Option INFO : Projet d'application

Récupération de données de navigation sur Android 8

Rapport final

Sheng GAO
Mitsinjosoa RAMANDANIAINA

Encadrante : Myriam Servières

Sommaire

Contexte 2
Objectifs 2
Cahier des charges fonctionnelle 3
Planning général 4
Prise en main du projet 5
  Principe de positionnement GNSS 5
  Analyse de l'existant 5
  Prise en main d'Android Studio 6
  Les APIs spécifiques nécessaires au calcul de position GPS 8
Diagramme UML 9
Travails effectués 10
Livrables 13
Conclusion 14

I. Contexte

Depuis Android 7, les données brutes de navigation sont accessibles directement via l'**API** 24 de l'appareil, ce qui a ouvert la porte aux développeurs pour développer des algorithmes de navigation différents de ceux proposés par le constructeur.

L'IFSTTAR qui est un acteur majeur dans l'intégration des nouvelles technologies appliquées à la navigation a décidé de développer ses propres algorithmes pour l'exploitation des données GPS sur un appareil Android 7. Ce projet a été mené à bien par un étudiant dans le cadre d'un stage, et IFSTTAR souhaite que nous reprenions ce projet en l'adaptant sur Android 8 et en y intégrant la possibilité d'exploiter les données sur 2 fréquences de transmission.

Nous travaillerons sur ce projet sous la supervision de l'équipe GEOLOC.

II. Objectifs

L'objectif principale de ce projet est donc de proposer une application de géolocalisation sous Android en utilisant les données GPS brutes. Dans un second temps, on va exploiter ces données en prenant les 2 bandes fréquences disponibles sur l'appareil Android qu'on nous a fourni..

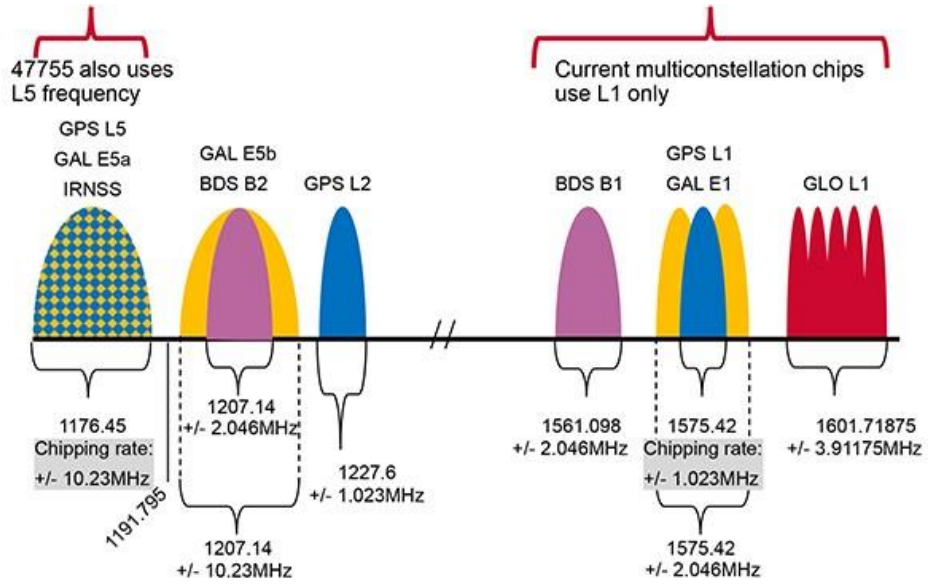


fig. 1 spectre en fréquence des signaux GNSS. *Google Images*

III. Cahier des charges fonctionnelle

Portage du code existant sur Android 7 vers Android 8 pour la localisation GPS

Spécifier le diagramme des classes
Effectuer des changements sur les méthodes dans les classes si nécessaires
Ajouter la javadoc et les commentaires
Effectuer des tests

Implémentation d'un algorithme pour exploiter la double fréquence GPS

Récupérer les données de la 2ème fréquence et comparer avec les données de GEO++
Calculer la position avec la 2ème fréquence, comparer, valider
Adapter le calcul de la position
Améliorer l'affichage sur la carte
Améliorer la précision en intégrant les 2 fréquences avec un problème des moindres carrés
Créer un algorithme qui permet de calculer les erreurs de transmission due à la couche ionosphérique

Intégration du système de navigation européen Galileo

Calculer les pseudodistance avec les satellites captés (déjà fait par José)
Calculer les éphémérides des satellites
Implémenter l'algorithme de trilatération et calculer la position

IV. Planning général

Semaine 01-02 01/10 au 15/10

prise en main du projet
diagramme de conception
4 oct. Réunion d'avancement

Semaine 03-04 15/10 au 29/10

Portage du code sur Android 8
Debogage

Semaine 05 **toussaint**

Semaine 06-07 05/11 au 19/11

Récupération des données provenant de la bande L5
Comparaison avec les données du GEO++ logger
9 nov. Réunion d'avancement

Semaine 08-09 19/11 au 10/12

Premier essai calcul de position fréquence L5
Affichage des fichiers enregistrés dans le logger
7 déc. Réunion d'avancement

Semaine 10-11-12 10/12 au 31/12

Réécriture des méthodes pour L5
Calcul des coordonnées
Tests de validation de la pseudodistance entre GEO++ et notre application
14 déc. Réunion
19 déc. Rapport de projet
21 déc. Soutenance

Cout total : 64 heures = 2 semaines en temps plein

V. Prise en main du projet

Principe de positionnement GNSS

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 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 absolue d'un point en utilisant la géométrie des sphères et en plus une correction du temps.

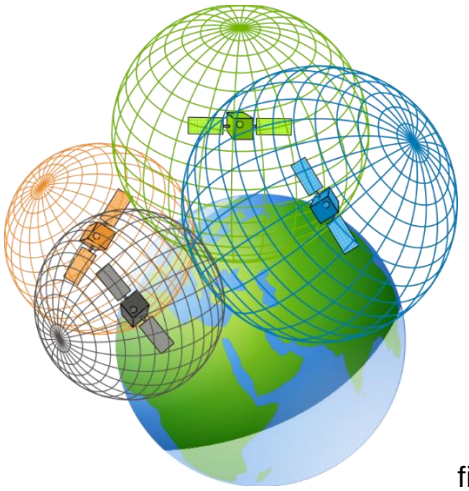


fig. 2 Illustration du positionnement GNSS

Analyse de l'existant

Nous avons reçu l'ensemble des codes sources réalisés sous Android Studio d'une application fonctionnelle sous Android 7 qui calcule les coordonnées GPS à partir des données brutes.

Cette application a été installé sur un smartphone Huawei P10 mis à notre disposition par l'entreprise.

Les fonctionnalités de cette application sont :

- La récupération des données brutes GPS
- Le calcul de pseudodistance
- Calcul de la position des satellites
- Calcul de la position de l'appareil et affichage sur une carte
- Enregistrement des données calculées dans un fichier texte

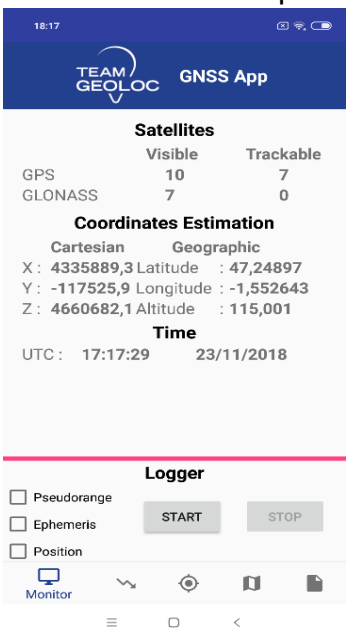


Fig. 3 fenêtre d'accueil de l'application

Prise en main d'Android Studio

Avant de pouvoir commencer à exploiter ce projet nous devions apprendre à utiliser le logiciel de développement d'application Android de IntelliJ : Android Studio. Le principal avantage de ce programme est que le langage utilisé est le Java, ou un dérivé qui est Kotlin, donc nous avions déjà les notions de bases sur les classes, les méthodes, les héritages etc.

Un projet dans Android Studio possède une structure normalisée. Il y a deux catégories de fichiers, ceux qui sont constamment utilisés et ceux qui ne sont là que pour la compilation et l'édition de l'APK (fichier exécutable Android).

Les fichiers utilisés en permanence sont :

- le **manifest** (AndroidManifest.xml) qui décrit l'application à construire et ses composants ;
- le dossier **res** (pour ressources) qui contient les ressources du projet (les layouts, les chaînes de caractères, les images, les fichiers internes, les fichiers XML, les couleurs et thèmes) dont les sous-dossiers sont :
    - *Drawable* (pour les images, usuellement vous trouverez trois dossiers, un par qualité haute, moyenne, faible),
    - *Layout* (pour la définition des différents fichiers de layouts des activités qui déclarent et positionnent les composants graphiques),
    - *Values* (pour les chaînes de caractères, les couleurs, les thèmes, …),
    - *Raw* (pour les fichiers),
    - *XML* (pour les fichiers XML) ;
- le dossier **src** pour les sources
- et enfin le dossier **test** pour les tests.

Les fichiers utilisés pour la compilation et le déploiement sont :

- le fichier de **build.xml** qui construit le projet ;
- les fichiers **properties** default et local qui sont utilisés par le build ;
- le dossier **asset** qui contient des fichiers pour le déploiement ;
- le dossier **bin** (pour binaries) qui contient les classes compilées ;
- le dossier **gen** (pour generated) qui contient le code source généré par les outils de compilation Android ;
- le dossier **libs** (pour librairies) qui contient les librairies.

Le fichier manifest est un élément clef du projet, il définit :

- le projet et comment il sera interprété par le système
- les dépendances systèmes (par exemple : la version du système, les instruments nécessaires, la géolocalisation, etc.)
- la version du projet
- les différents composants de l'application (Activity, Service, Content Provider)

La composante principale d'une application sous Android est l'activité (**Activity**). C'est le cœur de l'application et possède généralement une **View** au minimum, c'est-à-dire un écran graphique.

Les activités sont un concept de programmation spécifique à Android. Dans le développement d'applications traditionnelles, il existe généralement une méthode main statique, qui est exécutée pour lancer l'application. Avec Android, c'est différent, chaque application Android peut être lancée par le biais de toute activité enregistrée dans une application. Dans la pratique, la plupart des applications n'ont qu'une activité spécifique comme point d'entrée.

Le cycle de vie d'activité est implémenté en tant que collection d'appels de méthodes tout au long du cycle de vie d'une activité. Ces méthodes permettent aux développeurs d'implémenter les fonctionnalités qui sont nécessaires pour satisfaire les exigences de gestion de ressources et d'état de leurs applications.

Les méthodes principales sont :

OnCreate

OnCreate est la première méthode appelée lorsqu'une activité est créée. *OnCreate* est toujours remplacée pour effectuer des initialisations de démarrage qui peuvent être requis par une activité tel que :

- Création de vues
- L'initialisation de variables
- Liaison de données statiques à des listes

OnStart

OnStart est toujours appelé par le système après *OnCreate* est terminé. Les activités peuvent surcharger cette méthode s'ils doivent effectuer des tâches spécifiques avant que l'activité devienne visible, comme actualiser les valeurs actuelles des vues au sein de l'activité.

OnResume

Le système appelle *OnResume* lorsque l'activité est prête à commencer à interagir avec l'utilisateur. Les activités doivent surcharger cette méthode pour effectuer des tâches telles que :

- Renforcement des fréquences d'images
- Animations
- Tester les mises à jour GPS
- Afficher toutes les alertes pertinentes ou boîtes de dialogue
- Gérer les événements externes

OnPause

OnPause est appelée lorsque le système est sur le point de placer l'activité en arrière-plan ou lorsque l'activité est partiellement masquée. Les activités doivent surcharger cette méthode s'ils doivent :

- Valider les changements apportés aux données
- Détruire ou nettoyer des objets qui consomment des ressources

OnStop

OnStop est appelée lorsque l'activité n'est plus visible par l'utilisateur. Cela se produit lorsqu'une des actions suivantes se produit :

- Une nouvelle activité en cours de démarrage et couvre cette activité.
- Une activité existante est mise au premier plan.
- L'activité est en cours de destruction.

Les APIs spécifiques nécessaires au calcul de position GPS

GnssNavigationMessage

```
public final class GnssNavigationMessage
extends Object implements Parcelable
```

java.lang.Object
↳ android.location.GnssNavigationMessage

A class containing a GNSS satellite Navigation Message. ...

GnssMeasurement

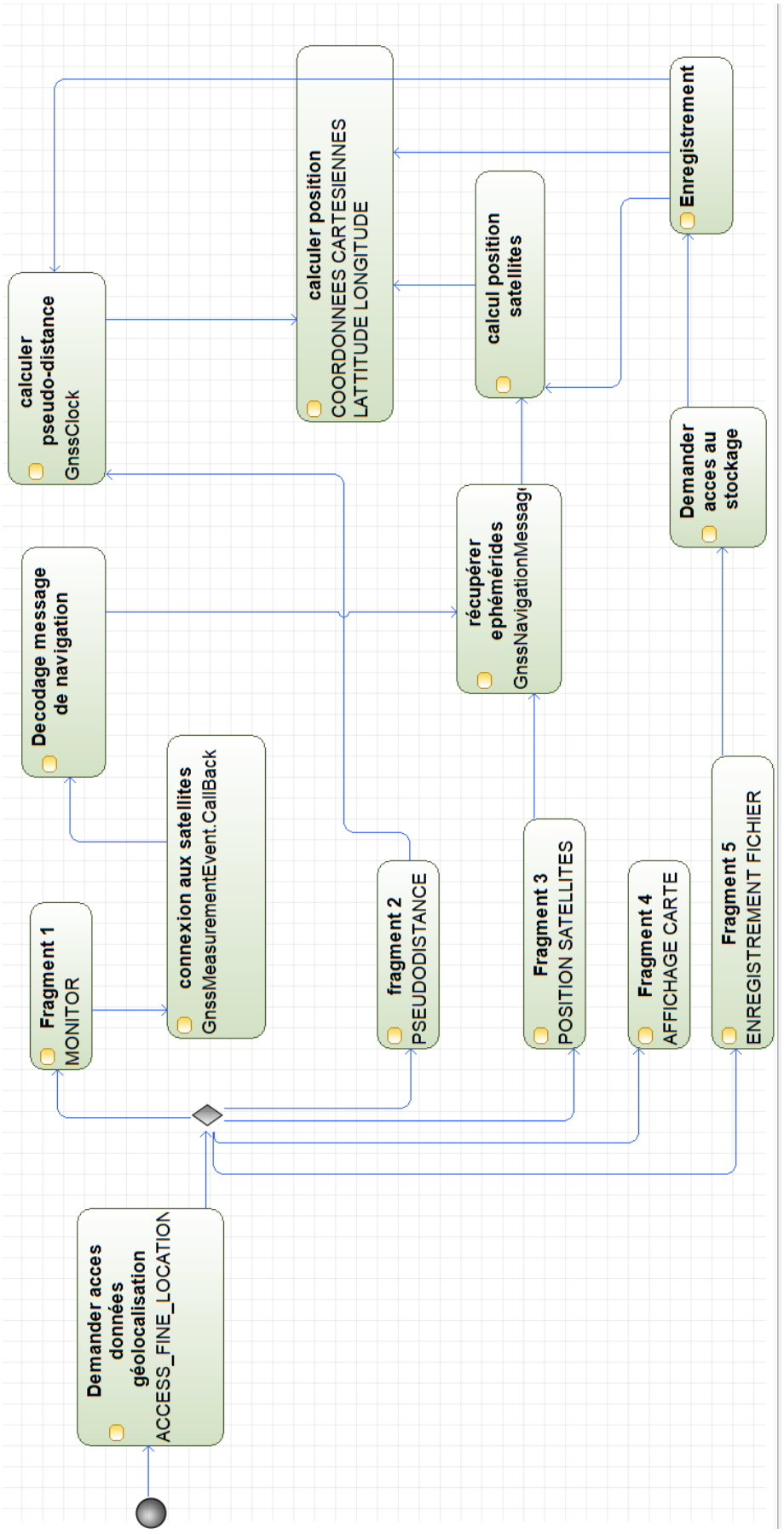
added in API level 24 ☆☆☆☆☆

```
public final class GnssMeasurement
extends Object implements Parcelable
```

java.lang.Object
↳ android.location.GnssMeasurement

A class representing a GNSS satellite measurement, containing raw and computed information.

VI. Diagramme des activités UML



VII. Travails effectués

Documentations sur les procédés GNSS

Calcul de pseudo distance

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
Android ne fournit pas directement la pseudo distance, mais tous les paramètres nécessaires pour le calculer sont fournis. Le principe, c'est basé à la différence de l'heure reçue et de l'heure transmise :

$$\rho = \frac{t_{Rx} - t_{Tx}}{1E9} * c$$

où t_{Tx} est l'heure reçue de GNSS au moment de la transmission du signal, c'est-à-dire l'heure de référence lorsque le signal a été transmis, t_{Rx} est le temps de mesure et c est la vitesse de la lumière.

t_{Tx} est fourni par Android système :

$$t_{Tx} = ReiceivedSvTimeNanos[ns]$$

t_{RxGNSS}, le temps de mesure, en temps GNSS complet, peut être reconstruit comme suivant :

$$t_{Rx_{GNSS}} = Timenanos + TimeOffsetNanos - (FullBiasNanos + Biasnanos)[ns]$$

où $Timenanos$ est la valeur du horloge à l'intérieur du récepteur GPS, $TimeOffsetNanos$ est le décalage horaire au moment de la mesure en nanosecondes, $FullBiasNanos$ est la différence entre le horloge à l'intérieur du récepteur GPS et $Biasnanos$ est le bias de l'horloge inférieur à la nanoseconde.

Pour trouver t_{Rx} à partir du temps GNSS, il y a deux façons de faire

I. $t_{Rx} = t_{Rx_{GNSS}} - weekNumberNanos[ns]$

où $weekNumberNanos$ est le nombre de nanosecondes qui compte à partir du début du temps GPS

$$weekNumberNanos = floor\left(\frac{-FullBiasNanos}{NumberNanoSecondsWeek}\right) * NumberNanoSecondsWeek$$

où $NumberNanoSecondsWeek$ est le nombre de nanosecondes dans une semaine, il vaut $604800e9$

II. $t_{Rx} = mod(t_{Rx_{GNSS}}, NumberNanoSecondWeek)[ns]$

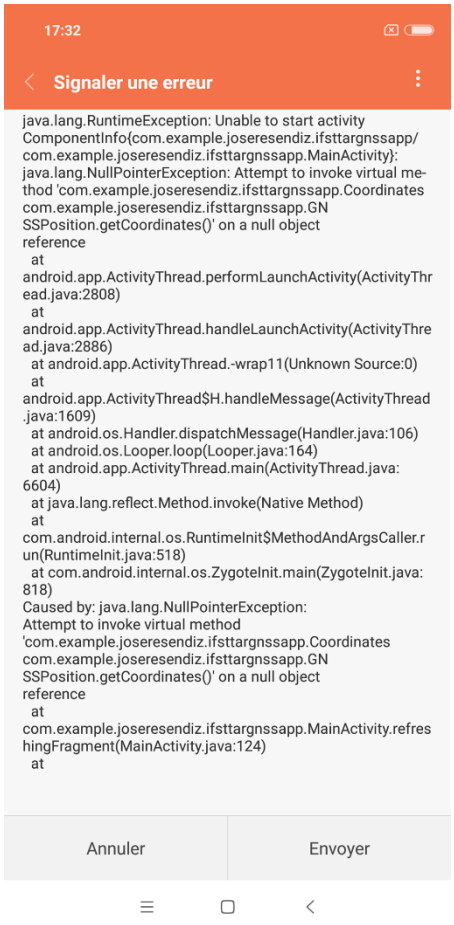
mod est l'opération de calcul du reste de la division euclidienne.
Ces deux méthodes donnent le même résultat

Ensuite, pour pouvoir résoudre l'équation qui permet d'obtenir la position de l'appareil, il nous faut connaitre les coordonnées des satellites.
Ces données peuvent être calculé à travers les éphémérides transmis dans les données GPS, l'algorithme effectuant ce calcul a déjà été implémenté durant la précédente réalisation, et donc nous allons nous baser sur cela.

Analyse du code existant et lecture du rapport du projet réalisé sur Android 7
Principalement de la documentation, nous essayions de comprendre le mécanisme implémenté, les méthodes utilisées et aussi les interactions avec le système de réception des données brutes.

Premiers tests de l'application sur Android 8 et correction des bugs

Le 1er essai de portage de l'application sans modification fait un crash au démarrage et nous donne le log suivant



.

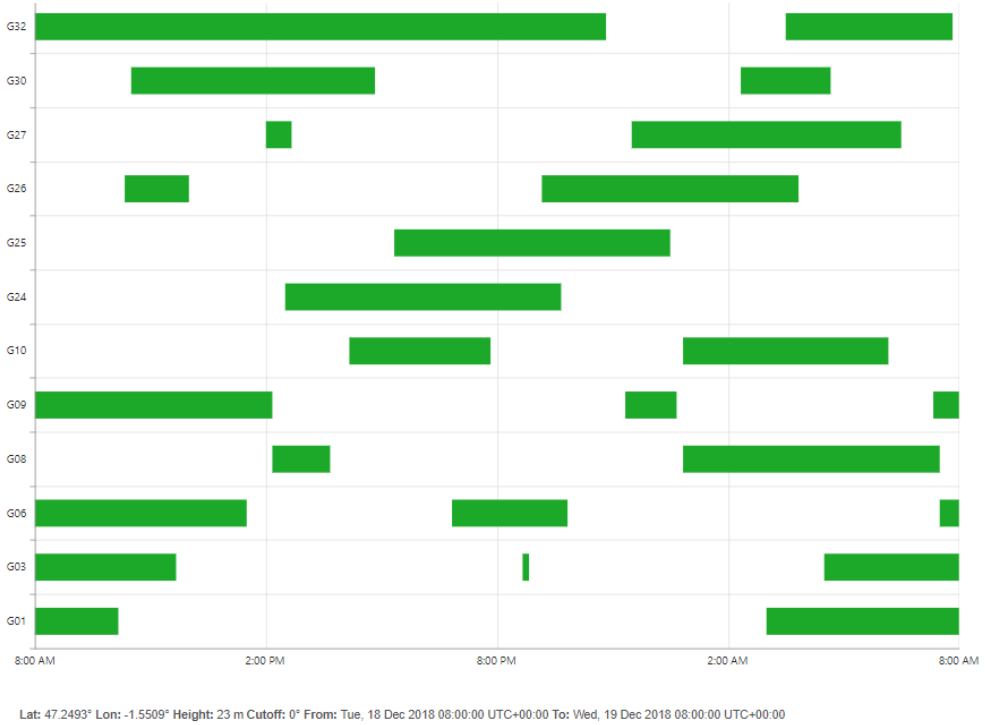
[BUG FIX] En analysant le code nous avions décelé la source du crash : quand le portable détecte moins de 4 satellites, il y a des objets qui sont vides notamment *mGnssPostion* qui est utilisé pour calculer les coordonnées du récepteur.
Pour corriger le bug, nous avons imbriqué les appels à ces objets dans des blocs try catch où on a levé l'exception NullPointerException

```
try{ co = mGNSSPosition.getCoordinates();
 bundle.putDouble("X",co.getX());
 bundle.putDouble("Y",co.getY());
 bundle.putDouble("Z",co.getZ()); }
 catch(Exception e)
 {e.printStackTrace();}
```

Et donc maintenant on a une 1ère version de l'application qui marche sans ajout de fonctionnalités

Récupération des données issues de la fréquence L5

Visibilité des satellites émettant sur la fréquence L5 en plus de L1, Lieu : Ecole Centrale de Nantes, Date 18/12/2018



Après un test de comparaison de jeu de données avec l'application, le GNSS Logger de Google et le nôtre, on a le résultat suivant :

GPS Tow	Sv	Geoloc	GnssLogger	Ecart
203630	GPS 1 L5	22047589.371	22048394.612	805,241
203630	GPS 3 L5	20407516.87	20407993.237	476,367
203631	GPS 1 L5	22048051.052	22048930.639	879,587
203631	GPS 3 L5	20407653.575	20408209.393	555,818

On soupçonne être à l'origine de ces écarts le *leap second* , cette hypothèse est en cours de vérification

VIII. Livrables

Première version de l'Application sur Android 8

La première étape de la réalisation de notre projet consiste à adapter l'application sur Android 8 en gardant toutes les fonctionnalités déjà mis en place sur la version fonctionnelle de l'application sous Android 7.

Pour cette première version, nous avions mis à jour les librairies utilisées dans Android Studio.

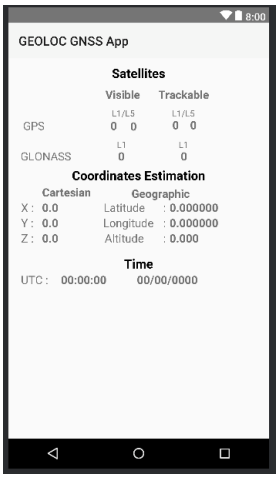
```
Ces configurations se trouvent dans le fichier build.gradle : il faut
mettre à jour les dépendances : implementation
'com.android.support:appcompat-v7:28.0.0'
```

.

Deuxième version : ajout des satellites visibles en L5 et affichage des fichiers enregistrés

[NEW FEATURES] Nombres de satellites visibles sur chaque fréquence

```
gpsVisL1.setText(String.valueOf(gpsSatellitesVisL1));
gpsVisL5.setText(String.valueOf(gpsSatellitesVisL5));
gpsTrackL1.setText(String.valueOf(gpsSatellitesTrackL1));
gpsTrackL5.setText(String.valueOf(gpsSatellitesTrackL5));
```



[NEW FEATURES] Affichage des fichiers enregistrés sous le logger fragment

Version livrée : ajout du calcul de position amélioré L1 + L5

```
294                 if (satellitePseudorange.getPseudoRange() != 0 && gnssMeasurement.getCarrierFrequencyHz()<1.3E9) {
295                     //
296                     //adding satellitePseudorange of L5 to our vector of SatellitePseudorange
297                     mSatellitePseudoranges.add(satellitePseudorange);
298                 }
299                 if (satellitePseudorange.getPseudoRange() != 0 && gnssMeasurement.getCarrierFrequencyHz()>1.3E9) {
300                     //adding satellitePseudorange of L1 to our vector of SatellitePseudoranges
301                     /**
302                      * pour tester que avec L5
303                      *
304                     mSatellitePseudorangesL1.add(satellitePseudorange);
305                      */
306                 }
```

Conclusion

Le sujet de projet qu'on nous a proposé ici en collaboration avec l'IFSTTAR a été très enrichissant de notre point de vue. Nous avions dû nous documenter sur divers procédés de calculs de positionnement, sur les formats de données GPS, le développement mobile sur Android, etc., ce qui nous a permis d'acquérir encore plus de compétence.

Nous avons pu livrer une version améliorée de l'application Geoloc sur Android 8, avec notamment la récupération des pseudodistances sur L5 et le calcul de position sur ce dernier.

Les difficultés notables que nous avions étaient surtout au début du projet où ce n'était pas forcément évident de se répartir les tâches. Heureusement nous avions des réunions d'échange avec l'équipe Geoloc de l'ifsttar qui permettait discuter de nos avancements sur le projet.

Par la suite, les améliorations qui seront intéressantes pour étendre les fonctionnalités de l'application seraient d'exploiter les autres constellations Galileo, Glonass pour les intégrer au calcul de la position et améliorer encore la précision, chiffrer les précisions, déterminer complètement les erreurs atmosphériques