

---

# **GNSS Compare Documentation**

**Sebastian Ciuban**

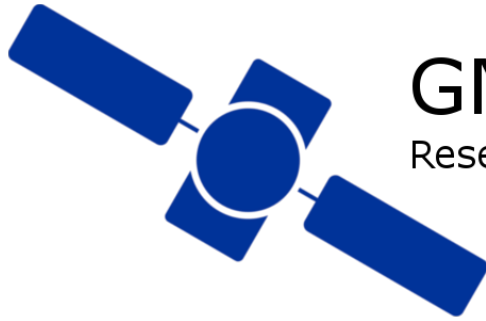
**Mateusz Krainiski**

**Mar 10, 2019**



<b>1</b>	<b>Description</b>	<b>3</b>
1.1	Project history . . . . .	3
<b>2</b>	<b>Useful Links</b>	<b>5</b>
2.1	Example of proposal . . . . .	5
2.2	Open source code on Github . . . . .	5
<b>3</b>	<b>Download</b>	<b>7</b>
<b>4</b>	<b>Hall of Fame</b>	<b>9</b>
4.1	Software Design . . . . .	10
4.2	GNSS Signal Processing . . . . .	11
4.3	Contact . . . . .	12
<b>5</b>	<b>Glossary</b>	<b>13</b>
5.1	Global Navigation Satellite Systems Glossary . . . . .	13
5.2	Android Glossary . . . . .	16
5.3	Software Engineering Glossary . . . . .	16
<b>6</b>	<b>Acknowledgements</b>	<b>17</b>
<b>7</b>	<b>Newsletter</b>	<b>19</b>
<b>8</b>	<b>Introduction</b>	<b>21</b>
<b>9</b>	<b>Getting started with the User Interface</b>	<b>23</b>
9.1	Application's Views . . . . .	23
9.2	Processing schemes . . . . .	24
<b>10</b>	<b>Getting started with the code</b>	<b>31</b>
10.1	Importing the project to Android Studio . . . . .	31
10.2	Using the Google Maps Viewer . . . . .	32
<b>11</b>	<b>GNSS basics</b>	<b>33</b>
11.1	Constellation . . . . .	33
11.2	Corrections . . . . .	33
11.3	PVT Estimator . . . . .	34

<b>12</b>	<b>Android GNSS raw measurements</b>	<b>35</b>
12.1	Galileo . . . . .	35
12.2	GPS . . . . .	36
<b>13</b>	<b>Implemented PVT Algorithms</b>	<b>39</b>
13.1	Extended Kalman Filter . . . . .	39
13.2	Weighted Least Squares . . . . .	43
<b>14</b>	<b>Example of analysis</b>	<b>45</b>
14.1	Static user . . . . .	45
14.2	Pedestrian user . . . . .	47
14.3	Dynamic user . . . . .	48



# GNSS Compare

Research lab in your pocket

**GNSS Compare now supports the Dual-Frequency Xiaomi Mi 8 smartphone!**

Welcome to GNSS Compare's documentation!

Please stay tuned, this is still a beta version, and we're making updates on a daily basis...



Check out the app on [Play Store](#).

Please note that you need Android 7.0+ to run the application. Also, please note that not all Android 7.0+ phones support the Galileo Satellite System or crucial for this project raw GNSS measurements. The list of compatible phones can be found [here](#).

So far we've been testing the application on:

- Samsung Galaxy S8
- Samsung Galaxy S8+
- Samsung Galaxy Note 8
- Xiaomi Mi 8

And we are very happy with the results!

If you want to be kept up to date with our updates, you can sign up to our [newsletter](#)!



# CHAPTER 1

---

## Description

---

The purpose of GNSS Compare is to make the life of developers and researchers easier. It's an easy to use and easy to extend Android-based framework for calculating the *PVT* based on the raw GNSS measurements.

### 1.1 Project history

GNSS Compare is the winning Android application developed by the Galfins as part of the internal [European Space Agency \(ESA\)](#) competition called [Galileo Smartphone App Challenge](#) introduced in November 2017. The Challenge was about creating a smartphone application, which will allow the user to choose which satellite constellation to use for *Position, Velocity and Time* estimation. The aim was to increase the awareness about ESA's [Galileo programme](#) and also to allow users from common public to compare the performance of Galileo signals with the performance from other global satellite navigation constellations.





In this section we have put some links that can be useful for the competitors of the [2nd Edition of the Galileo Android App Competition](#) (ran by the European Space Agency (ESA), European GNSS Agency (GSA), European Commission (EC) and supported also by Google):

### 2.1 Example of proposal

Feel free to use our original proposal, which we have submitted to the [first edition of the Galileo Android App Competition](#), as a source of inspiration. It's available to download [here](#). Just remember - our idea has changed in the course of the project, so not everything in that document is up to date!

### 2.2 Open source code on Github

Click [here](#).



## CHAPTER 3

---

Download

---



*GNSS Compare* is always available on the [Google Play Store](#):

Please note that you need Android 7.0+ to run the application. Also, please note that not all Android 7.0+ phones support the Galileo Satellite System. The list of Galileo compatible phones can be found [here](#).

So far we've been testing the application on:

- Samsung Galaxy S8
- Samsung Galaxy S8+
- Samsung Galaxy Note 8
- Xiaomi Mi 8

And we are very happy with the results!



## CHAPTER 4

---

### Hall of Fame

---

In this section we present you the amazing people that were fueled by enormous amounts of tea\*<sup>0</sup> in order to develop *GNSS Compare*!

And here they are! From the left to right we have: Mareike Burba, Sebastian Ciuban, Dominika Perz and Mateusz Krainski.



They all have an impressive set of skills that brought the 1st prize for *GNSS Compare* at the European Space Agency *Galileo Smartphone App Challenge*. Their knowledge and experience related to **Software Design** and **GNSS Signal Processing** are presented in short biographies so you can have an idea who is behind *GNSS Compare*!

---

<sup>0</sup> Typically it's coffee, but we like to approach things differently.

## 4.1 Software Design

### 4.1.1 Mateusz Krainski



Polish Young Graduate Trainee in the Directorate of Human Spaceflight and Robotic Exploration, at ESTEC, where he supports the European Robotic Arm (ERA) team. His main duties regard designing, developing and validating a robotic testbed for testing of ERA's on-board smart cameras. During his studies, Mateusz was one of the key board members of a robotic student society, where he managed numerous projects ranging from small teams for quick projects (this includes a Space Startup Weekend, an Android app hackathon and few during studies assignments), organizing robotic tournaments (with a team of over 15 people), up to technical projects counting over 30 people. Thanks to the Toastmasters International community, Mateusz has developed highly his public speaking skills. He not only helped start the first English speaking club in the area, but also received awards in presenting competitions on a semi-national level.

### 4.1.2 Dominika Perz



Polish Young Graduate Trainee in the Technology, Engineering and Quality Directorate at ESTEC, ESA. Currently working as a Project Manager for the Lunar Exploration Mission - internal project to determine a preliminary GNC (Guidance, Navigation and Control) design for the ascent, rendezvous and docking with the Deep Space Gateway station orbiting a Moon in the highly elliptical orbit. Her background is mainly in robotics and programming. She completed Control Engineering and Robotics master studies in Poland, during which she spent one semester in Madrid, Spain as an Erasmus exchange student. As a first international career experience, Dominika did a 6 weeks internship in R&D team at Vonderlande (Eindhoven, Netherlands), where she worked on optimisation of the search algorithm. During holidays in 2016 she participated in the Aerospace Information Technology Summer School in Würzburg, Germany. Before coming to ESA, Dominika worked for a year at a software company GlobalLogic as a Junior Software Developer for embedded systems.

## 4.2 GNSS Signal Processing

### 4.2.1 Sebastian Ciuban



Romanian Young Graduate Trainee in the Directorate of Navigation at the European Space Research and Technology Centre (ESTEC) working on the Galileo project. He holds a Master of Science degree in Aerospace Systems – Navigation and Telecommunications granted by the French Civil Aviation University (ENAC) from Toulouse, France. After his first year of master studies he worked a summer (2016) at Acorde Technologies in Spain in the area of GPS/INS integration. During the second year of studies he developed his master thesis at the German Aerospace Center (DLR) in Oberpfaffenhofen. While being at DLR he was responsible with designing and implementing algorithms that fused the Precise Point Positioning (PPP) technique with Inertial Navigation Systems (INS) in a tight coupling architecture. Moreover, he has also developed suitable integrity monitoring algorithms in order to measure the reliability of the designed fused systems in terms of fault detection sensitivity and protection level stability. His research interests are related to precise positioning, sensor fusion, integrity monitoring and GNSS receiver signal processing.

### 4.2.2 Mareike Burba



German National Trainee with EOP-SMS, currently working on the atmospheric correction for the Fluorescence Explorer (FLEX). She holds a M.Sc. in Meteorology with a minor in Scientific Computing from the University of Hamburg. Her Master's thesis was about optimizing the numerical computation of atmospheric radiative transfer for the Infrared Atmospheric Sounding Interferometer (IASI) on MetOp. Mareike previously joined the World Climate Research Program Joint Planning Staff in the World Meteorological Organization for a 6 months internship in the framework of the Carlo Schmid program. Thanks to her studies and jobs as student research assistant, she speaks fluently Python, Matlab, Bash and Fortran, is communicative in C, Java and IDL, and has some experience with parallel computing.

## 4.3 Contact

If you would like to reach any of the *GNSS Compare* developers, here are their e-mails:

- Mateusz Krainski: [mateusz@krainski.eu](mailto:mateusz@krainski.eu)
- Dominika Perz: [dominika.perz@gmail.com](mailto:dominika.perz@gmail.com)
- Sebastian Ciuban: [ciuban.sebastian@gmail.com](mailto:ciuban.sebastian@gmail.com)
- Mareike Burba: [mareikeburba@gmail.com](mailto:mareikeburba@gmail.com)



The aim of this section is to provide a small glossary on the subject of satellite navigation, Android and software engineering in general. You can find below descriptions of all used technical terms in this documentation.

## 5.1 Global Navigation Satellite Systems Glossary

### 5.1.1 Global Navigation Satellite Systems

A search on the internet about GNSS will give plenty of information about everything one needs to know. In this section we will briefly present the current status of the existing global satellite navigation systems.

#### **Galileo**

Galileo is the European GNSS with the following status and characteristics:

- In orbit satellites: 22 ( *the launch of another 4 satellites being scheduled for 25th of July 2018*)
- Satellites orbit: Medium Earth Orbit (MEO) with an altitude of approximately 23000 kilometers
- Orbit inclination: 56 degrees to the equator
- Orbital planes: 3
- Time System : Galileo System Time (GST)
- Used spread-spectrum technique: Code Division Multiple Access (CDMA)

#### **GPS**

GPS is the American GNSS with the following status and characteristics:

- In orbit satellites: 31
- Satellites orbit: MEO with an altitude of approximately 20000 kilometers
- Orbit inclination: 55 degrees to the equator
- Orbital planes: 6

- Time System : GPS Time (GPST)
- Used spread-spectrum technique: CDMA

### GLONASS

GLONASS is the Russian GNSS with the following status and characteristics:

- In orbit satellites: 25
- Satellites orbit: MEO with an altitude of approximately 19000 kilometers
- Orbit inclination: 65 degrees to the equator
- Orbital planes: 3
- Time System : GLONASS Time (GLONASST)
- Used spread-spectrum technique: Frequency Division Multiple Access (FDMA) and CDMA (recently)

### Beidou

Beidou is the Chinese GNSS with the following status and characteristics:

- In orbit satellites: 28 ( 13 satellites are not included in operational orbital constellation)
- Satellites orbit: Geostationary Orbit (GEO) and MEO
- Orbit inclination: 55.5 degrees to the equator for MEO
- Orbital planes: 3 (24 MEO)
- Time System : Beidou Time (BDT)
- Used spread-spectrum technique: CDMA

## 5.1.2 Pseudorange

For someone making his or hers first steps in the GNSS field, the term *pseudorange* might sound a little bit confusing. Afterall, the word *pseudo* is synonym with *false* and considering this, one might ask: why use this type of information? At the end of this section we hope to answer this question and also to make things more clear regarding this subject.

First let's start thinking (in general terms) how the receiver determines the distances towards the observed satellites. The range (R) is the difference between the time of signal reception and the time of signal transmission multiplied by the speed of light (c):

$$R = c \cdot (t_R - t^S).$$

Although the clocks (atomic clocks) of the satellites are highly accurate, they are still not perfect which lead them to be biased with respect to a certain GNSS time frame. Furthermore, considering that the quality of the clocks used in the typical GNSS receiver is inferior to the ones of the satellites, there is also a (significantly larger) bias in its time measurements. Therefore, let's take this into account in our equation expressed above:

$$R = c \cdot [t_R + \delta t_R - (t^S + \delta t^S)].$$

If we arrange a bit the newly obtained expression, we get:

$$R = c \cdot (t_R - t^S) + c \cdot (\delta t_R - \delta t^S).$$

Assuming that the time of signal reception and the time of signal transmission are free of their biases and other error sources, then their difference multiplied by the speed of light can be viewed as the equivalent of the geometric distance (rho) in 3D between the receiver and the observed satellite!

$$R = \rho + c \cdot (\delta t_R - \delta t^S).$$

Now that we got this settled, we also need to account for the effects that disturb the signal's travel from the satellite to the receiver such as the ionosphere (I) and troposphere (T). The local effects like the receiver's noise and multipath which for the sake of simplicity we gather them in a single term (epsilon). The number of effects that introduce errors in the range measurements is larger and we don't cover them here.

$$R = \rho + c \cdot (\delta t_R - \delta t^S) + I + T + \epsilon.$$

In the equation of the range above we correct for the effect of the satellite clock bias, ionosphere, troposphere mainly by mathematical models. However, what we can't remove directly is the receiver clock bias which is required to be estimated. And that term will always be present in our measurements! Therefore, our *range* equation becomes the *pseudorange* (PR) equation because of that.

$$PR = \rho + c \cdot (\delta t_R - \delta t^S) + I + T + \epsilon.$$

We do hope that the aspects related to this subject are more clear now.

### 5.1.3 Ephemeris

The process of obtaining the position in a certain coordinate system using GNSS technologies is based on a rather simple principle, which is *trilateration* (not triangulation, please be aware of that). Given an unknown point in a coordinate system from which we know the distances towards some known points in the same coordinate system, we can work out the coordinates of our unknown point. One can try this concept by defining a 2D coordinate system in which a triangle can be drawn with two of its vertices having known coordinates. And the problem relies on finding the coordinates of the third vertex.

We have already seen in the *Pseudorange* section that we can obtain the range information towards the observed satellites. And what is missing is how to determine the coordinates of those satellites. To compute the coordinates of the satellite we need some parameters that describe their orbits. For this we have to be grateful to the work of Johannes Kepler on his law of planetary motion as he discovered the six parameters also known as the *Keplerian elements* that define an orbit:

- Eccentricity,
- Semimajor axis,
- Inclination,
- Longitude of the ascending node,
- Argument of periapsis,
- True anomaly.

The definition of all of the above elements can be easily found with a quick search on any internet search engine. The idea is that those parameters (and many others) are contained in the navigation message that is modulated on the transmitted GNSS signals. The receiver will extract this information from the signal itself or get them from external means (e.g., concept of Assisted GNSS) and feed them into specific algorithms that will determine the satellite coordinates.

A good source for the satellite coordinate computation algorithms for Galileo and GPS is the book called [ESA GNSS Data Processing Volume I](#) starting with page 57.

### 5.1.4 Position, Velocity and Time

In GNSS we are mostly interested in the parameters of the user that describe the position, velocity and also time. Position is quite obvious - that's the whole point of navigation, to know where the user is located. Velocity can be estimated from consecutive position measurements, but can also be calculated directly from the satellite signals, due

to the Doppler's effect. It can be later used for more precise estimations of the user's position, for highly dynamic systems. Time is also crucial to be estimated, as the user's receiver clock contains a bias with respect to a certain GNSS time frame.

For more clarifications on how this process is handled, the reader is welcomed to check the *Implemented PVT Algorithms* section.

### 5.1.5 Clock bias

In GNSS, when we talk about *clock bias* we usually refer to the satellite clock bias and/or to the receiver's clock bias with respect to a certain GNSS time frame (e.g., Galileo System Time, GPS Time).

Let's take a look on how we correct for the satellite clock bias as explained in [ESA GNSS Data Processing Volume I](#) (pages 104-105):

$$\delta t^S = \tilde{\delta t}^S + \Delta t_{\text{rel}}.$$

In the above equation we can see that the satellite clock bias is also affected by a small relativistic effect caused by the orbit eccentricity. Is quite interesting to see that when dealing with time we do need to take into account these kind of phenomena! The correction for that relativistic effect is computed in the following way:

$$\Delta t_{\text{rel}} = -2 \frac{\mathbf{r}^S \cdot \mathbf{v}^S}{c^2},$$

where at the numerator we have the dot product between the satellite position vector and the velocity vector and at the denominator the speed of light squared.

And finally the satellite clock bias (without the relativistic effect) can be computed as:

$$\tilde{\delta t}^S = a_0 + a_1(t - t_0) + a_2(t - t_0)^2,$$

where the coefficients (a0, a1 and a2) are contained in the navigation message and t0 represents a given reference epoch.

Because we cannot account for the receiver clock bias beforehand we have to estimate it from the pseudorange equation:

$$PR = \rho + c \cdot (\delta t_R - \delta t^S) + I + T + \epsilon.$$

This one is more straight forward.

## 5.2 Android Glossary

## 5.3 Software Engineering Glossary

### 5.3.1 Polymorphism

According to [Wikipedia](#), *Polymorphism is the provision of a single interface to entities of different types*. In Java this is achieved due to class inheritance.

---

### Acknowledgements

---

We would like to mention the libraries that we are using in the *GNSS Compare* software framework:

- [Efficient Java Matrix Library](#) (EJML) for linear algebra operations,
- [goGPS](#) for GNSS related computations (satellite coordinates and velocities, etc.),
- [GNSS Logger](#) for using the raw data logging format as a feature.

Also we would like to thank the European Space Agency for the continuous support throughout the development of this application. Especially we would like to mention the ESA employees that shared their experience and knowledge with us: Paolo Crosta, Nityaporn Sirikan, Gaetano Galluzzo and Paolo Zoccarato.



## CHAPTER 7

---

### Newsletter

---

**Dual-Frequency capabilities for GNSS Compare are currently under development and testing. Stay tuned!**

GNSS Compare is currently under heavy development. If you want to be up to date with our changes, sign up to the newsletter [here](#).





## CHAPTER 8

---

### Introduction

---

Let's start with the basics. GNSS compare allows the user to calculate the phone's location, based on the phone's GNSS pseudorange measurements. This is normally done by the phone automatically, however with the recent release of Android API 24 and the [GnssMeasurement](#) class, developers gained access to unprocessed pseudorange measurements. This can be used by GNSS scientists and researchers to come up with new, more precise or less resource intense methods for precise positioning. GNSS Compare is basically a tool for such scientists to compare their algorithms. And if you're not an GNSS expert, it can be a tool for you to learn more on this subject. And believe me – there's a lot of interesting things to learn.

In order to fully understand what's under the hood for GNSS Compare, we must introduce a few terms we are using through the application. It's going to be a very general explanation – if something will be not clear, please take a look at our [Glossary](#). We hope you'll find answers to more detailed questions there. This section aims to provide a bridge between general GNSS knowledge, software practices, and GNSS Compare.



---

## Getting started with the User Interface

---

Here we show what you can see at the User Interface (UI) level and we also describe how to set up different processing schemes/calculation modules, effect of which can be studied in real-time.

### 9.1 Application's Views

We refer to a *view* as the current content displayed by the application. The user can change these views by swiping on the phone's screen. Currently we have the following views: **Main View**, **Satellite signal strength**, **Positioning error plot** and **Google Maps view**.

#### 9.1.1 Main View

When you launch the application this is the first view.

On top there is a blue “stripe” with the name of the application, a “+” and a “gearbox” icon. What are the functionalities of those icons we will see in the *Processing schemes* part.

Next is the **Constellation status** header and the information below that shows what GNSS constellations and how many satellites are used to compute PVT. In the GIF above we can see that a combination of Galileo+GPS, GPS only and Galileo only are considered in the algorithms. Moreover, you can notice that not all *Visible* satellites are being *Used* in the calculations. The reason behind this is explained in a dedicated chapter of this documentation called *Android GNSS raw measurements*. Shortly, it is because not all obtained pseudoranges pass a criteria that would allow them to be used in the PVT estimation.

Below the header called **Calculation results** are the results of the PVT estimators (EKF for this particular example) in terms of: latitude ( *Lat* ), longitude ( *Lon* ), altitude ( *Alt* ) and the receiver's clock bias ( *C.bias* ). The UI allows it's user to make some interesting analysis and to gain some intuitions about the importance of the number of the used satellites in PVT. As example, because there are only 3 Galileo satellites used in the EKF we do expect the estimations of the unknowns to be degraded, which is the case.

And lastly, the *START RAW LOG* allows the logging of the Android GNSS raw measurements in the exact same format as the Google's Application [GNSS Logger](#). This feature allows you also to analyze your data in post-processing!

To get to the next view swipe from right to left.

### 9.1.2 Satellite signal strength

This view is quite straight forward. Here you can monitor the signal strength of the satellites that are *Used* in the calculations.

To get to the next view swipe from right to left or to return to the previous one, from left to right.

### 9.1.3 Positioning error plot

To have an idea of how well the position is estimated, we provide this view that contains a plot with the horizontal position errors using as reference the *Android FINE location* (i.e., the best location output by the phone). The errors are expressed in meters in the north and east direction (local frame).

Below the plot there is the legend with the specific colors for the chosen processing schemes.

To get to the next view swipe from right to left or to return to the previous one, from left to right.

### 9.1.4 Google Maps view

In the last view there is the Google Maps on which the position estimations are displayed to be monitored. This can be useful especially when you are testing new PVT algorithms or change the settings of the existing ones (e.g., tuning the EKF). In the GIF below are presented the position estimations by the EKF while the user was in a bus. In this way you can study if your algorithms and their tuning are able to output estimations that follow your dynamics in real-time.

Another useful study that can be made in this view is the comparison of different PVT algorithms. In the example below, one can gain insights about the difference between WLS and EKF. It is interesting to see the performance of an estimator that relies only on measurements relative to an estimator that uses a dynamic model in addition.

To get to the the previous view swipe from left to right.

## 9.2 Processing schemes

By a processing scheme or a calculation module we refer to a set of settings that are considered for the estimation of the smartphone's position. The user can create a processing scheme in which he/she can choose the following:

- Constellation: *Galileo+GPS, GPS, Galileo*
- Correction modules: *Tropospheric correction, Klobuchar Iono Correction (only for GPS), Relativistic path range correction*
- Positioning method: *Weighted Least Squares, Static EKF, Dynamic EKF, Pedestrian EKF*
- Logging format: *Simple format, NMEA*

- Name: *The user shall specify the name of the processing scheme*

We will continue to show how to create a processing scheme or modify an existing one at the UI level.

### 9.2.1 Creating a new one

From the *Main View* select the “+” icon on the top right corner as shown below. A new view will prompt that will allow you to select among the available options.

The screenshot shows the GNSS Compare app interface. At the top, there's a blue header with the app name and a settings icon. Below the header, there's a section for constellation status and calculation results. To the right, there's a settings panel with options for constellation, correction modules, positioning method, logging format, and a name field. At the bottom, there's a 'START RAW LOG' button and a 'Create' button with a plus icon.

**GNSS Compare**  
Research lab in your pocket

**Constellation status**

	Visible	Used
Galileo + GPS	18	9
GPS	14	8
Galileo	4	1

**Calculation results**

	Lat	Lon	Alt	C. bias
Galileo+GPS	52.16956	4.48078	49.6	413536
GPS	52.16958	4.48078	48.3	413536
Galileo	52.16990	4.48107	20.7	413483

**Settings for the calculations**

**Constellation**  
This allows you to select what GNSS satellite constellations you'd like to use

**Correction modules**  
Select which corrections to apply

**Positioning method**  
Select which position, velocity and time estimation method you'd like to use

**Logging format**  
Select the format in which calculated position should be logged.

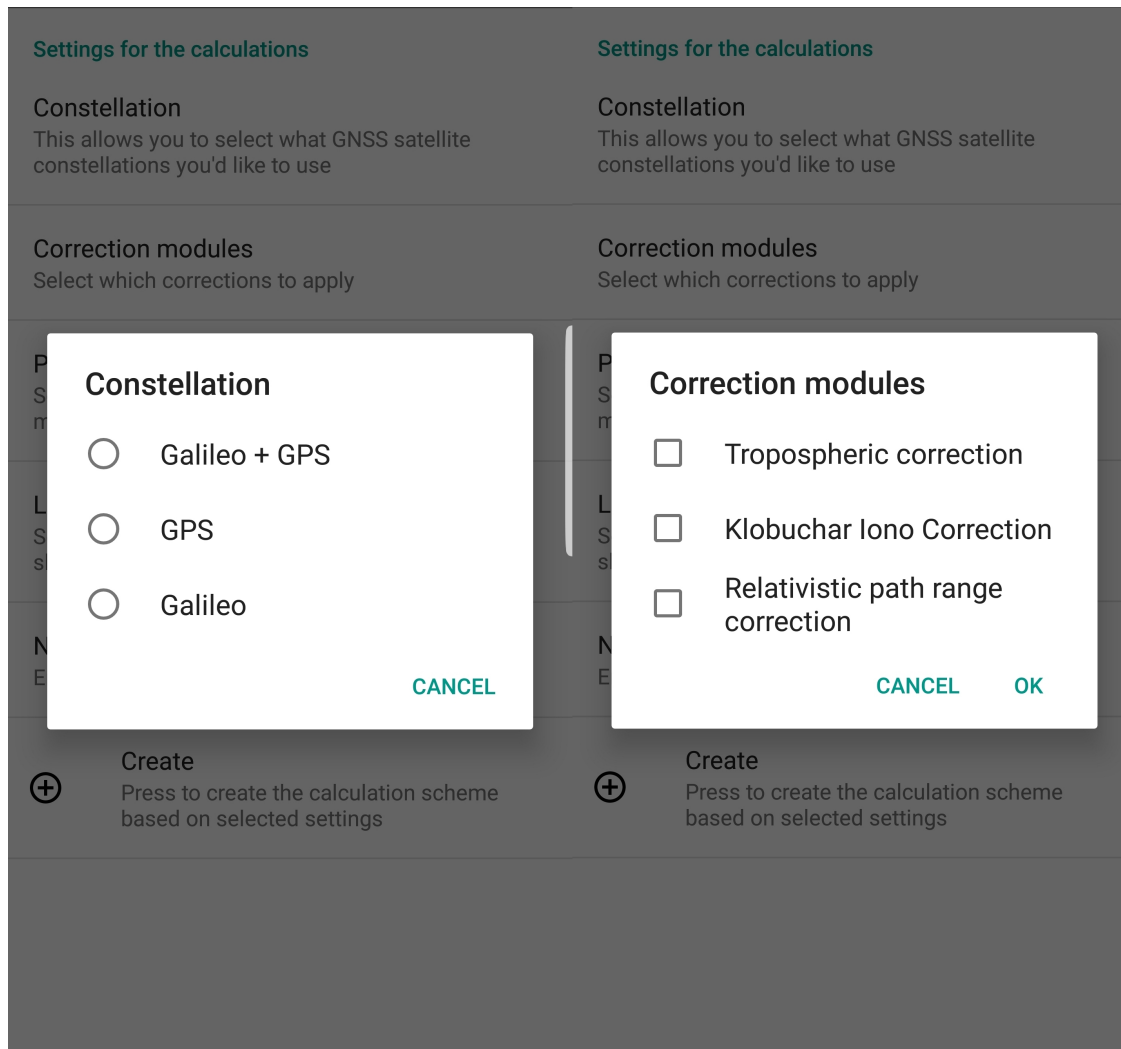
**Name**  
Enter name for this processing scheme

**Create**  
Press to create the calculation scheme based on selected settings

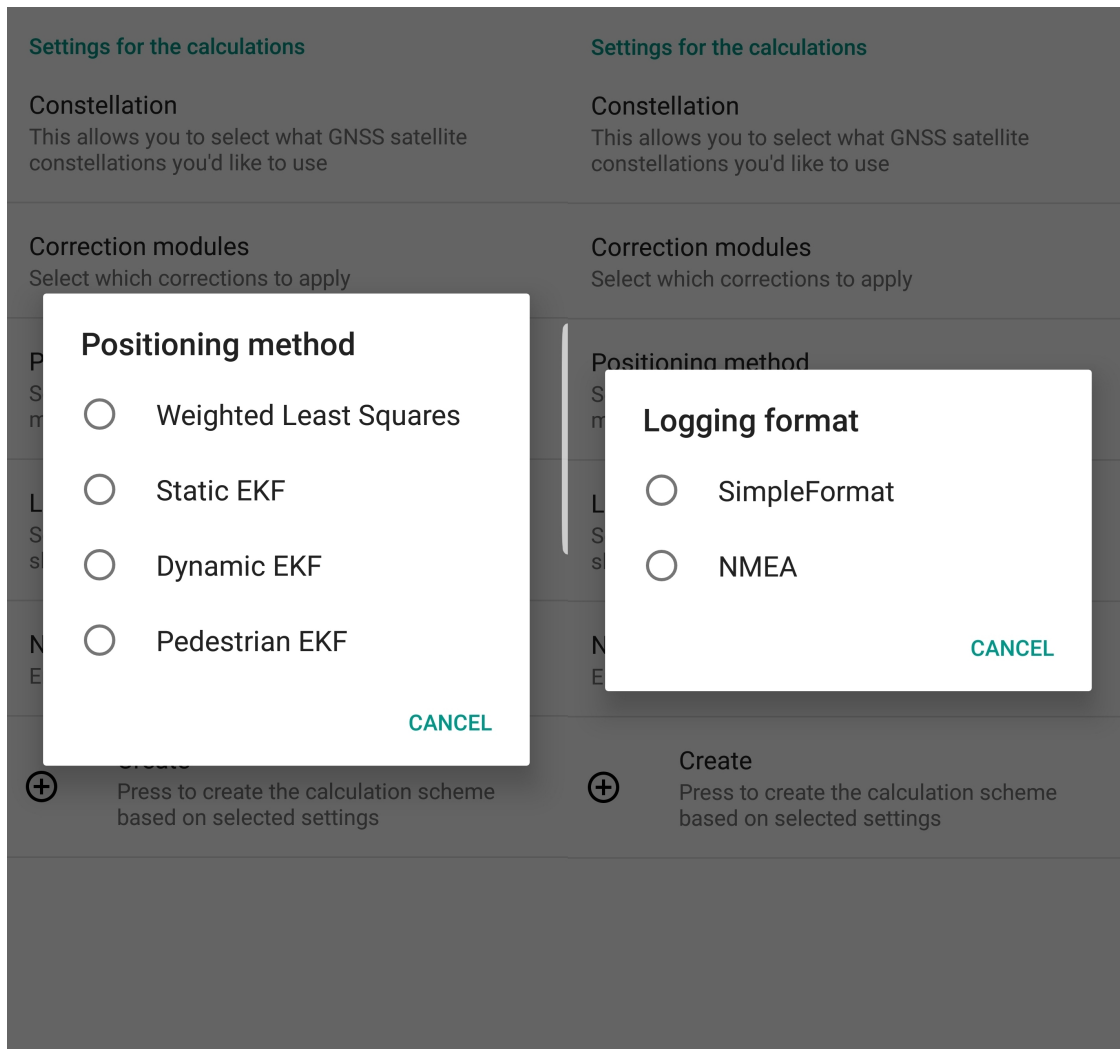
**START RAW LOG**

Developed with support of the European Space Agency

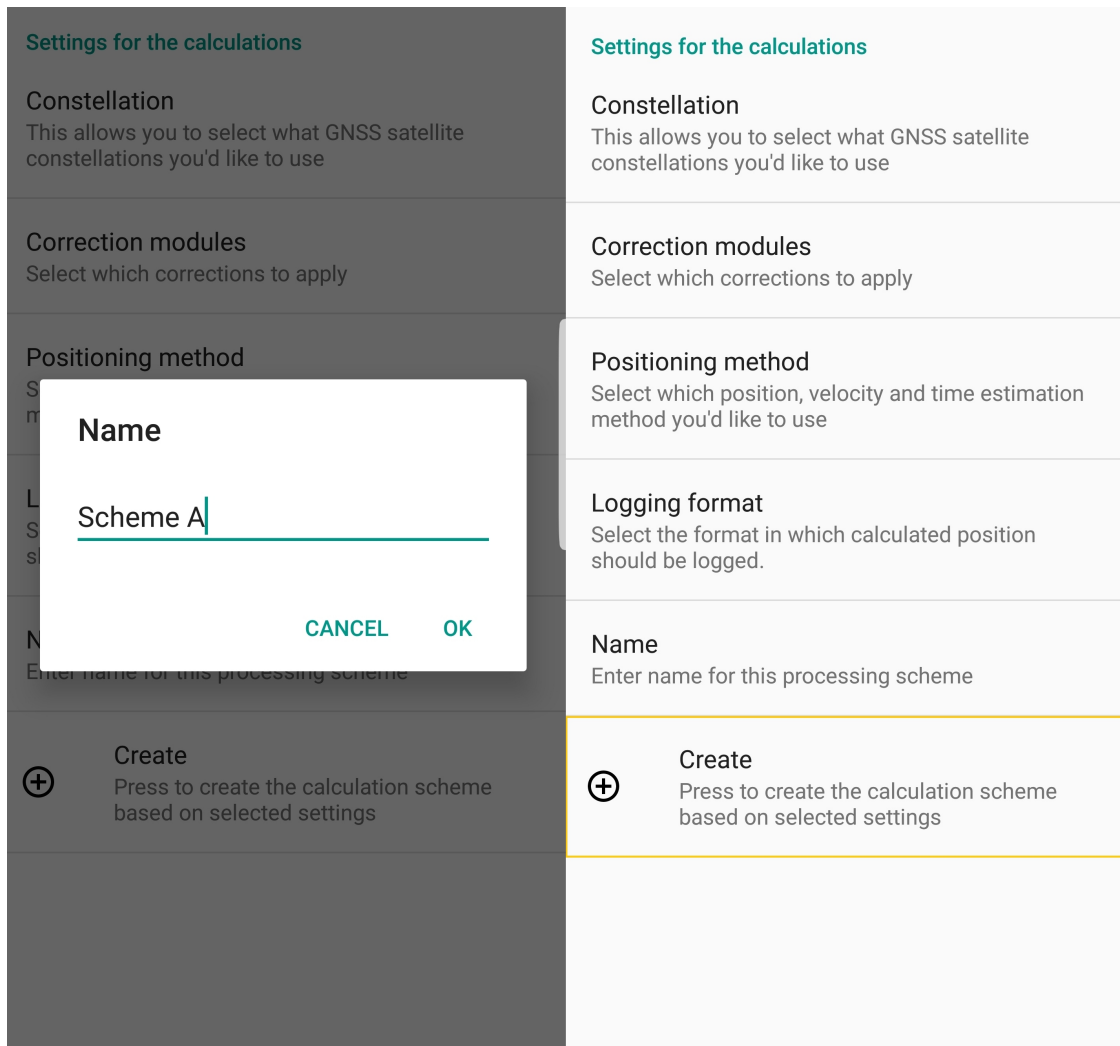
Let's begin by selecting the desired *Constellation* and *Correction modules*, one at a time.



Next we would like to select the *Positioning method* and the *Logging format*.




Finally we have to give a *Name* to the processing scheme and then press *Create*.



### 9.2.2 Modifying an existing one

To modify an existing processing scheme, from the *Main View* press the “gearbox” icon as shown below.



GNSS Compare


# GNSS Compare

Research lab in your pocket

## Constellation status


	Visible	Used
Galileo + GPS	18	9
GPS	14	8
Galileo	4	1

## Calculation results

	Lat	Lon	Alt	C. bias
Galileo+GPS	52.16956	4.48078	49.6	413536
GPS	52.16958	4.48078	48.3	413536
Galileo	52.16990	4.48107	20.7	413483

START RAW LOG

Developed with support of the European Space Agency



### Galileo+GPS

Galileo + GPS  
Dynamic EKF  
Tropospheric correction  
Relativistic path range correction  
NMEA

Activate: ☒

Save log: ☐

MODIFY

DELETE

### GPS

GPS  
Dynamic EKF  
Relativistic path range correction  
Tropospheric correction  
NMEA

Activate: ☒

Save log: ☐

MODIFY

DELETE

### Galileo

Galileo  
Dynamic EKF  
Relativistic path range correction  
Tropospheric correction  
NMEA

Activate: ☒

Save log: ☐

MODIFY

DELETE

In the right image you can find the created processing schemes. Additionally, there is a summary for each scheme with the chosen options. To change the settings of an existing scheme, press *Modify* and choose among the existing options. Moreover, in this view you also can enable/disable a processing scheme by pressing the *Activate* switch or enable/disable data logging with the *Save log* switch. A dedicated section will be made regarding the logging formats of GNSS Compare.



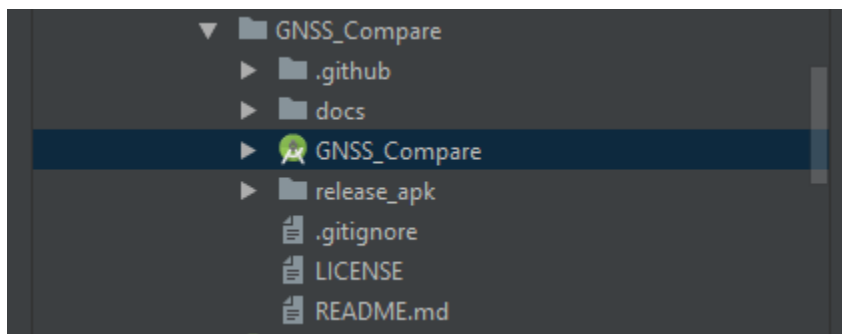
## Getting started with the code

### 10.1 Importing the project to Android Studio

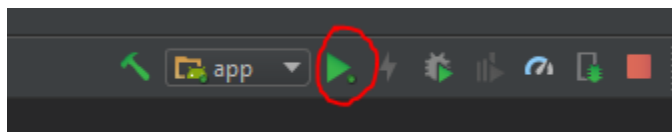
It's not necessary to use [Android Studio](#) to make modifications and build code, there are many other IDE's for Android development. It has been our choice so far, so we will be focusing our tutorials on that IDE at the moment.

Before importing the project, you have to clone the project [repository](#). The directory `GNSS_Compare` within that repository is the Android Studio project.

To import the project into Android Studio, find the *Import Project* option. It should be either under *File -> New -> Import Project*, or somewhere on the main screen of Android Studio. Navigate to the `GNSS_Compare` directory. You should find the `GNSS_Compare` project folder with the Android Studio logo next to it, as shown on image below.



Mark it, click Ok. The project should open and gradle will start to synchronize everything and generate its configuration files. This can take a moment. . . After gradle has finished, you're free to connect your phone and build the application by pressing the *Run app* button in the top, right corner.



## 10.2 Using the Google Maps Viewer

In order to use the Google Maps Viewer in GNSS Compare, you'll need to get your own Google Maps API key and paste it into the Android manifest (kind of).

To get the Google Maps SDK key, follow this [guide](#).

After you have the key (it will look like a string of random characters, starting with AIza), all you need to do is to copy and paste it to the `map_api_key.xml` file located in the `res/values` directory. Find the lines containing the following:

```
<string name="map_api_key">YOUR_API_KEY</string>
```

And replace `YOUR_API_KEY` with your API key. It should work right away. If not – you might need to clean and rebuild your project, or manually uninstall the application on your phone.

Remember not to share the api key with anyone! If you're using git, you can mark that file as one which should not be tracked with

```
git update-index --assume-unchanged GNSS_Compare/app/src/main/res/values/map_api_key.  
↪xml
```

This way, the file will remain in your repository, but any changes made to it will be not pushed to the remote.

### 11.1 Constellation

There are a few *Global Navigation Satellite Systems* on the planet. Or actually - around the planet: the *Global Positioning System* (GPS), owned by the US Government, *GLONASS*, owned by the Russian Federation, Chinese system *BeiDou*, and of course *Galileo*, owned by the European Union, which is set to soon become fully operational.

Each of those constellations consists of a set of 20-30 satellites orbiting above our heads at an altitude of around 20000 km. So many satellites are needed, because the system operator assures that from each spot on Earth, at any time of the day, at least four satellites are visible (and usually a lot more). Those satellites are constantly broadcasting a signal, which among other parameters contains a timestamp. The receiver then receives the signal, compares the timestamp with the current time, and thanks to that is able to calculate the distance to the satellite. Knowing the time of transmission and the satellites orbital parameters (retrieved from a special server in the form of *ephemeris* data or extracted from the received signal itself) it's also possible to calculate the satellite's position in space. Of course, the details of how those calculations are performed vary slightly from constellation to constellation.

In the context of GNSS Compare, a *Constellation*, is a class, which defines those two properties:

- is capable of converting the raw measurements extracted from the phone into pseudoranges,
- is able to calculate satellite's position in the time of transmission.

In this context, the constellations can be treated individually, e.g. we separate GPS from Galileo, and perform position determining calculations for them separately, or they can be treated together, as in our *Galileo+GPS* example. Developers must take care as combining constellations is not always that easy!

### 11.2 Corrections

As the signal travels from the satellite, it's prone to a number of sources of error (e.g., ionosphere, troposphere), which the user will have to take into account. For a more accurate positioning, we need to estimate the distance to the satellite with as good as possible, so we need to remove from the signal any disturbances we might know of.

Those disturbances are estimated using various, more or less complicated, mathematical models of the natural phenomena. Those models allow us to calculate corrections, which are later applied to the pseudoranges. The most commonly used corrections are for the ionosphere, troposphere and for those including the relativistic effects.

In the context of GNSS Compare, a `Correction` is a class, which provides a method to calculate the value of the correction, based on few parameters, which include:

- time of signal reception,
- receiver's approximate position,
- the satellite's position,
- additional data, stored in the *ephemeris* data

The general rule is simple – the more corrections are applied, the more accurate the final position.

### 11.3 PVT Estimator

The PVT estimators are algorithms which take as input satellite positions and pseudoranges to those satellites and aim to estimate the receiver's position, velocity and time. In some applications, it's sufficient to estimate just the position and time. Let's take a look at the parameters we wish to estimate. Position is quite obvious - that's what we would want to get from this whole process. In some cases, we can use the signal characteristics to improve the estimation of the receiver's velocity (e.g., using doppler measurements), thus increasing the accuracy of the position estimations. Additionally to the position related parameters we also need to estimate the receiver *clock bias* with respect to a certain GNSS time frame (e.g., Galileo System Time). This is handled by having the clock bias as one of the parameters to be estimated alongside with the position and velocity.

In the context of GNSS Compare, the `PvtMethod` class does exactly that. It's supposed to calculate the receiver's position, based on observed satellite parameters. Internally, it should be storing the calculated velocity and clock bias for enhanced processing, but from the point of view of GNSS Compare's framework, at the moment, the only value used outside of the `PvtMethod` class is the calculated position. But hey – there's of course room to improve.

---

## Android GNSS raw measurements

---

### (To be updated to support GNSS Dual-Frequency Android phones)

In the following sections we describe the algorithms used to compute the pseudoranges taken into account the used satellite navigation system. The following algorithms are based on the European GNSS Agency's (GSA) [White Paper on using GNSS Raw Measurements on Android devices](#).

At the code level, you can find the algorithms in the following Java classes:

- `GalileoConstellation`
- `GpsConstellation`

The variable names used in the description of the algorithms are the same as the ones in the GNSS Compare's code. Moreover, the definition of each used variable (e.g., *ReceivedSvTimeNanos*) can be found on the [Android Developer](#) webpage or in the white paper mentioned above. We will keep things straight forward in this section.

As an additional informative note, the pseudoranges computed here are based on the ranging codes that modulate the L1 carrier signal.

## 12.1 Galileo

Roughly speaking, the pseudorange is the difference between the time of signal reception and the time of signal transmission multiplied by the speed of light (for a more detailed definition check the [Glossary](#)). Therefore, let's see how we compute the time of signal reception with the Android GNSS raw parameters:

```
galileoTime = TimeNanos - (FullBiasNanos + BiasNanos);  
tRxGalileoTOW = galileoTime % Constants.NUMBER_NANO_SECONDS_PER_WEEK;  
tRxGalileoE1_2nd = galileoTime % Constants.NumberNanoSeconds100Milli;
```

It may look a bit strange that we compute two times of reception (*tRxGalileoTOW* and *tRxGalileoE1\_2nd*) however there is a reason behind this. We have to be aware of the fact that Galileo signals have more complex modulation schemes if compared with the legacy signals of GPS. In this sense, processing Galileo signals requires more effort from the GNSS receiver. Now in order to use the Galileo pseudoranges in the PVT estimation, these pseudoranges have to pass some kind of health check. One of these checks looks if the Time Of Week (TOW) parameter is decoded

or determined from other sources (e.g., mobile network), and the other one checks if the smartphone's GNSS receiver is locked on the Galileo E1 secondary code. We will see soon how this is handled. However, we will not deal with the theoretical background in order to reason the approach presented here because it does require some advanced receiver signal processing knowledge and at this point this is outside of our aims. In exchange, we can advise the curious minds to check a book on GNSS signal structures, like *Engineering Satellite-Based Navigation and Timing: Global Navigation Satellite Systems, Signals and Receivers* by John W. Betz.

Therefore we will use `tRxGalileoTOW` and `tRxGalileoE1_2nd` to compute two pseudoranges and we will use only one of them, the one that manages to pass the health check of course! Now let's compute the time of signal transmission:

```
tTxGalileo = ReceivedSvTimeNanos + TimeOffsetNanos;
```

The two pseudoranges are:

```
pseudorangeTOW = (tRxGalileoTOW - tTxGalileo) * 1e-9 * Constants.SPEED_OF_LIGHT;
pseudorangeE1_2nd = ((galileoTime - tTxGalileo) % Constants.
↳NumberNanoSeconds100Milli) * 1e-9 * Constants.SPEED_OF_LIGHT;
```

We have said that we need to test these two pseudoranges for some criteria. And the Java variable containing the *health status* or the *states* that we wish to find if they are true or not is:

```
int measState = measurement.getState();
```

With the help of the bitwise AND operation we can identify if the seeked states are true or not. Please check the [Android Developer](#) website to have a better insight of this process:

```
boolean towKnown = (measState & GnssMeasurement.STATE_TOW_KNOWN) > 0;
boolean towDecoded = (measState & GnssMeasurement.STATE_TOW_DECODED) > 0;
boolean codeLock = (measState & GnssMeasurement.STATE_GAL_E1C_2ND_CODE_LOCK) > 0;
```

Finally, we do the following check and we decide which computed pseudorange we use:

```
if ((towKnown || towDecoded)) {
    // use pseudorangeTOW
} else if (codeLock) {
    // use pseudorangeE1_2nd
}
```

## 12.2 GPS

We follow a similar approach for GPS also by starting to compute the time of signal reception:

```
gpsTime = TimeNanos - (FullBiasNanos + BiasNanos);
tRxGPS = gpsTime + TimeOffsetNanos;
```

In the next step we compute in a more straight forward way the GPS pseudorange:

```
weekNumberNanos = Math.floor((-1. * FullBiasNanos) / Constants.NUMBER_NANO_SECONDS_
↳PER_WEEK) * Constants.NUMBER_NANO_SECONDS_PER_WEEK;
pseudorange = (tRxGPS - weekNumberNanos - ReceivedSvTimeNanos) / 1.0E9 * Constants.
↳SPEED_OF_LIGHT;
```



We have to check if the computed pseudorange is usable in PVT or not. Therefore, we get the states status:

```
int measState = measurement.getState();
```

We apply again the bitwise AND operator to see if the TOW is decoded and if the receiver is locked on the signal's code:

```
boolean codeLock = (measState & GnssMeasurement.STATE_CODE_LOCK) > 0;
boolean towDecoded = (measState & GnssMeasurement.STATE_TOW_DECODED) > 0;
```

Additionally we can add an extra criteria, a criteria that checks for the uncertainty in the determined TOW:

```
private static final int MAXTOWUNCNS = 50;    // [nanoseconds]
boolean towUncertainty = measurement.getReceivedSvTimeUncertaintyNanos() <=
    MAXTOWUNCNS;
```

Finally we decide to use the GPS pseudorange if the following check is true:

```
if (codeLock && towDecoded && towUncertainty && pseudorange < 1e9) {

    // use pseudorange

}
```



## Implemented PVT Algorithms

In this section we provide the theoretical aspects behind the *GNSS Compare*'s PVT algorithms. The information here can be associated with the following Java classes:

- `StaticExtendedKalmanFilter`
- `DynamicExtendedKalmanFilter`
- `PedestrianStaticExtendedKalmanFilter` - this one sounds a bit strange, however bear with us as explanations will be given when the filter tuning is explained
- `WeightedLeastSquares`

### 13.1 Extended Kalman Filter

One of the estimation techniques implemented in the *GNSS Compare* framework is the Kalman Filter. Taking into account that the measurement model is linearized about the time predicted position, in fact the implementation is an Extended Kalman Filter (EKF).

In this section we describe the theoretical aspects of the EKF implementation such that the curious minds can understand easily what is behind *GNSS Compare*'s awesome algorithms. We are interested to implement the EKF for two types of users: a *static user* and a *dynamic user*.

Therefore we will describe how the state vector is defined, or in other words, the vector containing the parameters that we wish to estimate (hint: the parameters are related to the *GNSS Compare*'s user position!), and also what dynamic and measurements models we have considered. And as *bonus* we will also write about the tuning of the EKFs.

First things first! Let's remember the Kalman Filter equations, *the implemented ones*, in order to make the rest of this section more enjoyable.

We have the time prediction of the state vector ( $\mathbf{x}$ ) and it's variance-covariance matrix ( $\mathbf{P}$ ):

$$\begin{aligned}\hat{\mathbf{x}}_k^- &= \mathbf{F}_k \hat{\mathbf{x}}_{k-1}^+ \\ \mathbf{P}_k^- &= \mathbf{F}_k \mathbf{P}_{k-1}^+ \mathbf{F}_k^T + \mathbf{Q}_k.\end{aligned}$$

In the next step we can compute the innovation vector ( $\gamma$ ) and it's variance-covariance matrix ( $S$ ) with the help of the observation vector ( $z$ ), the observation matrix ( $H$ ) and the measurement noise matrix ( $R$ ):

$$\gamma_k = z_k - H_k \hat{x}_k^-$$

$$S_k = H_k P_k^- H_k^T + R_k.$$

We are almost there, we just need to compute the famous Kalman gain ( $K$ )!

$$K_k = P_k^- H_k^T S_k^{-1}.$$

Finally the measurement update step is:

$$\hat{x}_k^+ = \hat{x}_k^- + K_k \gamma_k$$

$$P_k^+ = (I_k - K_k H_k) P_k^-.$$

However, before explaining how the EKF for the *static user* and the *dynamic user* was implemented, we still need to talk about the measurement model based on the GNSS pseudoranges retrieved from the smartphone's GNSS receiver. If you are familiar with this concepts, you can skip the following section.

### 13.1.1 Pseudorange measurement model

For a code-based pseudorange (PRC) we have the following (non-linear) equation taking into account the satellite clock bias ( $\delta t^S$ ), the delay caused by the ionosphere ( $d_{ion}$ ), the delay caused by the troposphere ( $d_{trop}$ ) and the receiver noise ( $\epsilon$ ).

$$PR_c = \rho + \delta t_R - \delta t^S + d_{ion} + d_{trop} + \epsilon$$

We know, there are more effects that are perturbing the GNSS measurements, however we wish to keep things as simple as possible and the interested persons can always consider more error sources in the *GNSS Compare's* code.

The above equation is non-linear because of the geometric distance ( $\rho$ ) between the receiver and the GNSS satellite. Luckily we can linearize it if we have knowledge about an approximated position of the receiver ( $X_0, Y_0, Z_0$ ), which we do! We do have from the time prediction step of the EKF. Taking this into account and applying a first order Taylor series expansion we obtain:

$$PR_c - \rho_0 + \delta t^S - d_{0,ion} - d_{0,trop} = -\frac{X^S - X_0}{\rho_0} \Delta X - \frac{Y^S - Y_0}{\rho_0} \Delta Y - \frac{Z^S - Z_0}{\rho_0} \Delta Z + \delta t_R.$$

On the left side of the equation we have moved every term that can be computed. The subscript 0 means that those parameters are estimated by using the approximate receiver position information. On the right hand side we have the unknowns ( $\Delta X, \Delta Y, \Delta Z, \delta t_R$ ) and their coefficients. Based on the linearized pseudorange equation one can form the observation matrix ( $H$ ).

*Practical advise: Take care that the unknowns from the linearized pseudorange equations are not the same as the position related unknowns that we are estimating directly in the EKF state vector. Check the GNSS Compare code ( e.g., StaticExtendedKalmanFilter class ) to understand how this is handled.*

Good, now we can see how the EKF was implemented for the *static user* and the *dynamic user*!

### 13.1.2 Static user

In the case of a static user we have the following state vector at the epoch  $k$ :

$$x_k = \begin{pmatrix} X & Y & Z & \delta t_R & \dot{\delta t}_R \end{pmatrix}^T.$$

In the above expression X, Y and Z are the coordinates in Earth Centered Earth Fixed (ECEF) frame and the last two parameters are the receiver clock bias and the receiver clock drift. All the parameters are expressed in units of meters.

Now that the state vector is defined, we can move on by choosing the dynamic model. Before moving further let's think a bit about this aspect. A static user doesn't *change* his/hers position, therefore this means that over time the X, Y, Z coordinates remain the same! We only have to take care of how we model the dynamic behavior of the receiver's clock, which is approximated to be:

$$\begin{aligned}\delta t_{R,k} &= \delta t_{R,k-1} + \Delta T \dot{\delta t}_{R,k-1}, \\ \dot{\delta t}_{R,k} &= \dot{\delta t}_{R,k-1}.\end{aligned}$$

Having in view all of this information we can define the transition matrix (F) of the filter as:

$$\mathbf{F}_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta T \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

We are almost done with the dynamic model elements. The only thing that we need now is the process noise matrix (Q). Because the process noise matrix contains the uncertainty we have in the dynamic model that we consider, we have to define it accordingly. In the static case we can assume that the user is not moving and that the receiver clock has some frequency and phase errors. In order to fully understand this reasoning, the interested reader is advised to check the following book: *Introduction to Random Signals and Applied Kalman Filtering* by Robert Grover Brown and Patrick Y. C. Hwang. Therefore, the process noise matrix is approximated to be:

$$\mathbf{Q}_k = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & S_f + \frac{S_g \Delta T^3}{2} & \frac{S_g \Delta T^2}{2} \\ 0 & 0 & 0 & \frac{S_g \Delta T^2}{2} & S_g \Delta T \end{pmatrix}.$$

In the above expression the receiver clock related parameters are expressed as:

$$\begin{aligned}S_g &\approx 2\pi^2 h_{-2}, \\ S_f &\approx \frac{h_0}{2}.\end{aligned}$$

The parameter h-2 and h0 are the Power Spectral Densities (PSD) of the random walk frequency noise and of the white noise, as defined in the suggested book above. Some typical values for a low quality Temperature Compensated Crystal Oscillator (TCXO) are 2e-20 and 2e-19 (in seconds). A practical advise before using this values is to take care that we are dealing with the parameters of a variance-covariance matrix and also that they have to be converted in units of meters (remember that we have expressed the receiver clock states in units of meters).

So basically we are done with the *static user* case. That's great as we can move to the dynamic one!

### 13.1.3 Dynamic user

In the case of a dynamic user there are few aspects that one has to consider. Let's start again by defining the new state vector:

$$\mathbf{x}_k = \left( X \ U \ Y \ V \ Z \ W \ \delta t_R \ \dot{\delta t}_R \right)^T.$$

We can already observe that we have three more parameters to estimate (U, V, W) which are the velocities on the X, Y and Z directions. If our state vector is modified (with respect to the static case) then our intuition will tell us that we

need to define a new transition matrix and a new process noise matrix. Which is exactly what we are going to do next, therefore:

$$\mathbf{F}_k = \begin{pmatrix} 1 & \Delta T & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta T & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta T & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta T \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

For the process noise matrix we use the approach presented in the book of Robert Grover Brown and Patrick Y. C. Hwang ( *Introduction to Random Signals and Applied Kalman Filtering* ). Indeed, is the third time we refer to this book in the implemented PVT algorithms section, however you can trust us that is a very good one!

$$\mathbf{Q}_k = \begin{pmatrix} \frac{S_x \Delta T^3}{3} & \frac{S_x \Delta T^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{S_x \Delta T^2}{2} & S_x \Delta T & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{S_y \Delta T^3}{3} & \frac{S_y \Delta T^2}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{S_y \Delta T^2}{2} & S_y \Delta T & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{S_z \Delta T^3}{3} & \frac{S_z \Delta T^2}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{S_z \Delta T^2}{2} & S_z \Delta T & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & S_f + \frac{S_g \Delta T^3}{2} & \frac{S_g \Delta T^2}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{S_g \Delta T^2}{2} & S_g \Delta T \end{pmatrix}$$

The parameters  $S_x$ ,  $S_y$  and  $S_z$  are the spectral amplitudes that reflect the position random process. Unfortunately, setting their values is not as straight forward as for the receiver clock states. We have to rely on what we call the *tunning* process which is modifying the values in  $\mathbf{Q}$  and  $\mathbf{R}$  experimentally (i.e., trial and error). Just as a note, this can be avoided by designing and implementing *adaptive* estimators. Who knows, maybe you (the reader) will decide to implement some nice ideas now that this possibility is enabled with *GNSS Compare*'s flexible framework.

*Practical advise: When the observation matrix ( $\mathbf{H}$ ) is being built do consider that it's size is defined in the following way: the number of rows is the number of measurements and the number of columns is the number of unknowns. Therefore when switching from the static case to the dynamic case,  $\mathbf{H}$  changes also. We mention this just to be sure that a possible conceptual hiccup is avoided.*

### 13.1.4 Filter tunning

Because at the moment we are dealing with a standard EKF and not an adaptive one this means that we have to assign values in the process noise matrix ( $\mathbf{Q}$ ) and in the measurement noise matrix ( $\mathbf{R}$ ) such that the filter is tuned to our situation.

Let's start with the  $\mathbf{R}$  matrix. We set  $\mathbf{R}$  to be a diagonal matrix containing the variances of each pseudorange measurement. The measurement noise matrix being diagonal relies on the assumption that there is no cross-correlation between the measurements coming from different satellites ( *an assumption that is not entirely represeting the reality, however it fits most of the applications* ). Therefore, the diagonal elements of the  $\mathbf{R}$  matrix are:

$$\mathbf{R}_{ii,k} = \sigma_{ii}^2.$$

To keep things relatively simple, we can assign the value for the sigma 10 meters ( *don't forget to square it before putting it in  $\mathbf{R}$*  ). Another assumption that is made is that the measurements received at the  $k$ -th epoch have equal variances ( *ok, this assumption is not true at all* ). However here is an idea for you, maybe you can try investigating some interesting measurement weighing methods and then *compare* (the main keyword of the whole project) the results you get with our not so realistic assumption. Let the researcher within you thrive!

Let's move to the Q matrix now. For this we present three tuning examples: static, pedestrian and dynamic.

### Static tuning

For the static case we have already seen that we only have to take care about the process noise of the receiver clock states. So the values that we are assigning to the PSD of the random walk of the frequency noise and of the white noise are:

$$h_{-2} = 2e - 20 \, c^2,$$

$$h_0 = 2e - 19 \, c^2.$$

In the above we use the  $c$  notation for the speed of light.

### Pedestrian tuning

Intuitively we should have used the EKF designed for a dynamic user in this situation. It would only make sense as a pedestrian *changes* his/hers position over time. However, one must take into account that the raw measurements delivered by the smartphone's GNSS receiver are quite noisy and if there are no other means to detect the motion of the user (e.g., using an Inertial Measurement Unit) then estimating the velocities can make our results not so accurate. Having this situation in view we have found a workaround: we use the EKF designed for a static user and we let some process noise for the X and Y coordinates (*unless one of our users is not Superman we are not that interested in the Z direction*). This means that we have the following Q matrix:

$$\mathbf{Q}_k = \begin{pmatrix} 0.2 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & S_f + \frac{S_g \Delta T^3}{2} & \frac{S_g \Delta T^2}{2} \\ 0 & 0 & 0 & \frac{S_g \Delta T^2}{2} & S_g \Delta T \end{pmatrix}.$$

The value 0.2 was chosen by trial and error and it fits a *slow* walking pedestrian. We hope that the name of the Java class `PedestrianStaticExtendedKalmanFilter` makes a little bit more sense now.

### Dynamic tuning

Finally we have arrived at the final case regarding the tuning of the dynamic EKF. Again the following values were determined empirically:

$$S_X = S_Y = 0.8,$$

$$S_Z = 0.08.$$

## 13.2 Weighted Least Squares

*GNSS Compare* offers also the possibility to change the PVT estimator if the user wishes so. By not requiring knowledge about the dynamics, Weighted Least Squares (WLS) can be used to estimate the position using only the pseudorange measurements. However there are some drawbacks like: the quality of the estimations fully depends on the quality of the measurements and also the WLS requires a minimum number of measurements (typically 4 if we want to estimate the 3D position and the receiver clock bias).

Nevertheless is useful to have such an estimator as its behavior can be studied in real-time/post-processing in comparison with an EKF. And all this thanks to *GNSS Compare*!

Although the pseudorange measurement model was presented in the EKF description we will do it one more time just for the sake of completion.

### 13.2.1 Pseudorange measurement model

The linearized code-based pseudorange measurement is:

$$PR_c - \rho_0 + \underbrace{(\delta t^S - d_{0,\text{ion}} - d_{0,\text{trop}})}_{\text{Corr}} = -\frac{X^S - X_0}{\rho_0} \Delta X - \frac{Y^S - Y_0}{\rho_0} \Delta Y - \frac{Z^S - Z_0}{\rho_0} \Delta Z + \delta t_R.$$

Let's also express the unit line of sight vector and the position related unknowns as:

$$\mathbf{u} = \left[ -\frac{X^S - X_0}{\rho_0}, -\frac{Y^S - Y_0}{\rho_0}, -\frac{Z^S - Z_0}{\rho_0} \right],$$

$$\delta \mathbf{r} = [\Delta X, \Delta Y, \Delta Z].$$

For  $n$  observed satellites we have the following measurement model:

$$\underbrace{\begin{pmatrix} PR_c^1 - \rho_0^1 + Corr^1 \\ PR_c^2 - \rho_0^2 + Corr^2 \\ \vdots \\ PR_c^n - \rho_0^n + Corr^n \end{pmatrix}}_{\mathbf{z}} = \underbrace{\begin{pmatrix} \mathbf{u}^1 & 1 \\ \mathbf{u}^2 & 1 \\ \vdots & \vdots \\ \mathbf{u}^n & 1 \end{pmatrix}}_{\mathbf{H}} \underbrace{\begin{pmatrix} \delta \mathbf{r}^T \\ \delta t_R \end{pmatrix}}_{\mathbf{x}}.$$

To estimate the vector of unknowns ( $\mathbf{x}$ ) in the WLS sense, we proceed in the following way:

$$\hat{\mathbf{x}}_{\text{WLS}} = (\mathbf{H}^T \mathbf{W} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{W} \mathbf{z}.$$

*Practical advise: In the WLS case, as the position is concerned, we are estimating the difference between the approximated position and the true position until this difference is below a certain threshold. We encourage you to check the `WeightedLeastSquares` class to see how this is handled.*



---

## Example of analysis

---

(To be updated with an example of analysis for the GNSS Dual-Frequency Android phones)

This section provides information about the scenarios in which *GNSS Compare* was tested and the PVT performance obtained from its estimation algorithms (e.g., Extended Kalman Filter). Furthermore, the analysis presented here serves also as an example of how *GNSS Compare* can be used for algorithmic performance assessment. For a preliminary PVT performance assessment the following scenarios were considered: **Static user**, **Pedestrian user** and **Dynamic user**.

With this section we would like to give you an idea of how *GNSS Compare* can be used. The application allows data logging (e.g., results of the PVT estimations) in different formats, like NMEA and a custom one. These files can be retrieved from the phone and then processed in your favourite programming environment for analysis. More details about the logging formats of *GNSS Compare* will be given soon.

*Note 1: Please be aware that the results presented here are specific to the environment/time when they were generated and they cannot be interpreted in a general sense.*

*Note 2: The Extended Kalman Filters were initialized with the Android FINE location.*

### 14.1 Static user

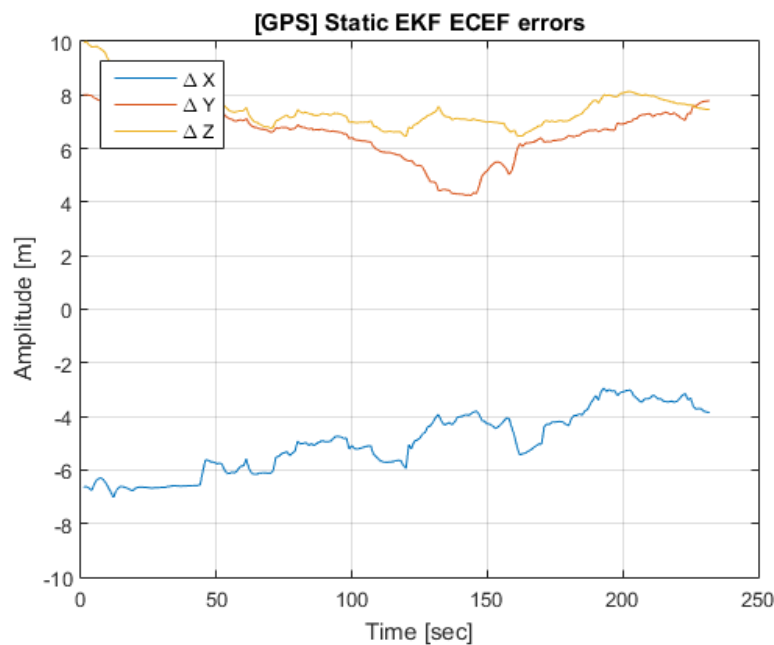
Let's take a look at some details about this scenario:

- Reference location: Latitude 52.16954469, Longitude 4.48089101, Altitude 55.48 m
- Data collection duration: approximately 4 minutes
- Enabled constellations: GPS, Galileo+GPS
- Number of used satellites: 4 Galileo and 5 GPS

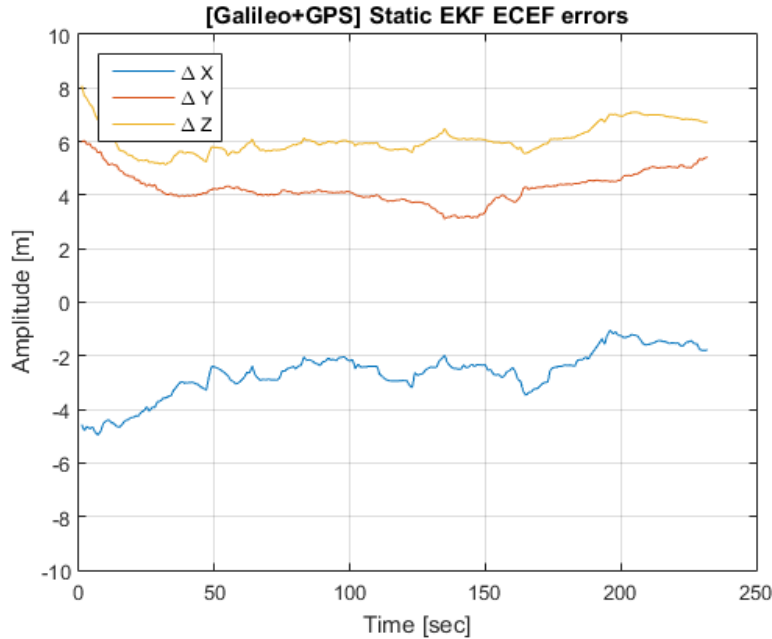
After the results of the PVT estimations were obtained from the logged files of *GNSS Compare*, they were projected in Google Earth as seen in the figure below for an initial analysis.



In this scenario one can observe in the above figure that the computations based on Galileo+GPS are closer to the reference when compared with GPS only. In order to understand these aspects in a more detailed manner, the behavior of the errors with respect to the reference can be studied. The errors are computed based on the cartesian coordinates within the Earth Centered Earth Fixed (ECEF) frame.



The error evolutions for GPS only PVT are presented in the above figure and it can be directly observed that they are quite large and with a high variance. Let's see what happens if we add Galileo in the processing.



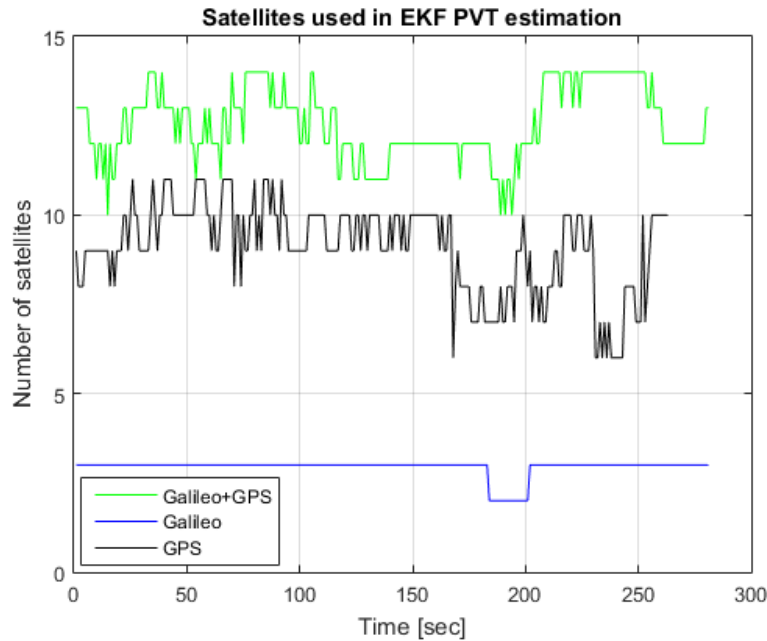
For the case when the PVT is computed using both Galileo and GPS, the above figure shows improvements when compared with the solution based only on GPS.

## 14.2 Pedestrian user

This scenario is defined in the following way:

- User dynamics: Walking pedestrian
- Location: The European Space Research and Technology Centre (ESTEC)'s parking lot
- Data collection duration: approximately 4 and half minutes
- Enabled constellations: GPS, Galileo+GPS
- PVT estimator: Extended Kalman Filter
- Number of satellites: On average 3 Galileo and 8 GPS

As for this case there is no reference trajectory available the results are analyzed at the observed satellite level and at the projection of the estimated position in Google Earth.



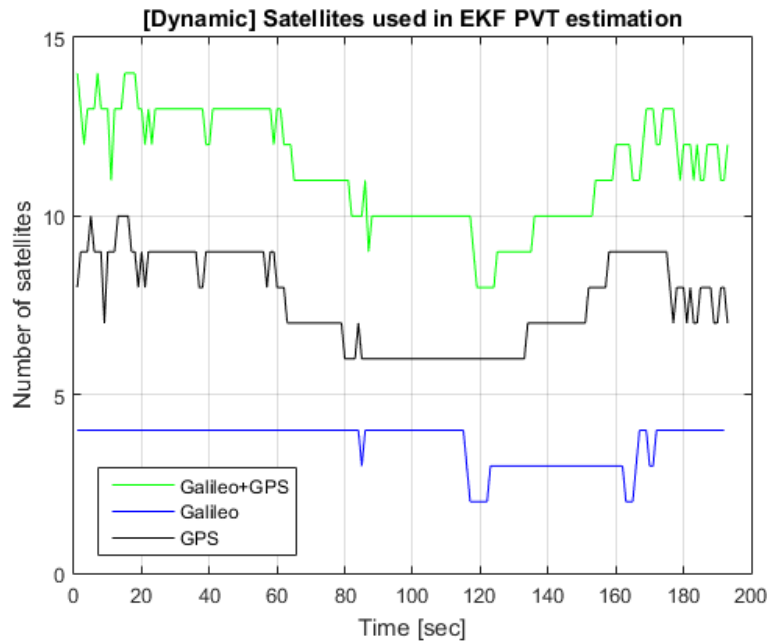
In the above figure the estimation of the trajectory that is based only on GPS does not follow too accurately the real pedestrian motion. However when both Galileo and GPS satellites are used together the position estimation is improved obtaining a pedestrian path closer to reality.

### 14.3 Dynamic user

And the last scenario has the following characteristics:

- User dynamics: Cycling user
- Location: ESTEC
- Data collection duration: approximately 3 minutes
- Enabled constellations: GPS, Galileo+GPS

- PVT estimator: Extended Kalman Filter
- Number of satellites: On average 4 Galileo and 8 GPS



Even with this rather simplistic analysis one can gain some interesting insights. We do hope that you have now a more clear idea about the possibilities that *GNSS Compare* can open!