The GPSTk: New Features, Applications, and Changes

Article

CITATIONS
5

READS
293

12 authors, including:

Robert Benjamin Harris
Raytheon Company
20 PUBLICATIONS **157** CITATIONS
SEE PROFILE

Tracie Conn
NASA
2 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Brian Tolman
University of Texas at Austin
25 PUBLICATIONS **187** CITATIONS
SEE PROFILE

Dagoberto Salazar
Trimble Navigation
17 PUBLICATIONS **203** CITATIONS
SEE PROFILE

The GPSTk: New Features, Applications, and Changes

R. Benjamin Harris, Tracie Conn, Thomas Gaussiran, Chris Kieschnick, Jon Little, Richard Mach,
David Munton, Brent Renfro, Scot Nelsen, Brian Tolman, Jonathan Vorce
Applied Research Laboratories, The University of Texas at Austin

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

BIOGRAPHY

R. Benjamin Harris is an Engineering Scientist at Applied Research Laboratories, The University of Texas at Austin (ARL:UT). He received a B.S. in Aerospace Engineering from the University of Texas at Austin (UT Austin) (1994), and is a Ph.D. candidate in the same department. He received an M.S. in Aeronautics and Astronautics from Stanford (2000).

Tracie Conn is an Engineering Scientist Associate at ARL:UT. She graduated with a B.S. in Aerospace Engineering from UT Austin (2006).

Thomas Gaussiran is the director of the Space and Geophysics Laboratory (SGL) at ARL:UT. He received his B.S. in Physics from UT Austin (1988) and M.S. (1991) and PhD. (1994) from Rice University.

Chris Kieschnick is an undergraduate majoring in Computer Science at 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).

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).

Jonathan Vorce is an undergraduate majoring in Aerospace Engineering at UT Austin.

Dagoberto Salazar is an Aeronautical Engineer (IUPFAN, 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, or GPSTk, is an open source project that provides a software suite to the GNSS community. The goal of the project is to free researchers to focus on research, not lower-level coding. The suite is composed of a library and a suite of applications built on the library.

Since the GPSTk was last presented at the ION GNSS 2006, the project has grown in scope and user base. Metrics provided by service providers SourceForge and Google quantify that growth. To accommodate the increase in demand and code complexity, the documentation system has been replaced with a community-managed website or wiki. The project has customized the wiki to support documentation commonly associated with GNSS software development.

Because the project is open source, new contributions are key to its success. The library has been reorganized to preserve contributions that are experimental or application specific. The new organization preserves usability and stability of the library's core functionality. The core library has been modified as well. Data structure and almanac processing classes have been added. New applications have also been contributed. This includes two suites of applications, one that simulates tracking the GPS signal and another that analyzes range residuals.

INTRODUCTION

The goal of this paper is to familiarize the reader with the purpose, status, and future direction of the GPSTk project. The overall capabilities of the applications and library are

briefly restated. New capabilities are discussed in detail. Examples of projects based on the GPSTk are described.

PROJECT SUMMARY

The goal of the GPSTk project is to provide a open source library and suite of applications to the satellite navigation community to free researchers to focus on research and not lower level coding.

GPS users employ practically every computational architecture and operating system. Therefore, the design of the GPSTk suite is as platform-independent as possible. Platform independence is achieved through use of the ISO-standard C++ programming language [1]. The principles of object-oriented programming are used throughout the GPSTk code base in order to ensure that the code is modular, extensible, and maintainable.

The GPSTk suite is composed of a library and a set of applications built on the library. The library provides a wide array of functions that solve processing problems associated with GPS such as the navigation solutions or reading standard formats such as RINEX. The applications support a greater depth of functionality. Each application implements a specific GNSS processing task.

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 latest code base is provided through a publicly accessible repository hosted by SourceForge. The repository is accessed using the *Subversion* utility. *Subversion* is a open source version control system that coordinates development among users over the Internet [2, 3].

Subversion is a console application that operates identically on a number of operating systems. 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. 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
```

Project Documentation

The GPSTk project's web site was initially hosted by SourceForge. It contained static content which was difficult to update. 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 [4] 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 capabilites. 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 format allowing for expressive 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 Doxygen documentation. 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.

Currently, the website hosts a PDF version of the GPSTk users manual. For the next revision of the manual, ARL:UT plans to convert it to a set of wiki pages. Then users will not only be able to update the manual, but also generate an offline copy of it using TWiki's PDF generation capability.

Project Activity Metrics

To give a brief insight into the number and diversity of visitors to the new wiki site, the following information was extracted from Google Analytics by looking at the traffic over the past 7 months. The website has seen on average over 1500 visits per month with over 1000 unique visitors per month. Visitors locate the website directly (about 30% of the time), by following a link from an other site (about 42% of the time), and by search engines (about 28% of the time). The website has seen visitors from 124 countries or territories with the top 10 countries of visitors being the United States, Germany, China, United Kingdom, Taiwan, Italy, Canada, Australia, Spain, and France. Approximately 30% of all vistors return to the website after the first visit. Figure 1 shows the distribution of visit worldwide.

The activity of the project is measured by SourceForge as well. SourceForge tracks project activity for each service it provides: web traffic, Subversion transactions, file downloads, bug entries, and more. This facility reports that the GPSTk binary packages have been downloaded more than 10,000 times from September, 2006, to the time of this writing, September 2007. The activity metrics can also be investigated over time. Two metrics are displayed in Fig-

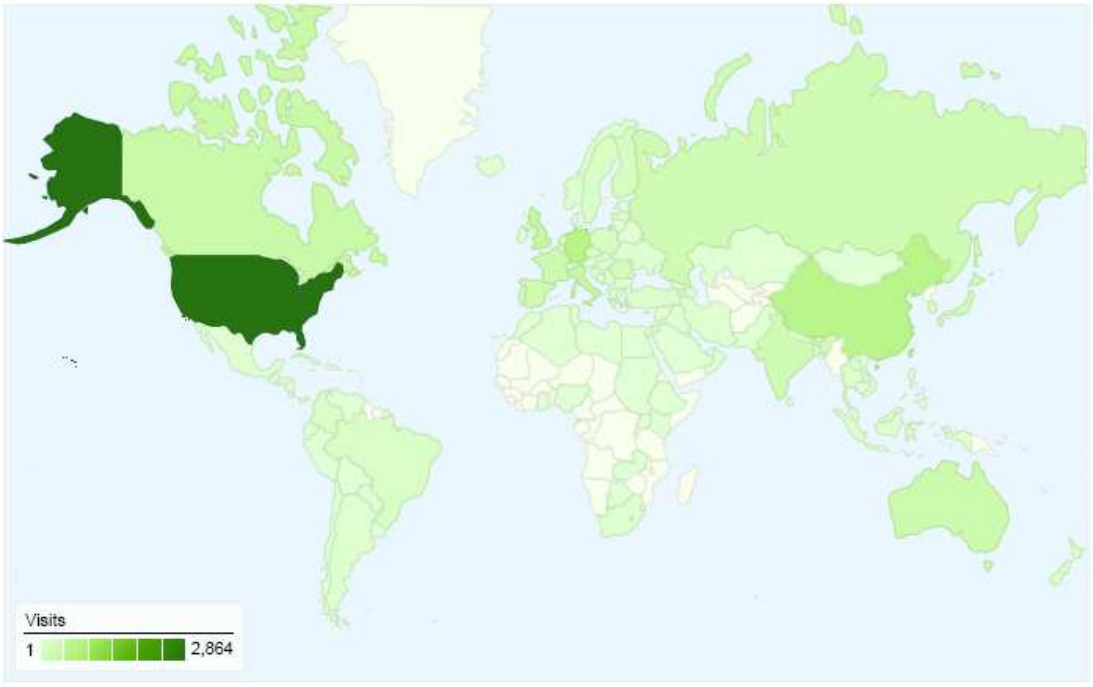

Fig. 1 Website visitor distribution, March to September 2007, courtesy Google Analytics

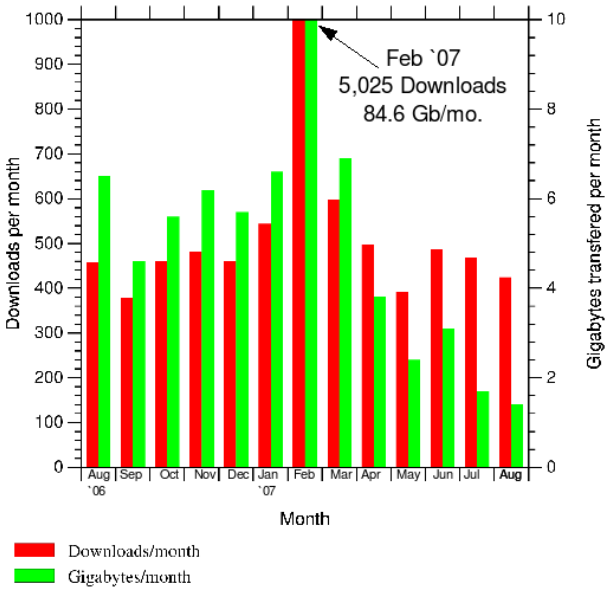

Fig. 2 Project activity through SourceForge site

ure 2. The number of packages downloaded per month rate has remained fairly constant while the amount of SourceForge traffic for the project has declined. It can be inferred that the project TWiki has offloaded traffic from the SourceForge site. The spike in February, 2007, has yet to be explained.

Two final metrics of project activity relate to the size and value of the code base. Version 1.3 of the GPSTk contains approximately 200,000 source lines of code. In comparison, version 1.2 contained 150,000 source lines of code. Version 1.1 contained 70,000. The value of the code base can also be estimated using the COCOMO model, which was first developed to manage aerospace software developments [5]. According to the COCOMO model, the estimated value of version 1.3 of the GPSTk source code is $7 million. These size and value metrics were generated using David A. Wheeler's SLOCCount utility [6].

Testing and the Release Process

As discussed in prior ION papers [7], testing of the GPSTk library is designed with three distinct goals in mind: testing is repeatable with a low amount of effort, testing is distributed along with the library to support both internal regression testing and to assure outside users and contributors of the quality of the library, and testing is designed to accommodate easy additions to the existing test suite. These goals have been retained, though the amount of the library that is covered by the test suite is expanding rather more slowly than anticipated.

Part of the reason for that slow expansion is the realization that testing is only one part of an overall release process. Following the move to on-line source management in July 2006 [7], the project has invested effort in defining and automating the release process. The release process is the process by which a stable and reviewed version of the code is first extracted from the Subversion repository, then

tested for quality. The ability to conduct the release process smoothly is critical to user support.

FUNCTIONAL SUMMARY

Library Functionality

The GPSTk library provides the base functionality for the GPSTk applications and for a number of other independent projects. There are several categories of functions:

- RINEX manipulation
- Time conversion, manipulation and storage
- Matrix computation
- Basic transforms of time and location
- Precise ephemeris processing
- Range prediction and error modeling
- Reference frame computations
- Statistics
- Troposphere delay models
- Earth orientation transforms
- FIC processing
- Almanac processing
- Low level BINEX input and output
- Broadcast ephemeris processing
- Clock models
- Code generation
- Cycle slip and discontinuity correction
- Expression evaluation
- Numerical integration
- Combinations and difference computations
- Data structures
- Navigation solution

A more detailed description of the functionality provided by the GPSTk library is generated from the code using the Doxygen utility [8]. An updated image of that documentation is kept on the GPSTk website [9]. A number of prior presentations, articles and papers discuss the capabilities of the library in more detail [10, 11, 12, 7, 13].

Application Functionality

A number of GPSTk applications are cross-platform. These applications are used to form the precompiled binary releases. The version 1.3 cross-platform applications are listed in Table 1. They are sorted into groups by their general purpose. Each row describes one GPSTk application or toolchain. A more detailed description of the applications can be found in References [7] and [13].

In contrast, other GPSTk applications are broadly useful but require more than the standard C++ environment to compile. These applications are summarized in Table 2.

RECENT ENHANCEMENTS

Yuma and SEM Almanac Processing

When performing GPS mission analysis or processing observations that span multiple months, an application may need to model the state of the constellation. Two publicly available formats provide this information, Yuma [14] and SEM [15]. Classes have been added to the core library of the GPSTk to process almanacs in these formats.

Library Restructuring

The core of the GPSTk is the library, and its source is located in the `dev/src` directory. This directory contains classes, algorithms, and data that are fundamental to GNSS processing and should compile on any system with a compiler that conforms with the international standard for C++ [1]. Often, however, the GPSTk contains code that could be useful in GNSS processing but do not fit this description. A number of such cases are present throughout the code base. The code could contain highly specialized algorithms, or be related to the message format of a specific receiver. The code could require libraries or system functions that are broadly available but not part of the C++ standard. To preserve code of this nature, the GPSTk project has introduced a new directory structure. This new code base is located in the `dev/lib` directory. The GPSTk source directory structure is illustrated in Figure 3.

The guidelines for adding to the `dev/lib` directory are similar to the rules for the `dev/apps` directory. Anything that is related to GNSS is acceptable. The intent is to encourage experimentation and development of new ideas. No files are directly placed in `dev/lib` and subdirectories are created to group related items. Subdirectories can have dependencies upon third party libraries and platform specific extensions. These subdirectories may depend upon the core GPSTk library (in `dev/src`) but they may not depend upon anything in `dev/apps`. More importantly, nothing in the core GPSTk library may depend upon items in `dev/lib`. This last point is imperative to maintain the portability and broad support of the GPSTk core library.

Cross Platform Applications

	Tool	Description	Execution Example
Transforms	calgps	generates a GPS calendar	`calgps -Y 2004`
	poscvt	converts a given input position to other position formats	`poscvt --geodetic="30.28 262.26700 167.64"`
	timeconvert	converts given input time to other time formats	`timeconvert --calendar="07 04 2006"`
	wheresat	outputs expected location of a satellite	`wheresat -b arl2100.06n -p 3`
Collect/Convert	ficfica/ficafic/fic2rin	convert FIC files between ASCII, binary, and RINEX formats	`fic2rin fic2100.06 rin121.06n`
	mdp2rinex	convert MDP files to FIC or RINEX files	`mdp2rinex -i mdpfile -o arl2100.06o`
	novaRinex	convert Novatel files to RINEX files	`novaRinex --input nova2100.06 --obstype L1`
	navdmp	dumps information from nav files to human readable formats	`navdmp -i arl2100.06n -o arl2100.06.dmp`
	RinexDump	flattens a RINEX file into a matrix	`RinexDump arl2100.06o 3 4 L1 L2`
Comparing & Validating	daa	performs a data availability analysis of collected data	`---`
	ddGen	computes double-differences from raw observations	`---`
	ephdiff	compares the satellite positions from two ephemeris sources	`ephdiff arl2100.06n fic2100.06`
	ficdiff	compares contents of two FIC files	`ficidff fic12100.06 fic22100.06`
	ficcheck/ficacheck	reads a FIC file and checks it for errors reporting the first found	`ficcheck fic2100.06 -t "07/20/2006 11:00:00"`
	findMoreThan12	predicts when more than twelve satellites are in view	`findMoreThan12 -e arl1256.06n -p "X Y Z" -m 10`
	mpsolve	computes statistical model of multipath combination	`mpsolve -o arl2100.06o -e arl2100.06n`
	ordGen/Clock/LinEst/...	models observed range deviations	`ordGen -o arl2100.06o -e apc14080 | ordEdit ... | ... | ...`
	RinSum	summarizes contents of a RINEX observation file	`RinSum -i arl2100.06o --EpochBeg 2006,07,20,13,20,00`
	rnwdiff/rnwdiff/rowdiff	compares contents of two RINEX files	`rowdiff arl1210.06o arl22100.06o`
	row/rnw/rnwcheck	read RINEX files and checks it for errors reporting the first found	`rnwcheck arl210.06n -e "07/20/2006 11:00:00"`
	rstats	computes standard, robust statistics on numbers in text stream	`rstats myfile.dat --col 3 --xcol 2`
Editing Data	DiscFix	cycle slip corrector	`DiscFix -i arl2100.06o --DT 1.5`
	EditRinex	edits a RINEX observations file based on user input	`EditRinex -IFalr2100.06o -TB1418,518000. -DSG17 -AOLW -AOPW`
	mergeFIC	sorts and merges input FIC files into a single file	`mergeFIC -i fic12100.06 -i fic22100.06 -o ficmerge2100.06`
	mergeRinObs/Nav/Met	sorts and merges RINEX files	`mergeRinNav -i arl2100.06n -i arl2110.06n arl210-211.06n`
	NavMerge	merges RINEX nav files into a single file	`NavMerge -oarlnavs.06n arl2100.06n arl2110.06n`
	rinexthin	decimates an input RINEX observation files to desired data rate	`rinexthin -f arl2100.06o -s 30 -o arl2100thin.06n`
	ResCor	edits RINEX files and computes corrections	`ResCor -IFalr2100.06o -OFarl2100mod.06o -DS12,12:00:00`
	sp3version	reads and writes from and to SP3 files	`sp3version --in mySP3a.dat --out mySP3c.dat --outputC`
Iono	IonoBias	solves interfrequency biases and a simple ionosphere model	`IonoBias --input arl2100.06o --nav arl2100.06n --XSat 3`
	TECMaps	creates maps of Total Electron Content (TEC)	`TECMaps --input arl2100.06o --nav arl2100.06n --LinearFit`
Positioning	DDBase	computes a network solution using carrier phase	`DDBase ... --ObsFile arl2100.06o --PosXYZ x,y,z,1 --Fix ...`
	ddmerge	ddmerge merges DDBase RAW and DDR output files	`ddmerge myRAW.dat myDDR.dat myData.dat`
	mergeSRI	combines solution and covariance results into a single result	`mergeSRI myResult1.out myResult2.out myResult3.out`
	posInterp	calculates user positions at a higher rate than observations	`PosInterp -o alr2100.06o -m 10 --outRinex myNewRinex.obs`
	PRSolve	generates autonomous position solution	`PRSolve -o alr2100.06o -n arl2100.06nn --XPRN 12`
	rinexpvt	generates autonomous position solution	`rinexpvt -o alr2100.06o -n arl2100.06n`
	vecsol	estimates short baseline using range or carrier phase	`vecsol station12100.06o station22100.06o`

Table 1 Cross platform applications.

Source Location	Function
`dev/apps/bindings`	Interfaces to Perl, Java, and other languages.
`dev/apps/swrx`	Simulation of the GPS signal and receiver tracking.
`dev/apps/RinexPlot`	Plotting observation combinations. Written in perl-Tk.
`dev/apps/MDPtools`	Processing TCP/IP or file-based observations in the MDP format.
`dev/apps/receiver`	Capture of data from an Ashtech receiver.

Table 2 Applications with dependencies outside standard C++

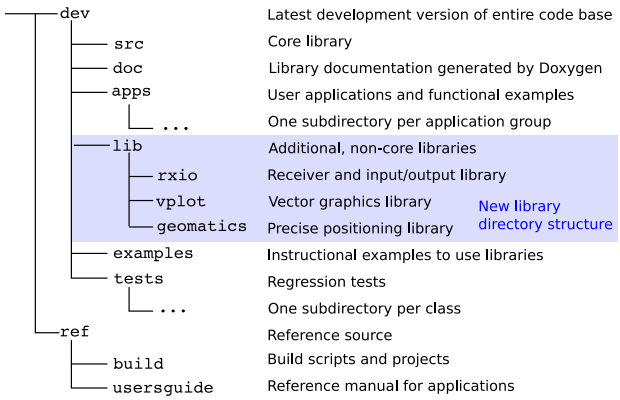

Fig. 3 Directory structure for GPSTk source

While anything GNSS related can be added under the `dev/lib` directory, the same is not true of the build process. It should only include items that will generally compile. To facilitate compiling the items in the new directory, the build will recurse into `dev/lib` by default. Just like in `dev/apps`, subdirectories that are listed in `dev/lib Jamfile/Makefile` will be built when the build is invoked from the `dev` directory. This also allows applications to be built and linked against the items under the `dev/lib` directory. While this guidance is somewhat vague, it does imply that the author is intending to keep the build from breaking under the common platforms that the GPSTk is used under (Linux/gcc and Windows/Visual C++). While the default build process will not recurse into every directory in `dev/lib` and `dev/apps`, individual directories may be built and installed manually. This is how libraries and applications that are platform specific are built and installed.

There are three subdirectories within `dev/lib`: `rxio`, `geomatics`, and `vplot`. The directory `rxio` contains `FFStream` definitions for the MDP receiver data type. The directory `vplot` contains a set of plotting routines. The `geomatics` directory contains precise positioning algorithms. Currently, there are several items that are in `dev/src` that are candidates to be moved to the `dev/lib` directory structure. Examples are the FIC and SMODF parsing routines as these formats are only used by a very small community.

Observed Range Deviation Toolchain

An Observed Range Deviation (ORD) is defined as the difference between an observed pseudorange and its predicted value [16, p.465]. A number of algorithms and terms are required to compute both the observed and predicted pseudoranges. Both can be corrected for known effects such as relativity, earth rotation and atmospheric delay. When all of these effects are accounted for, ORDs can be in the 10-30 cm range for a dual-frequency, geodetic quality receiver.

Prior to version 1.3 of the GPSTk, the *reszilla* application provided the capability to compute ORDs from GPS pseudorange, carrier phase, and Doppler data. However, as of fall 2006, *reszilla* had become quite complex and difficult to maintain. The program has since been redesigned into a suite of smaller applications that can be used in conjunction. Such a set of applications is often referred to as a *toolchain*.

The new applications are designed to be modular so that the output of one program can be used as the input to the others. The input and output can be captured to file, or streamed between applications in real time. In all of the platforms supported by the GPSTk, these two modes of are supported by redirection and piping. In UNIX and Windows command line environments, the redirect operator `>` captures standard output to a file. For example, `calgps > calendar.txt` writes a simple text based GPS calendar for the current month to a file `calendar.txt`. Piping is supported by the pipe operator `|`. An example of piping in the Windows command line environment is the command `calgps -Y 2004 | more` . By allowing the user to choose how to connect this toolchain, the new design allows for greater flexibility and ease of use. There are currently six tools that both replace and enhance the ORD computation capability of *reszilla*: *ordGen*, *ordEdit*, *ordClock*, *ordLinEst*, *ordStats*, and *ordPlot*.

The tool that must be run first in the chain is *ordGen*. Via command line options and data in a variety of file formats, the program accepts an antenna position, observation data, ephemeris information and weather data. The position is assumed to be a very accurate, surveyed location. The user may also specify which combination of observations are to be used, such as C/A-Code only or P/Y-Code on L1 and L2. ORD's will be flagged if the data seems erroneous based on

several criteria: low signal strength, extremely large pseudorange, large tropospheric correction, SV elevation below the horizon, or indication of bad SV health in subframe 1 of a valid 6-bit SV health bitfield. The ORD solutions for each SV are output in a format readable by the other ORD tools; it is called the ORD file.

The next tool that may be used is *ordEdit*. This program removes lines from the ORD file based on various criteria. Data may be filtered based on time, SV elevation, SV heath, or PRN number.

For many GPS receivers, the most significant effect to account for when correcting the observed range is the receiver clock offset, or the difference between a receiver's internal time and true GPS time. *ordClock* makes an estimate of the receiver clock offset by averaging the ORDs of all SVs for each epoch. The bias can be optionally removed from the ORDs for each epoch. These clock parameters may then be processed by *ordLinEst* to compute a linear clock model estimate. A natural byproduct of forming the linear model are deviations from the model, also known as *clock residuals*. Clock residuals are output by *ordLinEst*.

The ORD file can be processed by *ordStats* to summarize error statistics. The total number of ORDs and epochs that were flagged as erroneous are counted. For a set of user-specified elevation ranges, the standard deviation, mean and maximum values of the ORDs is listed, as well as the number of observations and the number flagged as erroneous. Looking at the results for each elevation bin is useful as ORDs tend to be much a higher when satellites are lower on the horizon.

Finally, *ordPlot* is a Python [17] script that uses matplotlib [18] to plot ORDs, clock offsets, linear clock models and clock model residuals. Basic graph properties may be specified, and the the final image may be saved in a number of raster- and vector- based image formats. Figure 4 is an example residual plot generated by *ordPlot*.

An example command line is shown below. In this example, observations are provided in the RINEX format and ephemeris information is provided as precise ephemerides. The results are edited using health information from a RINEX navigation message file and an elevation mask of ten degrees. A statistical summary is written to a text file. Finally, a PNG file is created with a plot of residuals and clock offsets. Note the use of pipes to complete the overall analysis procedure.

```
ordGen -o sample001.07obs -e apc14080 -e apc14081
-e apc14082 | ordEdit -e sample001.07nav -m 10 |
ordClock | ordLinEst | ordStats -o sampleStatsFile.txt |
ordPlot -s samplePlot.png
```

Toolchain to Prototype Software Receivers

The `dev/apps/swrx` directory contains classes and applications that simulate various parts of a GPS receiver. While this code has much of the functionality of a software GPS receiver, it is not intended to be a true software receiver. It is intended to be a research tool that is useful in understanding the formation of GPS signals, sampling them, tracking them, and forming observables. There are currently five separate applications in the *swrx* directory: *gpsSim*, *codeDump*, *plot*, *tracker*, *corltr*, and *iqdump*. As with other applications in the GPSTk, the swrx applications are a command-line toolchain. The command to plot the correlation curve of the P code on PRN 1 is listed below. The resulting plot is shown in Figure 5.

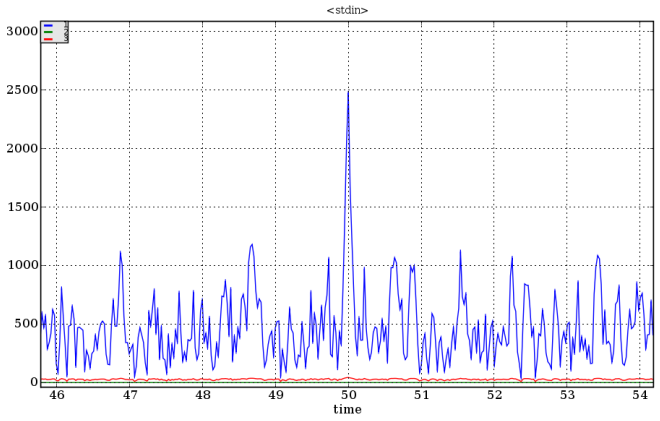

Fig. 5 P-Code correlation curve

```
gpsSim -q 2 -t 2e-3 -c p:1:1:50:0:p |
corltr -q 2 -c p:1:1:0:0 | plot
```

The *gpsSim* application simulates the digital output of the RF front end of a heterodyning GPS receiver. It first generates the RF signal for each code and carrier combination requested in the command line options. This RF signal can include the C/A- and/or P-Code, a 50 Hz square wave for the navigation bit transitions, a code offset, and a Doppler offset. This RF signal is then mixed with a local oscillator to form an IF and quantized to form in-phase and quadrature samples. These samples are then output as defined in the `IQStream` specified.

The *tracker* application implements a Costas tracking loop on an `IQStream` for a single code and carrier combination. The signal to be tracked is specified on the command line and includes a time and Doppler offset. While it can acquire phase lock without exact aiding, it does not implement any explicit acquisition algorithm. This tracker aligns the integrate and dump intervals with the nav bit transitions, demodulates the navigation data, and outputs the subframes of the navigation message.

The *corltr* application generates a local replica of the GPS signal and cross correlates this against the input `IQStream`.

The *codeDump* application simply dumps a section of the specified GPS spreading code. Its purpose is to verify code generation in the `CodeGenerator` classes against the IS-GPS-200 [19]. This application currently only outputs C/A- or P- Code.

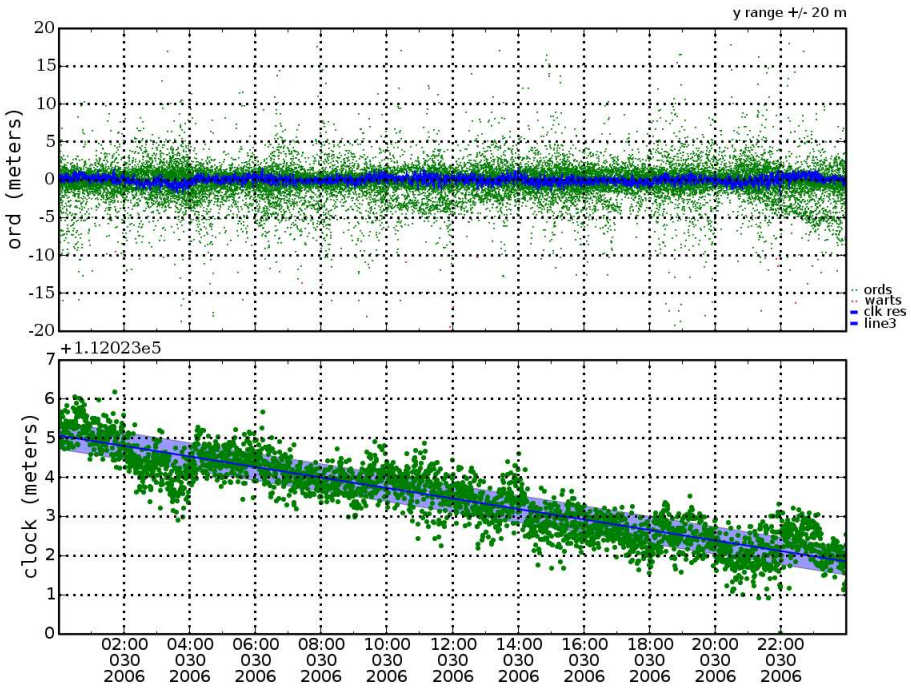

Fig. 4 Observed range deviations computed by the ORD toolchain

The *plot* application is a Python script that uses matplotlib to graph the outputs of *tracker*, *corltr*, and *codeDump*.

The *iqdump* application reads in a `IQStream` and dumps it as text to standard output. It also has an option to compute the percentage of time various bits are set.

Several classes form the source to the *swrx* applications. The `CodeGenerator` class is an abstract class that defines the interface to a pseudorandom ranging code. The intent is to make an interface that is similar to what could be easily built in hardware. The `CACodeGenerator` is an implementation of the GPS C/A-Code that uses the `ConstLinearRecurrentSequence` class. The `PCodeGenerator` class generates the GPS P-Code. This class currently uses the P-Code generator that is in `dev/src` but needs to be rewritten to use the `ConstLinearRecurrentSequence` class. The `ConstLinearRecurrentSequence` is a simulation of a linear feedback shift register. The `SVSource` class is a simulation of a single band of a GPS satellite.

The `CCReplica` class is used to generate a local replica of a single signal (a code/carrier combination). The `SimpleCorrelator` class is used to compute the correlation between a local replica and a sampled signal. The `EMLTracker` class is an early minus late Costas tracker. The `NavFramer` class takes the navigation message bits from a tracker and tries to find the start of a subframe.

`IQStream` is an abstract base class for a type of `IOStream` that is designed to read and write ADC in-phase and quadrature phase. It puts the samples in 500 byte frames, with a checksum and meta-data at the end of each frame. Currently there are three specific implementations of `IQStreams`

- `IQFStream`, using floating point numbers for each sample.
- `IQ2Stream`, using 2 bit sign/magnitude values.
- `IQ1Stream`, using 1 bit values.

NEW DATA STRUCTURES AND ALGORITHMS

A new set of data structures called *GNSS Data Structures* (GDS) have been introduced to the GPSTk library. Their purpose is to solve data management problems that are difficult to solve with simpler data structures such as vectors or matrices. These structures hold several kinds of GNSS-related data, properly indexed by station, epoch, satellite and type. These GDS, implemented using the `map` data structure defined by the C++ standard library, act like *white boxes*, in the sense that all the information within is available for the developer.

The GDS paradigm is complemented with several associated *processing* objects. These objects reach into the GDS and add, delete and/or modify what is needed (according to their function), and leave the results in the corresponding GDS object, appropriately indexed.

These processing objects are designed to use sensible defaults in their parameters, but those parameters may be

tuned to suit specific applications. This is usually done before using the objects, although in some cases they may auto-adjust according to input data.

A `ModeledPR` (Modeled Pseudorange) object may take as parameters observable type, ephemeris, ionosphere, and troposphere models, and will add to the incoming GDS derived data such as geometric range, satellite elevation and azimuth and prefit residuals. The `ModeledPR` will also automatically remove those satellites with missing critical data such as ephemeris.

A `CodeSmoother` object takes as parameters a given code observable type and a maximum window size. The `CodeSmoother` reads successive code and phase observations, as well as the cycle slip flag. It computes a new smoothed observable, self adjusting the window size along the way. At each epoch it replaces the original observable with the new smoothed estimate of range.

An important concept here is that the GDS grows or changes dynamically. Each processing operation can introduce derived data or filter existing data. This sets an order in which most processing objects should be used, with some at the beginning feeding others at the end.

The former ideas are coupled with a handy redefinition of operator $>>$. Typically, this operator is redefined like the following:

```
gnssRinex& operator>>(gnssRinex& gDat, CodeSmoother& cS)
```

Here, the GNSS data structure *gDat* is at the left of the operator, and the `CodeSmoother` object `cS` is at the right, meaning that the data *flows* from the data structure into the smoother.

Note the `>>` operator returns a reference to the **same** incoming data structure. This means that `gDat` feeds `cS`, it is modified, and the resulting, modified `gDat` is put in place of the original `gDat`.

The former allows concatenation of `>>` operators, allowing a programming style that clearly shows how the data is flowing along the processing steps. It reflects the pipes used in UNIX shell and Windows batch programming. More stream operation examples follow.

The first example involves simple `C1`-based processing, solving with a LMS solver. The results are given in a XYZ reference frame:

```
gRin >> myFilter >> model >> solver;
```

`gRin` is a GNSS data structure obtained from a RINEX observation file epoch record. `myFilter` is a `SimpleFilter` object that checks that `C1` is within reasonable limits. `model` is a `ModeledPR` object, and `solver` is a `SolverLMS` object.

If the previous results are needed in a North-East-Up frame, the system may be solved using $dLat$, $dLon$, dH and $(c \cdot dt)$, instead of dx, dy, dz and $(c \cdot dt)$:

```
gRin >> myFilter >> model >> baseChange >> solverNEU;
```

In this case, `baseChange` is a `XYZ2NEU` object that computes the corresponding $dLat$, $dLon$ and dH coefficients and adds then to `gRin2`. Also, `solverNEU` is a common `SolverLMS` object reconfigured to solve an equation system with $dLat$, $dLon$, dH and $(c \cdot dt)$ as unknowns.

The following example is more complex. First, the PC, LC, LI and MW (Melbourne-Wübena) combinations are formed and added to the GDS object. PC and LC are here used to denote the code and phase ionosphere-free combinations, respectively. LI stands for the the ionospheric combination (also called geometry-free combination). MW represents the linear combination of narrow-lane and wide-lane combinations that is very useful to find phase cycle slips. The combinations are used to mark cycle slips. The combinations and cycle slips are used to create a smoothed pseudorange. Finally position is computed using a Weighted-LMS method based in MOPS weights [20]. The processing is completed in a single processing line:

```
gRin >> getPC >> getLC >> getLI >> getMW >> markCSLI >>
markCSMW >> smoothPC >> pcFilter >> modelPC >> mopsW >>
baseChange >> solverWMS;
```

The final example applies code-based, differential GPS techniques. After some synchronization code (not shown here), the data from the reference station can be processed. The resulting GNSS data structure is then assigned as the reference data of a `DeltaOp` object, `delta`:

```
gRef >> myFilter >> modelRef;
delta.setRefData(gRef.body);
```

Then, the `rover` GPS receiver data is fully processed:

```
gRin >> myFilter >> model >> delta >>
baseChange >> solverNEU;
```

`delta` will subtract from `gRin` prefit residuals from the corresponding `gRef` prefit residuals, deleting (by default) the satellites with missing data. The rest of the data processing is as shown before.

In summary, this new GNSS data processing paradigm holds the promise of greatly enhancing and improving GNSS data processing in the GPSTk. It provides an encapsulated, uniform and intuitive way of transporting GNSS data along the processing chain. This area of the GPSTk is currently under active development.

PROJECTS

The GPSTk has been used to support GNSS-related projects across the globe since September 2006. These project include:

- Students at the University of Texas at Austin in the Department of Aerospace Engineering and Engineering Mechanics have established a GPS reference station using GPSTk. The reference station utilizes both data collection and validation applications. The observations are collected at a high data rate, 1.5 sec-

ond sample period, synchronized with a rubidium frequency standard. The station's documentation and data can be accessed at http://lagrange.ae.utexas.edu/.

- At the Research Establishment for Applied Science (FGAN), Germany, the GPSTk is being used to create a mobile radar tracking system. The system also incorporates software from the open source gpsd project [21] to communicate with receivers.
- Second year students in the Satellite Surveying class at the Technical University of Helsinki (TKK, formerly HUT), Department of Surveying, used *DDBase* to survey. They surveyed one short baseline and one long, over 300 km. Other applications were used to plan surveys and generate a priori estimates of station position and clock models.

FUTURE WORK

Because the GPSTk is an open source project, its future is open-ended. Some particular enhancements have been identified by developers within ARL:UT. In the near term, a Java-based front-end for GPSTk applications that has already been developed will be contributed. Also, the applications in `dev/apps/swrx` will be extended to simulate Binary Offset Carrier modulated signals and multipath. In addition, ARL:UT is in the process of proposing sponsored tasks that will directly enhance the GPSTk, including support for RINEX 3 [22] and precise point positioning (PPP).

The GPSTk has received one special request for support but as yet no student, developer, or group has been able to fulfill. The OpenMoko project's goal is to create the world's first open source phone operating system [23, 24]. The hardware platform for the OpenMoko is depicted in Figure 6. However, the only closed source part of the project is the GPS processing task or daemon. The project would like to replace the task with one based on the GPSTk.

Furthermore, it may be possible to get funding for a student to investigate this topic. Google sponsors a Summer of Code (SoC) initiative which funds students to spend a summer contributing to any one of a list of projects. The list of project is reviewed and prioritized by a number of open source organizations. In 2007, the project to integrate the GPSTk into the OpenMoko was listed, but not chosen by an SoC sponsored recipient [25, 26]. If Google again sponsors a SoC for 2008, the GPSTk and OpenMoko projects would help support a student interested in this project.

SUMMARY

After three years since its public release, the GPSTk continues to amass new contributions and new developers. The third year required the GPSTk team to consider new ways to collaborate with developers distributed on a global scale.

Fig. 6 OpenMoko hardware

The project website became a primary means to provide dynamic documentation and to host user questions. The code repository was restructured to accommodate innovation while preserving core functionality. Applications were added to support processes that are key to GNSS research: range measurement validation, and simulation of GNSS signal tracking. Both student and professional projects continue to choose the GPSTk as a processing basis.

ACKNOWLEDGMENTS

The authors would like to acknowledge support from Applied Research Laboratories, The University of Texas at Austin, and the U.S. Federal Aviation Administration. The authors would also like to recognize the developers who have contributed both code and time to the project. In particular Chris Keuthe at the University of Calgary for his support of the make system, and Dr. Martin Vermeer at TKK, for his broad support of the project. GPSTk users have motivated development discussions and provide descriptions of their work. This includes Ulla Kallio at TKK and Massimo Burcheri at FGAN. Drs. Lightsey, Buckley and Stearman at UT Austin, and many of their students, have provided feedback and numerous project reports as well.

REFERENCES

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

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

[3] Ben Collins-Sussman, Brian W. Fitzpatrick, and

C. Michael Pilato. *Version Control with Subversion*. O'Reilly, Sebastopol, CA, 2004.

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

[5] Barry W. Boehm. *Software Engineering Economics (Prentice-Hall Advances in Computing Science & Technology Series)*. Prentice Hall PTR, October 1981.

[6] *Counting Source Lines of Code (SLOC)*. http://www.dwheeler.com/sloc/.

[7] R. Benjamin Harris, Timothy Craddock, Tracie Conn, Thomas Gaussiran, Eric Hagen, Anthony Hughes, Jon Little, Richard Mach, Scot Nelsen, Brent Renfro, and Brian Tolman. Open Signals, Open Software:Two Years of Collaborative Analysis using the GPS Toolkit . In *Proceedings of the 18th International Technical Meeting of the Satellite Division of the Institute of Navigation*, Fort Worth, Texas, September 2006.

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

[9] GPSTk website. http://www.gpstk.org/.

[10] Brian Tolman and R. Benjamin Harris. The GPS Toolkit. *Linux Journal*, pages 72–76, September 2004.

[11] 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.

[12] Brent Renfro, R. Benjamin Harris, Brian W. Tolman, Thomas Gaussiran, David Munton, Jon Little, Richard Mach, and Scot Nelsen. The Open Source GPS Toolkit: A Review of the First Year. In *Proceedings of the 17th International Technical Meeting of the Satellite Division of the Institute of Navigation*, Long Beach, California, September 2005.

[13] R. Benjamin Harris and Richard G. Mach. GPSTk–An Open Source GPSTk Toolkit. *GPS Solutions*, 11, March 2007.

[14] *Definition of a Yuma Almanac*. http://www.navcen.uscg.gov/gps/gpsyuma.htm.

[15] *Definition of a SEM Almanac*. http://www.navcen.uscg.gov/gps/gpssem.htm.

[16] Bradford W. Parkinson and James J. Spilker Jr., editors. *Global Positioning Theory: Theory and Applications*, volume 1. AIAA Press, Reston, Virginia, 1996.

[17] G. van Rossum and F. L. Drake, editors. *Python Reference Manual*. Virginia, USA, 2001. Available at http://www.python.org/.

[18] John D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.

[19] LLC ARINC Engineering Services. Interface Specification IS-GPS-200, IRN-200D-001. Technical Report Revision D, GPS Joint Program Office, El Segundo, California, March 2006.

[20] Minimum Operational Performance Standards (MOPS), version C. Technical report, Radio Technical Commission for Aeronautics. RTCA/DO-229C.

[21] *gpsd - a GPS service daemon*. http://gpsd.berlios.de/.

[22] 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.

[23] *OpenMoko mobile phone development site*. http://www.openmoko.com/.

[24] *OpenMoko wiki*. http://wiki.openmoko.org/wiki/Main_Page.

[25] *Google Summer of Code 2007*. http://code.google.com/soc/2007/.

[26] *Open Moko's Summer of Code project list*. http://wiki.openmoko.org/wiki/Summer_of_code.