ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

Ecole Nationale des Sciences Géographiques

IFSTTAR

Institut Français des Sciences et Technologies des Transport, de l'Aménagement et des Réseaux

Thèse Professionnelle

Mastère spécialisé® Photogrammétrie, Positionnement, Mesure de Déformations

Development of a GNSS positioning application under Android OS using GALILEO signals

ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

Antoine Grenier

8 septembre 2019

☒ Non confidentiel ☐ Confidentiel IGN ☐ Confidentiel Industrie ☐ Jusqu'au ...

Ecole Nationale des Sciences Géographiques
6-8 Avenue Blaise Pascal - Cité Descartes - 77420 Champs-sur-Marne
Téléphone 01 64 15 31 00 Télécopie 01 64 15 31 07

Jury

Président de jury :

Serge BOTTON, IGN/ENSG/DPTS

Commanditaire :

Institut Français des Sciences et Technologies des Transport, de l'Aménagement et des Réseaux (IFSTTAR)

Encadrement de stage :

Valérie RENAUDIN

Enseignant référent :

Xavier COLLILIEUX, Référent ENSG
Lauren MOREL, Référent ESGT

Rapporteur expert :

Paul REBISCHUNG, IGN/LAREG

Responsable du MS® PPMD :

Jacques BEILIN, IGN/ENSG/DPTS
Jean-François HANGOUËT, IGN/ENSG/DIAS

Gestion du stage :

Anna CRISTOFOL, IGN/ENSG/DSHEI

Stage de fin d'étude du 23 avril au 27 septembre

Diffusion web : ☒ Internet ☒ Intranet Polytechnicum ☒ Intranet ENSG

Situation du document :

Rapport de stage de fin d'études présenté en fin de 3ème année du cycle des Ingénieurs

Nombres de pages : 89 pages dont 13 d'annexes

Système hôte : LaTeX

Modifications :

EDITION	REVISION	DATE	PAGES MODIFIEES
1	0	09/2019	Création

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

ECOLE SUPERIEURE DES GEOMETRES ET TOPOGRAPHES

MEMOIRE

présenté en vue d'obtenir

le DIPLOME D'INGENIEUR CNAM

SPECIALITE : Géomètre et Topographe

par

Antoine GRENIER

Développement d'une application de positionnement GNSS sous Android incluant les données GALILEO

Soutenu le 20 septembre 2019

JURY

Monsieur Serge BOTTON	Président du jury
Monsieur Paul RUBISHUNG	Rapporteur
Madame Valérie RENAUDIN	Maître de stage
Monsieur Laurent MOREL	Référent ESGT
Monsieur Xavier COLLILIEUX	Référent ENSG

To improve is to change, to perfect is to change often.

— Winstom Churchill

Acknowledgment

I want to express my most profound appreciation to all those who provided me the possibility to complete this master thesis. A special gratitude I give to my supervisor Valérie Renaudin (GEOLOC) and my reference teachers Xavier Collilieux (ENSG) and Laurent Morel (ESGT), whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

Furthermore, I would like to acknowledge all the GEOLOC team: Miguel Ortiz, Celine Ragoin, Gilles Le Roux, and Aigul Khamitova for their help throughout my internship, with specials thanks to Ni Zhu and Johan Perul, for the time they spent finding answers on my algorithms issues. I would also to thank the other interns who were with me during this internship, Haowen Tang, Abderrahmane Laanati and Yazheng Wei, for the good moments we spend all together.

I also want to thank all my friends for their help and support during this internship, especially Camille Parra, for her constructive remarks on my report.

Last but not least, I would also like to thank my parents for their support and encouragement during this internship, but also my studies and all my life.

Summary

Résumé (summary in French)

Ce mémoire présente mon travail au sein du laboratoire de recherche Geoloc de l'Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux (IFSTTAR), se déroulant d'avril à septembre 2019. Le sujet de ce stage est tourné autour du développement d'une application Android utilisant les mesures GNSS brutes reçues par le récepteur du téléphone pour le calcul d'une position en temps réel. Plus particulièrement, l'intégration des signaux GALILEO (GNSS européen) dans l'application est étudié dans le but d'améliorer le positionnement et calculer une solution multi-constellation et multi-fréquence. Avec le nouveau téléphone Xiaomi Mi8 sorti en 2018, il est en effet possible de recevoir les fréquences L1 et L5. En joignant les constellations GPS et GALILEO, il devient alors possible de réfléchir à du Positionnement de Point Précis en temps réel (PPP-RTK) directement à l'intérieur d'un téléphone. Ce travail de recherche liste les difficultés du positionnement par téléphone Android, ainsi que les données nécessaires à un positionnement précis multi-constellation. Les résultats de l'algorithme en Positionnement de Point Standard (SPP) développé durant ce stage sont étudiés et une proposition d'algorithme de PPP-RTK est détaillé, à des fins d'amélioration possible de l'application.

Mots-clés : GNSS, Android, Smartphone, SPP, PPP-RTK, RTCM streams, GALILEO.

Summary

This thesis presents my work within the Geoloc laboratory of the French institute of science and Technology for transport, development, and network (IFSTTAR). The subject is the development of an Android application using raw GNSS measurements acquired by the phone's receiver for computation of a position. More particularly, focus was given on integrating the GALILEO signals, the European GNSS system. The goal is to compute a multi-constellation and multi-frequency solution for enhanced precision. With the newly available Xiaomi Mi8 smartphone which came out in 2018, it is now possible to receive bi-frequency GNSS measurements (L1/L5) on an Android device. By joining GPS and GALILEO constellation, real-time Precise Point Positioning can now be studied for integration inside a smartphone. This research paper lists the positioning challenges coming with Android devices and the necessary data to perform precise positioning with multiple constellations. Results from the SPP algorithm developed during this internship are presented, and a proposal for a PPP-RTK algorithm is detailed for future enhancement of the application.

Keywords : GNSS, Android, Smartphone, SPP, PPP-RTK, RTCM streams, GALILEO.

Contents

List of Figures

List of Tables

Glossary

BKG Bundesamt für Kartographie und Geodäsie - Federal Agency for Cartography and Geodesy. 32

CNES Centre National d'Etudes Spatiales. 22

DCB Differential Code Bias. 36
DGNSS Differential GNSS. 31, 63

GALILEO European global navigation by satellite system. 21
GGTO GPS to GALILEO Time Offset. 44
GLONASS Russian global navigation by satellite system. 21
GNSS Global Navigation Satellite Systems. 39, 63
GPS Global Positioning System. 21
GSA European GNSS Agency. 23
GUI Graphical User Interface. 28

ICD Interface Control Document. 35
IGS International GNSS Service. 22
IoT Internet of Things. 21

NTRIP Networked Transport of RTCM via Internet Protocol. 32

PPP Precise Point Positioning. 19

RTCM Radio Technical Commission for Maritime Services. 31
RTK Real Time Kinematic. 19

SPP Standard Point Positioning. 11

TDCP Time Differenced Carrier Phase. 22

Introduction

In 2016, Google announced the availability of GNSS raw measurements directly through the Google API, starting with Android version 7 (Nougat). This led to the development of several positioning apps (GNSS Compare [15], PPP Wizlite [32], ...) used to assess for the first time the quality of GNSS signals received on smartphones. Since then, the number of research papers on precise positioning with a smartphone has kept growing, and the development of new generations of smartphones allowed to push back the limits of low-cost receivers capacities. In June 2018, Xiaomi released a new smartphone "Xiaomi Mi8", integrating a Broadcom BCM47755 GNSS receiver with dual-frequency (L1/L5) capacities. This was an essential step in smartphones and IoT developments, since it opens the way for highly precise positioning algorithm.

This research paper presents my work at the *Geoloc* laboratory (IFSTTAR), which took place from April to September 2019. The subject is the development of an Android application using raw GNSS measurements acquired by the phone's receiver for computation of a position. More particularly, focus was given on integrating the GALILEO signals, the European GNSS system, to compute a multi-constellation and multi-frequency solution for an enhanced precision. The laboratory already possessed an application called "GeolocPVT", developed during a previous internship by José Gilberto Resendiz Fonseca. My job was to enhance this existing solution for more precise positioning capacities. The laboratory recently acquired a Xiaomi Mi8 smartphone, and this application therefore needs to be upgraded to make use of these measurements and study the quality of them. After reviewing the initial state of the app, challenges with Android development and the raw GNSS measurements will be presented. A large part of this internship has been focusing on the retrieval of real-time streams for enhanced positioning. Details on their integration in an Android application are given. The developed algorithms to enhance the precision of the computed position are presented, along with analysis of the results obtained by the application. Finally, a proposal of a PPP-RTK algorithm for precise positioning on a smartphone is detailed for future updates of the application.

This work is part of the laboratory activity on the "Raw Measurements Task Force", a workforce organized by the European GNSS Agency (GSA). Its objective is to share the knowledge between laboratories around the world related to GNSS raw measurements coming from Android devices, especially those from GALILEO signals. Every year a conference is organized in the GSA headquarters (Prague, Czech Republic) for presenting the year's results. Since it happened during my internship, I went to present the lab's application "GeolocPVT" and the development I made during my first two months. As we'll see in the different sections, this allowed me to refocus my work on important points presented by other laboratories during this presentation.

Chapter 1
GNSS on Android

This chapter introduces the development of an application on Android, summarizing the State of the Art on computing a precise position using the raw measurements available from Google API. We also go into details on how GNSS observation are retrieved from those raw measurements and the challenges that come with them.

1.1 State of the Art

With the growing use of IoT devices for research and mass-market purposes, the need for low-cost precise positioning is increasing significantly. In 2016, Google announced the availability of GNSS raw measurements directly through the Google API, starting with Android version 7 (Nougat). This leads to the development of several positioning apps (GNSS Compare [15], PPP Wizlite [32], ...) used to assess for the first time the quality of GNSS signals received on smartphones. DGNSS-RTK technique was used to compute positions from those measurements, either in post-processing or in real-time [31]. However, real-time PPP or PPP-RTK are still not as easily manageable as DGNSS-RTK since measurement corrections of atmospheric effects need to be modeled.

Following that, Xiaomi released a new smartphone "Xiaomi Mi8" in June 2018, integrating a Broadcom BCM47755 GNSS receiver with dual-frequency (L1/L5) capacities. This was an essential step in smartphones and IoT developments, opening the way for highly precise positioning algorithm. With multiple frequency, signal combination, which is mandatory for ambiguity fixation and signal corrections (e.g., Iono-free combination), is possible.

Following the release of the phone, quality assessment of the GNSS receiver, in particular phase measurements' quality, has been done in several studies [36, 43, 12], where post-processing PPP using multiple constellations and frequencies have been realized too. Robustelli et al.[36] presented a positioning solution with ambiguity estimation, also known as *floating* ambiguities, with the Mi8 using multiple constellation (GPS, GLONASS, GALILEO) and only L1/E1 measurements. Using RTKLIB as a post-processing software, they achieved a sub-meter positioning in low-multipath environment after 1 hour of convergence time, showing that multi-constellation improved positioning compared to GPS only solution, which highlights the need for multi-constellation when computing PPP. Using a different model, Wu et al. [43] achieved a sub-meter positioning after a convergence time of 102 min, rapidly converging afterward to a 0.2 m accuracy after 116 min.

Nottingham Scientific Limited team (NSL) with the FLAMINGO initiative [12] (Fulfilling enhanced Location Accuracy in the Mass-market through Initial GalileO services) performed a kinematic positioning reproducing the movement of a pedestrian walking on a straight line. Although their protocol is different from previous studies, they showed a final RMSE of 2.23m using PPP, impacted mainly by the multipath effects of the experience environment. The positioning computations were once again performed on RTKLIB in post-processing. It is important to note that the FLAMINGO initiative's final goal is to develop a complete SDK (Software Development Kit) to be integrated inside applications with a positioning accuracy of a few decimeters in real-time. Right now, this initiative is still in the early stage of development.

In a study published in January 2019, Zhu et al. [26] demonstrated precise dynamic positioning in SPP (no ambiguity estimation), using smartphones Huawei P10 and Samsung S8. This was performed by combining GNSS and INS observations. Phase measurements are used in a Time Differenced

Carrier Phase (TDCP) algorithm for precise estimation of velocities, joined with a Pedestrian Dead Reckoning (PDR) algorithm which uses the phone's inertial sensors. This led to proper positioning with horizontals errors up from 1 m to 3 m, with a RMS value of 2.57 m for the GNSS/PDR approached, compare to 3.18m with GNSS only filtering. Results in poor environments are also given, showing how inertial sensors of the phone can keep a right positioning solution with short GNSS outages.

All these examples were performed in post-processing mode and not in real-time aboard the phone, resulting in more manageable PPP computations. Indeed, once in real-time, computations need to account for other corrections, starting with the lack of final precise orbits from the IGS (International GNSS Service). This leads to differences in satellite positions computations and worse accuracy of those positions, but also more IoT related issues like Internet data consumption on-board the app or battery consumption as well. Another solution is to use the precise corrections products from IGS Real-Time Service (IGS-RTS), a network of processing centers sending real-time streams under the RTCM protocol. With this, it is possible to obtain precise satellite positions in real-time, as well as VTEC contents of the ionosphere, satellites code and phase biases.

Wang et al. [42] analyzed, and compared the precise corrections products from IGS-RTS coming from different centers. Using the RTPosNavi_AOE software (Academy of Opto-Electronics, Beijing, China) based on the RTKLIB software, they compared real-time PPP using real-time streams products to post-processing PPP using IGS final products. This was done on geodesic grade receivers in static positioning and showed good results for real-time PPP. Moreover, it presented multi-GNSS positioning (GPS + BEIDOU), indicating that the CLK93 and CAS01 mount points of the IGS-RTS network send corrections for four constellations: GPS, GLONASS, GALILEO, and BEIDOU. This was not given on the CNES website at first and will be essential for the computations presented in this paper.

1.2 Initial state of the GeolocPVT

The *GeolocPVT* application is a program developed by the Geoloc laboratory working under Android OS. José Resendiz first developed it in 2018 [35] in Android 7 (API 25), then was ported to Android 8 (API 26) by Sheng Gao and Mitsinjosoa Ramandaniaina [17].

The app functionalities when this internship started are listed here:

— Retrieval of raw GNSS measurements for L1 & L5 frequencies, for GPS and GLONASS constellations;
— Computation of the satellites' positions by decoding the navigation message, for the GPS constellation only;
— Computation of user coordinates with pseudoranges measurements through least square estimation, for L1 frequency only, GPS constellation only;
— GUI displaying the user's coordinates, with Open Street Map (OSM) support;
— Logging of raw measurements, user position, satellites ephemerises into external files.

The main framework of the app was completed, and the Graphical User Interface (GUI) was setup. It allowed me to focus directly on the positioning computations, which were present but limited and yet to be verified. Before adding more precise positioning capacities to the app, it was necessary to assess different part of the computations code written before me, which were working but their solution had to be appropriately checked, as we will see in the next sections.

1.3 Raw measurements

Raw measurements can be accessed in Android devices using the Google API **android.location**. Not every smartphone is enabled with this functionality, and a list of compatible devices can be found

on the Google developer website [1]. As explained in the introduction, the device I have been working on during this internship is the Xiaomi Mi8. The Geoloc lab has just acquired this smartphone which came out during 2018 because of its new capacities: the first smartphone with the dual-frequency reception (L1/L5), with phase measurements support. As we have seen in section 1.1, many papers came out during the year assessing this phone measurements' quality, as it opens the door to real-time Precise Point Positioning (PPP-RTK).

First, we are going to look into how Android API manages the raw GNSS measurements and how to properly retrieve them, before talking about the challenges of computing a positioning solution with Android. This section is based on a White Paper written by the European GNSS Agency (GSA) [1] and the 2018 IPIN conference [16].

Raw measurements are available through an event defined in the application *MainActivity* class. The application can register these events using the *Callback* function defined in the class *LocationManager* to receive the measurements.

```
LocationManager.registerGnssMeasurementsCallback(mGnssMeasurementsEventCallback)
```

1.3.1 Pseudorange generation

Because Raw GNSS Measurements on Android was not made for researchers at first, but for the devices' manufacturers, the API was not build as a GNSS receiver would be. Pseudorange measurements are not provided directly by the system. Instead, it needs to be generated using the time measurements provided by the *GnssMeasurement* and *GnssClock* classes. Received time t_{Rx} and transmitted time t_{Tx}, provided in nanoseconds, are used with c the speed of light to compute the pseudorange as defined in equation 1.1.

$$P = \frac{(t_{Rx} - t_{Tx})}{1e9} c \tag{1.1}$$

The transmitted time t_{Tx} is directly provided by the API, but the t_{Rx} needs to be reconstructed through different time measurements. Receiver clocks is not align with the GNSS Time at first. Because of that, biases need to be applied to the received time associated with the measurements to align the t_{Tx} and t_{Rx} on the same time scale. This is done by using the *FullBiasNanos* and *BiasNanos* variable provided by the *GnssClock* class. Using the code described below, one can find the received time and use it with the transmitted time to compute the pseudorange.

```
t_rx_gnss = TimeNanos + TimeOffsetNanos - (FullBiasNanos + BiasNanos)
numberNanoSecondsWeek = 6.048e14

t_rx = mod(t_rx_gnss, numberNanoSecondsWeek)
```

Note that the equation is valid only for GPS and GALILEO measurements with their Time of Week (ToW) decoded. An inter-system bias is otherwise needed. For more information on the pseudorange generation, please refer to the GSA White Paper [1] and the Android Developer website [21].

An important thing to keep in mind when performing computations with GNSS measurements from Android devices is that receiver clocks used drift rapidly and needs to be corrected continuously, leading to problems in measurements. This will be detailed in section 1.4.

1.3.2 Carrier phase measurements

Carrier phase measurements are easier to retrieve, though they are not available on every device. Since API 26, they can be accessed through the function *GnssMeasurment.getAccumulatedDeltaRangeMeters*. The measurements are given in meters and not cycle. Information on the quality of the measurement

1. https://developer.android.com/guide/topics/sensors/gnss

can also be retrieved using the function *GnssMeasurment.getAccumulatedDeltaRangeStates*, which provide a bit string representing different flags about the phase, which can be useful for cycle-slip detection.

1.4 Challenges related to Android GNSS measurements

During this internship, I faced different challenges related to Android development when one wants to use the raw GNSS measurements for computing a position. I tried to summarize them in this section to explain the choices made during the design of the positioning algorithm.

1.4.1 Pseudorange and phase divergence

Pseudorange are reconstructed in the Google API using time measurements (see section 1.3.1). Because of the quality of the receiver clock, the *FullBiasNanos* parameter will show a 256 nanoseconds jump every 3 seconds, leading to discontinuities in the pseudorange measurement between certain epochs [18]. Since we are estimating the receiver clock error, this "rollover" is usually adsorbed in this parameter estimation and does not cause any issue. However, the phase measurements do not show this kind of jumps in time, since it is not obtained through time measurements but directly from the API. When comparing the two measurements over time, I found that the pseudorange tends to diverge from the phase. This leads to discrepancies between the measurements and errors in the computations if there are used together. This is why it is essential to keep the *FullBiasNanos* value fixed after the first measurement is received. No divergence between measurements can be seen anymore. However, it also means that our receiver clock estimation will vary very fast from one epoch to another. It is therefore important to take that into account later into our Kalman filter when accumulating epochs and to also estimate a clock drift in the state vector.

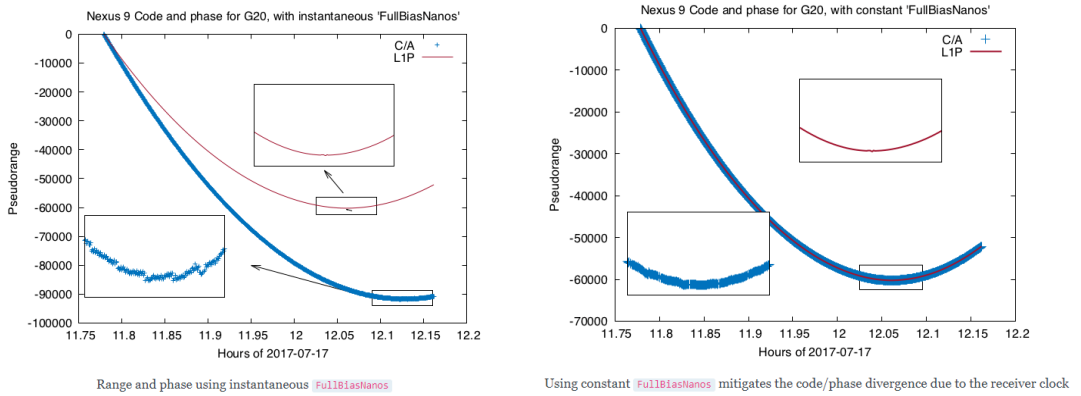


Figure 1.1 – Pseudorange generation with a non-fix and a fixed *Fullbias* (Source: Miquel Garcia [18])

.

1.4.2 Multiple systems and frequencies

Depending on the phone model, the GNSS receiver chipset varies, along with constellation the device can receive. As of today, five operational constellations of satellites are available around the world: GPS (USA), GLONASS (Russia), GALILEO (EU), BEIDOU (China) and QZSS (Japan). The Xiaomi Mi8 embeds a Broadcom receiver BCM47755 that can receive all those five constellations, depending on the user's location on the Earth (QZSS being a regional system only). Depending on the signal plan and the design of the constellation, multiple frequency of the satellites can be received in the device. As explained in section 1.1, the Xiaomi Mi8 can receive the L1 and L5 signals from the GPS satellites, but also the E1 and E5a from GALILEO satellites since there are located on the same frequency bands.

However, even if all those signals are received, this does not mean that every navigation messages are available. Only Broadcom receivers (in opposition to Qualcomm receivers) can decode navigation messages [30] depending on the constellation. After being tested in the Mi8, only GPS and GLONASS navigation were present. It is because signals containing the navigation are only tracked until the ToW is decoded. Once done, the GNSS receiver switches to pilot signals, which does not contain the navigation message, leading to incomplete navigation messages for GALILEO constellation. Therefore, it is mandatory to perform multi-constellation computations to acquire the navigation messages differently. It is usually done through the Internet, using either the Secure User-Plan Location (SUPL) Google service or another Real-Time Service like IGS-RTS. Both provide the ephemeris information sent in the navigation message of a system. As explained in section 1.2, the application was decoding the GPS navigation message to compute a position in its initial state, and one of these solutions had to be implemented in the app support GALILEO satellites. Details on real-time ephemeris retrieval are given in chapter 2 and explain the choices made for the application.

1.4.3 Satellites' tracking and signal noise

Another issue with Android device, more related to the design of the GNSS chipset, is the tracking capabilities of the receiver. While lots of signals tend to be visible, some of them are not correctly tracked and can not be used in computations. These measurements usually have a very low C/n0 (i.e., Carrier-to-noise ratio) and do not have there ToW decoded. Note that we will be referring here at the C/n0 values and not SNR values because only the former are available in the Xiaomi GNSS measurements. Since they are closely related to each other, this does not impact our computations nor our analysis. Furthermore, sometimes, a signal will tend to disappear, and another one will start at the same time. This can be seen in the figure 1.3 where the satellite E4 from GALILEO constellation stop being tracked on its E5a signal and the E5a signal from satellite E24 start being tracked, then stop again. Since there is no way to control these cut in observations from Google API, the code needs to adapt these changes. However, losing track of a satellite's signal means that we can not converge anymore on the phase ambiguity estimation of this signal, and this particularly crucial for Precise Point Positioning. Possible explanations for this signals' lost can be either an efficient use by the receiver of its canals, which decides to track the satellites with the bests C/n0, or only due to the poor antenna inside the phone. As pointed out by multiple researches, the GNSS antenna inside a phone has terrible properties compare to a geodesic grade antenna, leading it to be more subject to cycle-slips (e.g., figure 1.2) or multipath. Signals also tend to have a much weaker C/n0. The positioning algorithm needs to account for these issues with discarding wrong measurements, especially for real-time and dynamic positioning matters. In figure 1.4 is represented the evolution of the C/n0 values for the satellites received by the Xiaomi Mi8. We can see that the BEIDOU satellites tend to have weaker signals than the GALILEO and GPS constellation. Also, less signal from the L5 band are received, which is expected for GPS constellation since not all satellites send them, but all GALILEO satellites send E5a signals. However, they tend to have better C/n0 than the L1 signals when received.

The measurement's noise on Android devices are not directly related to the elevation of a satellite, as it is with geodesic grade receivers (figure 1.5). Therefore, it is important to use weighting matrix based on the SNR or C/n0 value of the signal, like it is done for positioning in challenging environments [45] and will be detailed in section 3.2.4. Because of that, satellites with a low C/n0, i.e. below 25 dB-Hz, need to be removed completely to enhance the positioning [29]. The same goes for satellites below 15° elevation.

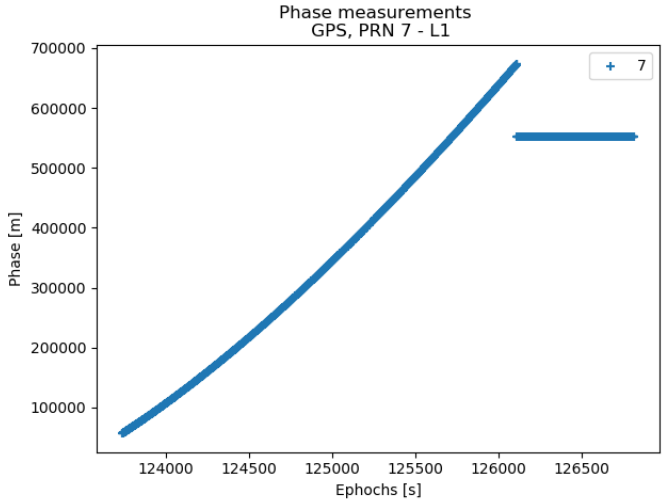


Figure 1.2 – Example of cycle slip in phase measurement.

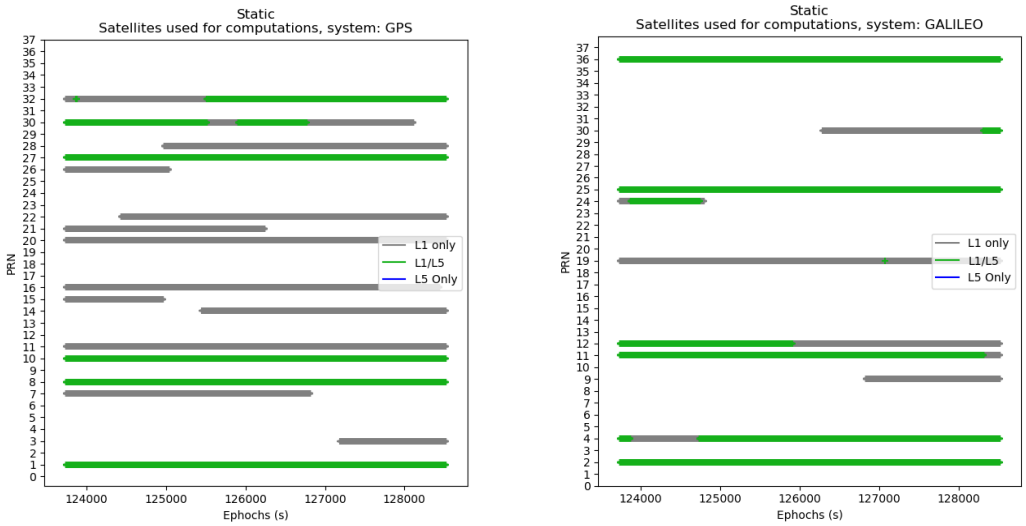


Figure 1.3 – Example of satellites and signals visible during an observation (26/08/2019).

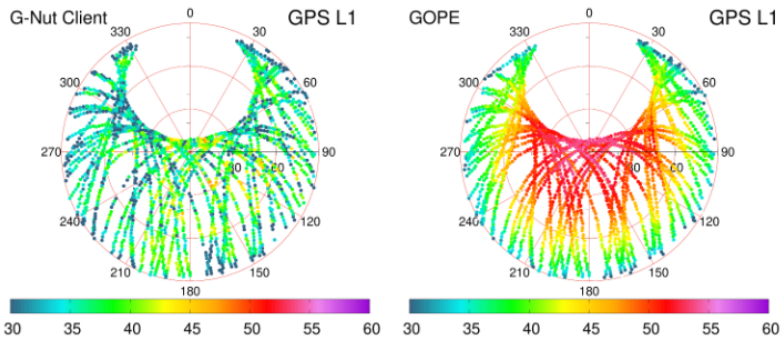


Figure 1.5 – C/n0 received by a Xiaomi Mi8 compared to a geodesic grade receiver [41].

The effect of the human body on the signals received by the phone is also an important issue related to Android devices. During testing of the position computations, the phone was set on

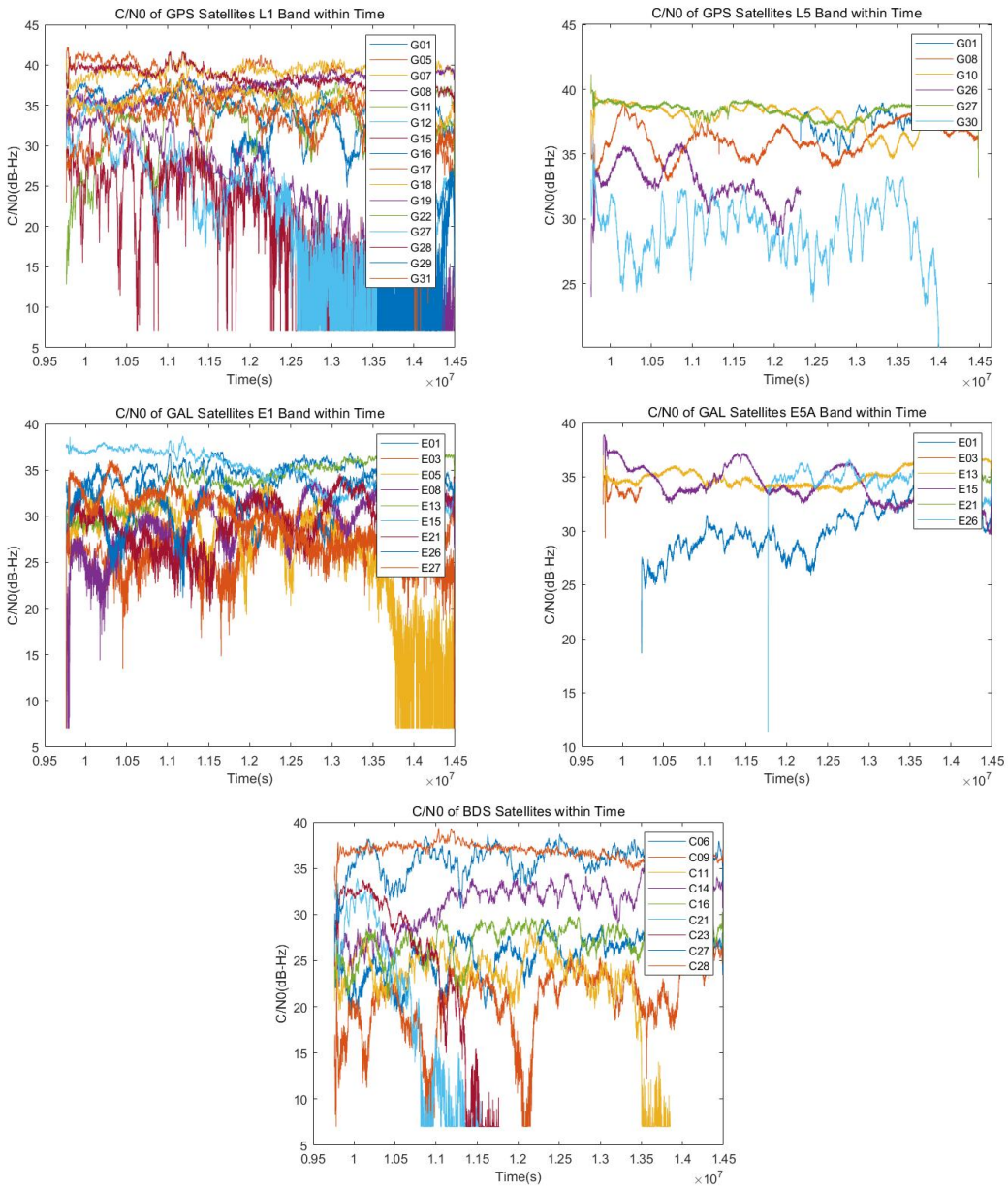


Figure 1.4 – C/n0 values observed by the Xiaomi Mi8 (Credits: Yazheng Wei).

an open-sky environment on pillar, where no significant mask was present, allowing us to test the capacities of the phone. However, the human body is a large mask for radio frequencies. Phones being used in a user's hand, it means that a large portion of the sky will be masked, impacting the positioning capacities. It has been observed in certain data sets, where error spikes are present at the beginning and the end of the positioning. These spikes are directly related to the operator being near the phone. The study of this phenomenon was not pursued because of lack of time, but future development should focus on this issue for correct handling of it. A possible way would be the development of measurement weighting model specifically designed for smartphones GNSS antenna.

1.5 Development with Android devices

Before this internship, I had basic knowledge and only a little experience of the Android development world. The same goes for Java, one of the native languages of Android, although I had some experience in object-oriented languages (C++, Python). Therefore, I had to learn the design

of an Android application, starting with the tools used for efficient development. The easiest way to develop Android applications is to use *Android Studio*, an IDE developed by Google, which help to compile the many files needed for an Android project. However, it is not made for analysis of results, as it is not suited to display graphs, which is why for results analysis I decided to develop a Python library to analyze the files outputted from the app.

1.5.1 Android development

Android applications are a compilation of different languages, not only Java. Indeed, Java is the core of the app and will deal with the computations, but also the Graphical User Interface (GUI). The GUI of an app is divided in code called *Fragments*, which divide for example the *Monitor* tab from the *Map* tab. Event handling is coded in Java, but the design is coded in XML language. Android Studio integrates a designer to build these XML files more easily, limiting the need to code all the interfaces in XML directly. To compile those languages together (*APK*), Android programming require to use a third language called *Gradle*. This language will define the resources and libraries needed for our app and "articulate" the compilation, similarly to what the *make* language does for C/C++ programming.

With Android Studio, all of these languages are taken care off, and once the files and the properties of the project are correctly defined, compilation on an Android device can be performed. When this is done, the app is installed on the device, which can be launched later. Android Studio also allows debugging in real-time of an app launched on a connected device, for step-by-step debugging. It is mandatory for efficient programming, especially when we are dealing with events and computations happening in real-time like it is done in GeolocPVT. However, for debugging of a GNSS positioning algorithm, it is better to acquire the data in the right environment (open-sky, urban canyon,...) and to replay the data inside the same code later, since we cannot receive those signals inside a building. Android Studio also allows this, and this is how the algorithms that will be developed in the next sections have been verified.

Several adjustments have been made to app GUI and organization during this internship, which are not presented in depth during this report. In summary, the following important adjustments have been made:

- Separation between computations and GUI has been made. All computations related classes have been added to a separate Android library called *Geoloclib*, to pass it more easily to other future apps;
- Some part of the code as been added to the *GoGPS* library, since the library contained bugs and has been enhanced to answer our purposes;
- Enhancement of all the fragments design to support the new functionalities of the app.;
- Extraction of RINEX are now possible in the app, for post-processing of the observations with other software.

In appendix A, a simplified diagram class can be found, representing the major classes of the app. Because of the large number of dependencies, they are not represented in the diagram to make it easier to read. Classes in grey are preexisting classes that were subject to no significant updates, whereas blue classes have been received substantial updates to enhance their capacities. Green classes are new classes added during this internship.

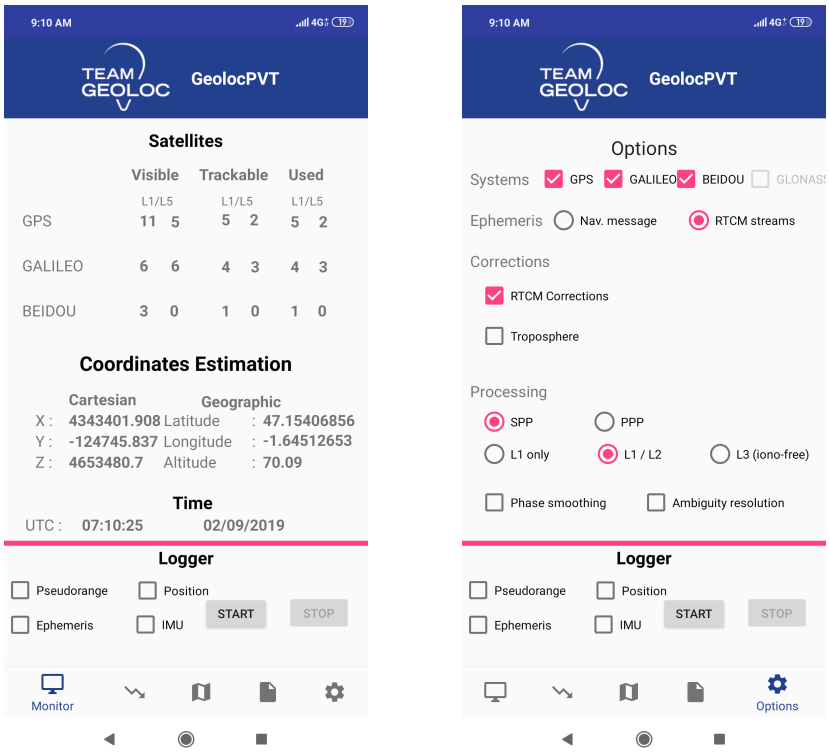


Figure 1.6 – Screenshots of the new app version (V2.1).

1.5.2 Python analysis tool

Files containing the observations used for the computations, but also the ephemeris and the precise corrections retrieved, can be extracted for each survey with the phone. To analyze the data, I developed a visualization library in Python language, the same way the GNSS Analysis Tool from Google analyzes the data outputted by the GNSS Logger application. This code is adapted to the custom file format of the app. Python code also has been used with the GNSS Toolbox library developed by Jacques Beilin (ENSG) and used for verification of specific parts of the code.

Chapter 2 Retrieval of precise orbits and clocks

This chapter presents the different precise products available in real-time and how to retrieve them. Details are given on the RTCM protocol, which is used by the IGS-RTS to send the precise corrections. Finally, the implementation of the real-time streams inside the GeolocPVT application is explained.

2.1 Real-time precise products

2.1.1 Ultra-rapid products

Precise products are diffused by the IGS (International GNSS Service), with different accuracy depending on the observation date. *Final* products, which give the best accuracy (about 2.5cm on orbits/75ps on satellite clocks), are only available after 15 days. For real-time purposes, earlier products need to be used, like *ultra-rapid* products. They are predicted, meaning they are based on previous observations and not the current ones, and are available in real-time but less precise (about 5cm/3ns).

Those products are available through the FTP servers of the IGS network in SP3 format. Files contain a position for each satellite of the constellation assigned to a timestamp in UTC, with a sampling rate of 15min. When one wants to compute the satellites' positions for a specific time, a 10th-order polynomial interpolation function is usually applied [2]. While this might work correctly for satellite positions, the interpolation process is not recommended for satellite clocks, also contained in ultra-rapid SP3 file, but with a sampling rate of 15min as well. According to ESA [2], products with a low sampling rate (i.e., higher than 30s) should not be interpolated because their evolution corresponds to a random walk process. This highlights an issue in using IGS ultra-rapid products for real-time PPP, since precise clocks estimation is mandatory for precise positioning.

Finally, it is essential to notice that those files are only available for GPS satellites and no other constellations on the IGS servers. As mentioned before, multiple constellation is the key for PPP, especially in constrained environments like urban canyon, which is where a smartphone is most likely to operate. This emphasizes another problem with ultra-rapid products, which is their availability for all constellations.

2.1.2 Real-Time Service and RTCM protocol

Since 2001 [23], IGS has also been developing another service called Real-Time Service (RTS). Its goal is to diffuse real-time products to users to perform real-time positioning. Joining the RTCM, they developed the State Space Representation (SSR) standard, enabling the diffusion of orbits and clocks corrections through the Internet using bytes streams.

The RTCM [1] is an international non profit scientific organization which develop standards for maritime radio-navigation and radio-communication systems. The RTCM protocol includes standards for encoding and sending DGNSS data. The protocol divides the data in several messages, which can be identified by an ID, representing what the message contents is. At first developed for GPS/GLONASS only, the latest version (10403.3) developed in 2016 now contains messages

1. http://www.rtcm.org/about.html

for GALILEO, BEIDOU and QZSS systems. For example, GPS constellation corrections are in the messages 1057 to 1062 [38].

— Message 1057 - SSR GPS Orbit Correction
— Message 1058 - SSR GPS Clock Correction
— Message 1059 - GPS Code Bias
— Message 1060 - SSR GPS Combined Orbit and Clock Corrections
— Message 1061 - SSR GPS URA
— Message 1062 - SSR GPS High Rate Clock Correction

Similar messages with different IDs can be found for other constellations: GLONASS (1063-1068), GALILEO (1240-1245), QZSS (1246-1251) and BDS (1258-1263). The RTCM standards version 10403.3 [37] details the contents of each message. For precise positioning, we are interested in the broadcast ephemeris containing the orbits/clocks parameters :

— Message 1019 - GPS broadcast ephemeris;
— Message 1042 - BEIDOU broadcast ephemeris;
— Message 1045 - GALILEO broadcast ephemeris;
— Message 1060 - GPS precise corrections;
— Message 1243 - GALILEO precise corrections;
— Message 1249 - BEIDOU precise corrections;

The IGS-RTS requires special access to retrieve those streams, and as for now, only research purposes projects are granted access. For this study, parallel access to three streams was granted to the IGS server “rt.igs.org".

The IGS-RTS offers different server to be connected to, each one offering a list of *mount points* (i.e., casters). Each of those corresponds either to a permanent base site or a processing center. They will send certain types of message, with specific rates, depending on the mount points specifications. It means that not all streams contain the same messages, therefore not all constellations either. Using the BKG software "BKG NTRIP Client" or BNC [5], an open sourced NTRIP software, one can look into the content of the stream sent by each point, to find the most suitable stream to receive the desired corrections.

2.1.3 Benefits of the IGS-RTS

By using this service, it becomes possible to retrieve the orbits and clocks information of all systems, depending on which mount points the device is connected to. According to Wang et al.[42], the mount point CLK93 from the CNES (Centre National d'Etude Spatiale, France) is one of the few mount point sending corrections for GPS, GLONASS, GALILEO and BEIDOU constellations. It has been verified with the introduction of these streams in the GeolocPVT app, to retrieve GPS and GALILEO corrections using this mount point.

Another benefit of using the streams corrections is the precision obtained for satellite positions and clocks. IGS-RTS products are as accurate as the predicted half of the ultra-rapid products [8] while offering satellite clocks corrections with a much higher sampling rate. It depends on the mount points specifications. A five seconds rate is standard for clocks corrections to avoid inaccuracies inherited with longer interpolation [23]. This is a considerable improvement compared to the ultra-rapid products, satellites clocks being one of the most significant error source in positioning that is not canceled out in PPP as it would be in DGNSS.

Note that we will focus only on the integration of those streams in the app and how precise corrections can be extracted from the RTCM streams. However, I will not talk about the Internet data consumption using those streams lead to, which is a problem I also studied. To answer this, I wrote an article in June to be published at the WNPC 2019 conference, which can be found in Appendix C of this report.

2.2 Computation of the precise orbits and clocks

2.2.1 Satellites coordinates from broadcast ephemeris

The same algorithm that one uses to compute satellite positions from the navigation message can be used here with those messages; details can be found in the appendix C.2 of the ESA open book [2]. The computations of the satellite's coordinates using broadcast elements were already coded by [35] in the GeolocPVT application when I arrived, since the navigation data was retrieved from the navigation message inside the app. However, I had to adapt the code so that GALILEO satellites could be computed too and that orbital parameters could also be set up from the streams and not only from navigation messages.

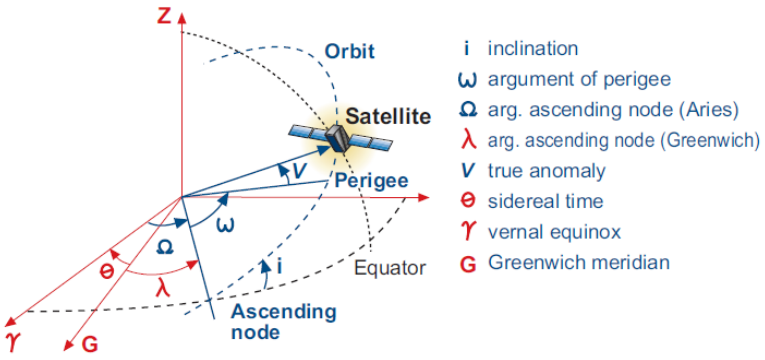


Figure 2.1 – GNSS satellites keplerian orbital elements [2]

2.2.2 Applying the corrections

Precise corrections to the satellite orbits and clocks are given in the satellite body referential and are designed to be applied directly on the positions computed by the broadcast ephemeris.

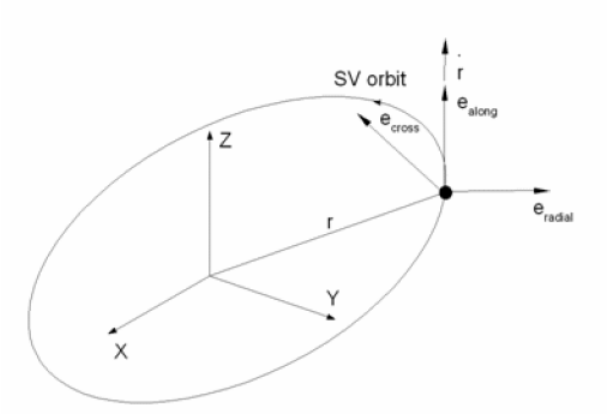


Figure 2.2 – Orbit corrections applied to satellite coordinates [38].

Similar to the one from navigation message, streamed ephemeris and corrections are marked by an ID called Issue of Data (IoD). This information is particularly important when one wants to apply the streamed correction to satellite positions. Indeed, IoDs need to be the same between the ephemeris and the correction message for the corrections to make sense and improve the accuracy of the satellite's position. To apply the received corrections, the reference frame first needs to be changed. Orbital corrections are given in satellite body frame (e_{radial}, e_{along}, e_{cross}) and need to be rotated to ECEF reference frame, where our satellites coordinates are. The following equations are extracted from [38].

$$X_{orbit} = X_{broadcast} - \delta X \tag{2.1}$$

$$\delta X = \begin{bmatrix} e_{radial} & e_{along} & e_{cross} \end{bmatrix} \delta O \tag{2.2}$$

Where:
δX is the corrections in the ECEF reference frame;
$X_{broadcast}$ is the satellite coordinates in the ECEF frame computed from the broadcast;

Note that the coordinates and the corrections are directly related to the position of the ionosphere-free combination phase center of the satellite's antenna.

$$e_{along} = \frac{\dot{r}}{|\dot{r}|} \quad e_{cros} = \frac{r \times \dot{r}}{|r \times \dot{r}|} \quad e_{radial} = e_{along} \times e_{cross} \tag{2.3}$$

Once the satellite positions are computed and the corrections are applied, precise coordinates are obtained and can then be used in the rest of the computations.

2.3 Implementation in the application

Regarding the implementation of stream retrieval in the app, this is done through the Java library GoGPS, an open-sourced code licensed under the GPL and available on Github [2]. It is an adaptation of a MATLAB version developed by the Geomatics Laboratory of Politecnico di Milano, Como campus (Italy), under the supervision of Dr. Mirko Reguzzoni, as a Master thesis project (Dominioni, Teruzzi). This code was developed for positioning using GPS only. Even if it is not designed as an Android app, it can be used as a library, since our app GeolocPVT is developed in Java too. For that, I transformed the Java library into an Android library (or module) using Android Studio. It gives us access to an extensive toolbox with a lot of already coded functions, including the ones to connect to a mount point sending RTCM streams and decode them. The only problem is that the functions need to be generalized and adapt for all the wanted systems, not only GPS, and in our case it means for GALILEO too.

The first part of my work after branching the library and add it to the GeolocPVT code was to understand the GoGPS code, which does not include any external documentation, how to use it and improve it. After a successful connection to a mount point, decoding capacities were enhanced to parse all the wanted RTCM messages which were not present in GoGPS.

2.3.1 Connection to mount points

The main benefit of GoGPS in our case is the handling of the connections to the servers. This is performed by the *RTCM3Client* class. GoGPS contains an interface called *StreamEventListener*, which can be implemented into a class for event handling of a new stream being received. However, even if the classes are all set up, no working example was found in the library. A new class needs to be created to make use of those different objects. Since we want the streams to be retrieved in the background, this custom class will need to create a new thread in the application. *RTCMRunnable* class is therefore created, implementing the *Runnable* interface of Java, for thread handling. We also implement the *StreamEventListener* interface in our own class called *StreamEphmerisHandler*. Then everything is wrapped inside a class called *Streams*.

The creation of the connection is executed in this order: once the app is started, the application will try to connect to the streams given the parameters set up in the user options. By instantiating a new object of the *Streams* class, it will create a thread for each stream we want to connect to, meaning a new *RTCMRunnable* object for each stream. When a *RTCMRunnable* object is created,

2. https://github.com/goGPS-Project/goGPS_Java

it will create a new *RTCM3Client* object and fill it with the parameters of the caster (e.g., server address, credentials, mount point). To this *RTCM3Client* will be attached a *StreamEphmerisHandler*, a listener which will be called each time a new stream is received and will handle its parsing. Contents will be saved in the appropriate classes (see section 2.3.2).

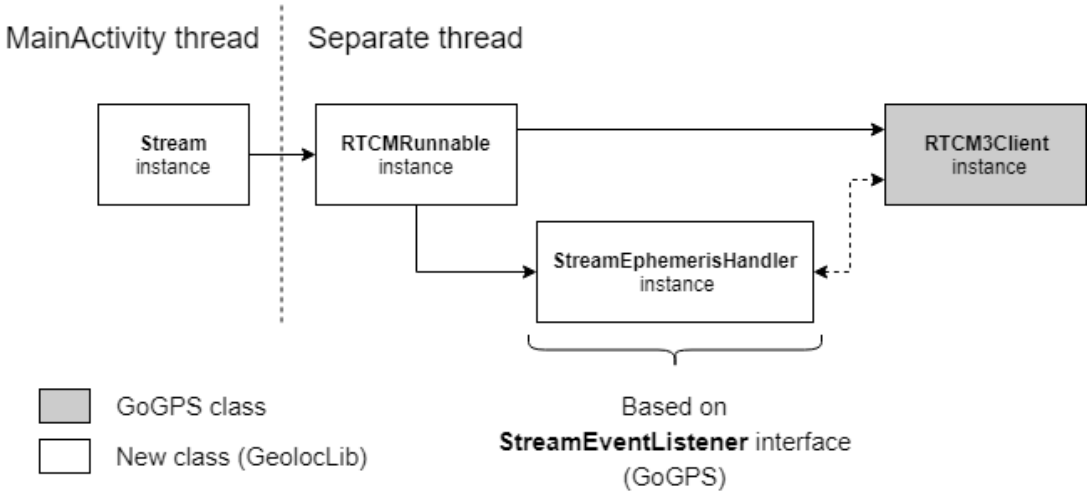


Figure 2.3 – Diagram summarizing the creation of a new stream handler.
Gray blocs represent GoGPS native classes, enhanced for new message handling. White blocs are newly created classes for GeolocLib.

The table 2.1 shows the mount point's parameters used during the development. We are interested in retrieving the ephemerises and the corrections to it. We will connect to two mount points of the IGS-RTS: "RTCM3EPH" for the ephemerises and "CLK93" for the corrections. Note that the "RTCM3EPH" was not available to the server at the beginning of this internship and only appeared in June in the mount points server's list. Before that, I used a stream from Toulouse called "TLSG00FRA0", which was sending only the satellites in view in Toulouse and with a longer sending rate. It would usually take 30 seconds before the app received all the satellites, compared to 5 seconds with the new "RTCM3EPH" mount point.

Mountpoints	Host	Port	Organization	Messages used
RTCM3EPH	rt.igs.org	2101	BKG	1019,1042,1045
CLK93	rt.igs.org	2101	CNES	1060,1243,1249,1059,1242,1260

Table 2.1 – Mount points used and the messages used from them.

2.3.2 Decoding of the streams

Once a stream is retrieved, it is in binary format and needs to be decoded before being usable. From looking at the GoGPS source code, we can see that GoGPS was not developed for decoding of ephemeris data, but rather to receive observation data from distances bases and log it. In the RTCM protocol, observation data is sent in messages 1004 (GPS) and 1012 (GLONASS), which explain why only those functions are coded in GoGPS. Since we are interested in other messages (see section 2.1.2), we need to implement the decoding class that goes with it. The encoding of the message is explained in the RTCM standards [37], the same way the GPS and GALILEO ICD's describe how is encoded the navigation message. It follows the underlying logic of electronics' bytes decoding, which is not interesting to explain in-depth here. For detailed information, please refer to the following reference [40]. It is important to note that a large part of the RTCM's messages decoding is already implemented in the well-known RTKLIB C/C++ open-source library. Since C/C++ code can be wrapped in Java and Android apps, I first thought about wrapping the RTKLIB

library in GeolocPVT's Java code, to use its decoding capacities at first, but also for later purposes when PPP functions will be developed. RTKLIB already had all the PPP functions wanted for the app and using them would allow interoperability of our code and certitude that our computations are correct, since the RTKLIB is developed and tested by a large community around the world. Yet, wrapping and compilation of C/C++ sources in Android Studio is not trivial. Therefore, after encountering diverse compilation issues, I decided that the integration of RTKLIB would require too much time and instead needed functions could be re-coded in Java directly, expanding the GoGPS library. Thus, using RTCM standards [37] and the RTKLIB source code [3], the decoding functions were coded in the app and add to the GoGPS library.

GoGPS handles the decoding of the messages by creating a new class for each message, all implementing the interface *Decode* and then linking the message number to the right class in a *Map* [4] object defined inside the *RTCM3Client* class. When a message is received, it will decode the header, compare this number to the keys inside the Map, and use the proper decoding function for this message. In total, nine classes were added throughout the internship, three for each GNSS constellations used (GPS, GALILEO, BEIDOU). The ephemeris precise corrections' messages 1060, 1043 and 1069 are all encoded the same way, with a few changes depending on the length of the header for each constellation. However, the messages 1019, 1042, and 1045 related to ephemeris data have differences in their contents. The RTCM standards manual is then mandatory to account for these few changes in encoding [37]. To verify if the streams were decoded correctly, I used the program SNIP [5] and also the broadcast ephemeris logged by the IGS. The last three messages concerns Differential Code Bias (DCB) corrections and will be explained later in section 4.2.2.

Decoded ephemeris data is placed inside the class *GNSSEphemeris* and corrections data are placed inside the *GNSSEphmerisCorrections*, containing several *PreciseCorrection* object, one for each corrected satellites. Since every new stream event is happening on his own, depending on the mount point specifications, nothing is synchronized, and nothing needs to be until the position computation. This is why I decided to implement every connection to mount point in its own thread, as we explained in the previous section. Every part of the app is designed that way, and the only thing linking them all together is the new raw GNSS measurement event, which happens at a 1Hz rate. It is this event that will launch the computation of a new solution and synchronizes the data retrieve from streams with the raw GNSS measurement. This is why every new stream received we will replace the previous stream data with the new stream data, discarding the previous data because it is inevitably outdated and not relevant anymore in our current epoch. This way, we do not accumulate data, and we do not have to synchronize the computations, only the relevant data will be there for the next new GNSS measurement event. When this event comes, we simply "request" the last received data which is contained in our *Stream* object in our *MainActivity* thread and computed the current position of the user with it, computing the satellites current position and applying the precise correction at this moment only and not before.

3. https://github.com/tomojitakasu/RTKLIB
4. A Map object in Java associate a "key" to a "value". Here the key is the message number. The value is an empty instance of the correct class.
5. https://www.use-snip.com/

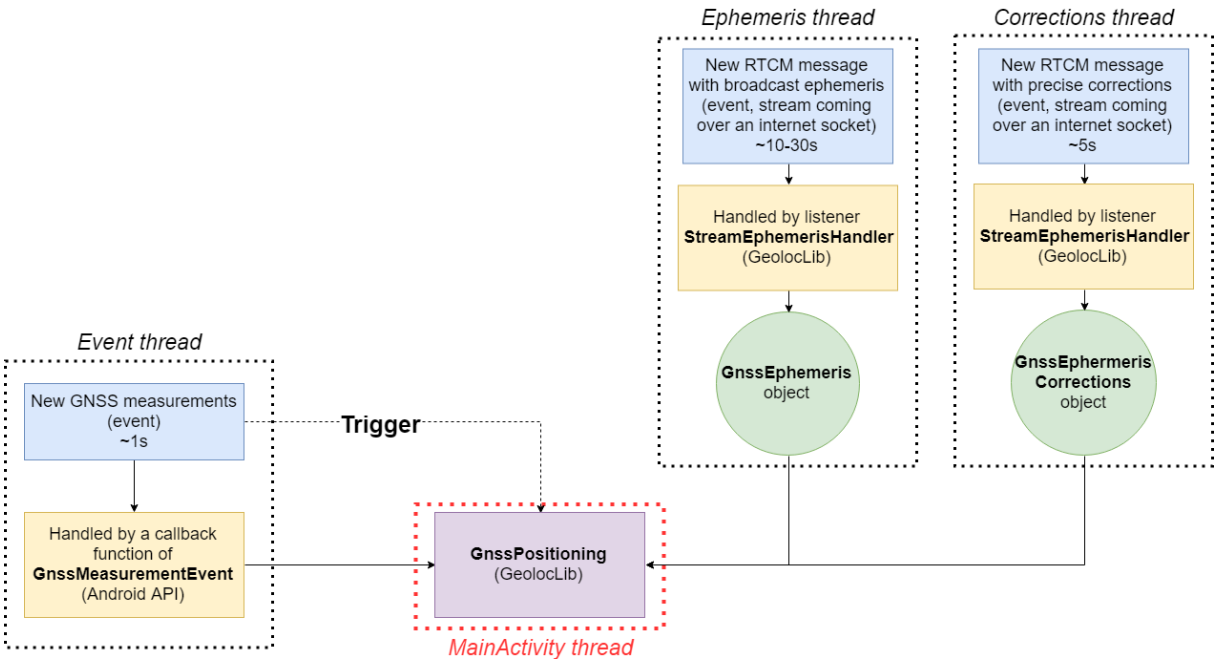


Figure 2.4 – Diagram representing the app's architecture.

2.3.3 Precise corrections verification

The app's module used for computation of the satellite coordinates using the ephemeris parameters available in navigation messages or RTCM streams was developed during the previous internship [35] and already coded when I arrived. However, by reading the report of this internship, I found that satellite coordinates have not been adequately checked, since only verified up to the first decimal of decimal degrees using the website `http://in-the-sky.org`. Because I was looking for a possible explanation for the offset found in the computations' results (see section 3.4), I decided to redo this positioning module and properly verify it, using for reference the Python GNSS Toolbox developed by Jacques Beilin. I designed a unit test in the GeolocPVT code so that given ephemeris parameters and specific time, I could compare the results to one computed by the Python library. This way, I confirmed that the new version of the module correctly calculated the satellite coordinates. The offset problem was not solved in the user position, but at least this part of the code is now verified.

For verification of the precise corrections, the SNIP software was used, which can display the contents of a RTCM stream. Correct decoding of the binary are therefore verifies. The second test performed was checking the norm of the corrections. As we said earlier, precise corrections are given in the satellite body-frame and need to be converted to the ECEF frame to be applied on the satellite coordinates. By verifying that the norm of the correction vector is the same after the change of system, we make sure that the precise corrections obtained in ECEF are correct. For verification of the precise satellite coordinates, I compared the satellites positions computed by the phone in real-time to the SP3 products of the IGS. For this, the Python GNSS Toolbox was used again, which can read SP3 files and interpolates between the positions written in the files. The results are shown in figure 2.5, and we can see that significant errors are still present in it. However, looking at the norm of these errors in figure 2.6, it seems to be constant. A possible explanation for these large errors in satellite positioning could be due to the computation of emission time. Indeed, we have seen before that the algorithm has been verified and work correctly, but the difference between the test I did and the real-time computations performed on the phone is which emission time is inputted in the algorithm. This is why I think this error is time-related. As said in section 1.3.1, retrieving the exact reception time is not trivial, and several offsets need to be accounted for. However, this time is also needed to compute the corrections received from the streams. After verification of the

emission time computation, I could not find the error in the code.

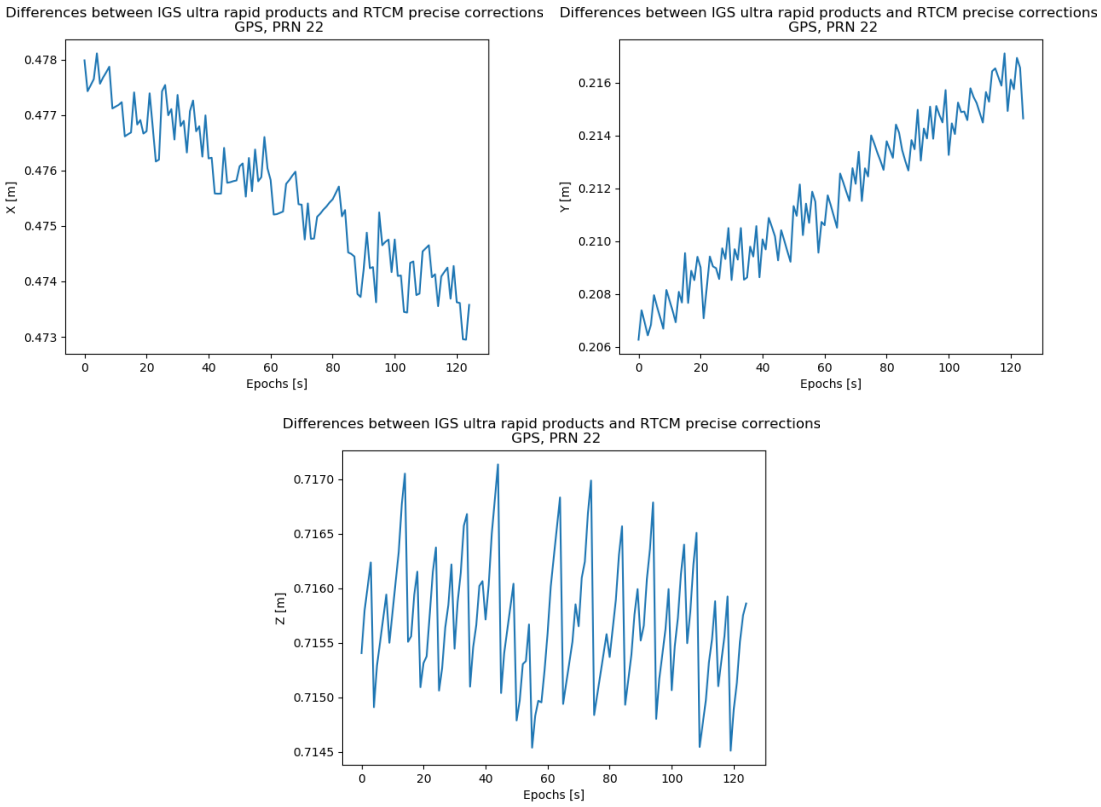


Figure 2.5 – Differences between the satellite's coordinates from ultra rapids orbits and RTCM corrections.

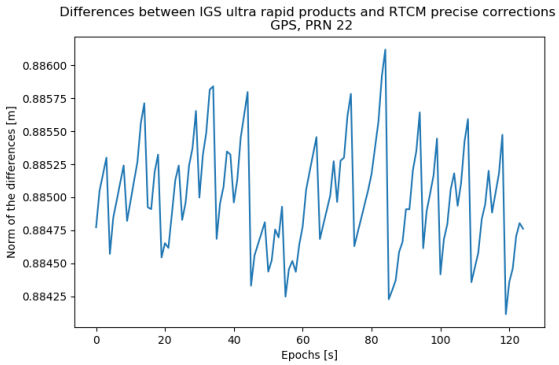


Figure 2.6 – Differences between the norm of the satellite's coordinate vector from ultra rapids orbits and RTCM corrections.

In summary, satellite position computations from orbital parameters and precise corrections received from the stream have been correctly verified. However, an error up to 1 meter on the satellites positions still exist. After some verification, everything seems to be linked to an error in the emission time computations. While this is not very important for SPP computations, it might be the reason for the offset I found in the computations presented in section 3.4. When the PPP algorithm is developed in the phone, attention should be given on this error, as the satellites positions should be precise at the centimeter-level.

Chapter 3 Standard Point Positioning

This chapter presents the Standard Point Positioning technique and the modeling of GNSS observations used in the app's computations. The Kalman filter model is also detailed for static and dynamic positioning.

3.1 Technique definition

Standard Point Positioning (SPP) is the most basic positioning technique. It is a trilateration method, relying on one receiver's observations only, using the pseudorange measurements to compute the user coordinates and receiver clock bias. All errors in the signals are either negated by signal combination using multiple frequency or modeled. Since it does not need to estimate the phase measurement ambiguity, it is easier to implement than PPP but give a worse precision. Yet it is a mandatory step since it allow to assess the receiver capacities and to set up several other important part of our algorithm, like the cycle and multipath detection algorithms, and the C/N0-Elevation weighting, which will be useful for our PPP computations.

3.1.1 Measurements modeling

Pseudoranges and phases measurements can be modeled as followed [2]:

$$\begin{aligned} P_f^j &= \rho^j + c \cdot (\delta t_r - \delta t^j) + \tau_{iono}^j + \tau_{tropo}^j + \epsilon \\ \Phi_f^j &= \rho^j + c \cdot (\delta t_r - \delta t^j) - \tau_{iono}^j + \tau_{tropo}^j + \lambda_f \cdot N_f^j + \epsilon \end{aligned} \tag{3.1}$$

Where:
P_f^j is the pseudorange measurement of the j^{th} satellite of frequency f;
Φ_f^j is the phase measurement of the j^{th} satellite of frequency f ($\Phi_f[m] = \lambda_f \cdot \phi_f[cycle]$);
ρ^j is the geometric distance between the satellite and the receiver coordinates;
c is the speed of light constant, e.g. $c = 299792458$ m/s
δt_r is the receiver clock error from the GNSS time scale;
δt^j is the satellite clock error from the GNSS time scale;
τ_{iono}^j is the error due to propagation in the ionosphere layer;
τ_{tropo}^j is the error due to propagation in the troposphere layer;
N_f^j is the carrier-phase ambiguity of the carrier signal with frequency f;
ϵ account for several errors, including multipath and noise.

3.1.2 Carrier-phase smoothing of pseudorange

The pseudoranges measurements are absolute distance measurements that allow us to compute a position easily using a trilateration algorithm (with least square for example). However, they are noisy measurements that lead to significant errors in our computations, leading to jumps between a position computed at epoch k compared to the one computed at epoch $k-1$ if no filtering algorithm is used (e.g., simple Weighted Least Square). On the contrary, the carrier phase measurements are

less noisy but ambiguous, meaning that we need to resolve the carrier-phase ambiguity if we want to use it as an absolute distance measurement. This process is called *ambiguity estimation*; it is the difference between a SPP and a PPP algorithm and will be discussed in chapter 5. With a geodesic-grade antenna and if no cycle slip occurs, the noise of phase measurements can be estimated as $\sigma_\Phi = 0.001$ m, whereas pseudorange measurements precision are only about $\sigma_\Phi = 3$ m. Figure 3.1 represents the pseudorange and phase rate between two epochs. They are computed by temporal differentiation of the measurements to compare them. It clearly shows the precision of the phase compare to the pseudorange.

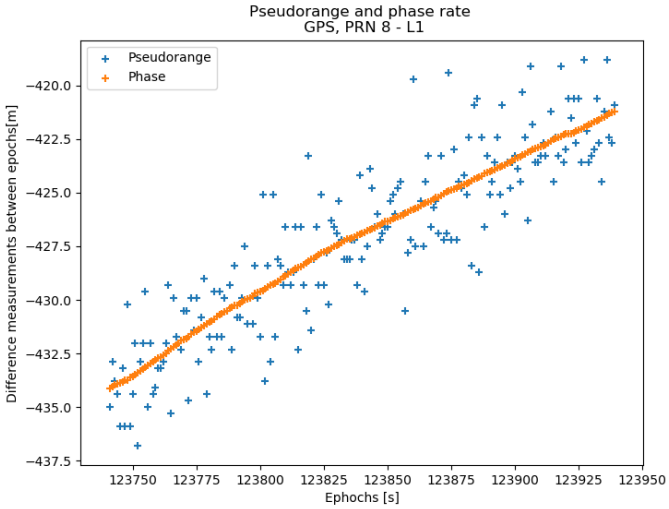


Figure 3.1 – Pseudorange and phase rate comparison.

A carrier-phase smoothing algorithm is a process that takes advantage of the absolute character of pseudoranges and the carrier-phase measurement precision by merging the two measurements along time. Using equation 3.2 [2], it is possible to reduce the noise of pseudorange measurements, giving more and more weight to the phase measurement updates along time.

$$\hat{P}_k^j = \frac{1}{n} \cdot P_k^j + \frac{n-1}{n}\left(\hat{P}_{k-1}^j + (\Phi_k^j - \Phi_{k-1}^j)\right) \quad \text{if } k < N, n = k$$
$$\text{else}, n = N \tag{3.2}$$

This technique is well-known and was presented again at the GSA workshop I attended, in order to improve the pseudorange quality that are particularly noisier on smartphones due to the bad quality of the antenna (see section 5.2.4). Since it showed good results, I decided to implement it inside GeolocPVT.

After a short convergence time, depending on the weight given to the phase measurements and the integration time, the noise of the measurement is significantly reduced, leading to a more precise positioning. This smoothing can be performed on L1 and L5 measurements separately, but the ionosphere effects can lead to divergence in the solution if measurements are integrated for too long. Since we are computing the L3 (ionosphere-free) combination, we can perform smoothing directly on those measurements instead, to avoid this divergence effect. Another way is to compute the Divergence-Free Smoother since we have dual-frequency measurements available. However, this last solution is more useful for static receivers with observations time longer than 12 hours [2], which is not our case since we are dealing with observations time smaller than 1 hour. Assumption can be made that this phenomenon does not impact us.

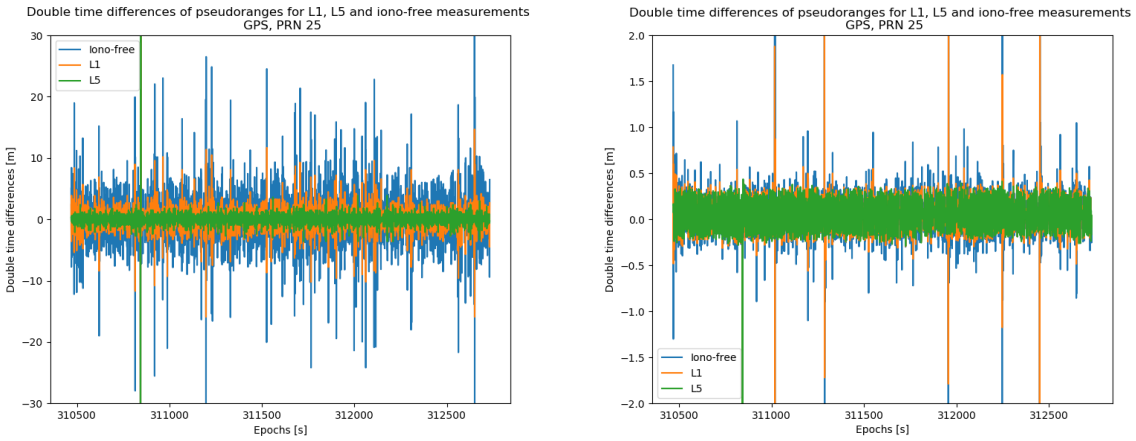


Figure 3.2 – Triple differences of pseudoranges for L1, L5 and L3 measurements, compare to their smoothed version.

	Raw pseudorange			Smoothed pseudorange		
Statistics	L1 [m]	L5 [m]	L3 [m]	L1 [m]	L5 [m]	L3 [m]
Standard deviation (1 σ)	2.791	1.936	5.453	0.223	0.151	0.395

Table 3.1 – Statistics for comparison of single epoch positioning and kalman filter in Static mode.

Figure 3.2 shows double time-differenced of pseudorange measurements. It allows keeping just the noise of the measurements, with no trends in the data for easy analysis of the algorithm performances. Note that the graph's scale containing the smoothed pseudoranges had to be adapted because of the good precision of the pseudoranges now. We can also notice than in the raw pseudorange graph (left), L1, and L5 measurements have a different precision, L5 measurements tend to be less noisy and more resilient to multipath [36].

3.1.3 Time-Differenced Carrier-Phase

Another way to use the carrier-phases measurements without resolution of the ambiguity is for user velocity estimation. Using a precise velocity estimation, we can apply constraints on our trajectory and our positions computed from our pseudoranges.

In the Google API, multiple measurements can be retrieved at each epoch, like phase or doppler measurements. Both can be used for velocity estimation, but Doppler is less precise than phase measurement [13]. This is why if we decide to use the phase measurements, it is not interesting to use the Doppler measurements. Therefore it has been decided to not retrieve the Doppler measurements for computations in the app. The technique associated with the phase measurements is called Time Differenced Carrier Phase (TDCP). This technique is well-known in the Geoloc laboratory and is used in other positioning algorithm they have, which led me to develop it inside GeolocPVT, where I wanted to use phase measurements in SPP computations. However, while doing researches on this technique, I found that the integration of TDCP for smartphone measurements has been performed in a study from Zhu et al. [26] published at the beginning of this year 2019. It comforted me in this decision to use TDCP inside GeolocPVT, since the study's results were great.

Time difference between phase measurements at epoch $k-1$ and k can be modeled as follows

$$\Delta\Phi^j = \dot{\rho}^j + c \cdot \Delta\delta\dot{t}_r - c \cdot \Delta\delta\dot{t}^j + \Delta\delta u^j - \Delta\tau^j_{iono} + \Delta\tau^j_{tropo} + \epsilon_{\Delta\Phi^j} \tag{3.3}$$

Since we are differentiating the phase measurements, we can assume here that the errors $\Delta\delta\dot{t}^j$, $\Delta\tau^j_{iono}$ and $\Delta\tau^j_{tropo}$ can be neglected, since our measurements rate is 1 Hz [13] [26]. We can therefore simplify the equation 3.3 and linearized it as equation 3.4 [26].

$$\Delta\Phi^j - \Delta D + \Delta g = -e_k \cdot \Delta r_u + c \cdot \Delta\delta\dot{t}_r + \epsilon \tag{3.4}$$

Where e is the user-satellite line of sight defined in equation 3.5

$$e^j = \frac{u_r - u^s}{||u_r - u^s||} = \frac{u_r - u^s}{\rho} \tag{3.5}$$

$$\Delta g = e_k \cdot r_{u,k-1} - e_{k-1} \cdot r_{u,j-1} \tag{3.6}$$

$$\Delta D = e_k \cdot r_{s,k} - e_{k-1} \cdot r_{s,j-1} \tag{3.7}$$

This part relates to the dynamic positioning algorithm, different from the static positioning first set up in the app. It means that the static algorithm does not make use of the TDCP and does not estimate velocity. It only uses the phase measurements for pseudoranges smoothing. See section 4.3 for the dynamic algorithm.

3.2 Corrections

3.2.1 Troposphere model

Two tropospheric models were implemented inside the GeolocPVT algorithm: the Saastamoinen model, based on averaged pressure, humidity and temperature values, and the model implemented in the GIPSY-OASIS II software, with an estimation of the wet part of the atmosphere as one of our system parameters.

The Saastamoinen model was used here for SPP computations. For integration of the model, I used the open-source code from the *GNSS Compare* application, developed by The Galfins in 2018 as part of an ESA competition [14]. Details on the equations can be found in the ESA Navipedia website [1] [10].

The other model does not require any atmospheric data either and is much more precise since it is based on the actual observations of the receiver. It was also developed but is more interesting for PPP purposes and not SPP [2]. Therefore, this model and its development are explained in chapter 5.

3.2.2 Ionosphere model

Correction of the ionosphere delays is essential for precise positioning and can be achieved either by modeling or combination of signals. The effects of the ionosphere are related to the signal frequency. Therefore when multiple frequencies are available on the receiver, this delay is usually canceled by creating the ionosphere-free combination.

Another way is to use ionosphere models, which are less precise but work with mono-frequency receivers, like the Klobuchar model. Parameters for this model can usually be found in the navigation messages of the signals. Details on how to compute this model are given in [2], p.116-121. As said in section 1.4.2, only GPS navigation messages are received, but the ionospheric parameters inside can be used for other systems (i.e., GALILEO and BEIDOU). Another way would be to retrieve those parameters from the RTCM streams, as they are sent under message 1264.

3.2.3 Signal combination

Because we are working with dual-frequency, it is possible to mitigate the ionosphere error by combining signals with different frequencies coming from the same satellite. This combination is called the "ionosphere-free" combination, also referred as $L3$.

1. https://gssc.esa.int/navipedia/index.php/Galileo_Tropospheric_Correction_Model

$$
\begin{aligned}
P_3^j &= \frac{f_{L1}^2}{f_{L1}^2 - f_{L5}^2} \cdot P_{L1}^j - \frac{f_{L1}^2}{f_{L1}^2 - f_{L5}^2} \cdot P_{L5}^j \\
\Phi_3^j &= \frac{f_{L1}^2}{f_{L1}^2 - f_{L5}^2} \cdot \Phi_{L1}^j - \frac{f_{L1}^2}{f_{L1}^2 - f_{L5}^2} \cdot \Phi_{L5}^j
\end{aligned}
\tag{3.8}
$$

Equation 3.1 then become

$$
\begin{aligned}
P_3^j &= \rho^j + c \cdot (\delta t_r - \delta t^j) + \tau_{tropo}^j + \epsilon \\
\Phi_3^j &= \rho^j + c \cdot (\delta t_r - \delta t^j) + \tau_{tropo}^j + B_3^j + \lambda_3 \cdot \omega^j + \epsilon
\end{aligned}
\tag{3.9}
$$

3.2.4 Measurements weighting

In GNSS positioning, measurements weighting is usually correlated to the satellite elevation, since higher satellites are less likely to have large errors due to the atmospheric delays. Other models exist, using the C/n0 or Signal-to-Noise Ratio (SNR) and is usually preferred for challenging environments, e.g., urban canyons. Mixed models also exist, with taking into account if the satellite is in sight of the receiver or not [45]. During the workshop in Prague about Android GNSS measurements, weighting of measurements using C/n0 values has shown to be the best one, as, in contrary to Geodesic grade antenna, quality of measurements are not correlated at all to the elevation of the satellite (see section 5.2.4). Therefore, I decided to integrate a model based on C/n0 values rather than elevation inside the application. The model most used in the studies [45, 26] is developed in equation 3.10.

$$
\sigma_{P^i} = a + b \cdot 10^{\frac{-C/n0^i}{10}} \tag{3.10}
$$

Where a and b are coefficients depending on the environment. For open-sky condition, these values are usually set to: $a = 10m^2$ and $b = 150^2m^2$ [26]. Ways exist to compute the appropriate values depending on the errors left in pseudoranges after correction [45] but demand more computation resources and time. Instead, some empirical testing of sets of values have been performed, but these turned out to be the best.The weighting matrix R can be built for each measurement using the σ_{P^j}.

$$
R^{-1} = \begin{bmatrix} (\sigma_P^1)^2 & & 0 \\ & \ddots & \\ 0 & & (\sigma_P^n)^2 \end{bmatrix}_{n_{sat} \times n_{sat}} \tag{3.11}
$$

3.3 Weighted Least Square Estimation

To solve this system, an excellent method to begin with is a simple Least Square Estimation. Our system being non-linear, it needs to be linearized, and *a priori* values need to be given. Note that for all developed algorithm here (least square or Kalman filter), we assume that at least one satellite of the three constellations (GPS, GALILEO, BEIDOU) is received in every epoch, and that inter-system time offset (GXTO) can be correctly estimated. If one constellation was missing, it would make the system unsolvable unless the corresponding time offset parameter is removed from the estimation. We are only developing here the system in its most complete form.

3.3.1 Linear model

The equation 3.1 can be linearized and put in a matrix format as $Y = HX$, where H is our observation (or design) matrix, Y is the observation vector and X our state or parameter vector. For

linearization, we proceed by doing Taylor's development on a small state perturbation. This is why we estimate dx, dy, dz rather than x, y, z directly. We need to add some other parameters to our model, referring to the inter-system time offset between GPS and GALILEO (GGTO) and between GPS and BEIDOU (GBTO). This offset exists because two systems are not perfectly synchronized with each other. While it is possible to predict this value with information contained in the navigation message [24], it is also possible to estimate it in our computations at the cost of one observation. This also means that if only one satellite from another constellation is visible, adding it to the system will not increase the redundancy of our system. Therefore at least two satellites of the other system need to be present in the observation. Since we are not receiving GALILEO nor BEIDOU navigation messages, we choose the second method and add two parameters to be estimated in our state vector: GGTO and GBTO.

$$X = \begin{bmatrix} dx & dy & dz & c \cdot dt_r & c \cdot GGTO & c \cdot GBTO \end{bmatrix}^T \tag{3.12}$$

$$Y = \begin{bmatrix} \vdots \\ P_i^{j,GPS} - \rho^{j,GPS} - c \cdot \delta t_r + c \cdot \delta t^{j,GPS} \\ \vdots \\ P_i^{j,GAL} - \rho^{j,GAL} - c \cdot \delta t_r + c \cdot \delta t^{j,GAL} - c \cdot GGTO \\ \vdots \\ P_i^{j,BDS} - \rho^{j,BDS} - c \cdot \delta t_r + c \cdot \delta t^{j,BDS} - c \cdot GBTO \\ \vdots \end{bmatrix}_{n \times 1} \tag{3.13}$$

$$H = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{x_r - x^{j,GPS}}{\rho^{j,GPS}} & \frac{y_r - y^{j,GPS}}{\rho^{j,GPS}} & \frac{z_r - z^{j,GPS}}{\rho^{j,GPS}} & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{x_r - x^{j,GAL}}{\rho^{j,GAL}} & \frac{y_r - y^{j,GAL}}{\rho^{j,GAL}} & \frac{z_r - z^{j,GAL}}{\rho^{j,GAL}} & 1 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{x_r - x^{j,BDS}}{\rho^{j,BDS}} & \frac{y_r - y^{j,BDS}}{\rho^{j,BDS}} & \frac{z_r - z^{j,BDS}}{\rho^{j,BDS}} & 1 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}_{n \times p} \tag{3.14}$$

Where n is the number of observations, and p is the number of parameters. Starting from an approximate solution, we can estimate the solution using the following equations.

$$X_0 = \begin{bmatrix} x_0 & y_0 & z_0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.15}$$

The normal matrix N is computed and the system is solved at epoch k, with the residual vector v been extracted

$$N_k = G_k^T R_k G_k \quad C = G_k^T R_k Y_k \tag{3.16}$$

$$\hat{X_k} = X_0 + N_k^{-1} \cdot C_k \quad \hat{v_k} = A_k \hat{X_k} - Y_k \tag{3.17}$$

Several iterations might be needed since the system has been linearized, depending on how far from the real values the a priori vector X_0 is. The stopping criteria here is the change of dX between two iterations. If the norm of that change is below $10e^{-4}$ or if we are doing more than ten iterations, we stop the loop. We are not using any previous observations and only perform a one-epoch solution. However, since we are placing ourselves into a static mode at the beginning of the positioning, we could start to accumulate data by changing our least square model into a recursive model.

3.3.2 Recursive Least Square

Recursive least square is a mathematical model where information from the solution computed in the previous epoch $k-1$ is reused in the current epoch k. It can be seen as a Kalman filter with no prediction model. It is therefore simpler to set up and can afterward be upgraded into a Kalman filter by adding the prediction part. The equations to set up a recursive system are defined in [11]. Our solution update is described in equation 3.18, where k refers to the current epoch.

$$\hat{X}_k = \hat{X}_{k-1} + K_k \cdot \gamma_m \tag{3.18}$$

We define here what we will be later our Kalman gain K and our innovation vector γ (3.20).

$$K_k = P_k^{-1} A_k^T R_k^{-1} \tag{3.19}$$

$$\begin{gathered} \gamma_k = \tilde{y}_k - \hat{y}_k \\ \tilde{y}_k = A\hat{X}_k \quad \hat{y}_k = A\hat{X}_{k-1} \end{gathered} \tag{3.20}$$

P^{-1} matrix is equal to the normal matrix defined in equation 3.16, $P^{-1} = N$. It will contain all the information of our previous epochs, getting incremented each time new measurements are coming (3.21).

$$P_k^{-1} = P_{k-1}^{-1} + A_k^T R_k^{-1} A_k \tag{3.21}$$

The Recursive Least Square technique is presented here because it was developed in the app at first. However, once the Kalman filter was integrated and correctly tuned, it became less relevant. Therefore, it will not be shown in the result, and the accumulation of data among epochs will be left to the Kalman filter chapter (chapter 4).

3.4 Results analysis

Starting from the solution obtained from the default state of the app when I first arrived, figure 3.3 represents a single epoch solution, with no accumulation of data between epochs. We can see a massive offset in the East direction, with a mean value of 26 m and a small offset in the north direction of 5m. The up direction also contains a large offset with a mean value of 26 m. Regarding the precision, the position solution have a significant dilution in horizontal.

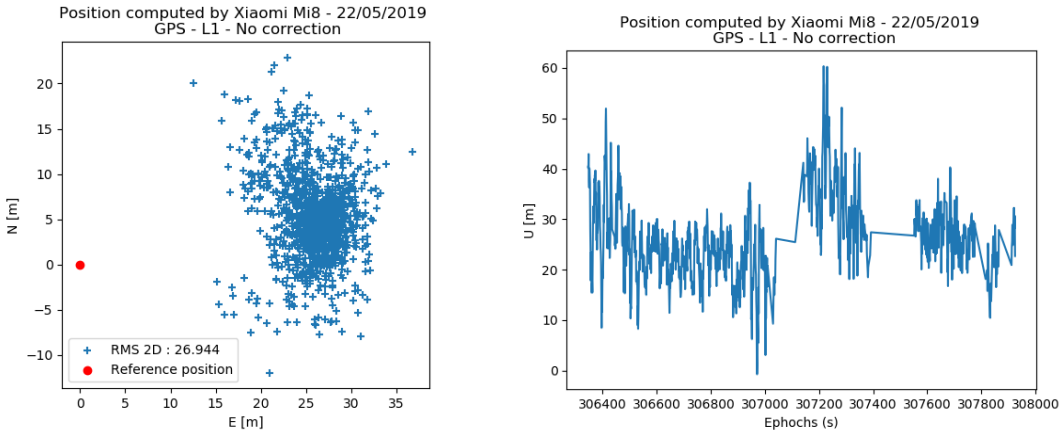


Figure 3.3 – Initial solution of the app, where large offset can be seen.

Because we are developing an algorithm in real-time here, it is difficult to compare different solutions computed from the phone, with different corrections, on the same dataset. For more coherent comparison, the results presented in this section are extracted from the same dataset and

have been "replayed" in Android Studio by reading the observations outputted by the app in our custom file format. That way, we can more easily change the corrections applied and see their impact on the solution. The date written in the graph is the acquisition date, the positions been computed with the last version of the code available today. Every position is tranformed to local ENU frame, the reference being the pillar's true position surveyed using a geodesic grade receiver.

3.4.1 Mono-constellation

Analyzing the results presented in figure 3.3, I found that the east offset was similar to one offset describe in ESA book [2] about the earth rotation correction. Indeed here we are computing the satellite position in the reference frame ECEF. This reference frame rotates with the Earth, meaning we need to compute the satellites' positions at reception time for our solution to be correct. Because the signals take out 0.07 seconds on average to go from the satellites to the Earth, a rotation needs to be applied on the satellites position computed at emission time, depending on the transmission time and the Earth rotation rate. If not applied, the position will suffer a 25 m meter offset to the East. When I looked at the code from the application, the satellites' positions were not corrected from this error, explaining the east offset. As we can see in figure 3.4, we apply this correction, reducing the offset greatly in the east.

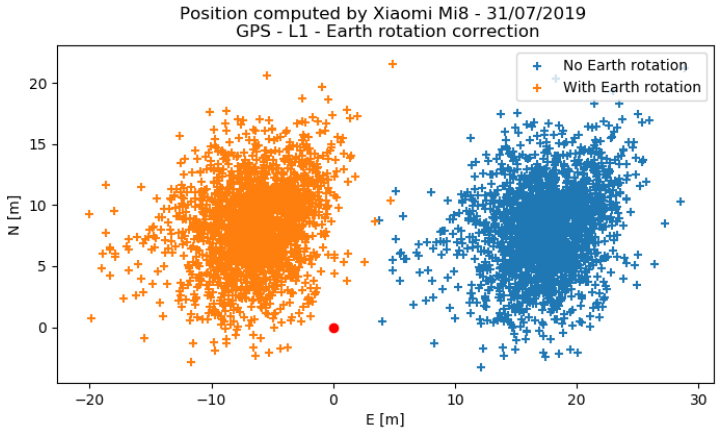


Figure 3.4 – Single epoch solution, with and without the earth rotation correction.

Note that at this point, our solution should be centered on the reference point, even with a bad precision. I tried different solutions to take care of this offset (see section 4.2.2), although I did not manage to remove it completely. It is a known issue in the code, which should be looked over in the future development of the app. For the rest of this section results, we will not discuss more about this inaccuracy issue and focus on the precision of the results.

In this solution, we are only using GPS satellites. We can now add GALILEO and BEIDOU satellites and see the effects on the solution.

3.4.2 Multi-constellation and multi-frequencies

Since we have integrated RTCM streams, it is possible to compute a multi-constellation solution. First, the GALILEO signals are added because the system is based on the same design as the GPS constellation. Indeed computation of satellite position is similar in GPS and GALILEO, contrary to GLONASS. The same goes for BEIDOU, which is why its integration also have been performed in the app. The goal is to increase our system redundancy as much as possible. In figure 3.5, we see our accuracy is better than with the GPS only solution, especially for all three components east, north, and up. We have more redundancy in our computations indeed. Statistics are displayed in

table 3.2 and confirm that the overall accuracy of the solution is better with the GALILEO satellites, with horizontal RMS going from nearly 8 meters to 4.5 meters. However, overall precision tends to stay the same with or without GALILEO constellation.

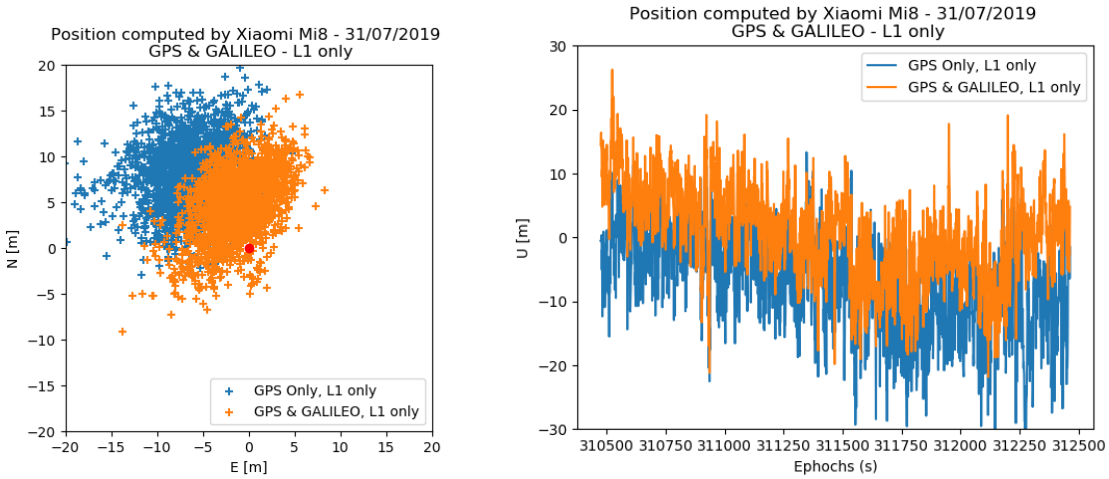


Figure 3.5 – Single epoch solution, with GPS and GALILEO constellations.

	GPS only			GPS & GALILEO		
Statistics	East [m]	North [m]	Up [m]	East [m]	North [m]	Up [m]
Mean	-6.440	8.237	-7.479	-1.225	4.817	0.626
Standard deviation (1 σ)	3.226	3.542	7.697	3.056	3.615	7.326
RMS (1 σ)	7.203	8.967	10.733	3.292	6.023	7.353
RMS 2D (1 σ)	11.502		-	6.864		-

Table 3.2 – Statistics for comparison of GPS and GPS & GALILEO solutions.

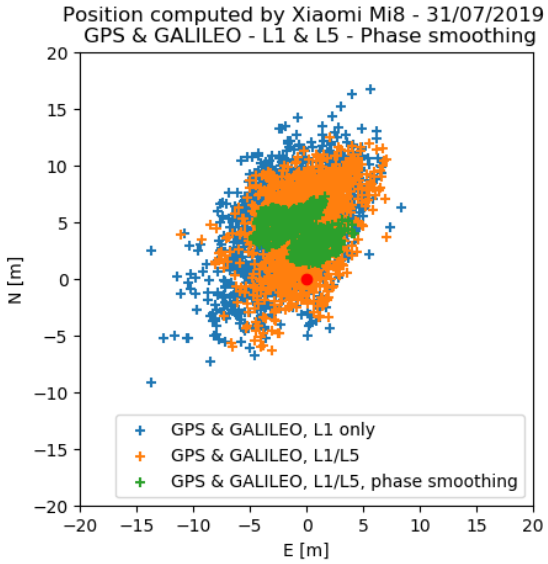


Figure 3.6 – Single epoch solution, using both L1 and L5 measurements and smoothing the pseudo-ranges.

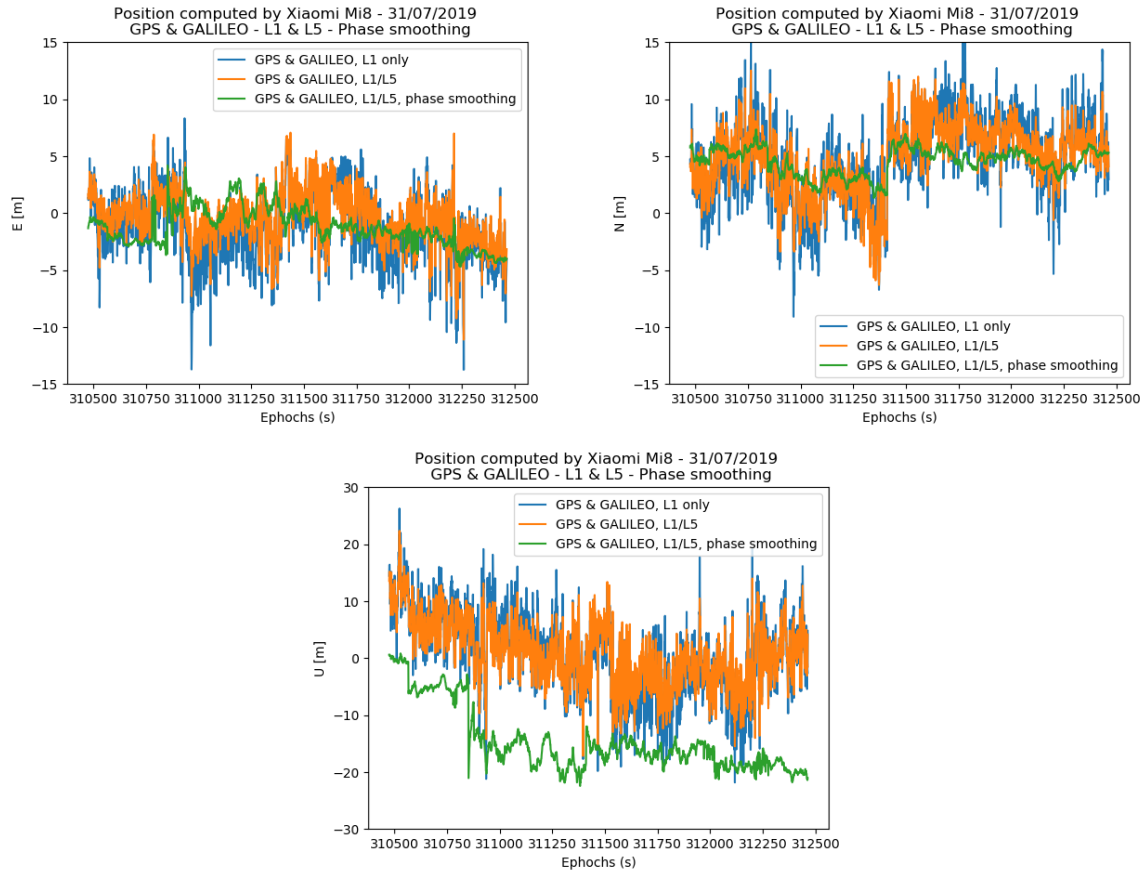


Figure 3.7 – Single epoch solution, using both L1 and L5 measurements and smoothing the pseudo-ranges (ENU).

As for now, we have only been using L1 measurements. However, now that we have the GALILEO satellites, which are all sending L5 signals, we can try to put those measurements into the computations. First, the L1 and L5 are added as separate measurements, increasing the redundancy of our system. In figure 3.6 and 3.7, we are adding the L5 measurements to the solution, and we are also performing the pseudorange smoothing described in section 3.1.2. Statistic of those computations are available in table 3.3. First of all, adding the L5 measurements improve our positioning significantly, as we add more measurement to the system. Moreover, when the smoothing is applied to the measurement, our solution's imprecision due to the noise of the pseudorange is greatly reduced. It results in an improved precision for the horizontal components. However, applying this smoothing seems to impact the precision of the up component greatly, with its RMS value going from 5 to 15 meters. The origin of this large inaccuracy is still unknown.

	L1 only			L1 & L5			L1 & L5 (Smoothed)		
Statistics	E [m]	N [m]	U [m]	E [m]	N [m]	U [m]	E [m]	N [m]	U [m]
Mean	-1.225	4.817	0.626	-0.350	4.862	0.919	-1.209	4.523	-14.571
Standard deviation (1 σ)	3.056	3.615	7.326	2.331	2.999	5.726	1.690	1.166	5.593
RMS (1 σ)	3.292	6.023	7.353	2.357	5.712	5.799	2.077	4.671	15.608
RMS 2D (1 σ)	6.864		-	6.180		-	5.112		-

Table 3.3 – Statistics for comparison of L1 only and L1/L5 solutions.

Another way to use the L1 and L5 measurements is to compute the ionosphere-free combination. In figure 3.9 and 3.8, the ionosphere-free combination is compared to the L1/L5 solution. We see that it largely increase the noise in our position. Two factors can explain this:

1. The ionosphere-free combination has an increased noise because combining the noise of the two measurements. This leads to a less precise solution but with a better accuracy in theory. One solution for this is to use a smoothing algorithm we detailed in section 3.1.2 and represented in the graphs 3.9 and 3.8, but this is not enough.

2. The geometry of our solution, with our system becoming less redundant. Since we are only using satellites from which we receive both frequencies. Note that the ionosphere-free combination was originally performed with L1/L2 measurements, but we are doing it here with L1/L5. While the physic behind is the same, L5 measurements are harder to get than L2 measurements since only 12 GPS satellites are sending it [33]. It means that only 4-5 satellites can be used in an open sky environment, which is barely enough to compute a solution for our system. Thankfully, all GALILEO satellites send on this frequency, and with 22 satellites operational, our system becomes redundant as we can expect about half of it in open sky. However, since we are cutting off many signals because of their low C/N0 ratio, much fewer signals are in reality available for computations. BEIDOU signal B3 also cover the L5 frequency. This is why I decided to implement BEIDOU signals also in the application, especially since the satellite position computations are similar to GPS and GALILEO. Since the Xiaomi Mi8 is a Chinese smartphone, I expected the phone to received B3 signals, which in reality was not the case. Only B1 (L1) signals are received from BEIDOU satellites, making them useless for the ionosphere-free combination. BEIDOU integration is explained later in this section.

To summarize, the system becomes much less redundant with only the L1/L5 satellites, which explain the noisier positioning and the error spike on the ionosphere-free curve. We can confirm this by looking at the Geometric Dilution of Precision (GDOP) values, which indicates the quality of our satellite geometry and represented in figure 3.10. When only dual-frequency L1/L5 signals are used, DOP values do not go under 2. Usually one should have a GDOP under 2, which is a general "thumb-rule" for proper quality positioning. This explains why our solution is so degraded with the ionosphere-free combination for certain epochs. What transpires from figure 3.9 is that doing SPP positioning with the ionosphere-free combination results in noisier and worse positions than without it. Because of that, I decided not to use this combination for the SPP algorithm. However, since we are going toward the development of PPP in real-time, the ionosphere combination has been kept inside the app. Once more GALILEO satellites will be available, it would be possible to use the dual-frequency capacities of the phone. Increasing the noise of the measurements with this combination will not be a problem anymore, since will be resolve the carrier-phase ambiguity, much more precise than pseudoranges.

To account for the ionosphere delay in our computation, I first assumed it could be easily negated by signal combination, thanks to the dual-frequency capacities of the smartphone. This turned out to be untrue as we just saw in this section. We are therefore left with modeling of the ionospheric delay, following the models described in section 3.2.2. Due to lack of time, I could not perform the decoding of the ionosphere parameters nor from the GPS navigation message or the RTCM streams. For future developments of the SPP algorithm, attention should be given on modeling this error in order to enhance the precision of the solution, either using navigation or RTCM stream messages.

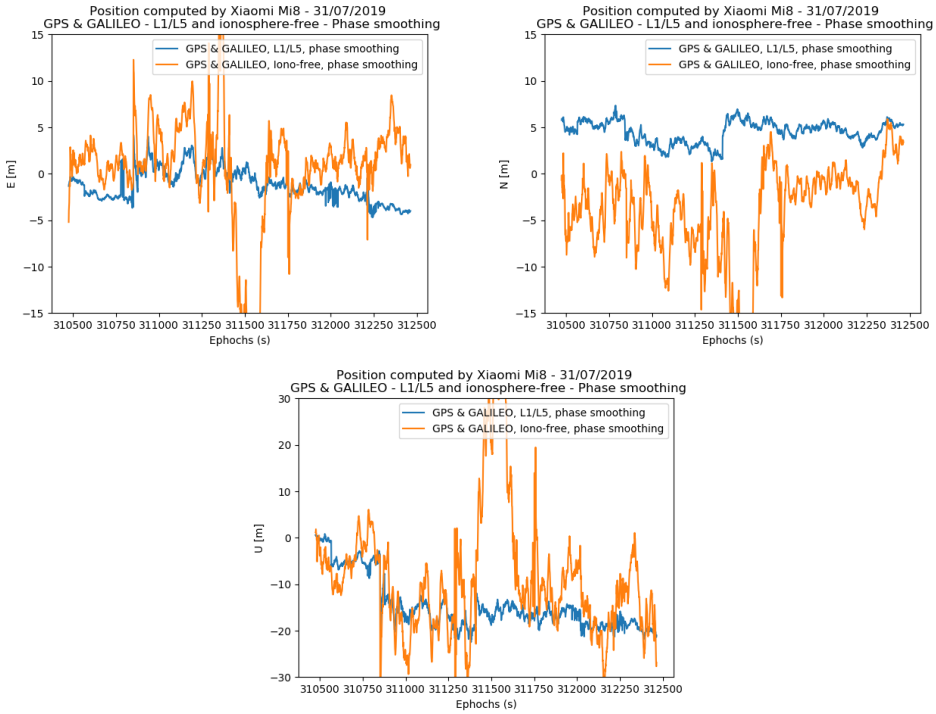


Figure 3.8 – Single epoch solution, using the ionosphere-free combination (ENU).

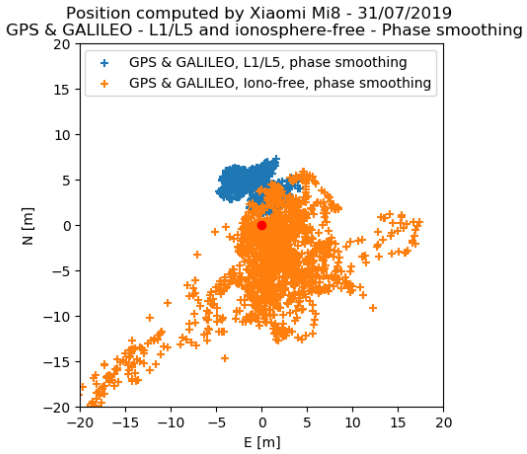


Figure 3.9 – Single epoch solution, using the ionosphere-free combination.

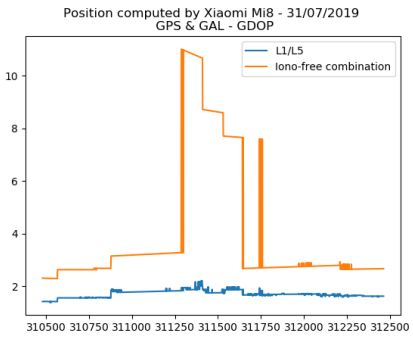


Figure 3.10 – GDOP of the L1/L5 solution compared to the iono-free solution.

BEIDOU is not yet integrated into the previous computations. As GALILEO, this system is defined similarly to GPS, which makes it easy to integrate with it. After adding the necessary RTCM streams to retrieve the satellites' information (see chapter 2), we can integrate it into our solution. However, we find out that our solution is unchanged in figure 3.11. This is because the number of BEIDOU satellites kept in the system is low, even if usually 7-9 BEIDOU satellites are visible in the sky. There are two reasons for that:

1. When we look into the ephemeris retrieved from the RTCM streams, not all the PRN for the BEIDOU constellation is present. Certain ephemeris are missing in the streams messages for unknown reasons, which is a problem also observed for certain GALILEO satellites but much more are missing for BEIDOU satellites. Therefore if we cannot compute the satellite position, we need to discard this measurement from our computations.
2. BEIDOU signals tend to have smaller C/n0 values, as seen in section 5.2.4. Because of this, a lot of them are removed by our cutoffs. With those reasons, if only one satellite is left, no information is added to the system, as we also have to estimate the GPS-BEIDOU Time Offset (i.e., GBTO, see section 3.3.1).

As we explained earlier, BEIDOU B3 signals close to the L5 band are not received by the phone, which was the reason why we integrate it in the first place. Integration of BEIDOU is kept in the app, but they will not be used in the rest of the results presented here, as they had too little to our system.

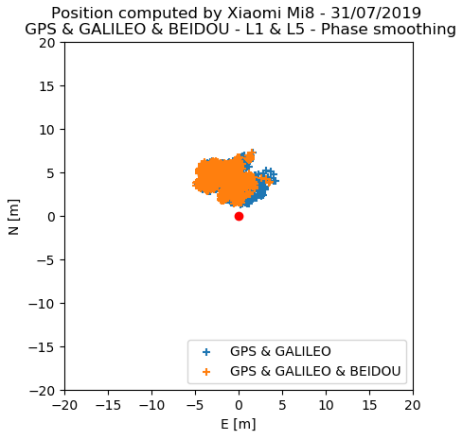


Figure 3.11 – Single epoch solution, using GPS, GALILEO and BEIDOU constellations.

In conclusion, our solution is improved by the integration of the algorithm described above. However, we did not use the TDCP algorithm developed in section 3.1.3 yet because we are only doing single-epoch positioning. In the next chapter, we will develop the Kalman filter that will allow us to accumulate data from previous epochs, for convergence of a solution.

Chapter 4 Extended Kalman Filter

This chapter presents the definition of our Kalman Filter for computation of position in static and dynamic mode.

4.1 Equation definition

A Kalman filter is an essential upgrade once dynamic surveys are performed since the receiver position will not be fixed anymore. Going from the Recursive Least Square equation defined in section 3.3.2, what is missing is the prediction of what will be the states of our system at epoch $k+1$. Since our model is non-linear, this filter will be an Extended Kalman Filter (EKF). It assumes that our prediction from epoch $k-1$ to epoch k is accurate enough, near to the actual position, so that one iteration of our Kalman is enough.

4.1.1 Prediction model

The general prediction model for a Kalman filter is established in equations 4.1 [2].

$$\begin{aligned} \hat{X}_k^- &= F_{k-1}\hat{X}_{k-1} \\ P_{\hat{X}_k^-}^- &= F_{k-1}\hat{P}_{k-1}F_{k-1}^T + Q_{k-1} \end{aligned} \tag{4.1}$$

Where:
$\hat{X^-}$ is the predicted state vector;
F is the transition matrix and defines the propagation of the vector parameter estimate $\hat{x}$;
Q is the process noise matrix, adding noise to our simple prediction model and avoid system lock;
P is the variance-covariance matrix of our parameters.

4.1.2 Update

Once the prediction is computed, an update should be made to compute the solution of the current epoch.

$$\gamma_k = Y_k - H_k \cdot \hat{X}_k^- \tag{4.2}$$

$$S_k = H_k P_k^- H_k^T + R_k \tag{4.3}$$

$$K_k = P_k^- H_k^T S_k^{-1} \tag{4.4}$$

$$\begin{aligned} \hat{X}_k &= \hat{X}_k^- + K_k\gamma_k \\ P_{\hat{X}_k} &= (I_k - K_k H_k)P_{\hat{X}_k^-} \end{aligned} \tag{4.5}$$

Where:
γ is the innovation vector;
S is the variance-covariance matrix of the innovation vector;
R is the variance-covariance matrix of our observations;
K is the Kalman gain matrix.

4.2 Static mode

4.2.1 Matrix definition

No estimation of the user velocity is performed first, to set up correctly the Kalman filter and make sure that everything was working correctly before going into dynamic mode. We have the same state vector defined in equation 3.12. The first thing to do when designing a Kalman filter is to set up the transition matrix F, which will be used to pass from a solution at epoch k to epoch $k+1$. Here, we now that our position similar from one epoch to another. However, the receiver clock error δt_r cannot be let constant. Since we do not have any estimation of the clock drift rate, we re-estimate it at each epoch and model it as a white noise with zero mean [2]. As for our inter-system offsets, they can assume stable between epochs, since their variances shall be better than $8e^{-14}$ over a day [24].

$$X = \begin{bmatrix} x & y & z & c \cdot dt_r & c \cdot GGTO & c \cdot GBTO \end{bmatrix}^T \tag{4.6}$$

$$F = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 0 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}_{p \times p} \tag{4.7}$$

The other important matrix to define is the process noise matrix Q. We know that our position did not change, so there is no need to add noise between epochs to our model. However, since we are not sure of the receiver clock stability, we give it a one-millisecond noise [2].

$$Q = \begin{bmatrix} 0 & & & & & \\ & 0 & & & & \\ & & 0 & & & \\ & & & \sigma_{cdt_r} & & \\ & & & & 0 & \\ & & & & & 0 \end{bmatrix}_{p \times p} \tag{4.8}$$

$$\sigma_{c \cdot dt_r} = 1ms \cdot c = 300km \text{ [2]} \tag{4.9}$$

For the first iteration, the precision of our a priori values need to be estimated in the matrix P_0. Because we want to start our filter with correct assumptions of our parameters, the first epochs received will be used to compute an approximation of our coordinates using the simple least square algorithm developed in section 3.3, given us good approximates for our solution. Therefore the following values can be used to estimate our first state precision.

$$P_0 = \begin{bmatrix} \sigma^2_{0,x} & & & & & \\ & \sigma^2_{0,y} & & & & \\ & & \sigma^2_{0,z} & & & \\ & & & \sigma^2_{0,cdt_r} & & \\ & & & & \sigma^2_{0,GGTO} & \\ & & & & & \sigma^2_{0,BGTO} \end{bmatrix}_{p \times p} \tag{4.10}$$

$$\begin{aligned} \sigma_{0,x} = \sigma_{0,y} = \sigma_{0,z} &= 10m \text{ [2]} \\ \sigma_{0,cdt_r} &= 1ms \cdot c \text{ [2]} \\ \sigma_{0,\Delta t} &= 0.25m \text{ [2]} \\ \sigma_{0,GGTO} = \sigma_{0,GBTO} &= 100ns \cdot c \text{ [24]} \end{aligned} \tag{4.11}$$

4.2.2 Results

Taking back the results from section 3.4 where we only smoothed our pseudoranges using the phase, we can now accumulate data between epochs. We put the phone into static mode, where it will use the prediction model described above, i.e., no position change. In figures 4.1 and 4.2, we can clearly see that the positioning solution is enhanced by the Kalman filter, although this was expected. Indeed we are putting a significant constraint on this filter, by defining the position is static and should not change. It explains the important inertia of the filter and its slow variation. Plus, we are no longer using only the data from the current epoch, and our solution is impacted by the previous epochs as well. As we see in table 4.1, standard deviation values are improved greatly but also attest to the inertia of our system. Our position is now precise, but not more accurate as the mean and RMS values are worse than before.

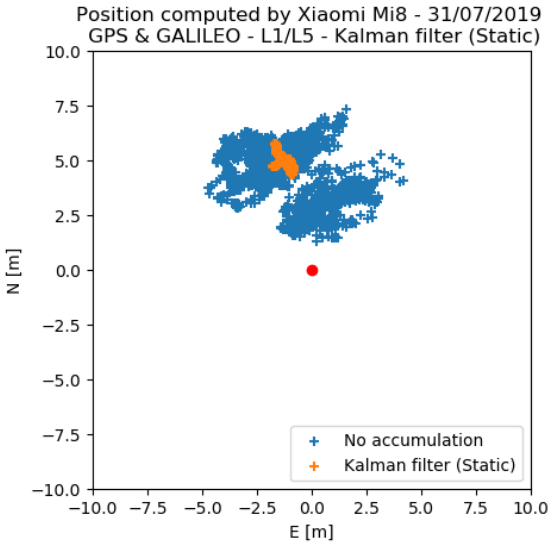


Figure 4.1 – Comparison of single epoch position and Kalman filter (Static).

	Single epoch solution			Kalman filter (Static)		
Statistics	East [m]	North [m]	Up [m]	East [m]	North [m]	Up [m]
Mean	-1.209	4.523	-14.571	-1.461	5.132	-7.494
Standard deviation (1 σ)	1.689	1.166	5.594	0.180	0.310	3.611
RMS (1 σ)	2.078	4.671	15.608	1.472	5.142	8.319
RMS 2D (1 σ)	5.112		-	5.348		-

Table 4.1 – Statistics for comparison of single epoch positioning and kalman filter in Static mode.

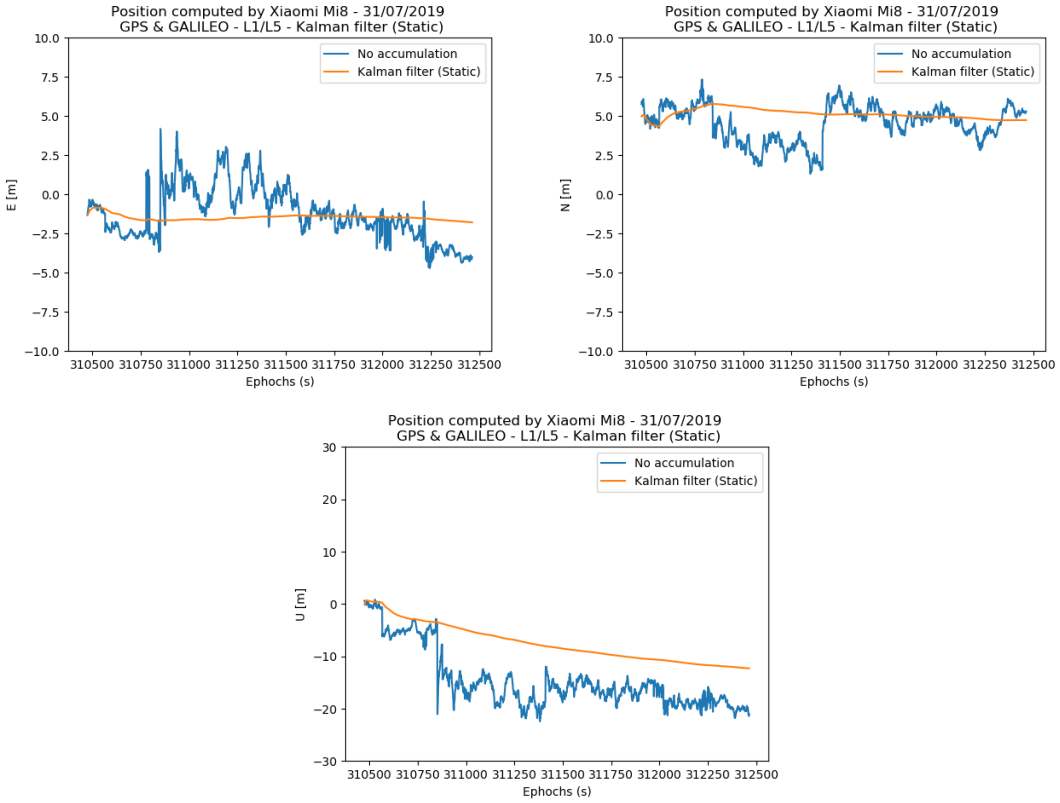


Figure 4.2 – Comparison of single epoch position and Kalman filter (Static) (ENU).

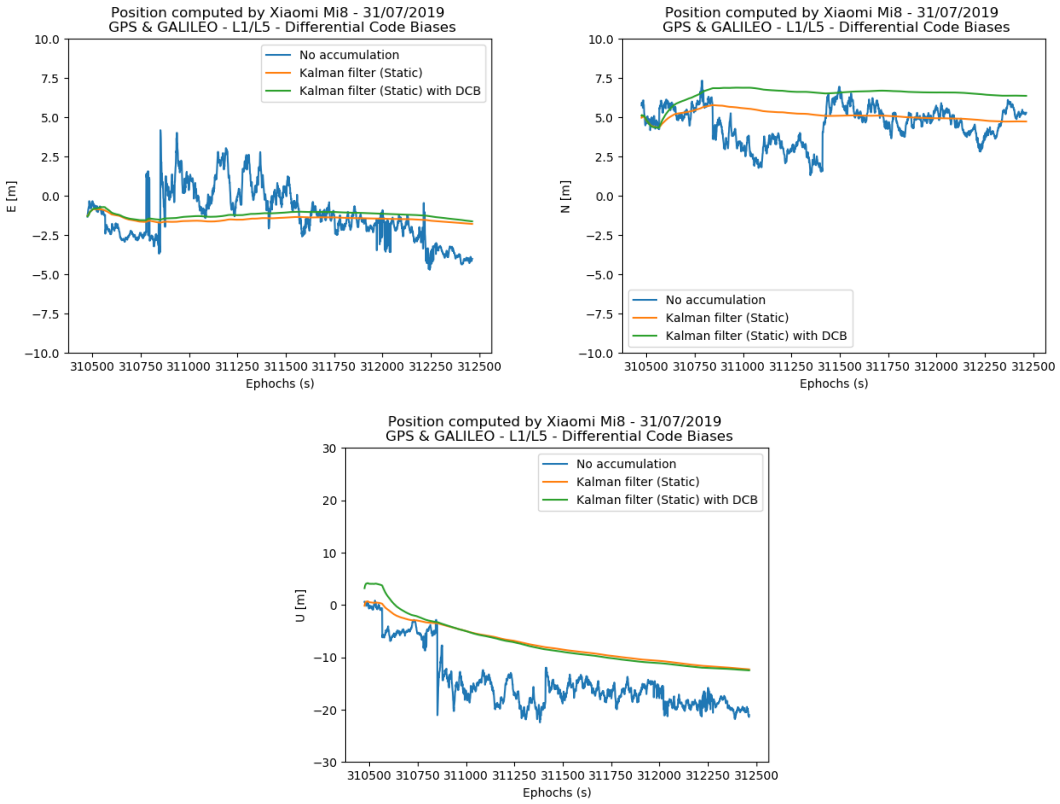


Figure 4.3 – Kalman filter solution (Static) with and without DCB corrections.

Moreover, we see at this point that our position is still impacted by the "jumps" in positions and the slow bias that can be seen in the single epoch positioning still impact our solution, which was expected. To try to solve this issue, I have gone through lots of different studies, while trying to confirm that the different part of my code was correct. I first thought about this issue being related to an error in the satellites' positions, which is why I spent much time verifying this part of the code (see section 2.3.3). However, it did not solve this issue, and I try to look at other possible explanations. Since we are not performing ionospheric corrections, this positioning bias could also be linked to that. But by looking at the ionosphere-free combination solution in figure 3.9 that we presented in the previous chapter, biases are still present even within the noise. I therefore assumed that the error was elsewhere. In a study [20] I found that Differential Code Biases (DCB), which are usually negated when doing DGNSS, have a significant impact on the accuracy of a solution. [20] explains it need to be accounted for in PPP, but I neglected them for SPP before. The study's results showed similar biases that are experienced in our results, especially the 5 meters offset visible in the north components, which was eliminated by applying the DCB corrections on the measurements. DCBs for all satellites of all constellations are available on the IGS website, being valid for a year in general. We are also receiving from the RTCM streams in messages 1059 (GPS), 1242 (GALILEO) and 1260 (BDS). Seeing the results from [20], I decided to decode those messages as well, in order to correct the pseudoranges from those errors. In figure 4.3, we see that the accuracy of the Kalman solution before and after applying the DCB corrections do not change and the north components is even worse than before. Therefore, the root of this offset problem must remain elsewhere. As for today, I still have not figured out the nature of this offset, which when we look closely was already present at the initial state of the app on figure 3.3.

4.3 Dynamic mode

The real interest in real-time positioning is the navigation purposes. We want to reconstruct the user's trajectory, and our model should contain parameters related to the dynamic part of the position, i.e., the user's velocity. It is another reason why a Kalman filter was developed since it is best suited for estimation and prediction of dynamic parameters from one epoch to another. To design it, I used the algorithm developed inside [2, 11] and the one developed by another application called GNSS Compare [14]. This last reference is an open-source code develop during the ESA competition 2018, which make it easy to see how the model described in the documentation has been coded in Java inside an Android application. However, since neither of them used TDCP inside of their design, I adapted their designs and equations so that velocity information can be extracted from them.

Taking back the matrices we developed in section 3.3.1, we need to add the dynamic components, which will be the estimation of the user velocity. In this section, we will also introduce the clock drift $\delta\dot{t}_r$, which will be estimated in our state vector to account for change in the receiver clock between two epochs.

$$X = \begin{bmatrix} x & \dot{x} & y & \dot{y} & z & \dot{z} & c \cdot \delta t_r & c \cdot \delta\dot{t}_r & c \cdot GGTO & c \cdot GBTO \end{bmatrix}^T \tag{4.12}$$

This time, we will use the TDCP algorithm developed in section 3.1.3, since carrier-phase measurements are very precise for velocity estimation [13]. It also allows us to had more measurements in our system, making it largely redundant for the velocity estimation. What will result from that is that velocity estimation will be very precise, much more than our position estimation. It will help us constrain even more our positioning solution and our predictions.

However, this also means that if we start using the phase measurements while starting on some distant coordinates, depending on the weight we give to the phase measurements, it will take time to find the correct absolute position. Because of our precise phase measurement, relative positioning is expected to be great, but absolute positioning will not be directly enhanced. Before passing into

a dynamic mode, we need to converge onto a correct absolute position. Otherwise, our solution will be precise but not accurate.

4.3.1 Linear model

Using the models from equation 3.4 and equation 3.1, we can build a new design matrix H and a new observation vector Y as detailed in equations 4.13 and 4.14.

$$Y = \begin{bmatrix} \vdots \\ P_i^{j,GPS} - \rho^{j,GPS} - c \cdot \delta t_r + c \cdot \delta t^{j,GPS} \\ \Delta\Phi^{j,GPS} - \Delta D + \Delta g + e_k \cdot \Delta r_u - c \cdot \Delta\dot{\delta t}_r \\ \vdots \\ P_i^{j,GAL} - \rho^{j,GAL} - c \cdot \delta t_r + c \cdot \delta t^{j,GAL} - c \cdot GGTO \\ \Delta\Phi^{j,GAL} - \Delta D + \Delta g + e_k \cdot \Delta r_u - c \cdot \Delta\dot{\delta t}_r \\ \vdots \\ P_i^{j,BDS} - \rho^{j,BDS} - c \cdot \delta t_r + c \cdot \delta t^{j,BDS} - c \cdot GBTO \\ \Delta\Phi^{j,BDS} - \Delta D + \Delta g + e_k \cdot \Delta r_u - c \cdot \Delta\dot{\delta t}_r \\ \vdots \end{bmatrix}_{n\times 1} \tag{4.13}$$

Note that since inter-system offsets are assumed constants between epochs, differentiation of phase measurement neglect them in the observation model.

$$H = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & & & \\ \frac{x_r - x^{j,GPS}}{\rho^{j,GPS}} & 0 & \frac{y_r - y^{j,GPS}}{\rho^{j,GPS}} & 0 & \frac{z_r - z^{j,GPS}}{\rho^{j,GPS}} & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{x_r - x^{j,GPS}}{\rho^{j,GPS}} & 0 & \frac{y_r - y^{j,GPS}}{\rho^{j,GPS}} & 0 & \frac{z_r - z^{j,GPS}}{\rho^{j,GPS}} & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & & & \\ \frac{x_r - x^{j,GAL}}{\rho^{j,GAL}} & 0 & \frac{y_r - y^{j,GAL}}{\rho^{j,GAL}} & 0 & \frac{z_r - z^{j,GAL}}{\rho^{j,GAL}} & 0 & 1 & 0 & 1 & 0 \\ 0 & \frac{x_r - x^{j,GAL}}{\rho^{j,GAL}} & 0 & \frac{y_r - y^{j,GAL}}{\rho^{j,GAL}} & 0 & \frac{z_r - z^{j,GAL}}{\rho^{j,GAL}} & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & & & \\ \frac{x_r - x^{j,BDS}}{\rho^{j,BDS}} & 0 & \frac{y_r - y^{j,BDS}}{\rho^{j,BDS}} & 0 & \frac{z_r - z^{j,BDS}}{\rho^{j,BDS}} & 0 & 1 & 0 & 0 & 1 \\ 0 & \frac{x_r - x^{j,BDS}}{\rho^{j,BDS}} & 0 & \frac{y_r - y^{j,BDS}}{\rho^{j,BDS}} & 0 & \frac{z_r - z^{j,BDS}}{\rho^{j,BDS}} & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & & & \end{bmatrix}_{n\times p} \tag{4.14}$$

4.3.2 Prediction model

We are no longer staying static on a point, and the prediction model needs to be changed so that the predicted solution uses the new velocity information we are computing. GNSS Compare application's Kalman filter was tuned by its developers using empirical values (S_X, S_Y, S_Z) and using the reference [7]. I decided to use their work to tune the Kalman filter I developed. Regarding the receiver clock error modeling, they used typical Allan Variance Coefficients values developed in [7], p.326-327, for low-quality receiver clocks. The new transition matrix F is presented in equation 4.15 and the process noise matrix in equation 4.16 [2, 14].

$$F = \begin{bmatrix} 1 & \Delta T & & & & & & & & & & \\ & 1 & & & & & & & & & & \\ & & 1 & \Delta T & & & & & & & & \\ & & & 1 & & & & & & & & \\ & & & & 1 & \Delta T & & & & & & \\ & & & & & 1 & & & & & & \\ & & & & & & 1 & \Delta T & & & & \\ & & & & & & & 1 & & & & \\ & & & & & & & & 1 & & & \\ & & & & & & & & & 1 & & \\ & & & & & & & & & & 1 & \\ & & & & & & & & & & & 1 \end{bmatrix}_{p \times p} \tag{4.15}$$

$$\mathbf{Q}_k = \begin{bmatrix} \frac{S_X\ \Delta T^3}{3} & \frac{S_X\ \Delta T^2}{2} & & & & & & & & & & \\ \frac{S_X\ \Delta T^2}{2} & S_X\ \Delta T & & & & & & & & & & \\ & & \frac{S_Y\ \Delta T^3}{3} & \frac{S_Y\ \Delta T^2}{2} & & & & & & & & \\ & & \frac{S_Y\ \Delta T^2}{2} & S_Y\ \Delta T & & & & & & & & \\ & & & & \frac{S_Z\ \Delta T^3}{3} & \frac{S_Z\ \Delta T^2}{2} & & & & & & \\ & & & & \frac{S_Z\ \Delta T^2}{2} & S_Z\ \Delta T & & & & & & \\ & & & & & & S_f + \frac{S_g\ \Delta T^3}{3} & \frac{S_g\ \Delta T^2}{2} & & & & \\ & & & & & & \frac{S_g\ \Delta T^2}{2} & S_g\ \Delta T & & & & \\ & & & & & & & & 0 & & & \\ & & & & & & & & & 0 & & \\ & & & & & & & & & & 0 & \\ & & & & & & & & & & & 0 \end{bmatrix} \tag{4.16}$$

$$\begin{aligned} S_X = S_Y &= 0.2m \text{ [14]} \\ S_Z &= 0.01m \text{ [14]} \\ S_f &= \frac{h_{-2}}{2} \text{ [7]} \\ S_g &= 2\pi^2 h_{-2} \text{ [7]} \\ h_0 &= 2e^{-19} \cdot c^2 \text{ [7]} \\ h_{-2} &= 2e^{-20} \cdot c^2 \text{ [7]} \end{aligned} \tag{4.17}$$

$$P_0 = \begin{bmatrix} \sigma^2_{0,x} & & & & & & & & & & & \\ & \sigma^2_{0,vel} & & & & & & & & & & \\ & & \sigma^2_{0,y} & & & & & & & & & \\ & & & \sigma^2_{0,vel} & & & & & & & & \\ & & & & \sigma^2_{0,z} & & & & & & & \\ & & & & & \sigma^2_{0,vel} & & & & & & \\ & & & & & & \sigma^2_{0,cdt_r} & & & & & \\ & & & & & & & \sigma^2_{0,c\dot{d}t_r} & & & & \\ & & & & & & & & \sigma^2_{0,GGTO} & & & \\ & & & & & & & & & \sigma^2_{0,BGTO} & & \end{bmatrix}_{p \times p} \tag{4.18}$$

$$\begin{aligned} \sigma_{0,x} = \sigma_{0,y} = \sigma_{0,z} &= 10m \text{ [2]} \\ \sigma_{0,vel} &= 0.01m/s \text{ [14]} \\ \sigma_{0,cdt_r} &= 1ms \cdot c \text{ [2]} \\ \sigma_{0,\Delta t} &= 0.25m \text{ [2]} \\ \sigma_{0,GGTO} = \sigma_{0,GBTO} &= 100ns \cdot c \text{ [24]} \end{aligned} \tag{4.19}$$

4.3.3 Results analysis

We can now perform a dynamic test of our navigation solution. The reference point CECIL is surrounded by a small wall (about 0.5 m high) which taken as a reference. After performing a pre-survey using a high-quality receiver Septentrio AsteRx3 to establish the reference base, I went performed several laps around this ring with the application. At the beginning and the end of the survey, I performed a static acquisition of about 1m30s to let the Kalman converge to a correct absolute position. In figures 4.4, the single epoch solution is compared to the dynamic Kalman filter defined in the previous section. As we can see the offset present in our previous computations, especially the 5 meters offset in the north direction is still largely visible here. In order to easily compare the precision of this dynamic solution, we can try to remove those offset and align the solution with the reference, which is known to be accurate to a centimeter-level. By removing 1 meter in the east direction and 5 meters in the north direction, which is the approximate mean of the offset found after convergence of our solution during the static phase at the beginning of the survey, we obtain figure 4.6.

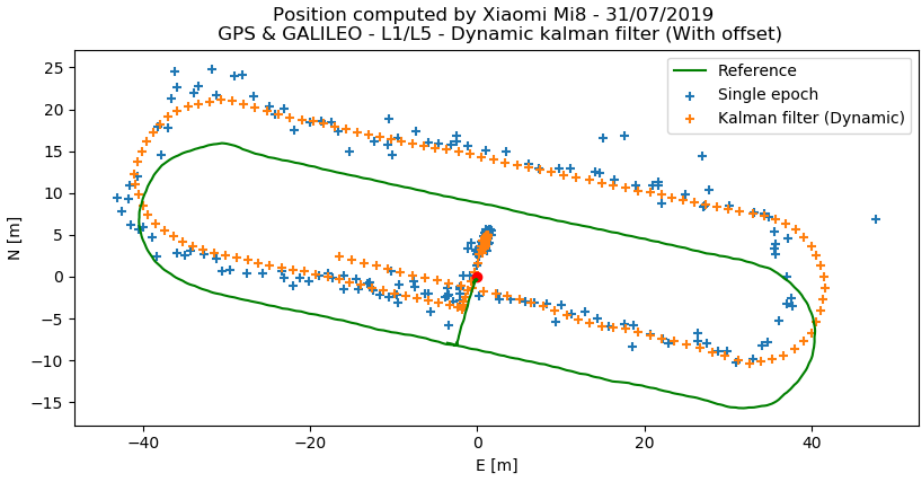


Figure 4.4 – Comparison of single epoch position and Kalman filter (Dynamic).

The solution is now accurate at the beginning of the survey, and we can study the precision of our solution. Several observations can be made looking at the new graph. First, our Kalman solution is much more precise than our single epoch solution, as one would expect from a Kalman filter. Like in our static Kalman, we are accumulating data from several epochs, but we also estimate our velocities using the TDCP algorithm. This gives us access to the very precise phase measurements. That way, we also estimate our clock drift, and we can *constrain* our solution with it. This explains the very smooth solution we can compute, more resilient than the pseudorange only solution. Indeed, We see that our trajectory on the east side of the ring is kept close to the reference, while our single epoch solution diverges from it. However, while our precision is very resilient, our accuracy still diverges at the end because our absolute positioning is not correct, as we "cheated" at the beginning of the survey by removing the offsets. This explains why our loop has a large offset of about 2 meters at the end and does not close correctly the circle made. Indeed if we continue for a second lap, like represented in figure 4.7, we see that our precision is still good, but we continue to diverge in absolute positioning.

Nevertheless, our positioning solution is still significantly improve by adding phase measurement with the TDCP method. Once the offset issue is fixed, we can expect a sub-meter accuracy and precision on the smartphone just by using the GNSS measurements.

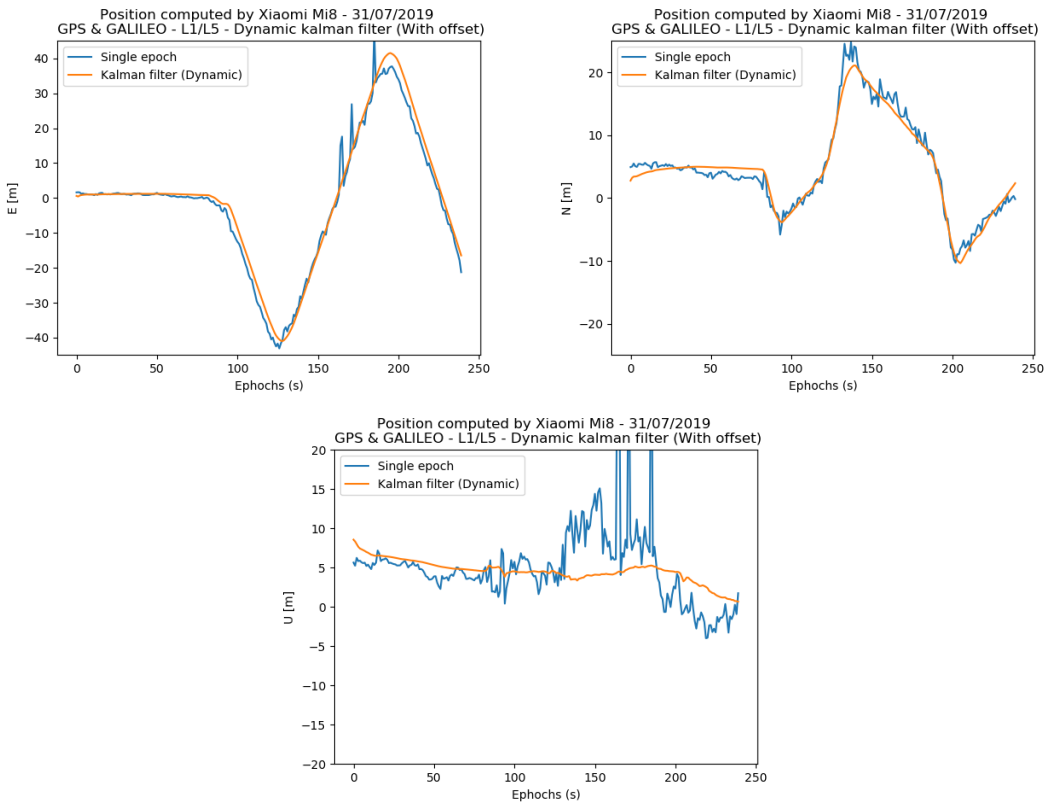


Figure 4.5 – Comparison of single epoch position and Kalman filter (Dynamic) (ENU).

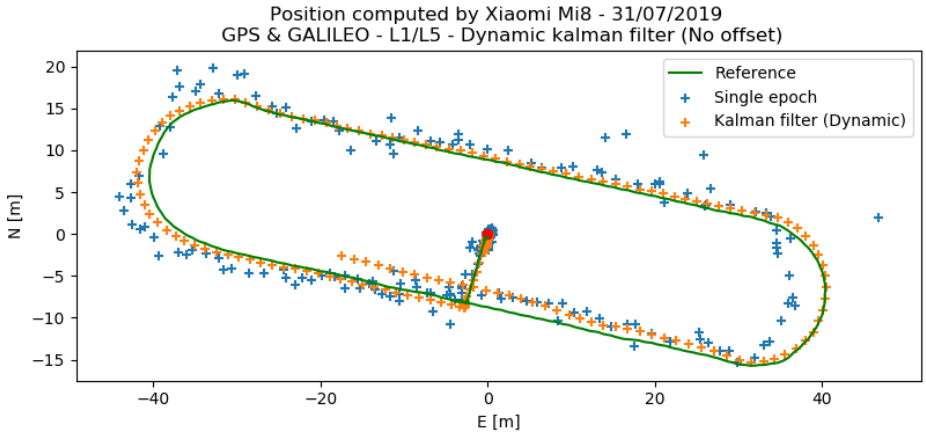


Figure 4.6 – Comparison of single epoch position and Kalman filter (Static).

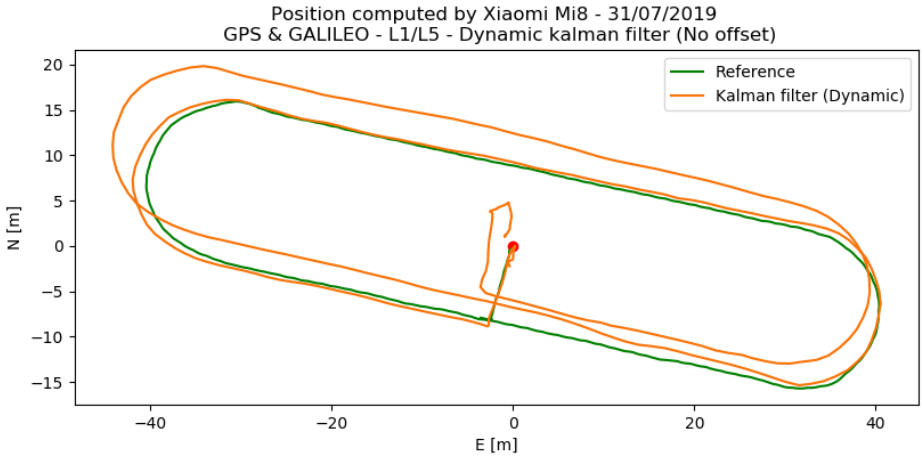


Figure 4.7 – Comparison of single epoch position and Kalman filter (Static), with two full laps.

Chapter 5
Precise Point Positioning

This chapter presents the Precise Point Positioning technique and introduce the challenges related to this method.

5.1 Technique definition

The Precise Point Positioning technique is a GNSS positioning method based on the observation of only one receiver. Contrary to the Differential GNSS (DGNSS), no measurements differentiation is made between two receiver to negate clocks and atmospheric errors. Therefore these need to be accounted for when computing the solution, either by signal combination (Iono-free, Melbourne-Wübbena combination) or modeling. Such methods allow removing a lot of the measurement error, which will allow us to find the carrier-phase ambiguity of the signal and is what differentiates PPP from SPP. This is why dual-frequency and access to precise ephemeris are mandatory to perform this technique. The PPP model presented here is a model defined by the ESA book [2] and from a lecture I had during my studies in ENSG presented by Pierre Bosser (ENSTA Bretagne)[6] and Xavier Collilieux (ENSG). I adapted it to the smartphone case, to only account for the interesting errors and to explain how the algorithms developed in chapters 4 and 5 can be enhanced for PPP computations.

The reason why I did not pursue the PPP development is because it has proven to be very challenging from the research papers I found. Especially when I attended the Prague workshop in late June, I found out that all research teams had trouble with real-time PPP. Even during my internship, new papers about post-processing PPP on the Xiaomi Mi8 were still being published [43] and nothing yet related to real-time PPP. I therefore decided to focus my work on enhancing the SPP solution, allowing real-time positioning with a good precision for pedestrian navigation, without the need for ambiguity estimation. While I did not have the time during this internship to introduce this algorithm inside the phone, I still performed much research on it because it was my primary goal at first. I therefore wanted to leave a summary of this work for the next person whose going to continue the development of the app, especially since he/she might not be specialized in GNSS positioning.

5.1.1 Measurement modeling

Taking back what's been developed in section 3.1, we need to introduce a few more corrections to build our PPP model. Recalling equation 3.1, we can enhance it for PPP purposes [2, 6]:

$$\begin{aligned} P_3^j &= \rho^j + c \cdot (\delta t_r - \delta t^j) + \delta u^j + \tau_{iono}^j + \tau_{tropo}^j + b_r^j + \delta r_{crust} + \epsilon \\ \Phi_3^j &= \rho^j + c \cdot (\delta t_r - \delta t^j) + \delta u^j - \tau_{iono}^j + \tau_{tropo}^j + \lambda_3 \cdot N_3^j + B_r^j + \lambda_3 \cdot \omega^j + \delta r_{crust} + \epsilon \end{aligned} \tag{5.1}$$

b_r^j is the Differential Code Bias between the receiver and the satellite;
B_r^j is the Differential Phase Bias between the receiver and the satellite;
λ_3 is the wavelength of ionosphere-free combination (L1/L5);
ω is the Wind Phase Up effect, due to polarization of the electromagnetic signal;

For the PPP computations, we are only going to use the ionosphere-free measurements computed from the combination of L1 and L5 signals, as we described in section 3.2.3. All the different corrections are related to this combination. Also, Differential Code Biases are not present here, because they are removed along with the ionospheric error when using the ionosphere-free combination.

5.1.2 Kinematic PPP and ambiguity resolution

We are talking here about ambiguity *estimation* and not *resolution* in this algorithm. In theory, since the ambiguity N is a number of cycles, it is defined an *integer*, meaning that is estimation as a float number is erroneous and contrary to the physics behind the electromagnetic signals. Therefore, for maximum precision, we should fix it to its actual integer value at some point in the estimation process. This process is called *fixing* or *resolution* of the ambiguity. However, it is very demanding in processing costs, because a lot of different values to be tested to resolve it. Several fixing algorithms with different approaches exist [28, 19] like in differential GNSS, with some being under patent. Since it would require much more time to develop an ambiguity-fixing part than the rest of the PPP, estimation is more than enough for real-time positioning on a smartphone. However with the ionosphere-free combination the ambiguity looses its *integer* quality, becoming a *float*. Therefore, another combination of signal would be more suitable for ambiguity resolution, like Melbourne-Wübbena combination, which also remove the ionosphere delay from the measurement while keeping the *integer* quality of the phase ambiguity. For details on other combinations that are more suitable for ambiguity resolution, please refer to [2], p.70.

Note that the process of only *estimating* the ambiguities is called *kinematic-PPP*, although it does not mean that our receiver will be moving, only that the ambiguities are not fixed.

5.2 Corrections

5.2.1 Precise orbits and clocks corrections

Precise satellites orbits and clocks are mandatory for PPP positioning, since an error in satellite position will directly be seen in the user position, especially when working at the decimeter and even centimeter-level with the phase measurements. This is why decoding of the precise corrections messages has been one of the first steps of the development during this internship, even though they are not mandatory for a correct SPP. Our goal is still to perform every computation in real-time, which makes the development of a PPP-RTK algorithm uneasy. For a detailed explanation of how the precise ephemeris and clocks are retrieved, please refer to chapter 2.

5.2.2 Tropospheric correction

The tropospheric correction can be partly estimated in the state vector for accurate modeling. τ_{tropo} can be modeled as in equation 5.2, using the satellite elevation and the Mapping of Niell, which does not require any surface meteorology measurements. More information of this model can be found in the ESA Open Book [2], p.125-126.

$$\begin{aligned} \tau^j_{tropo} &= Tr_0(E^j) + M_{wet}(E^j) \cdot \Delta Tr_{z,wet} \\ Tr_0(E^j) &= Tr_{z,dry}(E^j) \cdot M_{dry}(E^j) + Tr_{z_0,wet}(E^j) \cdot M_{wet}(E^j) \end{aligned} \tag{5.2}$$

$$\begin{aligned} Tr_{z,dry}(E) &= \alpha e^{-\beta H} \\ Tr_{z,wet}(E) &= Tr_{Z_0,wet} + \Delta Tr_{Z,wet} \end{aligned} \tag{5.3}$$

Where E^j is the elevation of the j^{th} satellite, H the height above sea level of the user and

$$\alpha = 2.3m; \quad \beta = 1.16e^{-4}m; \quad Tr_{Z_0,wet} = 0.1m. \tag{5.4}$$

M_{dry} and M_{wet} are related to the Mapping of Neil function, which gives the corresponding coefficients depending on the satellite elevation and the position of the user on Earth. The coefficients are linearly interpolated from the tables that can be found in Appendix B, extracted from the ESA book [2], p.126, tables 5.2 and 5.3.

$$\begin{aligned} M_{dry} &= m(E, a_d, b_d, c_d) + \Delta m(E, H) \\ \Delta m(E, H) &= (\frac{1}{\sin(E)} - m(E, a_{ht}, bht, c_{ht})) \end{aligned} \tag{5.5}$$

$$\begin{aligned} M_{wet} &= m(E, a_w, b_w, c_w) \\ \Delta m(E, a, b, c) &= \frac{1 + \frac{a}{1+\frac{b}{1+c}}}{\sin(E) + \frac{a}{\sin(E)+\frac{b}{\sin(E)+c}}} \end{aligned} \tag{5.6}$$

The constant part of the tropospheric delay $Tr_{Z_0,wet}$ is removed from the measurements and the variable part $\Delta Tr_{Z,wet}$ is estimated in the state vector.

5.2.3 Multipath handling

Multipath is an important issue when performing precise positioning and less easy to find out in measurements than cycle slips. It is even more critical for smartphone receivers, as they are not resilient at all to it compare to a geodesic grade antenna. In Google API, methods already exist to flag measurements that may be contaminated by multipath, similar to the cycle slips detectors explained in section 1.3.2. However, their robustness is yet to be proven and will be studied by Yazhend Wei, another intern at Geoloc laboratory, that will pursue this work in her internship. More multipath may need to be added to the application before PPP can be performed on the phone.

5.2.4 Antenna biases and orientations

Here, we introduce a new component called the *Phase Wind Up* effect, which a correction due to the relative orientation of the satellite antenna and the receiver's antenna. Since we are now going to use the phase measurement plainly and to find its ambiguity, we need to account for all the possible errors directly related to it. Details on how to compute this effect is given in [2], p.127-128.

Note that since we are using the ionosphere-free combination here, the satellites coordinates given to compute ρ will need to be related to the ionosphere-free phase center. Since we are using precise orbits from RTCM streams, coordinates are already referred to the satellite's Antenna Phase Center (APC) of the ionosphere-free combination signal. However, if one were to use the precise products from IGS, namely the ultra-rapid products, those are related to the satellite's mass center and therefore needs to be corrected. In theory, we would also need to account for the receiver's APC, but this would mean that we have a complete calibration model for the smartphone GNSS antenna. Those kind of models are usually performed on geodesic grade antenna but are not relevant in the smartphone case because of the poor antenna quality. To that with have to add the effect of the human body, creating big masks when in the vicinity of an antenna (see section). Since we want to compute the position while our smartphone being in our hand, the antenna calibration model become even less relevant.

Finally, [2] also mention that satellites under eclipse conditions need to be removed entirely from computations because of their significant orbital errors due to the solar radiation pressure that cannot be easily accounted for. Detailed on how to find the eclipse condition of a satellite can be found in [2], p.102.

5.2.5 Satellite and receiver phase bias

The Differential Phase Bias between satellite and receiver has been added in the new definition of our model given in equation 5.1. Also called *fractional part* of the ambiguities in [2], they represent the bias present in the antennas of the satellites and the receiver. If not accounted for, the estimation of the ambiguity N will contain them, and this ambiguity can not be viewed as an integer anymore. However, this is not a real problem for us here since we are already assuming our ambiguity as a float number. Indeed, we said in section 5.1.1 that we are only *estimating* and not *solving* the ambiguities. Moreover, we are using here the ionosphere-free combination, and our ambiguity has already lost its integer quality. This is why in the model developed by [2] for kinematic-PPP, no phase bias is being estimated, since kinematic-PPP is not precise enough to account for those.

Nonetheless, since it can still create bias in our float ambiguities, it is always interesting to remove as many errors as possible. Therefore I wanted to point out that since we are receiving RTCM streams for precise corrections, we can also receive the satellite phase biases computed by centers in real-time, since this part of the biases is common to every receiver. They are sent in messages 1059 (GPS), 1242 (GALILEO) and 1260 (BEIDOU), that the mount point "CLK93" managed by the CNES. For high accuracy purposes, these streams can be decoded in the app once the PPP algorithm is being set up inside, as it was done for the Differential Code Bias (see section 4.2.2).

5.2.6 Earth deformation effects modeling

Earth tide, *ocean loading* and *pole tide*, summarized as *earth deformation* effects, are essential phenomena to model and to take into account when performing PPP, to remove any error that is not related directly to the receiver noise. However, these errors start becoming visible when reaching sub-decimeter level precision [2], which is still a bit far from what we can expect from a PPP-RTK with only *estimation* of the phase ambiguities (i.e., no *resolution* yet). These can be neglected at first when the model is being set up. However, it will be interesting to add it once the ambiguity resolution has been reached, as it would greatly improve the vertical component of the position.

5.2.7 Model redefinition and linear form

By taken into account the different corrections we talked about in the previous sections, we can re-write the measurement model as follow

$$\begin{aligned} P_3^j &= \rho^j + c \cdot (\delta t_r - \delta t^j) + Tr_0(E^j) + M_{wet}(E^j) \cdot \Delta Tr_{z,wet} \\ \Phi_3^j &= \rho^j + c \cdot (\delta t_r - \delta t^j) + Tr_0(E^j) + M_{wet}(E^j) \cdot \Delta Tr_{z,wet} + \lambda_3 \cdot N_3^j. \end{aligned} \tag{5.7}$$

Leaving us with only the parameters that we are going to estimate. As it's been done in section 3.3.1 for our previous model, we can linearized this equation as a system $Y = G \cdot X$.

$$X = \begin{bmatrix} x & y & z & c \cdot \delta t_r & c \cdot GGTO & c \cdot GBTO & \Delta Tr_{z,wet} & \lambda_3 \cdot N^1 & \dots & \lambda_3 \cdot N^j \end{bmatrix}^T \tag{5.8}$$

$$Y = \begin{bmatrix} \vdots & \\ P_i^{j,GPS} - \rho^{j,GPS} - c\cdot\delta t_r + c\cdot\delta t^{j,GPS} - Tr_0(E^j) & \\ \Phi_i^{j,GPS} - \rho^{j,GPS} - c\cdot\delta t_r + c\cdot\delta t^{j,GPS} - Tr_0(E^j) - \lambda_3\cdot N_3^{j,GPS} & \\ \vdots & \\ P_i^{j,GAL} - \rho^{j,GAL} - c\cdot\delta t_r + c\cdot\delta t^{j,GAL} - Tr_0(E^j) & -c\cdot GGTO \\ \Phi_i^{j,GAL} - \rho^{j,GAL} - c\cdot\delta t_r + c\cdot\delta t^{j,GAL} - Tr_0(E^j) - \lambda_3\cdot N_3^{j,GAL} & -c\cdot GGTO \\ \vdots & \\ P_i^{j,BDS} - \rho^{j,BDS} - c\cdot\delta t_r + c\cdot\delta t^{j,BDS} - Tr_0(E^j) & -c\cdot GBTO \\ \Phi_i^{j,BDS} - \rho^{j,BDS} - c\cdot\delta t_r + c\cdot\delta t^{j,BDS} - Tr_0(E^j) - \lambda_3\cdot N_3^{j,BDS} & -c\cdot GBTO \\ \vdots & \end{bmatrix}_{n\times 1} \tag{5.9}$$

$$H = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{x_r - x^{j,GPS}}{\rho^{j,GPS}} & \frac{y_r - y^{j,GPS}}{\rho^{j,GPS}} & \frac{z_r - z^{j,GPS}}{\rho^{j,GPS}} & 1 & 0 & 0 & M_{wet}(E^j) & 0 & \dots & 0 & \dots & 0 \\ \frac{x_r - x^{j,GPS}}{\rho^{j,GPS}} & \frac{y_r - y^{j,GPS}}{\rho^{j,GPS}} & \frac{z_r - z^{j,GPS}}{\rho^{j,GPS}} & 1 & 0 & 0 & M_{wet}(E^j) & 0 & \dots & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{x_r - x^{j,GAL}}{\rho^{j,GAL}} & \frac{y_r - y^{j,GAL}}{\rho^{j,GAL}} & \frac{z_r - z^{j,GAL}}{\rho^{j,GAL}} & 1 & 1 & 0 & M_{wet}(E^j) & 0 & \dots & 0 & \dots & 0 \\ \frac{x_r - x^{j,GAL}}{\rho^{j,GAL}} & \frac{y_r - y^{j,GAL}}{\rho^{j,GAL}} & \frac{z_r - z^{j,GAL}}{\rho^{j,GAL}} & 1 & 1 & 0 & M_{wet}(E^j) & 0 & \dots & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{x_r - x^{j,BDS}}{\rho^{j,BDS}} & \frac{y_r - y^{j,BDS}}{\rho^{j,BDS}} & \frac{z_r - z^{j,BDS}}{\rho^{j,BDS}} & 1 & 0 & 1 & M_{wet}(E^j) & 0 & \dots & 0 & \dots & 0 \\ \frac{x_r - x^{j,BDS}}{\rho^{j,BDS}} & \frac{y_r - y^{j,BDS}}{\rho^{j,BDS}} & \frac{z_r - z^{j,BDS}}{\rho^{j,BDS}} & 1 & 0 & 1 & M_{wet}(E^j) & 0 & \dots & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}_{n\times p} \tag{5.10}$$

Where n is the number of observations, and p is the number of parameters. We also define our approximate solutions. Since we know our ambiguities multiply by the wavelength should have the same magnitude as the pseudorange measurement, they can be initialized for the first iteration with it.

$$X_0 = \begin{bmatrix} x_0 & y_0 & z_0 & 0 & 0 & 0 & 0 & P_3^1 & \dots & P_3^j \end{bmatrix}^T \tag{5.11}$$

5.3 Kalman filter

The PPP problem can hardly be solved instantaneously in one epoch computation, while it is true that recently Laurichesse at al. [27] have managed to perform it on a multi-system, multi-frequency approached. However, in general and especially for receivers sensible to significant errors in their measurements, a convergence time is needed to reach a sufficient amount of information to get a precise estimation of our parameters and before any ambiguity resolution. Since PPP has been introduced, this convergence time has been reduced more and more throughout the years. Nowadays, with a geodesic grade receiver, a convergence time of a few minutes can be expected for a decimeter level precision (see Fast-PPP, [2], p160-161). However, for receivers and antennas of lower quality, this time tends to be much higher, as we talked about during the State of the Art section 1.1.

Nevertheless, one of the ways to achieve this convergence is through a Kalman filter, which its prediction model is going to be described in this section. The equations developed in section 4.1 for resolution of the system are similar here, and will not be written again in this section.

5.3.1 Prediction model

The transition matrix F is first defined, taking back what has been defined in equation 4.7. Because of the difficulty to converge to a correct estimation of the N ambiguity, we first assume that we are in static mode. Once the precision is sufficient, we could go into dynamic mode with maximum precision, but we are assuming simple conditions here, as we will see that PPP estimation on smartphone in real-time is far from trivial. Except for our receiver clock, everything else is assumed constant here, especially the ambiguities N. Regarding the Q representing the process noise, only the receiver clock and the tropospheric coefficient are can change between epochs. The values are taken from [2], p.154.

$$F = \begin{bmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & 1 & & & & & & \\ & & & 0 & & & & & \\ & & & & 1 & & & & \\ & & & & & 1 & & & \\ & & & & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{bmatrix}_{p \times p} \tag{5.12}$$

$$Q = \begin{bmatrix} 0 & & & & & & & & \\ & 0 & & & & & & & \\ & & 0 & & & & & & \\ & & & \sigma_{cdt_r} & & & & & \\ & & & & 0 & & & & \\ & & & & & 0 & & & \\ & & & & & & \sigma_{\Delta t} + \frac{d\sigma_{\Delta t}}{dt} & & \\ & & & & & & & 0 & \\ & & & & & & & & \ddots \\ & & & & & & & & 0 \end{bmatrix}_{p \times p} \tag{5.13}$$

$$\sigma_{c \cdot dt_r} = 1ms \cdot c \quad \sigma_{\Delta t} = 0.25m \quad \frac{d\sigma_{\Delta t}}{dt} = 0.01m/h \approx 2.78 \cdot 10^{-6} m/s \text{ [2]} \tag{5.14}$$

Note that σ_{GGTO} and σ_{GBTO} is also assume to be 0 here. According to [24], its stability is expressed as an Allan deviation and should be better than $8 \cdot 10^{-14}$ over one day.

For the first iteration, the precision of our a priori values need to be estimated in order to initialize the P_0 matrix. Values have been chosen by using the reference given next to it.

$$P_0 = \begin{bmatrix} \sigma^2_{0,x} & & & & & & & & \\ & \sigma^2_{0,y} & & & & & & & \\ & & \sigma^2_{0,z} & & & & & & \\ & & & \sigma^2_{0,cdt_r} & & & & & \\ & & & & \sigma^2_{0,GGTO} & & & & \\ & & & & & \sigma^2_{0,GBTO} & & & \\ & & & & & & \sigma^2_{0,\Delta t} & & \\ & & & & & & & \sigma^2_{0,N^1} & \\ & & & & & & & & \ddots \\ & & & & & & & & \sigma^2_{0,N^j} \end{bmatrix}_{p \times p} \tag{5.15}$$

$$\begin{aligned}
\sigma_{0,x} = \sigma_{0,y} = \sigma_{0,z} &= 10m \text{ [2]} \\
\sigma_{0,cdt_r} &= 1ms \cdot c \text{ [2]} \\
\sigma_{0,GGTO} &= 100ns \cdot c \text{ [24]} \\
\sigma_{0,GBTO} &= 100ns \cdot c \text{ [24]} \\
\sigma_{0,\Delta t} &= 0.25m \text{ [2]} \\
\sigma_{0,N^j} &= 0m \text{ [2]}
\end{aligned} \tag{5.16}$$

As we detailed before, the system can then be solved using the equations defined in section 4.1 and the system should start slowly converging after several minutes.

5.3.2 Smart handling of ambiguities

However, all this is assuming that no cycle slips happened. Since we are dealing with the low-quality GNSS antenna present in the phone, it is common to see cycle slips in the measurements as we saw in section 5.2.4. When a cycle occurs, ambiguities can be assumed as white noise, meaning $\sigma_{0,N^j} = 1e^4m$ [2] for example, to reduce the weight of this measurement in the system.

Since everything is to be done in real-time, we need to account for new satellites appearing and disappearing in the sky. A new satellite means that our state vector size will need to be increased for a new ambiguity estimation. Satellites disappearing will mean that we have to remove its ambiguity from the estimation. These problems are more related to code development of the algorithm, but will need to be answered once the PPP capacities are added to the application.

Conclusion

The objective of this internship was the integration of the GALILEO signals received by a smartphone inside the Android application GeolocPVT for real-time positioning. The device studied was the Xiaomi Mi8 with dual-frequency capacities, which receive GALILEO signals but does not decode the navigation message from the signal. We showed that the ephemeris information of all the constellation could be retrieved by using RTCM streams sent by the IGS-RTS project. Moreover, precise corrections could also be retrieved, allowing users to perform PPP-RTK in the future. After the integration of those streams inside the application, we demonstrate that adding GALILEO signals enhanced the precision of our computed position greatly. We also performed the integration of BEIDOU constellation in order to enhance the multi-constellation capacities of the GeolocPVT app. This answered the main objective of this internship.

While the accuracy of the final position still contains offsets for unknown reasons, the precision of our positioning continued to be enhanced by adding phase smoothing capabilities. Precise kinematic positioning is now possible in real-time in the app by estimation of our velocity with a TDCP algorithm, using the precise phase measurements. A Kalman filter was designed for this, leading to a convergence on a precise position by accumulating information over epochs. We were able to tune the filter for pedestrian navigation, and tests in an open-sky environment have been conducted to analyze the app capacities. Future developments should focus on developing this algorithm on more challenging environments, e.g., urban canyoning, and with the presence of significant masks like the human body, as detailed in chapter 1. Integration of inertial data in the solution should also be looked into for this, for coupling of GNSS and INS, as it has shown excellent results in other studies [44].

A PPP algorithm also have been developed, PPP-RTK becoming now possible with the precise corrections retrieved in real-time by the phone. Due to lack of time, this PPP algorithm could not be integrated into the app yet, although the mathematical model has been described in length during the last chapter of this research paper. However, before integration of this algorithm, several points will need to be answered, i.e., the errors in satellites positions evoked in chapter 2 and the offset present in the north direction visible in the solutions from chapter 3 and 4. I believed these errors are linked and that answering the first one should correct the offset issue.

This project allowed me to extend my knowledge greatly in GNSS positioning, especially on the PPP technique. The future of GNSS positioning lies in the development of this technique. I learned the theory of it during my year at ENSG, and I was very interested in going in-depth in it during this internship. While PPP algorithms are getting better, it is still a challenge when it comes to low-cost GNSS receivers and antenna. However, many research teams around the world are studying it, and new smartphones like the Xiaomi Mi8 with enhanced GNSS capacities are coming out, increasing the positioning capacities of those receivers and getting closer to real-time PPP. This internship also allowed me to be part of a research laboratory and team. I have been interested in doing a PhD thesis after my engineering degree for a long time, and this internship comforted me in doing it.

Bibliography

[1] European GNSS Agency. *Using GNSS Raw Measurements on Android devices*. 2017. URL: https://www.gsa.europa.eu/system/files/reports/gnss_raw_measurement_web.pdf.

[2] European Space Agency. *GNSS Data Processing, Volume I: Fundamentals and Algotithms*. 2013. URL: https://gssc.esa.int/navipedia/GNSS_Book/ESA_GNSS-Book_TM-23_Vol_I.pdf.

[3] European Space Agency. *Signal-in-Space - Interface Control Document*. European Union. URL: https://www.gsc-europa.eu/system/files/galileo_documents/Galileo-OS-SIS-ICD.pdf.

[4] Angermann et al. *ITRS Combination Center at DGFI: A Terrestrial Reference Frame Realization 2003*. 2004.

[5] BKG. *BKG Ntrip Client (BNC)*. URL: https://igs.bkg.bund.de/ntrip/download.

[6] Pierre Bosser. *Positionnement Ponctuel Précis*. 2018.

[7] Robert Grover Brown and Patrick Y.C Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. 2012.

[8] Mohamed Elsobeiey and Salim Al-Harbi. "Performance of real-time Precise Point Positioning using IGSreal-time service". In: (2015).

[9] ESA. *GALILEO's Clocks*. URL: https://www.esa.int/Our_Activities/Navigation/Galileo/Galileo_s_clocks.

[10] ESA. *Navipedia*. URL: https://gssc.esa.int/navipedia/index.php/.

[11] Jay A. Farrel. *Aided Navigation, GPS with High Rate Sensors*. 2008.

[12] Marco Fortunato et al. "Precise Point Positioning Using Dual-Frequency GNSS Observations on Smartphone". In: (2019).

[13] Salvatore Gaglione. "How does a GNSS receiver estimate velocity?" In: *Inside GNSS* (2015). URL: https://insidegnss.com/how-does-a-gnss-receiver-estimate-velocity/.

[14] The Galfins. *GNSS Compare*. 2018. URL: https://gnss-compare.readthedocs.io/en/latest/description.html.

[15] The Galfins. *GNSS Compare*. URL: https://github.com/TheGalfins/GNSS_Compare/.

[16] Gaetano Galluzzo, Navarro-Gallardo Moises, and Martin Sunkevic. *Tutorial Using GNSS Raw Measurements on Android Devices*. URL: http://ipin2018.ifsttar.fr/fileadmin/contributeurs/IPIN2018/Tutorials/Using_GNSS_Raw_Measurements_on_Android_Devices_-_IPIN_2018_Tutorial_FV.pdf.

[17] Sheng Gao and Mitsinjosoa Ramandaniaina. *Récupération de données de navigation sur Android 8*. 2018.

[18] Miquel Garcia. *GNSS carrier phase from Nexus 9*. 2017. URL: https://www.rokubun.cat/gnss-carrier-phase-nexus-9/.

[19] M. Ge et al. "Resolution of GPS carrier-phase ambiguities in Precise PointPositioning (PPP) with daily observations". In: *Journal of Geodesy, Volume 82, Issue 7* (2008). URL: https://link.springer.com/article/10.1007/s00190-007-0187-4.

[20] Yulong Ge et al. "The Impact of Satellite Time Group Delay and Inter-Frequency Differential Code Bias Corrections on Multi-GNSS Combined Positioning". In: *Sensors* 17 (Mar. 2017), p. 602. DOI: 10.3390/s17030602.

[21] Google. *GnssMeasurement*. URL: https://developer.android.com/reference/android/location/GnssMeasurement.

[22] Maruti Gupta, Ali Taha Koc, and Rath Vannithamby. "Analyzing Mobile Applications and Power Consumption on SmartPhoneover LTE network". In: (2011).

[23] Tomasz Hadas and Jaroslaw Bosy. "IGS RTS precise orbits and clocks verification and quality degradation over time". In: (2014).

[24] Jörg H Hahn and Edward D. Powers. "Implementation of the GPS to Galileo Time Offset (GGTO)". In: (2005). URL: https://www.researchgate.net/publication/4212833_Implementation_of_the_GPS_to_Galileo_time_offset_GGTO.

[25] Martin Håkansson et al. "Review of code and phase biases in multi-GNSS positioning". In: (2016).

[26] Martin Håkansson et al. "Review of code and phase biases in multi-GNSS positioning". In: (2016).

[27] Denis Laurichesse and Simon Banville. "Innovation: Instantaneous centimeter-level multi-frequency precise point positioning". In: *GPS World* (2018). URL: https://www.gpsworld.com/innovation-instantaneous-centimeter-level-multi-frequency-precise-point-positioning/.

[28] Denis Laurichesse et al. "Integer Ambiguity Resolution on Undifferenced GPS Phase Measurements and Its Application to PPP and Satellite Precise Orbit Determination". In: *Navigation, Volume 56, Issue 2* (2009). URL: https://doi.org/10.1002/j.2161-4296.2009.tb01750.x.

[29] Wanke Liu et al. "Quality analysis of multi-GNSS raw observations and a velocity-aided positioning approach based on smartphones". In: *Elsevier* (2019).

[30] Moises Navarro-Gallardo. *Guidelines OS NMA implementation in smartphones*. URL: https://www.gsa.europa.eu/sites/default/files/expo/2.4_moises_navarro-gallardo_-_airbus_-_guidelines_os_nma_implementation_in_smartphones.pdf.

[31] Gabriele Pirazzi et al. "Preliminary performance analysis with a GPS+Galileo enabled chipset embedded in a smartphone". In: (2017).

[32] A. Privat, M. Pascaud, and D. Laurichesse. "Innovative Smartphone Applications for Precise Point Positioning". In: (2018).

[33] QZSS. *List of Positioning Satellites*. URL: http://qzss.go.jp/en/technical/satellites/index.html.

[34] Mirko Reguzzoni and Dominioni Teruzzi. *GoGPS*. Geomatics Laboratory of Politecnico di Milano, Italy. URL: https://github.com/goGPS-Project/goGPS_Java.

[35] José Gilberto Resendiz Fonseca. *Implementation des algorithmes de géolocalisation sur Android*. 2018.

[36] Umberto Robustelli, Valerio Baiocchi, and Giovanni Pugliano. "Assessment of Dual Frequency GNSS Observations from a Xiaomi Mi 8 Android Smartphone and Positioning Performance Analysis". In: (2019).

[37] RTCM. *RTCM 10403.3, Differential GNSS (Global Navigation Satellite Systems) Services*. Radio Technical Commission For Maritime Services. 2016. URL: http://www.rtcm.org/differential-global-navigation-satellite--dgnss--standards.html.

[38] Martin Schmitz. *RTCM State Space Representation Messages, Status and Plans*. Geo++ GmbH, 2012. URL: www.geopp.com/pdf/gpp_ppprtk12_msg_f.pdf.

[39] Tomoji Takasu. *RTKLIB*. URL: https://github.com/tomojitakasu/RTKLIB/tree/rtklib_2.4.3.

[40] Electronics Tutorials. *Signed Binary Numbers*. URL: https://www.electronics-tutorials.ws/binary/signed-binary-numbers.html.

[41] Pavel Vaclavovic. *Analyses of raw GNSS observations collected by the Android device Xiaomi Mi8*. RIGTC, 2019.

[42] Zhiyu Wang et al. "Assessment of Multiple GNSS Real-Time SSR Products from Different Analysis Centers". In: (2018).

[43] Qiong Wu et al. "Enabling High Accuracy Dynamic Applications in Urban Environments Using PPP and RTK on Android Multi-Frequency and Multi-GNSS Smartphones". In: (2019).

[44] Feng Zhu et al. "Walker: Continuous and Precise Navigation by Fusing GNSS and MEMS in Smartphone Chipset for Pedestrians". In: *Remote sensing* (2019). URL: https://doi.org/10.3390/rs11020139.

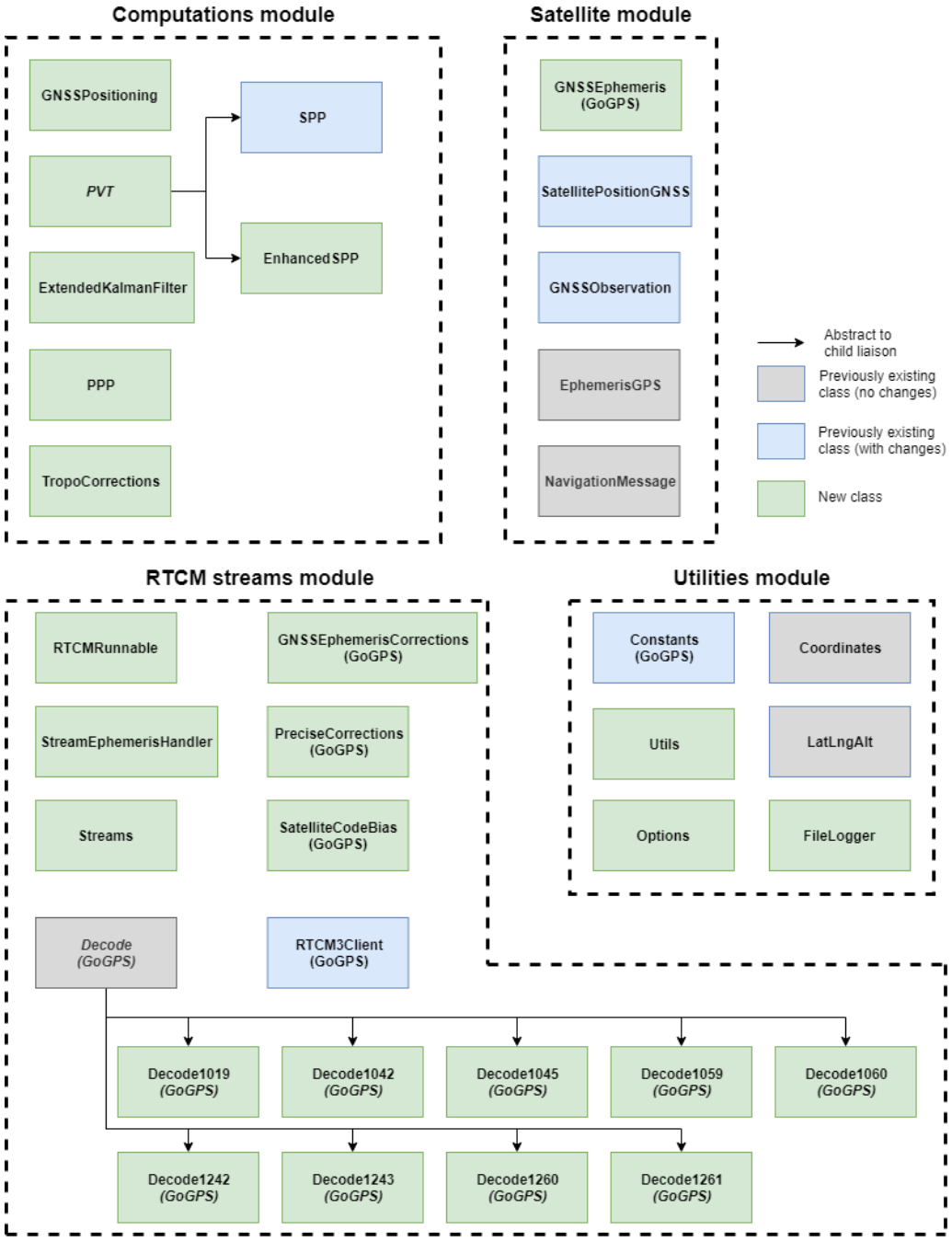
[45] Ni Zhu. "GNSS Propagation Modeling In Constrained Environments: Contribution to the Improvements of the Geolocation Service Quality". 2018.

Annexes

Simplified class diagram

Appendix A

Class diagram of the GeolocLib library, containing the computation functions for the GeolocPVT app. To make it easier to read, the dependencies liaisons have not been displayed, nor the methods and the variables. Only liaisons between abstract and child classes are represented.



Troposphere coefficients

Appendix B

Extracted from the ESA GNSS Book [2], p.126.

Table 5.2: Coefficients of the hydrostatic mapping function.

Coefficient	Latitude (φ)				
ξ	15°	30°	45°	60°	75°
	Average				
a	1.2769934e-3	1.2683230e-3	1.2465397e-3	1.2196049e-3	1.2045996e-3
b	2.9153695e-3	2.9152299e-3	2.9288445e-3	2.9022565e-3	2.9024912e-3
c	62.610505e-3	62.837393e-3	63.721774e-3	63.824265e-3	64.258455e-3
	Amplitude				
a	0.0	1.2709626e-5	2.6523662e-5	3.4000452e-5	4.1202191e-5
b	0.0	2.1414979e-5	3.0160779e-5	7.2562722e-5	11.723375e-5
c	0.0	9.0128400e-5	4.3497037e-5	84.795348e-5	170.37206e-5
	Height correction				
a_{ht}	2.53e-5				
b_{ht}	5.49e-3				
c_{ht}	1.14e-3				

Table 5.3: Coefficients of the wet mapping function.

Coefficient	Latitude (φ)				
ξ	15°	30°	45°	60°	75°
a	5.8021897e-4	5.6794847e-4	5.8118019e-4	5.9727542e-4	6.1641693e-4
b	1.4275268e-3	1.5138625e-3	1.4572752e-3	1.5007428e-3	1.7599082e-3
c	4.3472961e-2	4.6729510e-2	4.3908931e-2	4.4626982e-2	5.4736038e-2

Article: Efficient Use of SSR RTCM Streams For Real-Time Precise Point Positioning on Smartphones

Appendix C

This article is to be published to the WPNC 2019 conference and has been added to explain further how streams can be used in IoT devices. Since it is not published yet, its access here is **CONFIDENTIAL**.

Efficient Use of SSR RTCM Streams For Real-Time Precise Point Positioning on Smartphones

Grenier Antoine
GEOLOC
IFSTTAR
Nantes, France
antoine.grenier@laposte.net

Renaudin Valérie
GEOLOC
IFSTTAR
Nantes, France
valerie.renaudin@ifsttar.fr

***Abstract*—Since the availability of GNSS raw measurements from the Google Nougat API in 2016, researches have been assessing the smartphones performances and GNSS data's quality. The goal since then is to compute a precise position and to assess with exactitude the quality of this computation. As the Internet of Things (IoT) grows more everyday, embedded sensors spread everywhere to acquire, assess and monitor our environment, in which low-cost and precise positioning is a mandatory step toward this goal. The low-cost adjective here does not only relate to the financial cost of the GNSS receiver, but comprises also other costs like Internet data consumption, battery consumption, computations cost. Using the Google API and new generation smartphones, those costs can be assessed to produce smarter, more efficient and optimized positioning solution, which will later be ported on other IoT devices.**

This paper presents the group research on the development of real-time Precise Point Positioning (PPP-RTK) in an Android application. The RTCM streams are presented, leading to an analysis of their benefits for efficient positioning. Focus is given on assessing the cost of precise products in terms of internet data consumption.

***Index Terms*—PPP, PPP-RTK, Android Raw Measurements, RTCM streams, IGS-RTS**

I. Introduction

With the growing use of IoT devices for research and mass-market purposes, the need for low-cost precise positioning is increasing significantly. In 2016, Google announced the availability of GNSS raw measurements directly through the Google API, starting with Android version 7 (Nougat). This leads to the development of several positioning apps (GNSS Compare [1], PPP Wizlite [2], ...) used to assess for the first time the quality of GNSS signals received on smartphones. DGNSS-RTK technique was used to compute positions from those measurements, either in post-processing or in real time [3]. However, real-time PPP or PPP-RTK are still not as easily manageable as DGNSS-RTK since measurements corrections of atmospheric effects needed to be modeled

Following that, Xiaomi released a new smartphone "Xiaomi Mi8" in June 2018, integrating a Broadcom BCM4775X GNSS receiver with dual-frequency (L1/L5) capacities. This was an important step in smartphones and IoT developments, opening the way for highly precise positioning algorithm. With multiple frequency, signal combination, which is mandatory for ambiguity fixation and signal corrections (e.g. Iono-free combination), is possible.

Following the release of the phone, quality assessment of the GNSS receiver, in particular phase measurements' quality, has been done in several studies [4]–[6], where post processing PPP using multiple constellations and frequencies have been realized too. Robustelli et al.[4] presented a positioning solution with ambiguity estimation, also known as *floating* ambiguities, with the Mi8 using multiple constellation (GPS, GLONASS, GALILEO) and only L1/E1 measurements. Using RTKLIB as a post processing software, they achieved a sub-meter positioning in low-multipath environment after 1 hour of convergence time, showing that multi-constellation improved positioning compared to GPS only solution, which highlights the need for multi-constellations when computing PPP. Using a different model, Wu et al. [5] achieved a sub-meter positioning after a convergence time of 102 min, rapidly converging afterwards to a 0.2 m accuracy after 116 min.

Nottingham Scientific Limited team (NSL) with the FLAMINGO initiative [6] (Fulfilling enhanced Location Accuracy in the Massmarket through Initial GalileO services) performed a kinematic positioning reproducing the movement of a pedestrian walking on straight line. Although their protocol is different from previous studies, they showed a final RMSE of 2.23m using PPP, largely impacted by the multipath effects of the experience environment. The positioning computations were once again performed on RTKLIB in post processing. It is important to note that the FLAMINGO initiative final goal is to develop a complete SDK (Software Development Kit) to be integrated inside applications with a positioning accuracy of a few decimeters in real time. Right now this initiative is still in the early stage of development.

All these examples were performed in post processing mode and not in real-time aboard the phone, resulting in more manageable PPP computations. Indeed, once in real-time, computations need to account for other corrections, starting with the lack of final precise orbits from the IGS (International GNSS Service). This leads to difference in satellite positions computations and worse accuracy of those positions,, but also to more IoT related issues like Internet data consumption on-board the app or battery consumption as well.

GEOLOC is currently developing an Android application,

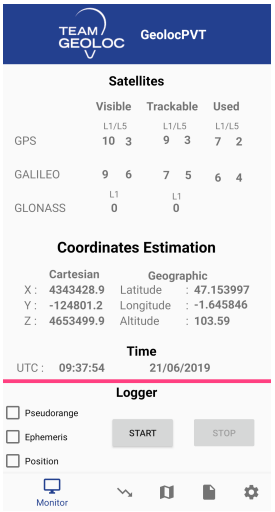


Fig. 1. Screenshot of the GeolocPVT Android application

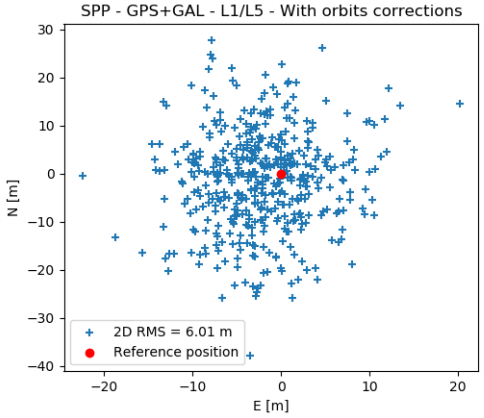


Fig. 2. Scatter plot a L1/L5 solution using both GPS and GALILEO signals. Blue cross represent a single epoch solution, the red point is the reference position found using a geodesic grade receiver and assumed to be exact

called "GeolocPVT" (fig. 1), using raw measurements for precise positioning. The app current development allows a position computation with a HRMS (Horizontal Root Mean Square) of about 6 meters in an open sky environment. A plot of the positioning capacities of the app is given in fig. 2. This is done using the IGS Real Time Service (IGS-RTS) products, sending navigation data under the Radio Technical Commission for Maritimes Services (RTCM) protocol also known as *RTCM streams*. It is through the integration of those streams and the amount of data needed to use them continuously that the question of how to optimize the stream retrieval became a challenge for the app development. The solutions to account for lack of precise ephemeris in real time are detailed in section II, where the basics of State Space Representation (SSR) streams are reviewed. Focus is then given on assessing the internet data needs of real time precise orbits and how these data consumption can be reduced with efficient use of RTCM streams. An integration of those streams inside an Android application is explained in section IV and details about the future app developments complete this paper.

II. Real-time precise orbits and clocks

For precise positioning, precise satellites' orbits and clocks are needed, since errors on those will directly impact the accuracy of the user's position. The accuracy of the broadcast orbits' is only about 1m/5ns and therefore not sufficient alone to be used in highly precise applications.

A. Ultra-rapid IGS products

Precise products are diffused by the IGS (International GNSS Service), with different accuracy depending on the observation date. *Final* products, which give the best accuracy (about 2.5cm on orbits/75ps on satellite clocks), are only available after 15 days. For real time purposes, earlier products need to be used, like *ultra-rapid* products. They are predicted, meaning they are based on previous observations and not the actual observations, and are available in real time but less precise (about 5cm/3ns).

Those products are available through the FTP servers of the IGS network in SP3 format. Files contain a position for each satellite of the constellation assigned to a timestamp in UTC, with a sampling rate of 15min. When one wants to compute the satellites' positions for a certain time, a 10th-order polynomial interpolation function is usually applied [7]. While this might work correctly for satellites positions, the interpolation process is not recommended for satellites clocks, also contained in ultra-rapid SP3 file, but with a sampling rate of 15min as well. According to ESA [7], products with a low sampling rate (i.e. greater than 30s) should not be interpolated because their evolution corresponds to a random walk process. This highlights an issue in using IGS ultra-rapid products for real-time PPP, since precise clocks estimation is mandatory for precise positioning.

Finally, it's important to notice that those files are only available for GPS satellites and no other constellations on the IGS servers. As mentioned before, multiple constellation is the key for PPP, especially in constrained environments like urban canyon, which is where a smartphone is most likely to operate. This emphasizes another problem with ultra-rapid products, which is their availability for all constellations.

B. IGS-RTS products

Since 2001 [8], IGS has also been developing another service called Real Time Service (RTS). Its goal is to diffuse real-time products to users to perform real-time positioning. Joining the RTCM, they developed the State Space Representation (SSR) standard, enabling the diffusion of orbits and clocks corrections through the Internet using bytes streams.

RTCM protocol is divided in messages, each having an ID used to identify their GNSS contents. For example, GPS constellation corrections are in the messages 1057 to 1062 [9].

- 1057 - SSR GPS Orbit Correction

- 1058 - SSR GPS Clock Correction
- 1059 - GPS Code Bias
- 1060 - SSR GPS Combined Orbit and Clock Corrections
- 1061 - SSR GPS URA
- 1062 - SSR GPS High Rate Clock Correction
- 1019 - Broadcast GPS orbits

Similar messages with different IDs can be found for other constellations: GLONASS (1063-1068), GALILEO (1240-1245), QZSS (1246-1251) and BDS (1258-1263). The RTCM standards version 10403.3 [10] details the contents of each message. Note that message 1019 is not really in the SSR section of RTCM protocol, but will need to be retrieved for satellites coordinates computations in real time.

The IGS-RTS requires special identification to retrieve those streams and as for now only research purposes projects are granted access. For this study, a parallel access to three streams was granted to the IGS server "rt.igs.org".

The IGS-RTS offers different server to be connected to, each one offering a list of *mount points* (i.e. casters). Each of those corresponds either to a permanent base site or a processing center. They will send certain types of message types, with a certain rates, depending on the mount points specifications. This means that not all streams contain the same messages, therefore not all constellations either. Using the BKG software "BKG NTRIP Client" or BNC [11], an open sourced NTRIP software, one can look into the content of the stream sent by each point, to find the most suitable stream to receive the desired corrections.

C. Benefits of the IGS-RTS

Using this service, it becomes possible to retrieve the orbits and clocks information of all systems, depending on which mount points the device is connected to. According to Wang et al.[12], the mount point CLK93 from the CNES (Centre National d'Etude Spatiale, France) is one of the only mount point sending corrections for GPS, GLONASS, GALILEO and BEIDOU constellations. This was verified with the introduction of these streams in the GeolocPVT app, to retrieve GPS and GALILEO corrections using this mount point. For receiving messages 1019 and 1045 (GALILEO) containing the broadcast ephemeris data, the "TLSG00FRA0" mount point was used. However, following changes on the used IGS server (rt.igs.org), this mount point became unavailable and instead the mount point named "RTCM3EPH" was used (see section III-D).

Another benefit of using the streams corrections is the final precision obtained for satellite positions and clocks. IGS-RTS products are as accurate as the predicted half of the ultra-rapid products [13] while offering satellite clocks corrections with a much higher sampling rate. This depends on the mount points specifications. A 5 seconds rate is common for clocks corrections to avoid inaccuracies inherited with longer interpolation [8]. This is a huge improvement compare to the ultra-rapid products, satellites clocks being one of the largest error source in positioning that is not canceled out in PPP like it would be in DGNSS.

III. Data consumption in positioning

In the IoT, an important variable when setting up a device is cost. Financial cost, but also energy and data costs. As IoT's devices are meant to be interconnected, Internet data usage need to be reduced, especially for completely independent devices, that must rely on efficient data and power management. In this paper, we focus on Internet data consumption, leaving out the power management, although those two are correlated with each other since higher internet consumption will induce higher battery usage [14].

We estimate the amount of necessary data to perform a multi-constellation PPP computation with a 1h convergence time. This duration was chosen according to state of the art studies, showing a 30 minutes minimum time to convergence in PPP-static mode using the IGS-RTS products [12] with geodesic grade receiver for a precision of decimeter-level, and up to nearly 2h using Xiaomi Mi8 phone's GNSS receiver [5].

A. Ultra-rapid products data requirements

IGS FTP mirror sites provide .SP3 ultra rapid products of about 186 KB (i.e. 191 000 Bytes). It contains precise positioning for 48h: the first 24h are the one previous to the uploaded time, realized with actual observations. These are not useful for us since we are doing real time positioning. The other 24h are predicted positions based of the past measurements. New versions are available every 6h. They are more precise than the previous ones since they are based on more recent observations. This means that every 6h a new file needs to be downloaded to have the most recent and precise data set. Furthermore, this only concerns the GPS constellation, since ultra-rapid products are made for the GPS constellation only. In theory, if the products existed, this amount of data would need to be multiplied by the number of constellations used in the computations.

B. RTCM streams data requirements

Regarding the streams, an accurate calculation of the amount of data used is a bit less trivial. Each message is encoded on certain amount of bits (8 bits = 1 bytes). Sizes are listed in table I, with their equivalents in Bytes. One message contains the info for one satellite. It is important to multiply by the number of satellites n_{sat} contained in the stream to find the size per message type. Depending on the mount point, the number of satellite varies. For example, some mount points only send the broadcast ephemeris of the satellites in view (e.g. 12 satellites for GPS). This is related to the mount points' specifications, evoked in section II. For the "RTCM3EPH" mount point, all satellites of a constellation are always present (e.g. $n_{sat} = 32$ for GPS).

Streams can be sent up to every 5 seconds (i.e. 12 streams/min or 720 streams/h), which is critical for precise satellite clocks computations. Using these presets, one can compute the amount of downloaded data when retrieving streams for 1h.

TABLE I
SSR MESSAGES SIZES [10]

Message ID	Size per satellite (bits)	Size per satellite (Bytes)
1019	488	61
1057	227	29
1058	167	21

$$n_{sat} = 32 \quad n_{streams} = 720$$
$$n_{Bytes,1019} = n_{sat} \times n_{streams} \times 61 = 1\,405\,440 \text{ Bytes} \quad (1)$$

$$n_{Bytes,1057} = n_{sat} \times n_{streams} \times 29 = 668\,160 \text{ Bytes} \quad (2)$$

$$n_{Bytes,1058} = n_{sat} \times n_{streams} \times 21 = 483\,840 \text{ Bytes} \quad (3)$$

$$n_{Bytes,total} = n_{Bytes,1019} + n_{Bytes,1057} + n_{Bytes,1058} = 2\,557\,440 \text{ Bytes} \quad (4)$$

The amount of data to retrieve them is far greater than the one needed for ultra-rapid orbits. However, looking only for the specific data in the streams' content would reduce the consumption.

C. Optimized use of RTCM streams

The broadcast ephemeris orbital parameters in message 1019 are sent every 5 seconds but updated only every 2 hours usually. Meaning that over a 1 hour interval, changes can only happen once at most, and only two messages of this type need to be downloaded through out the survey.

Secondly, according to [8], which looked into the degradation of IGS-RTS products in case of interruptions in the caster connections, it is possible to use the corrections products with longer update's time. If no model is applied, all corrections degrade quickly, with 5 cm additional error after 3 minutes for the GPS constellation. However, by fitting polynomials of degree 2 to 4 for interpolating the corrections, the study showed that the use of an orbital correction could be pushed up to 5 min for GPS satellites, staying under the 2 cm threshold of additional error. Regarding clocks corrections, since their variations can be associated to a random walk, polynomial fitting does not improve the interpolation, and other types of model are needed. Yet, judging by their results on the block II-F GPS satellites with Rubidium clocks hardware, the clocks corrections tend to be valid on longer intervals, staying below the 2 cm threshold of additional error with intervals up to 1 min. Another important information is that block II-F GPS is the only one sending both L1 and L5 signals [15]. In the future, IoT devices which will perform PPP will tend to receive both L1 and L5 signals at least, since the L5 signals offer a better resistance to environment degradation [4]. In our case, only those signals are received by the Xiaomi Mi8 with the Broadcom BCM4775X GNSS receiver. This means that PPP being only computed with satellites with both signals (for signal combination evoked in section I and ionosphere effects reduction), only GPS satellites from block II-F can be used. Therefore clocks corrections received for those satellites can be interpolated longer.

Note that [8] analysis was only performed on GPS and GLONASS constellations. GALILEO system and new generation GPS Block-III containing more accurate clocks (up to half a nanosecond [16]), the use of satellites clocks corrections could be valid on longer periods. This would need to be investigated by further studies in the future.

We can recompute eqs. (1) to (4) with a smarter streams per hour ratio.

$$n_{sat} = 32 \quad n_{streams} = 2$$
$$n_{Bytes,1019} = n_{sat} \times n_{streams} \times 61 = 3\,904 \text{ Bytes} \quad (5)$$

$$n_{sat} = 32 \quad n_{streams} = 12$$
$$n_{Bytes,1057} = n_{sat} \times n_{streams} \times 29 = 11\,136 \text{ Bytes} \quad (6)$$

$$n_{sat} = 32 \quad n_{streams} = 60$$
$$n_{Bytes,1058} = n_{sat} \times n_{streams} \times 21 = 40\,320 \text{ Bytes} \quad (7)$$

$$n_{Bytes,total} = n_{Bytes,1019} + n_{Bytes,1057} + n_{Bytes,1058} = 55\,360 \text{ Bytes} \quad (8)$$

The data consumption is strongly reduced in eq. 8 compared to eq. 4. Compared to ultra-rapid products, the data consumption with the usage of RTCM streams is nearly 3.5 times better. However, this also means that ultra-rapid product are still more interesting for long term computations. For example, for an acquisition longer than 4 h ultra-rapid products will be preferable, since streams clocks corrections can not be interpolated more than 1 min [8] and ultra-rapid products are valid for a period of 6 h before being updated. Yet, for precise satellite clock modeling, streams remain better than ultra-rapid products, which consequently provides a more accurate positioning solution. The data consumption estimations were made for computations using only the GPS constellation. But in case of multiple system being used, the IGS-RTS is the only remaining solution.

This smart use of the streams can be handled in two different ways: (1) on the client side, with connection to caster when orbits and clocks data are needed; (2) on the caster side, with corrections sent on extended periods.

D. Smart handling of streams on Client and Caster sides

Casters send streams continuously to connected devices, but without explicit requests of the devices for every new streams. One caster usually sends multiple types of messages, each with its own update rate. A device might only need to use a part of this stream, but it needs to download all the messages continuously to find the wanted one. If the caster sending data is configured to send clocks corrections every 5 seconds, orbits

corrections every 30 seconds and broadcast ephemeris every minute (a typical caster configuration), the amount of used internet data is very high, as we demonstrate in section III-B.

A first way to introduce an efficient use of RTCM streams is to adjust the caster's configuration. While configuring on a caster the exact need of a device is the most efficient way, it is not optimized and counter productive as the role of a caster is to be used by many devices with possibly different positioning protocol, algorithm and frequency update rates.

Another way is to leave this caster's configuration and to implement a connection rate on the device side. The device stays connected to receive the stream and disconnects afterwards from the caster. The re-connection would occur only when the device needs new information, preventing it to download unnecessary data. While this is a more optimized use of the device's internet data, it still downloads many unnecessary information every time it connects to the caster. It might only need clocks corrections update for example and not orbits updates. Moreover, it would multiply the connections to the casters for each connected devices, leading to an overloading of the caster, not meant to be used that way but rather as a sender to passive listener.

A third and more optimized way would be using both solutions and multiple casters instead of using only one. Different casters sending similar info can be found on the IGS-RTS servers, however all the messages are sent in high rates on all the casters. Recently, the "RTCM3EPH" mount point was added on the IGS server "rt.igs.net", offering combined broadcast orbits for all constellations (GPS, GLONASS, GALILEO, BEIDOU, QZSS) sent at a high rate of every 5 seconds, with different available mount point if only one constellation is needed (e.g. RTCM3EPH-GPS). Previous solution was to connect to a nearby caster receiving the same satellites as the device does, which usually was also sending observation data at the same time, unnecessary for PPP computations on the device and therefore a waste of data. With this new type of caster, efficient use of RTCM streams can be performed, so that only the needed data is retrieved by the IoT device.

As we mentioned before, broadcast ephemeris data only changes every 2h and its retrieval can therefore be done only once in that time. The receiver can verify the IoD of the corrections to be sure that it has the right data set. Once the IoD changed, he can reconnect to the caster and wait for the new ephemeris streams, then disconnect from it again. For data with low changing rate like the ephemeris, connection and disconnection could be performed without impacting the caster itself. However, for fast changing data like orbits and clocks corrections, the stream itself would need to be adapted. A possible way would be to set up two casters. One with a very high rate, in order to retrieve all the possible corrections in less than minute, while the other caster would have a lower rate, similar to the specifications in section III-C. Once booted up, the device would first connect to the high rate caster for initialization, then switch to the lower rate caster. The usage of internet would be minimized to the strictly required data while keeping a good precision for PPP positioning requirements.

While the device side of smart use of streams is easy to implement, the caster side might be hard to manage since they can not be regulated from the external devices. A possible answer is to set up its own caster, forwarding the data coming from a mount point in the network to another address to the desire rate. The BNC "Client" software allows this, forwarding the received corrections to a local network port where the machine is connected to. Another way is to use the BNC "Caster" software, but it must be purchased.

IV. RTCM STREAMS INTEGRATION IN ANDROID

Inside GeolocPVT application, RTCM streams are retrieved and decoded, for them to be used afterwards in real time computations. Their implementation was done by using *GoGPS*, an open source code license under the GPL available on Github and which has been ported in Java [17].

A. GoGPS as a Java Library

GoGPS Java is an adaptation of a MATLAB version and developed by [17]. It was developed for GPS positioning. Even if it wasn't developed as an Android app, it can be used as library with the app GeolocPVT developed in Java too. This gives us a very large toolbox with the functions to connect to a mount point sending RTCM streams and decode them. The functions need to be generalized and adapted for all GNSS systems since the app will use all the constellations available in the Xiaomi Mi8. GALILEO was first to be implemented and the integration of the BEIDOU system is ongoing.

The GoGPS Java port doesn't offer any documentation for implementing the library. Regarding the RTCM decoding capacities, it was coded to receive observation RTCM messages (e.g. 1004, 1005, ...) but no correction messages whatsoever (e.g. 1057,1058,...). After a stream is received, its header is decoded and compared to the contents of a Java Hashmap object, containing a list of decoding functions for each message's type. To implement the correction messages, a class was added for each message type. The decoding functions were coded using the RTCM manual [10] and the RTKLIB C++ library source code [18], which also implements SSR stream decoding in its latest version. The received streams are in a binary format (i.e. bits strings) and need to be converted to retrieve the data. RTCM standards [10] describe how this is done, the same way the GPS and GALILEO ICD's describe how to decode the navigation message. It follows the basic logic of electronics' bytes decoding [19]. Those classes were added to the HashMap object as new entries.

Connection to caster and mount points are also performed in GoGPS. First, a custom class instantiating the GoGPS Interface *StreamEventListener* needs to be created. Then, a *Runnable* class is created, adapting the code available in *RTCM3Client* to connect to desired *host* (i.e server's address) and mount point. Credentials need to be given too so the connection is authenticated.

Once those different adjustments are made, retrieval of the wanted stream can be done directly with the smartphone in real time and corrections can be computed. Detailed equations can

be found in [9]. Those corrections are then directly applied to the satellites' positions and clocks from the broadcast ephemeris.

B. Future developments

A real time PPP algorithm is in development, with ambiguities estimation, which surely would lead to better positioning capabilities where the effect of the precise orbits will be visible. Yet, because the results of several studies [4]–[6] showed long convergence time superior to one hour, data consumption is a reality to be accounted for in the app development. Smart use of the streams will be integrated in the app for a more optimized use of the internet data during surveys and testing of the PPP algorithm.

With the development of SSR to stage 2 [9], other types of corrections messages are available on "CLK93". Message 1264 gives a real time estimation of the Vertical Total Electron Content (VTEC) in the ionosphere, which is particularly useful for single frequency receivers. Messages 1265 to 1270 give a real time estimation of the satellite phase biases for the different constellations, which is also an important consideration for most PPP computations [20].

V. Conclusion

With Google giving access to raw GNSS measurements through the Android API, precise positioning using a smartphone is now a reality. We presented how this has been performed on several smartphones and why the Xiaomi Mi8 is a game changer in being the first smartphone with dual-frequency GNSS capacities. PPP is now a possible and real-time PPP will be soon be available for coming smartphones, while adding challenges to computations. Following that, we looked into the available precise products of satellites' orbits and clocks, while focusing on their internet data consumption. This was done on Android smartphone experience, even though this will apply to any IoT devices performing PPP computations. IGS-RTS products compared to ultra-rapid products have proven to be less Internet data consuming on short computation intervals if RTCM streams are gathered efficiently and interpolation are performed on the corrections. However, for continuous positioning (longer than 4h), ultra-rapid products are more suitable to minimize data consumption. Yet, for precise positioning like PPP, IGS-RTS showed better precision than the ultra-rapids products, especially on satellite's clocks estimation, while being available for nearly every type of constellation (GPS, GALILEO, QZSS, BEIDOU) on specific mount points. Finally, we showed that their retrieval directly through an Android application can be done by using the GoGPS Java code as Android library and by enhancing its capacities for retrieval of correction messages. Right now, only orbits and clocks products have been tested. For future developments, other corrections messages than orbits and clocks will be decoded, increasing the data consumption but resulting in better positioning and smaller convergence time.

References

[1] The Galfins. *GNSS Compare*. URL: https://github.com/TheGalfins/GNSS_Compare/.

[2] A. Privat, M. Pascaud, and D. Laurichesse. "Innovative Smartphone Applications for Precise Point Positioning". In: (2018).

[3] Gabriele Pirazzi, Augusto Mazzoni, Ludovico Biagi, et al. "Preliminary performance analysis with a GPS+Galileo enabled chipset embedded in a smartphone". In: (2017).

[4] Umberto Robustelli, Valerio Baiocchi, and Giovanni Pugliano. "Assessment of Dual Frequency GNSS Observations from a Xiaomi Mi 8 Android Smartphone and Positioning Performance Analysis". In: (2019).

[5] Qiong Wu, Mengfei Sun, Changjie Zhou, et al. "Enabling High Accuracy Dynamic Applications in Urban Environments Using PPP and RTK on Android Multi-Frequency and Multi-GNSS Smartphones". In: (2019).

[6] Marco Fortunato, Joshua Critchley-Marrows, Malgorzata Siutkowska, et al. "Precise Point Positioning Using Dual-Frequency GNSS Observations on Smartphone". In: (2019).

[7] European Space Agency. *GNSS Data Processing, Volume I: Fundamentals and Algotithms*. 2013. URL: https://gssc.esa.int/navipedia/GNSS_Book/ESA_GNSS-Book_TM-23_Vol_I.pdf.

[8] Tomasz Hadas and Jaroslaw Bosy. "IGS RTS precise orbits and clocks verification and quality degradation over time". In: (2014).

[9] Martin Schmitz. *RTCM State Space Representation Messages, Status and Plans*. Geo++ GmbH, 2012. URL: www.geopp.com/pdf/gpp_ppprtk12_msg_f.pdf.

[10] RTCM. *RTCM 10403.3, Differential GNSS (Global Navigation Satellite Systems) Services*. Radio Technical Commission For Maritime Services. 2016. URL: http://www.rtcm.org/differential-global-navigation-satellite--dgnss--standards.html.

[11] BKG. *BKG Ntrip Client (BNC)*. URL: https://igs.bkg.bund.de/ntrip/download.

[12] Zhiyu Wang, Zishen Li, Liang Wang, et al. "Assessment of Multiple GNSS Real-Time SSR Products from Different Analysis Centers". In: (2018).

[13] Mohamed Elsobeiey and Salim Al-Harbi. "Performance of real-time Precise Point Positioning using IGSreal-time service". In: (2015).

[14] Maruti Gupta, Ali Taha Koc, and Rath Vannithamby. "Analyzing Mobile Applications and Power Consumption on SmartPhoneover LTE network". In: (2011).

[15] QZSS. *List of Positioning Satellites*. URL: http://qzss.go.jp/en/technical/satellites/index.htmls.

[16] ESA. *GALILEO's Clocks*. URL: https://www.esa.int/Our_Activities/Navigation/Galileo/Galileo_s_clocks.

[17] Mirko Reguzzoni and Dominioni Teruzzi. *GoGPS*. Geomatics Laboratory of Politecnico di Milano, Italy. URL: https://github.com/goGPS-Project/goGPS_Java.

[18] Tomoji Takasu. *RTKLIB*. URL: https://github.com/tomojitakasu/RTKLIB/tree/rtklib_2.4.3.

[19] Electronics Tutorials. *Signed Binary Numbers*. URL: https://www.electronics-tutorials.ws/binary/signed-binary-numbers.html.

[20] Martin Håkansson, Anna B. O. Jensen, Milan Horemuz, et al. "Review of code and phase biases in multi-GNSS positioning". In: (2016).