GNSS Software Receiver for UAVs

Olesen, Daniel Haugård; Jakobsen, Jakob; von Benzon, Hans-Henrik; Knudsen, Per

Published in:
European Journal of Navigation

Publication date:
2016

Document Version
Peer reviewed version

GNSS Software Receiver for UAVs

Daniel Olesen, Jakob Jakobsen, Hans-Henrik Benzon, Per Knudsen, *DTU Space*

ABSTRACT

This paper describes the current activities of GPS/GNSS Software receiver development at DTU Space. GNSS Software receivers have received a great deal of attention in the last two decades and numerous implementations have already been presented. DTU Space has just recently started development of our own GNSS software-receiver targeted for mini UAV applications, and we will in in this paper present our current progress and briefly discuss the benefits of Software Receivers in relation to our research interests.

INTRODUCTION

The history of software receivers can be dated back to the early 1990s, where the US Department of Defense started the Speakeasy project. The project originated from an increasing challenge to ensure communication with current allies, avoid hostile interception and taking advantage of rapid technology changes [1]. Prior to Speakeasy, military radio systems was developed with a 30 year lifespan to support one function and optimized for a particular field application [1].

The effects of this paradigm change for radio communication have since triggered a growing interest for use of software radios in commercial and academic applications, as software radios allows for great flexibility, offering wide support for various modulation schemes, bandwidths and baseband algorithms. In terms of GNSS receivers, reported software implementations have steadily grown in numbers and capabilities over the years.

In contrast to commercial GNSS receivers, software implementations offer the ultimate level of control for the processing stage. The added control and access to parameters at the tracking loop level, allows for research in more advanced applications such as e.g. Ultra-Tight/Deep Integration of GNSS and Inertial Navigation Systems (INS). Another application, where a software receiver is useful is within Space Weather monitoring. Here the two scintillation indices S_4 and σ_φ are traditionally used as a measure for the scintillations caused by the ionosphere. Direct access to the correlators and the signal filtering gives control over the quality of these parameters, see [2]. This flexibility and low level access to the hardware is also very useful in other research areas such as multipath mitigation and GNSS based reflectometry.

In this paper, two development projects related to GNSS software receivers are described. The first project is a

Figure 1: Hexacopter used for experimental testing

small, portable GNSS sampler based on the low-cost Beagle Bone Black Single Board Computer (SBC) and the commercial GNSS RF Front-end, MAX2769. The advantages of this system are low cost, size and flexibility. The sampler has been developed with a distinct focus on portability to ensure it can be used in small UAV applications.

In order to process the raw GNSS samples, a software receiver is needed. We have initially used the MATLAB based SoftGNSS [3]. This has been an excellent educational tool and a good reference. However as our investment in software receivers have grown, we decided to start working on our own implementation in C++ using an event-driven object-oriented approach. The implementation is aimed at post-processing and not real-time capabilities. One of the design goals has been on modularity, such that the system easily can be expanded to include more advanced features and adapted to various applications.

In addition to the projects above, the authors have also worked on a small, low-cost, real-time embedded GPS/GNSS SDR. The design is based on a small low-cost parallel computing platform known as the Parallella. Preliminary results for this work and Proof-Of-Concept have been published in [4].

BASIC OPERATION OF A GNSS RECEIVER

In this section a brief introduction to the operation of a GNSS receiver is given. For simplicity, only reception and processing of the GPS L1 C/A code will be described.

The GPS Coarse / Acquisition (C/A) code is a 1023 chip Pseudo-Random Noise (PRN) sequence transmitted on the GPS L1 band with a carrier frequency of 1575.42 MHz. The C/A code is modulated on to the carrier by Binary Phase Shift Keying (BPSK). As all GPS satellites are using the same carrier-frequency, a Code Division

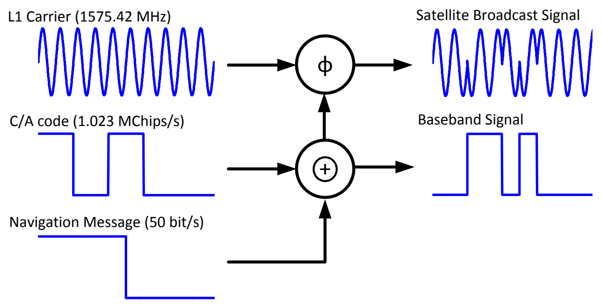
L1 Carrier (1575.42 MHz)
Satellite Broadcast Signal
C/A code (1.023 MChips/s)
Baseband Signal
Navigation Message (50 bit/s)
ϕ
⊕

Figure 2: Simplified illustration of C/A code and Navigation Message modulation for GPS L1.

Multiple Access (CDMA) coding scheme is applied on the C/A code. The C/A code is for each satellite a unique PRN code known as a Gold code. Gold codes have a bounded minimum cross-correlation with other Gold codes and hence the satellite transmitting the signal can be identified by correlation with a receiver generated replica.

The chiprate of the C/A code is 1.023 Mchips/s, corresponding to a code-sequence length of 1 ms. The C/A code is modulated with the satellite navigation message, which contains the ephemeris, time etc. for the satellite. The signal structure for the GPS L1 C/A signal is visualized in Figure 2.

Mathematically the broadcast signal is defined as:

$$S_{SV}(t) = \sin(2\cdot\pi\cdot f_{carr}\cdot t)\cdot\left(S_{C/A}(t)\oplus S_{Nav}(t)\right) \tag{1}$$

where $\oplus$ is modulo-2 addition, also commonly referred as an XOR operation. $S_{C/A}(t)$ is the unique gold code for the satellite and $S_{Nav}(t)$ is the satellite navigation message. We here assume binary levels of the C/A code and navigation message are $[-1;1]$.

At the reception side, the basic blocks of operation for a GNSS receiver can be seen in Figure 3. The Antenna receives the broadcasted signals from the visible GNSS satellites. An RF-Front end amplifies, filters and down-converts (mix) the received signals down to an Intermediate Frequency (IF). After the mixing the signal is normally processed by an Automatic Gain Control (AGC) circuit, which adjusts the signal amplitude before AD conversion.

The receiver then performs an operation known as Satellite Acquisition. This operation determines which satellites are in view and provides estimates of code-delay and Doppler frequencies for the satellites. The acquisition process can be thought of as a parametric sweep through combinations of possible Doppler frequencies and code-delays for the C/A PRN sequences used by the GPS satellites. Each combination of code-delay and Doppler is then correlated with the received signal and tested against a threshold.

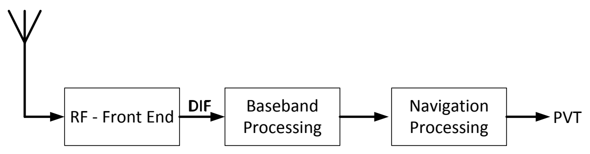

Figure 3: Generic GNSS Receiver Structure.

After acquisition, the receiver starts tracking the detected satellites and performs demodulation in order to extract the navigation messages from the satellites. The tracking of each of the visible satellites are typically performed by a variant of an Phase-Locked Loop (PLL) for tracking the Doppler on the down-mixed carrier (IF) and a Delay-Locked Loop (DLL) for aligning the receiver generated C/A-code with the incoming signal. A basic implementation of satellite tracking for one channel is shown in Figure 4. The tracking loop consists of six correlators, three for the In-phase (I) and Quadrature (Q) arms respectively. The correlators for each arm are typically separated with a lag of 0.5 chip between them. This lag allows the DLL to accurately keep hold of the C/A code alignment, as under perfect lock the energy in the prompt-output is twice the early and late versions. When the tracking loop has sufficient lock on a particular satellite, the navigation message can be extracted from the In-phase Prompt correlator output. For more information on satellite tracking see [3], [5] and [6].

Both Acquisition and Tracking are part of the Baseband processing block in Figure 3. In most commercial receivers the baseband processing algorithms is typically implemented in parallel using digital hardware on either Application Specific Integrated Circuits (ASIC) or on Field-Programmable Gate Arrays (FPGA). In a software-receiver the baseband processing algorithms are implemented in code.

Finally, the Navigation Processing calculates user position, velocity and time using the decoded navigation-message.

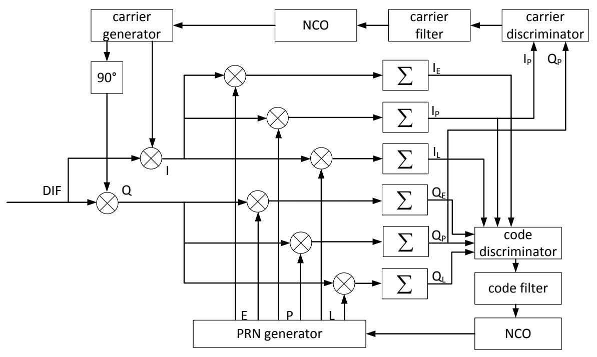

Figure 4: Tracking Loop for one channel.

GNSS SAMPLER

As explained in the introduction one of our goals is to collect raw GNSS samples from small, lightweight UAV's. This requirement naturally put some constraints on the allowed weight and size of the system. To the best of our knowledge the vast majority of commercial GNSS samplers designed for software-receivers are implemented with either USB or Ethernet interfaces and hence requires a host-computer for either real-time processing or data storage.

Due to the above considerations and a desire to ensure the system was as flexible as possible, we decided to use some effort in designing our own GNSS sampler.

The RF front-end of our GNSS sampler/data-collector has been chosen to be the MAX2769 from Maxim Integrated [7]. Incorporated in the chip is a complete RF processing chain which can be configured for GPS L1, GLONASS G1 and Galileo E1 reception. The front-end features a programmable single-conversion stage and supports both active and passive antennas. The component is available as an evaluation kit (EV-kit) equipped with a number of SMA connectors for evaluation of separate stages of the front-end circuitry. The EV-kit also features an input for use of an external oscillator, which makes it possible to evaluate the system using different grades of reference oscillators.

The local oscillator for the mixing-stage can be programmed to any given frequency in the range 1550-1610 MHz. In our application we have chosen to use a low side injection oscillation frequency of $F_{OSC} = 1571.328\,MHz$. This gives an IF Frequency of $F_{IF} = F_{RF} - F_{OSC} = 1575.42 - 1571.328 = 4.092\,MHz$. The IF bandpass filter can be programmed to have a bandwidth of 2.5 MHz, 4.2 MHz, 8 MHz and 18 MHz. In addition the filter can be implemented as a 3rd or 5th order polyphase filter. We have selected the bandwidth to 2.5 MHz using the 5th order filter setting. This effectively means that the system is configured to receive GPS L1 C/A code.

The MAX2769 chip includes an AD converter (ADC) with a configurable sample rate of up to 50 Msamples/s. The ADC has the ability to quantize the signal with up to 3 bits for real samples and (2+2) bit for I/Q sampling.

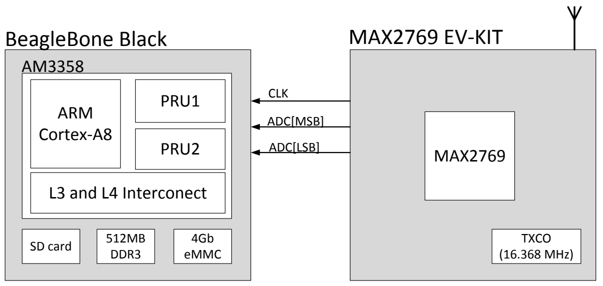

Figure 5: Block Diagram of GNSS sampler

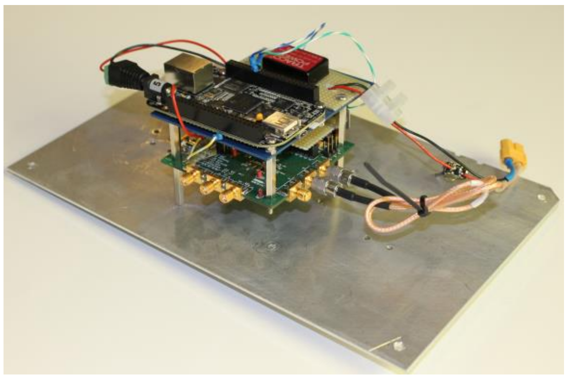

Figure 6: GNSS sampler mounted on 20 x 25 cm aluminium-plate (UAV payload). The MAX2769 EV-KIT is the bottom circuit with the BeagleBone Black and a power-supply board fitted on top.

In order to store the IF samples, a BeagleBone Black single-board computer (SBC) was used for data storage. This platform can be purchased for under $50 and has a number of hardware interfaces and GPIOs for external peripherals.

The transfer of the IF samples from the ADC is done using a parallel interface on the MAX2769 and three digital inputs on the BeagleBone Black. For our system we are using real sampling with 2 bits resolution and a sample-frequency of 16.368 MHz. A block diagram of the system is shown in Figure 5.

The BeagleBoard Black consists of a Texas Instruments Sitara AM335x Processor. This features a 1GHz ARM Cortex-A8. In addition the processor is also equipped with two Programmable Real-Time Units (PRUs) which runs with a clock-frequency of 200 MHz. The PRUs are especially useful in timing-critical applications, as they operate independently from the linux operating system on the ARM processor. In our design we have utilized one PRU to transfer data from the ADC on the MAX2769.

The PRU is configured to have access to an allocated portion of the system memory and stores the samples in that range. Two reception buffers are created to ensure continuous operation, such that one buffer can be filled while the other is being emptied. The PRU has been configured to generate an interrupt to the ARM processor, when a buffer is full. In this way we offload the ARM processor, as it only has to write the stored data onto a file on a SD card whenever an interrupt is triggered.

In Figure 6, the actual system mounted on a UAV payload plate is shown.

GNSS Software Receiver

As earlier stated we had initially used the MATLAB SDR implementation from [3], but as our interest and commitment in Software Receivers for GNSS has grown, it was decided to work towards making our own implementation from scratch. This choice was certainly not made due to a shortage of options, as there are a number of commercial and open-source implementations readily available. The reasoning for initiating our own project was primarily due to the educational value of working with all aspects first-hand and to design the implementation with future applications in mind.

In order to ensure high portability it was decided to make the implementation in C++, using a class-oriented and event-driven approach. The current state of the implementation is limited to processing of GPS L1 C/A code signals, but a roadmap to expand the support to GLONASS and Galileo signals are in-place. In this regard, our design philosophy is to make use of class-inheritance and polymorphism as signal processing of the different GNSS signals have a high degree of resemblance.

In Figure 7 an UML sequence diagram of the basic implementation is shown. A main thread is calling a satellite acquisition method which initially determines the satellites in view and provides estimates of Doppler and code-delay. The Satellite acquisition is based on the well-known Parallel Code Phase Search Algorithm, as shown in Figure 8. This algorithm effectively performs correlation in the frequency domain by using the cross-correlation theorem.

$$(f \star g)(x) \leftrightarrow \mathcal{F}^{-1}(F^{*}(u)G(u)) \qquad (2)$$

Here F^{*} and G are the Fourier Transforms of f and g.

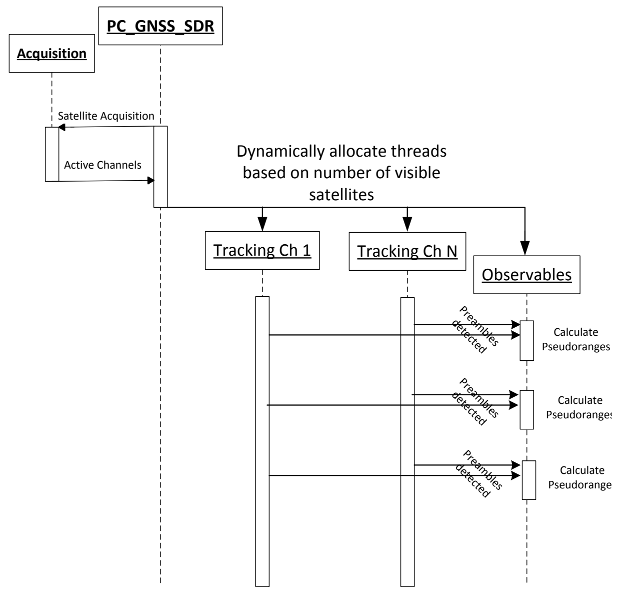

Figure 7: UML sequence diagram

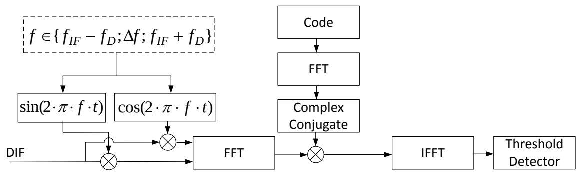
$f \in \{f_{IF} - f_D; \Delta f; f_{IF} + f_D\}$
Code
FFT
$\sin(2 \cdot \pi \cdot f \cdot t)$
$\cos(2 \cdot \pi \cdot f \cdot t)$
Complex Conjugate
DIF
FFT
IFFT
Threshold Detector

Figure 8: Parallel Code Phase Search

The implementation of the algorithm has been based on the FFTW library [8]. After the parallel-code phase search algorithm has completed, a number of objects are instantiated corresponding to each of the visible satellites. The tracking for each channel is started in separate threads. The tracking is essentially implemented according to Figure 4. In addition each of the tracking-threads implements a histogram detector for navigation-bit synchronization and estimates carrier-to-noise density estimate using the algorithms described in [9]. In parallel to the tracking-threads an event handling-class named Observables is instantiated and executed in a separate thread. This class is hooked up to events generated by the tracking-objects whenever a new preamble in the navigation message is detected. When all the active channels have detected the first preamble, pseudoranges are calculated based on the principle of Common Transmission Time [10].

The receiver implementation is not calculating a PVT solution on its own but merely produces raw observables and satellite ephemerides outputs to allow the computation afterwards. A script in MATLAB has been developed for subsequent tracking analysis and calculation of user position. The complete processing flow from data-capture to actual navigation solutions are shown in Figure 9. The GNSS sampler collects IF data and stores it in a binary file on a SD-card. The collected data is packaged with 4 samples per byte, in order to use space efficiently on the memory card. Before processing

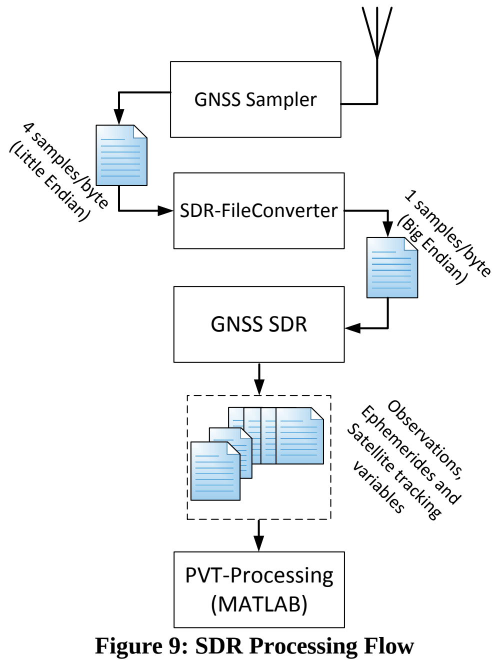
GNSS Sampler
4 samples/byte (Little Endian)
SDR-FileConverter
1 samples/byte (Big Endian)
GNSS SDR
Observations, Ephemerides and Satellite tracking variables
PVT-Processing (MATLAB)

Figure 9: SDR Processing Flow

of the data, a small stand-alone application converts the sampler format to 1 sample/byte and changes the byteordering of the file to Big Endian format. After this conversion the data can be processed by the software receiver. The receiver generates a file for pseudorange measurements and satellite ephemerides as well as multiple tracking files containing detailed information about correlator-values, C/N0 estimates, code-frequency and carrier-frequency for all the tracked satellites.

RESULTS

To verify the operation of the GNSS sampler and implemented software receiver, a static dataset was captured using a roof-antenna on top of the DTU Space building. The duration of this dataset was approximately 130 seconds. In Figure 10, the estimated receiver-position based on pseudoranges is shown. In Figure 11, the estimated C/N0 for the 8 satellites which was visible under the mission is shown. For the results presented below, it should be noted that only the broadcast ephemerides has been used for positioning, i.e. ionospheric corrections have not been applied.

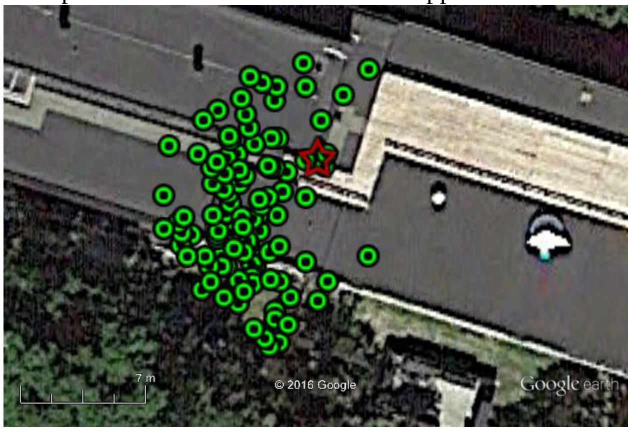

Figure 10: Google-Earth plot of Estimated Receiver position from Roof-antenna at DTU Space main building. The red star indicates the position of the GNSS antenna.

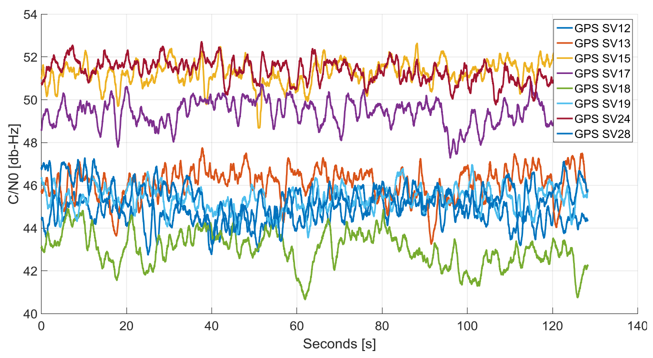

Figure 11: C/N0 density plots of tracked satellites during static test from Roof-antenna

σ_E	2.3360 m
σ_N	3.8757 m
σ_U	7.4598 m

Table 1: Std. Dev of ENU transformed coordinates

CONCLUSION AND OUTLOOK

In this paper, we have presented a small, flexible and low-cost solution for a simple GNSS sampler. The sampler was developed with a distinct focus on light-weight UAV applications and consist of commercial of the shelve components for simple integration.

In addition a software receiver implemented in C++ was presented. The implementation lays a foundation, in which we plan to build upon and add more functionality and refinement to in the future. There are certainly easier ways to get involved with software processing of GNSS signals than to build the entire systems from the bottom-up, but in this way we hope to get the most experience and know-how which we hope can be exploited in the future. Especially ensuring modularity of the solutions, have been a major concern as we wish to build our systems as generically as possible in order to make the system flexible with respect to adaptations to different applications. We see the involvement in software-receivers as a long term investment, as this tool-kit opens up for a variety of new applications and advanced research topics within GNSS-related research.

In terms of future developments and outlook, our next project is to work on a vector-based software receiver which can be utilized for Ultra-Tightly coupled GNSS and INS integration. In addition, we also plan to explore the benefits of Software Receivers in relation to Space Weather applications.

REFERENCES

[1] R. Lackley and D. Upmal, "Speakeasy: the Military Software Radio," *IEEE Communication Magazine,* pp. 56-61, 1995.

[2] J. Curran, M. Bavaro, J. Fortuny and A. Morrison, "Developing an Ionospheric Scintillation Monitoring Receiver," *InsideGNSS,* no. September/October, 2014.

[3] K. Borre, D. Akos, N. Bertelsen, P. Rinder and S. Jensen, A Software-Defined GPS and Galileo Receiver, Birkhauser, 2007.

[4] D. Olesen, J. Jakobsen and P. Knudsen, "Software-Defined GPS Receiver Implemented on the Parallella-16 board," in *Proceedings of ION GNSS+*, Tampa, Florida, 2015.

[5] E. D. Kaplan and C. J. Hegarty, Understanding GPS - Principles and applications, Artech House, 2006.

[6] J. B. Tsui, Fundamentals of Global Positioning System Receivers, John Wiley & Sons, 2000.

[7] Maxim Integrated, "MAX2769 Datasheet," San Jose, 2010.

[8] M. Frigo and S. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE, Vol 93, Issue 2,* pp. 216-231, 2005.

[9] A. J. V. Dierendonck, "GPS Receivers," in *"Global*

Positioning System: Theory and Applications", Volume I, Edited by B.W. Parkinson, J.J. Spiker Jr.

[10] M. Rao and G. Falco, "Code Tracking and Pseudoranges," *InsideGNSS*, no. January/February, 2012.