Porting goGPS from MATLAB to Java: performance analysis, tests and results

Eugenio Realini[1], Daisuke Yoshida[2], Lorenzo Patocchi[3], Mirko Reguzzoni[4], Venkatesh Raghavan[1]

[1]Graduate School for Creative Cities, Osaka City University, 3-3-138 Sugimoto, Sumiyoshi-ku, 558-8585 Osaka, Japan – realini@gscc.osaka-cu.ac.jp; raghavan@media.osaka-cu.ac.jp
[2]Faculty of Liberal Arts, Tezukayama Gakuin University, 2-1823 Imakuma, Osakasayama City, 589-8585 Osaka, Japan - yoshida@la.tezuka-gu.ac.jp
[3]CRYMS Sagl, Manno, Switzerland - lorenzo.patocchi@cryms.com
[4]Dept. of Geophysics of the Lithosphere - OGS, c/o Politecnico di Milano - Polo Regionale di Como, via Valleggio 11, 22100 Como, Italy - mirko@geomatica.como.polimi.it

Abstract

goGPS is an application for achieving sub-meter accuracy with low-cost GPS receivers, mainly through the use of RTK positioning, Kalman filtering, DTM aid and network-constrained navigation. It aims on one hand at achieving optimal positioning with the miniaturized GPS devices commonly used for car and personal navigation, on the other hand at lowering instrumentation costs by substituting proprietary hardware (e.g. GPS chipsets) with open source positioning software. goGPS can work either in real-time or post-processing, by acquiring raw GPS data in input and providing positioning (i.e. coordinates) in output. Though originally developed in MATLAB and targeted to a research-oriented environment, goGPS was recently ported to Java in order to allow a wider user base to develop and use it. Since real-time GPS positioning heavily relies on fast matrix computation, a careful selection of Java matrix libraries was carried out in order to obtain optimal performances. A client-server architecture was also developed and tested in order to let lightweight clients just acquire raw GPS data, send them to a server for processing and receive back the accurate positioning. The Java version of goGPS is being developed as an open source collaborative project, on the wave of a constructive synergy between universities and private companies both in Europe and Japan.

1. Introduction

Recently, as the word "ubiquitous society" describes, the technologies related to the Internet and to location information are developing remarkably. Handheld devices with built-in GPS capabilities, such as smart-phones, have spread conspicuously, many location based applications for mobile devices are being developed and more and more use of location information draws attention for business.

However, built-in GPS receivers in cellular phones and PNDs (Portable Navigation Devices) offer low-quality positioning with an accuracy of around 3 – 5 m. Besides, in locations of urban areas enclosed by skyscrapers, the accuracy degrades further.

Business with highly accurate location information has not been spread at consumer level yet, since getting high positioning accuracy is still costly. There are various grades in GPS devices: professional double-frequency receivers can get positioning with an accuracy of some centimeters by Real-Time Kinematic (RTK), professional single-frequency receivers can improve the accuracy to less than 1 meter using Differential GPS (DGPS) correction data or single-frequency RTK, while low-cost receivers, which are mainly used for cell-phones and car navigation devices, have accuracies of some meters. These different devices cost according to their hardware quality and therefore their accuracy: professional double-frequency ones cost US $ 20,000 ~ 30,000, professional single-frequency ones US $ 1,000 ~ 3,000 and low-cost ones less than US $ 100 (Figure 1). Besides, software in these GPS devices is proprietary, thus source code is not opened to the public. This means that positioning algorithms cannot be modified or tuned to improve the accuracy for specific applications.

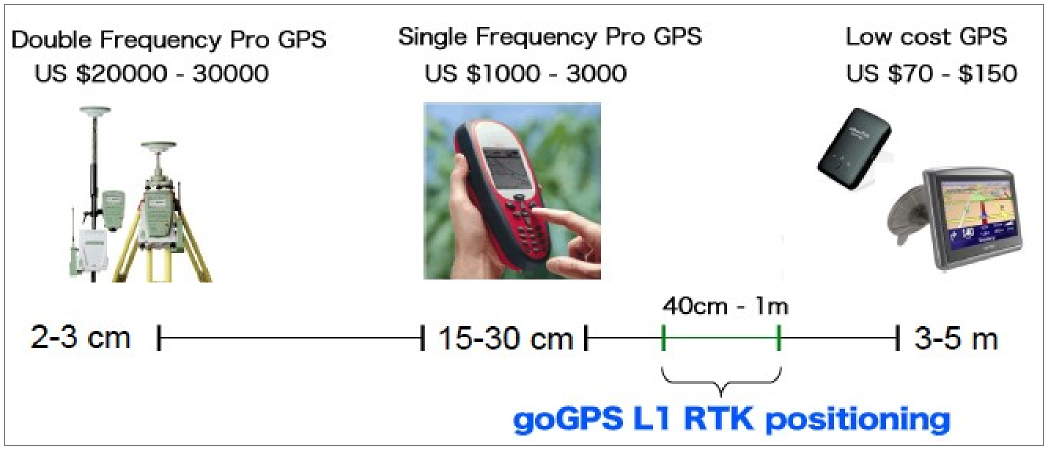


Figure 1. GPS accuracy and cost

goGPS (http://www.gogps-project.org) is an open source software that can achieve accuracies of less than 1 m, depending on sky visibility conditions, using low-cost GPS instrumentation (specifically, u-blox Evaluation Kits AEK-4T and EVK-5T are usually employed for tests). goGPS has the potential to reduce costs greatly since there are no limitation in development and distribution, in contrast to proprietary software. goGPS was developed since 2007 in MATLAB numerical computation language (MathWorks, Inc. - http://www.mathworks.co.jp/products/matlab/). MATLAB already implements a great number of mathematical functions, it excels in matrix manipulation and computation and it makes data visualization easy and straightforward. However its processing speed is relatively slow, especially for parsing big text files, like often GPS data files in RINEX format are, and for displaying data: therefore it is not of practical use to post-process big amounts of GPS data. Moreover, MATLAB is a commercial software platform that severely limits users to participate in the development or even just use goGPS. Therefore, it constricts goGPS development within universities and research institutions even though it is an open source project. In this work, the core processing functions of goGPS have been ported to Java code in order to help the software spread at both user and developer level, to improve the processing and visualization speed, and to obtain a way to provide goGPS processing as a web service, which is one of the main future goals in the goGPS project.

1.1 goGPS L1 low-cost RTK positioning

Low-cost GPS devices typically implement small patch or helix antennas and single frequency (L1) receivers functioning in stand-alone mode; moreover, they are often highly sensitive to GPS signal (even if degraded) in order to assure positioning in dense urban environments. The error budget can thus be identified both as a shift due to atmospheric delays and clock errors, not erased or negligible in stand-alone mode, and as noise due to the low quality of involved hardware and to multipath resulting from high sensitivity. The first part of the error budget can be removed by applying relative positioning with respect to a master station, which erases clock errors and makes atmospheric errors negligible if the rover is within 10 km from the master station. The most efficient way of doing this is to exploit RTK (Real Time Kinematic) services provided by a network of permanent GNSS stations, in particular using VRS (Virtual Reference Stations). The second part of the error budget cannot be systematically removed or made negligible, thus it is needed to minimize it: the use of Kalman filtering (Kalman, 1960; Hofmann-Wellenhof et al., 2003), observation weighing and external information sources not directly related to GPS (e.g. height values from a digital terrain model if the receiver is on the ground, line constraints if it is moving on a path known *a priori*, etc.) can help in this sense. The basic way of using a Kalman filter for stand-alone GPS-only navigation is to apply it directly on the estimated positions. The positioning is typically computed by least-squares adjustment from code and phase measurements and the Kalman filter acts subsequently with a smoothing effect on the estimated trajectory. This approach with low-cost devices results in a positioning as accurate as that provided by performing absolute positioning using phase-smoothed code measurements (5 - 10 m of error). This means that atmospheric delays and clock errors cannot be efficiently corrected, because this kind of receivers is usually not designed for relative positioning, and computed trajectories are often affected by large biases. The Kalman filter implemented in goGPS is not applied on estimated positions, but on double difference observations with respect to a reference station (i.e. relative positioning). The possibility of directly processing GPS observations allows us to remove most of the bias associated to absolute positioning, obtaining accuracies of less than 1 m. Its use in real-time obviously requires a wireless connection to the Internet in order to receive the master station observations and a GPS chipset that provides raw measurements as output. The goGPS Kalman filter is "modular", in the sense that it is conceived in such a way that it can be easily integrated with additional components. This means that new sources of measurements (i.e. observations) can be added or disabled without major changes in the algorithm. In its current state goGPS includes a first observation module applied to code double differences, a second one applied to phase double differences and a third one that exploits the information coming from a DTM (Digital Terrain Model). goGPS main targets are single-frequency low-cost devices, but its code and phase modules are designed to work either in single-frequency or double-frequency mode. Therefore, it can be used also with double-frequency receivers, if it is needed. Finally, goGPS includes also an alternative version of its main Kalman filter algorithm, designed to obtain line-constrained positioning (e.g. to navigate on a network of roads). In this case the DTM information is not used anymore, because the constraint is already three-dimensional. For a detailed description of goGPS Kalman filter, see Realini (2009).

2. Porting goGPS to Java

When it was decided that goGPS needed to be ported from MATLAB to a more widely used language, the main candidates were C/C++ and Java. Despite favoring at first the C/C++ option, mainly for concerns related to processing speed, when the group at Politecnico di Milano (Italy) and the IT company CRYMS Sagl started a collaboration on goGPS development the Java option was reconsidered and finally chosen. This section briefly illustrates why goGPS was initially developed in MATLAB, then, while introducing Java, it explains the reasons that lead goGPS to be ported to this language. Considerations about choosing a proper library for fast matrix computation are then made, followed by a performance benchmarking comparing goGPS MATLAB and Java versions.

2.1 goGPS MATLAB version

goGPS was initially developed in 2007 at the Geomatics Laboratory of Politecnico di Milano as a tool for teaching students about GPS positioning and Kalman filtering. MATLAB was chosen as a programming language in order to let students develop and test algorithms on their own, since it is largely used for many engineering-oriented university courses and they were already familiar with it. The prototype software that later became goGPS was used also for conducting research on GPS navigation, and also in this context MATLAB simplicity and efficiency in matrix operations made it an ideal development environment for quickly implementing and testing new approaches to positioning and Kalman filtering. Moreover, when goGPS development effectively started with the aim of obtaining a complete application, Prof. Kai Borre's EASY (Borre, 2003; Borre, 2010) open source software package for code-only standalone GPS positioning (written in MATLAB) was used as a basis for developing the basic algorithms and as a reference to check goGPS initial results and performance.

Nevertheless, MATLAB poses some drawbacks. For example, the text file parsing functions are very inefficient if text files are not already formatted in a simple delimited structure, and since the main format for storing and exchanging GPS raw data is RINEX, which employs quite complex data structures as ASCII text, parsing large datasets spanning several hours (tens of megabytes) becomes quite slow. Moreover MATLAB display functions, although easy to use and straightforward, are very inefficient too, especially in real-time functioning, when timing and synchronization between rover and master data streams become crucial issues. Last but not least, MATLAB is neither free nor open source software, and because of this goGPS usage and development was in fact limited to researchers and professionals who can afford to buy a MATLAB license.

2.2 goGPS Java version

Java technology was born towards the end of the 90s and since then it has grown at a fast pace, aided by the expansion of the Internet (one of its main target of application) and by its ease of use. The number of Java developers quickly increased, especially after Java became open source in 2007 (under GNU GPL license), up to the 9 millions of today. Thanks to its easy portability (write once, run anywhere), it is used worldwide to develop solutions for distributed environments (server, client, embedded), for network communication and data storage (Oracle Corporation now owns Java after it acquired Sun Microsystems in 2009/2010).

A typical argument against the adoption of Java is that it is slower than C/C++ because it is a pre-compiled language (i.e. it is not compiled with the native language of the machine were it is executed), interpreted by the Java Virtual Machine (JVM) which has to be installed on the host system. Nevertheless, performance benchmarks with C/C++ often show that modern Java implementations are getting nearer and nearer to C/C++ performance (sometimes they are even better), since Just In Time (JIT) compilation was introduced.

On the other hand, adopting Java brings benefits like being able to release platform independent executables, simplifying the source code and having the possibility to rely on numerous third-party libraries for the most diverse tasks. In the specific case of goGPS, switching from MATLAB to Java allows users to run the software without owning a MATLAB license (on any operating system that can install a JVM), besides providing the possibility to easily enable multiple threads to parallelize tasks.

Up to now the core algorithms of goGPS have already been ported to Java, which means that goGPS Java can read input data from RINEX files and perform positioning as code-only stand-alone epoch-by-epoch least squared adjustment (LSA), code double-differenced epoch-by-epoch LSA, and Kalman filtered solution using both code and phase. In every case observations can be weighted according to satellite elevation, signal-to-noise ratio or both. When using the Kalman filtered solution, changes in satellite configuration like satellite addition and loss, pivot change and cycle slips are managed in order to keep into account changes in the estimation of phase ambiguities.

The differences in the final positioning between MATLAB and Java implementations are always significantly below the accuracy achievable with the test instrumentation (goGPS with u-blox receiver). In any case these differences are never higher than 1 mm.

2.3 goGPS Java performance

A crucial issue to obtain good computation performance with goGPS is matrix computation: at every epoch Kalman filter equations are applied on matrices that, depending on the number of available satellites, Kalman filter order and whether one frequency or two frequencies are used, can have sizes ranging from ~30 to ~80 elements per line and/or column. Moreover, at every epoch several roto-translations are applied for getting satellite and receiver positions in different coordinate and reference systems (thus typically on 3x3 matrices).

Several open source Java matrix libraries exist: in some cases they focus specifically on handling matrices and linear algebra, while in other cases the matrix-related classes are part of a wider toolset for scientific computation (Table 1 includes the most complete libraries with regards to matrix manipulation and linear algebra). In order to choose one of them, particular care was adopted to get the best performance for the matrix size used in goGPS.

Table 1. Java matrix libraries

	Pure Java	Multi-threaded
Apache Commons Math	○	
Colt	○	○
EJML	○	
JAMA	○	
jblas		
JScience	○	○
MTJ	○	
ojAlgo	○	○
Parallel Colt	○	○
UJMP	○[1]	○

[1] UJMP may be optimized by including native code

Moreover, since goGPS could be run either locally on low-specification devices (even as embedded solutions) or in client-server architectures, specifically server-side, on multi-core machines with high computational capabilities, libraries performing well also on lower grade CPUs (thus possibly in single-threaded mode) were preferred. Finally, ease of use, good documentation, correctness of results and activity of the project were obviously taken into account. A useful open source tool for benchmarking different Java matrix libraries, called JMatBench (Abeles, 2010a) was used to check the libraries performance according to matrix size on different grades of CPU. Finally EJML (Abeles, 2010b) was chosen according to the requirements described above.

The performance of goGPS Java version were compared to those of goGPS MATLAB version on different machines with different operating systems. A server-grade PC and a client-grade laptop were used for the tests (specifications are reported in Table 2).

Table 2. Specifications of test machines

	Server-grade PC	Client-grade laptop
CPU	Intel Core i7-860 2.80 GHz (64 Bit)	Intel Pentium M 1.30 GHz (32 Bit)
RAM	8 GB	1 GB
Operating System	Linux Slackware64 13.0 (Kernel 2.6.29.6)	Dual boot with: Windows XP Professional SP3 Linux Slackware 12.2 (Kernel 2.6.27.7)

The Java Runtime Environment version was homogenized on all test machines: build 1.6.0_16-b01 was used. As for the Virtual Machine (VM), Java HotSpot(TM) 64-Bit Server VM (build 14.2-b01, mixed mode) was used on the server-grade PC, while Java HotSpot(TM) Client VM (build 14.2-b01, mixed mode, sharing) was used on the client-grade laptop. The same MATLAB version was used, that is R2009b.

The time needed to parse RINEX files and the actual computation time employed by the core algorithms were checked separately. The core algorithms roughly include:

- Bancroft positioning (executed once)
- Code double difference positioning (executed twice)
- Satellite position computation (with clock and Earth rotation corrections) (at each epoch)
- Satellite topocentric coordinates computation (at each epoch)
- Kalman filter matrices setup (with computation of parameters obtained from linearized observation equations and observation weighting) (at each epoch)
- Ambiguity estimation and cycle slip check (at each epoch)
- Kalman filter computation (at each epoch)

For the moment goGPS Java does not employ any multi-threaded process, so everything is computed consecutively both in MATLAB and Java.

The test dataset was a ~5 hour survey (17205 epochs) with a number of satellites ranging from 6 to 7 (satellites effectively used in the computation after applying a cutoff of 15 degrees on their elevation).

Table 3 reports the resulting elapsed times (total time in seconds, average single epoch time in milliseconds). Total times are rounded to the nearest second and epoch times are rounded to one decimal.

Table 3. Results of computation speed

		Core i7 Linux		Pentium M Linux		Pentium M Windows	
RINEX parsing	MATLAB	520 sec	30.2 msec	4640 sec	270.7 msec	4070 sec	236.6 msec
	Java	4 sec	0.2 msec	10 sec	0.6 msec	17 sec	1.0 msec
Core algorithms processing	MATLAB	263 sec	15.3 msec	576 sec	33.5 msec	355 sec	20.6 msec
	Java	9 sec	0.5 msec	57 sec	3.3 msec	54 sec	3.1 msec

It is clear that porting goGPS MATLAB code to Java made a significant improvement in performance (as it was expected), especially for parsing RINEX files (i.e. text files). As for the core processing, goGPS MATLAB code was partially at a disadvantage because it is not object-oriented and some operations had to be repeated more often than in Java code.

An additional remark has to be made about the use of EJML in goGPS. EJML provides two basic approaches to define and work with matrices: the DenseMatrix64F class, which is oriented to speed and memory optimization, and the SimpleMatrix class, which is a wrapper around DenseMatrix64F that provides an easier but more limited and slow way to perform matrix operations (Abeles, 2010c). When goGPS Java was started, the SimpleMatrix class was used because it was easy and straightforward, besides providing more readable code. All the tests above were run with this class, but an additional test was done by switching to DenseMatrix64F class the most processing intense operations (Kalman filter and roto-translations of coordinates). An overall improvement of about 10 percent in speed was noticed. Future developments will include more in-depth optimizations.

3. goGPS processing as an OGC-compliant web service

In recent information technology developments, there are remarkable trends that shift software processing from local computation units to remote ones in order to provide web services such as Software as a Service (SaaS) and Platform as a Service (PaaS). Cloud computing is the latest effort in delivering computing resources as a service: it represents a shift away from computing as a product that is purchased, to computing as a service that is delivered to consumers over the Internet from large-scale data centers – or “clouds” (Sriram and Khajeh-Hosseini, 2010). Web services are based on standard HTTP communication, so that the user does not need to install any special software; service requests can be done directly through a browser, making web services straightforward and easy to use. Moreover, multiple services can be connected or merged in what are called "mashups". For example, Yoshida et al., 2009 and 2010 developed a web-based POI (Points of Interest) management system which can import, manage and display GPS track logs together with POIs such as geo-referenced photos and videos. The system connects Flickr's photo album with a GPS track log by implementing Flickr's web Application Programming Interface (API) (http://www.flickr.com/services/api/). And it also provides its functionality as a web service. These open web API help reducing redundant work and development costs as the low-cost model of high availability in the information system field becomes mainstream. Obviously, these implementations require interoperability, which is pursued by the definition and application of standards. In the geospatial information field, ISO and Open Geospatial Consortium (OGC) promote standardization of data formats and services at international level. By complying to these standards, various applications can handle common data and processes and therefore reduce data and software development costs, besides contributing to the distributed processing network. These specifications do not need royalties and are widely used especially in public sectors around the world. The main database engines and GIS software packages already comply with open standards and this enables easy data connection and sharing without the need of generating new data, which reduces significantly costs and efforts. This leads also to a high level of interconnectivity, reduction of time of computation and it can guarantee the perpetuity of data and services.

goGPS processing will be provided as a standard Web Processing Service (WPS) by subsequent development steps, some of which have already been implemented. goGPS MATLAB code was used in a remote processing architecture, which was implemented and tested by using Java socket client and server programs that transmit raw data between a client GPS receiver and a server running goGPS MATLAB (Realini et al., 2010). The next step involves running goGPS Java code as a standard service using one of the available open source WPS platforms, specifically the ZOO framework (http://www.zoo-project.org/).

3.1 SoC, host-based, remote processing

There are trends in GPS receivers to shift from hardware-based to software-based processing, in other words shifting from System-on-Chip (SoC) architectures to host-based ones, where the host CPU can be exploited to perform as much computation as possible (Van Diggelen, 2009). Research and development efforts are being carried out to make devices as simple, small and low energy consuming as possible: for example, leaving only basic hardware parts in GPS such as antenna and signal tracking components for receiving satellite data and do the positioning processing as software running on the host machine (Söderholm, 2008). Further developments can increase miniaturization and lowering costs by switching to software also the signal acquisition and tracking component (Borre, 2007; Solé-Gaset, 2009). goGPS has been developed since the beginning with host-based processing in mind, also to allow web-based processing as a service. For instance, GPS rovers with Internet connection can send GPS raw data to goGPS server through socket communication and the server can process positioning in real-time. If no Internet connection is available, rovers can log locally the raw data in order to send them at a later time for post-processing. As a result, the rover can be a low-cost and low power consuming device, only a little more than a logger, because the advanced processing will not be required on the rover terminal. Future developments of this work will focus on building a low-cost prototype that allows to get highly accurate location information at several centimeters level through a goGPS server available on a network if it is able to acquire GPS raw data (code and phase measurements) from a hardware-based or software-based receiver.

3.2 Real-time data stream through socket communication

Basic parts of network based real-time positioning were implemented in goGPS Java. Socket programs were developed (Yoshida and Raghavan, 2008) and improved in order to carry out initial tests of network-based remote processing between a GPS rover and a server running goGPS. A socket is a kind of network address, a combination of IP address and a port number in a network computer. Socket communications work between a socket server and a socket client programs. The socket server opens a certain port and receives the data with TCP or UDP connection. The client designates the port and IP address of the server and sends data to the socket server. Socket server and client programs were developed using Java. The Java client program enables a GPS rover to send raw data (in this example: u-blox binary stream) to the server running goGPS.

There were some limitations in previous version of this function in goGPS MATLAB. One is that MATLAB does not support multi-threaded processing. Therefore, goGPS could not accept multiple

GPS rover clients at same time. This means that the goGPS server would have processed only one client while other clients would have had to wait for the communication to finish. On the other hand, Java supports multi-threaded processing and this functionality was implemented in goGPS Java in order to let it process more than one GPS rover data stream at the same time.

The other limitation in MATLAB is the socket support. There are incompatibilities between MATLAB and the socket server program which was developed previously. In order to connect goGPS MATLAB and the socket server program it was needed to redirect the data stream to a local UDP port, which was then opened by MATLAB standard UDP functions. From a data integrity point of view, UDP is not a suitable communication protocol. On the contrary, goGPS Java can directly receive the data stream from the socket client through a reliable TCP socket communication. Subsequent development step will involve integrating the socket server program directly into goGPS Java code. Figure 2 shows the socket server architecture before and after porting goGPS to Java.

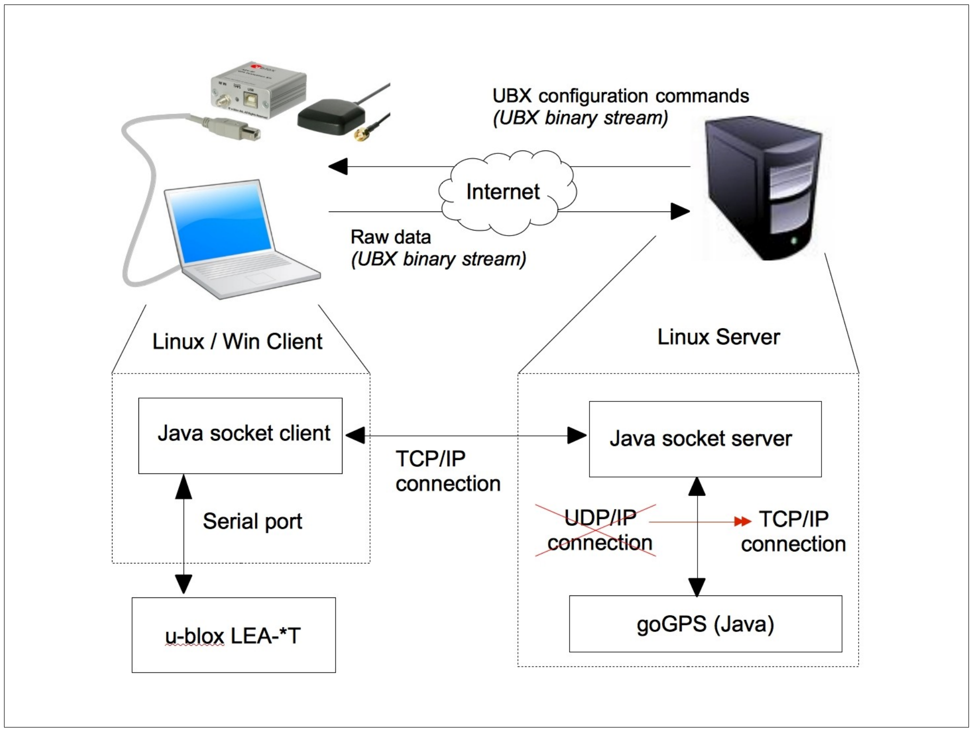


Figure 2. goGPS Java socket communication
(showing the switching from UDP to TCP connection after porting goGPS to Java)

goGPS network based real-time positioning was tested at a basic level for the purpose of this work: a u-blox AEK-4T was connected to a client laptop, providing data at 1 Hz; a Java socket client program was installed on the laptop; goGPS was installed on a Linux server, together with a Java socket server program. Once the connection was established between the socket client and server, u-blox binary data flew through the network and they reached goGPS. The transmission speed and the integrity of the received data were verified by making goGPS decode and display them in real time. This procedure was repeated both over a LAN network and over a mobile Internet connection. No data loss was experienced, but the data transmission was delayed slightly more when using the mobile connection, as it was expected.

3.3 OGC Web Processing Services

The OGC Web Processing Service (WPS) protocol defines a standardized interface that facilitates the publishing of geospatial processes, and the discovery of and binding to those processes by clients. "Processes" include any algorithm, calculation or model that operates on spatially referenced data. "Publishing" means making available machine-readable binding information as well as human-readable metadata that allows service discovery and use (The OGC Web Processing Service, 2007). Thus, WPS can standardize all GIS processes and calculations and provide them as web services via HTTP. The WPS interface specifies three operations that can be requested by a client and performed by a WPS server:

- GetCapabilities – This operation allows a client to request and receive back service metadata documents that describe the abilities of the specific server implementation. The GetCapabilities operation provides the names and general descriptions of each of the processes offered by a WPS instance.
- DescribeProcess – This operation allows a client to request and receive back detailed information about the processes that can be run on the service instance, including the inputs required, their allowable formats, and the outputs that can be produced.
- Execute – This operation allows a client to run a specified process implemented by the WPS, using provided input parameter values and returning the outputs produced.

3.4 WPS implementation using the ZOO Kernel

The ZOO project is an open source project for providing an open WPS platform. The ZOO Kernel is the core of the ZOO Project: it is a server-side C kernel which makes it possible to create, manage and chain WPS 1.0.0 compliant web services, by loading required dynamic libraries and handling them on-demand. Thus, it can easily connect to geospatial libraries and scientific models, but also with the common cartographic engines and spatial databases (Fenoy et al., 2009 and 2010).

The ZOO Kernel supports various programming languages (C, Python, Java, JavaScript, Fortran and PHP). This multi-language support is convenient for developers and allows, above all, to use existing code to create new web services. Open source GIS libraries or specific code (spatial based or not) can thus be ported server-side with very little modifications and also make them available as open processing services for various applications over the network. ZOO is one of the latest open source software projects which has attracted attention as an engine for WPS in the geospatial community.

In this research, a basic process of goGPS Java was implemented as WPS using the ZOO Kernel. The implemented process works for reading RINEX files, performing positioning by Kalman filter on

code and phase and exporting to KML.
The input parameters are provided via an URL as below:

*http://localhost/cgi-bin/zoo_loader.cgi?metapath=&
ServiceProvider=zgoGPS&Service=WPS&Request=Execute&Version=1.0.0&Identifier=go
GPS&DataInputs=Obs=perim2.08o;Nav=COMO1190.08n;mObs=COMO1190.08o*

Three RINEX files are used as input datasets: an observation file containing raw data for the GPS rover (perim2.08o), a navigation file containing satellite ephemerides and ionosphere parameters (COMO1190.08n) and an observation file containing raw data for the master station (COMO1190.08o).

Figure 3 illustrates on overview of ZOO WPS server and client communication. The client can be a web browser or a WPS-enabled GIS application such as uDig (http://udig.refractions.net/) and it must send the WPS request parameters (as the URL above) through standard HTTP protocol. The ZOO WPS server receives the request, it carries out the goGPS processing and finally returns the WPS response as the XML document shown in Figure 4.

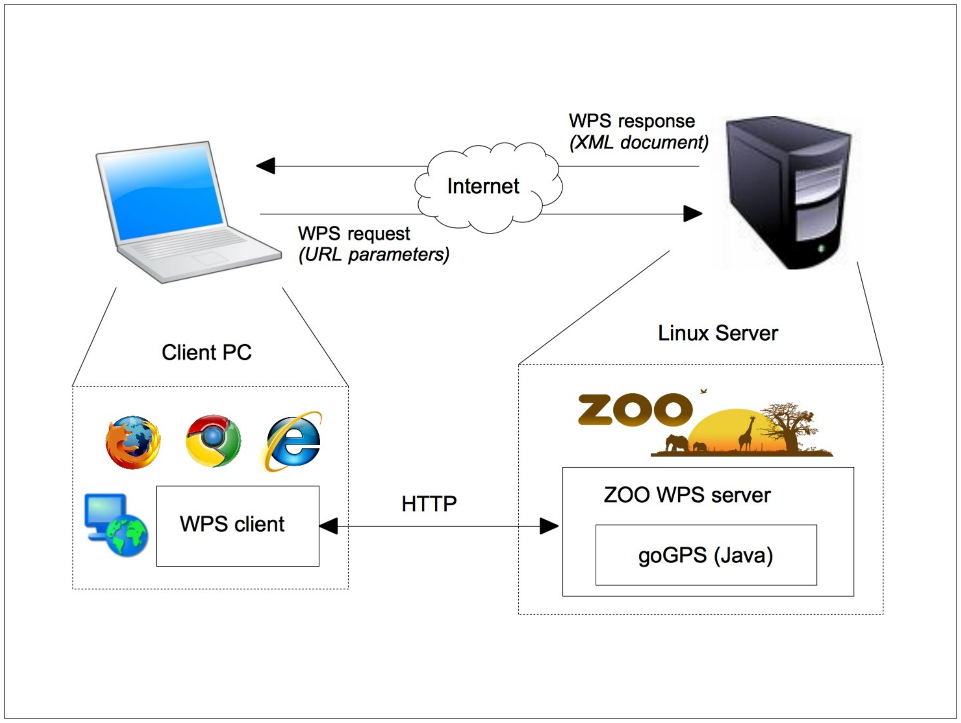
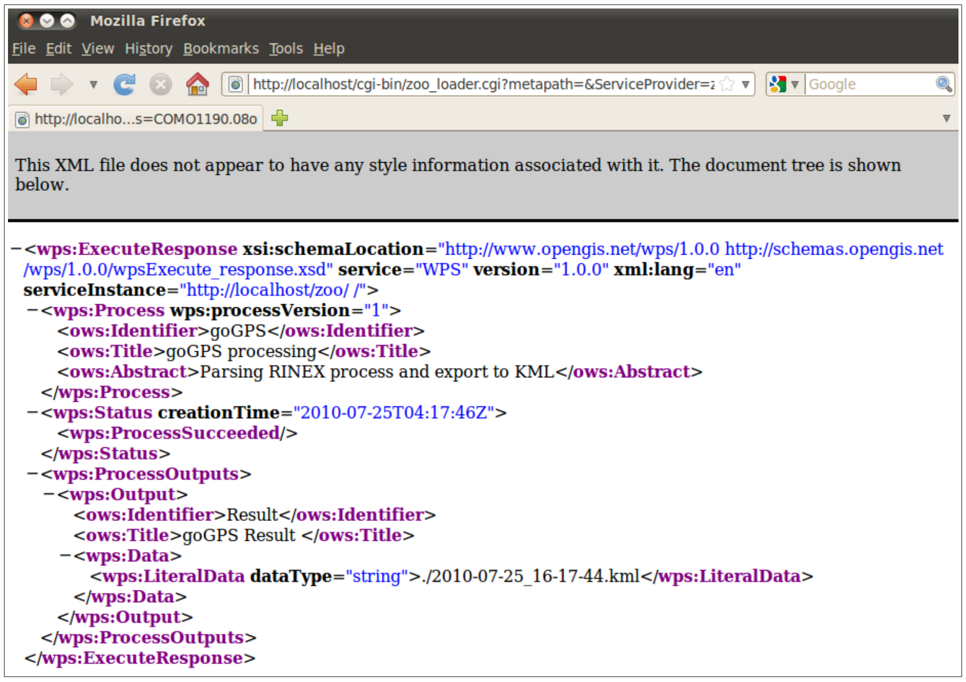


Figure 3. ZOO WPS server and client communication


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/cgi-bin/zoo_loader.cgi?metapath=&ServiceProvider=
Google
http://localho...s=COMO1190.080

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<wps:ExecuteResponse xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net
 /wps/1.0.0/wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en"
 serviceInstance="http://localhost/zoo/ /">
  -<wps:Process wps:processVersion="1">
     <ows:Identifier>goGPS</ows:Identifier>
     <ows:Title>goGPS processing</ows:Title>
     <ows:Abstract>Parsing RINEX process and export to KML</ows:Abstract>
   </wps:Process>
  -<wps:Status creationTime="2010-07-25T04:17:46Z">
     <wps:ProcessSucceeded/>
   </wps:Status>
  -<wps:ProcessOutputs>
    -<wps:Output>
       <ows:Identifier>Result</ows:Identifier>
       <ows:Title>goGPS Result </ows:Title>
      -<wps:Data>
         <wps:LiteralData dataType="string">./2010-07-25_16-17-44.kml</wps:LiteralData>
       </wps:Data>
     </wps:Output>
   </wps:ProcessOutputs>
 </wps:ExecuteResponse>
```

Figure 4. goGPS WPS result

For further research and development, several processes of goGPS Java are planned to be implemented as WPS in order to provide various web services for GPS positioning, for example:

– RTK processing by goGPS Kalman filter (as in the example above)
– Epoch-by-epoch least squares processing
– Kalman filter (on coordinates)
– Line Simplification (e.g. Douglas-Peucker algorithm)
– RINEX file generation from binary data (RTCM, UBX, etc.)
– GPS-related data format conversion (KML, NMEA, GPX, etc.)

These implementations will allow general users to use goGPS precise positioning in their web browser or their own applications since the goGPS web services are going to be open services compliant with WPS open standard.

4. Conclusions

This research described the performance results for several comparison tests between goGPS Java and MATLAB versions. The result clearly showed the differences in performance between the two and indicated that the Java version of goGPS can work much faster than the MATLAB version. The performance test also included the evaluation of different Java matrix libraries for efficient matrix computation.

Porting goGPS to Java code did not only contributed to its performance, but it enabled goGPS to handle network based real-time positioning, which had limitations using MATLAB, in a more robust and efficient way. Moreover, Java porting enabled goGPS processing to be provided as an OGC-compliant Web Processing Service (WPS). In this research, the open WPS platform ZOO was adopted as WPS server and goGPS positioning process was implemented using the ZOO Kernel. The ZOO server successfully provided goGPS processing as a web service.

Further developments include: enabling multi-threaded processing in goGPS Java code; optimizing the processing speed, with particular attention to matrix computation; developing a graphical user interface; developing real-time positioning for more than one rover client by accepting real-time data inputs from multiple rovers at the same time. Concerning goGPS WPS implementation, further goGPS processes are going to be implemented as WPS in order to provide a fully-featured web platform for GPS positioning. Web interfaces for goGPS server are also planned to be developed.

5. Acknowledgments

The authors acknowledge the positioning services IREALP GPSLombardia (Italy) and JENOBA (Japan) for their support. The first author acknowledges the Japan Society for the Promotion of Science (JSPS) for the post-doctoral fellowship under which this work was carried out. The authors would also like to thank the developer of EJML library, Peter Abeles, for his help during the matrix computation performance study and Gerald Fenoy for his helpful support in integrating goGPS into the ZOO Kernel. This research is supported by JSPS Grant-in-Aid for Scientific Research (Issue No. 2109737) entitled "Development of Ubiquitous LBS Web-Service using Free and Open Source Software".

6. References

Abeles, P 2010a, java-matrix-benchmark, viewed 28 July 2010, <http://code.google.com/p/java-matrix-benchmark/>.

Abeles, P 2010b, efficient-java-matrix-library, viewed 28 July 2010, <http://code.google.com/p/efficient-java-matrix-library/>.

Abeles, P 2010c, *SpeedSimpleMatrix – SimpleMatrix runtime performance study*, viewed 26 July 2010, <http://code.google.com/p/efficient-java-matrix-library/wiki/SpeedSimpleMatrix>.

Borre, K 2003, 'The Easy Suite - Matlab code for the GPS newcomer'. *GPS Solutions*, 7(1), pp 47-51.

Borre, K 2007, A Software-defined GPS and Galileo Receiver – A Single-frequency Approach, Birkhäuser, Boston.

Borre, K 2010, The EASY Suite, viewed 28 July 2010, <http://kom.aau.dk/~borre/easy/>.

Fenoy, G 2009, *ZOO project : an open WPS Platform*, International Conference FOSS4G2009, Sydney, Australia, viewed 28 July 2010, <http://2009.foss4g.org/presentations/>.

Fenoy, G, Bozon, N, Raghavan, V 2010, 'ZOO project: The Open WPS Platform', submitted to *WebMGS 2010*, Como, Italy.

Hofmann-Wellenhof, B, Legat, K, Wieser, M 2003, Navigation – Principles of Positioning and Guidance, Springer Wien New York.

Kalman, RE 1960,' A New Approach to Linear Filtering and Prediction Problems'. Transaction of the ASME - Journal of Basic Engineering, pp. 35-45.

Realini, E 2009, *goGPS - free and constrained relative kinematic positioning with low cost receivers*, Ph.D. Thesis, viewed 27 July 2010 <http://geomatica.como.polimi.it/corsi/tesi/E_Realini.tar>.

Realini, E, Yoshida, D, Reguzzoni, M, Raghavan, V 2010, 'Testing goGPS Low-cost RTK Positioning with a Web-based Track Log Management System', submitted to *WebMGS 2010*, Como, Italy.

Söderholm, S, Jokitalo, T, Kaisti, K, Kuusniemi, H, Naukkarinen H 2008, *Smart Positioning with Fastrax's Software GPS Receiver Solution*, ION GNSS 2008, Savannah, Georgia, USA. viewed 24 May 2010, <http://www.fastraxgps.com/products/softwaregps/>.

Solé–Gaset, M, Arribas, J, Fernández-Prades, C, Closas, P 2009, *GNSS Open Source Software Receiver*, NEWCOM++, ACoRN Joint Workshop, Barcelona, Spain. <http://www.cttc.es/resources/doc/090608-gnssdemo-64145.pdf>.

Sriram, I, Khajeh-Hosseini, A 2010, *Research Agenda in Cloud Technologies*, 1st ACM Symposium on Cloud Computing (SOCC 2010), viewed 27 July 2010, <http://arxiv.org/pdf/1001.3259>.

The OGC Web Processing Service 2007, Open Geospatial Consortium, viewed 27 July 2010, <http://www.opengeospatial.org/standards/wps>.

Van Diggelen, F 2009. 'The Smartphone Revolution'. *GPS World*, December 2009, viewed 24 May 2010, <http://www.gpsworld.com/wireless/smartphone-revolution-9183>.

Yoshida, D, Raghavan, V 2008, *Development of Real-time Tracking and Log Management Prototype System using Free and Open Source Software*, International Conference FOSS4G2009, Capetown, South Africa, viewed 28 July 2010, <http://www.foss4g.org/index.php/foss4g/2008/paper/view/138>.

Yoshida, D, Song, X, Raghavan, V 2009, *Development of Track Log POI Management System using Free and Open Source Software*, International Conference FOSS4G2009, Sydney, Australia, viewed 28 July 2010, <http://2009.foss4g.org/researchpapers/>.

Yoshida, D, Song, X, Raghavan, V 2010, 'Development of Track Log POI Management System using Free and Open Source Software', *Applied Geomatics*, vol. 2, no. 3, in print.