qquad (4\text{-}27)$$

where Φ_{k-1} is the transition matrix from epoch k-1 to k. The '$^-$' superscript on the variables indicate that they are prior estimates (i.e. propagated) whereas those with the '+' superscript indicate that they are updated estimates. The state error covariance matrix is then computed by the following equation.

$$P_k^- = \Phi_{k-1} P_{k-1}^+ \Phi_{k-1}^T + Q_{k-1} \qquad (4\text{-}28)$$

When the external measurement becomes available, the propagated state vector and its error covariance matrix are updated. If the measurement is given by the following equation,

$$z_k = H_k \cdot x_k + v_k \tag{4-29}$$

where, z_k is the measurement vector, H_k is the design matrix and v_k is the measurement error vector, the updated state vector and its updated error covariance matrix are given as follows.

$$x_k^+ = x_k^- + K_k\left[z_k - H_k \cdot x_k^-\right] \tag{4-30}$$

$$P_k^+ = \left[I - K_k \cdot H_k\right] \cdot P_k^- \tag{4-31}$$

$$K_k = P_k^- \cdot H_k^T \cdot \left[H_k \cdot P_k^- \cdot H_k^T + R_k\right]^{-1} \tag{4-32}$$

where, K_k is the Kalman gain matrix and R_k is the measurement error covariance matrix.

The following figure, Figure 4.31, shows the class diagram of the CKalmanFiltering class. In IMU data processing, an INS Kalman filtering class should be derived from this class and the specific formation of the transition matrix and the design matrix should be carried out.

The initialisation process is placed in the CKalmanFiltering main class. The state vector, the P matrix and the Q matrix are also placed in the CKalmanFiltering class. The aggregated CMeasurementUpdate and CPropagation class each have two pointer variables pointing to the P matrix and the state vector of the CKalmanFiltering class. Therefore, while the updating and propagating process are delegated to different classes, the updating and the propagating are done for the same state vector and P matrix of the CKalmanFiltering class. As shown in the figure, all the propagating and updating process and their relevant matrices are placed in the CPropagation class and the CMeasurementUpdate class.

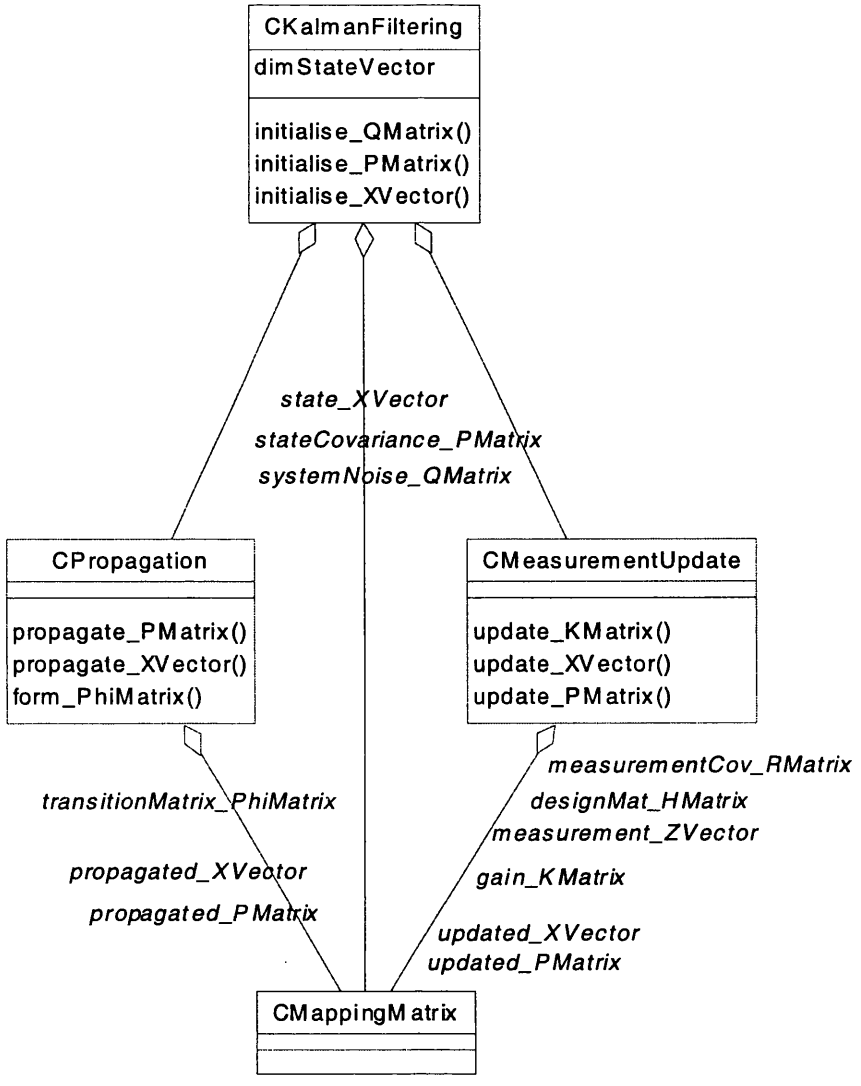

Figure 4.31 Class Diagram of Kalman Filtering

4.4 Summary of Chapter 4

In this chapter, the design of the classes involved in GPS data processing, IMU data processing and Kalman filtering were presented.

Much of the analysis of the GPS data processing was devoted to the resolution of the initial ambiguity which is necessary in GPS surveys with phase measurement observations. The most important classes in GPS data processing are the CSpaceVehicle class, the CObservation and the CDoubleDiff class. The main responsibilities of the CSpaceVehicle class are to compute its own position at an epoch and to hold all the observation data (i.e. code ranges, phase measurements, Doppler measurements and others) captured by a receiver

for the particular space vehicle. The main responsibilities of the CObservation class are to form the observation equation for subsequent least squares adjustment and to resolve the integer ambiguity. The CDoubleDiff class is used by the CObservation class in its formation of double difference equations. The classed designed for GPS data processing is shown in Table 4.1.

Table 4.1 Classes Designed for GPS Data Processing

Space Vehicle related	Station related	Observation related
CSpaceVehicle	CStation	CObservation
CSignalMeasurement	CRoverStation	CDoubleDiff
COrbit	CReferenceStation	CSingleDiff
CClockTime	CPosition	
	CReceiver	
	CAntenna	
	CPhaseCentre	
	CEvent	

Analysis of IMU data processing was focussed on the mechanisation of the IMU data and the initial alignment process. The main classes for the IMU data processing are the CIMUEpochMeasurement class, the CAccelerometer class and the CGyro class. The CIMU class is the main class which uses other aggregated classes to perform alignment, mechanisation and estimation of the trajectory. The CIMUEpochMeasurement class is responsible for mechanising and producing a set of position vector, velocity vector and attitude vector from each epoch data of the IMU. The CIMUEpochMeasurement class aggregates both the CAccelerometer class and the CGyro class. The force data are held by the CAccelerometer class which integrates force to velocity and then to position. The CGyro class holds the angular rate data from the gyro and processes these to attitude data. The CAccelerometer and the CGyro classes interact to mechanise each observation. Separate classes for coarse and fine alignment, CCoarseAlignment class and CFineAlignment class were designed to perform the alignment process.

Classes for the Kalman filtering process was also designed. Three classes, the CKalmanFiltering class, the CPropagation class and the CMeasurementUpdate class, were

designed to form the necessary matrices which will perform the estimation and the measurement updating process of Kalman filtering.

The classes designed for IMU data processing and Kalman filtering are shown in Table 4.2.

Table 4.2 Classes Designed for IMU Data Processing and Kalman Filtering

IMU related	Epoch Observation related	Kalman Filtering related
CIMU	CIMUEpochMeasurement	CKalmanFiltering
CCoarseAlignment	CAccelerometer	CPropagation
CFineAlignment	CGyro	CMeasurement
CTrajectory	CCoriolis	
CAttitude	CGravity	
CIMUTime	CQuaternionVector	
CXYZVector		
CKalman		

5. ANALYSIS AND DESIGN OF THE IMAGE POINT REFERENCING SUBSYSTEM

The image point referencing subsystem performs the task of geometrically relating the image coordinate frame to the ground coordinate frame. The main processes of this subsystem are the data reading process (image points, control points and sensor data) and the bundle adjustment process.

Most of the classes used in image point referencing have been identified in the image acquisition subsystem. Many of the processes in this image point referencing subsystem involve the formation of matrices using various classes (e.g. CImagePoint, CImageSensor, CControlPoint, ...) followed by algebraic computations using these matrices. A Use Case Diagram of the image point referencing subsystem is shown in Figure 5.1. As shown in the diagram, sensor data and control points are accessed from the data files or directly from the image acquisition subsystem or the positioning subsystem. Image points however can be read from the data file or by a selection of an image point (manual or automatic). The selection of the image point is then followed by its coordinates computation.

In case of a manual point selection, the human operator (GeoSpatial DB Specialist in this case) points an indicator on the image display to the selected point and records the coordinates by clicking the mouse. The same ground point appearing in other images (called conjugate points) are identified and recorded in the same manner.

In the case of automatic point selection, there are more complex algorithms involved. Selecting a point automatically is difficult because it involves the judgement as to which point is suitable for image point referencing purpose. To emulate the human judgement process would involve a very complex computer process. Instead for this purpose, many distinct points from a region of an image are arbitrarily selected using interest operators. These points are called interest points. This process is represented by the Use Case called Select Interest Points in the diagram. Conjugate points of the arbitrarily selected interest points are selected and measured in other images by the matching process. This is represented by the Match Points Use Case.

The image matching process can be classified into area based, feature based and relational, according to the matching strategy used. The most popular matching strategy is the area based least squares matching. Another area based matching method called the cross correlation matching method is usually used to compute the initial approximate pixel coordinates which are necessary in least squares matching.

In this chapter, the algorithmic aspects of the bundle adjustment will be explained. Sequence Diagrams will be used to illustrate the sequence of processes of the bundle adjustment as well as the objects that interact with each other.

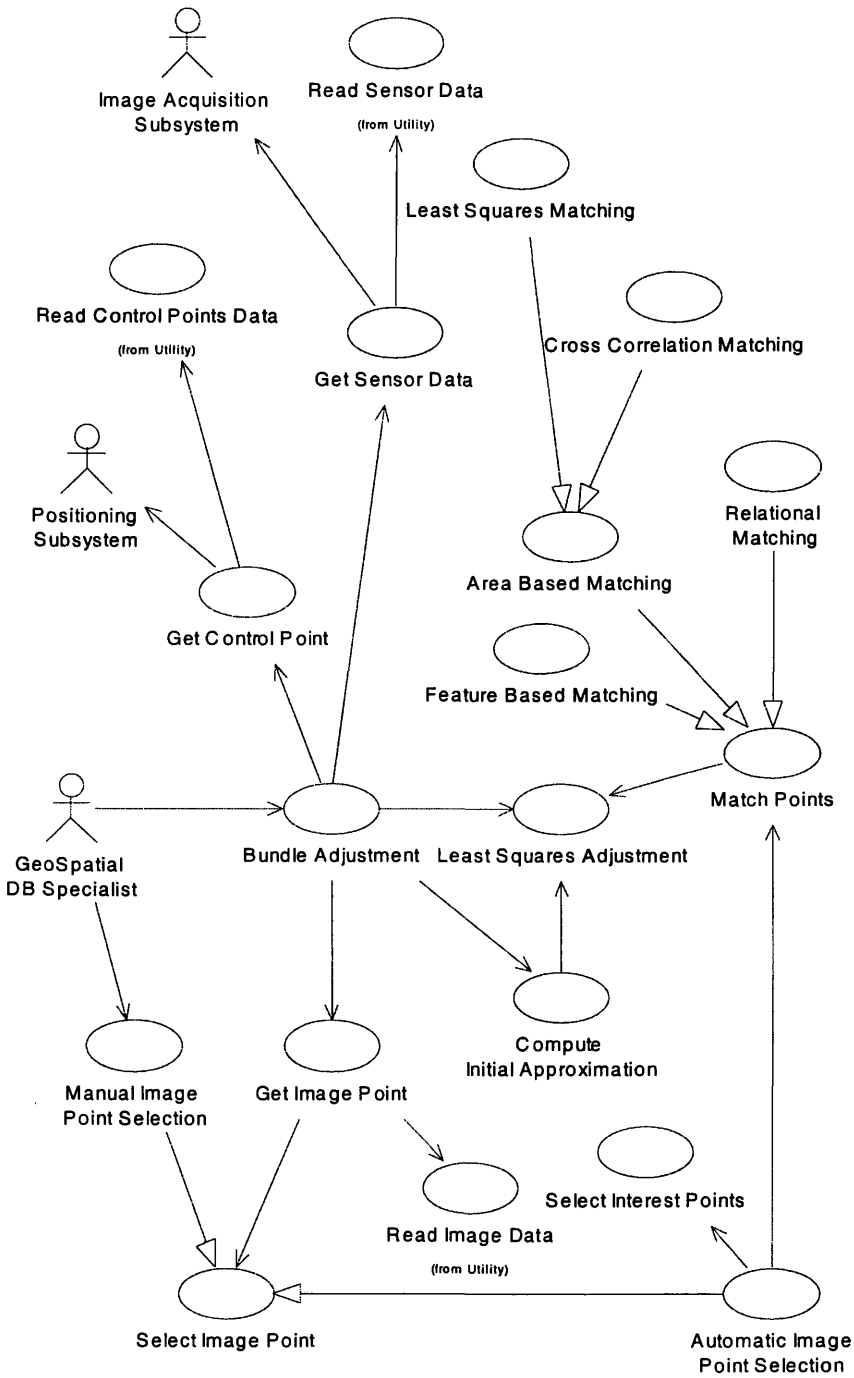

Figure 5.1 Use Case Diagram of Image Point Referencing Subsystem

5.1 The Collinearity Mathematical Model

A bundle of rays in photogrammetry refers to the lines connecting the ground points, the perspective points and the image points. This is shown in Figure 5.2.

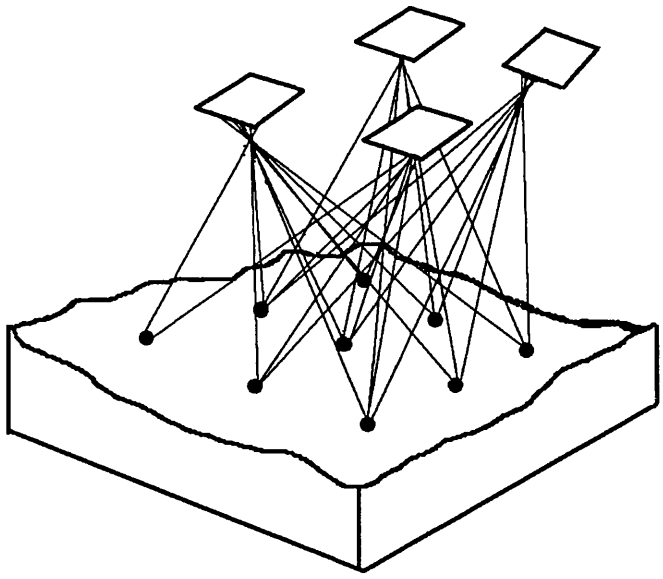

Figure 5.2 Bundle of Rays

The mathematical model in photogrammetry uses this concept of the 'bundle of rays' to set up the relationship between the image coordinate observations and the unknown parameters of the image, the sensor and the ground points. The rays are each modeled as a straight line passing through these points. This mathematical model, widely used in photogrammetry, is known as the collinearity condition. The collinearity equations are given as below (All basic equations used for image point referencing in this chapter have been referenced from Merchant. [1994]).

$$x = (x_p + \delta x) - c\frac{X'}{Z'}$$
$$y = (y_p + \delta y) - c\frac{Y'}{Z'} \qquad (5\text{-}1)$$

where, x, y are the image coordinates, x_p, y_p are the coordinates of the principal point, $\delta x, \delta y$ are image coordinate compensations for the systematic errors, c is the principal distance and X', Y', Z' are defined as follows.

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = R_{image} \begin{bmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{bmatrix} \qquad (5\text{-}2)$$

In the above equation (5-2), X, Y, Z are the ground coordinates, X_0, Y_0, Z_0 are the coordinates of the perspective point and R_{image} is the image rotation matrix formed by ω, φ, κ, the primary, secondary and tertiary rotation angles of the image at the moment of exposure. The rotation matrix elements are shown below.

$$R_{image} = \begin{bmatrix} \cos\varphi\cos\kappa & \cos\omega\sin\kappa + \sin\omega\sin\varphi\cos\kappa & \sin\omega\sin\kappa - \cos\omega\sin\varphi\cos\kappa \\ -\cos\varphi\sin\kappa & \cos\omega\cos\kappa - \sin\omega\sin\varphi\sin\kappa & \sin\omega\cos\kappa + \cos\omega\sin\varphi\sin\kappa \\ \sin\varphi & -\sin\omega\cos\varphi & \cos\omega\cos\varphi \end{bmatrix}$$

The observations in the collinearity model are the image coordinates, and the unknown parameters are classified into three sets: image parameters; sensor parameters; and, point parameters.

The image parameters, also known as exterior orientation parameters, refer to the coordinates of the perspective point and the rotation angles. For each image there are six parameters.

The sensor parameters, also known as interior parameters or additional parameters, refer to the correctional terms to the systematic error. These were explained in Chapter 3. In addition to the lens distortion and the film shrinkage corrections, the principal distance and the principal point coordinates are also included in this parameter set. If only one sensor is used for all the images, which is normally the case, one set of sensor parameters exists for the whole computation. In such a case there would be ten parameters in the sensor parameter set. If the sensor parameter set is included in the computation of parameters, the bundle adjustment becomes the self-calibration process. If the sensor parameters are known, the

sensor related observation equations are not included and the computation is then a normal case of bundle adjustment.

The point parameters are the x,y,z ground coordinates of the observed image point. There are three parameters for each ground point. Control point coordinates, i.e. ground points where the coordinates are known with high accuracy, are also included in the observation equations. These observations are given very large weights and they remain fixed through the computation and act as constraints in the adjustment to influence other parameters.

The observation equations which comprise observations and parameters can be divided into two main types: the Image Observation Equation and the Direct Observation Equation (also called Pseudo Observation Equation).

5.1.1 Image Observation Equation

The Image Observation Equation refers to the observation equations formed using the image point coordinates observations and the collinearity equations. The collinearity equations are linearised with respect to the parameters and iterated to solve for the unknown parameters. The coefficients therefore are the partial derivatives of the observation equations. One pair of equations, one for each x coordinate and y coordinate, exists for each image point observation. In the formation of the design matrix, i.e. coefficient matrix of the the collinearity equations, it is first linearised with respect to the unknown parameters. And as the result of linearisation the coefficients are grouped into parameter groups. The coefficients of the image parameter group, to be called Image Design, are computed as follows

$$\text{Image Design Submatrix} = \begin{bmatrix} \partial F(x_j) / \partial(\kappa,\varphi,\omega,X_0,Y_0,Z_0) \\ \partial F(y_j) / \partial(\kappa,\varphi,\omega,X_0,Y_0,Z_0) \end{bmatrix} \quad (5\text{-}3)$$

The coefficients of the sensor parameter group, to be called Sensor Design Submatrix, are computed as follows

$$\text{Sensor Design Submatrix} = \begin{bmatrix} \partial F(x_j) / \partial(Xp,Yp,CC,P1,P2,K1,K2,K3,A,B) \\ \partial F(y_j) / \partial(Xp,Yp,CC,P1,P2,K1,K2,K3,A,B) \end{bmatrix} \quad (5\text{-}4)$$

The coefficients of the ground point parameter group, to be called Point Design Submatrix, are computed as follows

$$\text{Point Design Submatrix} = \begin{bmatrix} \partial F(x_j) / \partial(X,Y,Z) \\ \partial F(y_j) / \partial(X,Y,Z) \end{bmatrix} \quad (5\text{-}5)$$

The discrepancy vectors are the numerical values obtained by evaluating the mathematical models with the original observations (x,y) and the current estimates of the unknown parameters. They are computed for each image point observation.

$$\text{Image Discrepancy Subvector} = \begin{bmatrix} x_{obs} - x_{computed} \\ y_{obs} - y_{computed} \end{bmatrix} \quad (5\text{-}6)$$

For each image point observed, the above submatrices of equations (5-3) to (5-6) are evaluated. These submatrices and the subvector will be used in the formation of the normal equation.

5.1.2 Direct Observation Equation

An observation model can be adopted to add the parameters as observations. This enables the incorporation of known standard deviations regarding the parameters into the observation model. If the parameters are unknown parameters, such as the coordinates of a tie point, a weight value close to zero can be given to make them free parameters. If the point is a surveyed control point with known standard deviations, the known precision value (for example 0.02 metres) can be used to fix, or constrain, the computation result within this uncertainty level. A simple direct model can be used for this purpose [Brown et al., 1964]. Addition of the direct observation equation to the observation model results in simple matrix addition in the normal equation as shown below:

$$\begin{bmatrix} B_{image}^T WB_{image} + W_{image} & B_{image}^T WB_{sensor} & B_{image}^T WB_{point} \\ & B_{sensor}^T WB_{sensor} + W_{sensor} & B_{sensor}^T WB_{point} \\ & & B_{point}^T WB_{point} + W_{point} \end{bmatrix} \begin{bmatrix} \Delta_{image} \\ \Delta_{sensor} \\ \Delta_{point} \end{bmatrix} \quad (5\text{-}7)$$

$$= \begin{bmatrix} B_{image}^T WE - W_{image} E_{image} \\ B_{sensor}^T WE - W_{sensor} E_{sensor} \\ B_{point}^T WE - W_{point} E_{point} \end{bmatrix}$$

where $B_{image}, B_{sensor}, B_{point}$ are design matrices related to image parameters, sensor parameters and point parameters. W and E are weight matrix and discrepancy vectors formed by applying the image observation equation. $W_{image}, W_{sensor}, W_{point}$ and $E_{image}, E_{sensor}, E_{point}$ are weight matrices and the discrepancy vectors formed by using the direct observation equations.

As the result of adding the direct observation equations, the diagonal submatrices of the normal matrix in equation (5-7) are added with the weight matrices. The Right Hand Side Vector submatrices of the normal equation are subtracted with the weight matrix multiplied by the discrepancy vector. For the first iteration cycle, the discrepancy subvectors $E_{image}, E_{sensor}, E_{point}$ are zero but subsequent iterations should use the updated parameters to compute the values of the discrepancy subvectors of direct observation.

In consideration of the resulting normal equation, it can be seen that processing the direct observation equations amounts to addition of their final contribution to the normal equation. The evaluation of the Design Submatrix or the Discrepancy Subvector is not needed as was the case in processing image observation equations.

5.1.3 Formation of the Normal Equation

After the design submatrices are evaluated for an observation, the next step is the formation of the normal equation and its solution.

There are two way of forming the normal equation. It can be formed by using the full design matrix or by direct formation of the normal matrix from observation submatrices without forming the full design matrix.

The first way of using the full design matrix is shown by the following equation:

$$B^T W B \cdot \Delta = B^T W E \qquad (5\text{-}8)$$

where B is the full design matrix.

The second way of forming the normal equation is a useful method if the observations are not correlated and there are not many variations to the computation process, such as in the bundle adjustment. It is possible to directly compute the contribution of each observation to the formation of the normal equation. This will save a lot of computation time and computer memory space. This procedure of direct formation requires the computation of the location in

the normal matrix to which the contribution of the observations will be accumulated. The computation of this location is simplified if the normal matrix is structured according to the parameter groups.

In this software design, the structure of the normal matrix for the bundle adjustment is configured as below [Merchant, 1984] in Figure 5.3.

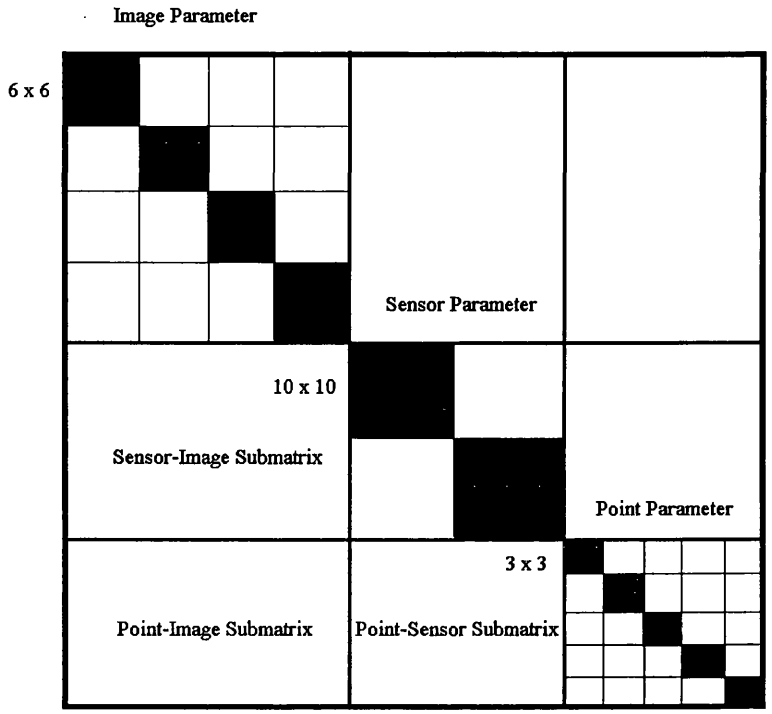

Figure 5.3 Structure of the Normal Equation for Bundle Adjustment

The initial location of the contribution of an observation for Image Design Submatrix is computed as:

$$\text{row} = (\text{Image Index}-1) * 6 + 1 \qquad (5\text{-}9)$$

where Image Index refers to the number of the image in a consecutive series of images.
The initial location of the contribution of an observation for Sensor Design Submatrix is computed as:

$$\text{row} = (\text{Total Image Parameters}) + (\text{Sensor Index}-1) * 10 + 1 \qquad (5\text{-}10)$$

where Total Image Parameters is computed as the total number of images multiplied by 6, which is the number of image parameters for each image.

The initial location of the contribution of an observation for Point Design Submatrix is computed as:

$$\text{row} = \text{Total Image Parameters} + \text{Total Sensor Parameters} + (\text{Point Index} - 1) * 3 + 1 \quad (5\text{-}11)$$

The Total Sensor Parameters value is computed by multiplying the total number of sensors by 10. For the diagonal submatrices, the initial column number is the same as the initial row number. However for off-diagonal submatrices the initial row and initial column numbers are different.

For example to compute these for the Sensor-Image Submatrix, the initial row and column is computed as:

$$\text{row} = \text{Total Image Parameters} + (\text{Sensor Index}-1)*10 + 1 \quad (5\text{-}12)$$

$$\text{col} = (\text{Image Index}-1) * 6 + 1 \quad (5\text{-}13)$$

Initial row and column for other off diagonal submatrices can be computed in a similar way.

The next figure, Figure 5.4, shows the sequence of processes in the formation of the observation equations for each image point.

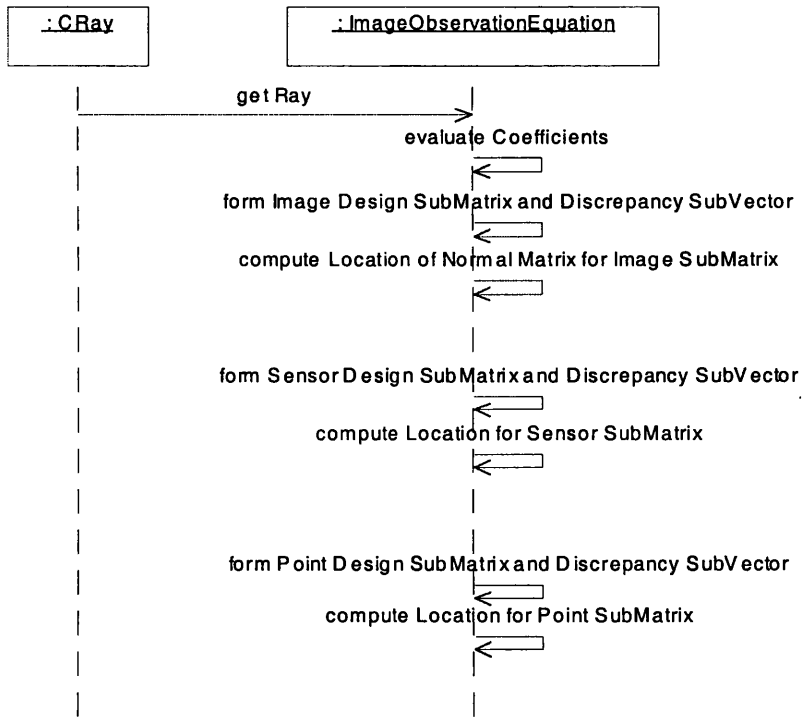

Figure 5.4 Sequence Diagram to Form Observation Equation

5.1.4 Design of the CCollinearityEquation Class

The collinearity equations involve: 1) the coordinates of the ground point, perspective point and the image point; 2) the rotation matrix of the image; and, 3) the principal distance and the principal point of the sensor. If one looks at the class carefully, all these values can be accessed through the CRay class.

The CRay class in itself only contains the three points of a ray as mentioned in item 1) above. The image point however has a member function to retrieve the image which contains the image point, GetImage() (refer to Figure 3.19 in Chapter 3). The CMappingImage class in turn aggregates the CRotation class and the CPerspectivePoint class (refer to Figure 3.19 of Chapter 3). This means that 2) and 3) are also accessible and also that the collinearity equations can be formed using only the CRay class. It can be seen that the introduction of the CRay class has made coupling weaker than when not using the CRay class (refer to subsection 2.1.2 of Chapter 2) thereby increasing modularity and reducing complexity.

The result of the class design is shown in Figure 5.5. CBundleEquation is derived from the virtual CEquation. This CBundleEquation class holds functions common to the image observation equation (i.e. the collinearity equations) and the direct observation equations. The CCollinearityEquation class and the CDirectObservationEquation class are derived from the CBundleEquation. The computation of the location is the same for both the CCollinearityEquation and the CDirectObservationEquation but the computation of contribution is different for each case.

The CDirectObservationEquation needs only to compute the normal matrix contribution and its location. Therefore the contribution function needs to be overridden in the derived classes. The location computation can be done using the base class functions of the CBundleEquation because they are the same for both the image observation equation and the direct observation equations.

The CCollinearityEquation also needs to compute the normal matrix contribution and its location. As in the CDirectObservationEquation, the location computational functions need not be overidden, but in the computation of the normal matrix contribution, extra functions to compute all the design submatrices for each parameter group are necessary. These are included as member functions and named appropriately by such as formImageDesign, formSensorDesign, formPointDesign,...

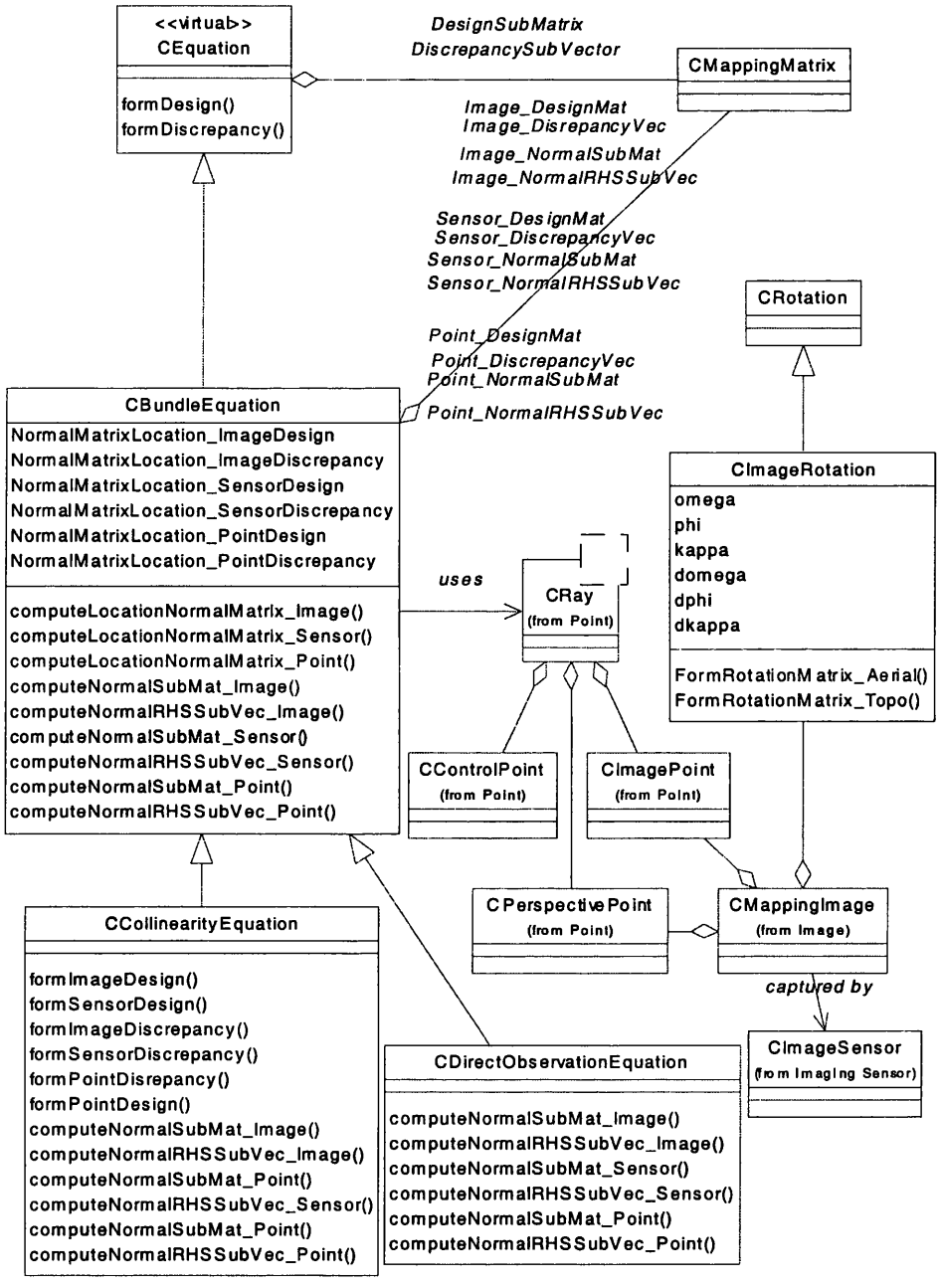

Figure 5.5 Class Diagram of the CCollinearityEquation Class

A new class called CImageRotation has been derived from the CRotation class. This class will handle only the formation of the rotation matrix from the rotational angles. This has been done to differentiate it from the rotational matrix of the Positioning Subsystem, which has different characteristics.

As can be seen from the class diagram, the CBundleEquation class only has one arrow extended to the CRay class. The rest of the information regarding to other various objects are handled by the CRay class. This implies that only the CRay class needs to be taken into account if any changes occur in the CBundleAdjustment class.

5.2 Solution of the Normal Equations

After the formation of the normal equation from the observation equations, the unknown parameters are solved using matrix computations. The CCollinearityEquation class explained in the former subsection was responsible for the formation of the design matrix of the observation equation and also the formation of the normal equation.

In this subsection a class will be designed whose main task will be to solve the normal equation.

The design of class to solve the normal equation should be made in consideration of the fact that although the the observation equation varies under different situations, the solution of the normal equation is processed in a quite standardised manner. Taking this into account, it was determined that this class (to solve the normal equation and to be called the CLSQAdjustment class) should use a virtual base class of the observation equation so that the observation equation can take many forms.

First the sequence of processes and interactions between different objects in the least squares adjustment are investigated. The result of this investigation is shown in the Sequence Diagram of Figure 5.6. After analysing each process in detail a class diagram of the CLSQAdjustment will be presented as the final result of the object oriented design of the least squares adjustment.

After the relevant matrices have been formed, the adjustment process is simply a series of matrix computations. In any matrix computation the dimension of the matrices to be computed must be known in advance. After allocating the memory space for the matrices all the values of the matrix elements are initialised to zero. This is referred to as the 'initialise Adjustment' process in Figure 5.6.

Also, before proceeding with the computation, the redundancy of the observations must always be checked. This means that the number of observations must always be greater than the number of parameters if the computation is to be solved.

The next step in the least squares adjustment is an iteration process which will solve for the correction to the previous approximate parameters. Before the solution of each parameter,

the normal equation matrices are newly formed using the updated parameter values (or the approximate values for the initial iteration). The iteration is continued until it meets the criteria to stop the iteration.

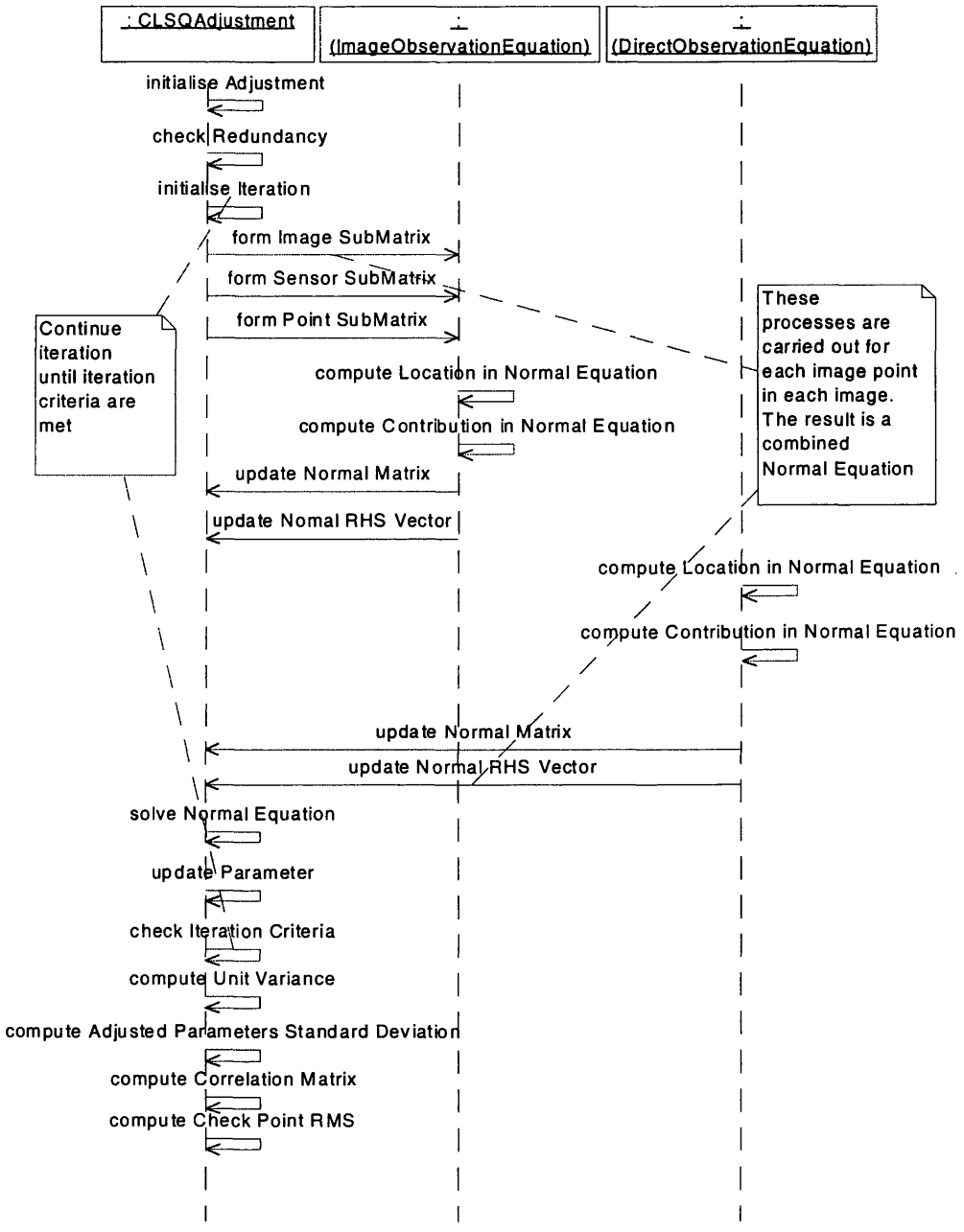

Figure 5.6 Sequence Diagram of Least Squares Adjustment

Usually the number of iterations and the *a priori* unit variance are used as the criteria to stop the iteration. The iteration stops in either of two cases: 1) if the number of iterations exceeds a certain limit (e.g. more than 5) which means something is wrong with the data or the

model; or 2) if the *a priori* unit variance is close to zero (e.g. smaller than 0.0001 metres) which means that a solution has been reached. If the criteria are not met the parameters are updated and the iteration continues.

The solution of the normal equation can be optimised to utilise the fact that the normal matrix is a symmetric and a positive definite matrix. The Cholesky factorisation technique is applied for this purpose.

5.2.1 Cholesky Factorisation

For any symmetric positive definite matrix A, there exists a unique lower triangular matrix with positive diagonal entries such that $A = GG^T$, where G is known as the Cholesky Triangle [Golub et al., 1996]. The factorisation of the A matrix into Cholesky Triangles is known as Cholesky factorisation or Cholesky decomposition. The application of the Cholesky factorisation enables the optimisation of the solution of the normal equation by computing the parameters without having to compute the inverse of the normal matrix. The normal equation (5-8) can be rewritten as:

$$N \cdot \Delta = U \tag{5-14}$$

where N is the normal matrix and U is the RHS vector of the normal equation. After Cholesky factorisation this equation becomes :

$$\left(G \cdot G^T\right) \cdot \Delta = U \tag{5-15}$$

or :

$$G \cdot \left(G^T \cdot \Delta\right) = U \tag{5-16}$$

where G is the Cholesky Triangle. The solution now can be acquired by a series of forward and backward substitutions. First, the equation is solved for $G^T \cdot \Delta$ then for Δ, the unknown parameters. This scheme is made even more ideal by the fact that G is already a triangular matrix.

5.2.2 Analysis of Result

When the solution has converged to a set of parameter values, the adjusted parameters are determined as the solution and a series of computations is carried out which will indicate the statistical precision and accuracy of the solution.

Computation of Unit Variance

After the final solution of the parameters, the residual vector (also known as the misclosure vector), is used to compute the *a posteriori* unit variance. The adjusted parameters are used in the computation of the residual vector computation. The unit variance is computed as:

$$\sigma_0^2 = \frac{v^T W v}{nObservation - nUnknown} \qquad (5\text{-}17)$$

where, σ_0^2 is the *a posteriori* unit variance and v, W are residual vector and weight matrix respectively. The weighted quadratic sum of the residuals is divided by the redundancy to get the unit variance.

Computation of Correlation Coefficient

The introduction of new parameters to a mathematical model can prove to be beneficial or detrimental according to their relevance to the actual physical phenomena. This is because the parameters can be correlated and if the adjustment is carried out without taking the correlation into account, the solution could be quite erroneous even though it has converged to one set of values. In the bundle adjustment it has been found that the principal distance and the principal point coordinates can be highly correlated with the lens distortion parameters, depending on the geometry of the photography [Moniwa, 1981]. Therefore, in a self-calibration adjustment, variation in the geometry should be provided by such methods as large rotation angles (such as in convergent photography) and different image to object distances. Even with such variations, the correlation coefficients of the parameters should be computed to check for correlations. The correlation coefficient between different parameters can be easily computed from the covariance matrix, which is the inverse of the normal matrix.

The correlation coefficient between two parameters are given as [Mikhail et al., 1976 (pp.294-303)]:

$$\rho = \frac{s_{xy}}{s_x \cdot s_y} \qquad (5\text{-}18)$$

where, ρ is the correlation coefficient, s_{xy} is the covariance between the parameter x and y and s_x, s_y are the standard deviations of the parameter x and y, respectively. The values of

correlation coefficients are between zero and one, with a value close to one signifying high correlation between the two selected parameters.

Computation of Check Point Root Mean Square Error

The *a posteriori* unit variance is a statistical measure of precision and it signifies how close the observations are to the mathematical model computed with the adjusted parameters. Sometimes it is necessary to check the result of an adjustment with some external known values of superior accuracy. This would be a check on the absolute accuracy rather than on the statistical precision. Check points are selected for this purpose in many photogrammetric tasks. The adjusted ground coordinates of selected points, which are treated as unknowns, are compared to the pre-surveyed coordinates. The root mean square error of all the check points is an indication of the accuracy of the adjusted parameters, with reference to a set of values obtained in a known superior method, such as ground control point survey. The planimetric root mean square error of check points are computed as:

$$RMS_{XY} = \frac{\sqrt{\sum_{1}^{n}\left(x_{adjusted} - x_{surveyed}\right)^2 + \sum_{1}^{n}\left(y_{adjusted} - y_{surveyed}\right)^2}}{2 \cdot n} \qquad (5\text{-}19)$$

where, n is the number of check points.

The height root mean square error of check points are computed as:

$$RMS_Z = \frac{\sqrt{\sum_{1}^{n}\left(z_{adjusted} - z_{surveyed}\right)^2}}{n} \qquad (5\text{-}20)$$

5.2.3 Storage Optimisation in the Least Squares Adjustment

For the purpose of reducing the required computer memory in an adjustment, a scheme is used to relate the position of a lower triangular matrix to its dynamically allocated memory address [Wolf et al., 1997 (pp. 441-457)]. The mapping scheme is shown in Figure 5.7. This allows memory allocation of only the lower triangular matrix, instead of the whole normal matrix. Thus approximately only half of the memory is required. The numbers along the top of the matrix are the column numbers and down the left side are the row numbers of the normal matrix. The numbers in the shaded boxes are the index numbers of the allocated computer memory.

	1	2	3	4	5
1	1				
2	2	3			
3	4	5	6		
4	7	8	9	10	
5	11	12	13	14	15

Figure 5.7 Memory Index of Column and Rows of Normal Matrix

The mapping between the rows and columns of the normal matrix and the memory index can be represented by the following equation:

$$MemoryIndex(row,column) = \left(column + \frac{row \times (row-1)}{2} \right) \qquad (5\text{-}21)$$

The size of the memory to be allocated for the normal matrix is given by the following equation:

$$MemoryDimension = \frac{row \times (row+1)}{2} + 1 \qquad (5\text{-}22)$$

For the illustration in Figure 5.7, the memory dimension is 16. One extra unit is added because the first block (zero index) is skipped and the first matrix value is stored in index one.

In C++ the usage of the above equations is shown by the example below.

```
#define GetIndex(row,col) (col + row*(row-1)/2)
int row = 5;
int nDimension = row*(row + 1)/2 + 1;
double *pm_NormalMatrix = new double[sizeof(double) * nDimension];
int index = GetIndex(3,4);
*(pm_NormalMatrix + index) = 5;
```

The above example defines the mapping function in the first line then computes the size of memory to be allocated in the third line, given that there are five rows in the matrix. The fifth line uses the mapping function to access the memory for the place of the third row and fourth column of the matrix, (3,4). Finally a value of 5 is located at this place. If many images

containing many control points are to be processed, this storage optimisation scheme may prove to be very cost effective and efficient.

5.2.4 Decorrelation of Correlated Observations

Although it is normally assumed that the observations for a bundle adjustment are uncorrelated, in some applications, such as the double difference equations of GPS data processing, they are correlated. In such cases, weight matrices have to be formed prior to the formation of the normal equation and this adds complexity to the process, especially when optimisation schemes have been introduced.

One way to resolve this situation is to decorrelate the design matrix. The decorrelation of the design matrix results in a transformed design matrix which excludes the weight matrix from the normal equation [Strang et al., 1997]. The first step is to apply Cholesky factorisation to the weight matrix as below:

$$W = G \cdot G^T \tag{5-23}$$

where, W, G are the weight matrix and Cholesky Triangle respectively. Substituting this equation into the normal equation of (5-8) results in the equation below:

$$B^T\left(G \cdot G^T\right)B \cdot \Delta = B^T\left(G \cdot G^T\right)E \tag{5-24}$$

After rearranging, the above equation is expressed as:

$$\left(G^T \cdot B\right)^T \cdot \left(G^T \cdot B\right) \cdot \Delta = \left(G^T \cdot B\right)^T \cdot \left(G^T \cdot E\right) \tag{5-25}$$

Substituting $G^T \cdot B$ with the transformed design matrix B_{new}, results in the new normal equation without the weight matrix.

$$B_{new}^T \cdot B_{new} \cdot \Delta = B_{new} \cdot E_{new} \tag{5-26}$$

It can be seen from the above equation that if the design matrix, B is multiplied by the transpose of the Cholesky Triangle of the weight matrix, G^T, the normal equation is without the weight matrix and it is implicit in the new design matrix, $G^T \cdot B$. The constant vector on the right hand side should also be multiplied by the Cholesky Triangle. By using the decorrelation method shown above, a normal equation can be formed in a simple way for correlated observation equations.

5.2.5 The Design of the CLSQAdjustment Class

The CLSQAdjustment class is designed to be responsible for solution of the normal equation and for the analysis of the adjustment result, as shown in Figure 5.8.

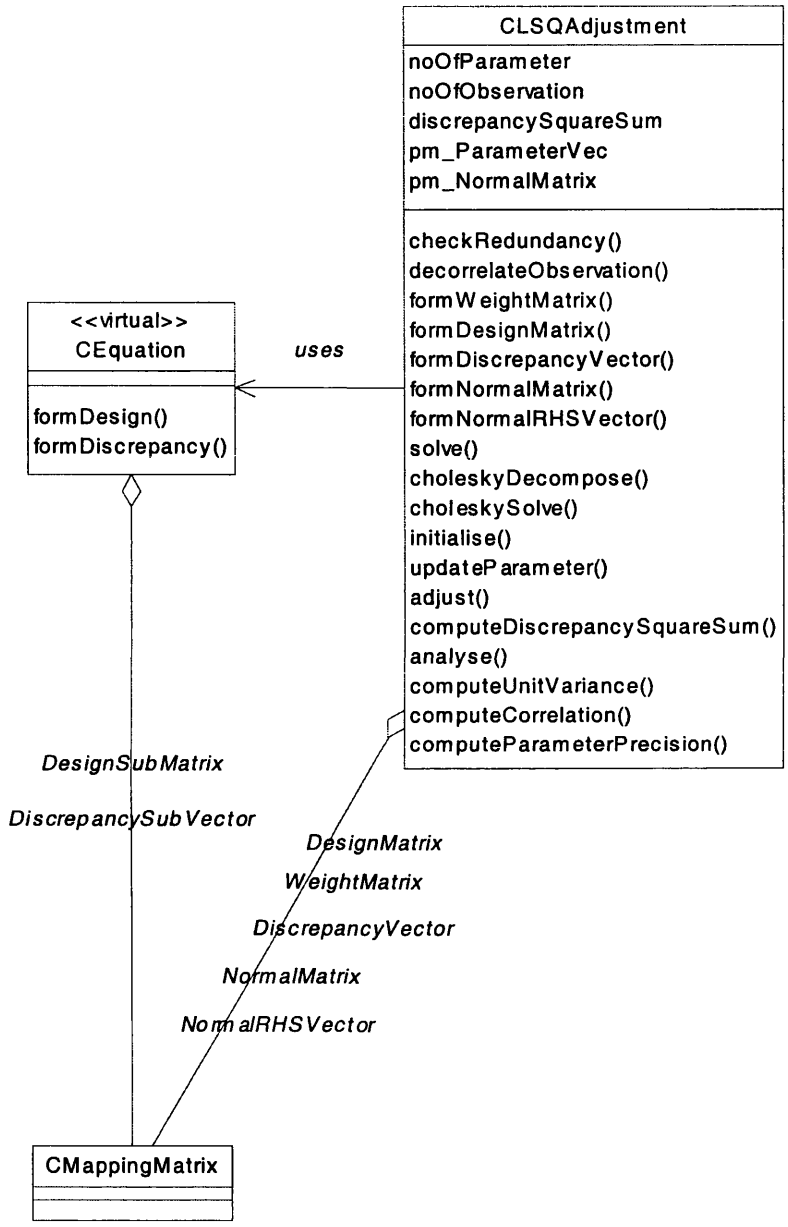

Figure 5.8 Class Diagram of the CLSQAdjustment Class

It uses the CEquation virtual class to form the normal equation from the observation equations. In the bundle adjustment it will use the CCollinearityEquation class and the

CDirectObservationEquation class. This can be done without any change to the CLSQAdjustment class because both of these classes are derived from the CEquation class.

Functions have been included which will solve the normal equation in various schemes. This has been done to make this class versatile and the user will have choice in selecting the one or more solution schemes mentioned below:

- apply standard matrix multiplication and inversion method.
- apply the Cholesky decomposition
- apply the storage optimisation
- apply decorrelation

5.3 The CMappingMatrix Class

In all of the proposed automated mapping subsystems, there is always some form of matrix computation involved in their processes. Therefore the matrix related class is a very important class and a well designed class should be reusable in many applications.

The basic responsibilities of a matrix class include:

- allocation of memory at its creation given the dimensions of the matrix
- returning the allocated memory address
- deallocation of memory on its destruction
- copying values of one matrix to another
- reporting the dimensions of the matrix

Other responsibilities which are closer to the actual matrix application involve functions which carry out various linear algebra computations. These computations can be programmed to use the familiar mathematical symbols, such as '+' and '-' through a function of the C++ language called 'overloaded operators'. This means that a matrix 'A' can be multiplied by matrix 'B' by simply specifying as 'A*B'. The function of the operator '*' is 'overloaded' to handle the matrix objects as it handles multiplication for integer or real value arguments.

The matrix class in this thesis followed the design proposed by Birchenhall [Birchenhall, 1993]. The next figure, Figure 5.9, shows the Class Diagram of the CMappingMatrix class and the aggregated class matMap which in turn aggregates another class called realArray.

The CMappingMatrix is where the actual high level matrix computation is handled. The basic responsibilities are handled by the matMap class and the realArray class. The matMap class performs the responsibility of mapping an element of a matrix, such as 'NormaMatrix (3,2)', to the actual storage memory address. The realArray class manages the basic functions of allocating and destroying memory for the real valued matrix elements.

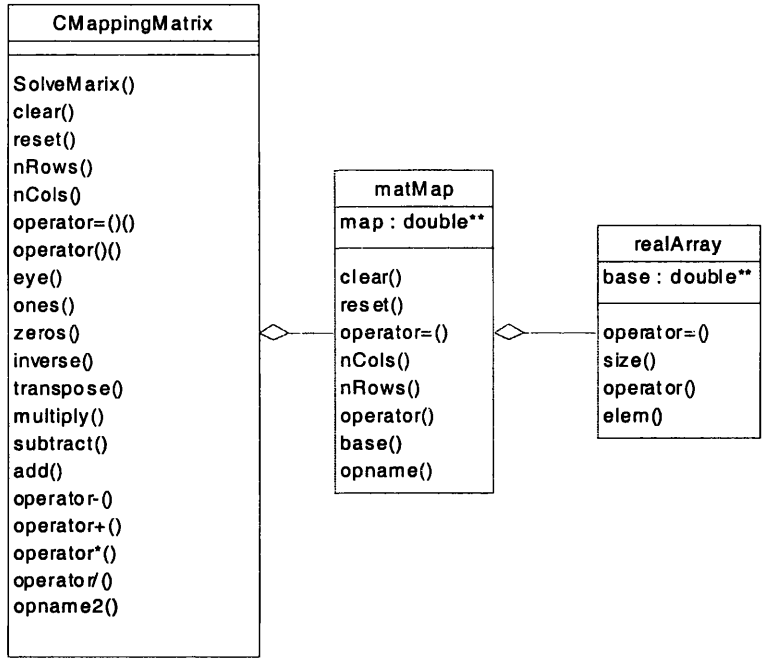

Figure 5.9 Class Diagram of the CMappingMatrix

This three-tiered structure facilitates the use of 'reference matrices'. Reference matrices are matrices which are created using the copies of the values of an original matrix. Therefore actual memory allocation for the reference matrices are not made, thereby saving valuable memory space and processing time.

5.4 Summary for Chapter 5

In this chapter, the classes of the Image Point Referencing Subsystem were designed after analysis of the photogrammetric process of image point referencing through the bundle adjustment.

In the analysis stage, a Use Case Diagram of the Image Point Referencing Subsystem was produced to give a general view of what this subsystem would be used for (Figure 5.1). An

interesting aspect of the diagram is that the Bundle Adjustment Use Case uses the Positioning Subsystem and the Image Acquisition Subsystem. These Subsystems are depicted as actors because they are external entities communicating with the Image Point Referencing Subsystem. It can also be seen that the Least Squares Adjustment Use Case is used by three Use Cases, namely Bundle Adjustment Use Case, Compute Initial Approximation Use Case and the Match Points Use Case. This indicates that the software component of the Least Squares Adjustment Use Case can be reused for different applications.

Identifying the potential for reuse is an important aspect of software design. It will not only make implementation simpler but it will also keep the software consistent during maintenance and further development. It can be said that potential for reuse of the Least Squares Adjustment was identified by taking an integrated approach of the whole mapping process.

The analysis and the design for the Image Point Referencing was performed at a very detailed level, and advanced knowledge and techniques used in the photogrammetry discipline were included in the design. Some of these techniques are the direct formation of the normal equations, inclusion of the direct observation equations, Cholesky factorisation, storage optimisation and decorrelation of correlated observations. Decorrelation of correlated observations is not used in the Image Point Referencing because the image observations are almost always assumed to be uncorrelated. However, it was included in this chapter because the least squares adjustment class design was treated in this chapter. Decorrelation will be valuable in dealing with correlated observations, such as the double difference equations of Chapter 4.

Two Sequence Diagrams were made to explain the procedures in the formation of the observation equations and the least squares adjustment. The least squares adjustment of the image observations in this case becomes the bundle adjustment. Classes and member functions were identified using these Sequence Diagrams.

The key class, CCollinearityEquation class was designed in detail. Although the diagram in Figure 5.5 depicted all the classes that are involved with the CCollinearityEquation class in the formation of the normal equations, it should be noted that only the CRay class interacts directly with the base class of CBundleEquation class. Classes such as CControlPoint, CImagePoint, CPerspectivePoint, CMappingImage, CImageSensor and CImageRotation are actually managed by the CRay class. This results in enhanced encapsulation and eventually high reusability of classes.

It should be emphasized that a deep understanding of software engineering skills as well as of domain knowledge are required for good results in software design.

The CLSQAdjustment class was design with reusability as its most important factor. It was designed to use CEquation class, which is actually a base class. Such design will enable the CLSQAdjustment to be used in most situations where observation equations of different application domains are derived from the CEquation base class.

Finally in this chapter, the CMappingMatrix class was introduced. Although simple in terms of complexity when compared to other important classes such as the CCollinearityEquation class or the CLSQAdjustment class, it a very important and fundamental class of the automated mapping class.

The classes designed for the Image Point Referencing Subsystem are shown in Table 5.1

Table 5.1 Classes Designed for Image Point Referencing Subsystem

Observation Equations related	Least Squares Adjustment related	Matrix related (adopted)
CBundleEquation	CLSQAdjustment	CMappingMatrix
CCollinearityEquation	CEquation	matMap
CDirectObservationEquation		realArray

6. IMPLEMENTATION OF DESIGN FOR CAMERA CALIBRATION AND GPS SURVEYING

Implementation of the design, i.e. programming in a selected programming language, transforms the UML Class Diagrams and Sequence Diagrams into program codes. During the design of various classes, the structures of the objects and the cooperation between related objects were the main focus. In the implementation phase, the inner functionality of individual objects is the main concern. The declarations of various classes are based on the Class Diagrams and the codes for the member functions are based on the messages of the Sequence Diagrams of Use Cases.

In this chapter, the classes designed in Chapters 3, 4 and 5 will be used to program the camera calibration application and the GPS surveying application. For each application the programming process will be explained followed by a description of the testing that was carried out with some sample data. The input data and results of these tests are included in Appendix A, Appendix B and Appendix C.

Source codes for the header files containing class interfaces are also included in Appendix D and Appendix E. Approximately 6,000 lines of programming code was written for the bundle adjustment program and about 8,000 lines for the GPS data processing program. The program was developed based on the design of the foregoing chapters. It should be noted that many iterations of analysis, design and implementation were performed over a period of more than two years to come to the present stage, even though the author was already knowledgeable in photogrammetry and programming at the beginning stage. The presented diagrams are the most current version of the software. More iterations of this software development are anticipated in the future. Program development for this research was done using Microsoft Visual C++ version 6.0.

6.1 Camera Calibration Implementation

The implementation of the camera calibration application will be explained in two parts. The programming of the design will be explained first, followed by testing of the developed program. The testing was carried out for:

- self-calibration for a digital camera (Kodak DCS260) owned by the University of Glasgow. A grid plate was used as the test field where the coordinates of the grid intersections were used as control points. These points were observed using an analytical plotter, and;

- bundle block adjustment of a data set made available from CIPA (International Committee for Architectural Photogrammetry), which includes 16 digital images of the Zurich City Hall building and 21 control points on the facades of the building.

6.1.1 Programming

For the implementation of the camera calibration application a new class called CProject was formed to handle the process of populating various objects of classes such as CImage and CControlPoint with data supplied by the data files. After the data are correctly placed in each object, they are then processed by the CLSQAdjustment class which handles the rest of the adjustment process.

During the programming process it is important to confirm that each member function is correctly implemented. In implementation using a procedural programming language, it is normal that a single function grows to be very large and this makes error tracing very difficult. This problem is alleviated in object oriented programming because each class is usually limited to a number of responsibilities and the member functions of each class are usually small and compact. If the number of member functions increases during implementation stage, changes should be made to the design and keep the number of functions in a class to a manageble size. In object oriented programming, the programmer is more focused on the implementation of the class under scrutiny, and is less distracted by other global influences. This is the advantage of the encapsulation characteristics of the object oriented methodology.

Verification of each module is important before performing any integrated test. In programming, early verification of the implemented code is very important because, as the size of the program increases, what may seem as a very simple mistake in the beginning may prove to be very costly, because of the time and effort in locating that mistake at a later stage.

In this research, many of the member functions in each class which require matrix computations were first implemented in a mathematical software package, Matlab, for quick and reliable solutions. Because the size of the member functions is small and the test data are also small in scale, a student edition of the Matlab software proved to be sufficient. The source code in Matlab programming language was then translated into C++ and the results from the two computations were compared.

Reliable test data are important for the verification process. It is also very difficult to get them. Simulated data are one solution but one must be careful in producing simulated data. If the simulated data are produced by the same programmer, then there is always the danger

that the simulated data themselves may be produced under a false assumption also found in the code implementation, and this may result in the test turning out to be perfect, when in fact it is not. Further development based on this false verification may prove to be very frustrating because it is now even more difficult to locate the error in the following stages. For verification of the developed bundle adjustment program, examples available from text books [Kraus et al., 1993] and a program manual [Kruck, 1986] were used as test data.

A program may go through major restructuring during the implementation phase, whether because the design was not as complete as it should have been or because the requirement was not foreseen at the design stage. In the implementation of the bundle adjustment program in this research, it was decided at a very late stage of implementation that an approximation program should be included to provide for the initial approximation necessary in the bundle adjustment. This proved to require a trivial effort, compared to the initial effort. A new class called CDLT was created and, using all the other existing classes implementation of the DLT (Direct Linear Transformation) to provide for the initial approximation was quite easy. This was because of the portability and the extendibility of existing classes made possible by the encapsulation characteristics of the object oriented methodology.

For testing the implementation, some programs were created which did not use the classes designed in the foregoing chapters, namely a simple graphic user interface and digitising software to collect the image coordinates.

Graphical User Interface (GUI) Implementation

A simple graphical user interface was implemented using the Microsoft Foundation Class (MFC). At the initial execution of the bundle adjustment program, three text data files are opened ("gubaio.img", "gubaio.gcp" and "gubaio.cam") from the subdirectory "infile" of the current directory. After carrying out the adjustment, the result of the adjustment is saved to a text file, named "gubaio.out", and this file is stored in another subdirectory called "outfile". (Details of the format of these data files are explained in Appendix A and Appendix B.)

At the end of the adjustment, a new window showing this result file is opened. The user can edit the data files as required from the opened windows of these files. The initial window is shown in Figure 6.1.

From the initial state, the adjustment is initiated by clicking the submenu item called "Adjust", upon which a dialog box is shown on the screen directing the user to make the following selections (Figure 6.2):

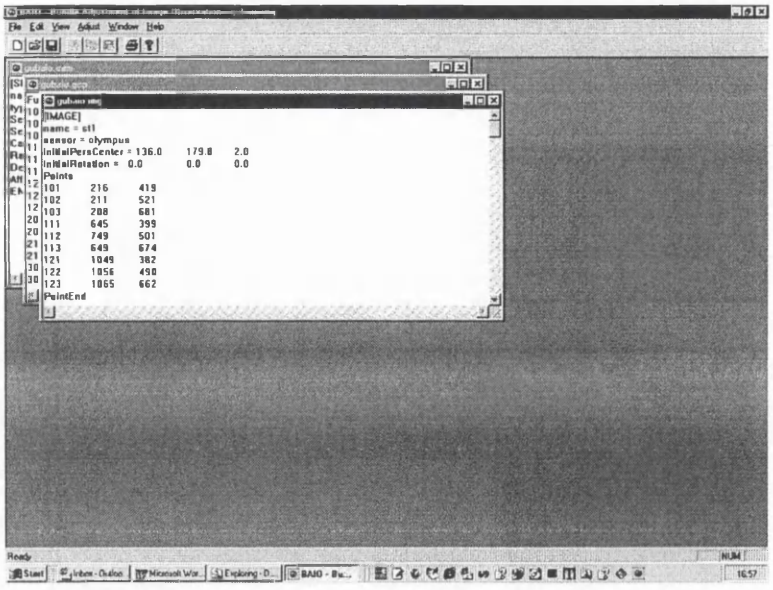

Figure 6.1 Initial Window of the Bundle Adjustment Program

- Bundle adjustment with or without self-calibration adjustment

  For the self-calibration case, sensor parameters are included in the normal matrix, whereas, in the normal bundle adjustment case they are not.

- Aerial platform or terrestrial platform

  Different rotation matrix and different partial differentiation values are added for each case respectively. For the terrestrial case, the rotation is in the order z-axis, x-axis and y-axis because, the imaging sensor being horizontal with the z-axis, the rotation matrix will not form a positive definite normal matrix, making the solution impossible.

- Choice of angular units

  Angular data can be in radians, degrees or GON. Using this selection the program will change the input data into radians, its working unit, and then compute back to the selected unit for the output.

- Maximum number of iterations

  The maximum number of iterations can be entered into the edit box provided. Sometimes when the initial approximation is within the convergence limit, but not good enough to converge within a few iterations, a large number such as 10 or 15 can be typed in.

- Test of convergence value

The computed unit standard deviation for each iteration is compared against this test of convergence value and then breaks the iteration if the computed value is smaller.

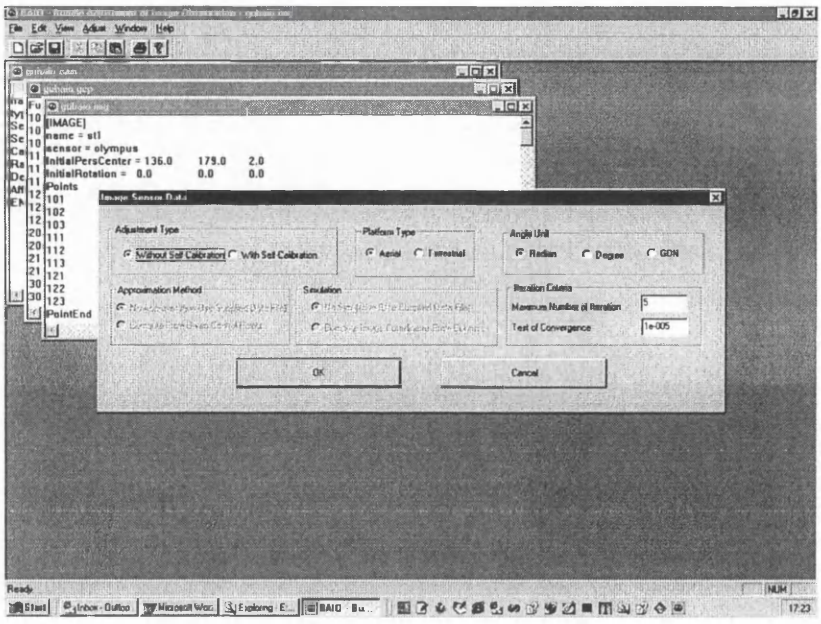

Figure 6.2 Dialog Box1 for Setting the Adjustment Environment

A second dialog box appears when the "OK" button of the first dialog box is clicked (Figure 6.3).

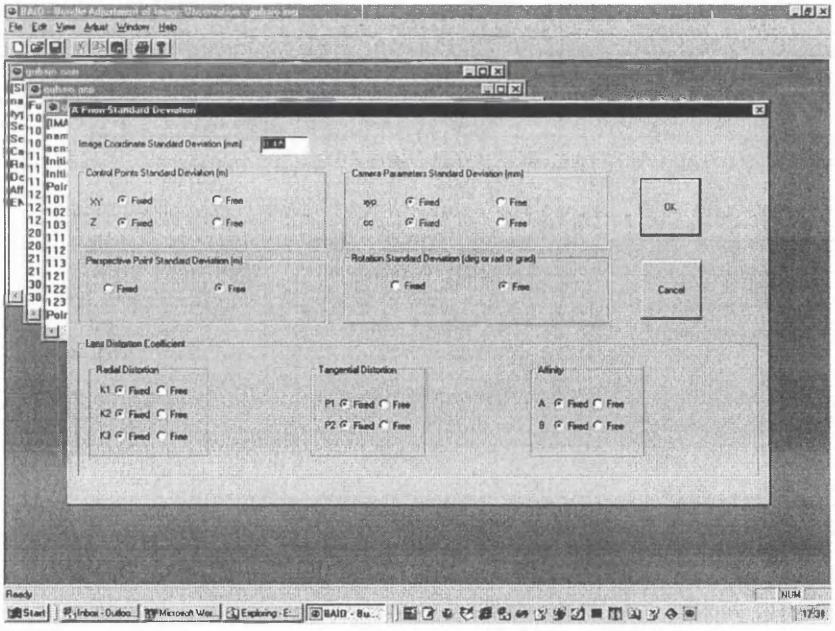

Figure 6.3 Dialog Box2 for Setting the Adjustment Environment

This dialog box is used to define the following factors:

- The estimated accuracy of the image observation in millimetres.
- The accuracy of the control points

If "Fixed" is chosen, a large value is assigned to the weight for the ground control points making the control parameters act as constants. If "Free" is chosen, a small value is assigned making them act as unknown parameters. If a known standard deviation value is added in the control point file, "gubaio.gcp", this selection is disregarded and the standard deviation is chosen to compute the weight for each control point. This applies to all the subsequent parameters of imaging sensor parameters, perspective point coordinates, rotational angles and the additional parameters of lens distortion and film shrinkage.

Finally, after the completion of the adjustment a new window is opened showing the result of the adjustment. The result can be saved under a chosen name and another adjustment can then be processed under a different adjustment environment.

Digitising software implementation

Digitising software is necessary to capture the image coordinates (i.e. the pixel coordinates) of the ground points (i.e. control points and tie points). Although for an integrated automated mapping system a very sophisticated image handling system has to be designed and implemented, for the purpose of testing the implementation in this research, rather simple digitising software was implemented. Its main role is to open an image file and record the pixel coordinate as pointed to by the cursor and selected by clicking the mouse. A marked circle was made to show on the screen the points which had been recorded, as shown in Figure 6.4.

Figure 6.4 Marking for Visual Checking of Registered Points

The pixel coordinates are then recorded to a file called "imgxy.txt" together with image information.

6.1.2 Calibration Testing

A calibration test was carried out for the digital camera Kodak DCS260 that is owned by the University of Glasgow. The photo-site was reported by the manufacturer to be a square one of 4.85 micron in height and width. Since the number of pixels in the images produced by the camera is 1024 (H) x 1536 (W), the dimension of the sensor chip was computed as 4.9964 (H) x 7.4496 (W) cm. An initial set of values of (0,0) for the principal point coordinates and 8 millimetres for the principal distance were used as the initial approximation data.

The control grid plate used for this testing was prepared by a mapping company in Korea called Air Korea Company. For control points, a grid of 22 horizontal and vertical lines which were one centimetre apart, was drawn on plastic transparency plate with a drafting machine. Each intersection of all the grid lines were then observed using an analytical plotter with the coordinate origin set approximately at the centre of the plate. The X and Y coordinates are the readings from the analytical plotter and Z is set as zero for all control points.

Digital images of this plate were then captured from various locations. Care had to be taken as to the orientation of the plate, with regard to the coordinate system used by the analytical plotter. The camera focal length was changed from auto mode to manual mode. As the images were taken indoors, the image quality was quite poor, as can be seen in Figure 6.5.

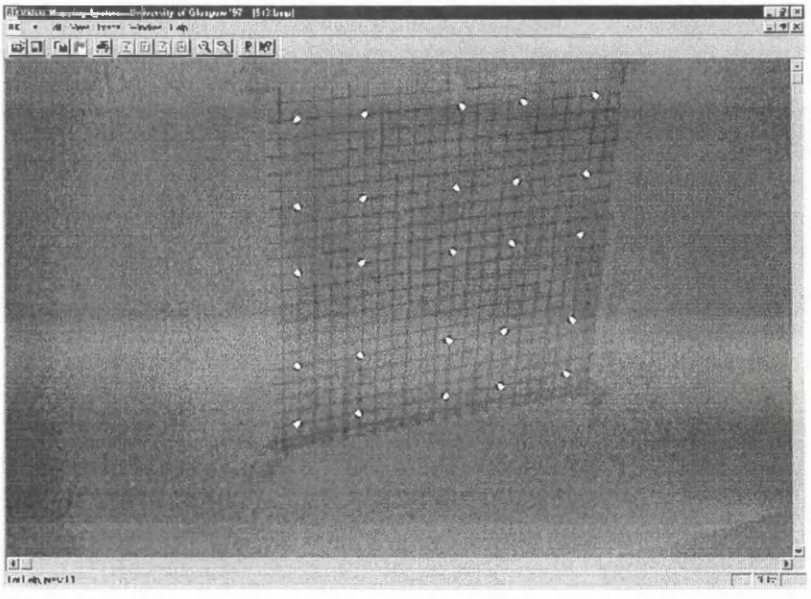

Figure 6.5 Image of the Control Points Grid Plate

Only 25 points were selected, and image coordinates of these points were acquired using the digitising program. Through visual judgment, the pointing accuracy could be defined as being approximately 3 - 5 pixels.

In the adjustment environment setting, the 'With Self-Calibration' option was selected and the platform was set to 'Aerial Mode' (Figure 6.2) because the control point coordinates are parallel to the camera. The angle unit was selected as 'Degree' because the approximation was given in degrees, and the iteration was increased to 15 to confirm any convergence. In the second dialog box (Figure 6.3), all parameters were set to 'Free' except the control points parameters. The result of this adjustment reported the principal coordinates as (0.013, 0.213) and the principal distance as 10.193 millimetres. But the correlation matrix showed high correlation between the principal point coordinates and the tangential lens distortion parameters (See Table 6.1).

Table 6.1 Correlation Matrix of Adjusted Parameters

	xp	yp	cc	K1	K2	K3	P1	P2	A	B
xp	1.00									
yp	0.01	1.00								
cc	0.31	0.30	1.00							
K1	0.20	0.11	0.39	1.00						
K2	0.00	0.00	0.00	0.00	1.00					
K3	0.00	0.00	0.00	0.00	0.00	1.00				
P1	**0.83**	0.10	0.02	0.16	0.00	0.00	1.00			
P2	0.10	**0.89**	0.08	0.09	0.00	0.00	0.11	1.00		
A	0.16	0.30	0.25	0.10	0.00	0.00	0.08	0.14	1.00	
B	0.27	0.09	0.07	0.09	0.00	0.00	0.13	0.01	0.03	1.00

This suggests that more variations should have been given to the distances of the images to the object and the rotation angles between images. Another possible explanation is that the grid is too small and the control points are clustered around the centre of the image. Evenly distributed control points covering the whole image would produce better results.

The details of the result of the adjustment as well as the input data files can be found in Appendix A. A summary of the adjustment result for the calibration is shown in Table 6.2

.Table 6.2 Summary of Adjustment Result for DCS260 Calibration

Number of Images	5
Number of Full Control Points	25
Number of Check Points	0
Number of Tie Points	0
Execution Time	2.03 seconds
No of Iterations	7
Unit Standard Deviation of Observations	0.004 mm
Principal Distance	10.193 mm
Principal Point Coordinate (x,y)	(0.013, 0.213) mm
Max Residual	0.016 mm for Point 14 at Image 5

6.2 Block Bundle Adjustment Testing

Bundle block adjustment was carried out with digital images captured with a digital camera, Olympus D1400L. The images were captured by members of the CIPA organisation. Details of the project can be found at the CIPA web page [Streilein et al., 1999]. The configuration of the image is shown in Figure 6.6. The West facade (image captured from St1) and the North-West view (image captured from St3) of the building are shown in Figure 6.7.

For the bundle block adjustment test of the program with this data set, many tie points (or new points) were needed. Their image coordinates were measured from images by selecting an appropriate point then numbering the point.

The digitising of the control points to capture the image coordinates was a simple task, but the process of handling the tie points was quite tedious and error prone. In aerial photography, selection of tie points is a relatively simple task because vertical photography is assumed in most cases.

With the terrestrial case, rotations occur in all directions. This makes planning of control points distribution difficult, and it is also difficult to select tie points from overlapping images because the overlaps are in many cases irregular. This is because photography is constrained by limited space and other obstacles. Also considering the fact that one of the main objectives of terrestrial photogrammetry is the three dimensional reconstruction of an object, it is normal that images are taken from all directions. In the Zurich City Hall case, an

image containing the whole building was available from the West side, but on the North and East side, images were taken from close ranges and only parts of the building were imaged, making inclusion of tie points necessary.

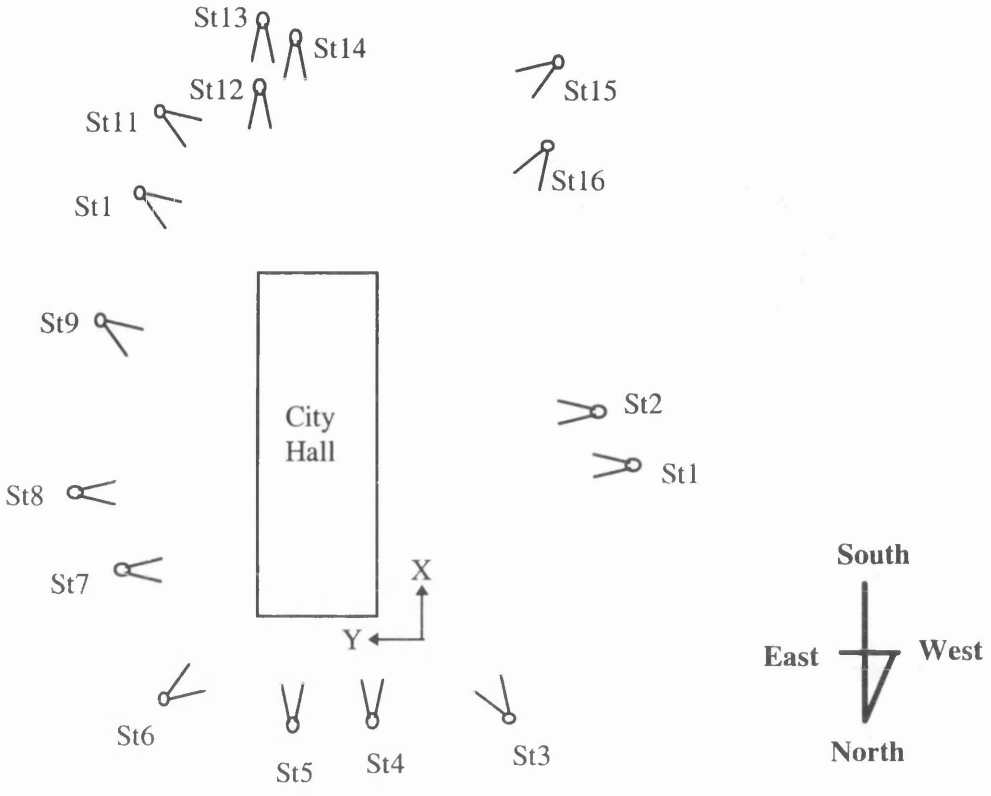

Figure 6.6 Camera Stations Layout for the Zurich City Hall Reference Data Set

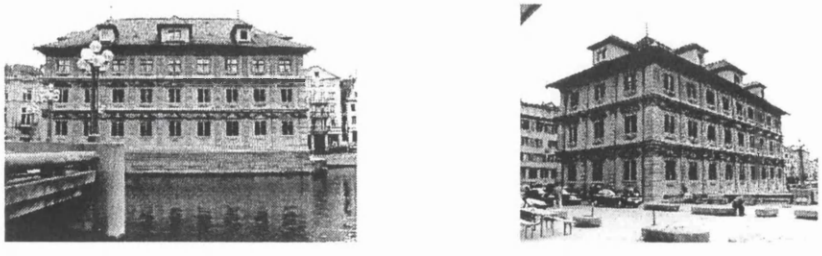

Figure 6.7 Images of the Zurich City Hall from St1 and St3

The image coordinates were captured and the bundle adjustment was carried out with the given camera data and the control point coordinates. In the adjustment dialog box, the platform for this case was selected as 'Terrestrial' because the camera was horizontal to the Z-axis. For the terrestrial case, the primary axis is the Z-axis (Alpha), the secondary axis is

the X-axis (Zeta) which is rotated 90^0 to bring the camera upright. In this program, 90^0 is implicitly added to the secondary rotation and all input and output data reflect the deviation from the horizontal position. The tertiary axis is the Y-axis as seen from the other negative direction (Kappa) [Kraus, 1997].

The control points and the sensor parameters were held fixed and the data were solved for the image parameters and tie points coordinates. Tie points coordinates get very large errors by default and are solved as unknown parameters. The middle row of control points on the West facade, No. 111. No 112 and No. 113, were used as check points. They were also given large errors to be solved as unknowns. The adjusted coordinates of the check points are then compared with the ground surveyed coordinates.

The details of the result of the bundle block for these images as well as the input data files can be found in Appendix B. A summary of the result is shown in Table 6.3.

Table 6.3 Summary of Adjustment Result for the Zurich City Hall Images

Number of Images		15
Number of Full Control Points		18 (3 used as check points)
Number of Check Points		3
Number of Tie Points		27
Execution Time		11.92 seconds
No of Iterations		10
Unit Standard Deviation		0.072 mm
Max Residual		0.5 mm for Point 202 at Image 6
Check Point RMS	XY	0.45 m
	Z	0.27 m

6.3 GPS Surveying Implementation

For the implementation of the program for processing GPS surveying data, all the classes of the Positioning Subsystem (Chapter 4) were used. The programming will be explained in the first part of this section then the adjustment and the result of the testing will be explained.

A test was carried out for a baseline where static GPS observations were made at both stations. Phase data processing for the baseline of these two stations was carried out. As single epoch data processing is carried out, each computation should result in the same baseline length, because both of the receivers were stationary.

6.3.1 Programming

As in the implementation for the calibration case, a new class called CProject was created to handle the reading of data from the observation files. Programming for the GPS data processing was relatively quick, compared to the initial effort of the calibration application, in terms of coding, because many of the classes involved in various adjustment tasks had already been implemented and these could be reused without very much editing.

For the reading of data files, the observation and navigation files for a single baseline in RINEX format should be placed in a subdirectory called "infile". It should be named with three letter extensions with the first two letters of the extension referring to the year of observation and the last letter being an "n" or "o" depending on whether it is a navigation file or an observation file. For example, a set of data files could be named as "test095a.99n" and "test095a.99o". This name contains the information that these data are about a project or a station called "test" which were observed on the 95th day of the year (5th of April) 1999 for session a.. Session refers to a set of observation that are to be processed in one adjustment computation. Usually a session is a set of data which were collected without any long interruption.

The developed program automatically looks for the last letter in the file extension and populates the orbit objects with the relevant navigation data and the space vehicle objects with the relevant observation data of pseudoranges in metres and phases in cycles.

After having read all the data in the navigation header and the navigation data, the program proceeds to read the observation header of the two files collected from the two stations. At this point the program synchronises the two file pointers (one is assigned as the reference station file and the other as the rover station file) by first taking the first record of the file which has the earlier observation time and defining this time as the reference time. The file

pointer for the other file proceeds through the records until it finds this reference time. The files are now defined as synchronised and observation pairs are processed sequentially. This algorithm will make the extension of the present post processing program to handle real-time processing easier because the data are being processed record by record from the data files, which is a similar scheme as in real-time processing where the data from the receivers are transmitted to the computer epoch by epoch.

Unlike the calibration application, GUI programming for the GPS surveying application is more complex. There is not much editing to do with the data files; however a visual representation of the geometry and the availability of the space vehicles, indications of cycle slips, and inclusion and exclusion of parts of observation data are some of the requirements necessary to make a reliable adjustment. Due to its complexity, it is deemed that a systematic GUI class for GPS data processing should be designed first before an attempt is made to implement it.

Implementing Wide Lane technique

Although not included in the design of the positioning subsystem, it was decided that the wide lane technique would be useful in the subsequent GPS data processing. The wide lane signal is defined as the difference of the L1 phase data minus the L2 phase data as shown in equation (6-1).

$$\Phi_{\omega} = \Phi_{L1} - \Phi_{L2} \qquad (6\text{-}1)$$

The frequency of this signal is 347.82 MHz and the corresponding wavelength λ_{ω} is 86.2 cm. The longer wavelength, as compared to the 19.0 and 24.4 cm of the L1 and L2 signals, makes it easier to solve for the integer ambiguity, by reducing the number of the search candidates. The constraint is that both L1 and L2 signals must be captured, which is possible with only the double frequency capable receivers. But because of the advantage of being able quickly to resolve the integer ambiguities, it is a technique that many are adopting in GPS phase data processing.

The inclusion of the wide lane signal to an existing design was a simple task. A set of data members and a member function relevant to the wide lane technique were added to the existing CDoubleDiff. This is shown in bold letters in Figure 6.8.

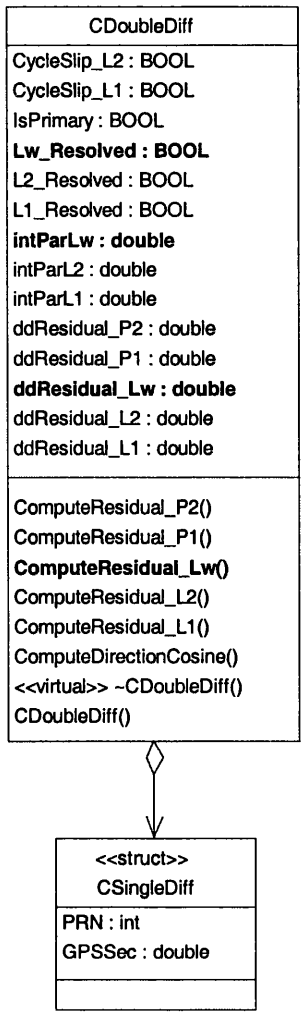

Figure 6.8 Wide Lane Members Added to the CDoubleDiff Class

It is thought that although this was a simple task, from the view of the Object Oriented method, using the polymorphism features in the design would prove to be more beneficial in terms of portability and extendibility of the software. The improved design should have a double difference base class, for example called CGenericDD, and a separate class derived for each signal, L1 ,L2 and Lw, called for example CDD_L1, CDD_L2 and CDD_Lw, as shown in Figure 6.9.

The base class would contain the virtual function to compute the residual which could be called 'ComputeResidual', without any convention for specific signal. In each of the derived classes, there would be an own version of the ComputeResidual function and, depending on the signal type, the correct version of the ComputeResidual function will be evoked.

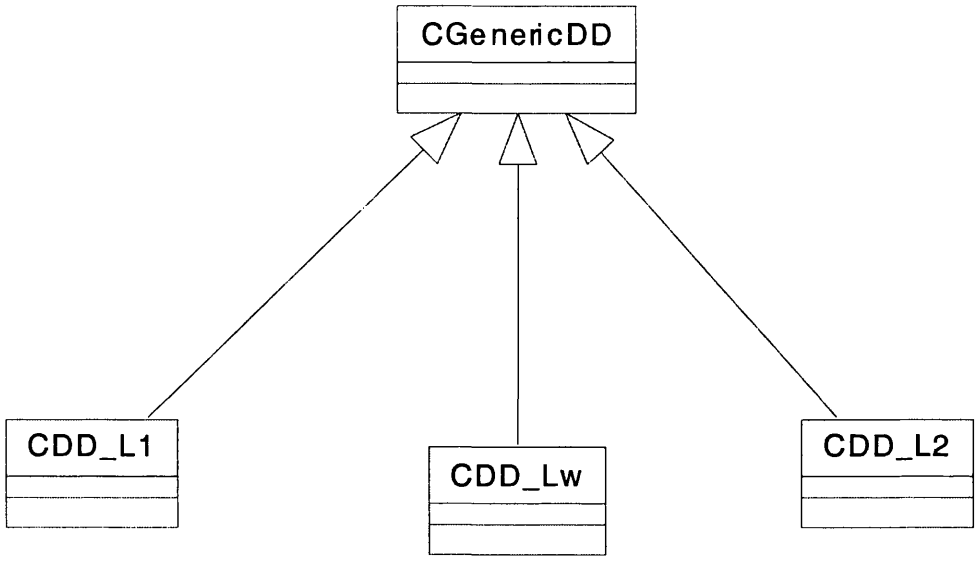

Figure 6.9 Design of a Polymorphic Double Difference Class

It is important to note that design and implementation are part of an iterative process. It is almost impossible to come up with a perfect design which will be implemented without any changes to the original design. New information and new ideas which develop during the programming phase bring about changes to the original design. Taking the above case of the polymorphic double difference class, if it was decided during the implementation phase that indeed the polymorphic double difference class was a better choice, and it was implemented by the programmer directly without going through the design process, there would be inconsistency between the design and the implementation. It is therefore now necessary to update the design to match the program. If one person is performing both the design and the implementation, designs could be updated to match the implemented program as changes occur during the implementation. But if many team members are involved, new proposals and information with respect to the design should be systematically incorporated to the design at the next cycle of the design and implementation iterations.

6.3.2 Testing

For testing the GPS implementation, a set of data from a GPS software company, TerraSat, was used [Terrasat, 1999]. The data were collected at 10 seconds interval and from two stations in static mode.

The result of the adjustment is included in Appendix C. This result is shown graphically in Figure 6.10.

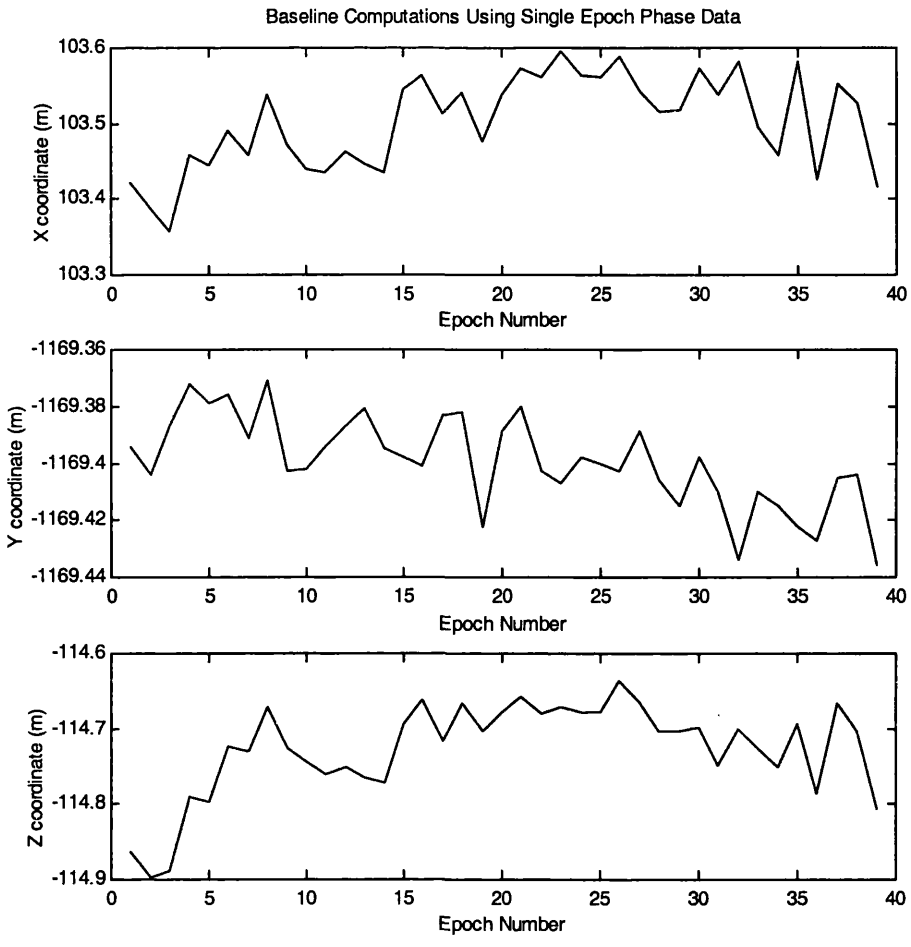

Figure 6.10 Adjustment Result of GPS Adjustment

A variation of about 30 centimetres in X, 8 centimetres in Y and 30 centimetres in Z in the baseline vectors, can be seen from the graphs. The large error could be because the atmospheric refraction and space vehicle clock error were not included in the model. As explained in Chapter 4 (Figure 4-14), Ambiguity Function Values are computed to reject integer ambiguities set candidates whose values are beyond a selected criteria. Figure 6.11 shows the computed Ambiguity Function Value (AFV) for each candidate of integer ambiguities set of the first epoch data. There are comparatively few candidates because this is a wide lane signal. For the L1 signal it is normal to have more than one thousand candidates. The selection criteria reduces the number of candidates even further by selecting candidates only above the selection criteria line and thus speeding up the computation.

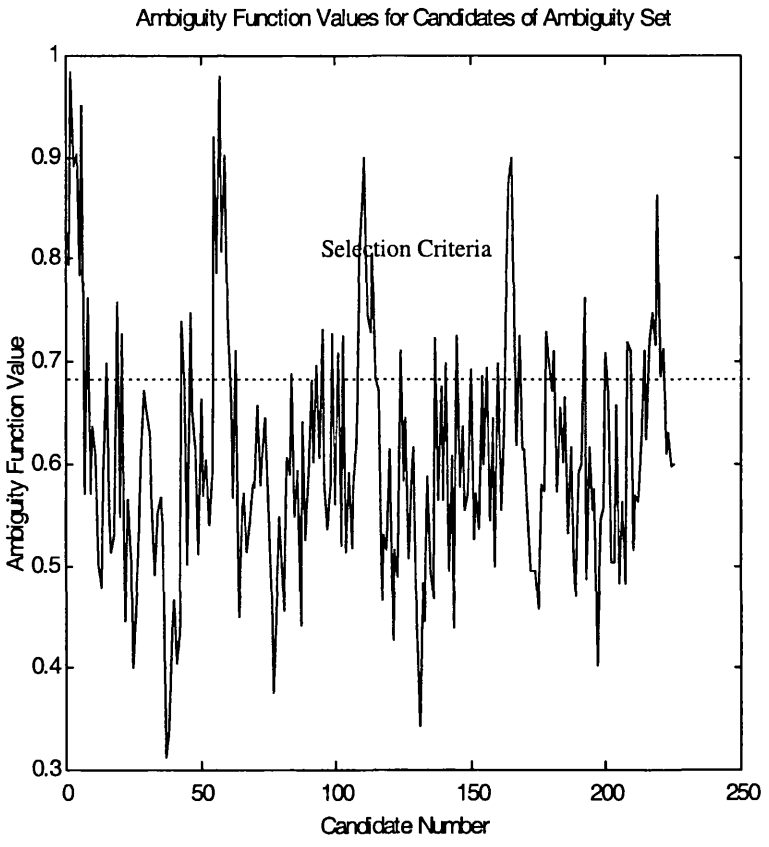

Figure 6.11 Ambiguity Function Values for Candidates of Ambiguity Set

As can be seen in Figure 6.11, if the selection criteria is set to be 0.7, much of the erroneous candidates below the dotted line are disregarded in further computations and only those which are more likely to the correct answer, shown above the line, are considered for further rigorous testing.

6.4 Summary of Chapter 6

In this chapter, implementation of the designed classes in two programs (i.e. bundle adjustment and GPS data processing) and the testing of the developed programs were explained.

Experiences acquired while implementing the designed classes were explained. Some of these experiences are :

- classes should be designed to be of manageable size, in terms of the total number of member functions.
- each member function should be verified immediately upon completion before attempting to implement another member function.

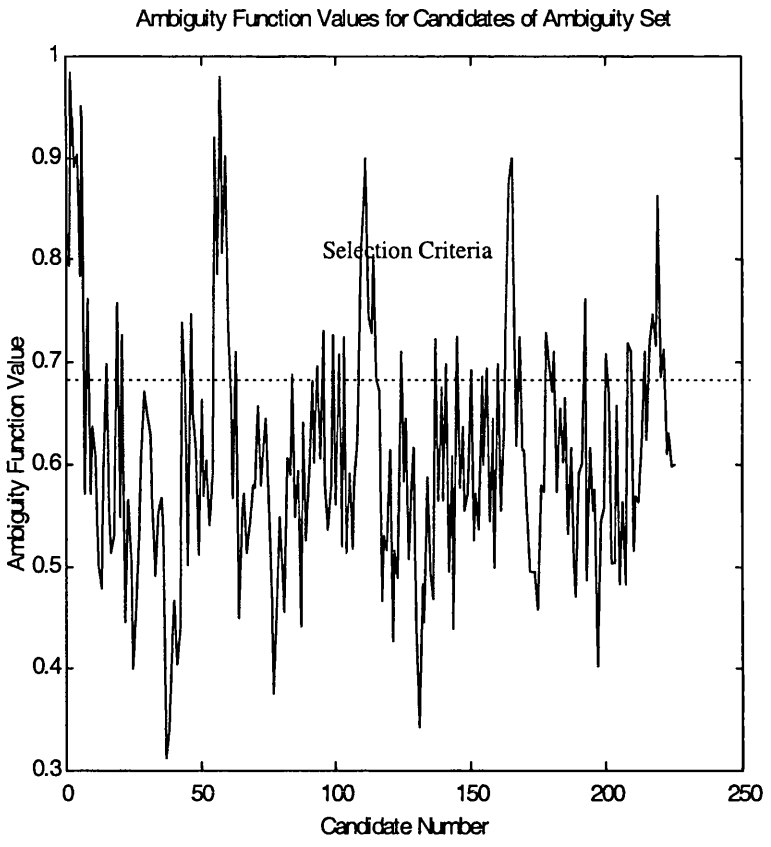

Figure 6.11 Ambiguity Function Values for Candidates of Ambiguity Set

As can be seen in Figure 6.11, if the selection criteria is set to be 0.7, much of the erroneous candidates below the dotted line are disregarded in further computations and only those which are more likely to the correct answer, shown above the line, are considered for further rigorous testing.

6.4 Summary of Chapter 6

In this chapter, implementation of the designed classes in two programs (i.e. bundle adjustment and GPS data processing) and the testing of the developed programs were explained.

Experiences acquired while implementing the designed classes were explained. Some of these experiences are :

- classes should be designed to be of manageable size, in terms of the total number of member functions.
- each member function should be verified immediately upon completion before attempting to implement another member function.

- reliable test data should be used for the verification and care should be taken with simulated data.
- mathematical software, such as Matlab, is very useful for quick prototyping member functions in the implementation phase of software development.
- updating of design and implementation should be systematically carried out to ensure consistency in the design and the implementation.

The completed bundle adjustment program was tested for two applications: camera calibration and block bundle adjustment.

The camera calibration testing was carried out on a digital camera using images of a grid plate. The coordinates of the grid plate were observed using an analytical plotter. The result of this test produced results with some correlation between parameters. The probable reason for this was explained as being due to lack of strength in the geometric configuration of observed points in the images.

Block bundle adjustment was also carried out with digital images of Zurich City Hall. These were provided by the CIPA organisation. New requirements for terrestrial images were discovered during this testing, especially regarding the inclusion of tie points in terrestrial images. Different aspects of the aerial images should be taken into account when automating image point referencing of terrestrial images.

The GPS data processing program was implemented using the design classes in the third section of this chapter. Post processing of phase measurements of a baseline collected in static mode was used as the test data. Among the various methods of processing phase measurements, the single epoch method of resolving integer ambiguities used in kinematic mode was selected.

One of the most important aspects of the Object Oriented method, namely reusability of software, was confirmed in this implementation. Classes of least squares adjustment and matrix which had been already implemented in the bundle adjustment implementation were reused for this application with few changes. Extendibility was also confirmed when the wide laning technique was incorporated at a later stage of implementation, although this was not designed in the initial phase.

The CDoubleDiff class, whose task is to form double differenced equations, proved to be a very useful class. Forming the double differenced observation equation and performing the least squares adjustment was simplified by using this class. Simplification of complex matters is the chief aim of Object Orientation.

The results of the test are shown graphically in Figure 6.10. The variations of each epoch reflect the errors of the computation for each epoch. This is because kinematic mode was applied to two static stations. In theory there should be no variations for all the epochs. Errors amounting to 30 centimetres were observed which could be due to unmodelled errors in the computation.

The implementation of the designed classes confirmed some important aspects of the design. Although successful implementation was one of the objectives (i.e. the external view as mentioned in Section 2.2), the emphasis was on investigating the validity of the designed classes, the reusability of the classes in various applications, the extendibility of classes and effective maintenance of the software in terms of consistency of the design with the implementation (i.e. the internal view). All these aspects of the Object Oriented design were confirmed through this implementation effort. It should be noted however that software design is an iterative process which changes constantly, although the changes should be minimal after maturity has been achieved.

7. CONCLUSIONS AND RECOMMENDATIONS

In this research, an integrated approach was taken for the design of an automated mapping system. Past research by different sub-disciplines of the Geomatics discipline has focused only on specific tasks of the mapping processes. Photogrammetrists focused on camera calibration, aerial triangulation, and automatic object recognition from images. Surveyors and geodesists mainly carried out research in GPS and IMU data processing. Cartographers focused their activities on GIS modeling and visualisation. These dispersed efforts, although appropriate for gaining a deeper understanding of each sub-discipline, were not very effective in terms of software development for the whole mapping process. The integrated approach of this study was taken with an ultimate aim of producing a software design that automates the mapping process, from the initial data acquisition to the visualisation of geospatial data.

7.1 General Conclusions

The objective of this research was to produce a software design prototype for the automated system which is:

- accurate in terms of meeting the user requirements;
- rapidly implementable; and,
- flexible in terms of extendibility, compatibility and reusability.

To this end, the conventional mapping process was analysed. An automated mapping system was defined and then structured into the Image Acquisition Subsystem, Positioning Subsystem, Image Point Referencing Subsystem and the Visualisation Subsystem.

The Object Oriented design methodology was applied to the design of these subsystems (except for the Visualisation Subsystem) and UML notations were utilised to produce Use Case Diagrams, Sequence Diagrams and Class Diagrams. Just as architectural design blueprints reflect different aspects of a building, these software design drawings serve to explain to a domain expert, a software designer or a programmer the different aspects of the automated mapping software artifacts.

From Chapter 3 to Chapter 5, narrative explanations of the domain knowledge of photogrammetry and geodesy were complemented with Use Case Diagrams and Sequence Diagrams. Such diagrams help the domain experts to understand the transition of their

knowledge to 'classes' and 'objects' and eventually to Object Oriented program codes, whilst the software designer will find them helpful in understanding the domain knowledge.

To verify and evaluate the design, the designed classes were then implemented in three applications: the Image Acquisition Subsystem (Camera Calibration), Image Point Referencing Subsystem (Bundle Block Adjustment) and Positioning Subsystem (GPS data processing). These implementations were each tested with test data. Some general conclusions are given below regarding the design and the implementation of an automated mapping system.

The table below shows the current development status of the automated mapping system proposed in this thesis.

Table 7.1 Development Status of the Automated Mapping System

Stage of Development	Image Acquisition Subsystem	Image Point Referencing Subsystem	Positioning Subsystem	Visualisation Subsystem
Analysis	Complete	Complete	Complete	Incomplete
Design	Complete	Complete	Complete	Not initiated
Implementation	Incomplete	Complete	Incomplete	Not initiated
Testing	Incomplete	Complete	Incomplete	Not initiated

For the Image Acquisition Subsystem, the analysis and design has been completed and the implementation of the camera calibration Use Case has been implemented and tested but more implementation and testing with regard to flight planning and monitoring and controlling the imaging sensor are incomplete. Analysis and design of flight planning and monitoring needs access to aircraft to be able to understand the true nature of the job, which would have been quite difficult to undertake as part of this thesis. This has not been pursued further for this reason but also, it is not very significant in terms of software complexity. With the GPS related classes having been already implemented, the implementation of the rest of the Image Acquisition Subsystem is deemed to be a relatively simple matter for the future.

The Image Point Referencing Subsystem has been completed through the implementation and the testing of bundle block adjustment.

For the Positioning Subsystem, Process GPS Data Use Case has been completed but the other half of Process IMU Data, which includes mechanisation of IMU data and Kalman filtering, has only been analysed and designed. The Process IMU Data Use Case is incomplete at this stage. Lack of resources during the study period was the main reason that the implementation and testing of this Use Case has not been completed. The limited resources available were invested in the acquisition of knowledge through a study visit rather than acquiring the IMU. It was through this study visit that the analysis and the design of the Process IMU Data Use Case was made possible. Based on this design, it is anticipated that the implementation and the testing of this Use Case could be performed successfully.

The Visualisation Subsystem was considered and mentioned in various parts of the thesis as considerations about this subsystem were made in conjunction with other classes (e.g. Figure 3.7 in Chapter 3) but the analysis is incomplete and the subsequent development stages of design, implementation and testing have yet to be initiated. The reason for so little attention being given to this subsystem is that visualisation is a subject much researched by both the cartographers and computer image specialist, although with slightly different aims. It is a well established subject and published materials on Object Oriented designs and codes are readily available. Therefore, more attention was given to other subsystems which needed more individual creative effort to produce an Object Oriented design.

General conclusions regarding the design

- **Object Oriented Design is essential for research activities in the Geomatics community**

  It was mentioned in Chapter 1 that this study has been undertaken from a Geomatics point of view. This is re-iterated here, to emphasize that the software design produced in this study is to enable the domain experts, photogrammetrists, geodesists, cartographers to focus more on the domain research and spend less of their resources on the programming aspects.

  The Object Oriented design of the automated mapping system produced in this study has been found to be very effective in reducing the development time through its reusable classes. As an example, in the GPS data processing program, about 8,000 lines of code were implemented with much of the program using the same codes developed for the bundle adjustment program. So with a good design serving as the basis, the size of the program in terms of number of lines of code is no longer an important issue.

Object Oriented software designs should be produced and maintained by research organisations in the Geomatics community enabling them to focus more on domain research.

- **UML notations and diagrams are effective in software design and software maintenance of the automated mapping system**

  In this study, the author did the investigating, the analysis, the design and the implementation. Due to this, there was little inconsistency in progressing from the analysis phase to the design and then on to the implementation phase. Even so, as the program grew larger, changes were inevitable and numerous in many stages of the design and implementation. Difficulties were experienced in understanding the author's own code when coming back to a part of the program which had been left untouched for a long time.

  In a much more difficult situation, where different software designers investigate different domains and then pass the designs to programmers who have little or no understanding of the domain problems, a lot of misunderstanding can be expected. The UML will prove to be crucial in these situations by effectively relaying the ideas and concepts of the designer to the programmer.

  It can be envisaged that in the future, an expert in Geomatics will produce the result of his research in UML diagrams and will be able to produce a working program, almost instantly, and then incorporate it into an existing automated mapping system.

- **Integrated software design of an automated mapping system will bring synergism to research in Geomatics**

  Cooperation between different domain fields is a very difficult task. This could be because of the different levels of understanding in certain subjects, different objectives and different terminologies.

  The integrated automated mapping system could present a common ground on which all these incompatibilities between different sub-disciplines dissolve. Each sub-discipline will not only use its respective subsystem of the automated mapping system but also other subsystems because they are easily accessible.

  The use of the integrated automated mapping system will make the researchers of a specific sub-discipline more aware of other sub-disciplines, thereby resulting in better coordinated research activities cooperation.

General conclusion regarding the implementation

- **The Object Oriented software design of this study enabled software extendibility and reusability**

  During the development of the software in this study, the initial design had to be changed and implementations had to be updated. The Camera Calibration program of the Image Acquisition Subsystem was extended with a function to produce approximate data with the Direct Linear Transformation algorithm. This was easily done with the addition of new CDLT class to the program class. Minimum changes were made to the existing programs.

  The same extendibility was demonstrated when the wide lane technique was added to the existing GPS data processing program. Few changes were necessary to the existing program.

  Reusability of the software was demonstrated by using large parts of the bundle adjustment program in the GPS data processing program, with only few changes.

  The extendibility and the reusability of the Object Oriented design was confirmed in the automated mapping system software design produced in this research.

- **Different requirements for aerial and terrestrial images should be taken into account for the automated mapping system design**

  In the testing of the bundle block adjustment application, it was found that requirements should be analysed separately for terrestrial images. Although the same theories of photogrammetry are applied for both the aerial and the terrestrial images, different aspects of automation exist for each type.

  For example, many of the automation techniques, such as locating conjugate points and tie point identification in aerial images, assume near vertical photography. This is not so with terrestrial images and the same image matching techniques would not be applicable.

  The end product of mapping with aerial images usually becomes part of a topographic database of a GIS system. With the terrestrial images, many of the end results are three dimensional reconstructions of the imaged objects.

  Control points or the positioning sensors provide a reference coordinate frame for the aerial images. In many cases of terrestrial images, they are used without reference frames and only relative distances and sizes are needed as their output.

It was found during the testing of the bundle block adjustment that these differences between the aerial and terrestrial images require a separate analysis of the close range application and that separate classes should be designed for the terrestrial images.

- **Mathematical software is useful in the implementation of class member functions in the development of the automated mapping system**

  In mapping applications, many linear algebra computations are involved. The use of mathematical software to validate the computed results of a developed program ensures the reliability of the developed software in terms of computational accuracy. The mathematical software is useful in validating the result of computations carried out by member functions.

  Mathematical software is not, however, suitable for applications which demand fast computational time, nor is it efficient for large programs that require considerable maintenance.

7.2 Recommendations for Further Development of the Automated Mapping System

- **Design and implementation of the CDisplayingImage class and the Visualisation Subsystem**

  The design and the implementation of the Visualisation Subsystem will require a class which will be able to handle the reading of image files in various formats, handling of radiometric pixel data and displaying the images. This class, shown in Figure 3.8, as CDisplyImage, will have to be versatile.

  Besides playing an important role in the Visualisation Subsystem, this class will also be responsible for many tasks which apply the image matching techniques of the Image Point Referencing Subsystem.

- **Implementation and testing of IMU and Kalman Filtering related classes**

  The IMU and the Kalman Filtering classes were designed in this research as part of the Positioning Subsystem. In future research, these classes should be implemented and tested to complete the Positioning Subsystem.

- **Extension of design to include various imaging sensors**

  Linear array sensors and laser scanners are some of the potential imaging sensors that are actively being researched at present. Although some initial consideration was given to these sensors in this research, future studies should be carried out to include these sensors as part of the automated mapping system. The automated mapping system will be suitable for the inclusion of these sensors, because these sensors interact with IMU and GPS and these positioning sensors have already been designed in the Positioning Subsystem.

- **Extension of design to include Real-Time Kinematic Mapping Systems**

  Real-Time Kinematic Mapping Systems are useful in many applications, especially in emergency situations such as monitoring forest fires and floods. The design for the extension to a Real-Time Kinematic Mapping System will involve extension of the GPS related classes of the Positioning Subsystem to handle radio signals from the reference station to the rover station.

- **Design and implementation of standard GUIs for mapping applications**

  GUI programming is relatively simple but tedious. It is the kind of activity a Geomatics domain expert would not like to spend too much time on. To have a set of standard GUIs available for mapping applications will be beneficial to the Geomatics community. Initially the GUIs could serve as a simple set of formatted input dialog boxes and output windows to assist the research activities, but these could later develop as standard GUIs. Such standard GUIs will be valuable for the Geomatics community not only in terms of saving development time but also for the end users that will have the advantage of using a familiar GUI, irrespective of the developer of the software.

7.3 Recommendations for Future Research Based on an Automated Mapping System

Recently in Geomatics, although basic research into mathematical modeling and accuracy improvement is regarded as being fundamental and important (as they should be), applied research has been more popular. Applied research in Geomatics has focused on two main themes: geospatial data production and geospatial data manipulation. In both of these themes, automation is inevitably involved.

Some of the research that is being carried out with regard to the automation of geospatial data production includes :

- sensor integration
- data fusion
- automated object recognition
- automation of aerial triangulation

For automated geospatial data manipulation recent research efforts include:

- three dimensional GIS data structures
- visualisation
- automated data quality reporting
- interaction with geospatial information using the Internet

Future Research Using the Automated Mapping System

The Automated Mapping System would be beneficial in the automation research mentioned above. This would be because an integrated approach is possible through the various subsystems that are available and because the time from the initial inception to the delivery of the research result will be shortened through the shorter implementation periods, made possible by the Object Oriented Design of the Automated Mapping System.

Some of the research themes that could be carried out using the Automated Mapping System are:

- Comparison of automatic relative orientation and IMU derived attitude.

  The IMU provides the final attitude data of an image through refinement of its initial data through updates which require GPS observations. It is also dependent on the synchronisation of the measurement epochs of various sensors (Imaging Sensor, GPS and IMU).

  The relative orientation of the images on the other hand is purely photogrammetric and is only effected by the image matching accuracy of conjugate points, if other unmodeled systematic errors are disregarded. The comparison of the image rotational angles computed from automatic relative orientation and those computed through GPS/IMU integration, might give indications as to whether updating of the IMU attitude data with the relative orientation data is justifiable.

- Investigation of geometric errors through tracing of selected points - from initial acquisition to GIS database.

Automated data quality reporting is a main concern with regard to assessing the geospatial data that are incorporated into a GIS. Software could be developed based on the Automated Mapping System design of this research, which would show a visual profile of selected points from the GIS database. Error tracing has been difficult in the past because much of the information from preceding processes of geospatial data production is not passed on to the next. It is even difficult to know which processes were involved. With the integrated Automated Mapping System, this information could be preserved and accumulated. A standard quality profile could be made available visually, using graphs and images.

- A study in data fusion for automatic object recognition using integrated sources.

  With an integrated approach, processing of data from various imaging sensors, positioning sensors and GIS databases will be made available easily. Data of various texture from different imaging sensors, geometric data from positioning sensors and semantic data from GIS databases will prove helpful in the very difficult task of interpreting complex images automatically.

- A study into an integrated approach to the visualisation of three dimensional images.

  Visualisation of aerial images usually involves creating perspective three dimensional views. Buildings and roads and other objects are also created from the aerial images. Visualisation from terrestrial images, on the other hand, involves object reconstruction of specific and smaller objects, such as monuments or sites. Better quality visualisation results could be produced if inhomogeneous data from these two types of image could be merged to a single visualisation image.

  Finally, further study into the visualisation of inhomogeneous data, and fuller utilization of the power of the Automated Mapping System could allow for a continuous transition of the visualisation of the urban data from an above ground perspective to the underground utilities. It is an important requirement in utilities management to be able to relate an underground utility, such as a gas pipe or a water main, to a manhole or a building on the street. This integrated visualisation could prove a most powerful decision support tool.

REFERENCE LIST

[1] Abidin, H. Z. Computational and Geometrical Aspects of On-The-Fly Ambiguity Resolution. 1993. Department of Geodesy and Geomatics Engineering - University of New Brunswick.
Ref Type: Thesis/Dissertation

[2] Ackermann, F. Sensor and Data Integration - The New Challenge. 3-10. 1995. Herbert Wichmann Verlag, Heidelberg.
Ref Type: Conference Proceeding

[3] Ackermann, F., Krzystek, P. "Complete Automation of Digital Aerial Triangulation." *Photogrammetric Record*, 1997, *15* (89), 645-656.

[4] Anderson, B. D. O.,Moore, J. B. *Optimal Filtering*. Englewood-CLiffs, NJ: Prentice-Hall, 1979.

[5] Axelsson, P. "Processing of Laser Scanner Data - Algorithms and Applications." *ISPRS Journal of Photogrammetry & Remote Sensing*, 1999, *54* 138-147.

[6] Beyer, H. A. Geometric and Radiometric Analysis of a CCD-Camera Based Photogrammetric Close-Range System. 18-42. 1992. Institut fur Geodasie und Photogrammetrie.
Ref Type: Thesis/Dissertation

[7] Birchenhall, C. R. A Draft Guide to MatClass : A Matrix Class for C++ Version 1.0d. 1993. Department of Econometrics and Social Statistics - University of Manchester. Web page (accessed in August 1997): http://les.man.ac.uk/ses/staff/crb/matclass/
Ref Type: Computer Program

[8] Boichis, N., Cocquerez, N, and Airault, S. A Top Down Strategy for Simple Cross Roads Automatic Extraction. XXXII(Part 2), 19-26. 1998. Cambridge, United Kingdom. Proceedings of the Commision II Symposium : Data Integration Systems and Techniques.
Ref Type: Conference Proceeding

[9] Booch, G. *Object-Oriented Analysis and Design*. Benjamin/Cummings, 1994.

[10] Britting, K. R. "Introduction," *Inertial Navigation Systems Analysis*. John Wiley & Sons, 1971.

[11] Brown, D. C., Davis, R. G., and Johnson, C. J. Research in Mathematical Targeting - The Practical and Rigorous Adjustment of Large Photogrammetric Nets. RADC-TDR-64-353. 1964. TDR-Report.
Ref Type: Report

[12] Corbett, S. J. GPS Single Epoch Ambiguity Resolution for Airborne Positioning and Orientation . 1994. Department of Surveying - University of Newcastle Upon Tyne.
Ref Type: Thesis/Dissertation

[13] Counselman, C. C., Gourevitch, S. A. "Miniature Interferometer Terminals for Earth Surveying: Ambiguity and Multipath with the Global Positioning System." *IEEE Transactions on Geoscience and Remote Sensing*, 1981, *GE-19* (4), 244-252.

[14] Cox, B. J. *Object-Oriented Programming: An Evolutionary Approach.* Reading, MA: Addison-Wesley, 1986.

[15] Douglass, B. P. "Introduction to Objects and the Unified Modeling Language," *Doing Hard Times - Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns.* Addison Wesley Longman, 1999, 3-55.

[16] Draper, C., Wrigley, W., Hovorka, J., *Inertial Guidanc.* Pergamon Press, 1960.

[17] Forstner, W. and Gulch, E. A Fast Operator for Detection and Precise Location of Distinct Points, Corners and Centers of Circular Features. ISPRS Intercommission Workshop, 281-305. 1987. Interlaken.
Ref Type: Conference Proceeding

[18] GDE-Systems. "Triangulation," *Socet Set User's Manual.* 1997, 14-1-14-67.

[19] Gelb, A. *Applied Optimal Estimation.* Cambridge, Ma.: MIT-Press, 1974.

[20] Goad, C., Yang, M. "A New Approach to Precision Airborne GPS Positioning for Photogrammetry." *Photogrammetric Engineering & Remote Sensing*, 1997, *September* 1067-1077.

[21] Goldberg, A.,Robson, D. *SmallTalk-80: The Language and Its Implementation.* 1983.

[22] Golub, H. G., Van Loan, C. F. "Positive Definite Systems," *Matrix Computations.* The Johns Hopkins University Press, 1996, 140-151.

[23] Gruen, A. and Li, H. Linear Feature Extraction with LSB-Snakes from Multiple Images. XXXI(B3), 266-272. 1996. Vienna. International Archives of Photogrammetry and Remote Sensing.
Ref Type: Conference Proceeding

[24] Gulch, E. "Automatic Control Point Measurement." *Photogrammetric Week '95*, 1995, 185-195.

[25] Gurtner, W., Mader, G. "Receiver Independent Exchange Format Version 2." *CSTG GPS Bulletin*, 1990, *3* (3).

[26] Haala, N., Brenner, C., and Statter, C. An Integrated System for Urban Model Generation. XXXII(Part 2), 96-103. 1998. Cambridge, United Kingdom . Proceedings of the Commision II Symposium : Data Integration Systems and Techniques.
Ref Type: Conference Proceeding

[27] Harman, M., Jones, R., *First Course in C++ : A Gentle Introduction.* McGraw Hill, 1997.

[28] Hatch, R. Ambiguity Resolution in the Fast Lane. 26-29. 1989. Colorado Springs, CO. Proceedings of the Second International Technical Meetings of the Satellite Division of the ION, GPS-89.
Ref Type: Conference Proceeding

[29] Hatch, R. Instantaneous Ambiguity Resolution. Schwarz, K. P. and Lachapelle, G. Symposium No. 107, 299-308. 1990. Banff, Canada, Springer-Verlag. Kinematic Systems in Geodesy, Surveying and Remote Sensing. International Union of Geodesy and Geophysics and International Association of Geodesy.
Ref Type: Conference Proceeding

[30] Heipke, C. Overview of Image Matching Techniques. 173-189. 1996. OEEPE.
Ref Type: Conference Proceeding

[31] Hofmann-Wellenhof, B., Lichtenegger, H., Collins, J. "Surveying with GPS," *GPS - Theory and Practice.* Springer-Verlag, 1994, 129-136.

[32] Horton, I. "Windows Programming and MFC," *Introduction to Microsoft VisualC++ 6.0 Standard Edition.* Wrox Press: 1998, 47-80.

[33] Jekeli, C. "Coordinate Frames and Transformations," *Inertial Navigations Systems with Geodetic Applications - Course Notes.* Dept. of Civil and Environmental Engineering and Geodetic Science, The Ohio State University, 1998, 1-1-1-23.

[34] Kalman, R. E. "A New Approach to Linear Filtering and Prediction Problems." *ASME Journal of Basic Engineering*, 1960, *82* 34-45.

[35] Keller, S. F. "Potentials and Limitations of Artificial Intelligence Techniques Applied to Generalisation," Muller, J. C., Lagrange, J. P., Weibel, R., *GIS and Generalisation - Methodology and Practice*. Taylor & Francis Ltd., 1995, 135-147.

[36] King, D. J. "Airborne Multispectral Digital Camera and Video Sensors: A Critical Review of System Designs and Applications." *Canadian Journal of Remote Sensing*, 1995, *21* (3), 245-273.

[37] Kraus, K. "Coordinate Systems and Transformations," *Photogrammetry - Advanced Methods and Applications*. Ferd. Dummlers Verlag, 1997, 12-43.

[38] Kraus, K., Waldhausl, P. "Photogrammetric Triangulation," *Photogrammetry - Fundamentals and Standard Processes*. Ferd. Dummlers Verlag, 1993.

[39] Kruck, E. "Computational Examples," *BINGO User's Manual*. Intergraph, 1986, 8-1-8-63.

[40] Lachapelle, G. GPS Observables and Error Sources for Kinematic Positioning. Schwarz, K. P. and Lachapelle, G. Symposium No. 107, 17-26. 1990. Banff, Canada, Springer-Verlag. Kinematic Systems in Geodesy, Surveying and Remote Sensing. International Union of Geodesy and Geophysics and International Association of Geodesy.
Ref Type: Conference Proceeding

[41] Lapucha, D. Precise GPS/INS Positioning For A Highway Inventory System. 1990. Department of Surveying Engineering, University of Calgary.
Ref Type: Thesis/Dissertation

[42] Maybeck, P. S. *Stochastic Models, Estimation, and Control*. Vol. 1, New York: Academic Press, 1979.

[43] Maybeck, P. S. *Stochastic Models, Estimation and Control*. Vol. 2, New York: Academic Press, 1982.

[44] McKeown, D. M., Harvey, W., Wixson, L. "Automating Knowledge Acquistion for Aerial Image Interpretation." *Computer Vision, Graphics and Image Processing*, 1989, *46* 37-81.

[45] Merchant, D. C. "Spatial Triangulation - The Bundle Method," *Analytical Photogrammetry : Theory and Practice Part - II.* Department of Geodetic Science and Surveying, The Ohio State University, 1984.

[46] Meyer, B. *Object-Oriented Software Construction.* Prentice Hall International (UK) Ltd., 1988.

[47] Microsoft. "Fundamentals of Object-Oriented Design," *Introduction to C++ - A Short Guide to Programming in C++.* Microsoft Corporation, 1997, 169-189.

[48] Mikhail, E. M., Ackermann, F. *Observations and Least Squares.* Harper & Row, 1976.

[49] Mills, J. P., Newton, I., Graham, R. W. "Aerial Photography for Survey Purposes with a High Resolution, Small Format, Digital Camera." *Photogrammetric Record*, 1996, *15* 575-587.

[50] Molenaar, M. *An Introduction to the Theory of Spatial Object Modelling for GIS.* Taylor & Francis Ltd., 1998.

[51] Moniwa, H. Analytical Photogrammetric System with Self-Calibration and Its Applications. 34-59. 1972. The University of New Brunswick.
Ref Type: Thesis/Dissertation

[52] Moniwa, H. "The Concept of Photo-Variant Self-Calibration and Its Application in Block Adjustment with Bundles." *Photogrammetria*, 1981, *36* 11-29.

[53] Ohlhof, T., Kornus, W. "Geometric Calibration of Digital Three-Line CCD Cameras." *ISPRS Commission I Symposium*, 1994, *30* (1), 71-81.

[54] OMG. OMG Unified Modeling Language Specification. 23-23. 1999.
Ref Type: Report

[55] Peipe, J. "Photogrammetric Calibration and Performance Test of Still Video Cameras." *ISPRS Commission I Symposium*, 1994, *30* (1), 108-113.

[56] Peipe, J., Schneider, T. "High Resolution Still Video Camera for Industrial Photogrammetry." *Photogrammetric Record*, 1995, *15* 135-139.

[57] Peng, W. Automated Generalisation in GIS. 1996. ITC Publication Series Nr. 40.
Ref Type: Thesis/Dissertation

[58] Pilouk, M. Integrated Modelling for 3D GIS. 1997. ITC Publication Series Nr. 40.
Ref Type: Thesis/Dissertation

[59] Pooley, R. J., Stevens, P. *Using UML : Software Engineering with Objects and Components.* Addison-Wesley, 1999.

[60] Remondi, B. W. Using the Global Positioning System (GPS) Phase Observable for Relative Geodesy: Modeling, Processing and Results. 1984. Center for Space Research, University of Texas at Austin.
Ref Type: Thesis/Dissertation

[61] Roux, M. and McKeown, D. M. Feature Matching for Building Extraction from Multiple Views. 331-349. 1994. Monterey, California, ARPA. Proceeedings of the ARPA IUW.
Ref Type: Conference Proceeding

[62] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. , Lorensen, W. *Object-Oriented Modeling and Design.* Englewood Cliffs: Prentice Hall, 1991.

[63] Schenk, T., Li, J. C., Toth, C. K. "Towards an Autonomous System for Orienting Digital Stereopairs." *Photogrammetric Engineering & Remote Sensing*, 1991, *57* (8), 1057-1064.

[64] Schenk, T., Toth, C. K. "Computer Vision and Digital Photogrammetry." *ITC Journal*, 1992, *1* 34-38.

[65] Schroeder, W., Martin, K., Lorensen, B. "Introduction," *The Visualization ToolKit : An Object-Oriented Approach to 3D Graphics.* Prentice Hall, 1998, 1-15.

[66] Schwarz, K. P. Sensor Integration and Image Georeferencing. Invited Lecture Duane C. Brown International Summer School in Geomatics. 1998. The Ohio State University.
Ref Type: Audiovisual Material

[67] Schwarz, K. P., Cannon, E., Wong, R. V. C. "A Comparison of GPS Kinematic Models for the Determination of Position and Velocity Along a Trajectory." *Manuscripta Geodetica*, 1989, *14* 345-353.

[68] Schwarz, K. P.,Wei, M. *ENSU 623 Lecture Notes.* Department of Geomatics Engineering, The University of Calgary, 1994.

[69] Sheffer, D. B., Herron, R. E. "Biosterometrics," Karara, H. M., *Non-Topographic Photogrammetry.* American Society for Photogrammetry and Remote Sensing, 1989, 359-366.

[70] Skaloud, J. Strapdown INS Orientation Accuracy with GPS Aiding. 1995. Department of Geomatics Engineering - The University of Calgary.
Ref Type: Thesis/Dissertation

[71] Combs, J. E. (Ed.), "Chapter VII : Planning and Executing the Photogrammetric Project", *Manual of Photogrammetry*, American Society for Photogrammetry and Remote Sensing, 1980, 367- 412.

[72] Strang, G., Borre, K. "Decorrelation and Weight Normalization," *Linear Algebra, Geodesy and GPS*. Wellesey-Cambridge Press, 1997, 400-403.

[73] Streilein, A. Videogrammetry and CAAD for Architectural Restitution of the Otto-Wagner-Pavillon in Vienna. "Optical 3-D Measurement Techniques III - Applications in Inspection, Quality Control and Robotics". Editors Gruen, A. and Kahmen, H. 305-314. 1995. Vienna, Wichmann.
Ref Type: Conference Proceeding

[74] Streilein, A., Grussenmeyer, P., and Hanke, K. Zurich City Hall - A Reference Data Set for Digital Close-Range Photogrammetry. 1999. Olinda, Brazil. XVII CIPA International Symposium.
Web page (accessed on December 1999) : http://cipa.uibk.ac.at

[75] Stroustrup, B. "A Tour of C++," *The C++ Programming Language*. Addison-Wesley, 1995, 13-42.

[76] TerraSat. Sample Data File. 1999.
Web page (accessed on December 1999) : http://www.terrasat.de/news.htm

[77] Waldhausl, P. Defining the Future of Architectural Photogrammetry. 29(B5), 767-770. 1992. International Archives of Photogrammetry and Remote Sensing.
Ref Type: Conference Proceeding

[78] Wei, M., K.P.Schwarz. "A Strapdown Inertial Algorithm Using an Earth-Fixed Cartesian Frame." *Journal of The Institute of Navigation*, 1990, *37* (2), 153-167.

[79] Wolf, P., Ghilani, C. D. *Adjustment Computations - Statistics and Least Squares in Surveying and GIS*. John Wiley & Sons, 1997.

[80] Wong, R. V. C. Development of a RLG Strapdown Inertial Surveys System. 41-47. 1988. Department of Surveying Engineering, University of Calgary.
Ref Type: Thesis/Dissertation

APPENDIX A : DATA FILES FOR CAMERA CALIBRATION

The Calibrate Image Sensor Use Case of the Image Acquisition Subsystem was implemented using the software design of Chapter 3.

There are three important input data files for this program: the image file; the ground control point file; and the image sensor file. All these input files should reside in a subdirectory called 'infile'.

The Image File, which should be named 'gubaio.img', contains all the data pertaining to the image. Each image begins with the '[IMAGE]' keyword and ends with the 'ImageEnd' keyword. The name of the image; the name of the sensor used to capture this image; the initial approximate coordinates of the perspective center; and the initial approximate rotations of the image are then entered on separate lines. Each keyword; the '=' symbol; and the values should be separated by a space between them as shown in the example below.

```
[IMAGE]
name = st1
sensor = dcs
InitialPersCenter = 0 0 .8
InitialRotation = 0 0 0
Points
 1   532.043478   185.565217
 2   649.434783   182.086957
.....
.....
PointsEnd
```

The 'Points' keyword marks the beginning of the image coordinates (or pixel coordinates) and the 'PointEnd' keyword marks the end of the points. Each image point is entered in a new line in the order of point number, x coordinate value and y coordinate value.

For the ground control point file, which should be named 'gubaio.gcp', 'FullControl' and 'FullControlEnd' keywords mark the beginning and the end of control point coordinates. Each line is then entered with the point number, X, Y and Z coordinates as shown in the example below.

```
FullControl
1        -.102767  .093382  0.0
2        -.062.821 .09373   0.0
.....
.....
FullControlEnd
```

For the Image Sensor File, which should be named 'gubaio.cam', each sensor begins with the '[SENSOR]' keyword and ends with the 'END' keyword. It is followed by the name of the sensor and the type of the imaging sensor. The keyword for the type of the imaging sensor can be 'DIGITAL', 'SCAN' or 'FRAME'. If 'DIGITAL' is selected, the sensor element should be given as the dimensions of height and width of the sensor chip in millimetres. If the type is 'SCAN', four fiducial coordinates (x, y) of the camera should be given, beginning with upper left then proceeding in anti-clockwise direction to the lower left, lower right and then upper right. An example is shown below.

```
[SENSOR]
name = RC30
type = SCAN
1 -105.996 105.999
2 -105.993 -105.996
3 105.996 -105.999
4 105.997 105.999
ScanResolution = 25
CameraConstants = 0.014 0.022 303.1
RadialDistortion = 0.0 0.0 0.0
DecenteringDistortion = 0.0 0.0 0.0
Affinity = 0.0 0.0
END
```

For 'SCAN' and 'FRAME' type, the **image file** should have the pixel coordinates or the photo coordinates of the fiducial points included immediately after the line specifying the sensor, as shown in the example below.

```
[IMAGE]
name = 131
sensor = RC30
1 359 382
2 363 8866
3 8845 8868
4 8845 385
InitialPersCenter = 166170.9614 196547.0127 1005.285568
InitialRotation = 0.1813257385 0.869770513 0.946245091
.....
```

These coordinates are used to perform the interior orientation of each image.

For the 'DIGITAL' type, this information is not necessary. The dimensions of the sensor chip are used instead, and the coordinates of the four corners of the sensor chip are used as the fiducial points. The following data are the actual data used for the camera calibration testing explained in chapter six.

1. Input Files

1.1 Image File (gubaio.img)

```
[IMAGE]
name = st1
sensor = dcs
```

```
InitialPersCenter = 0 0 .8
InitialRotation = 0 0 0
Points
 1   532.043478   185.565217
 2   649.434783   182.086957
 3   824.130435   178.173913
 4   939.000000   176.869565
 5   1082.913043   172.956522
 6   536.434783   301.913043
 7   652.086957   298.869565
 8   827.391304   295.826087
 9   942.608696   292.782609
 10   1086.173913   290.173913
 11   541.869565   476.434783
 12   658.826087   472.521739
 13   831.086957   469.913043
 14   947.173913   466.000000
 15   1089.434783   462.521739
 16   544.565217   593.478261
 17   660.652174   589.130435
 18   833.347826   584.347826
 19   947.695652   581.739130
 20   1091.260870   576.086957
 21   549.608696   736.782609
 22   664.826087   732.869565
 23   838.391304   728.086957
 24   952.304348   724.173913
 25   1094.130435   719.826087
PointEnd
ImageEnd
[IMAGE]
name = st2
sensor = dcs
InitialPersCenter = .5 .3 .7
InitialRotation = -10 45 -90
Points
 1   593.333333   724.666667
 2   592.000000   628.666667
 3   587.333333   466.000000
 4   586.666667   349.333333
 5   584.000000   190.333333
 6   715.333333   718.666667
 7   717.333333   620.666667
 8   722.000000   459.333333
 9   724.666667   342.666667
 10   728.666667   185.333333
 11   895.333333   707.333333
 12   904.666667   608.666667
 13   917.333333   450.666667
 14   926.666667   336.000000
 15   940.666667   181.333333
 16   1013.333333   700.000000
 17   1024.666667   601.333333
 18   1044.666667   446.000000
 19   1058.000000   330.333333
 20   1077.333333   176.333333
 21   1158.000000   690.333333
 22   1174.000000   594.333333
 23   1200.666667   437.666667
 24   1219.333333   325.000000
 25   1245.333333   170.333333
PointEnd
ImageEnd
[IMAGE]
name = st3
```

```
sensor = dcs
InitialPersCenter = .5 -.5 .8
InitialRotation = 10 45 90
Points
 1   1002.000000  289.000000
 2   1020.000000  345.000000
 3   1051.000000  438.000000
 4   1074.000000  502.000000
 5   1102.000000  589.000000
 6   929.000000  293.000000
 7   948.000000  351.000000
 8   979.000000  444.000000
 9   998.000000  511.000000
 10   1024.000000  595.000000
 11   819.000000  298.000000
 12   836.000000  360.000000
 13   861.000000  452.000000
 14   879.000000  520.000000
 15   903.000000  605.000000
 16   744.000000  303.000000
 17   760.000000  366.000000
 18   783.000000  459.000000
 19   799.000000  527.000000
 20   819.000000  614.000000
 21   649.000000  307.000000
 22   662.000000  370.000000
 23   681.000000  465.000000
 24   695.000000  534.000000
 25   714.000000  623.000000
PointEnd
ImageEnd
[IMAGE]
name = st4
sensor = dcs
InitialPersCenter = -.5 0 .5
InitialRotation = 0 -45 0
Points
 1   538.000000  186.333333
 2   630.500000  186.333333
 3   757.166667  184.666667
 4   830.500000  183.833333
 5   918.000000  185.500000
 6   541.333333  317.333333
 7   632.166667  309.833333
 8   758.000000  302.333333
 9   831.333333  298.166667
 10   917.166667  292.333333
 11   545.500000  507.833333
 12   635.500000  494.500000
 13   758.833333  474.500000
 14   832.166667  462.833333
 15   918.000000  451.166667
 16   548.833333  633.500000
 17   638.833333  614.333333
 18   760.500000  587.666667
 19   834.666667  571.833333
 20   921.333333  554.333333
 21   553.000000  787.500000
 22   643.000000  760.833333
 23   763.000000  727.500000
 24   836.333333  706.666667
 25   918.000000  681.666667
PointEnd
ImageEnd
[IMAGE]
```

```
name = st5
sensor = dcs
InitialPersCenter = -.7 .5 .7
InitialRotation = 10 -45 90
Points
 1   975.666667  299.000000
 2   966.500000  361.500000
 3   951.500000  451.500000
 4   942.333333  506.500000
 5   929.833333  571.500000
 6   899.000000  304.833333
 7   891.500000  369.000000
 8   878.166667  457.333333
 9   869.000000  512.333333
 10   860.666667  580.666667
 11   779.000000  314.833333
 12   774.000000  378.166667
 13   767.333333  467.333333
 14   762.333333  520.666667
 15   755.666667  589.833333
 16   699.833333  319.000000
 17   695.666667  382.333333
 18   690.666667  474.833333
 19   686.500000  530.666667
 20   682.333333  599.833333
 21   597.333333  327.333333
 22   595.666667  393.166667
 23   594.000000  484.833333
 24   591.500000  541.500000
 25   590.666667  609.833333
PointEnd
ImageEnd
```

1.2 Ground Control Point File (gubaio.gcp)

```
FullControl
1          -.102767 .093382  0.0
2          -.062.821 .09373  0.0
3          -.002838 .094302 0.0
4          .037149  .094683 0.0
5          .08713   .095141 0.0
6          -.10244  .053375 0.0
7          -.062441 .05375 0.0
8          -.002464 .054305 0.0
9          .037514  .054683 0.0
10         .087495  .055124 0.0
11         -.101895 -.006575 0.0
12         -.061914 -.006203 0.0
13         -.0019   -.005679 0.0
14         .038053  -.005289 0.0
15         .088037  -.004839 0.0
16         -.101.54 -.046598 0.0
17         -.061554 -.046264 0.0
18         -.001574 -.045656 0.0
19         .03843   -.045298 0.0
20         .088405  -.044789 0.0
21         -.101111 -.096581 0.0
22         -.061126 -.096203 0.0
23         -.001126 -.095672 0.0
24         .038831  -.095292 0.0
25         .088832  -.094836 0.0
FullControlEnd
CheckPoint
CheckPointEnd
TiePoint
TiePointEnd
```

1.3 Image Sensor File (gubaio.cam)

```
[SENSOR]
name = dcs
type = DIGITAL
SensorElement = 5.0052 7.5078
SensorPixels = 1024 1536
CameraConstants = 0 0 8
RadialDistortion = 0.0 0.0 0.0
DecenteringDistortion = 0.0 0.0 0.0
Affinity = 0.0 0.0
END
```

2. Output File (gubaio.out)

```
Adjustment Start Time : 1999 / 12 / 4 (14:41:32)

----------adjustment evironment----------

[GENERAL]

Adjusted WITH Self-Calibration
Platform type = AERIAL
AngleUnit = DEGREE
Converge Test criteria = 1e-005
Maximum iteration = 15
A priori Standard Deviation of Image Observation = 0.005 mm

[CAMERA PARAMETERS]

Principal Point = FREE (Sigma = 1e+009 mm)
Principal Distance = FREE (Sigma = 1e+009 mm)

[IMAGE PARAMETERS]

Rotation angles = FREE (Sigma = 1e+009 DEGREE)
Perspective Point Coordinate = FREE (Sigma = 1e+009 m)

[CONTROL PARAMETERS]

Planimetry Controls = FIXED
Height Controls = FIXED

[REDUNDANCY STATUS]

Number of Parameters = 115
Number of Equations = 365
Redundancy = 250

[CONTROL POINTS STATUS]

Number of Full Controls =25
Number of Check Points =0
Number of Tie Points =0


----------Iterations----------
Iteration No = 1
Sigma0 = 2.1323
```

Iteration No = 2
Sigma0 = 0.96045

Iteration No = 3
Sigma0 = 0.22957

Iteration No = 4
Sigma0 = 0.042114

Iteration No = 5
Sigma0 = 0.0056607

Iteration No = 6
Sigma0 = 0.0042132

Iteration No = 7
Sigma0 = 0.0042131

Time of execution(adjustment iterations only) : 2.14 seconds

----------Residual Report----------
Maximum residual : 0.016604
Image Name : st5
Point Name : 14

A POSTERIORI Sigma0 = 0.0042131

---------- Correlation matrix between sensor parameters for (dcs)----------

	xp	yp	cc	K1	K2	K3	P1	P2	A	B
xp	1.00									
yp	0.01	1.00								
cc	0.31	0.30	1.00							
K1	0.20	0.11	0.39	1.00						
K2	0.00	0.00	0.00	0.00	1.00					
K3	0.00	0.00	0.00	0.00	0.00	1.00				
P1	0.83	0.10	0.02	0.16	0.00	0.00	1.00			
P2	0.10	0.89	0.08	0.09	0.00	0.00	0.11	1.00		
A	0.16	0.30	0.25	0.10	0.00	0.00	0.08	0.14	1.00	
B	0.27	0.09	0.07	0.09	0.00	0.00	0.13	0.01	0.03	1.00

---------- Lens Distortion Coefficient(Initial Values) fordcs)----------

K1	K2	K3	P1	P2	A	B
-0.0000907070	0.00000	0.00000	-0.0000937410	-0.0000317171	-0.0025433164	0.0007809931
0.007	0.006	0.006	0.001	0.001	0.005	0.007

---------Adjusted Exterior Orientation Xo Yo Zo (m) omega phi kappa in DEGREE(Standard Deviations)-------
Image : st1
-0.050 0.025 0.725 -2.545 -2.229 -0.971
0.00274 0.00735 0.00828 0.16039 0.21214 0.17389

Image : st2
0.335 0.131 0.529 -9.973 32.599 -86.894
0.00370 0.00133 0.00565 0.07866 0.22527 0.38619

Image : st3
0.582 -0.274 0.812 15.353 34.338 77.270

```
   0.00112    0.00189    0.00130    0.03932    0.23441    0.20142

Image : st4
    -0.540     0.085     0.489    -10.551    -47.674    -8.055
   0.00393    0.00706    0.00584    0.03074    0.11356    0.24559

Image : st5
    -0.718    -0.238     0.783     16.866    -41.355     96.324
   0.01431    0.00192    0.00962    0.20263    0.19762    0.27777

---------Adjusted Camera Constants; xp yp cc in mm(Standard Deviations)-------
Sensor Name : dcs
     0.031     0.188    10.275
   0.00409    0.00962    0.00156

---------Control Points; x y z in m(Standard Deviations)-------
  Name            x             y             z      Control Type
-------------------------------------------------------------------------------

    1         -0.103         0.093         0.000  Control
              0.003         0.003         0.001

    2         -0.062         0.094         0.000  Control
              0.006         0.001         0.006

    3         -0.003         0.094         0.000  Control
              0.005         0.007         0.002

    4          0.037         0.095         0.000  Control
              0.004         0.001         0.015

    5          0.087         0.095        -0.000  Control
              0.004         0.006         0.005

    6         -0.102         0.053         0.000  Control
              0.005         0.005         0.011

    7         -0.062         0.054         0.000  Control
              0.007         0.005         0.006

    8         -0.002         0.054         0.000  Control
              0.008         0.006         0.004

    9          0.038         0.055        -0.000  Control
              0.018         0.014         0.001

   10          0.087         0.055        -0.000  Control
              0.020         0.011         0.001

   11         -0.102        -0.007         0.000  Control
              0.003         0.005         0.001

   12         -0.062        -0.006        -0.000  Control
              0.005         0.007         0.006

   13         -0.002        -0.006        -0.000  Control
              0.006         0.002         0.008

   14          0.038        -0.005        -0.000  Control
              0.004         0.001         0.005

   15          0.088        -0.005         0.000  Control
              0.005         0.001         0.006

   16         -0.101        -0.047        -0.000  Control
```

	0.004	0.001	0.001	
17	-0.062	-0.046	-0.000	Control
	0.005	0.006	0.004	
18	-0.002	-0.046	-0.000	Control
	0.006	0.006	0.007	
19	0.038	-0.045	-0.000	Control
	0.006	0.005	0.004	
20	0.088	-0.045	0.000	Control
	0.001	0.007	0.005	
21	-0.101	-0.097	0.000	Control
	0.002	0.022	0.001	
22	-0.061	-0.096	0.000	Control
	0.001	0.001	0.008	
23	-0.001	-0.096	0.000	Control
	0.003	0.005	0.006	
24	0.039	-0.095	0.000	Control
	0.006	0.015	0.002	
25	0.089	-0.095	0.000	Control
	0.006	0.004	0.003	

APPENDIX B : DATA FILES FOR BLOCK BUNDLE ADJUSTMENT

The same applies for the data files of block bundle adjustment as explained in Appendix A. A difference from Appendix A in the data file for this application, is the inclusion of tie points and check points. To include tie points and check points, only the ground control point file ('gubaio.gcp') needs to be appended with the keywords 'TiePoint', 'TiePointEnd' and 'CheckPoint', 'CheckPointEnd'. Each part is then filled with point number and approximate X, Y and Z coordinates. For check points, there are two lines for each point. The first line is filled with the point number and the surveyed or known X,Y and Z coordinates. This line is then followed by a second line of the approximate X,Y and Z coordinates. This is shown in the example below.

```
TiePoint
tpe22    146      256      10
tpe23    146      256      3
tps11    152      236      14
tps12    152      236      3
tps21    152      247      14
tps22    152      247      3
TiePointEnd
CheckPoint
102      118.13   235.49   10.37
         110      230      10
112      136.12   237.39   10.37
         130      230      10
122      152.02   239.39   10.48
         150      230      10
CheckPointEnd
```

The next data files show the actual data files used for the bundle adjustment of the terrestrial images explained in Chapter 6.

1. Input Files

1.1 Image File (gubaio.img)

```
[IMAGE]
name = st1
sensor = olympus
InitialPersCenter = 136.0    179.0    6
InitialRotation =  0.0       0.0      0.0
Points
101  215.870968  418.935484
102  212.967742  520.870968
103  209.741935  680.000000
111  643.774194  399.225806
112  646.354839  506.709677
113  649.258065  674.838710
121  1047.096774  382.806452
```

```
 122   1056.129032  491.838710
 123   1065.806452  661.612903
 tp11   137.096774  370.838710
 tp12   166.129032  486.419355
 tp13   163.225806  593.838710
 tp41   1128.935484  324.387097
 tp42   1105.387097  450.516129
 tp43   1115.064516  567.354839
PointEnd
ImageEnd
[IMAGE]
name = st2
sensor = olympus
InitialPersCenter = 136.0      179.0      2.0
InitialRotation =  0.0         0.0        0.0
Points
  101   185.517241  360.034483
  102   182.758621  471.068966
  103   176.551724  637.275862
  111   638.068966  358.655172
  112   640.482759  469.689655
  113   640.137931  638.655172
  121   1038.827586  361.068966
  122   1045.379310  469.344828
  123   1050.551724  634.862069
  tp11   104.551724  305.896552
  tp12   130.413793  431.068966
  tp13   125.241379  543.827586
  tp41   1125.103448  304.793103
  tp42   1094.758621  428.241379
  tp43   1102.689655  543.068966
PointEnd
ImageEnd
[IMAGE]
name = st3
sensor = olympus
InitialPersCenter = 79.0 200  2.0
InitialRotation =  -45         10.0       0.0
Points
 101   616.000000  292.000000
 102   619.000000  470.000000
 103   624.000000  754.000000
 111   949.000000  430.000000
 112   958.000000  558.000000
 113   965.000000  754.000000
 121   1104.000000  498.000000
 122   1113.000000  599.000000
 123   1124.000000  752.000000
 201   215.000000  384.000000
 202   201.000000  759.000000
 211   505.000000  290.000000
 212   502.000000  756.000000
 tp41   1156.000000  461.000000
 tp42   1135.000000  568.000000
 tp43   1144.000000  669.000000
 tp11   579.000000  151.000000
 tp12   572.000000  380.000000
 tp13   571.000000  582.000000
 tp21   110.000000  333.000000
 tp22   149.000000  486.000000
 tp23   142.000000  637.000000
 tpn11   250.000000  372.000000
 tpn13   236.000000  761.000000
PointEnd
ImageEnd
```

```
[IMAGE]
name = st4
sensor = olympus
InitialPersCenter = 96 237    2.0
InitialRotation =  -90 -0      90.0
Points
 201   364.000000  1008.000000
 211   368.000000  306.000000
 212   1052.000000 260.000000
 tp11   220.000000  89.000000
 tp12   518.000000  135.000000
 tp13   803.000000  105.000000
 tpn11   366.000000  911.000000
 tpn13   1066.000000  962.000000
PointEnd
ImageEnd
[IMAGE]
name = st5
sensor = olympus
InitialPersCenter = 96 246    2.0
InitialRotation =  -90 10      90.0
Points
  201   309.000000  766.000000
  202   987.000000  808.000000
  211   322.000000  62.000000
  212   1000.000000  19.000000
  tp22   456.000000  938.000000
  tp23   740.000000  967.000000
  tpn11   308.000000  654.000000
  tpn13   997.000000  694.000000
PointEnd
ImageEnd
[IMAGE]
name = st6
sensor = olympus
InitialPersCenter = 77 284    2.0
InitialRotation =  -135 10.0 0.0
Points
201 726 274
202 732 784
211 1082 356
212 1104 784
 301   137.368421  516.315789
 302   132.105263  627.315789
 303   122.631579  794.157895
 311   316.631579  425.157895
 312   307.684211  635.684211
 321   605.578947  281.421053
 322   603.473684  475.105263
 323   601.894737  787.315789
 tp11 1210 290
 tp12 1158 470
 tp13 1174 642
 tp21 622 120
 tp22 642 376
 tp23 642 596
 tp32   110.000000  592.157895
 tp33   104.210526  706.789474
 tpe11   436 358
 tpe12   434 522
 tpe13   426 784
 tpe21   208 478
 tpe22   198 600
 tpe23   188 790
 tpn11 790 288
```

```
 tpn13 800 784
 PointEnd
ImageEnd
[IMAGE]
name = st7
sensor = olympus
InitialPersCenter = 125 257 0
InitialRotation =  -180 10 90.0
Points
 321   393.000000  269.000000
 322   673.000000  247.000000
 323   1178.000000  213.000000
 tp21   213.000000  45.000000
 tp22   553.000000  96.000000
 tp23   881.000000  64.000000
 tpe11   404.000000  818.000000
 tpe12   680 836
 tpe13   1184.000000  866.000000
PointEnd
ImageEnd
[IMAGE]
name = st8
sensor = olympus
InitialPersCenter = 134 257 0
InitialRotation =  -200 10.0 0.0
Points
 321   1011.000000  242.000000
 322   1027.000000  477.000000
 323   1053.000000  859.000000
 311   80.000000  80.000000
 312   44.000000  496.000000
 tpe11   590.000000  166.000000
 tpe12   596 422
 tpe13   600.000000  858.000000
 tp21   1187.000000  142.000000
 tp22   1127.000000  399.000000
 tp23   1148.000000  649.000000
PointEnd
ImageEnd
[IMAGE]
name = st9
sensor = olympus
InitialPersCenter = 150 257   2.0
InitialRotation =  135 45     80.0
Points
 311   430.000000  310.000000
 312   784.000000  306.000000
 tpe11 554 94
 tpe12 750 82
 tpe13 1078 66
 tpe21 198 738
 tpe22 500 758
 tpe23 1036 794
PointEnd
ImageEnd
[IMAGE]
name = st11
sensor = olympus
InitialPersCenter = 175 257 0
InitialRotation =  100         10.0 0.0
Points
 301   834.000000  281.000000
 302   840.000000  496.000000
 303   842.000000  842.000000
 321   1147.000000  577.000000
```

```
 322   1154.000000  695.000000
 323   1163.000000  870.000000
 311   1046.000000  484.000000
 312   1056.000000  696.000000
 tp21 1198 538
 tp22 1170 656
 tp23 1174 772
 tp31   849.000000  97.000000
 tp32   818.000000  380.000000
 tp33   815.000000  626.000000
 tp41   88.000000  267.000000
 tp42   136.000000  473.000000
 tp43   116.000000  668.000000
 tpe11 1100 532
 tpe12 1108 662
 tpe13 1116 858
 tpe21 948 390
 tpe22 956 568
 tpe23 964 848
 tps11   238.000000  344.000000
 tps12   200.000000  832.000000
 tps21   705.000000  260.000000
 tps22   700.000000  834.000000
PointEnd
ImageEnd
[IMAGE]
name = st12
sensor = olympus
InitialPersCenter = 180 241 0
InitialRotation =  90.0 0.0 90.0
Points
 tp31   320.000000  135.000000
 tp32   515.000000  176.000000
 tp33   688.000000  171.000000
 tps11   414.000000  706.000000
 tps12   824.000000  726.000000
 tps21   418.000000  272.000000
 tps22   830.000000  266.000000
 tp41   328.000000  847.000000
 tp42   515.000000  810.000000
 tp43   683.000000  820.000000
PointEnd
ImageEnd
[IMAGE]
name = st13
sensor = olympus
InitialPersCenter = 185 250 0
InitialRotation =  90 0 -90
Points
 tp31   921.000000  631.000000
 tp32   769.000000  598.000000
 tp33   634.000000  592.000000
 tps11   840.000000  186.000000
 tps12   524.000000  180.000000
 tps21   848.000000  520.000000
 tps22   524.000000  520.000000
 tp41   908.000000  79.000000
 tp42   764.000000  112.000000
 tp43   632.000000  107.000000
PointEnd
ImageEnd
[IMAGE]
name = st14
sensor = olympus
InitialPersCenter = 185 241 0
```

```
InitialRotation =  90.0 0.0 -90.0
Points
tp31   951.000000  779.000000
 tp32   799.000000  747.000000
 tp33   662.000000  750.000000
 tps11   869.000000  327.000000
 tps12   547.000000  326.000000
 tps21   877.000000  670.000000
 tps22   552.000000  679.000000
 tp41   935.000000  218.000000
 tp42   791.000000  252.000000
 tp43   655.000000  249.000000
PointEnd
ImageEnd
[IMAGE]
name = st15
sensor = olympus
InitialPersCenter = 230 160 0
InitialRotation =  45.0 0.0    0.0
Points
 tp31   821.307692  382.153846
 tp32   800.538462  446.384615
 tp33   799.769231  505.230769
 tps11   669.000000  390.230769
 tps12   670.538462  551.769231
 tps21   777.076923  404.846154
 tps22   779.769231  553.692308
 tp41   649.769231  354.153846
 tp42   649.000000  428.384615
 tp43   648.230769  496.461538
 121   632.923077  391.461538
 122   632.538462  454.153846
 123   631.000000  551.846154
 111   482.538462  411.846154
 112   480.615385  468.769231
 113   481.384615  556.076923
 101   351.384615  431.461538
 102   351.384615  481.076923
 103   349.846154  559.153846
 tp11   318.307692  408.769231
 tp12   331.000000  464.153846
 tp13   330.615385  516.076923
PointEnd
ImageEnd
[IMAGE]
name = st16
sensor = olympus
InitialPersCenter = 215 160 0
InitialRotation =  45.0        0.0        0.0
Points
 tp31   925.000000  377.000000
 tp32   901.000000  456.000000
 tp33   902.000000  527.000000
 tps11   784.000000  375.000000
 tps12   787.000000  582.000000
 tps21   880.000000  403.000000
 tps22   882.000000  585.000000
 tp41   778.000000  325.000000
 tp42   768.000000  420.000000
 tp43   768.000000  507.000000
 121   739.000000  373.000000
 122   739.000000  456.000000
 123   738.000000  581.000000
 111   504.000000  401.000000
 112   504.000000  474.000000
```

```
 113   504.000000  585.000000
 101   306.000000  425.000000
 102   305.000000  490.000000
 103   304.000000  588.000000
 tp11   260.000000  398.000000
 tp12   278.000000  468.000000
 tp13   279.000000  535.000000
PointEnd
ImageEnd
```

1.2 Ground Control Point File (gubaio.gcp)

```
FullControl
101      118.13   235.51   14.61
103      118.20   235.44   3.66
111      136.09   237.41   14.65
113      136.08   237.45   3.67
121      151.99   239.43   14.76
123      151.99   239.37   3.69
201      114.77   247.55   14.26
202      114.72   247.28   3.61
211      115.99   236.07   14.25
212      116.03   236.25   3.64
301      151.22   256.00   14.79
302      151.22   256.00   10.49
303      150.75   256.11   3.67
311      136.66   254.48   14.68
312      136.78   254.02   8.20
321      117.01   252.11   14.88
322      117.02   252.09   10.55
323      117.10   252.06   3.63
FullControlEnd
TiePoint
tp11     114      235      16
tp12     114      235      11
tp13     114      235      6
tp21     114      250      16
tp22     114      250      11
tp23     114      250      6
tp31     152      250      16
tp32     152      250      11
tp33     152      250      6
tp41     152      235      16
tp42     152      235      11
tp43     152      235      6
tpn11    114      248      14
tpn13    114      248      4
tpe11    128      256      14
tpe12  128 256 10
tpe13    128      256      3
tpe21    146      256      14
tpe22    146      256      10
tpe23    146      256      3
tps11    152      236      14
tps12    152      236      3
tps21    152      247      14
tps22    152      247      3
TiePointEnd
CheckPoint
102      118.13   235.49   10.37
         110      230      10
112      136.12   237.39   10.37
         130      230      10
122      152.02   239.39   10.48
         150      230      10
CheckPointEnd
```

1.3 Image Sensor File (gubaio.cam)

```
[SENSOR]
name = olympus
type = DIGITAL
SensorElement = 6.6 8.245
SensorPixels = 1024 1280
CameraConstants = -0.1905 0.9305 8.595
RadialDistortion = 0.0 0.0 0.0
DecenteringDistortion = 0.0 0.0 0.0
Affinity = 0.0 0.0
END
```

2. Output File (gubaio.out)

```
Adjustment Start Time : 1999 / 12 / 9 (10:59:47)

----------adjustment evironment----------

[GENERAL]

Adjusted WITHOUT Self-Calibration
Platform type = TERRESTRIAL
AngleUnit = DEGREE
Converge Test criteria = 1e-005
Maximum iteration = 15
A priori Standard Deviation of Image Observation = 0.005 mm

[CAMERA PARAMETERS]

Principal Point = FIXED
Principal Distance = FIXED

[IMAGE PARAMETERS]

Rotation angles = FREE (Sigma = 1e+009 DEGREE)
Perspective Point Coordinate = FREE (Sigma = 1e+009 m)

[CONTROL PARAMETERS]

Planimetry Controls = FIXED
Height Controls = FIXED

[REDUNDANCY STATUS]

Number of Parameters = 225
Number of Equations = 679
Redundancy = 454

[CONTROL POINTS STATUS]

Number of Full Controls =18
Number of Check Points =3
Number of Tie Points =27


----------Iterations----------
Iteration No = 1
Sigma0 = 4.2258

Iteration No = 2
Sigma0 = 1.9123

Iteration No = 3
```

```
Iteration No = 3
Sigma0 = 0.77017

Iteration No = 4
Sigma0 = 0.27411

Iteration No = 5
Sigma0 = 0.13868

Iteration No = 6
Sigma0 = 0.10423

Iteration No = 7
Sigma0 = 0.077145

Iteration No = 8
Sigma0 = 0.072281

Iteration No = 9
Sigma0 = 0.072267

Iteration No = 10
Sigma0 = 0.072265

----------
Time of execution(adjustment iterations only) : 11.92  seconds

----------Residual Report----------
Maximum residual : 0.50994
Image Name      : st6
Point Name      : 202

A POSTERIORI Sigma0 = 0.072265

----------Check Point RMS Report----------
RMSE of XY Coordinates  : 0.45388
Maximum XY Error        : 0.72125  at point 102

RMSE of Z Coordinates   : 0.27363
Maximum Z Error         : -0.34073  at point 102

---------Adjusted Exterior Orientation Xo Yo  Zo (m) omega  phi  kappa in DEGREE(Standard Deviations)-------
Image : st1
   144.806   183.222     0.078    10.620    16.492     1.076
   1.22603   1.17176   4.50383   5.29539   0.75308   1.13575

Image : st2
   138.959   184.025     0.152     4.355    15.066     0.333
   0.01314   0.14853   0.22199  22.04812  33.05544   8.54322

Image : st3
    96.662   213.536     2.915   -45.058    17.582     1.327
   0.02379   0.07986   0.04005   0.43027   3.50553   8.01074

Image : st4
    95.858   237.147     3.885   -88.844    17.420    92.151
   0.57693   0.02691   0.02645   0.26488   6.34744  11.76073

Image : st5
    96.011   236.467     4.258   -76.636    14.220    88.775
   0.00264   0.25572   0.01157   6.08802   0.74372   2.28110

Image : st6
    89.553   265.100     3.578  -117.270    16.910     0.573
```

```
   0.06118   0.01157   0.02418   9.81715   2.82626   1.13848

Image : st7
   115.215   271.352     5.448  -169.035    17.245    90.869
   0.04378   0.10626   0.17134   1.22901   0.30157   1.85012

Image : st8
   133.370   276.043     5.094  -195.593    15.045    -1.866
   0.11078   0.10039   0.00025   2.32480   2.10794   1.13763

Image : st9
   152.250   269.401     3.535   136.435    19.362    90.027
   0.23051   0.03901   0.02419   0.95933   0.41741   0.61635

Image : st11
   172.981   267.835     3.962   128.867    19.273    -3.519
   0.01298   0.01687   0.04170   4.75683  17.96477   6.53327

Image : st12
   186.166   262.241     2.283   107.935    11.627    86.569
   0.02096   0.00999   0.17180  30.93133   1.36891   0.11470

Image : st13
   194.795   268.149     0.861   114.922     7.117   -94.128
   0.04933   0.04170   0.10956   3.55151   1.55187   0.41748

Image : st14
   197.360   264.185     1.420   115.465     4.780   -93.282
   0.00455   0.01207   0.02124   2.90266   3.12438   0.60969

Image : st15
   226.694   177.152     1.233    50.944     9.076    -0.061
   0.00526   0.08302   0.06199   0.18366   8.39069  19.22766

Image : st16
   196.342   178.715     1.207    41.379    10.693    -0.257
   0.00675   0.55937   0.01584   5.92998  27.41354   1.78153

---------Adjusted Camera Constants; xp yp cc in mm(Standard Deviations)-------
Sensor Name : olympus
    -0.191     0.930     8.595
   0.00000   0.00000   0.00000

---------Control Points; x y z in m(Standard Deviations)-------
[Check point std dev followed by std error(given - adjusted)]
  Name            x             y             z      Control Type
-------------------------------------------------------------------------------

   101        118.130       235.510        14.610       Control
              0.037         0.021          0.052

   103        118.200       235.440         3.660       Control
              0.028         0.009          0.057

   111        136.090       237.410        14.650       Control
              0.047         0.002          0.004

   113        136.080       237.450         3.670       Control
              0.049         0.180          0.103

   121        151.990       239.430        14.760       Control
              0.007         0.016          0.006

   123        151.990       239.370         3.690       Control
              0.252         0.019          0.315
```

201	114.770	247.550	14.260	Control
	0.236	0.078	0.009	
202	114.720	247.280	3.610	Control
	0.111	0.008	0.154	
211	115.990	236.070	14.250	Control
	0.314	0.002	0.045	
212	116.030	236.250	3.640	Control
	0.009	0.076	0.011	
301	151.220	256.000	14.790	Control
	0.008	0.103	0.070	
302	151.220	256.000	10.490	Control
	0.007	0.013	0.021	
303	150.750	256.110	3.670	Control
	0.024	0.001	0.002	
311	136.660	254.480	14.680	Control
	0.037	0.146	0.013	
312	136.780	254.020	8.200	Control
	0.019	0.045	0.003	
321	117.010	252.110	14.880	Control
	0.159	0.020	0.005	
322	117.020	252.090	10.550	Control
	0.027	0.045	0.012	
323	117.100	252.060	3.630	Control
	0.022	0.021	0.023	
102	118.111	234.769	10.029	Check
	0.011	0.008	0.009	
	0.019	0.721	0.341	
112	136.181	236.850	10.108	Check
	0.076	0.256	0.001	
	-0.061	0.540	0.262	
122	152.654	239.252	10.280	Check
	0.055	0.860	0.046	
	-0.634	0.138	0.200	
tp11	114.923	232.693	16.471	Tie
	0.047	0.010	0.011	
tp12	115.968	233.849	11.598	Tie
	0.048	0.013	0.174	
tp13	116.047	233.854	7.022	Tie
	0.143	0.011	0.009	
tp41	156.526	237.808	16.922	Tie
	0.146	0.013	0.085	
tp42	154.942	238.744	11.862	Tie
	0.151	0.001	0.025	
tp43	154.700	238.905	7.090	Tie

	0.070	0.003	0.021	
tp21	113.581	252.664	17.613	Tie
	0.121	0.102	0.020	
tp22	114.784	251.332	12.303	Tie
	0.478	0.009	0.320	
tp23	114.732	251.188	7.317	Tie
	0.021	0.001	0.186	
tpn11	115.507	246.479	14.886	Tie
	0.021	0.003	0.020	
tpn13	115.939	246.433	3.563	Tie
	0.013	0.060	0.033	
tp32	153.462	256.777	11.769	Tie
	0.105	0.184	0.029	
tp33	153.574	256.884	7.313	Tie
	0.017	0.023	0.002	
tpe11	126.891	252.279	15.680	Tie
	0.010	0.009	0.007	
tpe12	126.740	252.189	10.743	Tie
	0.034	0.064	0.002	
tpe13	126.553	252.034	3.312	Tie
	0.026	0.072	0.005	
tpe21	145.523	255.395	14.471	Tie
	0.173	0.027	0.010	
tpe22	145.486	255.537	10.094	Tie
	0.001	0.008	0.004	
tpe23	145.232	255.450	3.953	Tie
	0.001	0.045	0.019	
tp31	154.150	257.716	16.749	Tie
	0.002	0.320	0.003	
tps11	154.554	241.370	14.885	Tie
	0.012	0.097	0.023	
tps12	153.966	241.909	3.358	Tie
	0.003	0.134	0.003	
tps21	153.011	253.908	14.826	Tie
	0.001	0.025	0.041	
tps22	152.996	254.424	3.911	Tie
	0.001	0.016	0.007	

APPENDIX C : OUTPUT FILE FOR GPS DATA ADJUSTMENT

The input files for the GPS data testing were not included because they are in RINEX data format already explained in chapter four.

The output file for the single epoch relative positioning of phase data is shown here. The first column is the *a posteriori* standard deviation after least squares adjustment of the double differenced phase measurement. The second column shows the epoch number in sequential order and the next three columns show the X,Y and Z vector components of the baseline in metres. This result is graphically demonstrated in Chapter 6.

(Standard Deviation) Epoch No. Dx Dy Dz

(0.030) 1 103.422 -1169.394 -114.864
(0.012) 2 103.387 -1169.404 -114.898
(0.021) 3 103.357 -1169.387 -114.889
(0.023) 4 103.459 -1169.372 -114.789
(0.021) 5 103.444 -1169.379 -114.796
(0.012) 6 103.489 -1169.376 -114.724
(0.025) 7 103.458 -1169.391 -114.730
(0.022) 8 103.538 -1169.371 -114.670
(0.024) 9 103.471 -1169.403 -114.725
(0.015) 10 103.439 -1169.402 -114.745
(0.012) 11 103.434 -1169.394 -114.760
(0.006) 12 103.463 -1169.387 -114.751
(0.018) 13 103.447 -1169.381 -114.764
(0.023) 14 103.435 -1169.395 -114.771
(0.010) 15 103.546 -1169.398 -114.694
(0.017) 16 103.563 -1169.401 -114.662
(0.012) 17 103.514 -1169.383 -114.716
(0.009) 18 103.541 -1169.382 -114.667
(0.003) 19 103.476 -1169.422 -114.702
(0.009) 20 103.539 -1169.389 -114.679
(0.023) 21 103.572 -1169.380 -114.658
(0.023) 22 103.560 -1169.403 -114.680
(0.025) 23 103.595 -1169.407 -114.672
(0.031) 24 103.564 -1169.398 -114.679
(0.030) 25 103.562 -1169.400 -114.677
(0.034) 26 103.588 -1169.403 -114.637
(0.034) 27 103.542 -1169.389 -114.665
(0.021) 28 103.516 -1169.406 -114.704
(0.023) 29 103.517 -1169.415 -114.702
(0.011) 30 103.572 -1169.398 -114.698
(0.010) 31 103.538 -1169.410 -114.749
(0.013) 32 103.581 -1169.434 -114.701
(0.016) 33 103.495 -1169.410 -114.727
(0.020) 34 103.458 -1169.415 -114.752
(0.022) 35 103.582 -1169.422 -114.694
(0.023) 36 103.426 -1169.427 -114.786
(0.019) 37 103.553 -1169.405 -114.666
(0.019) 38 103.527 -1169.404 -114.703
(0.030) 39 103.417 -1169.436 -114.805

```
////////////////////////////////////////////////////////////////////
file defining global variables used in the program :  global.h
////////////////////////////////////////////////////////////////////

#include <iostream.h>
#include <fstream.h>
#include <IOMANIP.H>
#include <afxtempl.h> //for CList

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdarg.h>
#include <math.h>

#include "constants.h"

class CMappingMatrix;

enum ImageSensorType { DIGITAL, FRAME, SCAN};
enum PointType {CONTROL, TIE, CHECK};

enum GPSSensorType { GPS1, GPS2 };
enum INSSensorType { INS1, INS2 };
enum USEGPS {NOGPS, WITHGPS};
enum USEINS {NOINS, WITHINS};

enum ANGLEUNIT {RADIAN, DEGREE, GON};


//for matrix indexing
#define GetIndex(row,col) (col + row*(row-1)/2)

//used in GaussJordan Elimination
#define SWAP(a,b) {double temp=(a);(a)=(b);(b)=temp;}


extern int nTiePoint; //number of tie points
extern int nTieEstimate;
extern int nFullControl; // number of controls
extern int nCheckPoint;  // number of check points
extern int nImage;       // number of images
extern int nImgSensor;   // number of imaging sensors
extern int nImageObservation; // total number of image observations
extern int GPS;
extern int INS;
extern int nRedundancy;
extern int nDimension; //total number of memory allocation
extern int nObs;
extern int nPoint;  //number of points (control and tie points)
extern int nImageParameter;
extern int nImgSensorParameter;
extern int nPointParameter;
extern int nUnknown;
extern double ang;

extern double APRIORI_IMG; //Image Measurement Accuracy
extern double APRIORI_XYP; //Principal Point Coordinate Accuracy
extern double APRIORI_CC ; //Principal Distance Accuracy
extern double APRIORI_K1;
extern double APRIORI_K2;
extern double APRIORI_K3;
extern double APRIORI_P1;
extern double APRIORI_P2;
extern double APRIORI_A;
```

```
extern double APRIORI_B;
extern double APRIORI_ROT; //Rotation angles Accuracy
extern double APRIORI_PER; //Perspective Point Coordinate Accuracy
extern double APRIORI_SXY; //Planimetry Accuracy
extern double APRIORI_SZ ; //Height Accuracyextern  double  ConvergeTest;

extern  int  nMaxIteration;
extern  double ConvergeTest;
extern  int  AngleUnit;
extern  int  ImageSensorType;
extern  BOOL GetEstimate;
extern  BOOL AdditionalParameter;
extern  BOOL USEDLT_SENSOR;
extern  BOOL USECOLL_POINT;
extern  BOOL SimulateData;
extern  BOOL Platform;

extern ImageList Images;
extern ControlPointList ControlPoints;
extern ImageSensorList ImageSensors;
extern ImagePointList ImagePoints;

extern void CholeskyDecompose(double* NormalMat, int nRows);
extern void CholeskySolve(double* NormalMat, double* SolutionVec, int nRows);
extern double* SolveNormal(double *NormalMat,double *SolutionVec,int nRows);
extern void Inverse(double *NormalMat,int nRows);
extern void GaussJordan(CMappingMatrix& a, CMappingMatrix& b);

//////////////////////////////////////////////////////////////////////
typedef struct
{
        double K1,K2,K3,P1,P2,A, B;
        double dK1,dK2,dK3,dP1,dP2,dA,dB;
}AddParam_Struct;

//////////////////////////////////////////////////////////////////////
// Class:           CAdditionalParameter
//
// Implementation Status : Completed (200 program lines)
//
//////////////////////////////////////////////////////////////////////
class CAdditionalParameter
{
friend class CProject;
friend class CNormalEq;
friend class CAdjustment;

// Constructor(s) & destructor
public:
        CAdditionalParameter();
        virtual ~CAdditionalParameter();

// Attributes
private:
        AddParam_Struct  AdditionalParameter;
        AddParam_Struct  InitialAdditionalParameter;

// Operations
public:
        AddParam_Struct GetAdditionalParameters()
                { return AdditionalParameter; }
        AddParam_Struct GetInitialAdditionalParameter()
                { return InitialAdditionalParameter; }
        void SetAdditionalParameter(double K1, double K2, double K3, double P1, double P2, double A,
double B);
```

```
        void SetInitialAdditionalParameter(double K1, double K2, double K3, double P1, double P2, double A,
double B);
        void PrintAddParam(ofstream& filethis);
        void PrintInitialAddParam(ofstream& filethis);
        void PrintAddParamStdDev(ofstream& filethis);
        void PrintHeader(ofstream& filethis);
};

////////////////////////////////////////////////////////////////////
// Class:           CSensor
//
// Implementation Status : Completed (Empty Class)
//
////////////////////////////////////////////////////////////////////
class CSensor
{

// Constructor(s) & destructor
public:
        CSensor();
        virtual ~CSensor();

// Attributes
        char        SensorName[NameLength];
        int         SensorType;

// Operations
};

////////////////////////////////////////////////////////////////////
// Class:           CSensorGPS
//
// Implementation Status : Completed (Empty Class)
//
////////////////////////////////////////////////////////////////////
class CSensorGPS : public CSensor
{
friend class CProject;

// Constructor(s) & destructor
public:
        CSensorGPS();
        virtual ~CSensorGPS();
// Attributes

// Operations
};

////////////////////////////////////////////////////////////////////
// Class:           CSensorIMG
//
// Implementation Status : Completed (79 program lines)
//
////////////////////////////////////////////////////////////////////
class CSensorIMG : public CSensor
{
friend class CDLT;
friend class CProject;
friend class CNormalEq;
friend class CAdjustment;
friend class CMappingImage;

// Constructor(s) & destructor
public:
        CSensorIMG();
```

```
        virtual ~CSensorIMG();

// Attributes
        CPrincipalPoint* pm_PrincipalPt;
        CAdditionalParameter* pm_AddParam;
        double      InitialPrincipalDistance;
        // this indicates type of sensor (0=digital, 1=frame, 2=scanned image)
        int     type;
        // case digital camera
        double      SensorElementHeight;
        double      SensorElementWidth;
        // case scanned image with fiducial coords given
        int     VerPixelsNo;
        int     HorPixelsNo;
        int                         ScanResolution;
        double    Fiducial_photo[3][5];

        double      PrincipalDistance;
        double      PrincipalDistanceAccuracy;
        static int          count;
        int       index;

        //Std Dev calculated for DLT adjustment. This value is used in the approximation of
        //image and sensor parameters where the smallest value will be select in deciding
        // the value for xp, yp and cc
        double      CalibratedStdDev;

// Operations
public:
        CAdditionalParameter* GetpAddParam()
                { return pm_AddParam; }
        CPrincipalPoint* GetPrincipalPt()
                { return pm_PrincipalPt; }
        double GetPrincipalDistance()
                { return PrincipalDistance; }
        double GetInitialPrincipalDistance()
                { return InitialPrincipalDistance; }
        double GetSensorElementWidth()
                { return SensorElementWidth; }
        double GetSensorElementHeight()
                { return SensorElementHeight; }
        void PrintSensorIMG(ofstream& filethis);
        static int howManyIMGSensors()
                { return count; }
};

////////////////////////////////////////////////////////////////////
// Class:           CSensorINS
//
// Implementation Status : Completed (Empty Class)
//
////////////////////////////////////////////////////////////////////
class CSensorINS : public CSensor
{
friend class CProject;

// Constructor(s) & destructor
public:
        CSensorINS();
        virtual ~CSensorINS();

// Attributes

// Operations
};
```

```
////////////////////////////////////////////////////////////////////
// Class:              CProject
//
// Implementation Status : Completed (1,658 program lines)
//
////////////////////////////////////////////////////////////////////
class CProject
{
// Constructor(s) & destructor
public:
        CProject();
        virtual ~CProject();

// Attributes
public:
        char        imgfile[NameLength];
        char        gcpfile[NameLength];
        char        camfile[NameLength];
        char        gpsfile[NameLength];
        char        insfile[NameLength];
        char                    outfile[NameLength];
        char                    simfile[NameLength];

// Operations
public:
        void SetImages();
        CMappingImage* GetImage(char* imgName);
        void SetIMGSensor();
        void SetGPSSensor();
        void SetINSSensor();
        void SetControlPoints();
        void ReadTiePointEstimate();
        void ComputeEstimate();
        void ComputeEstimatePoint_2DLinear();
        void ComputeEstimate_Image();
        void ComputeEstimatePoint_Coll();
        void Simulate();
        void DeleteAllControls();
        void DeleteAllImages();
        void DeleteAllImageSensors();
        void Adjust();
};

////////////////////////////////////////////////////////////////////
typedef struct
{
        double MaximumResidual;
        char  ImageName[NameLength];
        char  PointName[NameLength];
        double Sigma0;
}Residual_Struct;

typedef struct
{
        double MaximumError_XY;
        char  PointName_XY[NameLength];
        double MaximumError_Z;
        char  PointName_Z[NameLength];
        double RMS_XY;
        double RMS_Z;
}RMS_Struct;
```

```
//////////////////////////////////////////////////////////////////
// Class:          CLSQAdjustment
//
// Implementation Status : Completed (380 program lines)
//
//////////////////////////////////////////////////////////////////
class CLSQAdjustment
{
friend class CProject;

public:
        CLSQAdjustment();
        CLSQAdjustment(CEquation* Equation);

        virtual ~CLSQAdjustment();
private:
        double *pm_NormalMatrix, *pm_ParameterVec;
        double DiscrepancySquareSum;
        int nUnknown;
        int nDimension;
        int nObs;
        int nRedundancy;
        int MaxIteration;
        double AdjustedUnitVariance;
        double MaxResidual;
        double AdjustedParameterPrecision;
        CMappingMatrix Residual;

public:

        //form normal eq
        void FormNormal(CEquation* Equation);
        void FormNormal(CMappingMatrix& DesignMat,int nRows,
                                                        int nCols,double variance);
        void FormRHS(CMappingMatrix& DesignMat, CMappingMatrix& DiscrepancyVec, int nRows, int
nCols, double variance);

        void Solve();//solve for given normal eq.(one iteration)
        void Compute_DiscrepancySquareSum(CMappingMatrix& DiscrepancyVec, int nRows);//square of
misclosure
        void ComputeAdjustedResidual(CMappingMatrix &DesignMat, CMappingMatrix &ConstantVec,
                                                                CMappingMatrix
&ParameterVec, CMappingMatrix &UnitWeight);
        void Initialize();
        void CholeskyDecompose(double* NormalMat, int nRows);
        void CholeskySolve(double* NormalMat, double* SolutionVec, int nRows);
        void LInverse(double *NormalMat,int nRows);//for Lower Triangular Matrix, used with Cholesky
Decompose
        void UpdateParameter(CMappingMatrix& InitialParVal);
};

//////////////////////////////////////////////////////////////////
// Class:          CDLT
//
// Implementation Status : Completed (530 program lines)
//
//////////////////////////////////////////////////////////////////
class CDLT
{
friend class CProject;
// Constructor(s) & destructor
public:
        CDLT();
        virtual ~CDLT();
```

```
// Attributes
public:
        double xp, yp, cx, cy, cc;
        double Xo, Yo, Zo, omega, phi, kappa;
        double m11, m12, m13, m21, m22, m23, m31, m32, m33;//rotation matrix element
        double affinity_a, affinity_b;//affinity

private:
        double *DLT_Parameter, *DLT_Normal;
        double *DLT_TiePtNormal, *DLT_TiePtParameter;
        double L11,L12,L13,L14,L21,L22,L23,L24,L31,L32,L33;
        double ximage, yimage, X, Y, Z;
  double DisSquareSum;
        double Sigma_dlt;
        double A,B,C;
        BOOL Iterative;
        int     TiePtIndex;
        int nTiePt_Parameter;
        int nTiePt_Dimension;
        CMappingMatrix DesignMat_DLT,DLTMat_Constant;
        CMappingMatrix DesignMat_TiePt,TiePt_Constant;

// Operations
public:
        // (Operation description)
        void FormDesignMat_DLT();
        void FormDLT_Constant();
        void FormNormal_DLT();
        void FormRHS_DLT();
        void FormDLT(CMappingImage* pIMG);//Forms the DLT normal equations for the image
        void Initialize_Normal();
        void ComputeTiePoint();
        void FormDesignMat_TiePt();
        void FormConstant_TiePt();
        void FormNormal_TiePt();
        void FormRHS_TiePt();
};

//////////////////////////////////////////////////////////////////////
typedef struct
{
        double omega, domega;
        double phi, dphi;
        double kappa, dkappa;
}Rotation_Angle;

//////////////////////////////////////////////////////////////////////
// Class:               CMappingImage
//
// Implementation Status : Completed (256 program lines)
//
//////////////////////////////////////////////////////////////////////
class CMappingImage
{
friend class CProject;
friend class CNormalEq;
friend class CDLT;
friend class CAdjustment;

// Constructor(s) & destructor
public:
        CMappingImage();
        virtual ~CMappingImage();

// Attributes
```

```
public:
        CRotation*      pm_Rotation;
        CPerspectivePoint* pm_PerspectivePt;
        CSensorIMG*     pm_SensorIMG;
        char            ImageName[NameLength];
        ImagePointList* pm_ImagePoints;
        static int      count;
  int       index;
        int             TotalPoints;

        //this value is assigned after computing the estimated a,b,Xo,Yo by 2D linear conformal
        //transformation. The image with smallest StdDev will be used to assign tie point estimate
        double          StdDev2D;

        //observed pixel coordinates of fiducial marks which will be used for affine transformation
        double          Fiducial_pixel[3][5];
        double          AffineParameters[7];
        CMappingMatrix          MatrixSolution;

// Operations
public:
        ImagePointList* GetImagePoints()
                { return pm_ImagePoints; }
        static int howManyImages()
                { return count; }
        int GetTotalPoints()
                { return TotalPoints; }
        void AddImagePoint(char* ptname, double x, double y, double dx, double dy);
        void DeleteImagePt(char* ptname);

        // do the affine transformation for scanned images,
        //return value is affine[1], affine[2],..., six transformation parameters
        //the fiducial pixel coordinate of this image is used with the fiducial photo
        //coordinates of the sensor used to capture this image
        void AffineTransform();
};

//////////////////////////////////////////////////////////////////////
// Class:               CRotation
//
// Implementation Status : Completed (216 program lines)
//
//////////////////////////////////////////////////////////////////////
class CRotation
{
friend class CProject;
friend class CNormalEq;
friend class CAdjustment;

// Constructor(s) & destructor
public:
        CRotation();
        virtual ~CRotation();

// Attributes
public:
        CSensorINS*     pm_SensorINS;
        CMappingMatrix          m_RotationMat;
        Rotation_Angle  Rotation;
        Rotation_Angle  InitialRotation;

// Operations
public:
        void SetRotation(double omega, double phi, double kappa,
                                                double domega, double dphi, double dkappa);
```

```
        void SetRotation(double omega, double phi, double kappa);
        void SetInitialRotation(double omega, double phi, double kappa,
                                                    double domega, double dphi, double dkappa);
        void SetInitialRotation(double omega, double phi, double kappa);
        void PrintRotation(ofstream& filethis);
        void PrintInitialRotation(ofstream& filethis);
};


////////////////////////////////////////////////////////////////////
// Class:           realArray
//
// Implementation Status : Completed (86 program lines)
//
////////////////////////////////////////////////////////////////////
class realArray
{
friend class matMap;

// Constructor(s) & destructor
public:
        realArray(INDEX nr, INDEX nc);
        virtual ~realArray();

// Attributes
private:
        double*     *base;
        INDEX       nrow, ncol;

// Operations
private:
        double& elem(INDEX i, INDEX j) const;
        double& operator()(INDEX i, INDEX j) const
                { return elem(i,j) ; }
        DWORD size(void ) const
                { return (DWORD) nrow * (DWORD) ncol ; }
        realArray& operator =(realArray& y);
};

////////////////////////////////////////////////////////////////////
// Class:           matMap
//
// Implementation Status : Completed (145 program lines)
//
////////////////////////////////////////////////////////////////////
class matMap
{
// Constructor(s) & destructor
public:
        matMap(void);
        matMap(INDEX nr, INDEX nc = 1);
        virtual ~matMap();

// Attributes
private:
        realArray*    pa;
        double*      *map;
        INDEX         mapSize, nrow, ncol;

// Operations
public:
        double* base(INDEX j) const
                { return pa->base[j] ; }
        double& operator()(INDEX i) const
                { return map[1][i] ; }
```

```
        double& operator()(INDEX i, INDEX j) const
                { return map[j][i] ; }
        INDEX nRows(void ) const
                { return nrow ; }
        INDEX nCols(void ) const
                { return ncol ; }
        matMap& operator =(const matMap& m);
        realArray* array(void ) const
                { return pa ; }
        void reset(INDEX nr, INDEX nc);
        void clear(void );

// Friend classes
friend class CMappingMatrix;
};

//////////////////////////////////////////////////////////////////////
// Class:          CMappingMatrix
//
// Implementation Status : Completed (142 program lines)
//
//////////////////////////////////////////////////////////////////////
class CMappingMatrix
{
// Constructor(s) & destructor
public:
        CMappingMatrix();
        CMappingMatrix(INDEX nr, INDEX nc = 1);
        virtual ~CMappingMatrix();

// Attributes
private:
        matMap*      pm;

// Operations
public:
        virtual CMappingMatrix& operator =(const CMappingMatrix& y);
        double& operator()(INDEX i, INDEX j) const
                { return pm->map[j][i] ; }
        double& operator()(INDEX i) const
                { return pm->map[1][i] ; }
        INDEX nCols(void ) const;
        INDEX nRows(void ) const;
        virtual void reset(INDEX nr, INDEX nc);
        virtual void reset(INDEX nr);
        virtual void clear(void );
};

//////////////////////////////////////////////////////////////////////
// Class:          CEquation
//
// Implementation Status : Empty Class
//
//////////////////////////////////////////////////////////////////////
class CEquation
{
friend class CLSQAdjustment;

public:
        CEquation();
        virtual ~CEquation();

private:
        double *pm_NormalMatrix, *pm_ParameterVec;
        int NumberOfObs, NumberOfUnknown;
```

```
public:
        virtual void FormNormalEquation();
};

//////////////////////////////////////////////////////////////////
// Class:           CCollinearityEquation
//
// Implementation Status : Completed (300 program lines)
//
//////////////////////////////////////////////////////////////////
class CCollinearityEquation
{
friend class CBundleEquation;

public:
        CCollinearityEquation();
        virtual ~CCollinearityEquation();

        // Form Observation Equation Submatrices
        void Initialise_Design(CImagePoint* pImgPt);
        void Initialise_Image(CMappingImage* pIMG);
        void ComputeDesignMatrix(CImagePoint* pImgPt);
        void FormDesignMat_Constant();
        void FormDesignMat_Image();
        void FormDesignMat_Sensor();
        void FormDesignMat_Point();

private:
        double r11, r12, r13, r21, r22, r23, r31, r32, r33;//rotation matrix
        double xp, yp ,PrincipalDistance; //sensor parameters
        double K1, K2, K3, P1, P2, A, B;  //lens distortion and affinity parameters
        double xcoord, ycoord;
        double delta_x, delta_y;
        double X, Y, Z, Xo, Yo, Zo;//ground coordinates and perspective center
        double COLL_N, COLL_Zx, COLL_Zy, M1, M2;//denominator and numerators of the collinearity
equation
        double CosOm, CosPhi, CosKap, SinOm, SinPhi, SinKap;
        double radius2, radius4, radius6;

        //Observation Equation Submatrices
        CMappingMatrix    DesignMat_Image, DesignMat_Sensor, DesignMat_Point;
        CMappingMatrix    DesignMat_Constant;
};

//////////////////////////////////////////////////////////////////
// Class:           CDirectObservationEquation
//
// Implementation Status : Completed (132 program lines)
//
//////////////////////////////////////////////////////////////////
class CDirectObservationEquation
{
friend class CBundleEquation;

public:
        CDirectObservationEquation();
        virtual ~CDirectObservationEquation();

        void ComputeContribution_Image(CMappingImage* pIMG);
        void ComputeContribution_Point(CControlPoint* pCntPt);
        void ComputeContribution_Sensor(CSensorIMG* pImgSensor);
private:
        double image_discrepancy[7], sensor_discrepancy[11], point_discrepancy[4];
        double image[7], sensor[11], point[4];
```

```
};

//////////////////////////////////////////////////////////////////
typedef struct
{
        double x,y,z;
        double dx,dy,dz;
}Coordinate_Struct;

//////////////////////////////////////////////////////////////////
// Class:               CGenericPoint
//
// Implementation Status : Completed (327 program lines)
//
//////////////////////////////////////////////////////////////////
class CGenericPoint
{
friend class CProject;
friend class CMappingImage;
friend class CNormalEq;
friend class CDLT;
friend class CAdjustment;

// Constructor(s) & destructor
public:
        CGenericPoint();
        virtual ~CGenericPoint();

// Attributes
public:
        CoordinateUnit  unit;
        ProjectionType  projection;

protected:
        char            PtName[NameLength];
        Coordinate_Struct InitialCoordinate;
        Coordinate_Struct PtCoordinate;
        Coordinate_Struct CheckCoordinate;

// Operations
public:
        char* GetPtName()
                { return PtName; }
        Coordinate_Struct GetPtCoordinate()
                { return PtCoordinate; }
        void SetPtName(char* aName)
                { strcpy(PtName,aName); }
        void SetPtCoordinate(double x, double y, double z);
        void SetInitialCoordinate(double x, double y, double z);
        void SetPtCoordinate(double x, double y, double z, double dx, double dy, double dz);
        void SetInitialCoordinate(double x, double y, double z, double dx, double dy, double dz);
        void SetPtCoordinate(double x, double y, double dx, double dy);
        void SetInitialCoordinate(double x, double y, double dx, double dy);
        void SetPtCoordinate(double z, double dz);
        void SetInitialCoordinate(double z, double dz);
        void SetCheckCoordinate(double x, double y, double z, double dx, double dy, double dz);
        void EmptyCoordinates();
        void EmptyInitialCoordinates();
        virtual void PrintCheckCoordinate(ofstream& filethis);
        virtual void PrintInitialCoordinate(ofstream& filethis);
        virtual void PrintCoordinate(ofstream& filethis);
        virtual void PrintCoordinateAccuracy(ofstream& filethis);
};

//////////////////////////////////////////////////////////////////
```

```
// Class:              CControlPoint
//
// Implementation Status : Completed (74 program lines)
//
////////////////////////////////////////////////////////////////////
class CControlPoint : public CGenericPoint
{
friend class CDLT;
friend class CProject;
friend class CMappingImage;
friend class CNormalEq;
friend class CAdjustment;

// Constructor(s) & destructor
public:
        CControlPoint();
        virtual ~CControlPoint();

// Attributes
public:
        int type; //CONTROL, TIE or CHECK

protected:
        ImagePointList*  pm_ImagePoints;
        int          m_nRays;
  int        index;

private:
        static int     count;

// Operations
public:
        ImagePointList* GetImagePoints();
        static int howManyPoints()
                { return count; }
        void PrintRays(ofstream& filethis);
};

////////////////////////////////////////////////////////////////////
// Class:              CMappingImagePoint
//
// Implementation Status : Completed (47 program lines)
//
////////////////////////////////////////////////////////////////////
class CMappingImagePoint : public CGenericPoint
{
friend class CProject;
friend class CNormalEq;
friend class CDLT;
friend class CAdjustment;

// Constructor(s) & destructor
public:
        CMappingImagePoint();
        virtual ~CMappingImagePoint();

// Attributes
private:
        CControlPoint*   pm_ControlPoint;
        CMappingImage*       pm_Image;
        static int     count;

// Operations
public:
        CControlPoint* GetLinkedControlPoint()
```

```
                { return pm_ControlPoint; }
        CMappingImage* GetImage()
                { return pm_Image; }
        static int howManyPoints()
                { return count; }
        void PrintCoordinate(ofstream& filethis);

// Friend classes
friend class CMappingImage;
};

////////////////////////////////////////////////////////////////////
// Class:               CPerspectivePoint
//
// Implementation Status : Completed (53 program lines)
//
////////////////////////////////////////////////////////////////////
class CPerspectivePoint : public CGenericPoint
{
friend class CProject;

// Constructor(s) & destructor
public:
        CPerspectivePoint();
        virtual ~CPerspectivePoint();

// Attributes
private:
        CSensorGPS*     pm_SensorGPS;
        static int      count;

// Operations
public:
        static int howManyPoints()
                { return count; }
        void PrintCoordinate(ofstream& filethis);
        void PrintInitialCoordinate(ofstream& filethis);
};

////////////////////////////////////////////////////////////////////
// Class:               CPrincipalPoint
//
// Implementation Status : Completed (62 program lines)
//
////////////////////////////////////////////////////////////////////
class CPrincipalPoint : public CGenericPoint
{

// Constructor(s) & destructor
public:
        CPrincipalPoint();
        virtual ~CPrincipalPoint();

// Attributes
private:
        static int      count;

// Operations
public:
        static int howManyPoints()
                { return count; }
        void PrintCoordinate(ofstream& filethis);
        void PrintInitialCoordinate(ofstream& filethis);
};
```

```
////////////////////////////////////////////////////////////////////
// Class:               CTiePoint
//
// Implementation Status : Completed (29 program lines)
//
////////////////////////////////////////////////////////////////////
class CTiePoint : public CControlPoint
{

// Constructor(s) & destructor
public:
        CTiePoint();
        virtual ~CTiePoint();

// Attributes
private:
        static int      count;

// Operations
public:
        static int howManyPoints()
                { return count; }
};

////////////////////////////////////////////////////////////////////
// Class:               CFullControl
//
// Implementation Status : Completed (29 program lines)
//
////////////////////////////////////////////////////////////////////
class CFullControl : public CControlPoint
{

// Constructor(s) & destructor
public:
        CFullControl();
        virtual ~CFullControl();

// Attributes
private:
        static int      count;

// Operations
public:
        static int howManyPoints()
                { return count; }
};
```

APPENDIX E : SOURCE CODE LISTING FOR CLASS INTERFACES OF GPS DATA PROCESSING PROGRAM

In this appendix, the source code listing for the class interfaces used in the GPS data processing program is presented. As in Appendix D, each class is introduced as it has been implemented in its header file (*.h) as well as the the size of the implementation file in number of lines.

```
//////////////////////////////////////////////////////////////////
// Class:              CClockTime
//
// Implementation Status : Completed (150 program lines)
//
//////////////////////////////////////////////////////////////////
class CClockTime
{
// Constructor(s) & destructor
public:
        CClockTime();
        virtual ~CClockTime();

// Attributes
public:
        double      Second;

        // 2 digits
        int         Year;
        int         Month;
        int         Day;
        int         Hour;
        int         Minute;

        // Julian Day
        double      JulianDay;
        double      ModifiedJulianDay;
        int         GPSWeek;
        int         DayofWeek;

        //GPS second of week
        double      GPSSec;
        BOOL        IsTimeSet;

//member functions
public:
        // Julian day is computed from given time of clock
        void ToJulian();
        void SetTime(int aYear, int aMonth, int aDay, int aHour, int aMinute, double aSecond);

        // computes GPS time as seconds of week and puts this value into the data member Epoch
        void ToGPSSec();
        void CheckTime(double aTime);
        void Output(ofstream& filethis);
};
```

```
//////////////////////////////////////////////////////////////////
struct CSingleDiff
{
        double GPSSec;
        int PRN;
        double phaseL1, phaseL2;
        double codeP1, codeP2;
        CPosition SVpos;
};

//////////////////////////////////////////////////////////////////
// Class:           CDoubleDiff
//
// Implementation Status : Completed (194 program lines)
//
//////////////////////////////////////////////////////////////////
class CDoubleDiff
{
friend class CGPSObservation;

public:
        CDoubleDiff();
        virtual ~CDoubleDiff();

        void ComputeDirectionCosine(CStation* St1, CPosition& aPos);
        void ComputeResidual_L1();
        void ComputeResidual_L2();
        void ComputeResidual_Lw();
        void ComputeResidual_P1();
        void ComputeResidual_P2();

private:
        CSingleDiff RefSobs;
        CSingleDiff OtherSobs;
        double phaseL1, phaseL2, phaseLw;//double differenced observed value
        double codeP1, codeP2;//double differenced observed value
        double rho_st2sv1, rho_st2sv2, rho_st1sv1, rho_st1sv2;//distance to the satellite
        double DirectionCosine_x, DirectionCosine_y, DirectionCosine_z;//first three (positional)components
of the design matrix
        double ddResidual_L1;//constant component of the rhs of the obs eq.
        double ddResidual_L2;
        double ddResidual_Lw;
        double ddResidual_P1;
        double ddResidual_P2;
        double intParL1;//DD integer ambiguity for L1
        double intParL2;//DD integer ambiguity for L2
        double intParLw;//DD integer ambiguity for Lw
        BOOL L1_Resolved;
        BOOL L2_Resolved;
        BOOL Lw_Resolved;
        BOOL IsPrimary;
        BOOL CycleSlip_L1;
        BOOL CycleSlip_L2;
};

//////////////////////////////////////////////////////////////////
// Class:           CEventObs
//
// Implementation Status : Completed (28 program lines)
//
//////////////////////////////////////////////////////////////////
class CEventObs
{
// Constructor(s) & destructor
public:
```

```
        CEventObs();
        virtual ~CEventObs();
//attribute
public:
        // Time of observation
        CClockTime    m_ClockTime;//event epoch
        CClockTime    m_ClockTime2;//in case of lock loss end of lock loss

        // Describes type of event
        //enum EVENT {CycleSlip, LockLoss, PhotoExposure, RejectedObs, Other};
        int         EventType;
        int         PRN;//in case of cycleslip or lockloss
        int         PhaseType;
        CString       FileName;
        CString       StationName1;
        CString       StationName2;
        int         SessionNo;
};

////////////////////////////////////////////////////////////////////
// Class:              CLSQAdjustment
//
// Implementation Status : Completed (367 program lines)
//
////////////////////////////////////////////////////////////////////
class CLSQAdjustment
{
        friend class CGPSObservation;
        friend class CSpaceVehicle;

public:
        CLSQAdjustment();
        virtual ~CLSQAdjustment();

        CLSQAdjustment(int nPar, int nEq);

private:
        //Collapsed Full Normal Matrix and RightHandSide Vector
        //index of the matrix is accessed by the mapping function GetIndex(row, col)
        double *pm_NormalMatrix, *pm_ParameterVec;

        double DiscrepancySquareSum;
        int nUnknown;
        int nDimension;
        int nObs;
        int nRedundancy;
        int MaxIteration;
        double AdjustedUnitVariance;
        double MaxResidual;
        double AdjustedParameterPrecision;
        CMappingMatrix Residual;

public:

        //form normal eq
        void FormNormal(CMappingMatrix& DesignMat, int nRows, int nCols, double variance);
        void FormRHS(CMappingMatrix& DesignMat, CMappingMatrix& DiscrepancyVec, int nRows, int
nCols, double variance);
        void Solve();//solve for given normal eq.(one iteration)
        void Compute_DiscrepancySquareSum(CMappingMatrix& DiscrepancyVec, int nRows);//square of
misclosure
        void ComputeAdjustedResidual(CMappingMatrix &DesignMat, CMappingMatrix &ConstantVec,
                                                                        CMappingMatrix
&ParameterVec, CMappingMatrix &UnitWeight);
        void Initialize();
```

```
        void CholeskyDecompose(double* NormalMat, int nRows);
        void CholeskySolve(double* NormalMat, double* SolutionVec, int nRows);
        void LInverse(double *NormalMat,int nRows);//for Lower Triangular Matrix, used with Cholesky
Decompose
        void UpdateParameter(CMappingMatrix& InitialParVal);
};

//////////////////////////////////////////////////////////////////////
// Class:           realArray
//
// Implementation Status : Completed (101 program lines)
//
//////////////////////////////////////////////////////////////////////
class realArray
{
friend class matMap;

// Constructor(s) & destructor
public:
        realArray(INDEX nr, INDEX nc);
        virtual ~realArray();

// Attributes
private:
        double*     *base;
        INDEX       nrow, ncol;

// Operations
private:
        double& elem(INDEX i, INDEX j) const;
        double& operator()(INDEX i, INDEX j) const
                { return elem(i,j) ; }
        DWORD size(void ) const
                { return (DWORD) nrow * (DWORD) ncol ; }
        realArray& operator =(realArray& y);
};

//////////////////////////////////////////////////////////////////////
// Class:           matMap
//
// Implementation Status : Completed (146 program lines)
//
//////////////////////////////////////////////////////////////////////
class matMap
{
// Constructor(s) & destructor
public:
        matMap(void);
        matMap(INDEX nr, INDEX nc = 1);
        virtual ~matMap();

// Attributes
private:
        realArray*    pa;
        double*     *map;
        INDEX       mapSize, nrow, ncol;

// Operations
public:
        double* base(INDEX j) const
                { return pa->base[j] ; }
        double& operator()(INDEX i) const
                { return map[1][i] ; }
        double& operator()(INDEX i, INDEX j) const
                { return map[j][i] ; }
```

```
        INDEX nRows(void ) const
                { return nrow ; }
        INDEX nCols(void ) const
                { return ncol ; }
        matMap& operator =(const matMap& m);
        realArray* array(void ) const
                { return pa ; }
        void reset(INDEX nr, INDEX nc);
        void clear(void );

// Friend classes
friend class CMappingMatrix;
};

//////////////////////////////////////////////////////////////////
// Class:          CMappingMatrix
//
// Implementation Status : Completed (690 program lines)
//
//////////////////////////////////////////////////////////////////
class CMappingMatrix
{
// Constructor(s) & destructor
public:
        CMappingMatrix();
        CMappingMatrix(INDEX nr, INDEX nc = 1);
        virtual ~CMappingMatrix();

// Attributes
private:
        matMap*     pm;

// Operations
public:
        // all argument matrix are destroyed
        virtual CMappingMatrix& operator =(const CMappingMatrix& y);
        virtual CMappingMatrix& operator +(const CMappingMatrix& x) ;  // Mat+Mat
        virtual CMappingMatrix& operator -(const CMappingMatrix& x) ;  // Mat-Mat
        virtual CMappingMatrix& operator !(); // transpose

        //arguments matrix values are preserved
        CMappingMatrix& add(CMappingMatrix& x, CMappingMatrix& y) ;
        CMappingMatrix& subtract( const CMappingMatrix& x, const CMappingMatrix& y);
        CMappingMatrix& multiply( const CMappingMatrix& x, const CMappingMatrix& y);
        CMappingMatrix& transpose(CMappingMatrix& x);
        CMappingMatrix& inverse(CMappingMatrix& y);

        friend void SolveMatrix(CMappingMatrix& a, CMappingMatrix& b);//returns the solution in b and the
inverse in a
        CMappingMatrix& inv(CMappingMatrix& x);//luDecompse and returns the inverse matrix
        CMappingMatrix& zeros(INDEX nr, INDEX nc); //creates a zero matrix of nr x nc
        CMappingMatrix& ones(INDEX nr, INDEX nc); //creates a zero matrix of nr x nc
        CMappingMatrix& eye(INDEX nr, INDEX nc); //creates a zero matrix of nr x nc
        double& operator()(INDEX i, INDEX j) const
                { return pm->map[j][i] ; }
        double& operator()(INDEX i) const
                { return pm->map[1][i] ; }
        double& mat( INDEX i, INDEX j = 1 ) const
    { return pm->map[j][i] ; }
        INDEX nCols(void ) const;
        INDEX nRows(void ) const;
        virtual void reset(INDEX nr, INDEX nc);
        virtual void reset(INDEX nr);
        virtual void clear(void );
        void GaussJordan(CMappingMatrix& a, CMappingMatrix& b);
```

```
};


//////////////////////////////////////////////////////////////////
struct CVector
{
        double dx;
        double dy;
        double dz;
        double sigma_dx;
        double sigma_dy;
        double sigma_dz;
        double sigma0;
        int    nIteration;
};

//////////////////////////////////////////////////////////////////
// Class:               CGPSObservation
//
// Implementation Status : Completed (3,422 program lines)
//
//////////////////////////////////////////////////////////////////
class CGPSObservation
{
friend class CProject;
friend class CSpaceVehicle;

// Constructor(s) & destructor
public:
        CGPSObservation();
        virtual ~CGPSObservation();

// Attributes
        SVList*       SVs;
        SVList*       SyncSVs;
        CClockTime    m_ClockTime;

        // initially read from data file
        double                          ReceiverClockOffset;
        double       ReceiverClockOffsetAccuracy;

        // Number of SV's observed in this observation
        int          initHowManySVs;//this is initially read value from data file
        int          nvalidSVs;//number of synchronized svs
        int                                     ReferenceSV_PRN;
        int          nDD;// number of DoubleDiffs
        BOOL           ObservationValidity;//if false do not use this observation
        BOOL           Synchronized;
        SingleDiffList*  SingleDiffs;
        DoubleDiffList*  DoubleDiffs;
        CGPSObservation*    pObs2;//the sychronized observation
        DoubleDiffList  PrimaryDDs;
        DoubleDiffList  SecondaryDDs;
        DoubleDiffList  CombinedDDs;

        //filtered double differenced ambiguity paramters
        CList<double,double> PrimaryAmbListL1[5];
        CList<double,double> PrimaryAmbListLw[5];

// Operations
public:
        // Returns the list of observed space vehicles
        SVList* GetSVList()
                { return SVs; }
```

```
        CClockTime GetTime()
        {return m_ClockTime;}
        int HowManyValidSVs()
        {return nvalidSVs;}
        void DeleteAllSVs();
        void Initialize();
        void EmptySyncSVs();
        void DeleteAllSingleDiffs();
        void DeleteAllDoubleDiffs();
        void DeleteAllPrimaryDDs();
        void DeleteAllSecondaryDDs();
        void EmptyGlobalCandidateList();

        //code point positioning
        CPosition ComputeSingleEpochCodePosition(double X, double Y, double Z, double RCVdt);
        int ComputePosition(CMappingMatrix& B, CMappingMatrix& ResultMatrix,
                double& varx, double& vary, double& varz, double& vart);
        void SmoothSVs();//sv P1, P2 and phase values are used to compute ambiguities
        BOOL ComputeDDCodePosition(CStation *St1, CStation *St2);
        void ComputeSVPosition(CPosition *aPos);

        //double difference relative positioning. non synchronous svs and below elev are deleted
        void UpdateSynchronousSVs(); //observation to be compared to

        void FormDDWeight(double *DD_WeightMat, int nObs, double var);
        void FormDDWeight(CMappingMatrix &WeightMat, int sigtype);
        void FormDDUnitWeight(CMappingMatrix &UnitWeight);
        void FormDDWeight_Dual(double *DD_WeightMat, int nObs, double var);
        void FormDDWeight_Dual(CMappingMatrix &WeightMat, int sigtype);
        void FormDDUnitWeight_Dual(CMappingMatrix &UnitWeight);

        void FormDDObsEq(int sigtype, CMappingMatrix& DesignMat, CMappingMatrix& ConstantVec,
                                                CDoubleDiff* dd);
        void SetReferenceSV();
        CSpaceVehicle *GetSV(int PRN);//returns the SV from the SVlist;
        BOOL FindSV(int PRN);
        BOOL ComputeDDPhasePosition(CStation *St1, CStation *St2);
  void FormSingleDiffs();//observation to form the difference
        void FormDoubleDiffs();//observation to form the difference
  void UpdateSVPos(CPosition *aPos, DoubleDiffList *DDs);
        void ComputeBaselineDD(CStation *St1, DoubleDiffList *DDs,
                                                CPosition&  aPos,  CPosition
&newPos, BOOL IsInverse);
        void EstimateSVAmbiguity(CStation *St1, CStation *St2);
        void EstimateSVAmbiguity_w(CStation *St1, CStation *St2);
        double ComputeIonCorrection(double dResL1, double dResL2);
        double ComputeDDWideLane(CDoubleDiff *dd);
        double ComputeDDN1(CDoubleDiff *dd);
        void Find2Smallest(CList<double, double>& candList, double* sortArray);
        void FindPrimaryDD(CStation *St2);
        void FindPrimaryDD();
        double ComputeGDOP(CStation *St2, CDoubleDiff *testDD);
        double ComputeGDOP(CPosition &TestPos);
        void SelectAmbPar(CStation *St1, CStation *St2);
        void SelectAmbPar_w(CStation *St1, CStation *St2);
        void ComputeAmbTestPos(CStation *St1, CStation *St2, CMappingMatrix& DesignMat1,
                                                CMappingMatrix&
Weight1, CPosition &TestPos, BOOL &init);
  void CheckNewCandidate(CList<double, double> *ThisCandidate);
        BOOL ComputePrimaryL2Amb(CStation *St1, CPosition& TestPos, int &ion, int& wlane);
        void ComputeSecondaryAmb(CStation *St1, CPosition &TestPos);
        void ComputeTestPosResidual(CStation* St1, CPosition& TestPos, CMappingMatrix &Residual);
        void ResolveWideLaneAmb(CStation *St1, CStation *St2);
        void ComputeSecondaryAmb_w(CStation *St1, CPosition &TestPos);
        void ComputeAmbTestPos_w(CStation *St1, CStation *St2, CMappingMatrix& DesignMat1,
```

```
                                                                    CMappingMatrix&
Weight1,CPosition &TestPos, BOOL &init);
        void ComputeTestPosResidual_w(CStation* St1, CPosition& TestPos, CMappingMatrix &Residual);
        double ComputeAmbiguityFunctionValue(CStation *St1, CPosition &TestPos);
        double ComputeAmbiguityFunctionValue_w(CStation *St1, CPosition &TestPos);
        BOOL Initialized;//observation is initialized by calling ComputeSingleEpochCodePosition()
                //satellite positions will be computed as well as the clock offset
};

////////////////////////////////////////////////////////////////////
// Class:       CPosition
//
// Implementation Status : Completed (215 program lines)
//
////////////////////////////////////////////////////////////////////
class CPosition
{
friend class CProject;
friend class CSpaceVehicle;
friend class CGPSObservation;
friend class CStation;
friend class CDoubleDiff;

// Constructor(s) & destructor
public:
        CPosition();
        virtual ~CPosition();

// Attributes
private:
        double                          X,Y,Z,dx,dy,dz;
  //latitdue, longitude in degrees
        double      height, latitude, longitude, dh, dlat, dlong;
        //degree, minute and seconds
        int         latdeg, latmin, longdeg, longmin;
        double      latsec, longsec;
        double      Sigma0;//a priori standard deviation of parameters
        double      UnitVariance;//a posteriori unit variation of observations
        double      MaxResidual;
        double      pdop;
        int         nIteration;

// Operations
public:
        // Compute Cartesian coordinate X,Y,Z given geodetic coordinates latitude, longitude and height and
reference ellipsoid
        void Geodetic2Cartesian();
        void Cartesian2Geodetic();
        void Local2Cartesian(double phi, double lambda, double &x, double &y, double &z);
        void Deg2DMS();
        void Initialize();
};

////////////////////////////////////////////////////////////////////
// Class:       CProject
//
// Implementation Status : Completed (1,640 program lines)
//
////////////////////////////////////////////////////////////////////
class CProject
{
// Constructor(s) & destructor
public:
        CProject();
        virtual ~CProject();
```

```
// Attributes
private:
        char *indir ;
        char *outdir;
        int WaveLengthFactor_L1[TotalnSV+1];//WaveLengthFactor value of L1 phase for each satellite
        int WaveLengthFactor_L2[TotalnSV+1];
        OrbitList Orbits[TotalnSV+1];//List of orbits for each satellite read from navigation file
        int PRNinstances[TotalnSV+1];//number of occurrences of SV in observations
        SVList SameSV1,SameSV2;//static data where simultaneous tracked svs from two observations are
stored
                //pObs1 sv and pObs2 will alternate
        CStation *RefSt, *RovSt;
        FILE *RefFile, *RovFile;
        CGPSObservation *RefObs, *RovObs;
        CPosition FixedPos;

// Operations
public:
        //read observation files
        void ReadObsHeader(FILE *fpobs);
        void SynchronizeFiles();
        void ReadEpochObs(CString aLine, FILE* fp, CStation *pSt, CGPSObservation *pObs);

        //read navigation files
        void ReadNavHeader(FILE* file, CAlmanac &Almanac);
        void ReadNavData(FILE* file, CAlmanac &Almanac);

        COrbit GetOrbit(int PRN, CClockTime Time);

        //observation adjustment
        void AdjustBaseLine();
        void ProcessDD();
};

//////////////////////////////////////////////////////////////////////
struct CPhaseCenter
{
        // correction factor a
        double      CorrectionFactor;
        // HorizontalOffset(b)
        double      HorizontalOffset;
        // Phase center offset L1(c1)
        double      Offset1;
        // Phase center offset L2 (c2)
        double      Offset2;
};



//////////////////////////////////////////////////////////////////////
struct CAntenna
{
        CPhaseCenter   m_PhaseCenter;
        // Name of antenna
        CString     Number;
        CString     Type;
        double      delta_H;
        double      delta_E;
        double      delta_N;
        double      SlantHeight;
        double      Radius;
};
```

```
////////////////////////////////////////////////////////////////////
// Class:          CReceiver
//
// Implementation Status : Completed (62 program lines)
//
////////////////////////////////////////////////////////////////////
class CReceiver
{
friend class CProject;
friend class CStation;

// Constructor(s) & destructor
public:
        CReceiver();
        virtual ~CReceiver();
        EventList* GetEvents()
                {return ExternalEvents;}

// Attributes
private:
        CAntenna*    m_pAntenna;
        // The name of the receiver
        CString      ReceiverNumber;
        CString      ReceiverType;
        CString      SoftwareVersion;
        EventList*    ExternalEvents;

// Operations
public:
        // Get antenna specifics
        CAntenna* GetAntenna()
                { return m_pAntenna; }
        // Corrects the instrument eccentricities and updates the CorrectEcc
        void CorrectEcc();
};

////////////////////////////////////////////////////////////////////
// Class:          CSignalMeasurement
//
// Implementation Status : Completed (229 program lines)
//
////////////////////////////////////////////////////////////////////
class CSignalMeasurement
{
friend class CSpaceVehicle;
public:
        CSignalMeasurement();
        virtual ~CSignalMeasurement();

//Atributes
public:
        int         LLI_L1;//Loss of Lock Indicator
        int         SS_L1;//Signal Strength
        int         LLI_L2;//Loss of Lock Indicator
        int         SS_L2;//Signal Strength
        int         LLI_C1;//Loss of Lock Indicator
        int         SS_C1;//Signal Strength
        int         LLI_P2;//Loss of Lock Indicator
        int         SS_P2;//Signal Strength
        int         LLI_P1;//Loss of Lock Indicator
        int         SS_P1;//Signal Strength
        int         LLI_D1;//Loss of Lock Indicator
        int         SS_D1;//Signal Strength
```

```
        int         LLI_D2;//Loss of Lock Indicator
        int         SS_D2;//Signal Strength
        int         LLI_T1;//Loss of Lock Indicator
        int         SS_T1;//Signal Strength
        int         LLI_T2;//Loss of Lock Indicator
        int         SS_T2;//Signal Strength

private:
        double      L1_Value; //cycles
        double      L2_Value;
        double      C1_Value; //meters
        double      P1_Value;
        double      P2_Value;
        double      D1_Value;//Hz
        double      D2_Value;
        double      T1_Value;//transit integrated doppler
        double      T2_Value;//transit integrated doppler
        double      SmoothedCode_Value;
        double      Lw_Value;
        double      CombinedCodeRange;
        double      nDuration;//to  comute weight factor in code smoothing

// Operations
public:
        double GetSignalValue(int sigtype);
        double GetTravelTime(int sigtype);
        void  SetSignalValue(int sigtype, double value);
        void  SmoothCode(CSignalMeasurement *prevSignal, double Interval);
        void  ComputeCombinedCodeRange();
        void  InitializeCodeSmoothing();
        BOOL  L1_IsCycleSlip;
        BOOL  L2_IsCycleSlip;
        BOOL  IsInitial;
};

//////////////////////////////////////////////////////////////////////
struct CIonParameter //almanac parameters
{
        double A0, A1, A2, A3;
        double B0, B1, B2, B3;
};

//////////////////////////////////////////////////////////////////////
struct CUTCParameter //almanac parameters
{
        double A0, A1; //terms of polynomial
        int T;      //reference time for UTC data
        int W;       //UTC reference week number
};

//////////////////////////////////////////////////////////////////////
struct CAlmanac
{
        int         LeapSeconds;
        CIonParameter   m_IonPar;// Ionosphere parameters of almanac
        CUTCParameter   m_PUTCPar;// almanac parameters to compute time in UTC
};

//////////////////////////////////////////////////////////////////////
struct COrbit
{
        // SV PRN number
        int         PRN;
        // Transmission of time of message(sec of GPS week)
        double      TransmissionTime;
```

```
        // Time of Clock in year, month, day, hour, minute, second
        CClockTime    Toc;
        // Coefficient to compute the second order polynomial describing the satellite clock offset
        double        ClockBias;
        // Coefficient to compute the second order polynomial describing the satellite clock offset
        double        ClockDrift;
        // Coefficient to compute the second order polynomial describing the satellite clock offset
        double        ClockDriftRate;
        // Issue of Data, Ephemeris
        double        IODE;
        double        Crs;
        double        Delta_n;
        double        MO;
        double        Cuc;
        // Eccentricity
        double        e;
        double        Cus;
        double        sqrt_A;
        // Time of Ephmeris
        double        Toe;
        double        Cic;
        double        OMEGA_0;
        double        Cis;
        double        i_0;
        double        Crc;
        double        omega;
        double        OMEGA_DOT;
        double        I_DOT;
        // Codes on L2 channel
        double        Code_L2;
        // To go with TOE
        double        GPS_Week_No;
        // L2 P data flag
        double        L2_P_Flag;
        double        Accuracy;
        double        Health;
        double        TGD;
        // Issue of Data, Clock
        double        IODC;
        CAlmanac      Almanac;
};

////////////////////////////////////////////////////////////////////
// Class:              CSpaceVehicle
//
// Implementation Status : Completed (739 program lines)
//
////////////////////////////////////////////////////////////////////
class CSpaceVehicle
{
friend class CProject;
friend class CGPSObservation;

// Constructor(s) & destructor
public:
        CSpaceVehicle();
        virtual ~CSpaceVehicle();

// Attributes
private:
        CSignalMeasurement*     m_pSignalMeasurement;
        CPosition     m_SVPosition;
        COrbit        m_Orbit;
        CClockTime    m_ObsTime;
```

```
public:
        // Elevation of space vehicle at a given time
        double      Elevation;//in degrees
        double      Azimuth;//in degrees
        double      IonCorrection;
        double      TropoCorrection;
        double      RelativisticCorrection;
        double      SVTime;
        double      SVTimeError;
        double      Rho; //distance between station and SV
        double      SigmaRho;
        double      TravelTime;//travel time of preferred code
        BOOL        PositionKnown;
        double      dphase;
        double      dtime;
        double      phase_rate;
        double      SlipCorrection;
        BOOL        FirstEpoch;
        BOOL        AmbiguityComputed;
        BOOL        LockLoss;
        // Space Vehicle PRNnumber
        int         PRN;
        double      Corrected_Code;
        int         WaveLengthFactor_L1;// (FULL,HALF,SINGLE)
        int         WaveLengthFactor_L2;// (FULL,HALF,SINGLE)
        double      N1; //L1 Ambiguity parameter
        double      N2; //L2 Ambiguity parameter
        double      Nw;//WideLane Ambiguity parameter
        double      Smoothed_Rho;
        double      Smoothed_IonCorrection;
        double      Smoothed_N1;
        double      Smoothed_N2;
        double      Smoothed_Nw;
        double      Smoothed_dN1;
        double      Smoothed_dN2;

// Operations
public:
        void ComputeIonCorrection(CPosition* aPos);
        void ComputeTropoCorrection(double station_height);//in meter
        void ComputeTropoCorrection();//in meter
        void ComputeRelativisticCorrection();//in meter
        void ComputeSVTime(int PreferCode, double RCVTimeError);
        void ComputeSVTime(double RCVTimeError);//uses travel time
        void UpdatePosEarthRotation(double time); //m_SVPosition is updated for earth rotation
        void ComputeAzElevRho(CPosition* aPos);
        CSignalMeasurement* GetSignalMeasurement()
                {return m_pSignalMeasurement;}
        CPosition GetSVPosition();
        // Computes and updates the data member m_pPosition of the Space Vehicle from orbit and time given
in gps time.
        void ComputeCurrentPosition(double SVGPSTime);
        //code value used to compute SVtime, sv position initially computed, travel time computed,
        //SVtime recomputed, sv position recomputed and then updated for earth rotation;
        void ComputeCurrentPosition(int PreferCode, double RCVTimeError,CPosition* StationPos);
        void ComputeTravelTime(CPosition* StationPos);
        void ComputeTravelTime();
        void ComputeN1Ambiguity(CGPSObservation *pObs);
        void ComputeWideLaneAmbiguity();
        void SmoothDualCode();

private:
        void InitializeSignals();
};
```

```
//////////////////////////////////////////////////////////////////////
// Class:              CStation
//
// Implementation Status : Completed (154 program lines)
//
//////////////////////////////////////////////////////////////////////
class CStation
{
        friend class CProject;
        friend class CGPSObservation;
        friend class CDoubleDiff;

public:
        CStation();
        virtual ~CStation();

        void StPos2AntennaPos(CPosition* aPos);
        void AntennaPos2StPos(CPosition* aPos);
        void Initialize();

private:
        CPosition* knownStPos;//if a position is fixed
        CPosition* initialAntPos;//if a position is fixed
        CPosition* approxPos; //given position read from file
        CPosition* CodePos;//position computed with code
        CPosition* TDPos;//position computed with TD
        CPosition* FloatDDPos;//position computed with DD float
        CPosition* FixedDDPos;//position computed with DD fixed

        CString Name;
        CString Number;
        BOOL initialPosKnown;
        BOOL FixedType;
        BOOL CodeAdjusted;
        double PDOP;
        static int      count;

        CReceiver*                  m_pReceiver;
        CString                     ObsFileName;
        CClockTime      m_FirstObsTime;
        CClockTime      m_LastObsTime;
        double                      Interval;
        EventList*      Events;
        PositionList*   Positions;
        int noObsType;
        int ObsType[10];
        int PRNinstances[TotalnSV+1];//number of occurrences of SV in observations
};
```