Licence Professionnelle SEICOM

Rapport de stage :

Implémentation des algorithmes de géolocalisation sur Android

Réalisé par :

José Gilberto RESÉNDIZ FONSECA

20 juillet 2018

Stage encadré par :

Miguel ORTIZ
Ingénieur de recherche
GEOLOC de l'IFSTTAR
miguel.ortiz@ifsttar.fr

Harold MOUCHÈRE
Enseignant à l'IUT
de Nantes
harold.mouchere@univ-nantes.fr

I Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé de près ou de loin au bon déroulement de mon stage de fin d'études.

Je tiens d'abord à remercier Miguel Ortiz, maître de stage, de m'avoir donné l'opportunité de travailler sur ce projet qui m'a fait découvrir le milieu de la recherche et la géolocalisation. Ses connaissances dans le domaine et son dynamisme m'ont aidé à comprendre et assimiler les notions de la géolocalisation.

Je remercie Valérie Renaudin, directrice du laboratoire GEOLOC, pour m'avoir accueillie au sein du laboratoire ainsi que pour sa bonne attitude, elle m'a toujours encourager à essayer de nouvelles expériences pour le bien de ma carrier professionnelle.

Je remercie Johan Perul et Antoine Blin pour m'aider à comprendre les principes de positionnement GNSS et spécialement à Taia Alaoui pour tout le temps passé à m'expliquer les principes de positionnement GNSS vus du coté mathématique pour la bien compréhension de la géolocalisation.

Je remercie de façon plus large tout l'équipe GEOLOC pour son accueil et ambiance chaleureuse.

Je remercie également tout le corps professionnel de l'IUT de Nantes pour le travail énorme qu'il effectue pour nous crée les conditions les plus favorables pour le déroulement de nos études.

Enfin, je remercie le programme MEXPROTEC pour m'avoir donné l'opportunité d'étudier en France, et tous les personnes qui ont fait possible mon séjour en France.

Table des matières

II Introduction

En mai 2016, Google a annoncé la disponibilité des données brutes du chipset GNSS embarqué dans les nouveaux smartphones avec Android 7. Pour la première fois, les développeurs pourraient avoir accès aux mesures de phase et code et décoder des messages de navigation des satellites à partir des dispositifs Android.

Dans le cadre de ma Licence Professionnel, j'ai souhaité réaliser mon stage dans une entreprise dans le domaine des systèmes embarqués, car ce sujet m'intéresse et je souhaitais savoir si mes compétences acquises au cours de mes études étaient appropriées pour travailler dans ce domaine, en même temps acquérir des nouvelles compétences et surtout agrandir mes connaissances en programmation orienté objet (POO) qui est le sujet qui m'intéresse le plus.

L'institut français IFSTTAR est connu comme un acteur majeur de la recherche européenne sur la ville et des territoires. Avec le lancement de la nouvelle version d'Android et l'annonce de Google, le laboratoire GEOLOC de l'IFSTTAR m'a proposé le stage sur l'implémentation des algorithmes de géolocalisation sur Android en utilisant cette nouvelle caractéristique de l'API Android 7. Un smartphone nous ne pouvons pas le considérer comme un système embarqué, mais les applications qui fonctionnent sous Android sont développées en utilisant la POO en Java, et c'est ça qui m'a fait prendre ce stage qui m'a permis de mettre en pratique mes compétences et d'apprendre un peu sur la géolocalisation par satellite.

Dans un premier temps nous décrirons l'IFSTTAR et ses axes de recherche, et plus en détail le laboratoire GEOLOC ou j'ai réalisé mon stage. Ensuite, nous présenterons les différentes étapes du projet qui à la fin vont permettre de localiser un point sur terre en utilisant la plateforme Android. Nous terminerons par une conclusion sur le projet en citant les résultats que j'ai obtenus et les difficultés rencontrées, et pour finir une conclusion personnelle.

III IFSTTAR

Acteur majeur de la recherche européenne sur la ville et les territoires, les transports et le génie civil, L'IFSTTAR, l'Institut français des sciences et technologies des transports, de l'aménagement et des réseaux, est né le 1er janvier 2011 de la fusion de l'INRETS et du LCPC.

L'IFFSTAR est un établissement public à caractère scientifique et technologique, placé sous la tutelle conjointe du Ministère de l'Environnement, de l'Énergie et de la Mer et du ministère de l'enseignement supérieur et de la recherche.

III.1 Quelques chiffres clés

- 1094 agents (soit 1069 ETP)
- 104 M€ de budget
- 6 sites en France
- Plus de 50 équipements scientifiques remarquables
- 90 projets européens (7ème PCRD)
- Plus de 50 équipements scientifiques remarquables
- 80 brevets actifs
- 90 thèses soutenues
- 160 contrats de recherche

III.2 Implantations

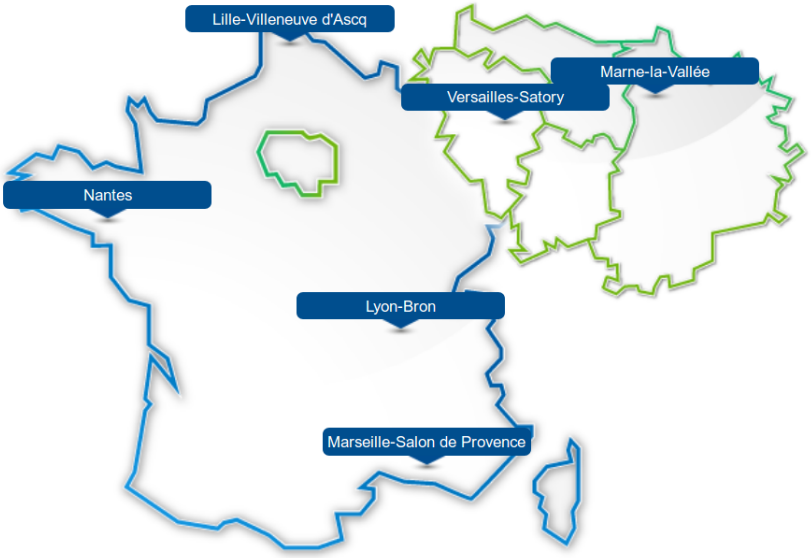


Figure 1 – Implantations de l'IFSTTAR

III.3 Mission

L'institut a pour missions de réaliser ou faire réaliser, d'orienter, d'animer et d'évaluer des recherches, des développements et des innovations dans les domaines du génie urbain, du génie civil et des matériaux de construction, des risques naturels, de la mobilité des personnes et des biens, des systèmes et des moyens de transports et de leur sécurité, des infrastructures, de leurs usages et de leurs impacts, considérés des points de vue technique, économique, social, sanitaire, énergétique, environnemental et humain.

L'institut a notamment vocation à :

- Conduire des recherches fondamentales et appliquées, des études méthodologiques et des développements d'essais et de prototypes
- Mener tout travaux d'expertise et de conseil dans les domaines mentionnés au premier alinéa du présent article
- Mettre en œuvre une politique d'information scientifique et technique et assurer la diffusion des connaissances acquises, notamment par les publications, la réglementation technique et la normalisation
- Mener une politique de valorisation des résultats de ses travaux de recherche scientifique et technologique, notamment sous forme d'appui technique, de transfert de technologie, d'essai et de certification
- Contribuer à la formation à la recherche et par la recherche ainsi qu'à la formation initiale et continue
- Contribuer au rayonnement international et à l'exportation de l'expertise et des techniques qu'il développe

III.4 Organigramme

Le laboratoire GÉOLOC ou j'ai réaliser mon stage se trouve sur le site Nantes dans le département AME.

FIGURE 2 – Organigramme départements/laboratoires

III.5 Laboratoire GÉOLOC

Le laboratoire GEOLOC mène des recherches et expertises sur les méthodes et systèmes de géolocalisation afin d'accompagner l'évolution des pratiques de mobilité et des moyens de transport. Il répond aux défis technologiques et sociétaux liés aux transitions numériques et à une plus grande connectivité de tous les acteurs du transport. Ses activités sont structurées autour de deux défis :

- Inventer des algorithmes de géolocalisation selon une approche ubiquitaire
  - ➤ Fusion multi-capteurs (indoor/outdoor)
  - ➤ Adaptation au contexte de déplacement et au profil de mobilité. Intégration de l'humain dans le développement d'algorithmiques
  - ➤ Respect de la protection des données privées dès la couche technologique
  - ➤ Fabrication de plateformes de recherche dédiées aux humains
- Définir et proposer des méthodes d'évaluation des performances de localisation au regard des besoins des systèmes de transport intelligents (ITS)
  - ➤ Transport routier (véhicule connecté, transport public)
  - ➤ Voyageurs (partage du transport, itinéraires individualisés au profil de mobilité)
  - ➤ Logistique
  - ➤ Enquêtes ménages automatisées

Pour relever ces défis, GEOLOC adopte une approche pluridisciplinaire : compétences de positionnement par satellites GNSS, de traitement de signal RF, d'informatique, de robotique et de géomatique. Il s'appuie sur des partenariats académiques et industriels et accompagne la politique européenne en matière d'homologation des systèmes de localisation GNSS et d'exploitation de Galileo (applications grand public, IoT). GEOLOC participe à des projets financés par l'Europe, l'ANR, la DGA, le FUI, la Région Pays de Loire... Il contribue aussi à l'animation scientifique interne avec le co-pilotage du projet fédérateur Mobilités et Transitions Numériques et internationale avec l'organisation d'événements internationaux (conférence IEEE IPIN).

IV Principe de positionnement par satellite

La géolocalisation par satellite en quelques mots consiste en un ensemble de satellites appelés constellation, ces satellites orbitent la terre à environ 20,000 km d'altitude, chacun des satellites diffuse des données sur ses coordonnées terrestres et des mesures du temps à travers des ondes radio, ces ondes radio sont reçues par l'antenne du récepteur GNSS qui décode les données pour après les passer au programme qui récupère les informations du satellite à partir des données, et une fois que nous avons les informations de 4 satellites au même temps, nous pouvons finalement calculer la position du récepteur.

IV.1 Constellations et temps de référence

Actuellement il y a 4 systèmes de satellites de navigation globaux (GNSS), ce sont la constellation GPS par les État-Unis, GLONASS par la Russie, GALILEO par l'Union Européenne et BEIDOU par la Chine. Chacune des constellations a environ 30 satellites et utilise un propre temps de référence. Chaque temps de référence a un décalage par rapport au temps UTC, ces décalages sont montrés dans le diagramme ci-dessus.

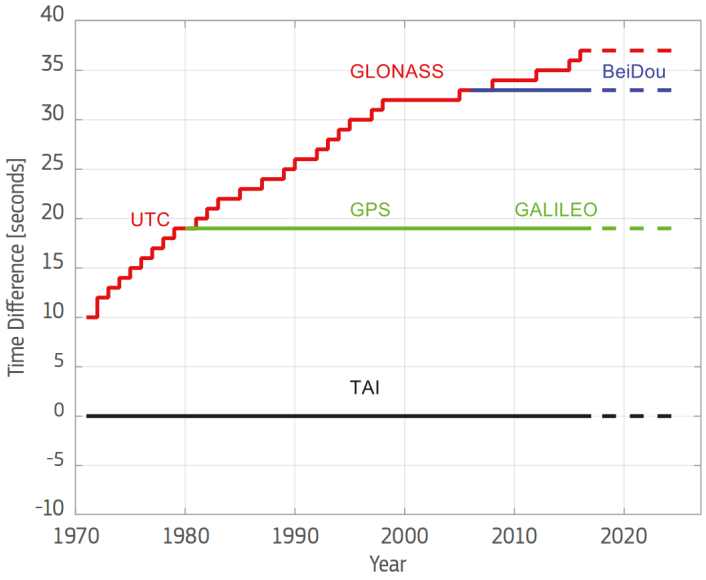


Figure 3 – Temps de référence par constellation

IV.2 Calcul de position par trilatération

Pour pouvoir calculer un point sur terre il faut savoir au moins les informations de 4 satellites différents, ces informations sont la pseudodistance entre le récepteur et le satellite, les coordonnées terrestres du satellite et l'erreur d'horloge du satellite. Avec ces informations nous pouvons imaginer que la pseudodistance est le rayon d'une géante sphère et le satellite est le centre de la sphère, donc nous pouvons utiliser une méthode mathématique appelée trilatération qui permet de trouver la position relative d'un point en utilisant la géométrie des sphères et en plus une correction du temps.

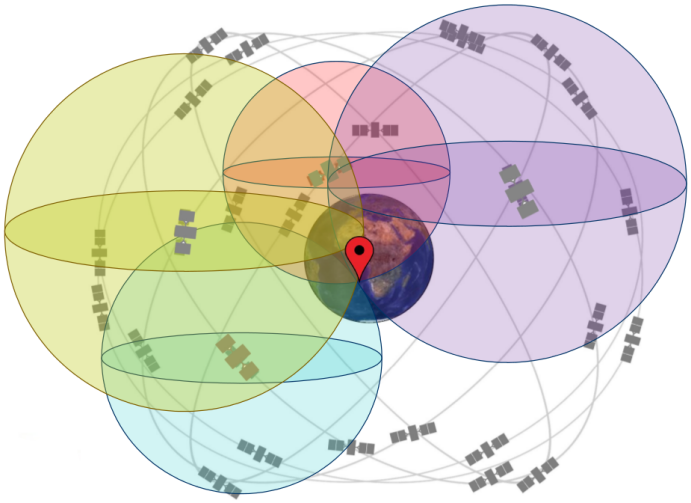


FIGURE 4 – Trilatération avec des sphères

Nous pouvons résumer la figure au-dessus comme un système de 4 équations, une pour chaque satellite, et 4 inconnues qui sont les coordonnées du récepteur en x, y et z, et une erreur de distance à cause de l'horloge du récepteur, donc les équations seraient :

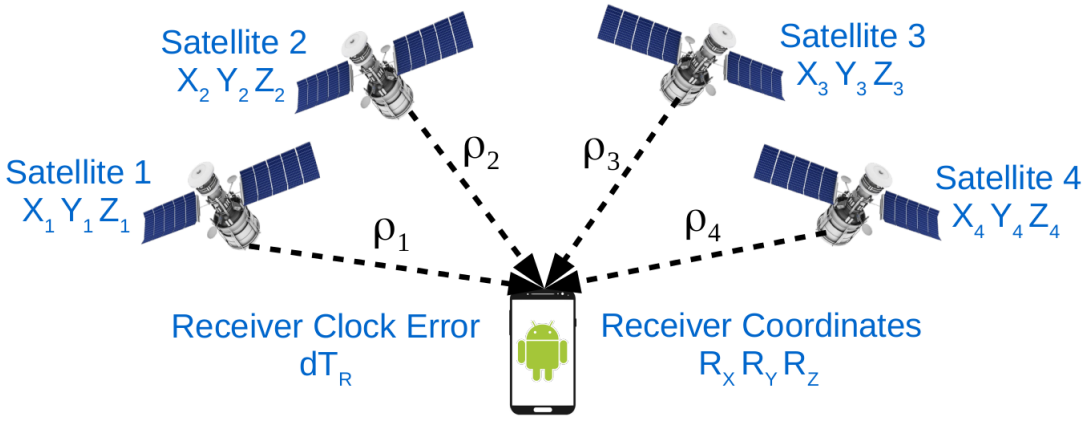


FIGURE 5 – Solution de navigation

$$\rho_1 = \sqrt{(X_1 - R_x)^2 + (Y_1 - R_y)^2 + (Z_1 - R_z)^2} + c(dT_R) \quad (1)$$

$$\rho_2 = \sqrt{(X_2 - R_x)^2 + (Y_2 - R_y)^2 + (Z_2 - R_z)^2} + c(dT_R) \quad (2)$$

$$\rho_3 = \sqrt{(X_3 - R_x)^2 + (Y_3 - R_y)^2 + (Z_3 - R_z)^2} + c(dT_R) \quad (3)$$

$$\rho_4 = \sqrt{(X_4 - R_x)^2 + (Y_4 - R_y)^2 + (Z_4 - R_z)^2} + c(dT_R) \quad (4)$$

avec :

— ρ_n : la pseudodistance du satellite n
— X_n, Y_n, Z_n : les coordonnées du satellite n
— R_X, R_Y, R_Z : les coordonnées du récepteur
— dT_R : l'erreur d'horloge du récepteur
— c : la vitesse de la lumière

IV.3 Pseudodistance

Le calcul de pseudodistance est fait en utilisant la vitesse des ondes radio, en l'occurrence la vitesse de la lumière, et les temps de transmission et réception du signal, on l'appelle pseudodistance car ce n'est pas la distance absolue entre le satellite et le récepteur puisque le signal souffre des perturbations atmosphériques qui font retarder le signal, ce qui signifie qu'on a une fausse distance. Le signal que diffuse le satellite contient le temps de transmission, et c'est le récepteur qui calcule le temps de réception, donc le calcul de la pseudodistance est obtenu avec la formule :

$$\rho = (t_{Rx} - t_{Tx}) * c \quad (5)$$

avec :

— ρ : la pseudodistance
— t_{Rx} : le temps de réception
— t_{Tx} : le temps de transmission
— c : la vitesse de la lumière

IV.4 Coordonnées satellite

La récupération des coordonnées du satellite dépend de la constellation à laquelle le satellite appartient, on appelle message de navigation les données sur la position du satellite, diffusé par le même satellite. La structure du message de navigation, les données que le message contient et le calcul de la position du satellite sont expliques dans les ICD's de chaque constellation, le ICD est un document gouvernemental qui nous explique tout sur le message de navigation et il est en libre accès sur internet. Les messages de navigation sont divisés en différentes *subframes* qui reconstruisent le message complet. Les données du message ne changent pas pendant une certaine période, et le début de cette période s'appelle époque de référence.

IV.5 Méthode des moindres carrés

La méthode des moindres carrés permet alors de minimiser l'impact des erreurs, dans ce cas les erreurs dans les pseudodistances, en ajoutant de l'information dans le processus de mesure, ça veut dire plus de satellites. L'objectif est d'approximer les coordonnées du récepteur (x_r, y_r, z_r) et le décalage d'horloge (dt_r) avec un ensemble de n mesures de pseudodistance et de n positions de satellites, l'équation de pseudodistance peut s'écrire comme :

$$P_n = \sqrt{(x_{satn} - x_0)^2 + (y_{satn} - y_0)^2 + (z_{satn} - z_0)^2} + c * dt_r 0 \tag{6}$$

La quantité minimale de satellites visibles pour le calcul de position est de 4 comme nous avons déjà parlé, mais comme la pseudodistance contient plusieurs sources d'erreurs, c'est mieux d'utiliser tous les satellites visibles pour le calcul de position. Donc nous ajoutons plus d'équations au système et pour résoudre le système nous utilisons un linéarisation itérative, en commençant par une position approximative (x_0, y_0, z_0, dt_{r0}) du récepteur.

$$\rho_n - P_n = \frac{x_0 - x_{satn}}{P_n} dx + \frac{y_0 - y_{satn}}{P_n} dy + \frac{z_0 - z_{satn}}{P_n} dz + c * dt_r \tag{7}$$

avec :

- P_n : la pseudodistance calculée
- ρ_n : la pseudodistance mesurée
- dx : $r_x - x_0$
- dy : $r_y - y_0$
- dz : $r_z - z_0$

Cette équation peut être représentée comme la matrice suivante :

$$\begin{bmatrix} \rho_1 - P_1 \\ \vdots \\ \rho_n - P_n \end{bmatrix} = \begin{pmatrix} \frac{x_0 - x_{sat1}}{P_1} & \frac{y_0 - y_{sat1}}{P_1} & \frac{z_0 - z_{sat1}}{P_1} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \frac{x_0 - x_{satn}}{P_n} & \frac{y_0 - y_{satn}}{P_n} & \frac{z_0 - z_{satn}}{P_n} & 1 \end{pmatrix} \begin{bmatrix} dx \\ dy \\ dz \\ c * d(dt_r) \end{bmatrix} \tag{8}$$

Pour plus de 4 satellites (n > 4), nous avons un système surdéterminé qui peut être résolu avec la méthode des moindres carrés. Après avoir résolu l'équation, nous avons une mise à jour des coordonnées du récepteur qui est égale à :

$$\begin{bmatrix} x_r \\ y_r \\ z_r \\ dt_r \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ dt_{r0} \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ dz \\ dt_r \end{bmatrix} \tag{9}$$

Dans une étape consécutive, le système d'équations peut être linéarisé de nouveau en utilisant la nouvelle position du récepteur (x_0, y_0, z_0), et nous continuons jusqu'à ce que la différence entre deux itérations consécutives soit plus petit que le seuil fixé.

V Projet de stage

V.1 Contexte

Dans les smartphones Android la géolocalisation est possible grâce au chipset GNSS embarqué, c'est lui qui reçoit tous les données brutes diffusées par le satellite, pour après avec une couche logicielle faite par Google, utiliser ces données pour le calcul de position. L'accès aux données brutes n'était pas possible avant la version 7 (Nougat) d'Android, donc avant nous étions limités à utiliser la position donnée et calculée directement par Google, mais nous ne savions pas comment cette position avait été calculée et quelle était sa précision. Cette position calculée par Google n'avait aucune valeur pour la recherche et les laboratoires de géolocalisation par exemple.

Maintenant avec la disponibilité des données brutes GNSS, il est possible de créer des nouvelles applications sur Android en implémentant des propres algorithmes de géolocalisation, et potentiellement utiliser le smartphone comme outil d'éducation et recherche pour la géolocalisation. Malgré ces avantages et opportunités, l'utilisation des données brutes GNSS sur le smartphone n'est pas simple, car d'abord les experts en GNSS ont besoin d'assistance pour comprendre comment les données brutes sont structurés sur la plateforme Android, puisque les formats standard pour la géolocalisation comme RINEX ou NMEA, ne sont pas disponibles sur la plateforme Android. Et deuxièmement, les développeurs Java qui connaissent la plateforme Android ne sont pas trop familiarisés avec les détails du positionnement GNSS.

Donc il m'a été demandé durant ce stage de réaliser une application sur Android qui à partir des données brutes GNSS peut calculer sa position, afin de collaborer à ce nouveau sujet sur la géolocalisation, en facilitant la compréhension sur comment les données brutes sont donnés sur Android et rendre ces données dans un format plus convivial pour les chercheurs GNSS, pour que les chercheurs puissent utiliser le smartphone comme outil de recherche dans la géolocalisation.

V.2 Cahier des charges application Android

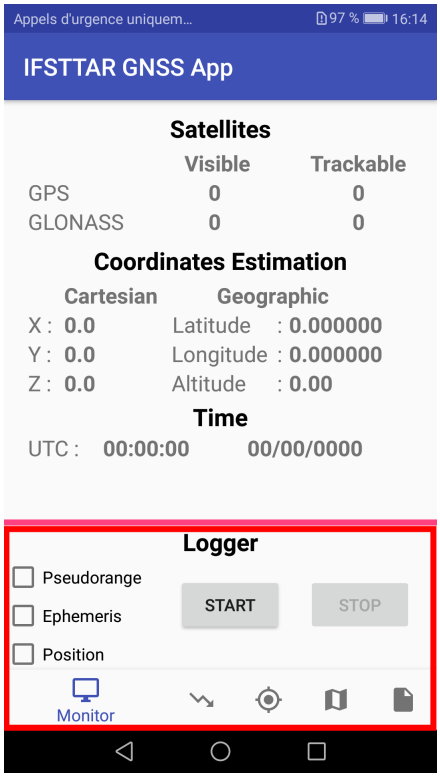
V.2.1 Objectifs

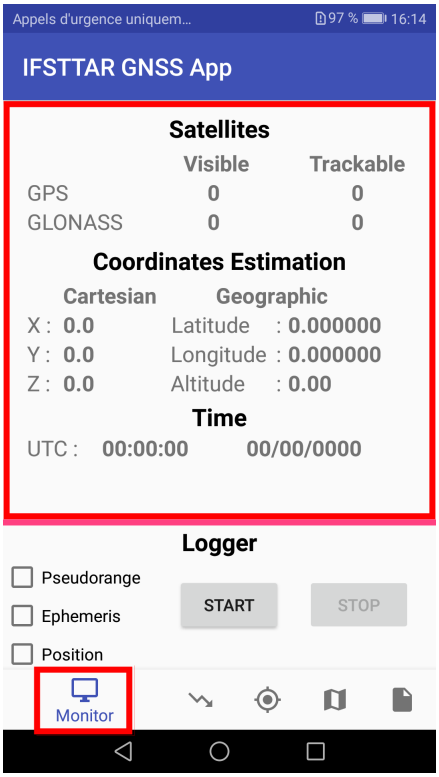
L'application doit récupérer les données brutes diffusées par le satellite pour après réaliser des calculs de positionnement GNSS, et finalement afficher les coordonnées géographies sur l'application, soit afficher les 3 composants géographies (latitude, longitude, et altitude) ou soit montrer un point sur la carte. Cette application doit aussi faciliter les données de navigation et les sauvegardes dans un fichier avec un format qui ressemble aux formats standard de navigation, ainsi ce fichier pourra après être utilisé pour faire du post-traitement.

V.2.2 Spécificités techniques

- Android version 7.0
- 6 différents écrans :

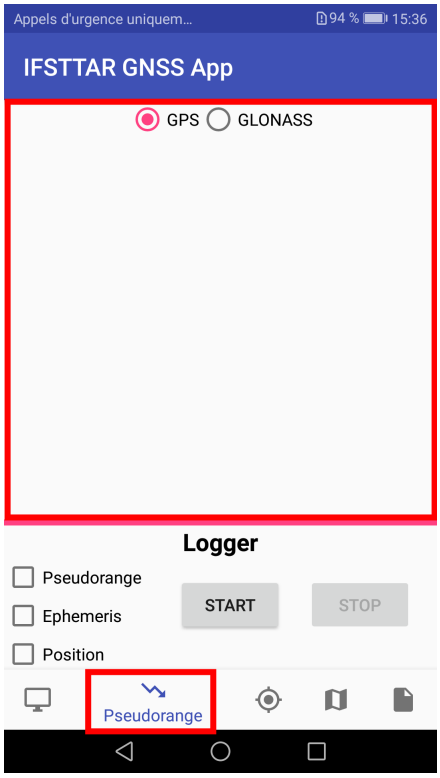
➤ Activité principale : cet écran contient l'interface pour l'enregistrement des données de navigation et la barre de navigation entre écrans.

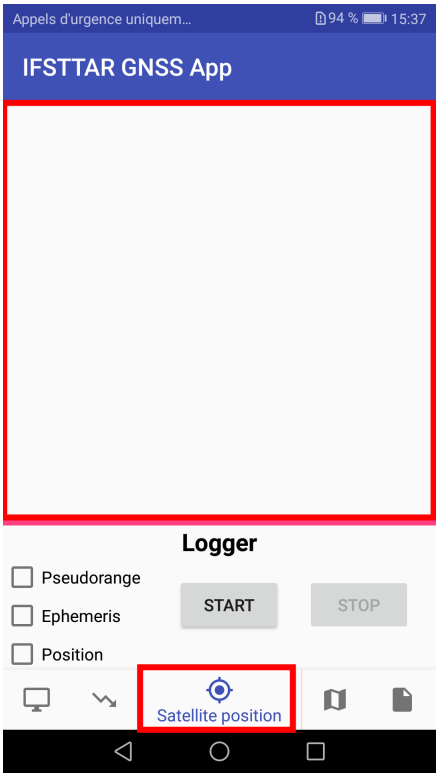




➤ Monitor : cet écran contient les coordonnées terrestres avec le temps de mesure, ainsi que la quantité de satellites visibles.

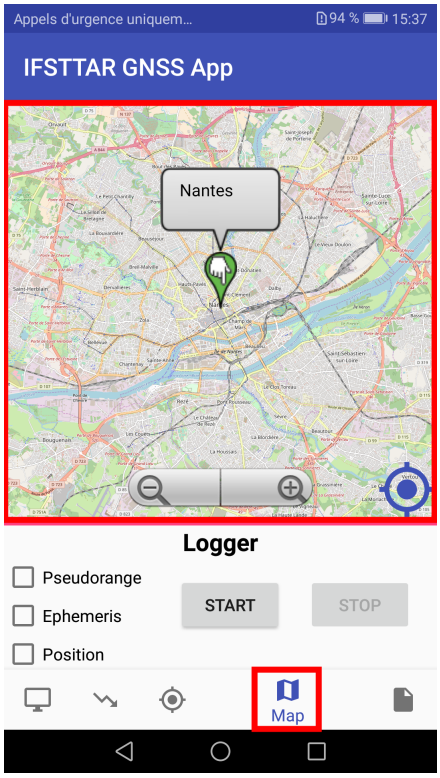
➤ Pseudorange : cet écran contient la pseudo-distance vers chacun des satellites.

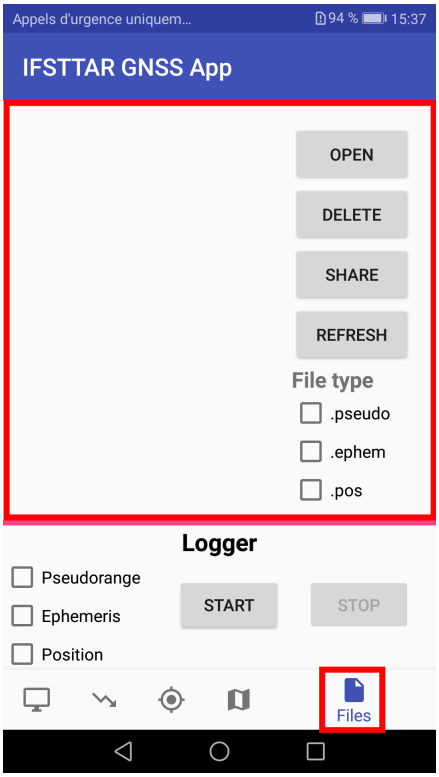




➤ Satellite Position : cet écran contiendra les coordonnées des satellites visibles, cet écran n'a pas encore été codé, ceci sera fait pour la suite du stage. L'idée est de montrer la position du satellite dans le ciel, avec un *sky plot*.

➤ Map : cet écran contient une carte avec la position calculée, la carte est fournie par *Open Street Map*.





➤ Files : cet écran contient l'interface pour l'administration des fichiers avec les données de navigation, cet écran n'a pas non plus encore été codé, mais l'idée est de faciliter l'accès aux fichiers générés par le *logger*.

- Permissions :
  - ➤ ACCESS_FINE_LOCATION : utilisé pour permettre l'accès aux données brutes du chipset GNSS

- INTERNET : utilisé pour télécharge la carte
- WRITE_EXTERNAL_STORAGE : utilisé pour l'enregistrement des fichiers de navigation

V.3 Matériels et logicielles utilisés

V.3.1 Huawei P10

FIGURE 6 – Smartphone Huawei P10

Le smartphone utilisé pour tester l'application a été le Huawei P10, ce smartphone peut capter des signaux GPS, GLONASS, GALILEO, BeiDou et QZSS, mais pour les objectifs de ce stage, j'ai utilisé seulement la constellation GPS pour le calcul de position.

V.3.2 Android Studio

Pour le développement de l'application j'ai utilisé Android Studio qui est la plateforme officielle de Google pour développer des applications mobiles sur Android. En concret j'ai travaillé avec la version 3.1.2 d'Android Studio avec l'API 24 d'Android, c'est à partir de cet API que l'accès aux données brutes est disponible.

Dans les deux diagrammes suivants nous pouvons voir les fonctionnalités de l'API 23 et 24, nous pouvons voir que toutes les fonctionnalités de l'API 23 sont conservées dans l'API 24 et en plus nous avons les nouvelles fonctionnalités de l'API 24. Maintenant avec les vieilles et nouvelles fonctionna-

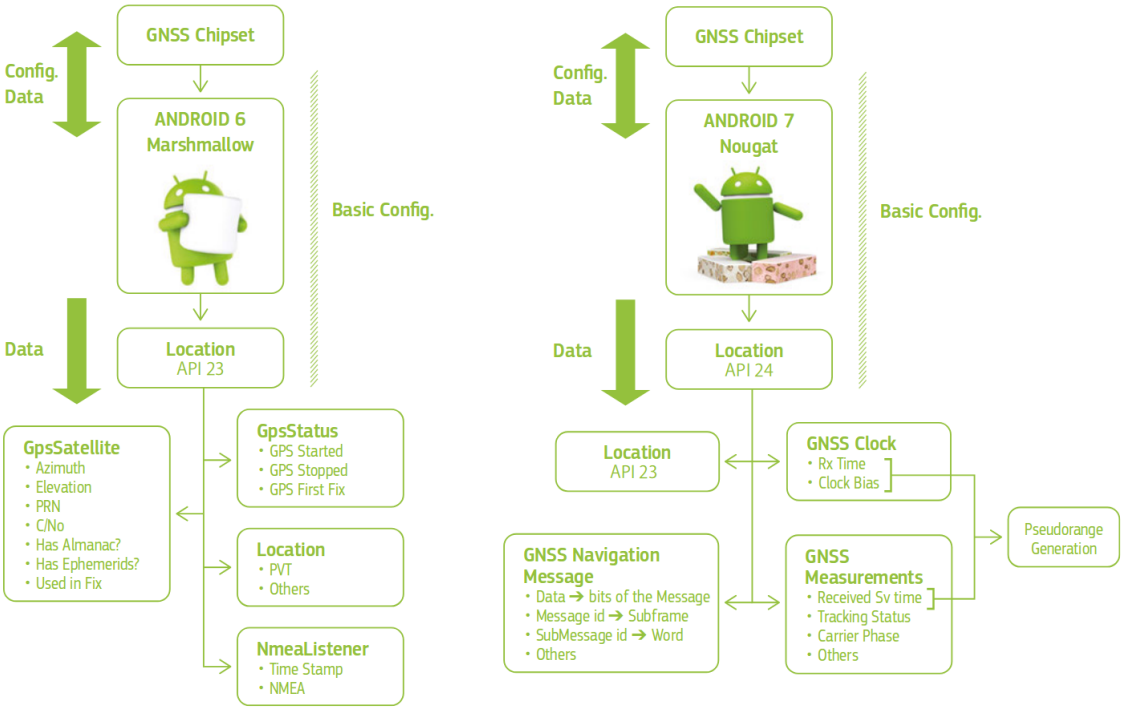


FIGURE 7 – Location API en Android API niveau 23 à gauche et 24/25/26 à droite

lités nous pouvons soit récupérer la position directement calculée par l'API, soit calculer la position nous-mêmes avec les données brutes. Toutes les classes utilisées pour le développement du projet sont documentées sur le site web *Android Developers* dans la partie *android.location*.

V.3.3 Git

L'outil Git est un logiciel de versions décentralisé, il s'agit du logiciel de gestion de versions. Pour ce stage j'ai utilisé l'outil Git pour versionner le code et la plateforme GitLab de l'entreprise pour le sauvegarder dans le nuage et le partager avec l'équipe GEOLOC.

V.3.4 GNSS logger

L'application Android *GNSS logger* créé par Google est un enregistreur de données brutes de navigation, cette application ensemble avec l'application de bureau *GNSS analysis* sous Ubuntu, ont été utilisées pour la validation des résultats du projet avec l'Huawei P10.

V.3.5 GNSS analysis app

Ce logiciel est un outil d'analyse des données GNSS, ce logiciel utilise les données enregistrées par l'application mobile GNSS loger pour calculer plusieurs informations, comme la pseudodistance et la position du satellite, donc j'ai utilisé ce logiciel pour valider mes calculs avec les calculs faits par Google.

V.4 Calcul de pseudodistance

Pour le calcul de pseudodistance nous utilisons les classes *GnssMeasurement*, *GnssClock*, *GnssMeasurementEvent* et *GnssMeasurementEvent.Callback*, mais d'abord il faut accéder au service de location du système avec en utilisant la classe *LocationManager* et en ajoutant la permission *ACCESS_FINE_LOCATION* dans le fichier *AndroidManifest.xml* de l'application.

AndroidManifest.xml :

```
<uses-permission android:name="android.permission.
   ACCESS_FINE_LOCATION"/>
```

MainActivity.java :

```
LocationManager mLocationManager = (LocationManager)
   getApplicationContext().getSystemService(Context.LOCATION_SERVICE)
   ;
```

Pour commencer à recevoir les données brutes des mesures du chipset GNSS il faut enregistrer un *Callback* du type *GnssMeasurementEvent.Callback* dans notre *LocationManager*, avec l'interface *registerGnssMeasurementsCallback*. Si la permission de location n'a pas été demandée et acceptée par l'utilisateur, nous ne pouvons pas enregistrer le *Callback*.

```
mGnssMeasurementsEventCallback = new GnssMeasurementsEvent.Callback()
    {
        @Override
        public void onGnssMeasurementsReceived(GnssMeasurementsEvent
           eventArgs) {
                super.onGnssMeasurementsReceived(eventArgs);
                GnssMeasurementsEvent mGnssMeasurementsEvent =
                   eventArgs;
                GnssClock mGnssClock = mGnssMeasurement.getClock();
                Vector<GnssMeasurement> mGnssMeasurements = new
                   Vector<>(mGnssMeasurementsEvent.getMeasurements())
                   ;
        }
};
mLocationManager.registerGnssMeasurementsCallback(
   mGnssMeasurementsEventCallback);
```

L'objet "evenArgs" est un objet du type *GnssMeasurementEvent*, cet objet contient une collection d'objets du type *GnssMeasurement* qui représente tous les satellites visibles, et il contient aussi un objet *GnssClock* qui contient les mesures du temps du récepteur. Une fois que nous avons les données brutes de la mesure, nous passons au calcul de pseudodistance.

Avant de continuer il faut savoir que le chipset GNSS du smartphone utilise le temps GPS, donc si nous voulons calculer la pseudodistance pour une autre constellation il faut faire la conversion du temps selon le tableau suivant :

Conversion	Formule
$GPST \Leftrightarrow TAI$	$TAI = GPST + 19s$
$GST \Leftrightarrow TAI$	$TAI = GST + 19s$
$GLONASST \Leftrightarrow TAI$	$TAI = GLONASST - 3h + leapsecond_{UTC-TAI}$
$BDT \Leftrightarrow TAI$	$TAI = BDT + 33s$
$UTC \Leftrightarrow TAI$	$UTC = TAI - leapsecond_{UTC-TAI}$

FIGURE 8 – Relation entre les différents temps de référence

Le temps de transmission t_{tx} est obtenu à partir de l'objet *GnssMeasurement* et le temps de référence dépend de la constellation à laquelle le satellite appartient.

```
ttx = mGnssMeasurement.getReceivedSvTimeNanos();
```

Le temps de réception t_{rx} est calculé à partir de l'objet *GnssClock*, pour ceci nous calculons d'abord le vrai temps GNSS qui est aussi le vrai temps GPS depuis le 6 janvier, 1980.

```
trxGnss = mGnssClock.getTimeNanos() + mGnssMeasurement.
   getTimeOffsetNanos() - (mGnssClock.getFullBiasNanos() + mGnssClock
   .getBiasNanos());
```

Maintenant que nous avons le vrai temps GNSS nous devons le convertir aux différents temps de référence, dans le cas du GPS, il utilise le TOW qui sont les secondes passés à chaque semaine en commençant par le dimanche à 0h00, alors nous devons calculer les secondes déjà passées pour la semaine actuelle. Donc nous soustrayons les secondes des semaines qui sont passées depuis le début du temps GPS.

```
trx = trxGnss - Math.floor((-mGnssClock.getFullBiasNanos())/
   numberNanoSecondsWeek)*numberNanoSecondsWeek;
```

Où *numberNanoSecondsWeek* est la quantité de secondes dans une semaine, et finalement la pseudodistance est calculée simplement en utilisant la vitesse de la lumière.

```
pseudoRange = (((trx - ttx) / numberNanoSecondsSecond) * c);
```

Où *numberNanoSecondsSecond* est la quantité de nanosecondes dans un seconde, tous ces calculs ont été pris du *White paper : Using GNSS Raw Measurements on Android Devices* fait par la GSA. La classe *SatellitePseudorange* du projet contient tous les calculs ici mentionnés.

V.5 Calcul de position du satellite

Pour la réception des messages de navigation il faut enregistrer dans notre *LocationManager* un nouveau *Callback* appelé *GnssNavigationMessage.Callback* qui retourne un objet de type *GnssNavigationMessage* qui contient le dernier message reçu par le chipset GNSS. Et nous enregistrons ce *Callback* de la même manière que pour la pseudodistance.

```
        mGnssNavigationMessageCallback = new GnssNavigationMessage.
           Callback() {
```

```
            @Override
            public void onGnssNavigationMessageReceived(
               GnssNavigationMessage event) {
                super.onGnssNavigationMessageReceived(event);
                GnssNavigationMessage mGnssNavigationMessage = event;
            }
        };
        mLocationManager.registerGnssNavigationMessageCallback(
           mGnssNavigationMessageCallback);
```

Les messages de navigation ne sont pas complètement envoyés d'un seul coup, ils sont divisés en différents *subframes* et le nombre des *subframes* dépend de la constellation. Pour la constellation GPS il y a 5 *subframes* ce qui fait un *frame*, et tout le message contient 5 *frames* ce qui prend 12m30s pour être diffusé par le satellite, le diagramme suivant décrit les données contenues dans chacune des *subframes* GPS.

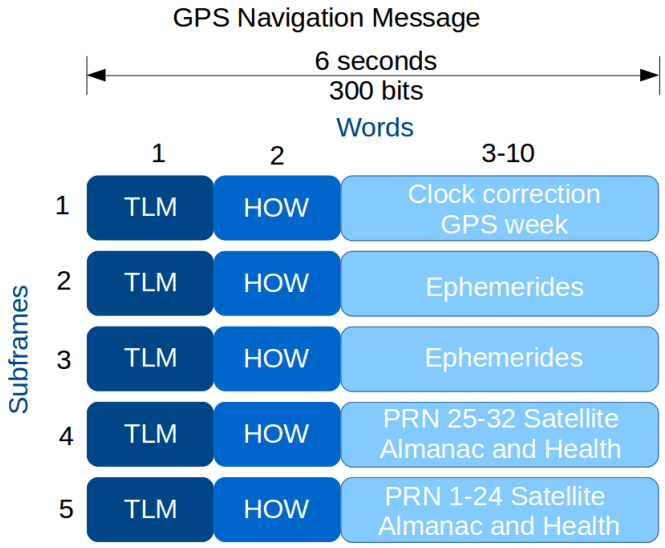


FIGURE 9 – Contenu du message de navigation GPS

V.5.1 Décodage de message de navigation

Malgré le temps que prends le message de navigation pour être diffusé complètement, nous n'avons pas besoin de tout le message, nous voulons juste les éphémérides (*subframe* 2 et 3) et les corrections du temps (*subframe* 1), ces *subframes* sont diffusés chaque 30s. Pour savoir le numéro de *subframe* et obtenir le contenu de message on utilise la méthode, *getSubmessageId* et *getData* de la classe *GnssNavigationMessage*.

```
int subFrameId = mGnssNavigationMessage.getSubMessageId();
byte[] data = mGnssNavigationMessage.getData();
```

Pour la reconstruction du message de navigation, nous enregistrons tous les messages qui arrivent au chipset GNSS, et nous attendons jusqu'à ce que les 3 *subframes* que nous cherchons arrivent. Une fois que nous avons les 3 *subframes* du message de navigation, nous pouvons passer au décodage en utilisant le ICD de GPS. Pour pouvoir arriver aux objectifs du projet, le message de navigation GPS a été le seul décodé puisque le temps qui a pris le décodage et le calcul des coordonnées du satellite a été presque d'un mois, donc nous avons décidé de continuer avec la suite à la place de décoder tous les messages de navigation pour toutes les constellations.

Le tableau ci-dessous montre les noms des données, la position dans le *subframe* et la largeur en bits, pour les trois *subframes* qui nous intéresse.

Donnée	**subframe**	**word**	**bits**
PRN	*	1	9-14 (6 bits)
Epoch	*	2	1-17 (17 bits)
Week number	1	3	1-10 (10 bits)
toc	1	8	9-24 (16 bits)
af2	1	9	1-8 (8 bits)
af1	1	9	9-24 (16 bits)
af0	1	10	1-22 (22 bits)
Crs	2	3	9-24 (16 bits)
IODE	2	3	1-8 (8 bits)
Δn	2	4	1-16 (16 bits)
M_0	2	4-5	17-24, 1-24 (32 bits)
Cuc	2	6	1-16 (16 bits)
e	2	6-7	17-24, 1-24 (32 bits)
Cus	2	8	1-16 (16 bits)
$\sqrt{A}$	2	8-9	17-24, 1-24 (32 bits)
toe	2	10	1-16 (16 bits)
AODO	2	10	18-22 (5 bits)
Cic	3	3	1-16 (16 bits)
Ω_0	3	3-4	17-24, 1-24 (32 bits)
Cis	3	5	1-16 (16 bits)
i_0	3	5-6	17-24, 1-24 (32 bits)
Crc	3	7	1-16 (16 bits)
ω	3	7-8	17-24, 1-24 (32 bits)
$\dot{\Omega}$	3	9	1-24 (24 bits)
IODE	3	10	1-8 (8 bits)
IDOT	3	10	9-22 (14 bits)

* : tous les *subframes* contient ce donnée

FIGURE 10 – Contenu des *subframes* du message de navigation GPS

Ces données correspondent aux éléments de l'orbite du satellite, appelés aussi *Keplerian Elements*, la méthode *getData()* de la classe *GnssNavigationMessage* nous retourne un tableau de 40 octet, ces 40 octet sont divisé en *words* de 4, ce qui fait 10 *words* de 4 octet ou 32 bits. Chaque *word* du message

de navigation contient 30 bits seulement, donc nous enlevons les deux premières bits de chaque *word*, comme ça nous pouvons prendre les bits de chaque donnée en appliquant le tableau au-dessus.

Finalement pour décoder chaque chaîne de bits, nous utilisons le tableau suivant, qui contient le facteur d'échelle de la donnée, si la donnée est signée ou non signée avec l'opération complément à deux avec le signe (+ ou -) occupant le bit de poids fort, et l'unité de la donnée.

Donnée	**Signé**	**Facteur d'échelle**	**Unité**
PRN	non	1	sans unité
Epoch	non	6	secondes
Week Number	non	1	semaines
toc	non	2^4	secondes
af2	oui	2^{-55}	sec/sec^2
af1	oui	2^{-43}	sec/sec
af0	oui	2^{-31}	secondes
Crs	oui	2^{-5}	mètres
IODE	non	1	sans unité
Δn	oui	2^{-43}	demi-cercles/sec
M_0	oui	2^{-31}	demi-cercles
Cuc	oui	2^{-29}	radians
e	non	2^{-33}	sans unité
Cus	oui	2^{-29}	radians
$\sqrt{A}$	non	2^{-19}	$\sqrt{\text{mètres}}$
toe	non	2^4	secondes
AODO	non	900	secondes
Cic	oui	2^{-29}	radians
Ω_0	oui	2^{-31}	demi-cercles
Cis	oui	2^{-29}	radians
i_0	oui	2^{-31}	demi-cercles
Crc	oui	2^{-5}	mètres
ω	oui	2^{-31}	demi-cercles
$\dot{\Omega}$	oui	2^{-43}	demi-cercles/sec
IDOT	oui	2^{-43}	demi-cercles/sec

FIGURE 11 – Décodage des données de navigation GPS

Une fois que nous décodons les éphémérides et les corrections du temps, nous pouvons maintenant passer au calcul de position du satellite, qui est aussi documenté dans le ICD. D'abord pour calculer la position du satellite au moment de la mesure de pseudodistance, nous avons besoin du temps de transmission de la classe *GnssMeasurement*, après nous corrigeons ce temps avec les coefficients qui le satellite diffusé dans le message de navigation et finalement une correction relativiste.

$$t = t_{sv} - \Delta t_{sv} \tag{10}$$

$$\Delta t_{sv} = a_{f0} + a_{f1}(t - toc) + a_{f2}(t - toc)^2 + \Delta t_r \tag{11}$$

$$\Delta t_r = Fe\sqrt{A}\sin E_k \tag{12}$$

Avec :

— t_{sv} : le temps satellite de transmission au moment de la mesure

— Δt_{sv} : l'erreur d'horloge du satellite

— toc : temps de référence des coefficients d'horloge du satellite (en secondes comptées depuis le début de la semaine GPS)

— a_{f0} : décalage de l'horloge

— a_{f1} : dérive de l'horloge

— a_{f2} :"accélération" de l'horloge

— Δt_r : l'erreur relativiste

— F : -4.442807633e^{-10} s/m$^{1/2}$

— e : excentricité de l'orbit du satellite

— $\sqrt{A}$: racine carrée du demi-grand axe

— E_k : Anomalie excentrique

La valeur de t dans l'équation 11 nous l'approximons avec t_{sv}. La valeur de t doit tenir compte des croisements de début et de fin de semaine, donc si $t - t_{oc}$ est plus grand que 302,400 secondes, nous soustrayons 604,800 secondes de t. Et si $t - t_{oc}$ est plus petit que -302,400 secondes, nous additions 604,800 à t.

Une fois que nous calculons le temps satellite corrigé, nous pouvons l'utiliser pour calculer sa position. Avec les calculs suivants nous allons obtenir les coordonnées cartésiennes du satellite.

Nom du paramètre	valeur
WGS 84 valeur de la constante gravitationnelle de la Terre pour l'utilisateur GPS	$\mu = 3.986005e10^{14}\text{mètres}^3/\text{sec}^2$
WGS 84 valeur du taux de rotation de la terre	$\dot{\Omega}_c = 7.2921151467e10^{-5}\text{rad/sec}$
Demi-grand axe	$A = (\sqrt{A})^2$
Mouvement moyen calculé	$n_0 = \sqrt{\frac{\mu}{A^3}}$
Temps depuis l'époque de référence des éphémérides	$t_k = t - t_{oe}$
Mouvement moyen corrigé	$n = n_0 + \Delta n$
Anomalie moyenne	$M_k = M_0 + nt_k$
Équation de Kepler pour l'anomalie excentrique (peut être résolu par itération) (radians)	$M_k = E_k - e\sin E_k$
Vrai anomalie	$v_k = \tan^{-1}\frac{\sqrt{1-e^2}\sin E_k/(1-e\cos E_k)}{(\cos E_k - e)/(1-e\cos E_k)}$
Argument de latitude	$\Phi_k = v_k + \omega$
Perturbations du seconde Harmonique : -Argument de la correction de latitude -correction du rayon -correction d'inclination	 $\delta u_k = c_{us}\sin 2\Phi_k + c_{uc}\cos 2\Phi_k$ $\delta r_k = c_{rs}\sin 2\Phi_k + c_{rc}\cos 2\Phi_k$ $\delta i_k = c_{is}\sin 2\Phi_k + c_{ic}\cos 2\Phi_k$
Argument de latitude corrigé	$u_k = \Phi_k + \delta u_k$
Rayon corrigé	$r_k = A(1 - e\cos E_k + \delta r_k)$
Inclination corrigé	$i_k = i_0 + \delta i_k + (IDOT)t_k$
Position dans le plane orbital	$x'_k = r_k\cos u_k$ $y'_k = r_k\sin u_k$
Longitude corrigée du nœud ascendant	$\Omega_k = \Omega_0 + (\dot{\Omega} - \dot{\Omega}_e)t_k - \dot{\Omega}_e t_{oe}$
Coordonnées cartésiennes	$x_k = x'_k\cos\Omega_k - y'_k\cos i_k\sin\Omega_k$ $y_k = x'_k\sin\Omega_k - y'_k\cos i_k\cos\Omega_k$ $z_k = y'_k\sin i_k$

FIGURE 12 – Constants et variables pour le calcul de position du satellite

Tous ces calculs ont été programmés en Java en utilisant les formules ci montrées et des exemples du code MatLab que mes collègues du laboratoire avaient déjà programmé. La classe *Navigation-Message* du projet divise les données en *words* et décode des information de base du satellite comme l'identifiant du satellite et l'époque. La classe *EphemerisGPS* utilise les *words* crées par la classe *NavigationMessage* pour décoder les éphémérides et les corrections du temps et finalement la classe *SatellitePositionGPS* utilise les éphémérides et les corrections du temps pour calculer la position du satellite.

V.6 Estimation de la position

Pour le calcul de position du récepteur une fois que nous avons la quantité minimal de satellites visibles, nous utilisons la méthode des moindres carrés expliquée au début du rapport. Comme la méthode l'explique il faut mettre un point approximatif de départ, comme par exemple Paris.

Pour faire plus simple les opérations entre matrices, nous utilisons la classe SimpleMatrix par *Efficient Java Matrix Library* (EJML), et pour pouvoir l'utiliser avec Android studio nous ajoutons les lignes suivantes au fichier *build.gradle* (app).

```
compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
}
```

Tout le calcul de position a été réalisé dans la classe *GNSSPosition* du projet.

V.7 Résultats

V.7.1 Pseudodistance

L'écran appelée *Pseudorange* montre la pseudodistance des satellites visibles, nous pouvons constater avec les captures d'écrans que c'est bien une distance d'environ 20,000 km. Pour le moment nous n'avons la pseudodistance que pour les satellites GPS et GLONASS et quand nous visualisons le satellite mais il n'y a pas de mesure de pseudodistance, ceci veux dire que l'état de suivi n'est pas optimal.

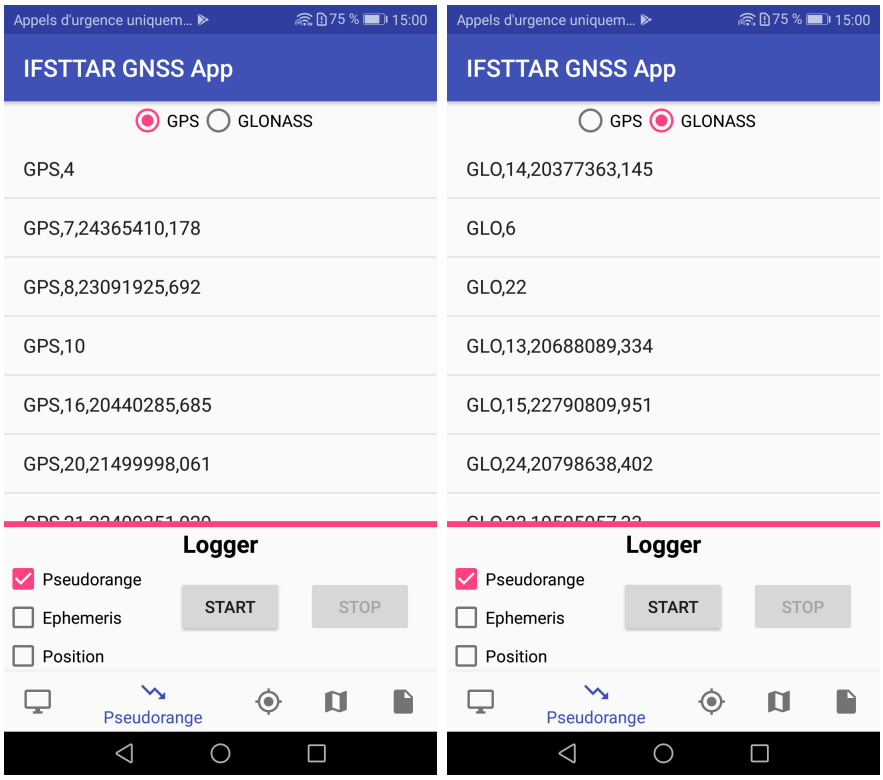


FIGURE 13 – Test de pseudodistance avec le Huawei P10

V.7.2 Coordonnées satellite

Les positions des satellites obtenus avec l'application ne sont pas encore affichés dans l'application, donc pour valider les résultat, les positions ont été récupérer en utilisant la console *logcat*.

```
TOW : 394852.99967872
SvId : 26
Coordinates : -3.0285,10.3113,2.0256E7

TOW : 394852.99967872
SvId : 10
Coordinates : 23.8882,27.1135,2.0280E7

TOW : 394852.99967872
SvId : 16
Coordinates : 19.5370,0.4727,2.0339E7

TOW : 394852.99967872
SvId : 15
Coordinates : 43.8497,110.3949,1.9935E7
```

D'après cet affichage nous avons 4 satellites (GPS) à une époque 394853 (TOW), nous convertissons TOW en heurs, minutes et secondes du jour courant, et pour la validation de la position de chacun des satellites nous utilisons la page *in-the-sky.org*.

Satellite	UTC	CEST	Latitude	Longitude	Altitude
26	13h40m53s	15h40m53s	-3.0285	10.3113	20256 km
10			23.8882	27.1135	20280 km
16			19.5370	0.4727	20339 km
15			43.8497	110.3949	19935 km

FIGURE 14 – Tableau des positions des satellites calculées avec l'application

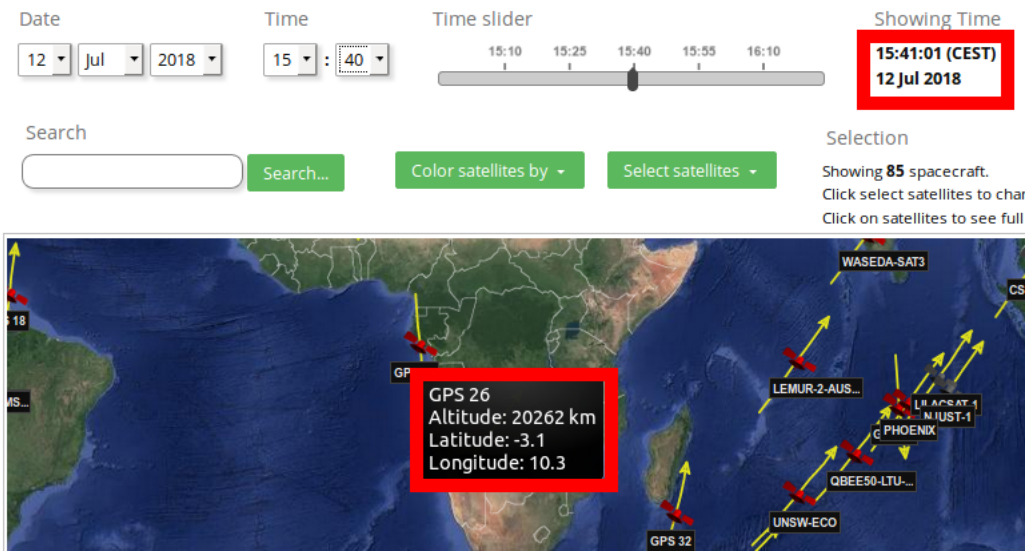


FIGURE 15 – Position GPS 26

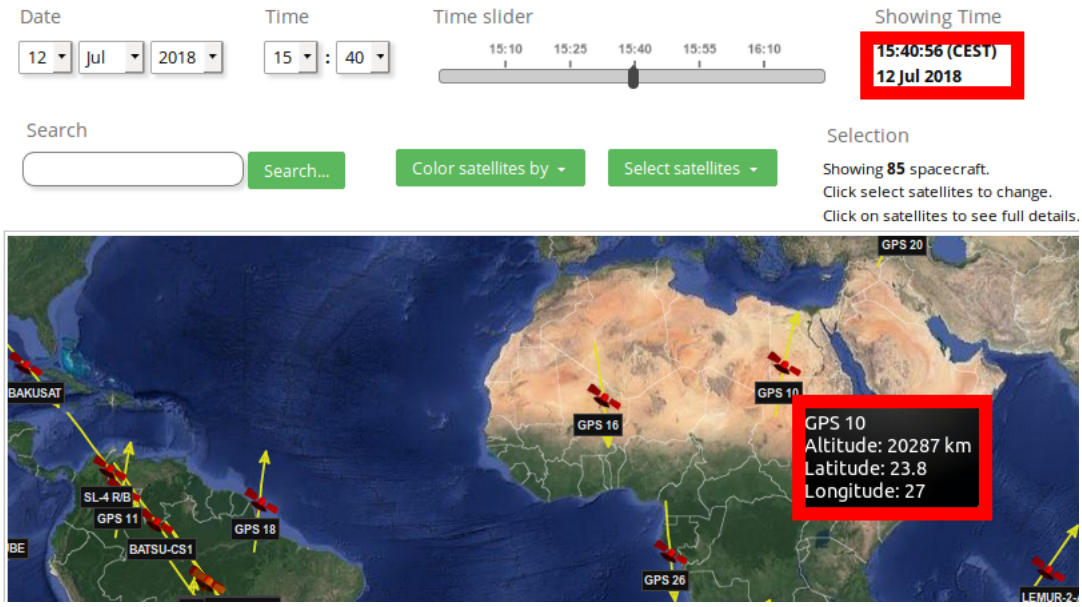


FIGURE 16 – Position GPS 10

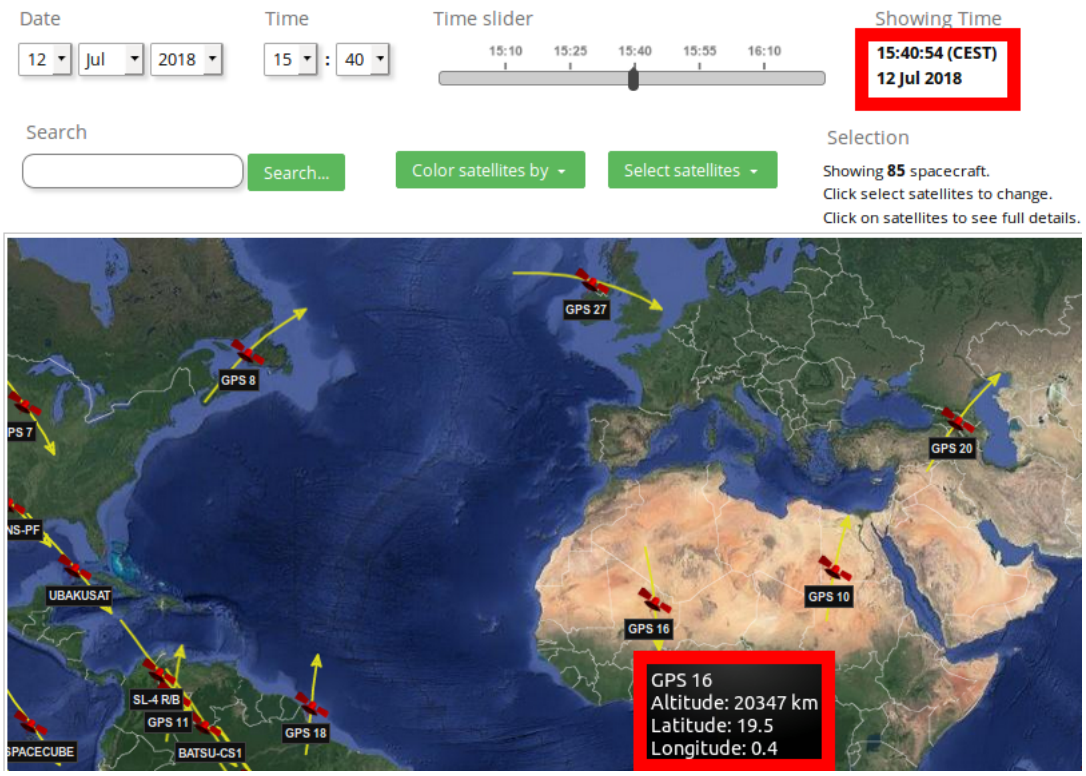


FIGURE 17 – Position GPS 16

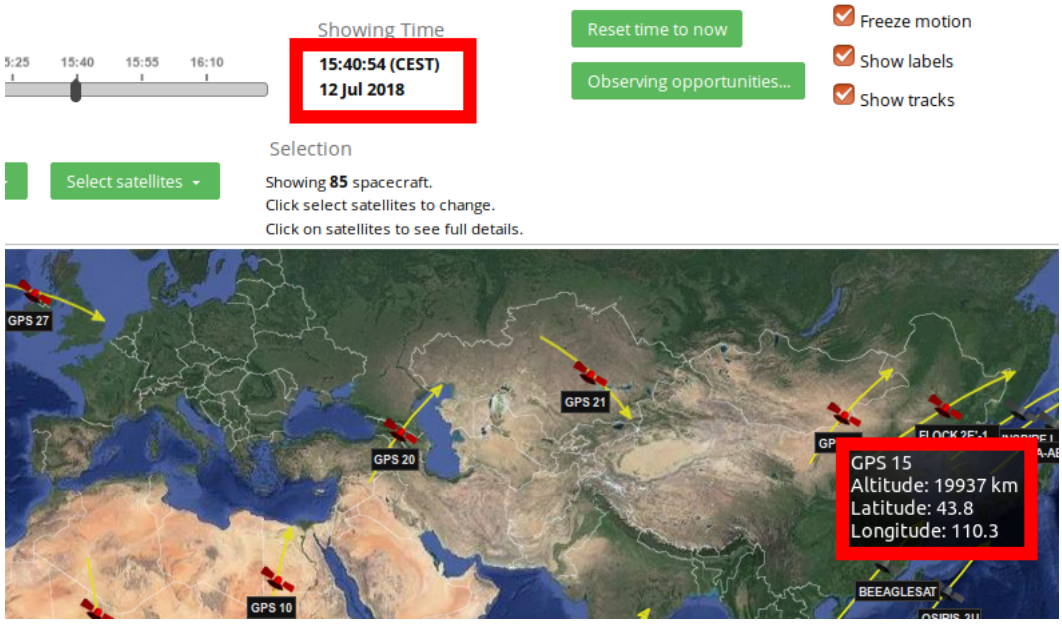


FIGURE 18 – Position GPS 15

Nous observons que nous arrivons à trouver la même position, la différence des décimaux dans quelques valeurs est du à cause de la différence des secondes entre l'application et la page sur internet. La page sur internet ne donne pas la position tous les secondes, et il y a des sautes de 3 secondes dans les mesures.

V.7.3 Position

Pour valider la position obtenu il n'y a pas eu des test de précision, ça sera la suite pour les prochains stagiaires, mais nous pouvons observer avec la carte de Google maps et la carte sur mon application que la différence n'est pas supérieur à 10m.

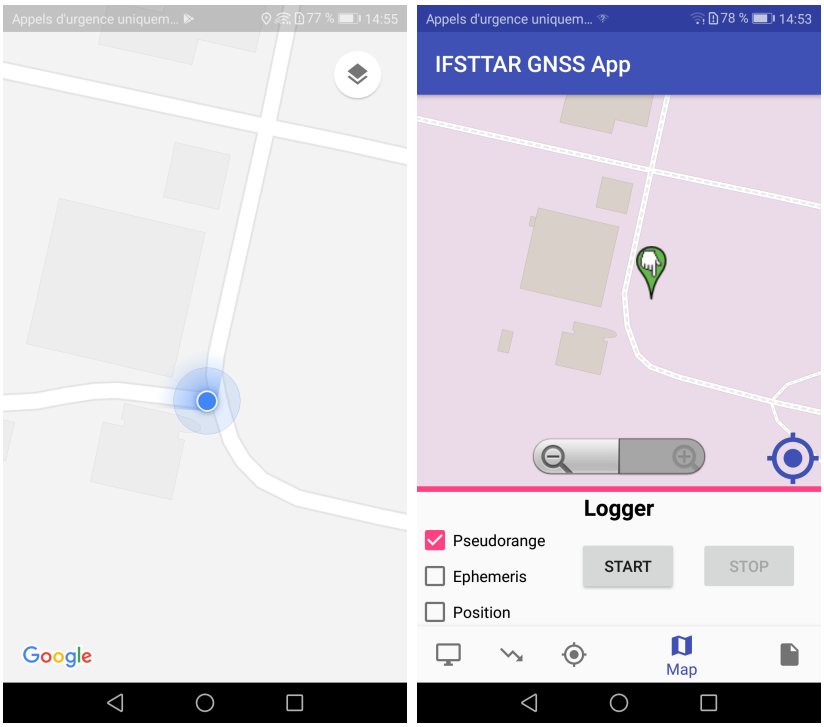


FIGURE 19 – Comparaison 1 entre Google maps et l'application

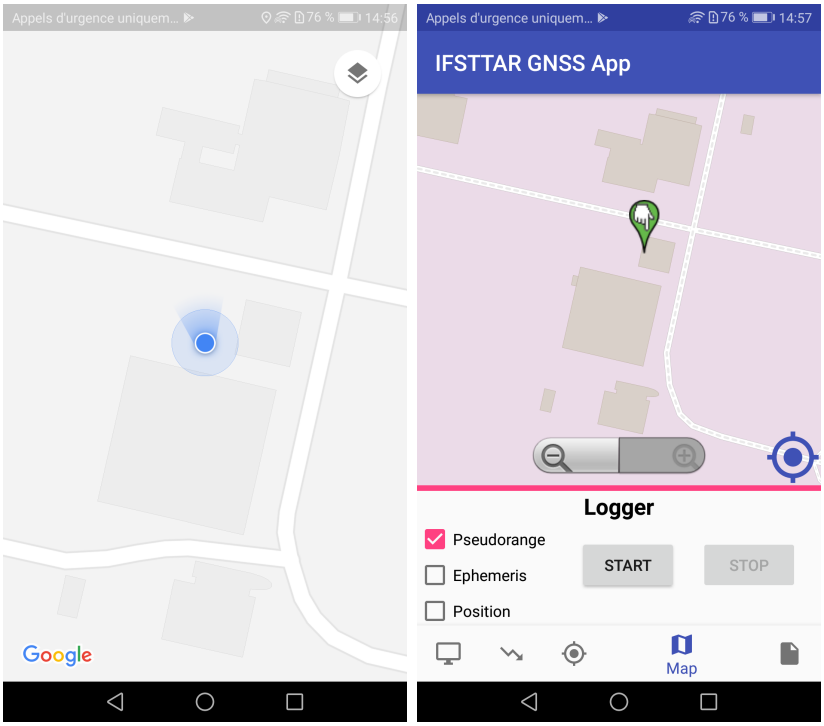


FIGURE 20 – Comparaison 2 entre Google maps et l'application

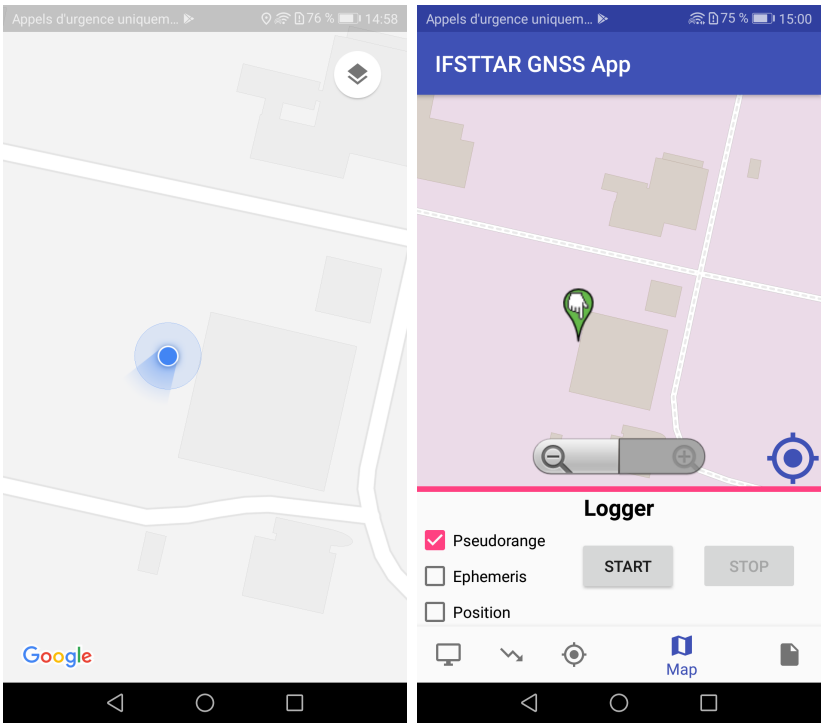


FIGURE 21 – Comparaison 3 entre Google maps et l'application

VI Conclusion

VI.1 Conclusion sur le projet

L'objectif principal du calcul de positionnement a été atteint, il faudra maintenant améliorer le calcul de positionnement, et faire des tests de comparaison avec d'autres dispositifs de navigation plus précis que le smartphone.

La plus grande difficulté durant ce projet a été le décodage des *subframes* du message de navigation, car nous n'avions pas anticipé que Google avait fait le minimum pour la réception des messages de navigation.

Les résultats de position ont été mieux que les attendus, car la formule pour le calcul de pseudo-distance n'est pas complète, il y a encore des corrections de atmosphérique qui n'ont pas été pris en compte, mais ça sera la suite pour les prochains stages qui travaille sur ce sujet.

VI.2 Conclusion personnelle

Ce stage m'a fait beaucoup appris sur le milieu de la recherche et la géolocalisation, concrètement la géolocalisation par satellite. Je n'attendais pas que la géolocalisation fût tellement complexe et intéressant, même si je ne suis pas arrivé à comprendre tous les aspects techniques et mathématiques. J'ai beaucoup aimé mettre en œuvre les compétences en programmation acquises à l'IUT, pour le développement d'une application pour la géolocalisation.

Maintenant je pense que travailler dans la recherche est une très bonne option si je veux apprendre des nouvelles choses, et au même temps mettre en œuvre mes compétences.

VII Glossaire

GNSS	*Global Navigation Satellite System*
Constellation	Groupe de satellites du même GNSS
Android	Système de exploitation pour le smartphone
POO	Programmation Orienté Objet
API	*Application Programming Interface*
JAVA	langage de programmation orienté objet
IoT	*Internet of Things*
RINEX	*Receiver Independant EXchange Format*
NMEA	*National Marine Electronics Association*
ICD	*Interface Control Document*

VIII Bibliographie

Using GNSS Raw Measurements on Android Devices
GPS ICD
GPS et GNSS pour Professionnels Géospatiaux
Location API sur Android Developers
Carte avec les positions des satellite
Outils Google pour des mesures GNSS

IX Annexes

Diagramme de classes de l'application :

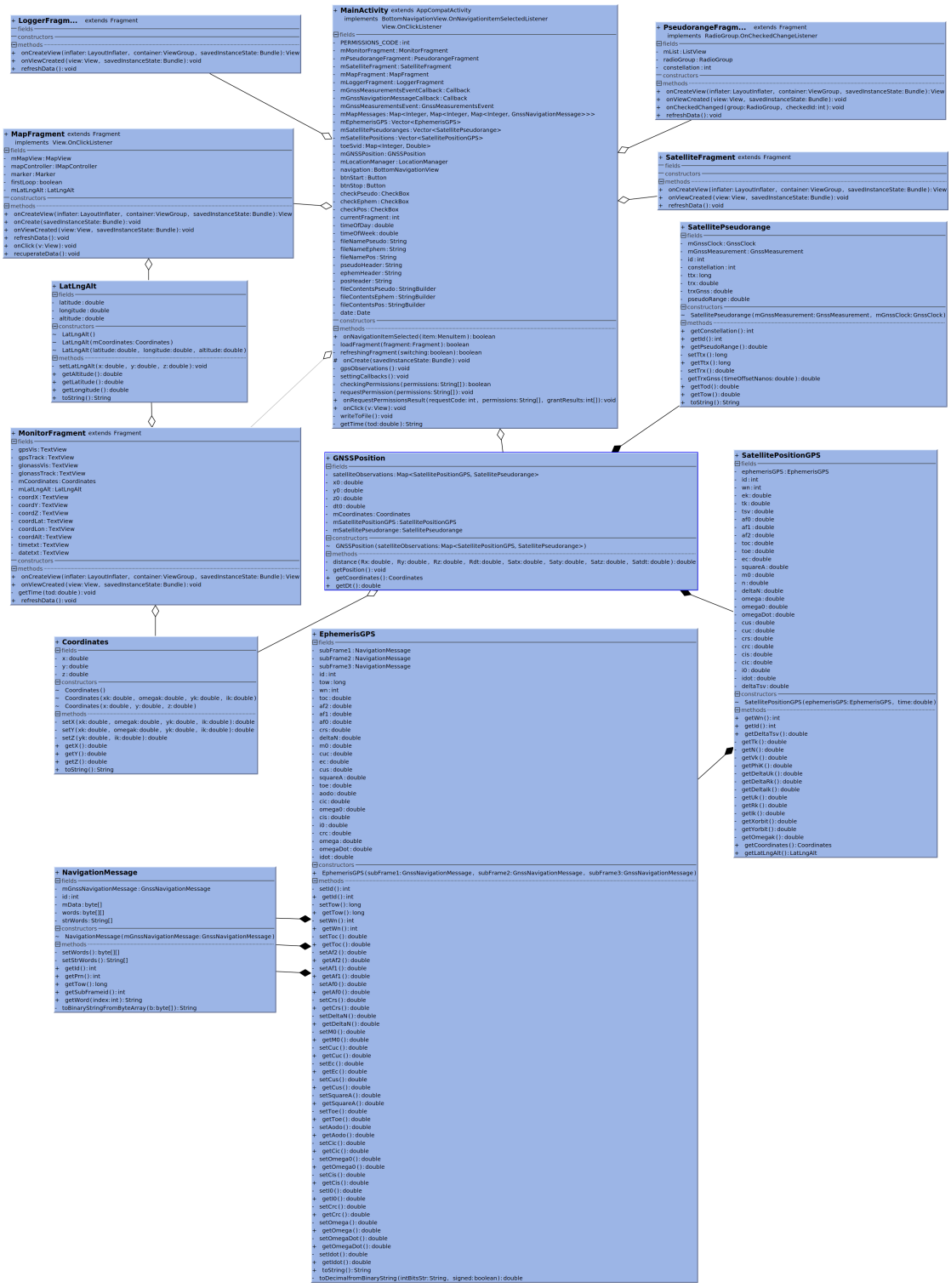
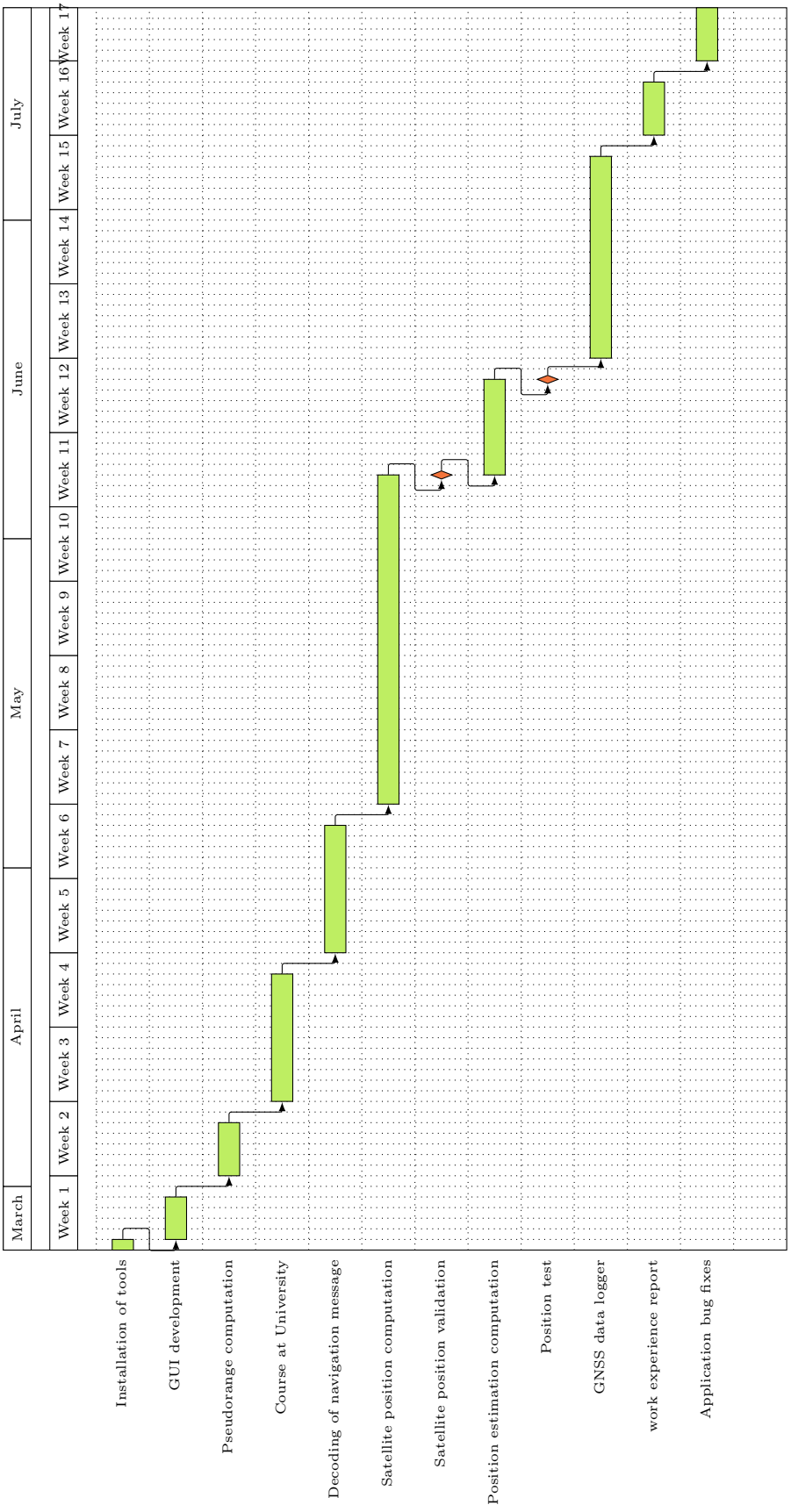


Diagramme de Gantt du projet



SatellitePseudorange.java

```
import android.location.GnssClock;
import android.location.GnssMeasurement;
import android.location.GnssStatus;

import java.text.DecimalFormat;
import java.text.NumberFormat;

import static android.location.GnssMeasurement.*;

/**
 * Created by joseresendiz on 29/03/18.
 */

public class SatellitePseudorange {
    private GnssClock mGnssClock; //Gnssclock
    private GnssMeasurement mGnssMeasurement; //satellite measurements
    private int id; //satellite id
    private int constellation; //satellite constellation
    private long ttx; //transmission time
    private double trx; //reception time corresponding to constellation reference time
    private double trxGnss; //reception time in GNSS time
    private double pseudoRange; //satellite pseudorange


    SatellitePseudorange(GnssMeasurement mGnssMeasurement, GnssClock mGnssClock){
        this.mGnssMeasurement = mGnssMeasurement;
        this.mGnssClock = mGnssClock;
        id = getId();
        constellation = getConstellation();
        ttx = 0;
        trx = 0;
        trxGnss = 0;
        pseudoRange = getPseudoRange();
    }

    //getting satellite constellation
    public int getConstellation(){
        return mGnssMeasurement.getConstellationType();
    }

    //getting satellite id
    public int getId(){
        return mGnssMeasurement.getSvid();
    }

    //pseudorange computing
    public double getPseudoRange() {
        double c = 299792458; //speed of light in m/s
        double numberNanoSecondsSecond = 1e9; //nanoseconds in one second

        //checking that the carrier to noise density is above our threshold
        if(mGnssMeasurement.getCn0DbHz() >= 18) {

            //checking to which constellation we are going to compute
            switch (constellation) {
                //Gps constellation
                case GnssStatus.CONSTELLATION_GPS:
                    //checking the sync states of the associated satellite
                    if ((mGnssMeasurement.getState() &
                            (STATE_TOW_DECODED | STATE_TOW_KNOWN)) != 0) {
                        //if the sync state is correct we continue to
                        //get the transmitter and reception time
                        trx = setTrx();
                        ttx = setTtx();

                    }
                    break;

                //same for GLONASS
                case GnssStatus.CONSTELLATION_GLONASS:
                    if ((mGnssMeasurement.getState() &
                            (STATE_GLO_TOD_DECODED | STATE_GLO_TOD_KNOWN)) != 0) {
                        trx = setTrx();
                        ttx = setTtx();
                    }
                    break;

                    //this constellations are not used because pseudorange is not coherent and
                    //navigation messages for this constellations are not decoded still
                    /*
                //same for GALILEO
                case GnssStatus.CONSTELLATION_GALILEO:
                    if ((mGnssMeasurement.getState() &
                            (STATE_GAL_E1C_2ND_CODE_LOCK | STATE_TOW_KNOWN)) != 0) {
                        //Log.i(TAG,"GALILEO");
                        trx = getTrx();
                        ttx = getTtx();
                    }
                    break;
```

```
90
91                  //and same for BEIDOU
92                  case GnssStatus.CONSTELLATION_BEIDOU:
93                      if ((mGnssMeasurement.getState() &
94                              (STATE_TOW_DECODED | STATE_TOW_KNOWN)) != 0) {
95                          //Log.i(TAG,"BEIDOU");
96                          trx = getTrx();
97                          ttx = getTtx();
98                      }
99                      break;
00                      */
01                  //values by default
02                  default:
03                      ttx = 0;
04                      trx = 0;
05                      break;
06              }
07          }
08
09          //finally we compute pseudorange using time of travel and the speed of light
10          pseudoRange = (((trx - ttx) / numberNanoSecondsSecond) * c);
11          //Log.i(TAG, "trx - ttx : " + (trx-ttx));
12          //if measurement is not coherent we discard this value
13          if(!(18000000 < pseudoRange && pseudoRange < 27000000)){
14              pseudoRange = 0;
15          }
16          //returning pseudorange
17          return pseudoRange;
18      }
19
20      //getting transmission time directly from satellite signal
21      private long setTtx(){
22          //Log.i(TAG, "getTtx :" + mGnssMeasurement.getReceivedSvTimeNanos());
23          return mGnssMeasurement.getReceivedSvTimeNanos();
24      }
25
26      public long getTtx(){
27          return ttx;
28      }
29
30
31      //to get reception time we need to convert time into the different time references, which is
32      //different for each constellation
33      private double setTrx(){
34          double trx;     //reception time depending on each constellation
35          double timeOffsetNanos; //offset at which measurement was taken
36          long fullBiasNanos; //difference between hardware clock and the true gps time
37
38          //some constants
39          double numberNanoSecondsWeek = 6.048e14;
40          double numberNanoSecondsDay = 8.64e13;
41          double numberNanoSeconds100millis = 1e8;
42          double numberNanoSecondsHour = 3.6e12;
43          double numberNanoSecondsSecond = 1e9;
44
45          //getting offset
46          timeOffsetNanos = mGnssMeasurement.getTimeOffsetNanos();
47          //getting difference in clocks
48          fullBiasNanos = mGnssClock.getFullBiasNanos();
49          //getting reception time in true gps time
50          trxGnss = getTrxGnss(timeOffsetNanos);
51          //converting Gnss time in all the different time references
52          switch (constellation) {
53
54              case GnssStatus.CONSTELLATION_GLONASS:
55
56                  // trx = trxgnss - dayNumberNanos + 3h - leapsecond
57                  trx = trxGnss - Math.floor((-fullBiasNanos)/numberNanoSecondsDay)*numberNanoSecondsDay + 3*
                        numberNanoSecondsHour - 18*numberNanoSecondsSecond;
58                  break;
59
60              case GnssStatus.CONSTELLATION_GPS:
61                  // trx = trxgnss - weekNumberNanos
62                  trx = trxGnss - Math.floor((-fullBiasNanos)/numberNanoSecondsWeek)*numberNanoSecondsWeek;
63                  break;
64
65                  /*
66              case GnssStatus.CONSTELLATION_BEIDOU:
67                  // trx = trxgnss - weekNumberNanos - 14s
68                  trx = trxGnss - Math.floor((-fullBiasNanos)/numberNanoSecondsWeek)*numberNanoSecondsWeek - 14*
                        numberNanoSecondsSecond;
69                  break;
70
71              case GnssStatus.CONSTELLATION_GALILEO:
72                  // trx = trxgnss - milliSecondsNumberNanos
73                  trx = trxGnss - Math.floor((-fullBiasNanos)/numberNanoSeconds100millis)*numberNanoSeconds100millis;
74                  //Log.i(TAG, "getTrxGALILEO :" + trx );
75                  break;
76                  */
77              default:
78                  trx = 0;
```

```
                break;
            }
        //returning reception time
        return trx;
    }

    private double getTrxGnss(double timeOffsetNanos){
        long timeNanos; //time of receiver's clock at measurement time
        long fullBiasNanos;
        double biasNanos; //clock's sub-nanoseconds bias

        timeNanos = mGnssClock.getTimeNanos();
        fullBiasNanos = mGnssClock.getFullBiasNanos();
        biasNanos = mGnssClock.getBiasNanos();

        //computing Gnss time (or true GPS time)
        return timeNanos + timeOffsetNanos - (fullBiasNanos + biasNanos);
    }

    public double getTod(){
        double numberNanoSecondsDay = 8.64e13;
        double numberNanoSecondsSecond = 1e9;
        double tod;
        tod = trxGnss - Math.floor((-mGnssClock.getFullBiasNanos())/numberNanoSecondsDay)*numberNanoSecondsDay -18*
             numberNanoSecondsSecond;
        tod /= 1e9;
        return tod;
    }

    public double getTow(){
        double numberNanoSecondsWeek = 6.048e14;
        double numberNanosecondsSecond = 1e9;
        double tow;
        tow = trxGnss - Math.floor((-mGnssClock.getFullBiasNanos())/numberNanoSecondsWeek)*numberNanoSecondsWeek -
             18*numberNanosecondsSecond;
        tow /= 1e9;
        return tow;
    }

    @Override
    public String toString(){
        String txtConstellation;
        String str;
        switch(constellation){
            case GnssStatus.CONSTELLATION_GPS:
                txtConstellation = "GPS";
                break;

            case GnssStatus.CONSTELLATION_GLONASS:
                txtConstellation = "GLO";
                break;

                /*
            case GnssStatus.CONSTELLATION_GALILEO:
                txtConstellation = "GALILEO";
                break;

            case GnssStatus.CONSTELLATION_BEIDOU:
                txtConstellation = "BEIDOU";
                break;
                */

            default:
                txtConstellation = "UNKNOWN";
                break;
        }
        if(pseudoRange!= 0) {
            NumberFormat format = new DecimalFormat("#.###");
            str = txtConstellation +","+ id +
                    "," + format.format(pseudoRange);
            return str;
        }
        else{
            str = txtConstellation + "," + id;
            return str;
        }
    }
}
```

NavigationMessage.java

```
import android.location.GnssNavigationMessage;

import java.math.BigInteger;

/**
 * Created by joseresendiz on 24/04/18.
 */

public class NavigationMessage {
```

```
10      private GnssNavigationMessage mGnssNavigationMessage; //navigation message from current satellite
11      private int id;                 //satellite id
12      private byte[] mData;           //40 bytes Navigation Message
13      private byte[][] words;           //10 words of 4 bytes
14      private String[] strWords;        //10 words of 30 bits
15
16      NavigationMessage(GnssNavigationMessage mGnssNavigationMessage){
17          this.mGnssNavigationMessage = mGnssNavigationMessage;
18          id=this.mGnssNavigationMessage.getSvid();
19          mData = this.mGnssNavigationMessage.getData();
20          words = setWords();
21          strWords = setStrWords();
22      }
23
24      //segmenting the 40 bytes of data into 10 words of 4 bytes
25      private byte[][] setWords(){
26          byte[][] words = new byte[10][4];
27          for (int i = 0; i<10 ; i++){
28              words[i] = new byte[]{mData[i * 4], mData[i * 4 + 1], mData[i * 4 + 2], mData[i * 4 + 3]};
29          }
30          return words;
31      }
32
33      //converting words to 30 character strings
34      private String[] setStrWords(){
35          String[] strWords = new String[10];
36          for (int i = 0; i<10 ; i++){
37              strWords[i] = toBinaryStringFromByteArray(words[i]);
38          }
39          return strWords;
40      }
41
42      //getting satellite id
43      public int getId(){
44          return id;
45      }
46
47      //decoding PRN from navigation message
48      public int getPrn(){
49          return Integer.parseInt(strWords[0].substring(8,14),2);
50      }
51
52      //decoding TOW from navigation message
53      public long getTow(){
54          String tow = strWords[1].substring(0,17);
55          return Long.parseLong(tow,2)*6;
56      }
57      //getting subframe id of navigation message
58      public int getSubFrameid(){
59          return mGnssNavigationMessage.getSubmessageId();
60      }
61
62      //getting string word by index
63      public String getWord(int index){
64          return strWords[index];
65      }
66
67      //converting 4 byte word to 30 character string
68      private String toBinaryStringFromByteArray(byte[] b){
69          BigInteger intWord = new BigInteger(b);
70          String strWord = String.format("%30s", Integer.toBinaryString(intWord.intValue())).replace(' ', '0');
71          return strWord;
72      }
73  }
```

EphemerisGPS.java

```
1   import android.location.GnssNavigationMessage;
2
3   public class EphemerisGPS{
4       //subframes needed to recuperate all ephemerides
5       private NavigationMessage subFrame1;
6       private NavigationMessage subFrame2;
7       private NavigationMessage subFrame3;
8
9       private int id; //satellite id
10      private long tow; //time of week
11
12      //subframe 1 (time corrections)
13      private int wn;          //week number
14      private double toc;      //clock data reference time in seconds
15      private double af2;      //polynomial coefficient (sec/sec^2)
16      private double af1;      //polynomial coefficient (sec/sec)
17      private double af0;      //polynomial coefficient (seconds)
18
19      //subframe 2 (ephemerides part 1)
20      private double crs;      //Amplitude of the Sine Harmonic Correction Term to the Orbit Radius
21      private double deltaN;   //Mean Motion Difference From Computed Value
22      private double m0;       //Mean Anomaly at Reference Time
23      private double cuc;      //Amplitude of the Cosine Harmonic Correction Term to the Argument of Latitude
```

```
24      private double ec;       //Eccentricity
25      private double cus;      //Amplitude of the Sine Harmonic Correction Term to the Argument of Latitude
26      private double squareA; //Square Root of the Semi-Major Axis
27      private double toe;      //Reference Time Ephemeris
28      private double aodo;     //Age of data offset
29
30      //subframe 3 (ephemerides part 2)
31      private double cic;      //Amplitude of the Cosine Harmonic Correction Term to the Angle of Inclination
32      private double omega0;   //Longitude of Ascending Node of Orbit Plane at Weekly Epoch
33      private double cis;      //Amplitude of the Sine Harmonic Correction Term to the Angle of Inclination
34      private double i0;       //Inclination Angle at Reference Time
35      private double crc;      //Amplitude of the Cosine Harmonic Correction Term to the Orbit Radius
36      private double omega;    //Argument of Perigee
37      private double omegaDot;//Rate of Right Ascension
38      private double idot;     //Rate of Inclination Angle
39
40
41      public EphemerisGPS(GnssNavigationMessage subFrame1 ,GnssNavigationMessage subFrame2, GnssNavigationMessage
            subFrame3) {
42          this.subFrame1 = new NavigationMessage(subFrame1);
43          this.subFrame2 = new NavigationMessage(subFrame2);
44          this.subFrame3 = new NavigationMessage(subFrame3);
45
46          id = setId();
47          tow = setTow();
48
49          wn = setWn();
50          toc = setToc();
51          af2 = setAf2();
52          af1 = setAf1();
53          af0 = setAf0();
54
55          crs = setCrs();
56          deltaN = setDeltaN();
57          m0 = setM0();
58          cuc = setCuc();
59          ec = setEc();
60          cus = setCus();
61          squareA = setSquareA();
62          toe = setToe();
63          aodo = setAodo();
64
65          cic = setCic();
66          omega0 = setOmega0();
67          cis = setCis();
68          i0 = setI0();
69          crc = setCrc();
70          omega = setOmega();
71          omegaDot = setOmegaDot();
72          idot = setIdot();
73      }
74
75      //setting satellite id
76      private int setId(){
77          return subFrame1.getId();
78      }
79
80      //getting satellite id
81      public int getId(){
82          return id;
83      }
84      //all methods from here to method toString are methods to decode the corresponding value
85
86      private long setTow(){
87          return subFrame2.getTow();
88      }
89
90      public long getTow(){
91          return tow;
92      }
93
94      private int setWn(){
95          return (int)toDecimalfromBinaryString(this.subFrame1.getWord(2).substring(0,10), false);
96      }
97
98      public int getWn(){
99          return wn;
00      }
01
02      private double setToc(){
03          return toDecimalfromBinaryString(this.subFrame1.getWord(7).substring(8,24), false)
04                  * Math.pow(2,4);
05      }
06
07      public double getToc(){
08          return toc;
09      }
10
11      private double setAf2(){
12          return toDecimalfromBinaryString(this.subFrame1.getWord(8).substring(0,8),true)
13                  * Math.pow(2,-55);
```

```
14      }
15
16      public double getAf2(){
17          return af2;
18      }
19
20      private double setAf1(){
21          return toDecimalfromBinaryString(this.subFrame1.getWord(8).substring(8,24),true)
22                  * Math.pow(2,-43);
23      }
24
25      public double getAf1(){
26          return af1;
27      }
28
29      private double setAf0(){
30          return toDecimalfromBinaryString(this.subFrame1.getWord(9).substring(0,22),true)
31                  * Math.pow(2,-31);
32      }
33
34      public double getAf0(){
35          return af0;
36      }
37
38      private double setCrs(){
39          return toDecimalfromBinaryString(this.subFrame2.getWord(2).substring(8,24), true)
40                  * Math.pow(2,-5);
41      }
42
43      public double getCrs(){
44          return crs;
45      }
46
47      private double setDeltaN(){
48          return toDecimalfromBinaryString(this.subFrame2.getWord(3).substring(0,16), true)
49                  * Math.pow(2,-43) * Math.PI;
50      }
51
52      public double getDeltaN(){
53          return deltaN;
54      }
55
56      private double setM0(){
57          return toDecimalfromBinaryString(this.subFrame2.getWord(3).substring(16,24)
58                  + this.subFrame2.getWord(4).substring(0,24), true) * Math.pow(2,-31) * Math.PI;
59      }
60      public double getM0(){
61          return m0;
62      }
63
64      private double setCuc(){
65          return toDecimalfromBinaryString(this.subFrame2.getWord(5).substring(0,16), true)
66                  * Math.pow(2,-29);
67      }
68      public double getCuc(){
69          return cuc;
70      }
71
72      private double setEc(){
73          return toDecimalfromBinaryString(this.subFrame2.getWord(5).substring(16,24)
74                  + this.subFrame2.getWord(6).substring(0,24), false) * Math.pow(2,-33);
75      }
76
77      public double getEc(){
78          return ec;
79      }
80
81      private double setCus(){
82          return toDecimalfromBinaryString(this.subFrame2.getWord(7).substring(0,16), true)
83                  * Math.pow(2,-29);
84      }
85      public double getCus(){
86          return cus;
87      }
88
89      private double setSquareA(){
90          return toDecimalfromBinaryString(this.subFrame2.getWord(7).substring(16,24)
91                  + this.subFrame2.getWord(8).substring(0,24), false) * Math.pow(2,-19);
92      }
93      public double getSquareA(){
94          return squareA;
95      }
96
97      private double setToe(){
98          return toDecimalfromBinaryString(this.subFrame2.getWord(9).substring(0,16), false)
99                  * Math.pow(2,4);
00      }
01      public double getToe(){
02          return toe;
03      }
04
```

```
05      private double setAodo(){
06          return toDecimalfromBinaryString(this.subFrame2.getWord(9).substring(17,22),false)
07                  * 900;
08      }
09      public double getAodo(){
10          return aodo;
11      }
12
13      private double setCic(){
14          return toDecimalfromBinaryString(this.subFrame3.getWord(2).substring(0,16), true)
15                  * Math.pow(2,-29);
16      }
17      public double getCic(){
18          return cic;
19      }
20
21      private double setOmega0(){
22          return toDecimalfromBinaryString(this.subFrame3.getWord(2).substring(16,24)
23                  + this.subFrame3.getWord(3).substring(0,24), true) * Math.pow(2,-31) * Math.PI;
24      }
25      public double getOmega0(){
26          return omega0;
27      }
28
29      private double setCis(){
30          return toDecimalfromBinaryString(this.subFrame3.getWord(4).substring(0,16), true)
31                  * Math.pow(2,-29);
32      }
33      public double getCis(){
34          return cis;
35      }
36
37      private double setI0(){
38          return toDecimalfromBinaryString(this.subFrame3.getWord(4).substring(16,24)
39                  + this.subFrame3.getWord(5).substring(0,24), true) * Math.pow(2,-31) * Math.PI;
40      }
41      public double getI0(){
42          return i0;
43      }
44
45      private double setCrc(){
46          return toDecimalfromBinaryString(this.subFrame3.getWord(6).substring(0,16), true)
47                  * Math.pow(2, -5);
48      }
49      public double getCrc(){
50          return crc;
51      }
52
53      private double setOmega(){
54          return toDecimalfromBinaryString(this.subFrame3.getWord(6).substring(16,24)
55                  +this.subFrame3.getWord(7).substring(0,24), true) * Math.pow(2, -31) * Math.PI;
56      }
57      public double getOmega(){
58          return omega;
59      }
60
61      private double setOmegaDot(){
62          return toDecimalfromBinaryString(this.subFrame3.getWord(8).substring(0,24), true)
63                  * Math.pow(2,-43) * Math.PI;
64      }
65      public double getOmegaDot(){
66          return omegaDot;
67      }
68
69      private double setIdot(){
70          return toDecimalfromBinaryString(this.subFrame3.getWord(9).substring(8,22), true)
71                  * Math.pow(2,-43) * Math.PI;
72      }
73      public double getIdot(){
74          return idot;
75      }
76
77      @Override
78      public String toString() {
79          String str = id +
80                  "," + tow +
81                  "," + wn +
82                  "," + toc +
83                  "," + af2 +
84                  "," + af1 +
85                  "," + af0 +
86                  "," + crs +
87                  "," + deltaN +
88                  "," + m0 +
89                  "," + cuc +
90                  "," + ec +
91                  "," + cus +
92                  "," + squareA +
93                  "," + toe +
94                  "," + aodo +
95                  "," + cic +
```

```
                    "," + omega0 +
                    "," + cis +
                    "," + i0 +
                    "," + crc +
                    "," + omega +
                    "," + omegaDot +
                    "," + idot;
        return str;
    }

    //converting to decimal from binary string either it is a binary signed number or not
    private double toDecimalfromBinaryString(String intBitsStr, boolean signed){
        long unsigned = Long.parseLong(intBitsStr,2);
        if (!signed) {
            return unsigned;
        }
        else{
            long mask = (long) Math.pow(2, (intBitsStr.length()-1));
            return -(unsigned & mask) + (unsigned & ~mask);
        }
    }

}
```

SatellitePositionGPS.java

```
public class SatellitePositionGPS {

    private EphemerisGPS ephemerisGPS;
    private int id;             //SvId
    private int wn;             //week number
    private double ek;          //Eccentric Anomaly
    private double tk;          //Time from ephemeris reference time
    private double tsv;         //time at message transmission time (GPS time in seconds)
    private double af0;         //polynomial coefficient (seconds)
    private double af1;         //polynomial coefficient (sec/sec)
    private double af2;         //polynomial coefficient (sec/sec^2)
    private double toc;         //clock data reference time in seconds
    private double toe;         //Reference Time Ephemeris
    private double ec;          //Eccentricity
    private double squareA;     //Square Root of the Semi-Major Axis
    private double m0;          //Mean Anomaly at Reference Time
    private double n;           //Corrected mean motion
    private double deltaN;      //Mean Motion Difference From Computed Value
    private double omega;       //Argument of Perigee
    private double omega0;      //Longitude of Ascending Node of Orbit Plane at Weekly Epoch
    private double omegaDot;    //Rate of Right Ascension
    private double cus;         //Amplitude of the Sine Harmonic Correction Term to the Argument of Latitude
    private double cuc;         //Amplitude of the Cosine Harmonic Correction Term to the Argument of Latitude
    private double crs;         //Amplitude of the Sine Harmonic Correction Term to the Orbit Radius
    private double crc;         //Amplitude of the Cosine Harmonic Correction Term to the Orbit Radius
    private double cis;         //Amplitude of the Sine Harmonic Correction Term to the Angle of Inclination
    private double cic;         //Amplitude of the Cosine Harmonic Correction Term to the Angle of Inclination
    private double i0;          //Inclination Angle at Reference Time
    private double idot;        //Rate of Inclination Angle
    private double deltaTsv;    //code phase time offset


    SatellitePositionGPS(EphemerisGPS ephemerisGPS, double time){
        this.ephemerisGPS = ephemerisGPS;
        id = ephemerisGPS.getId();
        wn = ephemerisGPS.getWn();
                                            //example ephemeris values :
        tsv = time;                         //2.0507347300000000e5;
        af0 = ephemerisGPS.getAf0();        //0.0003307778388260000;
        af1 = ephemerisGPS.getAf1();        //-2.38742359215000e-12;
        af2 = ephemerisGPS.getAf2();        //0;
        toc = ephemerisGPS.getToc();        //208800;
        toe = ephemerisGPS.getToe();        //208800;
        ec = ephemerisGPS.getEc();          //0.00721086398698000;
        squareA = ephemerisGPS.getSquareA();//5153.58441734000;
        m0 = ephemerisGPS.getM0();          //0.572839125963000;
        deltaN = ephemerisGPS.getDeltaN();  //4.21160400164000e-09;
        omega =  ephemerisGPS.getOmega();   //0.986168231602000;
        omega0 = ephemerisGPS.getOmega0();  //1.49866635214000;
        omegaDot =  ephemerisGPS.getOmegaDot(); //-8.27855912093000e-09;
        cus = ephemerisGPS.getCus();        //1.36345624924000e-06;
        cuc = ephemerisGPS.getCuc();        //1.05984508991000e-06;
        crs = ephemerisGPS.getCrs();        //19.9687500000000;
        crc = ephemerisGPS.getCrc();        //365.750000000000;
        cis = ephemerisGPS.getCis();        //7.63684511185000e-08;
        cic = ephemerisGPS.getCic();        //-7.07805156708000e-08;
        i0 = ephemerisGPS.getI0();          //0.986297856385000;
        idot = ephemerisGPS.getIdot();      //-1.15361948145000e-10;
        n = getN();
        tk = getTk();

    }
    public int getWn(){
```

```
65          return wn;
66      }
67
68      //getting satellite id
69      public int getId(){
70          return id;
71      }
72
73      public double getDeltaTsv(){
74          return deltaTsv;
75      }
76
77      //computing time from ephemeris reference time
78      private double getTk(){
79          double f = -4.442807633e-10;     //constant = (-2/square(mu))/c^2
80          double deltaTr;                  //relativistic correction term in seconds
81          double t;                        //GPS time in seconds
82          double tk1;                      //Time from ephemeris reference time without relativistic correction
83          double mk;                       //Mean anomaly
84          double diff = 1;                 //iteration constant
85
86          deltaTsv = af0 + af1*(tsv - toc) + af2*Math.pow((tsv - toc),2);
87
88          t = tsv - deltaTsv;
89
90          tk1 = t - toe;
91
92          mk = m0 + n*tk1;
93          ek = mk;
94
95          //Kepler's Equation for Eccentric Anomaly solved by iteration (radians)
96          while(Math.abs(diff)>1.0e-13){
97              diff = (mk+ec*Math.sin(ek)) - ek;
98              ek+= diff;
99          }
00
01          deltaTr = f * ec * squareA * Math.sin(ek);
02          deltaTsv += deltaTr;
03          tk = tsv - toe - deltaTsv;
04
05          //beginning and end of week crossovers correction
06          if(tk > 302400){
07              tk -= 604800;
08          }else if(tk < -302400){
09              tk += 604800;
10          }
11          return tk;
12      }
13
14      //computing corrected mean motion
15      private double getN(){
16          double mu = 3.986005e14;            //value of Earth's universal gravitational parameters
17          double a = Math.pow(squareA,2);
18          double n0 = Math.sqrt(mu/Math.pow(a,3));     //Computed mean motion (rad/sec)
19
20          return n0 + deltaN;
21
22      }
23
24      //computing true anomaly
25      private double getVk(){
26          double vk;
27          vk = Math.atan2( (Math.sqrt(1 - Math.pow(ec,2))*Math.sin(ek))/(1 - ec*Math.cos(ek)) ,
28                           ((Math.cos(ek) - ec)/(1-ec*Math.cos(ek))) );
29
30          return vk;
31
32      }
33
34      //computing argument of latitude
35      private double getPhiK(){
36          double phiK;
37          double vk = getVk();
38
39          phiK = vk + omega;
40
41          return phiK;
42      }
43
44      //computing argument of latitude correction
45      private double getDeltaUk(){
46          double phiK = getPhiK();
47          double deltaUk;
48
49          deltaUk = cus*Math.sin(2*phiK) + cuc*Math.cos(2*phiK);
50
51          return deltaUk;
52      }
53
54      //computing radius correction
55      private double getDeltaRk(){
```

```
56          double phiK = getPhiK();
57          double deltaRk;
58
59          deltaRk = crs*Math.sin(2*phiK) + crc*Math.cos(2*phiK);
60
61          return deltaRk;
62      }
63
64      //computing inclination correction
65      private double getDeltaIk(){
66          double deltaIk;
67
68          deltaIk = cis*Math.sin(2*getPhiK()) + cic*Math.cos(2*getPhiK());
69
70          return deltaIk;
71      }
72
73      //computing corrected argument of latitude
74      private double getUk(){
75          double phiK = getPhiK();
76          double deltaUk = getDeltaUk();
77          double uk;
78
79          uk = phiK + deltaUk;
80
81          return uk;
82      }
83
84      //computing corrected radius
85      private double getRk(){
86          double A = Math.pow(squareA,2);
87          double deltaRk = getDeltaRk();
88          double rk;
89
90          rk = A*(1 - ec*Math.cos(ek)) + deltaRk;
91
92          return rk;
93      }
94
95      //computing corrected inclination
96      private double getIk(){
97          double deltaIk = getDeltaIk();
98          double ik;
99
00          ik = i0 + deltaIk + idot*tk;
01
02          return ik;
03      }
04
05      //computing x in orbital plane
06      private double getXorbit(){
07          double rk = getRk();
08          double uk = getUk();
09          double x;
10
11          x = rk*Math.cos(uk);
12
13          return x;
14      }
15
16      //computing y in orbital plane
17      private double getYorbit(){
18          double rk = getRk();
19          double uk = getUk();
20          double y;
21
22          y = rk*Math.sin(uk);
23
24          return y;
25      }
26
27      //computing corrected longitude of ascending node
28      private double getOmegak(){
29          double omegaDote = 7.2921151467e-5;  //WGS 84 value of the earth's rotation rate in rad/sec
30          double omegaK;
31
32          omegaK = omega0 + (omegaDot - omegaDote)*tk - omegaDote*toe;
33
34          return omegaK;
35      }
36
37      //computing Earth-fixed coordinates
38      public Coordinates getCoordinates(){
39          double xk = getXorbit();
40          double yk = getYorbit();
41          double omegak = getOmegak();
42          double ik = getIk();
43
44          return new Coordinates(xk, omegak, yk, ik);
45      }
46
```

```
47      //converting Earth-fixed coordinated to geographic coordinates
48      public LatLngAlt getLatLngAlt(){
49          Coordinates mCoordinates = getCoordinates();
50          return new LatLngAlt(mCoordinates);
51      }
52  }
```

Coordinates.java

```
1   public class Coordinates {
2
3       //Earth-fixed coordinates
4       private double x;
5       private double y;
6       private double z;
7   numbers=left
8       Coordinates(){
9           this.x = 0.000;
10          this.y = 0.000;
11          this.z = 0.000;
12      }
13
14      //xk : x of orbital plane
15      //yk : y of orbital plane
16      //omegak : corrected longitude of ascending node
17      //ik : corrected inclination
18      Coordinates(double xk, double omegak, double yk, double ik){
19          this.x = setX(xk, omegak, yk, ik);
20          this.y = setY(xk, omegak, yk, ik);
21          this.z = setZ(yk, ik);
22      }
23
24      Coordinates(double x, double y, double z){
25          this.x = x;
26          this.y = y;
27          this.z = z;
28      }
29
30      //computing x in Earth-fixed coordinates
31      private double setX(double xk, double omegak, double yk, double ik) {
32          return xk*Math.cos(omegak) - yk*Math.cos(ik)*Math.sin(omegak);
33
34      }
35
36      //computing y in Earth-fixed coordinates
37      private double setY(double xk, double omegak, double yk, double ik){
38          return xk*Math.sin(omegak) + yk*Math.cos(ik)*Math.cos(omegak);
39      }
40
41      //computing z in Earth-fixed coordinates
42      private double setZ(double yk, double ik){
43          return yk*Math.sin(ik);
44      }
45
46      public double getX() {
47          return x;
48      }
49
50      public double getY() {
51          return y;
52      }
53
54      public double getZ() {
55          return z;
56      }
57
58      @Override
59      public String toString() {
60          return "X:" + x + ",Y:" + y + ",Z:" + z;
61      }
62  }
```

LatLngAlt.java

```
1   /*
2   * Class to convert cartesian coordinates (X,Y,Z) to geographic coordinates(Lat, Lon, Alt)
3   */
4   public class LatLngAlt {
5
6       private double latitude;
7       private double longitude;
8       private double altitude;
9
10      LatLngAlt(){
11          this.latitude = 0.000000;
12          this.longitude = 0.000000;
13          this.altitude = 0.000000;
14      }
```

```
15      //initializing geographic coordinates with Earth-fixed coordinates
16      LatLngAlt(Coordinates mCoordinates){
17          setLatLngAlt(mCoordinates.getX(),mCoordinates.getY(), mCoordinates.getZ());
18      }
19
20      LatLngAlt(double latitude, double longitude, double altitude){
21          this.latitude = latitude;
22          this.longitude = longitude;
23          this.altitude = altitude;
24      }
25
26      private void setLatLngAlt(double x, double y, double z) {
27          //WGS84 ellipsoid constants
28          int a = 6378137;
29          double e = 8.1819190842622e-2;
30
31          //calculations
32          double b = Math.sqrt(Math.pow(a,2) * (1 - Math.pow(e,2)));
33          double ep = Math.sqrt((Math.pow(a,2) - Math.pow(b,2)) / Math.pow(b,2));
34          double p = Math.sqrt(Math.pow(x,2) + Math.pow(y,2));
35          double th = Math.atan2(a*z, b*p);
36          longitude = Math.atan2(y,x);
37          latitude = Math.atan2((z+Math.pow(ep,2)*b* Math.pow(Math.sin(th),3)), (p-Math.pow(e,2)*a*Math.pow(Math.cos(
                th),3)));
38          double n = a/Math.sqrt(1 - Math.pow(e,2)* Math.pow(Math.sin(latitude),2));
39          altitude = p/Math.cos(latitude) - n;
40
41          longitude = longitude%(2*Math.PI);
42
43          if(Math.abs(x) < 1 && Math.abs(y)<1){
44              altitude  = Math.abs(z-b);
45          }
46
47          //converting radians to degrees
48          longitude = Math.toDegrees(longitude);
49          latitude = Math.toDegrees(latitude);
50
51
52      }
53
54      public double getAltitude() {
55          return altitude;
56      }
57
58      public double getLatitude() {
59          return latitude;
60      }
61
62      public double getLongitude() {
63          return longitude;
64      }
65
66      @Override
67      public String toString() {
68          return latitude + "," + longitude +","+ altitude;
69      }
70  }
```

GNSSPosition.java

```
1   import org.ejml.simple.SimpleMatrix;
2   import java.util.Map;
3
4   public class GNSSPosition{
5
6       private Map<SatellitePositionGPS,SatellitePseudorange> satelliteObservations;
7       private double x0;
8       private double y0;
9       private double z0;
10      private double dt0;
11
12
13      GNSSPosition(Map<SatellitePositionGPS,SatellitePseudorange> satelliteObservations){
14          this.satelliteObservations = satelliteObservations;
15          x0 = 4274232.9392;
16          y0 = 11590.2498;
17          z0 = 4718407.8887;
18          dt0 = 0;
19          getPosition();
20
21      }
22
23      private double distance(double Rx, double Ry, double Rz, double Rdt, double Satx, double Saty, double Satz,
            double Satdt){
24          double c = 299792458;
25          return Math.sqrt(Math.pow(Rx - Satx,2) + Math.pow(Ry - Saty,2) + Math.pow(Rz - Satz,2)) + c*Rdt - c*Satdt;
26      }
27
28      //least square method to compute position
29      private void getPosition(){
```

```
30          double c = 299792458;
31          double omega = 0.001;
32          int i;
33
34          SimpleMatrix X = new SimpleMatrix(new double[][]{new double[]{x0,y0,z0,dt0}}).transpose(); //coordinates and
                time
35          SimpleMatrix A = new SimpleMatrix(satelliteObservations.size(),4); //Observation matrix (model)
36          SimpleMatrix B = new SimpleMatrix(satelliteObservations.size(),1); //measurement
37          SimpleMatrix Q = SimpleMatrix.identity(satelliteObservations.size());       //covariance matrix
38          SimpleMatrix C;
39          SimpleMatrix N;
40          SimpleMatrix Result;
41
42          while (true) {
43
44              i = 0;
45              for(Map.Entry<SatellitePositionGPS,SatellitePseudorange> entry : satelliteObservations.entrySet()){
46                  double pseudorange = entry.getValue().getPseudoRange();
47                  double satX = entry.getKey().getCoordinates().getX();
48                  double satY = entry.getKey().getCoordinates().getY();
49                  double satZ = entry.getKey().getCoordinates().getZ();
50                  double satDt = entry.getKey().getDeltaTsv();
51
52                  A.set(i, 0, 2 * (X.get(0) - satX) * (1 / (2 * Math.sqrt(Math.pow(X.get(0) - satX, 2) + Math.pow(X.
                        get(1) - satY, 2) + Math.pow(X.get(2) - satZ, 2)))));
53                  A.set(i, 1, 2 * (X.get(1) - satY) * (1 / (2 * Math.sqrt(Math.pow(X.get(0) - satX, 2) + Math.pow(X.
                        get(1) - satY, 2) + Math.pow(X.get(2) - satZ, 2)))));
54                  A.set(i, 2, 2 * (X.get(2) - satZ) * (1 / (2 * Math.sqrt(Math.pow(X.get(0) - satX, 2) + Math.pow(X.
                        get(1) - satY, 2) + Math.pow(X.get(2) - satZ, 2)))));
55                  A.set(i, 3, c);
56
57                  B.set(i, 0, pseudorange - distance(X.get(0), X.get(1), X.get(2), X.get(3), satX, satY, satZ, satDt))
                        ;
58                  i++;
59              }
60
61
62              C = A.transpose().mult(Q).mult(B);
63              N = A.transpose().mult(Q).mult(A);
64              Result = N.invert().mult(C);
65              if(Result.normF() < omega){
66                  break;
67              }
68
69              X = X.plus(Result);
70          }
71          x0 = X.get(0,0);
72          y0 = X.get(1,0);
73          z0 = X.get(2,0);
74          dt0 = X.get(3,0);
75      }
76
77      public Coordinates getCoordinates(){
78          return new Coordinates(x0,y0,z0);
79      }
80
81      public double getDt(){
82          return dt0;
83      }
84  }
```