PROTOTYPE OF IOT GNSS SENSOR FOR CLOUD GNSS SIGNAL PROCESSING

L.M. Romero-Holguín [(1)], V. Lucas-Sabola [(1)], J.A. del Peral-Rosado [(1)], G. Seco-Granados[(1)], J.A. López-Salcedo [(1)], J.A. García-Molina [(2,3)]

[(1)] Department of Telecommunication and Systems Engineering, IEEC-CERES, Universitat Autònoma de Barcelona (UAB), Spain
[(2)] European Space Agency (ESA/ESTEC), The Netherlands
[(3)] HE-Space, The Netherlands

ABSTRACT

The emergence of cloud computing platforms opens the door for offloading the computationally demanding tasks of Global Navigation Satellite Systems (GNSS) receivers to the cloud. This has two major advantages: first, it makes accessible advanced GNSS signal processing to any user, since the computational burden is carried out remotely; second, it drastically reduces the size and energy consumption of the user terminal, thus paving the way for low-power internet of things (IoT) applications. The present work focuses on the latter by presenting the development of a prototype IoT GNSS sensor. This prototype serves as a proof of concept for the machine-to-machine (M2M) interface of the Cloud GNSS receiver developed by authors. At the same time, it is aimed at sparking the interest on the use of cloud GNSS signal processing and stimulating the development GNSS-based IoT applications.

1. INTRODUCTION

The advent of internet of things (IoT) has sparked a renewed interest in positioning-based applications, especially in the context of Smart Cities, ambient assisted living or secure societies, just to mention a few. Global Navigation Satellite Systems (GNSS) play today a prominent role in the provision of position, velocity and timing (PVT) information, and they have been widespread adopted worldwide. Nevertheless, GNSS-enabled IoT devices face many challenges due to the stringent hardware limitations of miniaturized sensors in terms of low cost and low power consumption. This makes it difficult to cope with the harsh working conditions of GNSS signals in IoT applications, which are mostly deployed in urban areas. While low-power on-chip GNSS modules are starting to be commercialised for IoT, their capabilities are usually limited in performance and energy efficiency, since all the GNSS signal processing is still carried out on-chip [1]-[2].

Recent contributions have unveiled the potential of cloud GNSS signal processing [3], which exploits the communication capabilities of IoT devices and brings advanced features to simple user terminals, such as high-sensitivity, signal integrity monitoring, authentication and access to restricted (i.e., encrypted) services. The concept of cloud GNSS signal processing is based on the remote processing of GNSS signal samples in a cloud computing platform, thus moving all the computation load to the cloud and leaving the user terminal as a simple data grabber. This approach fits very well into the IoT paradigm, where miniaturized sensors take care of sensing the environment and reporting to a central control unit [4].

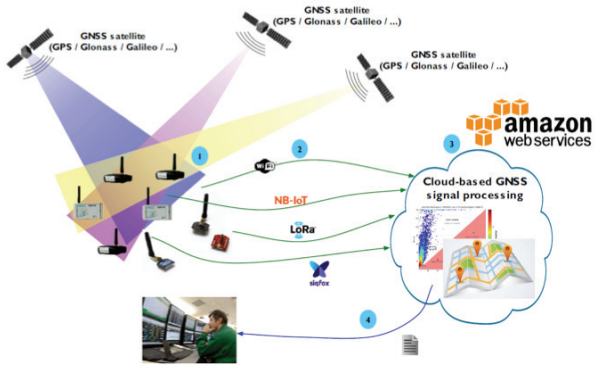


Figure 1. Architecture of an IoT GNSS sensor network using cloud GNSS signal processing services.

This paper presents the development of a prototype IoT GNSS sensor that is able to exploit the promising concept of cloud GNSS signal processing, by streaming GNSS signal samples to the Amazon Web Services (AWS) cloud platform, where an advanced high-sensitivity GNSS (HS-GNSS) software receiver is implemented. The paper is organised as follows. Section 2 focuses on the proposed architecture and Section 3 describes in detail the prototype of IoT GNSS sensor. Results obtained from a validation test campaign using the prototype IoT GNSS sensor are shown in Section 4. In Section 5, the advantages and potential applications provided by IoT GNSS sensors relying on cloud GNSS signal processing are discussed. Finally, the conclusions are drawn in Section 6.

2. CLOUD GNSS M2M-BASED SOLUTION

Cloud GNSS receiver services can be implemented with

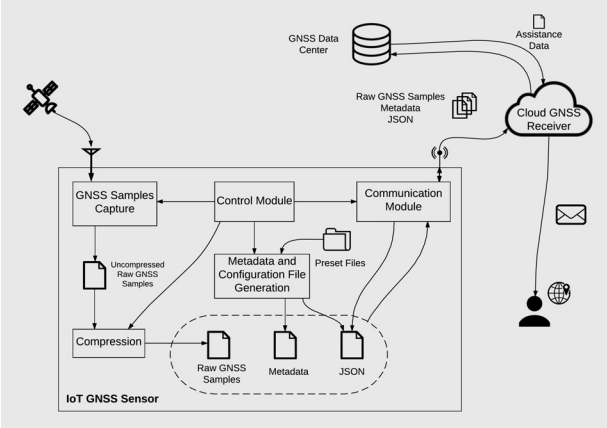


Figure 2. M2M-based architecture of the Cloud GNSS receiver with the prototype IoT GNSS sensor.

two different approaches. The first one, known as the human-to-machine (H2M) interface, requires the user interaction between the data grabbing device and the cloud processing platform in order to perform the remote GNSS signal processing. This is the case of the H2M service developed by the authors in [5], which allows users to access the cloud GNSS services through a dedicated webpage. The second approach, known as machine-to-machine (M2M) interface and presented herein, is based on the autonomous interaction between the GNSS user terminal and the Cloud GNSS receiver. This allows the user terminal to operate without any human supervision and to periodically upload GNSS samples to the cloud, as required in monitoring sensor networks, asset tracking or many other IoT-based applications.

The M2M architecture proposed in this work is based on a network of IoT GNSS sensors using a centralized cloud GNSS signal processing platform, as it is shown in Fig. 1. The IoT GNSS sensor operation consists essentially on the raw GNSS data grabbing and the GNSS data streaming to the cloud platform, where the data can be processed using a broad range of tools and computational resources. To do so, the sensor uploads both a file with the raw samples and a file with the requested configuration file for running the Cloud GNSS receiver. The obtained results are then sent back to the IoT GNSS sensor or to the user (i.e., control manager) of the sensor network, in case a centralized management of the network is involved.

The proposed M2M architecture is shown in Fig. 2, where the tasks carried out within the IoT GNSS sensor are grouped into four main functional blocks, as follows. Firstly, the GNSS samples capture module performs the signal conditioning by down-converting the GNSS signal impinging onto the sensor antenna and converting it into a binary data file with the raw GNSS samples. An additional compression functionality is performed in case it was necessary to reduce the required storage at the sensor, the streaming delay and the network bandwidth. This is typically the case when using a low-cost radio-frequency (RF) front-end originally designed for a non-GNSS application. For instance, the RTL-SDR dongle for receiving DVB-T signals [6], which can be tuned to the GNSS band but provides 8 bits per sample, far beyond the needs of mass-market GNSS receivers. This quantization value is reduced within the compression module of the IoT sensor down to 1 bit for efficiency purposes. Secondly, since these raw samples can be stored in a wide range of different formats, the implementation of some metadata file describing the inner data structure becomes critical. This file will later on allow the Cloud GNSS receiver to properly decode and read the received raw GNSS samples file. In our implementation, the GNSS metadata standard from the Institute of Navigation (ION) is implemented. A configuration file is also attached to indicate the cloud GNSS signal processing settings defined by the user. Thirdly, a communication module performs the streaming of the raw GNSS samples, metadata and configuration files to the cloud GNSS platform and requests the execution of GNSS signal processing. Finally, a main control unit is responsible for managing the sequential execution of the previously defined modules. Once all the required files are uploaded, the Cloud GNSS receiver itself generates and downloads the necessary assistance and navigation data, performs the GNSS signal processing as indicated in the configuration file, and finally sends the results to the IoT sensor or the sensor network manager.

3. IOT GNSS SENSOR PROTOTYPE

The IoT GNSS sensor prototype presented in this paper is based on the RTL-SDR dongle as a pluggable RF front-end and the Intel Galileo Gen 2 board implementing the control and communication unit, as it is shown in Fig. 3.

The Intel Galileo Gen 2 is a single-board microcontroller, here used as the main workstation where the different hardware modules are combined and the software routines will be executed. This low-power consumption board with modest resources is used here as a proof-of-concept for the development of IoT GNSS-based applications. The main specifications of the Intel Galileo Gen 2 board are the Intel Quark SoC X1000 32-bit single core processor at 400 MHz, 256 MB DRAM, up to 32 GB storage size by using a micro-SD card, a USB 2.0 host port, a Linux kernel, an Ethernet port, and a mini-PCIe slot [7]. The Linux kernel can be remotely accessed through a secure shell (SSH) protocol communication to control and configure the Intel Galileo board. Furthermore, an advanced Yocto-built Linux distribution must be installed to provide the board some additional required

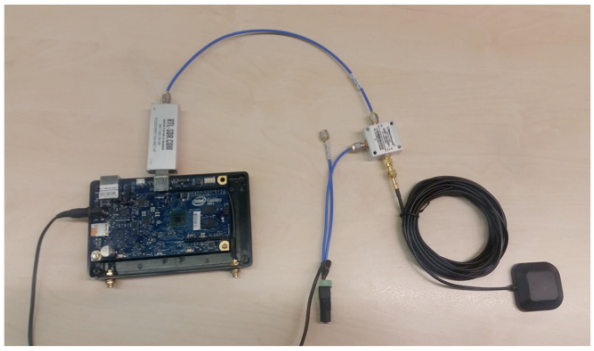


Figure 3. Designed IoT GNSS sensor prototype.

functionalities for the IoT GNSS sensor development that are not available on the default Linux distribution built into the board. The advanced distribution includes GNU Compiler Collection (GCC), Python, Node.js and additional drivers for many Intel hardware modules, to name a few [8]. The next subsections describe in detail the four main functional modules of the proposed IoT GNSS sensor prototype.

3.1. GNSS samples gathering module

The first module performs the RF front-end tasks to capture and obtain the baseband digital samples of the GNSS signal impinging on the sensor antenna. Then, a compression algorithm is applied on the raw GNSS samples file, in order to reduce the storage, delay and bandwidth requirements.

This module requires a receiving antenna compatible with the desired GNSS bands. In this work, a patch antenna ANT-555 has been used, which is designed to work on the Global Positioning System (GPS) L1 and Galileo E1 band at 1575.42 MHz, with a 50 Ω input impedance [9]. This antenna requires DC power to feed a built-in preamplifier, so a bias-tee is needed. In order to fit in the ergonomy and small-size design requirements of IoT applications, future prototypes may consider planar antennas, as they are commonly used in RF integrated circuits (RFIC) for small-size mobile devices.

The RF front-end tasks are carried outside of the Intel Galileo Gen 2 board, by an external hardware module. The designed prototype is based on an RTL-SDR v3 USB dongle, which is a low-cost and small size pluggable Software Defined Radio (SDR) device with open source hardware and software. The RTL front-end can be adjusted to operate in the GNSS band of interest, and it provides an IQ sampling rate up to 3.2 MHz with an 8-bit quantization. However, when working at this maximum sampling frequency, the RTL-SDR may drop samples. Thus, our prototype works at a sampling rate of 2.8 MHz to avoid sample drops. The input impedance of the RTL-SDR dongle is 50 Ω, hence matching the ANT-555 antenna [10]. The main issue with the RTL-SDR front-end is that, due to its built-in temperature compensated crystal oscillator (TCXO), one must be in charge of compensating the potential frequency drifts. To do so, a previous calibration of the RTL dongle must be performed to estimate such offset and to compensate for that in the intermediate frequency (IF) of the raw GNSS samples reported to the cloud.

Open Source Mobile Communications (Osmocom) provide the open source drivers for the RTL-SDR, which consist on some C libraries for the use and configuration of the front-end dongle [11]. These libraries can be modified to change some software-controlled settings, such as the samples byte encoding or to enable the internal bias-tee of the RTL-SDR dongle. The samples captured by the dongle come in the form of 8-bits integers (int8 type) and they come as the result of executing the pre-defined function *rtl_sdr*. The input parameters to this function are the central RF frequency, the sampling rate, the number of samples to be taken, the clock frequency error and the name of the file containing the samples that is going to be generated.

As mentioned above, the RTL provides 8-bit samples in its output file. However, for mass-market GNSS applications, such quantization is excessive and one can operate instead with a 1-bit quantization. This is done here by keeping the most significant bit (MSB) of the received samples using an "AND" boolean operation with a binary mask that selects the first bit of each input byte. Then, the MSBs from eight consecutive samples are packed together in each of the output bytes that are stored and uploaded to the cloud. This process is applied in a loop to compress the raw samples file coming from the RTL-SDR dongle.

3.2. Metadata and configuration file generation module

The metadata and configuration files are also generated together with the signal capture in order to assist the Cloud GNSS receiver. These files are required to correctly decode and read the samples file at the cloud, and to process the raw GNSS samples according to the user-defined settings.

The proliferation of different GNSS SDR front-end receivers in the recent years has generated a heterogeneity of file formats. With the objective of putting some order and allowing interoperability between GNSS SDR systems, the ION has defined a metadata standard to describe the structure of the GNSS samples obtained by SDR front-end receivers, as well as some additional useful information for GNSS processes [12]. The standard is of special relevance in our architecture, since the Cloud GNSS receiver is a centralized GNSS processing unit that will receive

GNSS data from a plethora of SDR front-end receivers. The ION metadata file has an XML (eXtensible Markup Language) extension. The file scheme is described in an XSD (XML Schema) file that can be found on the documentation of the ION repository for the metadata application programming interface (API) [13]. The information contained in the ION metadata file is organized in twelve main classes (session, system, cluster, source, band, stream, lump, chunk, block, lane, file and fileset) that provide information about the samples file structure, the system, and session information. For the basic capture topology implemented by our prototype IoT GNSS sensor, the main metadata fields that must be indicated are mentioned below:

- **System**. Basic settings of the SDR front-end system, such as the sampling frequency.
- **Band**. Information about the frequency bands involved in the RF front-end process, such as the RF and IF central frequencies.
- **Stream**. Information about the structure of the data stream captured, such as the quantization, the samples format (e.g., IF, IQ, etc.) and the encoding (e.g., sign, sign-magnitude, etc.).
- **File**. Information of the GNSS samples file, such as the file name and the file path.

For an easier deployment of the metadata standard, the ION provides a C++ API, which contains the required classes definitions and functions to develop metadata standard-related applications. Using the ION API functions, we have developed a simple program that reads the specific front-end and system specifications contained in a preset file and generates the corresponding metadata file.

In addition to the format of the raw samples, the settings of the GNSS signal processing need to be indicated in a configuration file. This is done using JavaScript Object Notation (JSON), which consists on a basic plain text message with an attribute-value pair data structure. It is formed by four main fields:

- **General data**. This field contains information such as the ID execution, user e-mail where the report will be sent, and the user privileges, among others.
- **Receiver configuration**. Contains the settings of the GNSS signal processing to be applied, such as the considered GNSS (i.e., GPS), the coherent integration time, or the number of non-coherent integrations.
- **Raw samples**. This field indicates the samples file and the metadata file to be uploaded. The information that has already been included in the ION metadata file does not have to be fulfilled again. Instead, it is just needed to enable the flag that indicates the use of the ION metadata file.
- **Assistance data**. This field indicates a coarse reference position and time so that the cloud can automatically obtain an assistance data file for reducing the search space by obtaining the Doppler of the visible satellites, and a navigation file (i.e., Receiver Independent Exchange, RINEX) to compute the PVT information.

3.3. Communication module

Once the raw GNSS samples are captured and the corresponding files are generated, the communication module streams the data to the Cloud GNSS receiver. Even though the Intel Galileo Gen 2 board provides an Ethernet network interface, a wireless communication module must be added to the prototype for the achievement of a mobile solution. In this work, an Intel Centrino Wireless-N 135 module has been installed on the Intel Galileo Gen 2 board through the mini-PCIe slot. This single-band Wi-Fi module provides a network bandwidth up to 150 Mbps [14]. It is immediately compatible with the Intel Galileo since the installed distribution includes drivers of many Intel hardware modules. Depending on the application alternative wired or wireless communication modules can be considered, such as LoRa or Sigfox, among others.

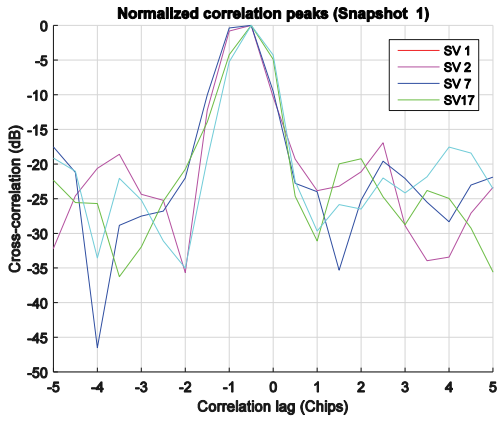
Since the Cloud GNSS receiver is built on AWS, the functions to connect and interface with the cloud servers are provided by the AWS Software Development Kit (SDK). AWS provides SDKs in different programming languages such as Java, PHP or C++, among others. The IoT GNSS sensor prototype uses boto, the Python SDK provided by AWS [15].

A Python script developed for the communication process uploads the raw GNSS samples and metadata files to the AWS Simple Storage Service (S3) server. It is required to include in the JSON configuration file the created endpoints of the files, so the Cloud GNSS receiver can access to this data. Lastly, the JSON file is sent as a request message to the AWS Simple Queue Service (SQS) to queue until it can be processed.

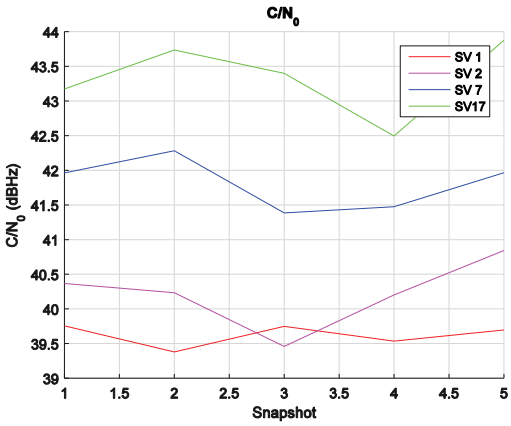
3.4. Control module

The above stated modules implement the required steps to obtain, generate, and stream all the necessary data to the Cloud GNSS receiver. These tasks are managed by the control module, a program that sequentially runs these functions providing to each module its required inputs. In case that the IoT GNSS sensor performs a continuous GNSS signal monitoring, the control unit runs the sequential steps in a loop and manages the periodical timing between monitoring cycles.

This module has as inputs the length of the signal snapshots and the period between working cycles, whilst the rest of the capture settings and the metadata and configuration files information are obtained from predefined files.



(a) Normalized correlation peaks.



(b) C/N_0 measurements.

Figure 4. Results of the cloud GNSS signal processing using the IoT GNSS sensor prototype.

4. RESULTS

The correct operation of the IoT GNSS sensor prototype has been verified with a validation test. This test has been performed by capturing a GPS L1 band signal, with a 2.8 MHz sampling rate and snapshots of length equal to 20 ms. The prototype has been placed in a windowsill at a medium height of a building and thus, in half-sky conditions. After running the validation test, we have successfully received the results report from the cloud, meaning that the prototype has correctly performed its functionalities. A pair of figures contained in the report are shown in Fig. 4. Fig. 4a shows the normalized correlation peaks of the 4 acquired GPS satellites. In Fig. 4b the cloud reports the carrier-to-noise-density ratio (C/N_0) measurements of the acquired satellites for a series of 5 snapshots. Due to the half-sky condition, the number of visible satellites has been halved to just 4 acquired satellites.

5. POTENTIAL IOT APPLICATIONS OF CLOUD GNSS SIGNAL PROCESSING

The Cloud GNSS receiver developed by the authors opens the door for a new range of GNSS applications and features, some still to be explored. One of these features is that of simplifying the GNSS user terminal functions and resources in order to reduce the size and the power consumption of the device. This optimization would allow designing low-cost GNSS modules with high potential in the IoT domain. This is, in addition, a good solution to implement some of the following emerging GNSS applications.

5.1. Positioning for Smart Cities

GNSS positioning applications are continuously arising and improving. The IoT GNSS sensor concept is an efficient solution to provide GNSS positioning information to any connected object by using a few simple low-cost and low-power modules. This allows introducing GNSS advents into the IoT domain, but fitting them into the stringent IoT terminal requirements. In this context, the IoT GNSS sensor can have a leading role in positioning applications for Smart Cities, where such requirements are wanted. For instance, GNSS positioning will be a key element in the efficient management of urban resources [16]. Parking management systems and networks of vehicles sharing real-time traffic information are a few more examples of Smart Cities applications that would require of GNSS positioning solutions. The simplistic design of the IoT GNSS sensor would allow embedding this module on many different type of *things* without having to apply substantial modifications on their physical structure. Its low-cost implementation would facilitate its deployment and its low power consumption would provide a longer battery life solution.

5.2. GNSS signal monitoring

The IoT GNSS sensor can be used as a generic data grabbing station for the collection of raw GNSS data and streaming to the Cloud GNSS receiver for a subsequent analysis. This workflow would allow the deployment of low-cost and low-power sensor networks for covering strategic regions with the objective of extracting certain information. In this context, many applications can be put into practice, such as integrity monitoring, interference detection, ionospheric monitoring, etc.

Most of the potential applications imply some level of software development at the sensor side, and they are therefore very well-suited for emerging new companies focusing on IoT-based services. However, it is also true that an IoT GNSS sensor similar to the one described in this work can easily be implemented by an amateur, and used as a personal positioning device by accessing the Cloud GNSS receiver on demand. Similarly, the IoT sensor described herein is also a valuable tool for researchers in the GNSS field.

6. CONCLUSIONS

This paper has presented the development of an IoT GNSS sensor prototype as a proof-of-concept of an M2M interface with the Cloud GNSS receiver. The prototype has been able to apply an advanced HS-GNSS signal processing without having the on-board required computational resources for such purpose.

The prototype has been built by means of a few low-cost and low-power consumption modules in a simple building architecture, thus proving the viability of the introduced IoT GNSS sensor. Due to its design features, the IoT GNSS sensor may have potential in IoT and Smart Cities applications, such as urban resource positioning and GNSS data gathering for several purposes. It is important to mention, though, that the IoT GNSS sensor developed herein serves merely as a proof of concept of the M2M interfacing with the cloud GNSS receiver developed by the authors. Evolved versions of the IoT GNSS sensor could be implemented by combining alternative single-board microcontrollers, antennas, RF front-end receivers and network communication technologies. In summary, the cost-effective design requirements together with the simplicity and adaptability make the IoT GNSS sensor a promising concept for the GNSS community.

7. ACKNOWLEDGEMENTS

The views presented in this paper represent solely the opinion of the authors and not necessarily the view of ESA. This work was partly supported by the European Space Agency (ESA) under contract No. 4000119070/16/NL/GLC and by the Spanish Government under grant TEC2014-53656-R.

8. REFERENCES

1. u-blox. Position & Time Products. Online at *https://www.u-blox.com/en/position-time* (as of 20 November 2017)

2. Sierra Wireless. GNSS Positioning Modules for the Internet of Things (IoT). Online at *https://www.sierrawireless.com/products-and-solutions/embedded-solutions/gnss-positioning-modules/* (as of 20 November 2017)

3. V. Lucas-Sabola, G. Seco-Granados, J.A. López-Salcedo, J.A. García-Molina, y M.Crisci. (2016). Cloud GNSS receivers: New advanced applications made possible. In *Proc. 2016 International Conference on Localization and GNSS (ICL-GNSS)*, Barcelona, Spain.

4. V. Lucas-Sabola, G. Seco-Granados, J.A. López-Salcedo, J.A. García-Molina, y M.Crisci. (2017). Efficiency Analysis of Cloud GNSS Signal Processing for IoT Applications. In *Proc. ION GNSS+*, Portland, Oregon.

5. ESA and UAB. (2017). GNSS Signal Processing in the Cloud. Online at *http://spcomnav.uab.es/cloudGNSSrx* (as of 20 November 2017)

6. R. W. Stewart, L. Crockett, D. Atkinson, K. Barlee, D. Crawford, I. Chalmers, M. McLernon, E. Sozer, "A low-cost desktop software defined radio design environment using Matlab, Simulink, and the RTL-SDR", IEEE Communications Magazine, vol. 53, no. 9, pp.64-71, Sept. 2015.

7. M.C. Ramon. (2014). *Intel Galileo and Intel Galileo Gen 2. API Features and Arduino Project for Linux Programmers*, Apress, Berkeley, CA, ch. 1.

8. M. de Sousa. (2015). *Internet of Things with Intel Galileo*, Packt Publishing, Birmingham, ch. 5.

9. Radio Export. ANT-555 Magnetic Mount GPS Antennas. Online at *http://www.radioexport.com/wp-content/uploads/pdfs/028121206adebe6272862129843d3f37.pdf* (as of 20 November 2017)

10. RTL-SDR.COM. RTL-SDR Blog V3 Datasheet. Online at *https://www.rtl-sdr.com/wp-content/uploads/2017/06/RTL-SDR-Blog-V3-Datasheet.pdf* (as of 20 November 2017)

11. GitHub osmocom. rtl-sdr. Online at *https://github.com/osmocom/rtl-sdr* (as of 20 November 2017)

12. ION GNSS SDR Metadata Working Group. GNSS Software Defined Receiver Metadata Standard. Online at *http://sdr.ion.org/* (as of 20 November 2017)

13. GitHub JamesTCurran. GNSS-Metadata-Standard. Online at *https://github.com/JamesTCurran/GNSS-Metadata-Standard* (as of 20 November 2017)

14. Intel. Intel Centrino Wireless-N 135, Single Band. Online at *https://ark.intel.com/products/66887/Intel-Centrino-Wireless-N-135-Single-Band* (as of 20 November 2017)

15. Amazon Web Services. AWS SDK for Python (Boto3). Online at *https://aws.amazon.com/sdk-for-python/* (as of 20 November 2017)

16. Srdjan Tadic, Alfredo Favenza, Christoforos Kavadias, Vassilis Tsagaris (2017). GHOST: A novel approach to smart city infrastructures monitoring through GNSS precise positioning. In *Proc. 2016 IEEE International Smart Cities Conference (ISC2)*, Trento, Italy