The GPSTk: GLONASS, RINEX Version 3.00 and More

Thomas L. Gaussiran, Eric Hagen, R. Benjamin Harris, Chris Kieschnick, Jon C. Little, Richard G. Mach, David C. Munton, Scot L. Nelsen, Colin P. Petersen, David L. Rainwater, Brent A. Renfro, Brian W. Tolman
Applied Research Laboratories, The University of Texas at Austin

Dagoberto Salazar
Grupo de Astronomia y Geomatica, Universitat Politècnica de Catalunya

BIOGRAPHY

Thomas Gaussiran is the director of the Space and Geophysics Laboratory (SGL) at Applied Research Laboratories, The University of Texas at Austin (ARL:UT). He received his B.S. in Physics from UT Austin (1988) and M.S. (1991) and Ph.D. (1994) from Rice University.

Eric Hagen is a graduate student in the Department of Aerospace Engineering and Engineering Mechanics at UT Austin.

R. Benjamin Harris is an Engineering Scientist at ARL:UT. He received a B.S. (1994) and Ph.D. (2008) in Aerospace Engineering from UT Austin and his M.S. in Aeronautics and Astronautics from Stanford (2000).

Chris Kieschnick is an Engineering Scientist Associate at ARL:UT. Chris received his B.S. (2009) in Computer Science from UT Austin.

Jon Little is a Senior Engineering Scientist at ARL:UT. He obtained a B.S. (1988) and a M.S. (1990) from Auburn.

Richard Mach is a project lead at ARL:UT. He has been involved with GPS applications since 1990. He holds a B.S. (1990) and M.S. (1992) in Aerospace Engineering from UT Austin.

David Munton is a Research Associate at ARL:UT. He earned a B.S. in Physics from Sonoma State University (1982), and a Ph.D. in Physics from UT Austin (1992).

Scot Nelsen is an Engineering Scientist at ARL:UT. He earned a B.S. in Electrical Engineering at UT Austin (1998).

Colin Petersen is an Engineering Scientist Associate at ARL:UT. He received a B.S. (2004) and M.S. (2006) in Electrical and Computer Engineering from UT Austin.

David Rainwater is a Research Associate at ARL:UT. He received a B.S. in Physics from the University of Missouri (1993) and a Ph.D. in Physics from the University of Wisconsin Madison (1999).

Brent Renfro is a program manager at ARL:UT. He has a B.A. in Physics from Wabash College (1979) and a M.A. in Computer Science from UT Austin (1983).

Brian Tolman is a Research Scientist at ARL:UT. He holds a Ph.D. in theoretical physics from UT Austin (1982).

Dagoberto Salazar is an Aeronautical Engineer (IUP-FAN, Venezuela, 1992). After working several years both for government and private industry, he pursued postgraduate studies in Instrumentation and Control (UCV). He is currently a Ph.D. candidate in Aerospace Science and Technology at the Universitat Politècnica de Catalunya (UPC, Spain).

ABSTRACT

The GPS Toolkit (GPSTk) project is an attempt to bring the power of the open source world to the satellite navigation community. The GPSTk is intended to help eliminate the “black box” nature of many commercial applications, and to enable rapid prototype development for data analysis applications. It is also intended to support software development and systems engineering associated with GNSS data collection systems.

Because satellite navigation is ubiquitous, its users employ practically every computational architecture and operating system. Therefore the GPSTk suite is intended to be as platform-independent as possible. This is achieved through use of ISO-standard C++. The principles of object-oriented programming are used throughout the GPSTk code base in order to ensure that it is modular, extensible and maintainable.

The software suite consists of a core library, auxiliary libraries, and a large set of advanced applications. The libraries provide the wide array of common functions that applications use to handle data processing associated with GPS. Furthermore, programmers can also access the library code to develop new processing applications.

The GPSTk was initially designed and developed in a highly collaborative environment of software engineers and scientists in the Space and Geophysics laboratory (SGL) at the Applied Research Laboratories, The University of

Texas at Austin (ARL:UT). SGL decided in 2003 to open-source much of their basic GPS processing software as the GPSTk under the GNU Lesser General Public License (LGPL) version 2.1. The GPSTk's source code and documentation are now hosted on public servers so that project members from multiple academic and commercial institutions can freely collaborate in development.

In the last two years, a number of new applications and library capabilities have been added to the GPSTk. New applications have been added to characterize clock stability. New library capabilities have been added to process GLONASS observations and to fully support the Receiver INdependent EXchange (RINEX) version 3 file format. New library code also provides the ability to generate highly customizable plots compatible with LaTeX and web browsers. A new library has been added that provides a framework for precise point positioning (PPP).

INTRODUCTION

The goal of the GPS Toolkit (GPSTk) project is to provide an open source library and suite of applications to the satellite navigation community—to free researchers to focus on research, not lower level coding. In this paper, we explain the organization of the GPSTk project and its software suite. Because the GPSTk is a collaboration, it grows over time with new capabilities. Capabilities developed over the last two years, some of which are still under test, are described.

For exchange of observation data, the GPS community has relied on the **R**eceiver **IN**dependent **EX**change format [1, 2, 3]. Since the GPSTk has been released it has specifically support RINEX version 2 (R2). To support multi-GNSS receivers and data analysis, RINEX has evolved from RINEX version 2 to version 3.00, which includes coherent schemes for multi-GNSS data, as well as greatly enhanced data records specifically designed for kinematic applications [4]. Some of the RINEX formats had to be radically restructured to do so. The GPSTk has developed support for reading and writing R3 files and mechanisms for storing the additional data defined by new standard. Existing applications have been upgraded to use R3.

A key, underlying challenge to truly support R3 is the ability to integrate observations from multiple Global Navigation Satellite Systems (GNSSs). Each GNSS defines its own coordinate and time systems. Reconciling disparate systems could be accomplished on a case-by-case basis. In the GPSTk, this translates to modifying processes at the application level. A more seamless solution was sought for the GPSTk, one that could exist at the library level. The R3 code transparently provides the translation of coordinate and time systems during ephemeris evaluation. However this convenience has implications. The design of the R3 implementation and its implications are a subject of this paper. Also discussed is how the R3 implementation enables the integrated processing of GPS and GLONASS observations.

A topic common to all GNSS processing is clock stability. When satellite observations are used to support time transfer or orbit determination, clock stability is key factor. The new GPSTK application *Clock Tools* provides the GPS and precise time communities free access to basic clock stability analyses. Current *Clock Tools* capabilities include the computation of Allan and Hadamard stability metrics, along with data parsing, grooming, and plotting. We present several typical applications.

Fundamental contributions have been made to the GPSTk library as well. A new auxiliary library has been added that supports precise point positioning (PPP). Another library has been added to provide customizable plotting in LaTeX and HTML.

PRIOR WORK

Many aspects of GNSS technology are open. The signals themselves are defined in public technical documents. File formats associated with observations and ephemerides are open as well. The case is different for GNSS processing software. Commercial software packages, often written by receiver manufacturers, are proprietary by nature. Others packages offered freely must be licensed to a site or an individual. Often MATLAB ® scripts that are publicly shared are accompanied by a license restriction prohibiting their redistribution or modification.

In contrast, the GPSTk project source code is free to use, modify, and redistribute, per the terms of it license, the GNU Lesser General Public License (LGPL) version 2.1. The GPSTk is not alone in providing an open source suite to the GNSS research and development community. Several other projects offer software under the terms of the LGPL, the GPL or other open source licenses.

The focus of the OpenSourceGPS project is to create open source receivers [5]. The first hardware targeted by the project were PC cards controlled using the Mitel/Plessy/Zarlink chip set. Because the chip set is no longer manufactured, the company GPS Creations has partnered with the project to new cards. In recent years, OpenSourceGPS and GPS Creations have focused on developing a narrow-band software receiver [6, 7].

In contrast to targeting receiver internals, the bulk of open source projects target the external interfaces of commercial units. Two notable projects are GPSBabel and gpsd [8, 9]. GPSBabel translates way-points so they can be moved among receiver manufacturers. The gpsd provides a network service that allows a personal computer to communicate real-time, differential corrections over a network.

PROJECT OVERVIEW

The GPSTk suite is composed of a set of libraries and more advanced applications built upon them. The libraries provides a wide array of functions that solve common GNSS processing tasks such as the navigation solution or reading a RINEX file. The applications can be used by non-programmers to solve specific, well known tasks.

Design

The design goals of the GPSTk library are portability, modularity, clarity, extensibility, and maintainability. These goals allow the GPSTk to maximize the audience and lifetime of the library while decreasing the costs associated with long-term maintenance. Another factor in the design is that GNSS users employ practically every computational architecture and operating system. Therefore, the design of the GPSTk suite is as platform-independent as possible. For these reasons, the GPSTk source is build using C++, and strictly adheres to its ISO standard [10]. The language by nature supports Object Oriented Analysis and Design (OOA/D), a technique well-known to support the project's design goals. OOA/D are used throughout the design of the libraries and most of the applications. The ubiquity of C++ allows the GPSTk to support all major desktop and server platforms, most notably Linux, Windows, Solaris and Mac OS X. Windows users have two versions to choose from, a native version, built using the Microsoft compiler, and a version that executes in the Cygwin environment.

OOA/D contrasts with procedural design supported by the C and FORTRAN languages. In procedural programming, a function library is provided to the user. In Object Oriented (OO) programming, a class library is provided. Each class is an independent module that can be invoked by the user as an object or extended by the user in the form of a new class. Classes can build upon each other through a number of object-oriented principles, such as inheritance, the use of metaclasses, polymorphism, and aggregation. The GPSTk library relies heavily on the Standard Template Library [11], which is part of the ISO standard for C++. The STL provides OO data structures (containers). These include linked lists, vectors, and maps. The STL also provides standard algorithms for these containers, e.g., the quicksort algorithm.

All of the GPSTk code includes documentation designed for extraction by the *doxygen* [12], a freely available application that generates a HTML-based documentation from the code itself. Like the GPSTk, *doxygen* is platform-independent. The *doxygen* documents are available on the GPSTk web site or can be easily generated from the code in the distribution.

Getting the GPSTk

The GPSTk can be downloaded over the web via links provided on the project website, http://www.gpstk.org/ . Precompiled binaries are available for many platforms. Access to the source code base—current as well as well as a history of all changes–is provided through a publicly accessible repository hosted by SourceForge. The repository can be accessed via a web browser, or using a client from the Subversion project [13, 14]. The Subversion project provides two kinds of clients, one with a graphical user interface and one based on the command line. Because the command line version works identically on all of the development platforms supported by the GPSTk, use of the command line tool *svn* is documented in the GPSTk website and this paper.

The following command can be used to retrieve the latest GPSTk source from SourceForge and write it into a directory structure on the user machine in the current working directory:

```
svn co https://gpstk.svn.sourceforge.net/svnroot/gpstk
```

Note that it is not necessary to provide any user information or password to retrieve the code. This form of access is referred to as anonymous in the *Subversion* documentation. Once written, any directory in this structure can be updated to the latest code by first making it the current working directory then executing the following command:

```
svn update
```

For further detail describing how to build the GPS Toolkit, please refer to the on-line documentation.

Project Documentation

To facilitate true cross organizational development and user interaction, ARL:UT established a dynamic website, also known as a wiki. The wiki site utilizes an open source product called TWiki [15] that is not only a wiki but a full featured and easily extensible development platform. The project has customized the base TWiki installation to add two capabilities. One capability allows for users to submit questions or answers about the project. Another provides a framework for capturing development documentation, such as brainstorming or designs. The site also supports the LaTeX expression for content such as equations. Finally, users can choose to be notified when topics are changed via email or web feed.

The website hosts a copy of the documentation generated by *doxygen*. This documentation is updated from the code repository on a daily basis to ensure easy access to the latest documentation changes. Additionally, an IRC channel was established to create a real-time avenue of developer communication. Finally, the new website contains new documentation that describe the GPSTk development process, the release process, how to get started with the GPSTk, and more. The goal of this added information was to help new

developers become familiar with the project operation so they can become effective contributors.

The user manual is also hosted on the TWiki. Two manuals exist at this time. An older manual, written in LaTeX, is posted as a PDF. A newer manual is under development purely within the TWiki. Development on the older manual has stopped. We encourage that all new documentation be added to the new TWiki-based manual. This encourages the users to keep the manual accurate.

Branching

In the last two year, the GPSTk has adopted *branching* to support development and release. A branch is in essence a duplication of the source. The duplicated code, referred to simply as the branch, can then be modified separately from the original, sometimes referred to as the *trunk*. The idea is to support simultaneous efforts using multiple viable code bases. However, in the end, the branches and the trunk must be reconciled. This practice is known as *integration*.

Branching has been used to support two kinds of activities in the GPSTk. First, branching was used to enable stable releases. For such a large project, with a large number of developers, it is sometimes impossible to ask developers to stop contributing new code so that a stable version can be released. Branching was used to generate the stable releases associated with version 1.5 and 1.6. Second, branching can be used to separate large scale modifications. The development of RINEX-3, described later in detail, utilizes a new branch.

LIBRARY ENHANCEMENTS FOR RINEX3

Contributions are the force of change in an open source project. Over the last two years, thousands of modifications have been submitted to the GPSTk subversion repository. Few of those modifications have changed the interface to existing classes. Most of the changes are new classes with new capabilities. RINEX-3 support is largely complete, and is available through library functions in a new branch. It is the subject of this section. The remainder of the library modifications occurred in the trunk. These new capabilities are described in the next section.

RINEX VERSION 3.00 LIBRARY ENHANCEMENTS

The substantial changes between RINEX Version 3.00 (herein note R3) and RINEX Version 2 (R2) resulted in the consideration of several upgrade paths for implementation. The major design considerations for discriminating advantages and disadvantages between choices were as follows.

- Full and straightforward backwards compatibility.
- Ease of testing.
- Minimal modifications required across the remainder of the toolkit to integrate the new features.

Ultimately the choice was made to create an independent set of R3 classes with distinct names, simultaneously branching the GPSTk within its Subversion repository. This choice allows backwards compatibility with R2, and allows for separate known versions for comparison testing of R3 versus R2. Figure 1 depicts the class structure introduced to support R3 capabilities.

Time System

The GPSTk originally assumed that all times are given in the GPS time system. Since this is not true in a multi-GNSS world, a Time System (TS) data structure was added to keep track of what time frame epochs are actually reported in. The R3 standard specifies that epoch data for a given GNSS be given in that GNSS's native time system. As a result, the GPSTk has been designed to automatically encode the TS data structure along with the observations.

Coordinate System

In the same vein, each GNSS uses its own realization of the International Terrestrial Reference Frame (ITRF) as its coordinate system. For GPS this is WGS84(G1150), while for GLONASS it is PZ-90. Galileo will use the Galileo Terrestrial Reference System (GTRF). Japan's QZSS will use JGS, which differs from the GTRF by less than 2 cm. The disagreement between different systems varies in general, but the agreed-upon realization goal is 2-3 cm in the near future.

We have added support for the standard reference frames to the GPSTk, as well as the ability to transform data between reference frames via a Helmert transformation [16], defined by

$$\mathbf{X}_A = \mathbf{a} + \mu \mathbf{R} \mathbf{X}_B.$$

in which $\mathbf{X}$ denotes a position vector as a column matrix, A and B denote reference frames, μ is a scale factor and $\mathbf{a}$ is an offset or bias.

Data Storage and Access

Observation, navigation, and meteorological data have to be stored for access (there are only those three data types in R3). This data storage should be efficient. The current implementation of R2 handling uses a "map of maps" (MoM) approach [17]. In this approach, the actual data is stored in instances of a data element class, which become the values in a map keyed by epoch (time). Instances of that map are then stored as values in another map, this one keyed by satellite vehicle ID. The advantage of this approach is that this creates a poor-man's "database" that is easily accessible by the most identifiable tag, time. Other possible

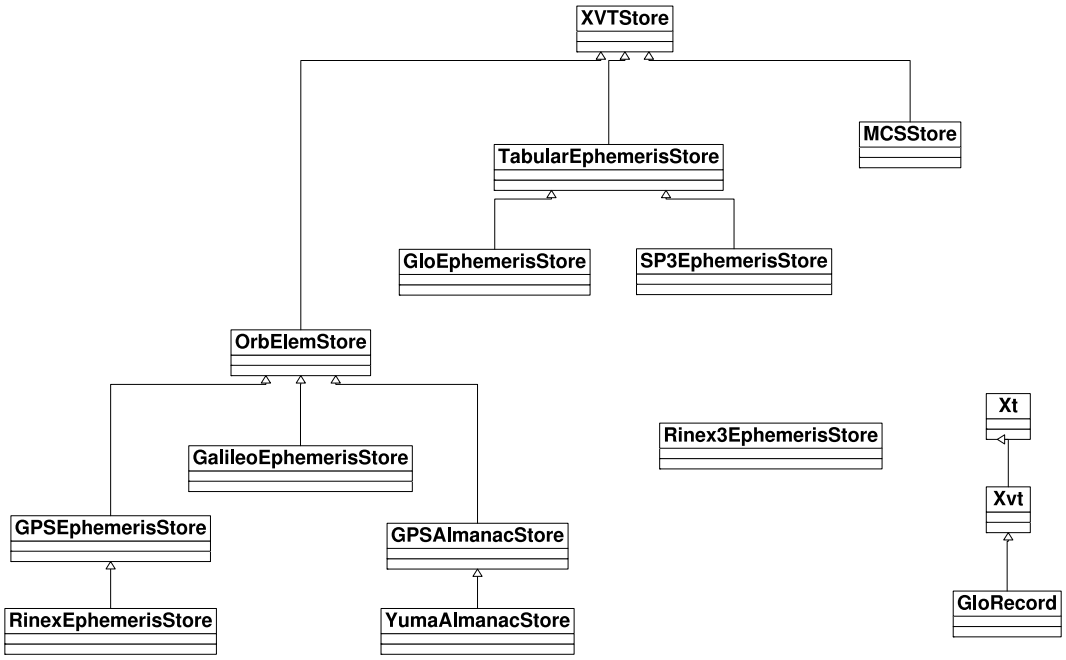

Fig. 1 Unified Modeling Language (UML) diagram showing inheritance relationships among RINEX Version 3.00 classes.

approaches include filling matrices, valarrays, linked lists or possibly other data types.

For R3, we choose to stay with the MoM design based on an assessment that the current implementation of data storage in R2 is acceptable in usability and speed, as well as being quite flexible. However, there are some revisions. For example, each GNSS has its own MoM, belonging to its own storage class. A new class specific to reading R3 files contains an instance of each possible GNSS MoM, and inserts new data into the appropriate MoM as it is read in. Look-ups can then be targeted to a specific GNSS, or scanned over all MoMs, depending on the user's requirements.

Navigation Data

R3 expanded the navigation broadcast message ("Nav") data type to include general GNSS Nav messages. These formats can be quite different. For example, Galileo's broadcast ephemerides format is similar to GPS [16]. However, GLONASS, the Space-based augmentation system (SBAS), and QZSS, use tabular ephemerides [18]. A major restructuring of the GPSTk ephemeris processing classes was required in order to be able to handle the R3 standard, though it currently includes only GLONASS.

The data storage class was more cleanly split into two branches. One is the parent class for all tabular-type GNSSs (GLONASS, QZSS), while the other is for those which broadcast orbital element data (GPS, Galileo). It should be noted that precise ephemerides in SP3 files are presented in tabular format. Thus, the SP3 classes remain a subclass of `TabularEphemerisStore`. We remark here only that separate naming for the R3 classes allowed us to keep the R2 classes in place, and with invisible backwards compatibility. That is, R2 users will not have to change their existing applications to use the new GPSTk library.

Observation Data

Observation ("Obs") data refers to any measurement of (pseudo)range, phase, Doppler or other signal quantity made with a GNSS receiver. R3 Obs data files have changed significantly, providing support for additional data, e.g. moving receivers or antenna details, but there were subtle, significant structural changes. Chief among these is Obs type code list and format. Obs files may now include multiple GNSS data in a file, which requires more detailed header information on what data will is present for each GNSS. There are new data structures for satellite ID. Epochs are presented differently.

The new R3 Obs classes in the GPSTk had to be significantly modified to handle these changes. Some of the data is stored in maps (and even maps of maps) to simplify access over different GNSSs. The extent of changes in the R3 standard will require Obs data users to significantly change the way they write their applications. It has always been up to the user to store Obs data once read in; we continue to not provide any inherent Obs data storage classes (e.g. MoMs) in this upgrade.

ADDITIONAL LIBRARY ENHANCEMENTS

In contrast to the RINEX version 3.00 development, smaller scale enhancements have been conducted in the trunk. The *procframe* library, which provide precise point positioning capabilities, has been upgraded. Also, a new library has been developed to support vector-based graphical output such as plots.

Processing Framework

The GPSTk Processing Framework is based on the "GNSS Data Structures" (GDS) and the associated "GDS Processing Paradigm" first presented at [19]. Its purpose is to solve data management problems that are difficult to solve with simpler structures like vectors and matrices.

Moreover, the "GDS Processing Paradigm" provides an unified and consistent way to handle such structures, providing *processing classes* that handle both data and corresponding *metadata*. The objects from these *processing classes* reach into the GDS and add, delete and/or modify what is needed (according to their function), and leave the results in the same GDS, appropriately indexed. These processing objects are designed to use sensible defaults in their parameters, but may be tuned to suit specific needs.

The former ideas are coupled with a handy redefinition of C++ operator `>>`, implemented in such a way that several operators may be concatenated. This allows a programming style that clearly shows how the data is *flowing* along the processing steps, in a way that is similar to Unix "pipes". A simple example of this approach is presented. Here, just a single epoch worth of data will be extracted out of a RINEX observation file, and put into a GDS:

```
1   RinexObsStream rinexFile("ebre0300.02o");
2   gnssRinex gpsData;

3   rinexFile >> gpsData;
```

Line #1 declares an object named *rinexFile* of class `RinexObsStream`, which is used to handle RINEX observation files. That object will take care of handling "ebre0300.02o" RINEX observation file. Line #2 declares an object of class `gnssRinex`, which is a very common and handy GDS (several structures are available). Data will be stored in this object, called *gpsData*.

Then, line #3 does the real work: It will take one epoch of data out of *rinexFile* and will pour it into *gpsData*. No further coding is needed for this action, and line # 3 is referred to as the "processing line".

Taking this approach to GNSS data management and processing as a starting point, the capabilities of the Processing Framework have been greatly expanded during last year. Several sophisticated processing classes have been added, and a "Precise Point Positioning" (PPP) implementation [20] has been fully developed.

PPP is a complex task, and issues like wind-up effects, solid, oceanic and polar tides, and antenna phase centers variations, must be taken into account. Also, the data rates of typical IGS products being used often do not match with data rates from available observations, and therefore some time management issues also arise. In order to deal with these tasks, the GPSTk now provides some accessory classes like the following:

`ConfDataReader`: Powerful class to parse and manage configuration files. It supports multiple sections, variable descriptions and value descriptions (such as units), and a wide range of variable types.

`AntexReader` and `Antenna`: These classes allow one to properly use antenna phase center variations, both in "relative" as well as in "absolute" modes, reading them from IGS ANTEX format files. These classes are complemented with processing classes that will take care of applying the corresponding corrections: `CorrectObservables` to manage receiver antenna corrections, and `ComputeSatPCenter` to handle satellite antenna corrections.

`SolidTides`, `OceanLoading` and `PoleTides`: Compute tidal effects due to solid tides, ocean loading and pole displacements, respectively.

For decimation, the `Decimate` class is used. Given the usual data rate mismatch between IGS precise products (900 s per sample) and RINEX observations (30 s per sample) this is a recommended procedure. `Decimate` class takes advantage of GPSTk's sophisticated exception handling mechanism, and if data epoch is not a multiple of 900 seconds (or other preset time interval) then the `Decimate` object will issue an "exception" (effectively halting further processing of data for that epoch) and the program will continue processing the next epoch. Decimation is thence achieved in a effective and compact way.

A typical PPP processing line looks like follows:

```
gpsData >> requireObs >> linear1 >> markCSLI >> markCSMW
        >> markArc >> decimateData >> basicModel
        >> eclipsedSV >> grDelay >> svPcenter >> corr
        >> wUp >> computeTropo >> linear2 >> pcFilter
        >> phaseAlign >> linear3 >> baseChange >> cDOP
        >> pppSolver;
```

This GDS processing data chain is a single C++ line, although for the sake of clarity it spans several physical lines. This line must be enclosed within a *while* loop to process all available epochs, and also within a *try-catch* block to manage exceptions. Further details are provide by examples in the source distribution as "example8.cpp", "example9.cpp" and "example10.cpp". Detailed information is also availble in the Doxygen documentation provided on the website. The Doxygen documentation can also be generated from the source code as described in prior sections.

The following is a brief explanation about what some of these objects do, and what classes they belong to:

- *requireObs* (an object of class `RequireObservables`): Checks if required *TypeID*s (usually observations) are present.

- *linear1*, *linear2* and *linear3* (`ComputeLinear`): They compute linear combinations such as ionospheric, Melbourne-Wubbena, ionosphere-free, etc.
- *markCSLI* (`LICSDetector2`) and *markCSMW* (`MWCSDetector`): They detect and mark cycle slips using the ionospheric and Melbourne-Wubbena combinations, respectively. Besides, *markArc* (`SatArcMarker`) keeps track of satellite arcs.
- *decimateData* (`Decimate`): Decimates data if epoch is not a multiple of 900 s.
- *basicModel* (`BasicModel`): Computes the basic components of a GNSS signal propagation model.
- *eclipsedSV* (`EclipsedSatFilter`): Removes from the GDS the satellites in eclipse.
- *grDelay* (`GravitationalDelay`): Computes gravitational delay effect due to changing gravity field along satellite-receiver ray.
- *svPcenter* (`ComputeSatPCenter`): Computes the effect of satellite antenna phase center.
- *corr* (`CorrectObservables`): Corrects observables from tides, receiver antenna phase center, eccentricity, etc.
- *wUp* (`ComputeWindUp`): Computes phase wind-up.
- *computeTropo* (`ComputeTropModel`): Models delays due to tropospheric effects.
- *pcFilter* (`SimpleFilter`): Filters out spurious values in the PC (ionosphere-free) combination.
- *phaseAlign* (`PhaseCodeAlignment`): Aligns phase with code values, preserving ambiguities integer nature.
- *baseChange* (`XYZ2NEU`): Prepares GDS to use a North-East-UP reference frame in object *pppSolver*.
- *cDOP* (`ComputeDOP`): Computes DOP values.
- *pppSolver* (`SolverPPP`): Solves the equation system using an Extended Kalman filter (EKF).

Particular mention deserves object *pppSolver*. This object is preconfigured to solve the PPP equation system in a way consistent with [20]: Coordinates are treated as constants (static), receiver clock is considered white noise, and residual vertical wet tropospheric effect is processed with a random walk stochastic model.

`SolverPPP` objects are very easy to configure. For instance, if the solution is needed in a North-East-Up reference frame (instead of the default ECEF), we just need to insert a `XYZ2NEU` in the processing chain (like *baseChange* above) and configure the solver object:

```
pppSolver.setNEU( true );
```

In order to carry out kinematic instead of static positioning, the coordinates stochastic model being used is changed. For example, we consider coordinates as white noise with sigma equal to 100.0 meters:

```
WhiteNoiseModel newCoordinatesModel( 100.0 );
pppSolver.setCoordinatesModel( &newCoordinatesModel );
```

Moreover, it is very easy to set different stochastic models for each dimension. For instance, a surface vehicle could be modeled with restrictions in the vertical coordinate, like this:

```
WhiteNoiseModel horizontalModel( 100.0 );
RandomWalkModel verticalModel( 0.01 );

pppSolver.setNEU( true );
pppSolver.setXCoordinatesModel( &horizontalModel );
pppSolver.setYCoordinatesModel( &horizontalModel );
pppSolver.setZCoordinatesModel( &verticalModel );
```

In summary, the Processing Framework approach has demonstrated being very flexible and powerful, and results obtained with the current PPP implementation are comparable with other GNSS data processing suites ([21], [22]). The current development efforts within this framework aim to provide a uniform data structure powered by classes able to carry out network-based data processing, programmable on-the-fly generic solvers, and ambiguity fixing capabilities.

Vector Plotting

Unlike MATLAB ®, C++ does not provide graphics. However graphic representations such as scatter plots are vital to interpreting the results of many GNSS algorithms. Typically, GPSTk applications generate text results, and those results are imported to another package for visualization. In Microsoft Windows, Excel ™ is a popular choice to interpret the results of GPSTk. For that same purpose, many Linux users employ *gnuplot*. A few applications within the GPSTk provide visualization. In each case, another programming language other than C++ is employed. Python is used by the application *ordPlot*, and perl/Tk is used by *RinexPlot*.

A method was sought to integrate graphics support more directly into GPSTk applications. One choice we considered was to adopt an external library compatible with C++. Many such libraries exist for C and C++, however during our search, none were appropriate for use or redistribution with the GPSTk. Each either was incompatible with the LGPL, or could not support Windows. Adding the desired feature for the ability to generate novel visualizations, we decided to experiment with making our own new library.

What resulted was not one but three libraries, two of which have been implemented. These three libraries are depicted in Figure 2 in terms of dependencies. The focus of each library is described in Table 1. The dependencies mean that libraries build upon one another, just as GPSTk

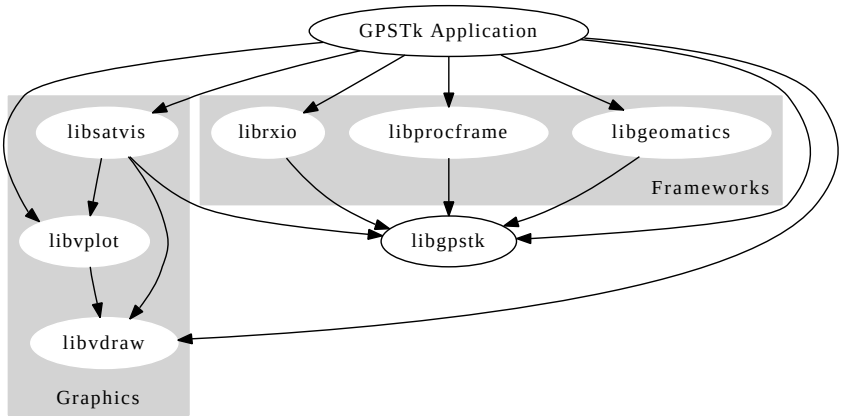

Fig. 2 Possible calling dependencies amoı

Table 1 Capabilities of the graphics libraries.

libvdraw	Basic shapes to postscript and SVG. Spacing and sizing logic.
libvplot	Generic scatter, line and surface plots.
libsatvis	Skyplots, satellite visibility timelines and calendars.

application build upon the libraries. For example, *libvplot* uses the basic shapes provided by *libvdraw*. However, each is fully accessible by a GPSTk app. The highest level library, *libsatvis*, as of this writing has not yet been implemented.

The most basic library is *libvdraw*. The "v" stands for vector, as the goal of the library is to provide an abstract mechanism for drawing to vector formats. In concept, that can include graphical user interfaces such as those provided by Windows and X-Windows. However, as of today, three file-based formats are supported: postscript, encapsulated postscript and Scalable Vector Graphics (SVG). The user can specify basic shapes (a.k.a. primitives in graphics parlance) and streams those shapes to an image. The alignment and distribution of those primitives can be controlled using the `Frame` and `Layout` classes. A `Frame` creates in essence a local coordinate system, nested within another `Frame`. A `Layout` generates a series of `Frame`s, following a logic associated with stacking or spacing. Of course these classes have been specialized. As an example, the `GridLayout` is used heavily by the application *calgps* to draw calendars. The command

```
calgps -e calgps.eps -3
```

generated the calendar in Figure 3

The *libvplot* library builds up *libvdraw* to create standard visualizations. These visualizations are comparable to those that can easily be generated using Excel, *gnuplot* or MATLAB. Figure 4 is an example of a surface plot that can be made with this library. The figure shows observation availability for a National Geospatial-Intelligence Agency (NGA) reference station. The availability is mapped to the topocentric (station-centric) frame, into bins of similar az-

Jul 2009

1538				1 182	2 183	3 184	4 185
1539	5 186	6 187	7 188	8 189	9 190	10 191	11 192
1540	12 193	13 194	14 195	15 196	16 197	17 198	18 199
1541	19 200	20 201	21 202	22 203	23 204	24 205	25 206
1542	26 207	27 208	28 209	29 210	30 211	31 212	

Aug 2009

1542							1 213
1543	2 214	3 215	4 216	5 217	6 218	7 219	8 220
1544	9 221	10 222	11 223	12 224	13 225	14 226	15 227
1545	16 228	17 229	18 230	19 231	20 232	21 233	22 234
1546	23 235	24 236	25 237	26 238	27 239	28 240	29 241
1547	30 242	31 243					

Sep 2009

1547			1 244	2 245	3 246	4 247	5 248
1548	6 249	7 250	8 251	9 252	10 253	11 254	12 255
1549	13 256	14 257	15 258	16 259	17 260	18 261	19 262
1550	20 263	21 264	22 265	23 266	24 267	25 268	26 269
1551	27 270	28 271	29 272	30 273			

Fig. 3 GPS calendar generated using *calgps*.

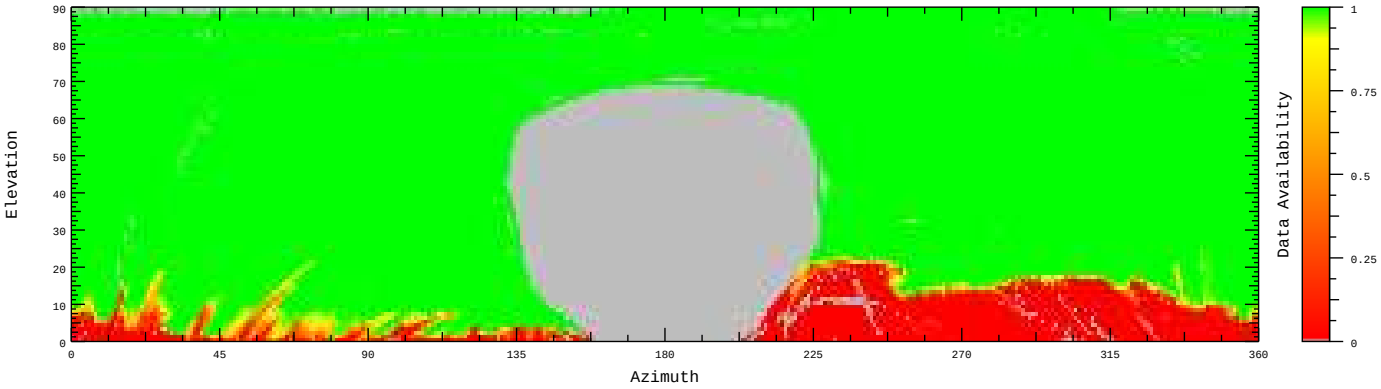

Fig. 4 Observation availability, for years 2007 and 2008, mapped to topocentric coordinates, in one degree by one degree bins, for the National Geospatial-Intelligence Agency (NGA) station in South Korea.

imuth and elevation angles. Two years of observations were compiled in this model. The presence of trees can be seen in the western azimuth (200 to 360 degree azimuth). On a modern PC, using an AMD 2 GHz CPU running Linux, this analysis took approximately one hour. However, the code for this example has not yet been submitted to the GPSTk. That work is in progress as of this writing.

NEW CLOCK STABILITY APPLICATION

Clock Tools, a recent addition to the GPSTk, allows for basic clock stability analyses. *Clock Tools* implement clock time-domain frequency stability metrics as well as data editing, noise identification, and plotting routines. The *Clock Tools* are interoperable with other GPSTk programs, allowing clock analyses to be easily run alongside other GPS analyses.

Given RINEX GPS observation and navigation files, clock estimates may be generated using the GPSTk *Observed Range Deviation (ORD) Tools*. Kalman filter and time interval analyzer data-sets provide another source of clock data. For instance, the TSC 5110A Time Interval Analyzer (TIA) compares two input clocks and outputs the phase difference between them at a 10- to 100-Hz sampling rate. These clock estimates are used for time-domain frequency stability analysis.

Frequency stability measures the ability of a clock to maintain its nominal frequency over a given period of time. From input clock data, both short-term and long-term stability can be assessed. The Allan variance represents the variance of a clock from its nominal value over a range of averaging times. The Allan variance is an IEEE standard for stability analysis [23]. An Allan variance plot is often helpful in stability analysis.

The *Clock Tools* suite currently computes the Allan, overlapping Allan, modified Allan, total Allan, overlapping Hadamard, and dynamic Allan deviation stability metrics. These *Clock Tools* may be run from the command line with their output piped to other GPSTk programs. Data formatting and outlier removal routines for GPS and time interval analyzer data-sets facilitate compatibility with the *Clock Tools* stability programs.

Design

The *Clock Tools* applications are encapsulated, so that the output of one tool acts as the input to the next. By using redirection and piping the user is able to connect the *Clock Tools* together in a way that allows for flexibility in analysis and ease of use.

Routines within the *Clock Tools* suite fall into four functional categories: input parsing, data grooming, data analysis, and plotting. Input parsing takes data generated outside of *Clock Tools* and puts it in a format understood within the suite. Data grooming cleans the data of outliers before analysis. Data analysis performs the frequency stability calculations. Plotting creates a graphical representation of the frequency stability.

Implementation

As mentioned, *Clock Tools* has four functional categories: input parsing, data grooming, data analysis, and plotting. Table 2 presents a description and example use of each tool. Note that *ordClock* makes an estimate of the receivers clock offset by averaging the ORDs of all the GPS satellites at each epoch. An epoch is an instant in time in which GPS data is recorded. The input parsers are the classes *ORDPhaseParser* and *TIAPhaseParser*. The data grooming tool is *rmoutlier*. The data analysis tools are *nallandev*, *oallandev*, *mallandev*, *totvar*, *ohadamarddev*, and *dallandev*. The plotting scripts are *allanplot* and *ddevplot.m*. Data flows from the parsers to data grooming to data analysis and finally to plotting.

Table 2 Description and example invocations of *Clock Tools*.

Tool	*Description*	*Example*
ORDPhaseParser	Parses data generated by the *ORD Tools*	`ordGen o input.o e input.n | ordClock|ORDPhaseParser>parsed.dat`
TIAPhaseParser	Parses data generated by the TSC 5110A Timing Interval Analyzer.	`TIAPhaseParser < raw.dat > parsed.dat`
rmoutlier	Removes outlier data within a set of data.	`rmoutlier < parsed.dat`
nallandev	Computes the Allan deviation.	`nallandev < parsed.dat`
oallandev	Computes the overlapping Allan deviation.	`oallandev < parsed.dat`
mallandev	Computes the modified Allan deviation.	`mallandev < parsed.dat`
totvar	Computes the total Allan deviation.	`totvar < parsed.dat`
ohadamarddev	Computes the overlapping Hadamard deviation.	`ohadamarddev < parsed.dat`
dallandev	Computes the dynamic Allan deviation.	`dallandev < parsed.dat`
allanplot	Plots the output of *nallandev*, *oallandev*, *ohadamarddev*, *totvar*, and *mallandev*.	`nallandev < parsed.dat | allanplot`
ddevplot.m	Plots the output of dallandev.	`octave ddevplot.m`

Example Usage and Plots

Two example command lines are given below. The first command generates clock estimates using the GPSTk ORD tools (*ordGen* and *ordClock*), parses the data (*ORDPhaseParser*), then removes outliers (*rmoutlier*), computes the overlapping Allan deviation (*oallandev*), and finally writes the data to a file (`output.oadev`). The second example takes data in from a file produced by the Timing Interval Analyzer (`raw.dat`), parses the data (*TIAPhaseParser)*, computes the overlapping Hadamard variance (`ohadamard`), and plots the output (`allanplot`).

```
ordGen   o  input.o   e  input.n | ordClock
| ORDPhaseParser | rmoutlier |
oallandev > output.oadev

raw.dat | TIAPhaseParser |
ohadamard | allanplot
```

Another example is shown below. This example is an analysis performed on a reference station that is part of the NGA GPS Monitor Station Network (MSN).

This analysis compares the results of running an overlapping Allan deviation calculation on data generated by *ordClock* from unsmoothed raw RINEX files versus data output by the NGA Kalman filter. The *ordClock* and Kalman filter data sets cover different time periods, but both data sets are good representatives of nominal MSN data. The overlapping Allan deviation of the *ordClock* generated data is represented by the blue line. The overlapping Allan deviation of the Kalman filter generated data is represented by the green line. The lighter slanted red line shows the 5071A CFS time-domain stability specification. The darker red line along the bottom represents the cesiums flicker floor. Note that drift removal may be necessary depending on the input dataset and the deviation computation performed. Figure 5 shows that the frequency stability of the clock estimates generated by the Kalman filter lies within the CFS specifications, but that the frequency stability of the output generated by *ordClock* from the raw data does not. Using raw GPS data and *ordClock* a stability trend is apparent, but further filtering and smoothing of the data is necessary to get more accurate clock stability estimates.

FUTURE

The most significant change planned in the near term for the GPSTk is that the RINEX-3 branch that will be integrated into the main. This will result in a major revision change for the source code, resulting in the release of GPSTk 2.0. As with other projects, when a major version number changes, that indicate the new code is not backwards compatible. Given the magnitude of such a change, a pre-release version of GPSTk 2.0 may be released to facilitate testing. The details regarding this transition will be discussed on the developer's mail list hosted by SourceForge, `gpstk-devel@lists.sourceforge.net`. Once finalized, the plan for transition to GPSTk 2.0 will be announced using another SF hosted mail list, `gpstk-announce@lists.sourceforge.net`.

Smaller changes are also planned. The *libsatvis* library will be added. This library will require extracting detailed visualizations from existing software, some of which are yet to be contributed to the GPSTk. Another library modification involves providing library calls to other programming languages such as Python and Octave. The interface that supports these languages is being redesigned and updated. Applications modifications are underway as well. The *mpsolve* application is being extended to solve and remove the biases associated with carrier phase. This technique will provide an essentially bias-free estimate of range-minus-phase multipath. These detailed extensions to the GPSTk will be the subject of future papers.

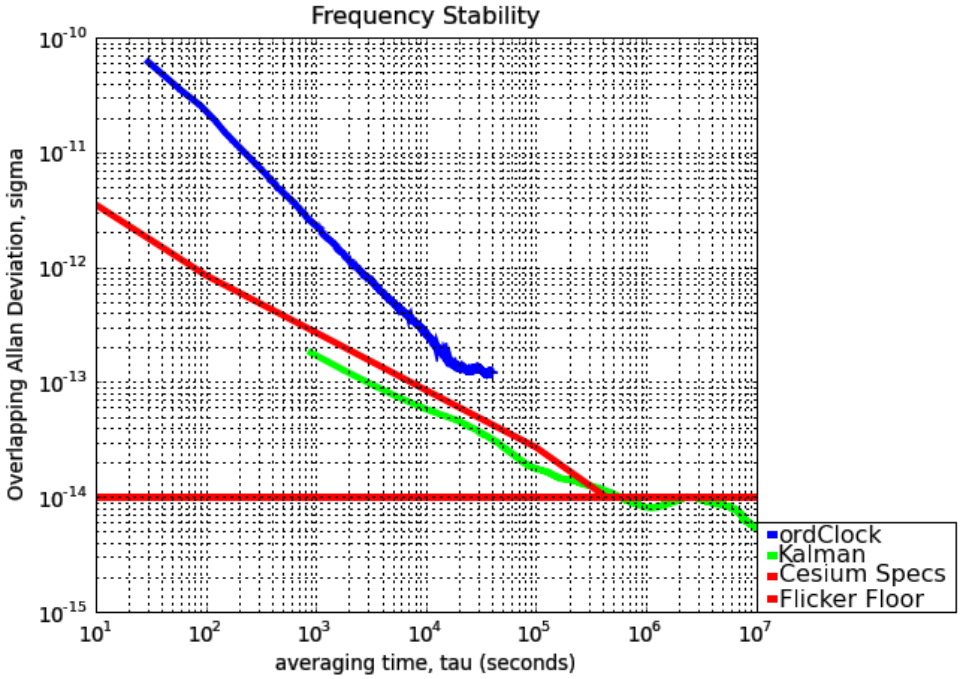

Fig. 5 Frequency Stability of *ordClock* and Kalman Filter Data

The aforementioned future modifications are only those planned by contributors at ARL:UT. The GPSTk is a community project, therefore, like all GNSSs, its future is open-ended.

ACKNOWLEDGMENTS

The authors would like to acknowledge the ARL:UT's long term support for the GPSTk. In addition, some of the work described in this paper was supported by NGA contracts N00024-07-D-6200-5-20, N00024-07-D-6200-5-36 and N00024-07-D-6200-5-19. Figure 4 was generated using code designed and written by Aaron Brogley, a graduate student at the University of Texas at Austin in the Department of Aerospace Engineering and Engineering Mechanics.

REFERENCES

[1] Werner Gürtner and Gerald M. Mader. *The RINEX Format: Current Status, Future Developments*. http://navcenter.org/ftp/GPS/REPORTS/rinex.txt, 1990.

[2] Werner Gürtner. *RINEX: The Reciever Independant Exchange Format Version 2.10*. http://www.ngs.noaa.gov/CORS/Rinex2.html, 1993.

[3] Werner Gürtner and Lou Estey. *RINEX: The Reciever Independant Exchange Format Version 2.11*. ftp://igscb.jpl.nasa.gov/igscb/data/format/rinex211.txt, 2006.

[4] Werner Gürtner and Lou Estey. *RINEX: The Reciever Independant Exchange Format Version 3.00*. ftp://igscb.jpl.nasa.gov/igscb/data/format/rinex300.pdf, 2006.

[5] Clifford Kelley. OpenSource GPS Open Source Software for Learning about GPS. In *Proceedings of the 17th International Technical Meeting of the Satellite Division of the Institute of Navigation*, Long Beach, California, September 2005.

[6] Clifford W. Kelley, Frederick Niles, and Douglas Baker. Development of the Open Source GPS Software Receiver Emulator. In *Proceedings of the 20th International Technical Meeting of the Satellite Division of the Institute of Navigation*, Fort Worth, Texas, September 2007.

[7] F. A. Niles. Integrating the GP2021 and Linux in Open Source GPS. In *Proceedings of the 18th International Technical Meeting of the Satellite Division of the Institute of Navigation*, Fort Worth, Texas, September 2006.

[8] GPSBabel website. http://www.gpsbabel.org/.

[9] gpsd website. http://gpsd.berlios.de/.

[10] *ISO/IEC 14882:2003: Programming languages: C++*. 2003.

[11] SGI's Overview of the Standard Template Library. http://www.sgi.com/tech/stl/.

[12] *The Doxygen Project*. http://www.stack.nl/ dimitri/doxygen.

[13] *The Subversion Project Home*. http://subversion.tigris.org/.

[14] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. O'Reilly, Sebastopol, CA, 2004.

[15] *TWiki(tm) - an Enterprise Collaboration Platform*. http://www.twiki.org/.

[16] Bernard Hofmann-Wellenhof, Herbert Lichtenegger, and Elmar Wasle. *GNSS - Global Navigation Satellite Systems*. SpringerWien, 2008.

[17] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2000.

[18] Ministry of Defence and Russian Space Agency. GLONASS Interface Control Document, Version 5.0. Technical report, Coordination Scientific Information Center, 2002.

[19] R. Benjamin Harris, Tracie Conn, Thomas Gaussiran, Chris Kieschnick, Jon Little, Richard Mach, David Munton, Brent Renfro, Scot Nelsen, Brian Tolman, Jonathan Vorce, and Dagoberto Salazar. The GPSTk: New Features, Applications and Changes. In *Proceedings of the 20th International Technical Meeting of the Satellite Division of the Institute of Navigation*, Fort Worth, Texas, September 2007.

[20] J. Kouba and P. Heroux. Precise Point Positioning Using IGS Orbit and Clock Products. *GPS Solutions*, 5, 2001.

[21] D. Salazar, M. Hernandez-Pajares, J. M. Juan, and J. Sanz. High accuracy positioning using carrier-phases with the open source GPSTk software. In *Proceedings of the 4th ESA Workshop on Satellite Navigation User Equipment Technologies (NAVITEC 2008)*, Noordwijk, The Netherlands, December 2008.

[22] D. Salazar, M. Hernandez-Pajares, and J. Sanz. Phase-based GNSS data processing (PPP) with the GPSTk. In *Proceedings of the 8th Geomatic Week*, Barcelona, Spain, February 2009.

[23] IEEE-1139-1999. *Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology-Random Instabilities*. Institute of Electrical and Electronics Engineers.

[24] R. Benjamin Harris, Brian Tolman, Tom Gaussiran, David Munton, Jon Little, Richard Mach, Scot Nelsen, and Brent Renfro. The GPS Toolkit: Open Source GPS Software. In *Proceedings of the 16th International Technical Meeting of the Satellite Division of the Institute of Navigation*, Long Beach, California, September 2004.

[25] *NAVSTAR Global Positioning System Interface Specification (IS-GPS-200), Revision D*.