GNSS Integrity & Robustness Tool

Albert Becerra Hervás

Resum– Les aplicacions mòbils són una de les principals tecnologies del present i del futur a causa de l'ús incremental dels *Smartphones*. Moltes d'aquestes aplicacions requereixen poder obtenir la ubicació de l'usuari i, per tant, els conceptes d'integritat i robustesa, referits a la qualitat dels resultats dels mètodes de posicionament basats en satèl·lit, són cada cop més importants. Aquest article presenta una aplicació que proporciona diverses eines per calcular la ubicació de l'usuari implementant un algorisme de posicionament configurable, que pren com a entrada les mesures obtingudes pel dispositiu i els diferents paràmetres de posicionament seleccionats per l'usuari, i retorna la ubicació amb informació sobre la fiabilitat dels resultats obtinguts. Després, l'entrada de l'algorisme s'enviarà a Firebase Cloud Storage per tal de permetre estudis de post-processat. El present document explicarà el desenvolupament de l'aplicació i la implementació al *Cloud*.

Paraules clau– Android, Kotlin, Clean Architecture, VIPER, UI/UX, Firebase Cloud Storage, Sky-plot, Dual-Frequency, mètodes de posicionament amb GNSS

Abstract– Mobile applications are one of the main technologies of the present and future due to the incremental usage of the *Smartphones*. Many of these applications require being able to obtain the user location and thus, the concepts of integrity and robustness, which are referred to the quality of the satellite-based positioning methods results, are growing in importance. This article presents a mobile application that provides several tools to compute the user location implementing a configurable positioning algorithm that takes as an input the measurements obtained by the Smartphone and the different positioning settings selected by the user and returns the computed location with information about the reliability of the obtained results. Then, the input of the algorithm will be sent to Firebase Cloud Storage in order to allow post-processing studies. The development of the Android application and the cloud implementation are explained in the present document.

Keywords– Android, Kotlin, Clean Architecture, VIPER, UI/UX, Firebase Cloud Storage, Sky-plot, Dual-Frequency, GNSS positioning methods

1 Introduction

Satellite positioning methods has been the reference since the Global Navigation Satellite System (GNSS) deployment in 1990, where the first constellation of satellites was officially complete. Many GNSS receivers were developed and, with the arrival of smartphones, which grew exponentially, everyone started to wear a receiver in the pocket that uses the GNSS signals to obtain the user location. The fact that any wearable could use satellite-based positioning methods led to some inconsistencies in GNSS positioning, because the most common usage scenarios suddenly start to be closed environments such as cities. For this reason, the integrity and robustness of the measures obtained by the receivers grew in importance. Some of the issues that appeared in cities were the multi-path [1] effects due to the signal bouncing into buildings, which caused and imprecision in the travel time and therefore the obtained position corruption. Other corruption sources of this scenarios can be the jamming [2] and spoofing [3] interference.

This document will explain the pocket solution offered to that problem, in form of an Android application named GNSS Tool, which has it's basis on the app developed on the Galileo App Competition, organized by the European Space Agency (ESA) with the collaboration of European Comission (EC) and Google. The application allows the study and analysis of the integrity and robustness of GNSS using a single Android Smartphone as a receiver. This application will use the GNSS data provided by the device and will compute the receiver's location with the usage of this data and the specification of the computation settings selected by

• E-mail de contacte: albert.becerra@e-campus.uab.cat
• Menció realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Daniel Egea Roca (TES)
• Curs 2019/20

the user. This computation settings are a set of parameters that can be configured for the algorithm execution such as the used constellations, elevation mask or dual-frequency corrections, among others.

In order to prove the integrity and robustness of the results, the user is provided with statistics graphs that shows information, based on device sensors data and the output of the algorithm, about the quality of the results. This can help the user to understand the environment requirements and to know which configuration is more suitable in each situation. For instance, if the user is in a city environment, probably it will be a better configuration if he sets an elevation mask in order to filter the signals that has more possibilities to have suffered multi-path and maybe he will also select to compute will dual-frequency approach in order to detect this multi-path effect.

Next, the motivation and objectives of this project will be defined.

2 Motivation and Objectives

The increase in the use of smartphones and, with it, the increase in the number of mobile applications that use satellite positioning methods has raised the importance of the integrity and robustness of GNSS signals. The fact that these devices are constantly used in environments such as cities, where buildings and other obstacles can affect causing the signal to bounce and arrive at two different angles to the receivers, fact known as multipath, requires an study of the quality of the obtained positions.

The motivation of this thesis is to build an Android tool and a modular Position, Velocity and Time (PVT) algorithm that helps the study of the integrity and robustness of the obtained GNSS measures and the reliability of the obtained results. At the same time, this application can help researchers to collect and analyse the measurements received by Smartphones and students to get introduced to navigation world. This application has to let the user, independently on the experience level, to configure the desired parameters of the computation and to observe the graphs in order to obtain information about the received measures and the obtained positions by the PVT algorithm. In addition, a cloud solution will be implemented in order to allow the post-processing of the obtained measurements. This will allow the researchers to obtain and process data from different environments collected by all the users without actually moving to the places.

In the figure 1, it can be seen how the satellite signals that are being received by the Smartphone are sent to the cloud for the post-processing tasks and then this measurements are being processed through the PVT algorithm of the application, which will compute the user location and print it in the application map.

Once the motivation has been established, the above considerations are reflected into the following thesis goals listed below.

- **User Interface (UI)/User Experience (UX):** Design an easy-to-use and pleasant UI , with a fluid UX even thought the high load execution processes carried out.
- **Real-Time visualization of obtained positions**

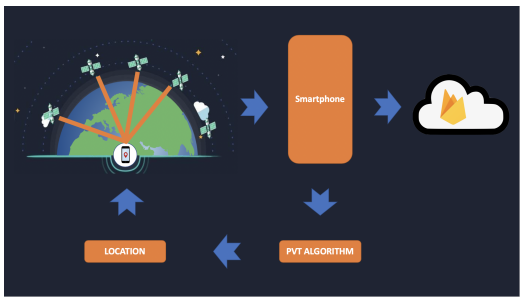


Fig. 1: Main application components.

- **Graphs:** Show information about the environment integrity and the reliability of the obtained measures and results in a set of graphs.
- **Sky-plot:** Graphic of satellite paths in the sky studied from the observation point and their Carrier-to-Noise-Density Ratio (C/N0).
- **Cloud Storage [4]:** Allow the user to save measurements and result data to the cloud in order to allow post-processing features.
- **PVT Algorithm:** Develop a modular and configurable PVT algorithm to allow the study of the integrity and robustness.

3 State of the art

In recent years, GNSS-based applications [5] have been focusing in the integrity and robustness concepts, specially in the urban environment where a great challenge is introduced to common receivers such as Smartphones. This is mainly because the GNSS positioning performance can be severely degraded by the limited satellite visibility, multipath effect and interference such as spoofing or jamming [6].

The Smartphones are becoming one of the principal consumers of the positioning methods based on GNSS measurements. The particularity on this devices is that they are being used most of the time in contaminated environments, where there are different sources of error such as high buildings which produces the multi-path effect and many interference effects. Many applications on these devices are required to obtain the user position. Some examples of that are Google maps, Whatsapp or banking apps that require precise location in order to perform some risky transactions. This location is mainly accomplished, among other methods as device sensors or 4G, by the GNSS signals so, there appears a requirement in the society of offering a precise location whether the device is being used in open-sky conditions or in harsher environments, such as urban canyons or even indoors.

In order to offer this precise location services, the Smartphones have evolved during the last years, implementing more powerful hardware and antennas to improve the quality as a receivers. One of the latest improvements that has been implemented in last generation devices is the dual-frequency antennas, which allows to discard some of the corrupted signals [7]. With this evolution, different applications has been developed to show the user data about the

state of the received GNSS signals and sensors information. The present application offers, in advantage, the possibility to select between the used constellations, bands, corrections or algorithms and to set the elevation or C/N0 masks to discard those satellites that might be corrupted due to multipath. The user is provided with a set of graphs that offers information about the reliability on the results and some sensors data to help the settings selection process. Therefore, this tool will be helpful for studying the integrity and robustness of the Smartphones positioning methods.

4 USED TECHNOLOGIES

In order to develop this product, it has been used the Android Studio tool to implement both the application and PVT module, previously developed in MATLAB, which will be used then as a post-processing tool. The used programming language is KOTLIN, which has become the first official Android Software Development Kit (SDK) language and the device where the tests will be performed will be a Xiaomi MI8, as it is one of the few Android devices which has a dual-frequency receiver.

On the server-side, it will be used the Firebase technologies because it offers many possibilities to the development as Cloud Storage, Crashlytichs, etc.

Finally, referring to the planning tasks, Trello will be used as a task tracker, and Github will be the version control system.

5 METHODOLOGY

In order to start the development of the application previously specified the first step that has been done is to think about the objectives that had to be accomplished. During the process of defining those objectives, the wire frames of the application screens has been done in order to know how to translate those objectives into a mobile screen.

A common mistake that many developers do, after defining the objectives or even before, is that they start immediately developing features without any kind of planning. This will not be a good practice in any case because it can cause problems like the late detection of unrealistic objectives or exceeding the dead-line, among many others so, here, once the objectives were defined the next step taken was to divide those objectives into different sub-projects and tasks in order no to plan the hole project all at once. To do that, the *Agile* planning methodologies has been applied with the aim to think about how to divide the goals into different pieces of valuable work and, once they has been defined, each of them could be spread into different sub-tasks. The goal of this division process is to let the time estimation process to be more precise and therefore, this has helped in order to define if the established objectives are realistic enough to start the project. Then each of this valuable work pieces, which will be referred as for now on as the *epics*, has been planned into different sprints that lasts for two weeks. The process of dividing in tasks is called *refinement* and will be explained in the following subsection. Then, once the refinement is completed, the planning has been done in order to set the order of execution.

5.1 Refinement

In the refinement all the objectives has been divided into six different phases, which are the project setup, the android application development, the Firebase Cloud integration, the PVT algorithm development, the extras implementation and the project closure. Once the different tasks has been defined, an estimation of the time required to develop each of them is done in order to help the planning, that will be explained in the section 5.2. To do this estimation, the Fibonacci sequence has been applied assuming that, as more time is needed to perform a task, more uncertainty has to be added to that time in order to prevent planning issues. Then, if a task is estimated to last for four days, according to the sequence, it has to be estimated as five days.

Next, the different phases will be explained in the following subsections.

5.1.1 Project setup

The first thing that needed to be done was the Android project creation, including all the configuration files and the version control implementation and. Then, this phase has been divided into this two single tasks without creating an epic.

5.1.2 Android Application Development

Here, as the name implies, different epic tasks had to be defined in order to reduce the uncertainty of the estimation. Therefore, the five different epics defined in the list below were created according to the different functional screens except the last one, that applies to all screens.

- Map Screen: This epic is related to the map screen development, which includes the implementation of the Google map, the functionality to print the locations and some other accesses that will be defined in section 8.2.
- Settings Screen: The settings screen needs to allow the user to select between the different computation configurations. Thus, this epic is divided into the three tasks referred to the architectural layers that will be explained in the section 6.1.
- Sky-plot Screen: The implementation of the sky-plot, defined in section 8.2, only contains a task related to the presentation layer because the domain layer has been previously developed and the data layer does not apply here.
- Statistics Screen: The objective of this screen is to provide information to the user about the reliability of the obtained result during the computation process and to show information about different sensors and data retrieved in the GNSS background service.
- GNSS Background Service: This epic is referred to the development of a background service, which will be responsible of the GNSS measurements acquisition. It has to be a background service in order to allow the application to run on background, even if the screen is off.

5.1.3 Cloud implementation

Another phase of this project was the Cloud implementation. As the objectives of this are completely unrelated processes, three different tasks had been created, one referring to the Firebase integration in the project, the another one that covered the Chrashlytichs [8] configuration and finally the last task to implement the communication between the application and Firebase Cloud Storage in order to allow the saving files process.

5.1.4 PVT Algorithm

This phase is referred to the PVT algorithm implementation in Kotlin. It was big enough to divide it into the tasks listed below. Each of this tasks, except the test one, is noted later in section 7:

- Tests: The tests task is involved during all the PVT development process because the Test Driven Development (TDD) paradigm, which will be explained later on this document, has been followed.
- Input: This is the task for the input module implementation. Here the data structure that is received from the background service developed in the application phase, needs to be converted to the data structure defined to perform the acquisition.
- Acquisition: The acquisition process needs to prepare all the data necessary to execute the PVT algorithm using the data provided by the input. Here the pseudorange estimation will be done.
- Corrections: This task is defined for the pseudorange applied corrections, that will be performed in order to obtain a more precise location. For more information about this module, see [9].
- PVT Algorithm: The implementation of the pvt algorithm was the most important task of this phase. The goal was to modularize an old PVT algorithm that has been taken as a reference in order to allow it to be configurable, testable and maintainable.
- Output: Finally, the data structure that is returned from the PVT algorithm needs to be prepared in this module, that will send the results again to the application modules, where they will be displayed.

5.1.5 Extra features

Once the project is stable, some extra features has been planed in order to complete the information that the different graphs show, save the measurements files in NMEA format or add some extra design features.

5.1.6 Project closure

Finally, this phase is dedicated to test all the different modules integrated, which leads to the problem solving tasks that are always present in the closure of a project. In order to be able to define the death line it was necessary to reserve some of the development time for the mentioned tasks.

5.2 Planning

Once all objectives has been divided into different phases of the project, which at the same time those has been divided into different epics and tasks, and an estimation of the time that they will take has been given for each one, the sprint planning has been done. Each sprint has been divided in groups of two weeks work. Then after each sprint is completed a meeting with the supervisor of the project has been done in order to review the process of the project and to communicate the different troubles that has been taken care of and that they were not planned. Some of the tasks that can be seen in section 10 needed to be discarded for the project as they did not fit in the planning.

6 Android development

This section will summarize the development of the android application module. In the first place the description of the code architecture and design patterns will be noted. Later the android modules functionality and implementation will be explained.

6.1 Code Architecture and Design Patterns

The first thing that was done before starting developing the application was to decide which architecture to follow, as it is one of the main points of a project, specially if the intention is to maintain and keep it always functional. Due to the experience in developing Android applications, it was decided to follow a Clean Architecture [10] with an approximation of VIPER [11] implementation.

6.1.1 Clean Architecture

This approach is defined by Robert Cecil Martin alias Uncle Bob [12], and it is based in the separation of concerns principle. It defines several layers with a single responsibility that communicates to each other following the dependency rule seen in 2, which defines the relationship between the different layers.

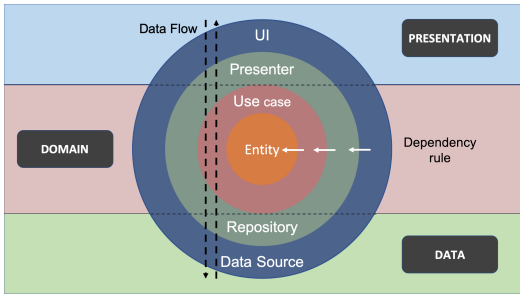


Fig. 2: Clean architecture layers.

- **Presentation:**
  - **UI:** This layer is the responsible of defining the UI classes and is tightly coupled to the framework.
  - **Presenters:** The Presenters layer defines all the view interactions in order to abstract the UI's logic in a non-dependent framework class. It is im-

portant to note that this layer is not the responsible of the business logic, it only knows when to contact to the following layer to perform any action.

- **Domain:**
  - **Use Cases:** This layer is the entry point to the app domain layer. It defines every single action that can be performed on the application. The use case does not know who triggered it and how the results are going to be presented.
  - **Entity:** Entities encapsulate Enterprise wide business rules. If you do not have an enterprise, and are just writing a single application, then these entities are the business objects of the application.
- **Data:**
  - **Repository:** Repositories are responsible to coordinate data from the different Data Sources.
  - **Data Source:** Responsible to access and retrieve data from Application Programming Interface (API), Data Base (DB), files, sensors, etc.

6.1.2 "VIPER" approximation

Once the concept of Clean Architecture is defined, as it was mentioned at the beginning of the section 6, an approximation based on VIPER, that follows the principles established by Clean Architecture has been used. This implementation is divided in the sections shown in figure and explained in the following list.

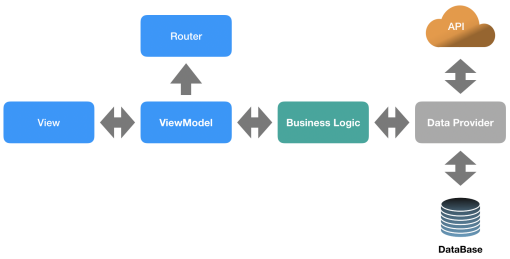


Fig. 3: VIPER-based architecture layers.

- **View:** The view layer where the UI elements will be placed.
- **ViewModel:** This layer will contain the view logic and will decide also when an action is required to navigate and as a cause when the router is needed to be used.
- **Router:** This will be the class responsible of performing the logic needed to the navigation process and will use the application's navigator to perform the transitions.
- **Business Logic:** In the business logic layer, the core logic of the application will be placed, and will act similarly to a use case.
- **Data Provider:** The data provider will be used and defined by the Business Logic, and will be responsible of retrieving the data from API services or DB, for example.

As it can be seen, this approach has the same separation of concerns that Clean Architecture defines, because all the dependencies are directed to the business logic and the same layers are implemented, divided in three different modules which are represented by the colors on the figure 3. The blue modules are referred to the presentation, the green module to the domain module and the gray one to the data module.

In the next section, the implementation of this architecture in the Android application modules will be represented.

6.2 Android modules

In this section the android modules, which are the application module and the acquisition module will be explained.

Before, it is important to explain the basics of an Android project. On the one hand a project has the configuration files. This files will set up all the project dependencies [13] in the gradle file [14]. The project will have a configuration file for every module, where the module dependencies and the compilation settings will be defined. On the other hand, the project contains all implementation classes, that for this project has been implemented in KOTLIN as this language has been chosen by Google as the official language since last year. Here there will be two types of files, the ones that depends on the Android framework and the implementation files that are independent of this framework.

The main Android framework-based files are summarized in the list below:

- **Application class:**

  This will act as the application entry point. Here all the global app configurations will be performed and this class will persist on memory always that the application is active.

- **Activities:**

  An Activity [15] is a class that contains the responsibility of the Android life-cycle [16]. This will act as the screen container and a common usage of this classes is to show inside this container the different views. This activity class is a full-screen view and in the project it has been implemented two different activities. One for the main content and another for the settings, that will explained later on this document in the chapter 8.2.

- **Fragments:**

  A Fragment [17] is a class that needs to be shown inside an activity container. This class has also the Android life-cycle but it depends directly on life-cycle of the activity. In the project there are three different fragments, which represent the map, the skyplot and the statistics screens that will be explained later on this document in the chapter 8.2.

6.2.1 app module

The app module is the main module in the Android project. This will be divided into four main directories which are listed and explained below.

- **app:**

  This will contain the Application class, all the base classes needed to accomplish the open-closed principle of SOLID [18], which are in example the base fragment or activity classes. This module will also contain the dependency injection [19] modules, which will help to reduce the application computation charge. Finally, in this module the general navigation classes, which will perform the transitions between the different screens will be found.

- **Presentation:**

  This module, as described in section 6.1.1, will contain all the presentation classes, which are activities, fragments and ViewModel classes. The design pattern that has been applied in this project is the Model-View-ViewModel with Live Data. For further information the user is derived to [20].

- **Domain:**

  The domain module is the responsible for the application main logic, and as the main requirement is to make this module independent of all other classes, because it has the most important logic of the application.

- **Data:**

  The data module will contain all repository-related data, that in this project is the Firebase petitions to save the measurements data. This feature will be explained in section 6.3.

6.2.2 acquisition module

The acquisition module will implement the classes related to the acquisition phase of the PVT algorithm, which includes both the input and the acquisition modules that will be defined later in section 7.

Here an Android Service [21] will be implemented in order to start listening to GNSS events in background in order to let the application continue working without interruptions as it is executed in a different thread. This will also let the user to keep tracking location data even if the application is in background.

This measurements will be processed through the acquisition modules and will serve as an entry point of the PVT module.

6.3 Firebase Implementation

In this section the Firebase implementation will be explained. This platform has been used in order to let the user store the measurements data in the cloud for the post-processing tasks. Every time that the user starts the PVT computation process, a local file on the phone will be generated, where the acquired measurements by the phone will be written. When the user choose the option to stop computing, this file will be sent to the cloud using the Cloud Storage kit. For more information about how does the Cloud Storage works, the reader is derived to [4].

Once Firebase is integrated in the application, allow the usage of the Crashlytichs [8] plugin, which tracks the application information about the produced crash and the users behaviour among many other features. This platform usage can lead to further implementations that will be noted at the end of this document.

In figure 4, a the administration panel of Firebase can be seen including some of the computation results. Here, the stored files can be downloaded and used for post-processing.

7 PVT ALGORITHM DEVELOPMENT

In this section the PVT algorithm development process will be shortly explained as it is not one of the main goals of this thesis. For more information about the PVT algorithm see [9].

The algorithm has been divided into different modules, in order to help to accomplish the single responsibility software pattern defined in SOLID principles. This tells that each piece of code should have a single and unique responsibility so, the solution was to divide the different actions that the old algorithm did into different modules. The other SOLID principles has been also applied to the refactor for instance by applying the dependency inversion principle, which recommends to inject all dependencies that the module needs instead of instantiating them inside. This will offer the possibility to make every piece of code testable. Otherwise, this modularization helps to reduce the code duplication, fulfilling the Don't Repeat Yourself (DRY) software principle.

Now the responsibility of the different modules, which are represented in figure 5, will be explained in the list below.

- Input:

  This module will be the responsible of defining the input data that the developed PVT algorithm needs to be able to compute a position. This data is provided by the Google's location library, which triggers actions for each GNSS measurement received on the phone. This library also provides the GNSS status data, used to obtain the google's provided satellites azimuth and elevation. Finally, the input module defines the selected conditions by the user which can be seen in 8.2. This data structures will be explained in more detail in the acquisition module.

- Acquisition:

  The acquisition module is the one that prepares all necessary data that the PVT algorithm requires. The input will be the google's provided data through the input module and the output will be the data structure for the algorithm.

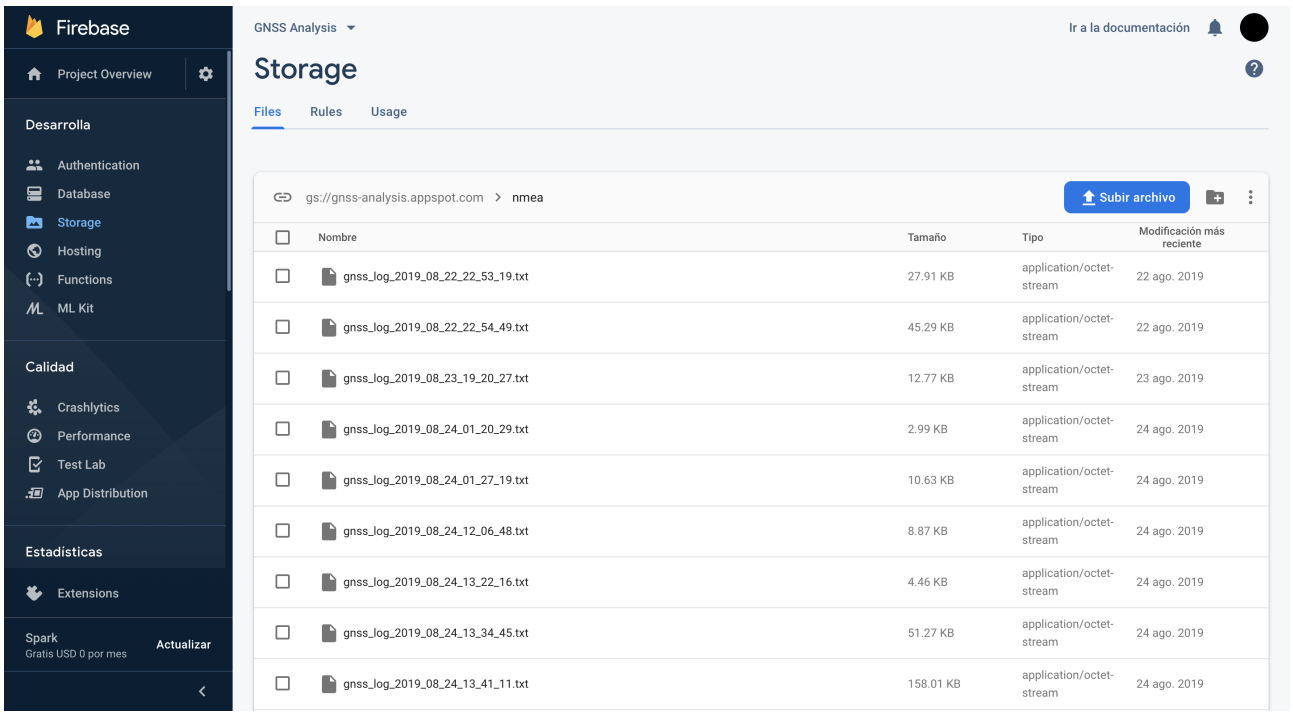


Fig. 4: Firebase control panel.

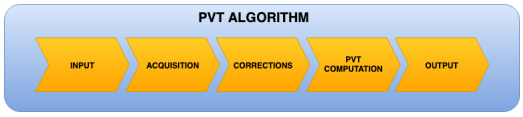


Fig. 5: PVT Algorithm modules.

Using the input data, here a data structure called *Epoch* will be prepared to serve as the output of this module. This epoch, contains at top level three important fields, which are referred to the time-of-week (TOW), now and transmission time. Then, it also contains a list of all satellite measurements. This module will iterate over each measurement received in order to perform the acquisition of all input parameters that are described in the list below.

1. **Svid:** The id of the satellite which has provided the measurement.
2. **Constellation:** The constellation where this satellite belongs.
3. **State:** A flag that indicates whether the receptor's acquisition was performed in optimal conditions or not.
4. **Multipath:** A flag indication if the measure can have multipath error.
5. **Carrier frequency:** The frequency of transmission that can be $L1/E1 = 1575,42MHz$ or $L5/E5a = 1176,45MHz$ for Global Positioning System (GPS) and Galileo respectively.
6. **Transmission time:** The time when the satellite trasmitted the correspondent navigation message
7. **Reception time:** The time when the phone received the signal
8. **C/N0:** The C/N0 of the satellite's signal received.
9. **Pseudorange:** The distance to the satellite measured as $D = c\Delta T$
10. **Azimuth:** The azimuth degrees of the satellite provided by the status library.
11. **Elevation:** The elevation degrees of the satellite provided by the status library.
12. **Ephemeris:** The satellite exact ephemeris provided by the supl library from Google.

- Corrections:

  This module will apply some range corrections due to troposphere and ionosphere propagation errors. For more information the user is derived to [22].

- Computation:

  This computation module is the one that implements the PVT algorithm itself. It defines the input data needed to compute a position and executes the algorithm given the specified conditions by the user in the settings screen 8.2. In order to execute the algorithm, it first needs to prepare the data, and here is where the dual-frequency and last corrections will be applied in order to obtain a data structure to iterate using the Least Squares (LS) algorithm. For more information about the least squares algorithm the reader is derived to [23]

- Output:

  Finally, once the algorithm has been executed, the results will be represented in the data structure that is represented in the list below. This data will be used to

fill up the statistics graphs in order to offer the user information about the integrity and robustness of the obtained results and to print the obtained positions in a map.

1. **PVT fix:** A structure that contains the latitude, longitude, altitude and clock bias data about the computed position.
2. **Reference location:** The reference location provided by Google in order to offer the user information about the error in meters of the result of the algorithm.
3. **Computation settings:** This will let to know the computation settings used to compute the position. Hence, the results can be identified by constellation in a map.
4. **Corrections:** Information about the corrections applied to the computation. This can be shown in the graphs in order to give information about the reliability on the results.
5. **Satellite number:** The used satellites number to compute the position. Here, information about how the position error decreases when the satellites number used increases.

8 Results

GNSS Tool is the Android app implemented as a result of this thesis. This application was developed focusing on the goal of offering the possibility to study the GNSS measurements using an Android receiver. Therefore, in order to satisfy the requirements, the application contains three main sections that will be explained below.

8.1 Application Modules

The application is developed in three different decoupled modules, reflected in 6 and explained in the list below.

- **ANDROID APPLICATION:** The main module is the android one, which contains all the implementations of the Android Application part. This module contains different sub modules that has been explained in section 6.2 and is in charge of the different app screens and a sub-module containing the PVT background Service that retrieves all GNSS data and performs the communication with the PVT Algorithm and Firebase.
- **FIREBASE:** This module performs the connection with Firebase Cloud Storage [4] in order to allow post-processing tasks. Here, the obtained GNSS measurements by the PVT Service defined in the Android Application module are saved in NMEA format to the Cloud. This module is scalable, so the chances to upload any kind of data are open in the future. The Firebase implementation also allows to keep the tracking of the produced bugs.
- **PVT ALGORITHM:** This module receives as an input the processed PVT Service data and the computation settings selected by the user. Next, it executes the algorithm to return the result.

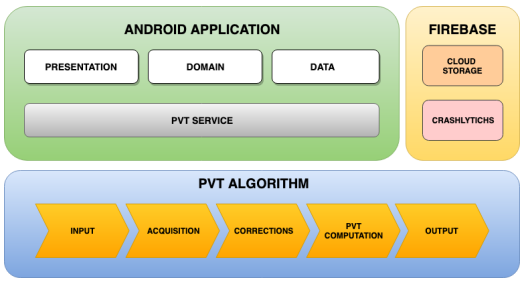


Fig. 6: Application modules.

8.2 Application Screens

The application is presented in four different screens. The first screen contains a map where the computed positions will be represented and the access to the settings screen. The next screen contains the Skyplot, and a representation of the selected elevation mask. Finally, the app contains a graphs screen, where both measurements and computation results will be displayed in different plots.

8.2.1 Map Screen

The first screen that will be seen when the application is opened is the screen shown in figure 7a. This section contains a map implementation where the user is able to start computing positions with the PVT Algorithm and where the obtained results will be displayed. Another important feature of the Map Screen is that it has the access to the Settings Screen, an important section that will be explained later.

This screen also has an option to change the map terrain, in order to allow the user to choose between a political one or the satellite type, which is more realistic, and, in addition, it has the option to see the legend of the selected computation settings. This can help the user to identify which color belongs to each setting, and also to remember the selected ones.

8.2.2 Settings screen

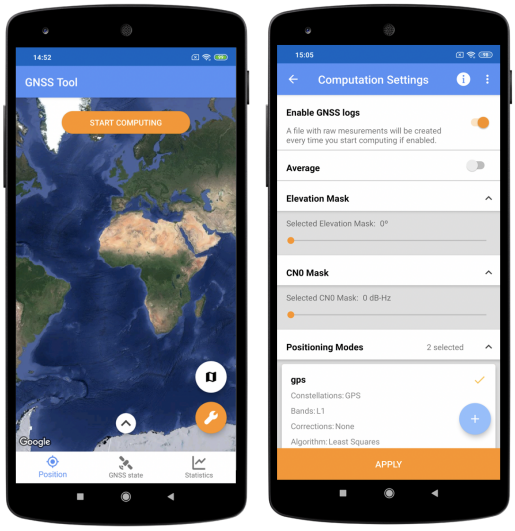
In the computation settings menu, there is a total of 5 options which can be configured according to user necessities. These options are listed and shortly explained below. Additionally, there is a menu which contains predefined positioning modes which also stores the profiles the user creates so that they can be accessed at anytime.

- **Enable GNSS logs:**
  If this option is enabled, a file containing raw measurements will be created every time the user starts a new computation. This will also allow send the generated file to the cloud once the user stops the computation.
- **Average:**
  This option allows the user to set the time in which measures will be averaged so as to reduce their variance.
- **Elevation Mask:**
  The elevation mask option establishes a Satellite elevation threshold from which satellites will be discarded from the positioning algorithm. Elevation refers to the

angle between the Satellite-to-Receiver direct path and the local horizontal plane.

- **C/N0 Mask:**
  The C/N0 mask establishes the satellite rejection threshold based in the Carrier to Noise Ratio.

- **Positioning Modes:** This option offers the user a list of computation settings that are created by the user in the new settings menu or, in his absence some default combinations.



(a) Map screen (b) Settings screen

Fig. 7: Map and Settings screens.

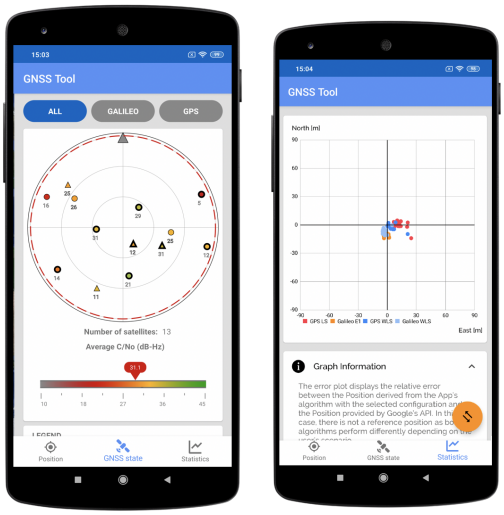
8.2.3 Skyplot

The Skyplot section (figure 8a) is aimed to show the user information about the satellites which are currently in view and may participate in the position solution. The graphic in the center is called Skyplot, because it displays Satellites from different constellations around the user's location in real time, as if seen from outside the Earth. Besides, it also contains the Elevation which was configured in Positioning Settings menu. Below the sky-plot, there is a bar which indicates the averaged Carrier to Noise ratio levels. Finally, in the bottom there is a legend which is aimed to help understand or remind the user of what is being displayed in the Skyplot.

9 Conclusions

In the last years the integrity and robustness issues on the GNSS field has been increased specially due to the quantity of applications implementing and using it in city environments, where the errors of multi-path and jamming are more present.

This thesis has offered a solution for those applications implemented in Smartphone devices, giving the user proper tools to analyse the integrity and robustness of the results that are being obtained after the execution of the algorithm,



(a) Skyplot screen (b) Statistics screen

Fig. 8: Skyplot and Statistics screens.

using the device as a GNSS receiver. It can help the user to understand the best computation setting for each environment and to understand the possible reliability problems of each computation, showing information about the results and the environment conditions in different graphs.

First, the used technologies and the methodology that has been followed during the thesis development has been explained in sections 4 and 5. Then, on the one hand, the Android application implementation has been explained in section 6. This section explains the Android programming best practices to implement the solution, and the decision of the modules that has been chosen to show all information and the tools to select the conditions the user wants the algorithm to be executed. Here, the cloud solution for the results and measurements storage has been also noted. On the other hand, in section 7 an overview of the PVT algorithm implemented modules has been explained.

The results has been finally presented in section **??**, starting by the presentation of the application features divided in the four main screens, which are the map to show the location result of the algorithm, the sky-plot to give information about the seen satellites conditions, the graphs screen to offer the combination of the algorithm results, the device's receptor information and the provided information about the satellites signals conditions offered by a Google API to the let the user understand the conditions and finally, the settings screen to allow the user to select the desired parameters combination to execute the algorithm.

At this point, it can be concluded that the objectives defined at the beginning of the project has been accomplished having developed an application that offer the user the possibility to see his real-time position using a PVT algorithm that can be configured with single and multi-constellation and other parameters such as dual frequency corrections, and then the feedback on the reliability of the results is provided in the graphs in order to let the user know about the integrity of the obtained position. Also the cloud implementation has been integrated to the application in order to allow the mentioned post-processing tasks, as the one performed on the static test in MATLAB.

To sum up, GNSS Tool is an Android application that can be used to obtain information about the best positioning configuration that can be selected on each environment and can be scalable as the algorithm is completely modularized. Some of this scalability ideas are presented in the next section.

10 FUTURE WORK

The realization of this project has opened many possible application features and ideas that have not been implemented due to their were out of the planning and the project timing has not let to dedicate the time to the extra features that could be investigated and are presented next.

First, the cloud implementation has opened a world to the data sharing possibilities such as user session to keep the computed data or the pvt execution on the cloud so that it can be used in different receptors. Therefore, it would be interesting to develop this features in Firebase, which has been the platform used.

In terms of the Android application features, many ideas came out during the design process that had to be discarded in order to have a realistic project planing. Some of them are presented in the list below.

- **Show used satellites:**

  Use the sky-plot to indicate the satellites that are being used for the computation.

- **Select satellites:**

  Give the user the possibility to select in the sky-plot which in-view satellites wants to use for the computation. This would help to discard satellites that are not giving accurate measurements.

- **Automatic configurations:**

  Automate the process of computation settings selection using environment data to decide which is better for each scenario. This will fit in many applications that use the GNSS measurements to obtain position, for instance selecting an elevation mask restriction in a city environment and changing it when the scenario has better conditions.

- **Reference Position:**

  Allow the user to set a reference position in the error plot in order to obtain a more realistic positioning error.

These and maybe other questions that may arise to the reader are left for future work on the matter.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Prof. Daniel Egea Roca for his patient guidance and useful recommendations on the development of this project. Also, I would like to thank him for offering as the mentor of the Galileo App Competition from which this project has been based. Finally, I am also very grateful to my family and friends for their personal support during the realisation of this project.

REFERENCES

[1] J. Sanz Subirana, J.M. Juan Zornoza, and M. Hernández-Pajares. Multipath. Navipedia, 2011.

[2] C. Tsang. Jamming detection and snr/snjr estimation. In *2011 Aerospace Conference*, pages 1–7, March 2011.

[3] Mark L. Psiaki and Todd E. Humphreys. GNSS Spoofing and Detection. *Proceedings of the IEEE*, 104(6):1258–1270, June 2016.

[4] Cloud Storage. Firebase, 2019.

[5] Rui Barradas Pereira. Gnss applications. Navipedia, 2011.

[6] Ni ZHU, Juliette MARAIS, David Betaille, and Marion Berbineau. GNSS Position Integrity in Urban Environments: A Review of Literature. *IEEE Transactions on Intelligent Transportation Systems*, page 17p, January 2018.

[7] Sean Barbeau. Dual-frequency gnss on android devices. Medium, 2018.

[8] Crashlytics. Firebase, 2019.

[9] Albert Becerra. Gnss integrity & robustness tool. UAB, 2020.

[10] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall Press, USA, 1st edition, 2017.

[11] Ray Wenderlich. Android VIPER Tutorial, may 2018.

[12] Robert C. Martin, January 2020. Page Version ID: 935471072.

[13] Add build dependencies. Android Developers, 2019.

[14] Configure your build. Android Developers, 2019.

[15] Activity. Android Developers, 2019.

[16] Understand the Activity Lifecycle. Android Developers, 2019.

[17] Fragments. Android Developers, 2019.

[18] MindOrks. SOLID Principles - explained with examples. Medium, 2018.

[19] Dependency injection in Android. Android Developers, 2019.

[20] Hazem Saleh. MVVM architecture, ViewModel and LiveData (Part 1), October 2018.

[21] Services overview. Android Developers, 2019.

[22] P. Misra and P. Enge. *Global Positioning System: Signals, Measurements, and Performance*. Ganga-Jamuna Press, 2011.

[23] Steven J. Miller. The method of least squares. Brown University, 2012.