GNSS-SDR, un receptor definido por software abierto

Dr. Carles Fernández Prades
Centre Tecnològic de Telecomunicacions de Catalunya
Motivation
• Location has become an **embedded feature** in electronic devices (phones, digital cameras, portable gaming consoles).

• This massive deployment of GNSS receivers requires a high level of integration, low cost, small size and low power consumption. GPS integrated circuit (IC) manufacturers offer **single-chip solutions** easy to integrate in multi-function devices.

• This approach is very convenient for location based services and applications, since users and developers are interested in *using* the location information but not in *how* the position has been obtained.

• Example: Android’s API. It provides a location package that contains classes with methods such as `getLatitude()`, `getLongitude()`, `getAltitude()`, `getSpeed()`, `getAccuracy()`, and so on.
• Location has become an **embedded feature** in electronic devices (phones, digital cameras, portable gaming consoles).

• This massive deployment of GNSS receivers requires a high level of integration, low cost, small size and low power consumption. GPS integrated circuit (IC) manufacturers offer **single-chip solutions** easy to integrate in multi-function devices.

• This approach is very convenient for location-based services and applications. Users and developers are interested in using the location information, but not in how the position has been obtained.

• Example: Android’s API. It provides a location package that contains classes with methods such as `getLatitude()`, `getLongitude()`, `getAltitude()`, `getSpeed()`, `getAccuracy()` and so on.
• Location has become an **embedded feature** in electronic devices (phones, digital cameras, portable gaming consoles).

• This massive deployment of GNSS receivers requires a high level of integration, low cost, small size and low power consumption. GPS integrated circuit (IC) manufacturers offer single-chip solutions easy to integrate in multi-function devices.

• This approach is very convenient for location-based services and applications, since users and developers are interested in using the location information but not in how the position has been obtained.

• Example: Android’s API. It provides a location package that contains classes with methods such as getLatitude(), getLongitude(), getAltitude(), getSpeed(), getAccuracy() and so on.
What about the pros?

• Professional GNSS receivers feature dual-frequency reception and wireless connection for retrieving differential data (i.e., *corrections*) from reference stations in order to provide high-accuracy (cm-level) positioning.

• Receivers are *expensive* (5-10 k€), correction messages are *proprietary* (and with 1.5-5 k€ annual fees).

• You still do *not know* what your receiver is actually doing.
Wish list for a GNSS software receiver
Introducing GNSS-SDR

• The acronym stands for ‘Global Navigation Satellite System – Software Defined Receiver’.
• It is a tool for researchers, specially focused in signal processing.
• Flexible, fully configurable, easily extendible.
• Testbed for GNSS synchronization algorithms.
• Programmed in C++. Portable (currently builds in Linux and MacOS operating systems. The execution in embedded processors FPGA/ARM is ongoing work)
• **Open Source**: free as in *free beer and as in free speech*. Released under GPLv3.
The Software Defined Radio concept

**Nice features w.r.t ASICS:**
- Operational flexibility.
- Upgradability.
- Compatibility.
- Lower maintenance costs.
- Lower operational costs.

**Unique features:**
- Fair benchmarking of architectures and algorithms.
- Possibility of re-running the receiver over the same signal but with different architectures/algorithms/parameters.
- Allows a “GNSS receiver in the cloud”.
- **If open**: you know what your receiver is doing.
Current features

- Multiplatform (Linux and Mac OS X, 32 and 64 bit architectures).
- Multithreaded.
- Works with files and several RF front ends, including those compatible with the Universal Hardware Driver (UHD).
- Acquisition of **GPS L1 C/A**, **Galileo E1B** and **E1C** signals.
- SIMD-enabled for most popular processors.
- Implemented tracking loops: DLL + PLL, DLL + PLL/FLL, VEML.
- Connection to Matlab/Simulink via TCP for rapid prototyping and algorithm validation.
- Demodulation and decoding of the navigation message GPS NAV and Galileo INAV.
- Computation of PVT (Position - Velocity - Time) solution in real-time.
- Position solution exportable to KML files (can be opened by Google Earth and other similar tools) and to a serial port via NMEA.
- Generation of RINEX files (observables and navigation), v2.1 and v3. RTCM 3.2 is ongoing work.
Philosophy
Design principles

• Do not reinvent the wheel.
• Follow design patterns.
• Follow a clean coding style.
• Test-driven development approach.
• Follow standards.
• Try to produce well-written software.
Do not reinvent the wheel

• On the shoulder of giants:

**GNU Radio** is a software development toolkit that provides a framework to implement software radios. GPL v3.

**FFTW** is a C subroutine library for computing the discrete Fourier transform. GPL v2.

**Boost** is one of the most highly regarded and expertly designed set of libraries for the C++ programming language. Boost Software License.

**Google C++ Testing Framework** provides tools for writing C++ tests. BSD 3-Clause License.

**Armadillo** is a C++ linear algebra library aiming towards a good balance between speed and ease of use. Mozilla Public License Version 2.0.
Design patterns

- Software design patterns are **descriptions of solutions to common software problems** arising in different contexts, capturing recurring structures and dynamics among software participants to facilitate reuse of successful, thoughtfully proven designs.

- They generally **codify expert knowledge of design strategies**, constraints and best practices. Following a pattern helps to resolve key design forces such as flexibility, extensibility, dependability, predictability, scalability, and efficiency.

- **They are not code recipes but generalized solutions** to commonly occurring problems.

---

**References**

- E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Upper Saddle River, NJ, 1995.

Coding style

• Following programming guidelines and code conventions not only helps to avoid introducing errors, but cuts maintenance costs and favors effective code reuse.

• The following rules capture the most important aspects of coding style:
  – All should be as **understandable** as possible.
  – All should be as **readable** as possible, except when it would conflict with the previous rule.
  – All should be as **simple** as possible, except when it would conflict with the previous rules.

• Any violation to the guide is allowed if it enhances readability.

http://gnss-sdr.org/documentation/coding-style
Test-driven development

Test–driven development:

- facilitates change,
- simplifies integration,
- automatizes documentation,
- helps to separate the interface from the implementation,
- increases developers productivity, and
- plays a central role in the software quality assurance process.
For unit testing, we found the Google C++ Testing Framework (gtest) useful and lightweight.
Software architecture
Adding new algorithms

- I have designed an awesome acquisition algorithm and I want to test it in the framework of GNSS-SDR. What I have to do?
  - Create a GNU Radio block, derived from gr::block, implementing your algorithm (.cc and .h files) and putting it in *src/algorithms/acquisition/gnuradio_blocks/*
  - Create the adapter of such new block to AcquisitionInterface (.cc and .h files) and putting it in *src/algorithms/acquisition/adapters/*
  - Tell the system that a new block is available and include it in the ‘production line’: add the instantiation of your new block in *src/core/receiver/GNSSBlockFactory.cc*
  - Update the **CMakeLists.txt** files accordingly to tell the compiler that a new block exists.

Easy, examples are available
• Combined with certain DVB-T USB dongles, it constitutes the lowest cost solution available for GNSS software radio:

![Image of a GPS antenna, Bias tee, and DVB-T USB dongle with a combined cost of less than 100 €]


• First open source solution to achieve Galileo-based position fixes (Nov. 2013).

![Image of a GPS antenna and software receiver setup]

Development ecosystem
Infrastructure for project management, code development and efficient communication among users and developers is a key aspect in software projects.

- The **website** has been designed in terms of usability, functionality and extendability, ensuring an enjoyable and appealing user experience.
- **Public mailing list** for users and developers.
- For the source code, we used the service provided by SourceForge, which allows access to the code using a revision control system.
- **Documentation** is of paramount importance for users, developers, testers, software architects and students. We used Doxygen (HTML, LATEX, RTF or XML output formats).
- The project web page provides with detailed instructions about the installation, usage, **coding style** and general information about the software.
• The source code is freely accessible through Subversion, an open source version control system.
• Both the source code repository and the public mailing list are hosted at Sourceforge.
• The build process should be easily maintained and highly portable. We use CMake, a tool that takes care about compiling the sources with the right options.
• Regarding the compiler, the GNU Compiler Collection and clang (BSD license) can be used.
• Logging is important for debugging and tracing purposes. In GNSS-SDR, logging is handled by the Logging Library for C++ (google-glog), a library that implements application-level logging.
Conclusions
• The proposed software receiver targets multi-constellation / multi-frequency architectures, and provides code and phase observables in standard formats, enabling RTK and Precise Point Positioning techniques.

• Goals: efficiency, modularity, interoperability and flexibility demanded by user domains that require non-standard features.

• The source code was released under the GNU General Public License (GPL), thus ensuring the freedom of modifying, sharing, and using the code for any purpose.

• This secures practical usability, inspection, and continuous improvement by the research community, allowing the discussion based on tangible code and the analysis of results obtained with real signals.

• It is also intended to be a framework for algorithm testing and an educational tool.
Publications

• C. Fernández-Prades, J. Arribas, P. Closas, "Turning a Television into a GNSS Receiver", in Proc. of the ION GNSS+ 2013 Conference, Nashville, Tennessee (USA), 15-16 Sept. 2013.


Available at http://gnss-sdr.org/documentation/publications
Thank you!

Please visit [http://gnss-sdr.org](http://gnss-sdr.org) and find:

- An overview about the project.
- Instructions for downloading and building the source code.
- Documentation and tutorials.
- Links to the mailing list.
- Suggestions for participation.