Rapport De Projet

Projet 12: Mise en oeuvre d'un module GPS RTK (Real Time Kinetic)

Date : 22/05/2023
Version : 2.0
Formation/année : FISE A1
Année scolaire: 2022/2023
Destinataires :
ENCADRANT(S) : Nicholas ATTWOOD, François GALLEE, Patrice PAJUSCO
COPIL DE L'UE CODEVSI

Auteurs:
FATMI Farah
farah.fatmi@imt-atlantique.net
GRIMAUD Evan
evan.grimaud@imt-atlantique.net
ADHAR Emna
emna.adhar@imt-atlantique.net
étudiants en FISE A1, à l'IMT Atlantique

RÉSUMÉ:

Rédacteur: Emna ADHAR, relecteur: Farah FATMI

Le rapport présent détaille les étapes entreprises au cours du projet CODEVSI n°12, intitulé “Mise en œuvre d’un module GPS RTK”, réalisé sous la supervision de M. Nicholas ATTWOOD. Celles-ci comprennent une phase de contextualisation au début, avec des recherches plus ou moins approfondies de notre part, dont le fruit est restitué dans l’introduction du rapport. S'ensuit une phase d’analyse minutieuse du besoin, qui nous a permis d’éditer plusieurs versions de cahier de charges, jusqu’à en obtenir une qui satisfait le client et dont on parle en détail dans ce rapport. Y succède une étape de conception qui cherche à satisfaire tous les besoins du client, en utilisant une architecture modulaire, qui permettra non seulement de nous répartir les tâches, mais aussi et surtout de faciliter l’édition du programme en parallèle et de corriger mutuellement nos erreurs. Nous avons également veillé à diminuer le plus possible le nombre d’erreurs, en réalisant plusieurs campagnes de tests, qui aident notre produit à être le plus fiable et le plus robuste possible. Ce produit est appelé tout au long du rapport “le démonstrateur”, et est capable finalement de décoder les trames de données GNSS de plusieurs constellations de satellites, sous le format NMEA, d’afficher le trajet effectué sur un graphe, mais également d’afficher de nombreuses autres informations. Le démonstrateur possède également une interface interactive qui permet à l’utilisateur d’effectuer diverses actions.

Sommaire:

Index des Illustrations:

Index des Tableaux:

1. Introduction

Rédacteur: Farah FATMI, relecteur: Evan GRIMAUD

Depuis le début du XXIe siècle, les systèmes de navigation par satellites, connus sous le nom de GNSS (Global Navigation Satellite System), sont devenus un outil de positionnement essentiel. Initialement, ces systèmes ont commencé avec le GPS, développé par l'armée américaine dans les années 1970, mais ils sont réellement devenus opérationnels depuis le début des années 1990. Par la suite, d'autres systèmes spatiaux de même type, tels que le Glonass russe, le Beidou/Compass chinois et le Galileo européen (le seul système entièrement civil), sont également devenus pleinement opérationnels.

Aujourd'hui, les GNSS sont largement utilisés par une grande variété d'utilisateurs techniques tels que la navigation aérienne, le déplacement en automobile, les travaux des géomètres, le génie civil, ainsi que par le grand public. Les récepteurs GNSS sont aujourd'hui intégrés à presque tous les téléphones portables, permettant la navigation piétonne, la géolocalisation des photos, la navigation automobile, etc. [1]

Le fonctionnement général d'un système de géolocalisation par satellite s'explique plutôt simplement. Bien qu'il y ait plusieurs constellations différentes en orbite autour de la Terre, tous les satellites sont synchronisés sur la même temporalité. En revanche, le récepteur ne fait pas forcément partie de cette temporalité. Il y a donc quatre paramètres inconnus au niveau du récepteur: sa longitude, latitude et altitude, ainsi que son décalage temporel avec l'échelle de temps des satellites. On observe donc simultanément au moins quatre satellites, ce qui permet de calculer la position du récepteur dès que l'on connaît celle des satellites.

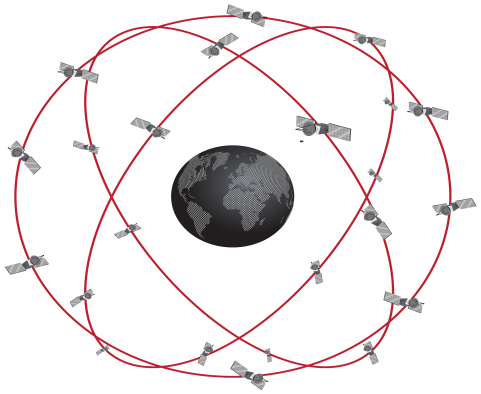

Figure 1 - La constellation de satellites GNSS

En raison des limitations physiques inhérentes au système comme les erreurs d'horloge, la géométrie des satellites, ou la diffusion des ondes dans l'atmosphère, le fonctionnement des GNSS permet généralement d'obtenir en temps réel une précision de quelques mètres. Ces performances sont généralement adéquates pour les applications militaires ainsi que pour les utilisations grand public au quotidien. Cependant, elles restent limitées en topographie, où une précision de l'ordre du centimètre est recherchée. Ainsi se présente la Cinématique Temps Réel (ou Real Time Kinematic en anglais, abrégé par RTK). C'est une

technique de positionnement par satellite basée sur l'utilisation de mesures de la phase des ondes porteuses des signaux émis par les systèmes GPS, GLONASS ou Galileo. Une station de référence fournit des corrections en temps réel permettant d'atteindre une précision de l'ordre du centimètre.[2]

Le projet trouve son cadre dans une étude menée par le département micro-ondes de l'école. Cette étude vise à analyser des canaux de propagation d'ondes en milieu ferroviaire, ce qui nécessite entre autres de connaître avec une précision accrue la position des appareils de mesure.

Par conséquent, le projet vise à mettre en œuvre cette solution avancée en développant un démonstrateur capable d'acquérir et de visualiser la position d'un récepteur en temps réel. Ce démonstrateur doit être développé en utilisant le langage de programmation Python, et doit donc permettre l'acquisition et la visualisation d'un trajet réalisé avec le module U-Blox. Pour atteindre ces résultats, nous établirons un cahier des charges détaillant les spécifications du démonstrateur, un planning de travail et un rapport technique, qui servira à expliquer en détail le fonctionnement du démonstrateur.

La première section de ce rapport présente plus en profondeur le contexte du projet, en mettant en évidence les besoins spécifiques du client. Dans la continuité de cette approche, la deuxième section fournit une explication détaillée du code que nous avons élaboré. Elle met en évidence le fonctionnement de chaque classe et de chaque méthode, offrant ainsi une vision complète du développement réalisé. Cette plongée dans la logique interne de notre démonstrateur constitue une étape cruciale pour comprendre l'ensemble du système. Enfin, la dernière section du rapport aborde l'interconnexion des différents modules et présente un protocole de validation pour vérifier la conformité aux exigences du cahier des charges. Nous analysons également les résultats obtenus et discutons des éventuels problèmes rencontrés lors de la mise en œuvre du système. Cette section nous permettra d'évaluer les performances et la conformité du démonstrateur.

Ainsi, grâce à cette structure claire et progressive, notre rapport offrira une vision globale du projet, en couvrant l'analyse initiale, le développement du code et les tests de validation.

II. Conception du Produit

II.1 Spécification du Besoin:

Rédacteur: Evan GRIMAUD, relecteur: Emna ADHAR

Dès le début du projet, le client nous a présenté ses besoins. Pour étudier les canaux de propagation des ondes en milieu ferroviaire, il est nécessaire de connaître avec précision la géolocalisation des appareils de mesure. Cette précision doit être de l'ordre du centimètre afin d'obtenir des mesures exploitables. Cependant, une géolocalisation par GNSS classique ne permet d'atteindre uniquement qu'une précision de l'ordre du mètre tout au mieux. C'est pour cette raison que le client désire une solution de géolocalisation implémentant la méthode RTK, avec laquelle il est possible d'atteindre un niveau de précision de l'ordre du centimètre. Le client souhaite également que la solution intègre un

module GNSS de la marque U-Blox, précisément le modèle C102-F9R. Ce dernier peut faire l'acquisition de trame GNSS et de données de correction RTK en continu, ce qui permet une visualisation de la position avec un délai extrêmement faible, y compris pendant un déplacement. Le client nous a également demandé une solution pour stocker et afficher un trajet effectué durant l'acquisition de données de position. Il désire également avoir accès à toute autre information pertinente contenue dans trames NMEA reçues, tel que le rapport signal à bruit de chaque satellite visible, ou encore leur position dans l'espace (azimut et élévation). Finalement, le client souhaite que la solution soit dotée d'une interface graphique lisible et prenable en main facilement, codée à l'aide de la librairie PyQT5.

II.2 Fonctions Et Contraintes:

Rédacteur: Farah FATMI, relecteur: Emna ADHAR

Nous avons, à partir de nos entretiens avec notre encadrant et de la fiche projet, défini les fonctions que notre produit devrait respecter. Celles-ci, décrites notamment dans le cahier des charges fonctionnel, sont reprises ici :

Fonction	Énoncé de la fonction	critère d'appréciation	niveau d'exigence
FP1	Extraction et traitement des données NMEA pour récupérer les coordonnées pertinentes	compréhensibilité des données	claires pour un utilisateur non spécialisé dans le domaine de télécommunications
FP2	Visualisation de la position sur un graphe superposé à une carte	précision	1 cm dans la réalité
FP3	Visualisation des positions des constellations et des propriétés de satellites	nombre des constellations	intégralité des données fournies par le GPS

Tableau 1. liste des fonctions principales, leurs critères d'appréciation et leur niveau d'exigence

(1) *Cela peut inclure des opérations telles que la conversion des coordonnées géographiques dans le système de projection approprié, ou encore le calcul de vitesses ou d'azimuts.*

On pourrait compléter le module par ces fonctions secondaires, qui sont optionnelles et assez accessoires.

Fonction	Énoncé de la fonction	critère d'appréciation	niveau d'exigence
FS1	Visualisation du trajet parcouru	précision	erreur inférieure à 2 cm

		délai entre deux positions occupées successives	1 seconde
FS2	Interagir avec l'utilisateur[2]	Facilité d'utilisation de l'interface	interface intuitive et fonctionnelle
FS3	Enregistrement des données de position [3]	taille maximale de données	700 Mo (taille d'un CD)
FS4	Calcul de la vitesse et de l'accélération	précision	à 1km/h près

Tableau 2. liste des fonctions secondaires, leurs critères d'appréciation et leur niveau d'exigence

[2] *Cela peut inclure des fonctionnalités telles que la sélection des paramètres d'acquisition, le contrôle de l'affichage graphique, l'enregistrement des données de position, etc.*

[3] *Le démonstrateur peut inclure une fonctionnalité permettant d'enregistrer les données de position acquises dans un fichier ou. Cela permet à l'utilisateur de conserver une trace du trajet réalisé pour une utilisation ultérieure ou une analyse plus approfondie.*

Tableau 3. liste des fonctions contrainte, leurs critères d'appréciation et leur niveau d'exigence

II.3 Modules:

Rédacteur : Emna ADHAR, relecteur : Evan GRIMAUD

Les fonctions citées au-dessus nous ont amenées à considérer un squelette modulaire pour notre futur produit, afin que chaque fonction soit effectuée indépendamment. Cela permet de répartir les tâches d'un côté et d'assurer la viabilité de la solution d'un autre (pouvoir corriger une erreur facilement s'il y en a, sans avoir à modifier tout le code). Il nous faut cependant se mettre d'accord sur la façon avec laquelle interagissent les modules développés les uns avec les autres, et ce en Parmi les contraintes de la réalisation de ce produit :

Contraintes	Description	critère d'appréciation	niveau d'exigence
FC1	gestion de l'acquisition de la position en un temps quasi-réel	délai maximal	2 secondes

FC2	implémentation du code sous python	conformité à l'implémentation sous Python	l'intégralité du code est sous python

précisant exactement les variables d'entrées et les variables de sorties de chaque module, comme cité dans le *tableau 4*.

Module	Description	Entrée(s)	Sortie(s)	Fonction
Décodeur	un module qui lit des trames NMEA et en extrait les informations voulues	liste de trames	des données, sur les constellations, les satellites, la localisation et la datation	**FP1**
Affichage_position	un module qui récupère les données de géolocalisation du "décodeur" et affiche la position sur un graphe	les données de géolocalisation sorties du décodeur	un graphe qui donne la position	**FP2**
affichage_constellations_satellites	un module qui affiche chaque constellation et des informations sur les satellites visibles	des données sur les constellations et les satellites	une sous-fenêtre qui détaille ces informations de façon claire	**FP3**
affichage_trajet	un module qui à partir des données restituées peut retracer tout le trajet effectué	données de positions enregistrées au fil du temps	un graphe qui retrace le chemin effectué	**FS1**
regroupement	un module qui regroupe tous les graphes sous une seule fenêtre ergonomique	les graphes faits par les autres modules	une fenêtre ergonomique et complète	**FS2**
enregistrement	un module qui permet d'enregistrer les données traitées	des données de géolocalisation récupérées instantanément	données de positions enregistrées au fil du temps	**FS3**
calcul vitesse	un module qui traite les données enregistrées pour	données de positions enregistrées au fil	vitesse calculée	**FS4**

	calculer la vitesse instantanée	du temps		

Tableau 4. liste des modules, avec leurs variables d'entrée et leurs variables de sortie respectives, en plus des fonctions qu'ils effectuent

Les relations entre les différents modules sont mieux explicitées dans la figure 2, qui vient tout après.

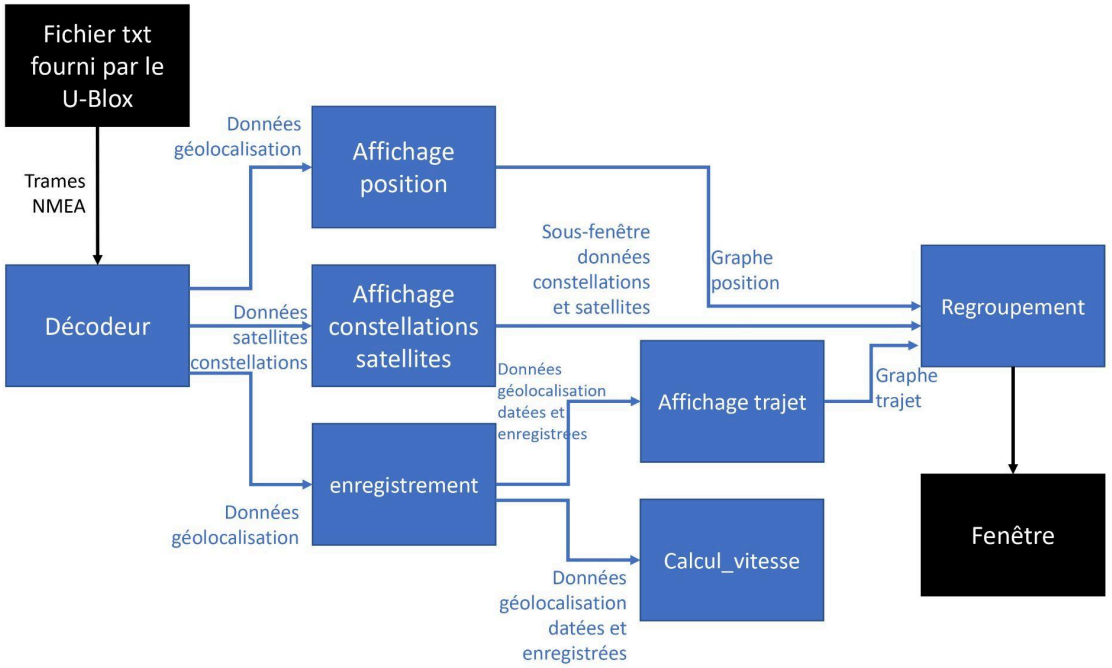

Figure 2. Relation entre les modules

III. Structuration du Programme

III.1 Démarche:

Rédacteur: Evan GRIMAUD, relecteur: Farah FATMI

De manière générale, nous avons suivi plusieurs étapes afin de structurer notre programme. Premièrement, nous avons discuté avec notre encadrant de l'intérêt d'un programme orienté objet dans le cadre de ce projet. En effet, de nombreux éléments que le programme doit manipuler sont itérables, comme les constellations et les satellites. Il est également intéressant de regrouper dans une même classe des méthodes traitant de données similaires. C'est par exemple le cas pour les conversions d'unité des coordonnées de localisation, ou encore les nombreuses opérations possibles sur l'heure et la date.
Après nous être mis d'accord sur le mode objet, nous avons (à l'aide de notre encadrant) divisé le programme en différentes classes qui nous semblaient pertinentes (voir le *diagramme des classes* section III.1.3) avant de nous les répartir entre nous. Nous pouvons regrouper les différentes classes sous trois bannières : l'extraction des données des trames,

l'exploitation de ces données et l'affichage des données exploitées. Afin de faciliter la mise en commun de nos parties de code respectives, nous avons discuté entre nous pour s'accorder sur les entrées et sorties de chacune de nos fonctions. Ainsi, l'intégration de nos trois programmes n'a pas été une réelle difficulté.
De manière plus spécifique, voici la démarche suivie pour développer un programme Python répondant aux exigences explicitées par le client :

1. *Acquisition de données de position*

- Importer la bibliothèque PyUBX2, qui permet de communiquer avec le module U-Blox.
- Établir la connexion avec le module U-Blox.
- Établir une boucle permettant l'acquisition du cycle données GNSS reçues par le module U-Blox à chaque seconde, ainsi que le stockage de ce cycle dans un fichier texte.
- Définir une fonction de décodage pour extraire les informations pertinentes des données brutes reçues, notamment les trois coordonnées géographiques latitude, longitude et altitude.[3]

2. *Traitement des données de position*

- Importer la bibliothèque math pour effectuer des opérations de traitement des données.
- Extraire des trames GGA les coordonnées géographique (en degré) du récepteur et les stocker dans deux listes Longitude et Latitude.
- Appliquer des opérations de conversion sur les coordonnées géographiques dans le système de projection souhaité, en fonction des besoins spécifiques à l'affichage de la position sur une carte ou du désir de l'utilisateur.

3. *Visualisation du trajet*

- Importer les bibliothèques de visualisation de graphe pyqtgraph
- Utiliser les bibliothèques mentionnées pour afficher sur une carte les positions successives précédemment stockées sous forme de points, permettant ainsi de visualiser le déplacement.
-

4. *Interface utilisateur conviviale*

- Utiliser la bibliothèque d'interface utilisateur PyQt5 pour créer une interface graphique lisible et prenable en main facilement.
- Concevoir une interface permettant à l'utilisateur d'interagir avec le programme, permettant notamment le lancement et l'arrêt de l'acquisition, la sélection du mode de représentation des coordonnées reçues, l'enregistrement des données de position dans un fichier texte.

5. *Respect des contraintes spécifiques*

- Optimiser le programme pour garantir une acquisition et un traitement rapides des données afin d'afficher toutes les informations en moins d'une seconde (délais d'envoie des données GNSS par les satellites).

III.2 Fonctionnement Général de l'Application:

Rédacteur: Emna ADHAR, relecteur: Farah Fatmi
Le fonctionnement général de cette application peut être décrit comme suit :

1. L'utilisateur lance l'application Python.

2. L'application établit la connexion avec le module U-Blox pour acquérir les données de position en temps réel.

3. L'application commence à recevoir les données de position à partir du module U-Blox. Elle les décode et extrait les informations nécessaires telles que les coordonnées géographiques (latitude, longitude) et l'altitude, la date et heure, les satellites, etc.

4. Les données de position sont ensuite traitées pour les rendre exploitables. Cela peut inclure des opérations telles que la conversion des coordonnées géographiques, le filtrage des données bruitées ou aberrantes, ou encore le calcul de vitesses.

5. L'application propose une interface utilisateur conviviale qui permet à l'utilisateur d'interagir avec le programme. Il peut déplacer et recadrer les sous-fenêtres, mettre en pause l'acquisition des données, changer l'unité des coordonnées affichées et revenir à l'organisation des fenêtres par défaut. L'interface utilisateur est implémentée à l'aide de la bibliothèque PyQt5.

6. L'application affiche le trajet réalisé à partir des données de position sur une carte géographique ou d'autres représentations graphiques appropriées. Cela permet à l'utilisateur de visualiser et de suivre son déplacement.

7. L'application peut également enregistrer les données de position acquises dans un fichier ou une base de données. Cela permet à l'utilisateur de conserver une trace du trajet réalisé pour une utilisation ultérieure ou une analyse plus approfondie.

8. En plus de la visualisation du trajet, l'application peut calculer et afficher des indicateurs de performance tels que lla distance parcourue, le temps écoulé, etc. Cela fournit à l'utilisateur des informations supplémentaires sur le trajet réalisé.

9. L'application continue d'acquérir et de traiter les données de position en temps réel, permettant à l'utilisateur d'interagir avec le programme et d'obtenir des informations sur son déplacement. L'utilisateur peut également arrêter l'acquisition de données et fermer l'application lorsque nécessaire.

III.3 Description des Classes et des Méthodes:

Rédacteur: Farah FATMI, Relecteur: Evan GRIMAUD

III.3.1. Diagramme des classes:

La division de notre solution en module comme vu dans la partie II.3 nous invite adopter une architecture orientée objet (Vu qu'on a les données transmises entre les modules prennent la forme d'objets séparés : satellite ,date ,localisation…), soutenue par un diagramme de classe pour mieux visualiser l'organisation et la structure du système. Nous présentons ici le diagramme de classe correspondant à l'architecture modulaire permettant de mettre en évidence les différentes classes, leurs attributs et leurs relations, offrant ainsi une vue claire de la conception du système.

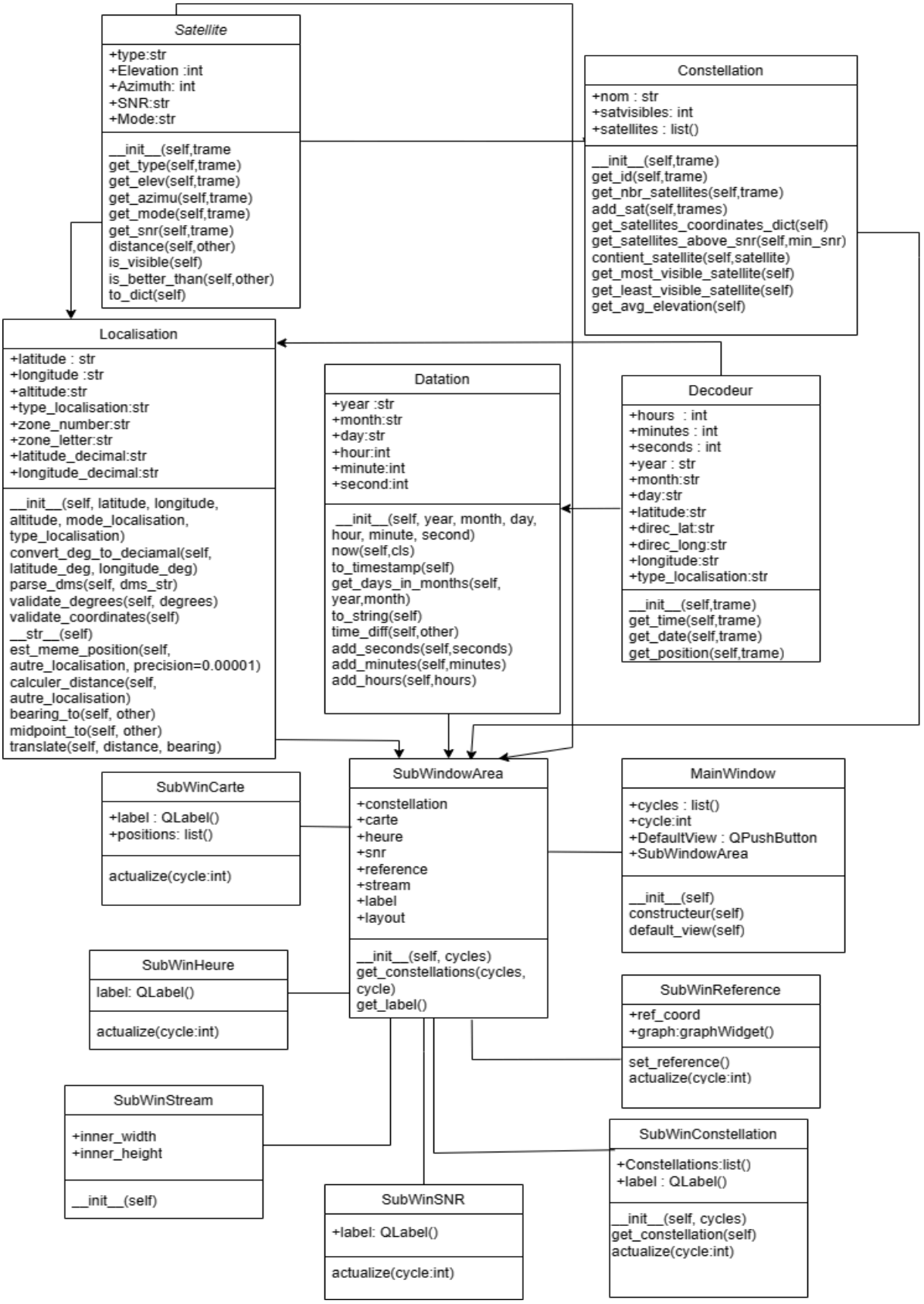

Figure 3. Diagramme des classes

III.3.2.Extraction des données:

Rédacteur: Farah FATMI, relecteur : Evan GRIMAUD

- L'extraction des données des classes **Datation** et **Localisation** se fait principalement via la classe **Décodeur**. En effet , cette classe parcours la trame et y extrait les différents attributs de la datation et la localisation notamment :
    - hours, minutes, secondes via la méthode **get_time()**
    - year, month ,day en utilisant la méthode **get_date()**
    - latitude, longitude, altitude, type_localisation, zone_number, zone_letter , latitude_decimale, longitude_decimale avec la methode **get_localisation()**
    - Pour les classes **Constellation** et **Satellite** , on a opté pour une extraction de données locale . Chacune de ces deux classes parcours la trame et y extrait ses propres attributs, plus précisément:
    - type : le numéro indiquant le type du satellite avec la méthode **get_type(trame)**
    - Elevation: L'élévation du satellite en degrés (90° maximum) via **get_elev(trame)**
    - Azimu: L'azimuth du satellite en degrés(de 000° à 359°) avec **get_azmu(trame)**
    - SNR: Le SNR du satellite en dB (de 00dB à 99dB) avec **get_snr(trame)**
    - mode: le mode du satellite (Differentiel , autonomous,RTK ….) avec **get_mode(trame)**
    - nom : l'id de la constellation (GPS , GALILEO,GLONASS…) via **get_id()**
    - satvisibles: le nombre des satellites visibles dans la constellation avec **get_nbr_satellite()**

III.3.3 - Traitement des données:

Rédacteur: Farah FATMI, relecteur: Evan GRIMAUD

Après l'extraction des données , on passe aux méthodes permettant le traitement de ces données qui serviront notre interface .

- Pour la classe *Constellation*:
    - La méthode **add_sat()** permet d'ajouter la satellite contenant dans la trame dans la liste des satellites de la constellation
    - On peut obtenir la moyenne des élévations de tous les satellites de la constellation avec la méthode **get_avg_elevation ()**
    - On peut comparer la visibilité des satellites de la constellation en utilisant **get_most_visible_satellite()** pour avoir le satellite le plus visible ou **get_least_visible_satellite()** pour obtenir le moins visible dans la constellation
    - Finalement la méthode **get_satellites_above_snr(min_snr)** qui prend en entrée une valeur de snr minimum permettant de déterminer les satellites de la constellation qui ont un certain niveau de signal bruité minimum.

- Pour la classe *Satellite* :

- La méthode **to_dict()** permet d'obtenir un dictionnaire contenant toutes les propriétés du satellite.
- La méthode **distance(other)** prend en paramètre un autre satellite. Elle permet de calculer la distance entre les positions des deux satellites
- La méthode **is_visible()** vérifie si le satellite est visible (elevation > 0)
- Finalement, **is_better_than(other)** prend en paramètre un autre satellite. Elle compare les SNR des deux satellites pour déterminer lequel est le meilleur.

- Pour la classe *Datation*:
  - La méthode **get_days_in_month(year, month)** permet d'obtenir le nombre de jours dans un mois donné.
  - La méthode **now(cls**) permet de créer une instance de la classe à partir de l'heure courante.
  - La méthode **to_string()** sert à convertir la date et l'heure en une chaîne de caractères au format ISO 8601.
  - **to_timestamp()** est une méthode pour convertir la date et l'heure en timestamp UNIX. On l'utilisera pour la méthode suivante.
  - **time_diff(other)** est une méthode pour calculer la différence de temps entre deux datations.
  - Les méthodes **add_seconds(seconds), add_minutes(minutes), add_hours (hours)** et **add_days(days)** permettent respectivement d'ajouter des secondes, des minutes, des heures et des jours à la date ou l'heure actuelle.

- Pour la classe *Localisation*:
  - **convert_deg_to_decimal(latitude_deg, longitude_deg)** est une methode qui permet la conversion des coordonnées du degré en décimal.
  - La méthode **parse_dms(dms_str)** analyse une chaîne de caractères représentant les coordonnées en degrés, minutes et secondes (DMS) et retourne une liste de valeurs [degrés, minutes, secondes, direction]. (Exemple : '15°52'48''N' -> [15, 52, 48, 'N']).
  - **validate_degrees(degrees)** vérifie les valeurs des degrés, minutes et secondes pour s'assurer qu'elles sont valides.
  - Pour vérifier les coordonnées de latitude et longitude afin de s'assurer qu'elles sont valides, on utilise la méthode **validate_coordinates().**
  - **__str__()** est une méthode qui retourne une représentation en chaîne de caractères de l'objet (la datation) .
  - La méthode **est_meme_position (autre_localisation, precision=0.00001)** vérifie si la localisation est la même qu'une autre localisation, avec une certaine précision.
  - La méthode **calculer_distance (autre_localisation)** calcul la distance entre deux points (la position actuelle et autre localisation) en utilisant la formule de Haversine.
  - **bearing_to(other)** est une méthode qui calcul l'azimut (angle de direction) entre deux points : la position actuelle et other.

- **midpoint_to(other)** prend en paramètre un point (ses coordonnées) dont elle va calculer le point milieu avec le point actuel (la position).
- **translate (distance, bearing)** est une méthode qui permet le calcul de la nouvelle position après translation d'une distance donnée dans une direction donnée.

III.3.4 - Interface Utilisateur

Rédacteur: Evan GRIMAUD, relecteur: Farah FATMI

Afin d'afficher toutes les informations relatives au projet et de permettre à l'utilisateur d'interagir avec elles, nous avons utilisé la librairie python PyQT5, sur conseil de M. Nicholas ATTWOOD. Voici les différentes classes qui composent cette partie :

- La classe **MainWindow** est l'entité maître du programme puisqu'elle représente la fenêtre d'affichage principale. Elle contient une instance de **SubWindowArea**, ainsi que le bouton **DefaultView**. C'est aussi elle qui initialise la liste **cycles** grâce à sa méthode **constructeur**.
  - La méthode **constructeur()** permet d'initialiser la liste **cycles**, qui contient l'ensemble des trames du fichier **Deplacement_IMT**, en définissant chaque cycle comme un élément de cette liste. Au final, **cycles** est donc une liste de listes de chaîne de caractères.
  - La méthode **default_view()** est appelée quand l'utilisateur presse le bouton DefaultView. Elle a pour but de revenir à la vue par défaut, c'est-à-dire revenir à l'état de l'interface tel que lorsque l'on lance le programme.
  - La méthode **play_pause()** est appelée en cliquant sur le bouton Play, elle enclenche et met en pause la lecture des données reçues.
  - La méthode **switch_loc()** est appelée en cliquant sur le bouton Switch Localisation Type, elle permet de changer le type d'unité des coordonnées.
  - La méthode **actualize_all()** qui appelle toutes les méthodes d'actualisation afin de rafraîchir l'affichage global.
- La classe **SubWindowArea** représente la zone de la fenêtre principale où s'affichent toutes les sous fenêtres (détaillées juste après). Lors de son initialisation, elle crée et ajoute les six sous fenêtres de l'afficheur.
- La classe **SubWinConstellation** est la sous fenêtre chargée d'afficher les informations relatives aux constellations de satellites. Elle est constituée de deux méthodes:
  - La méthode **get_constellations(cycle)**, qui crée les instances des constellations du cycle et les stock dans la liste **constellations**, puis ajoute chaque satellite dans sa constellation.
  - La méthode **actualize(cycle)** qui appelle **get_constellation** puis rafraîchit l'affichage de la fenêtre en plaçant sur un graphe la position de chaque satellite visible.
- La classe **SubWinCarte** est la sous fenêtre chargée d'afficher les différentes positions relevées en superposition d'une carte de type Maps (calque satellite) ou OpenStreet. Elle est constituée d'une méthode :
  - La méthode **actualize(line)** qui rafraîchit l'affichage de la fenêtre, en écrivant les nouvelles coordonnées. Elle ajoute également ces coordonnées à un

fichier texte afin de sauvegarder les données sous le format choisi (décimal ou degré).

- La classe **SubWinHeure** est la sous fenêtre chargée d'afficher l'heure à laquelle sont reçues les trames GNSS. Elle est constituée d'une méthode :
  - La méthode **actualize(line)** qui rafraîchit l'affichage de la fenêtre.
- La classe **SubWinSNR** est la sous fenêtre chargée d'afficher le rapport signal à bruit (SNR en anglais) des satellites dont on reçoit les trames. Elle est constituée d'une méthode :
  - La méthode **actualize()** qui rafraîchit l'affichage du graphe avec les nouvelles valeurs du SNR de chaque satellite.
- La classe **SubWinReference** est la sous fenêtre chargée d'afficher un graphe illustrant la marge d'erreur des mesures satellites, par rapport à un point de référence, c'est-à-dire dont on connaît la position avec une grande précision. Elle est constituée de deux méthodes :
  - La méthode **actualize()** qui ajoute un nouveau point sur le graphe, représentant les nouvelles coordonnées reçues.
- La classe **SubWinStream** est la sous fenêtre chargée d'afficher des informations relatives au paquet de trames reçu à chaque seconde (le cycle).Elle est constituée d'une méthode :
  - La méthode **actualize(cycle)** qui affiche le nouveau jeu de trame reçu.

Finalement, voici le rendu actuel du démonstrateur :

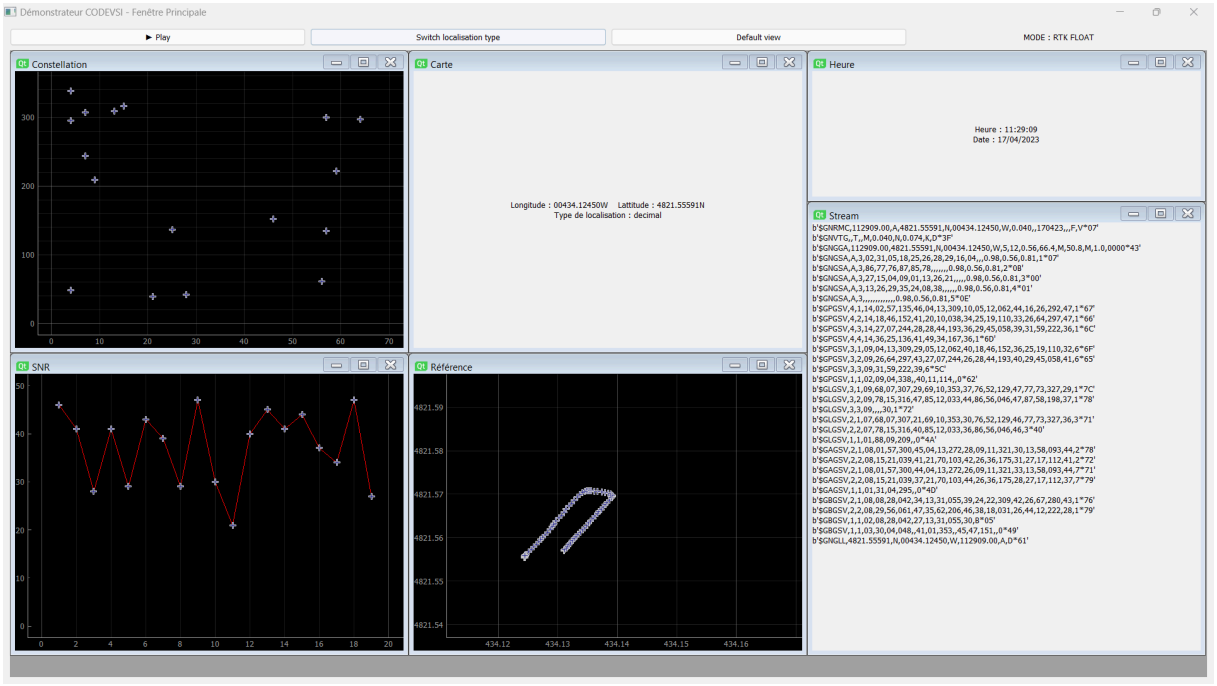

Figure 4 - Capture d'écran du démonstrateur

IV. Validation:

IV.1 Phase des Tests Unitaires:

Rédacteur: Farah FATMI, relecteur: Emna ADHAR

Dans le processus de validation du prototype, une étape cruciale consiste à effectuer des tests unitaires approfondis. Ces tests visent à vérifier la fiabilité et la robustesse de chaque composant du système. Nous expliquons ici comment nous avons mis en œuvre ces tests pour valider la solution.

Tout d'abord, nous avons réalisé un test de connexion avec le module U-Blox. L'objectif était de s'assurer que l'application établisse avec succès la connexion avec le module U-Blox pour acquérir les données de position en temps réel. Ce test a permis de valider la capacité du système à communiquer efficacement avec le module et à récupérer les données nécessaires.

Ensuite, nous avons effectué des tests d'acquisition du signal GNSS. Cela impliquait de vérifier si le système GNSS était capable de détecter et d'acquérir les signaux des satellites. Ces tests ont été réalisés dans différents environnements pour évaluer la performance du système dans des conditions variées. Par exemple, nous avons même effectué des tests sous la pluie pour simuler des conditions météorologiques défavorables.

Pour valider le code implémenté, nous avons créé des fonctions de test pour chaque classe du système. Ces fonctions de test ont été conçues pour vérifier la validité des résultats obtenus à partir des méthodes de chaque classe. Cela nous a permis de réaliser une validation locale approfondie en testant minutieusement chaque aspect du code et en réfléchissant aux cas extrêmes afin d'aboutir à des solutions précises. En se basant sur les résultats de ces tests , on a pu remanier notre code et à la fin aboutir à des solutions plus satisfaisantes .

Les tests unitaires ont encore un avantage précieux: celui de faciliter la maintenance du système lors d’un dysfonctionnement. En effet, grâce à l'utilisation d'un test_runner, il devient facile d'exécuter l'ensemble des tests unitaires et d'obtenir des résultats détaillés sur leur exécution. Si une erreur se produit, le test_runner fournira des informations précises sur la méthode qui a échoué, ce qui facilite grandement le processus de débogage et de correction, comme le montre la figure 5.

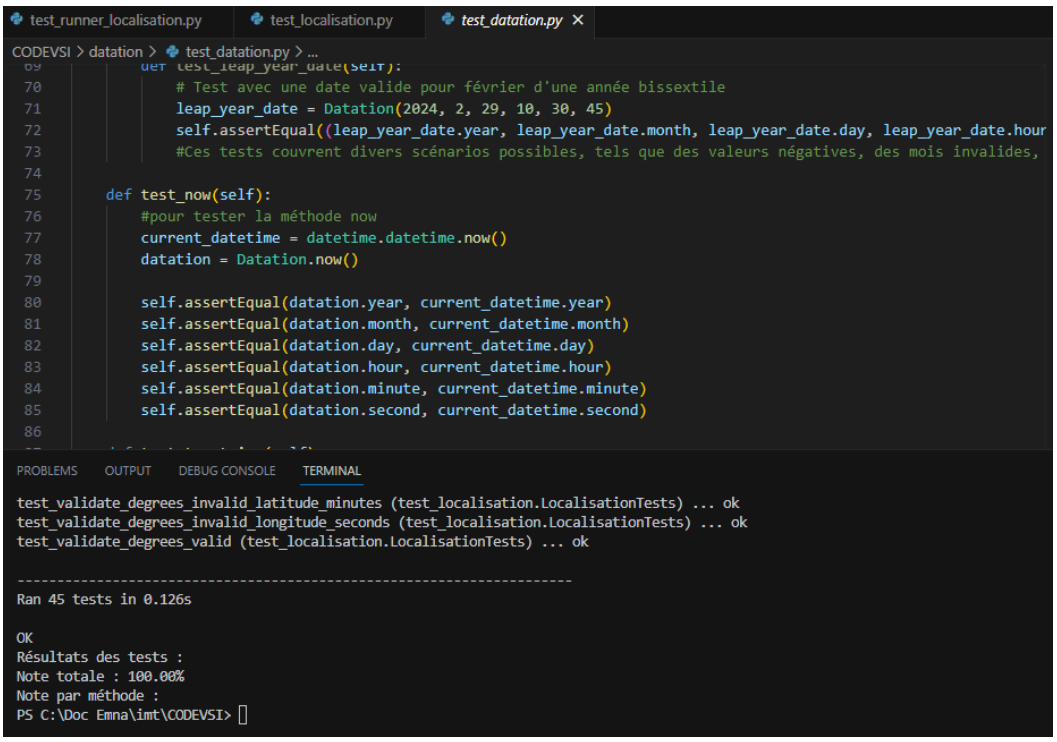

Figure 5 - Capture d'écran d'un des tests runners

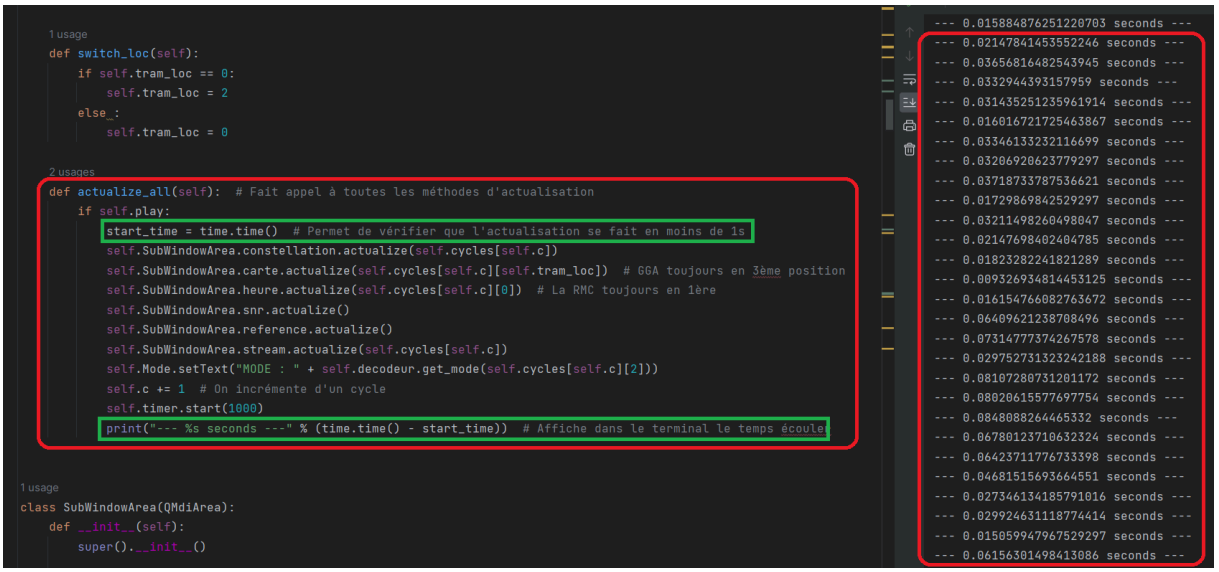

Figure 6 - Capture d'écran de test du temps d'actualisation du démonstrateur

IV.2 Phase des Test Globale:

Rédacteur: Farah FATMI, relecteur: Emna ADHAR

Le premier test de fonctionnement global que nous avons effectué est celui du temps d'exécution. En effet, puisque l'on reçoit les données GNSS à raison d'un cycle par seconde, il est nécessaire que l'application s'actualise entièrement durant ce laps de temps. Heureusement, grâce à la programmation orientée objet, l'actualisation de toute l'application se résume à la courte fonction *actualize_all* de la fenêtre principale. Nous avons pu mesurer le temps d'exécution de cette fonction avec la librairie Time et sa méthode *time* (voir annexe n°4). Le résultat est très satisfaisant, puisque malgré les nombreux éléments à afficher, le temps d'exécution de la fonction ne semble jamais excéder 0.1 seconde.

Malheureusement par manque de temps, nous n'avons pas encore été en mesure d'effectuer d'autres tests de fonctionnement globaux de l'application, et donc de sa correspondance aux attentes du client. Cependant, nous présentons ici une batterie de tests qu'il serait intéressant de faire afin de valider le bon fonctionnement de notre programme, et nous espérons pouvoir les faire très prochainement.

Un moyen de validation consiste à placer un téléphone mobile au même endroit que l'antenne GNSS. En comparant les positions enregistrées par le téléphone mobile avec celles du système GNSS, il est possible de vérifier la cohérence et la précision des données obtenues. Cela permettra de valider la fiabilité du système.

Une autre méthode de validation importante est la comparaison avec les résultats RTK. Les résultats obtenus à partir du démonstrateur doivent être comparés aux données fournies par une solution RTK, qui offre une précision de l'ordre du centimètre. Cette comparaison permettra de valider la précision du système GNSS et de s'assurer que les positions enregistrées sont cohérentes avec les attentes.

La vérification de l'interface est également un aspect essentiel. L'application doit pouvoir afficher un grand nombre d'éléments, dont quatre graphiques, et l'utilisateur doit pouvoir interagir avec elle. Afin de vérifier le bon fonctionnement de l'application, nous pouvons tester manuellement chacune des fonctionnalités sur le fichier texte "Deplacement_IMT" fourni par notre encadrant, puisque ce fichier représente un jeu de données GNSS type dans ce projet. Nous pouvons également essayer de rentrer des valeurs aberrantes dans l'espace de saisie utilisateur dédié à la référence, et s'assurer que cela ne déclenche pas une erreur dans le programme.

Enfin, il est important de valider l'intégrité des données fournies par le système GNSS. Cela implique de vérifier si les informations telles que l'heure, la date, les vitesses et les altitudes sont correctement enregistrées et cohérentes. Cette vérification permettra de s'assurer de la fiabilité et de la validité des données fournies par le système.

Plusieurs de ces tests et validations sont inspirés par les fonctions principales, secondaires et contraintes du projet. En effet, en cherchant à satisfaire les niveaux d'exigence de ces derniers, on garantit la fiabilité, la précision et l'intégrité du système.

V. Conclusion

Rédacteur: Emna ADHAR- Relecteur: Evan GRIMAUD

Malgré la nouveauté des concepts et la difficulté relative du projet, nous avons aboutit à une solution que nous espérons présentable, ainsi qu'à la hauteur des attentes du client. Les fonctions principales ont été intégralement remplies, les contraintes et exigences ont été respectées; nous avons même réussi à réaliser quelques fonctions secondaires. Le périple a certes été jalonné de difficultés, mais on en a beaucoup appris: à se remettre en question, à ne pas hésiter de révolutionner, de réinventer, à chercher à se dépasser, à avoir le courage de tout mettre à côté et à recommencer de zéro. On a aussi et surtout appris à travailler ensemble, à être plus tolérants, plus respectueux l'un envers l'autre, à s'entendre et s'entraider. On a aussi appris à prendre en compte les demandes de notre client, même si elles nous paraissent au début trop exigeantes, mais on a su les utiliser pour améliorer notre travail, et ceci a bien payé. Ainsi, ce projet n'a pas pour seule valeur ajoutée une solution informatique, mais des compétences qui seront précieuses aux futurs ingénieurs que nous deviendrons un jour.

Références Bibliographiques

[1]Michel KASSER - Du GPS historique aux GNSS : utilisation pour le positionnement de haute précision

[2] PDF- Le réseau GNSS en temps réel Teria

[3] NMEA-0183 messages: Overview (trimble.com)

GLOSSAIRE

Azimuth du satellite : l'angle formé par le nord géographique et la direction du satellite

Constellation : ensemble de satellites d'un même groupe en orbite autour de la terre.

Elévation du satellite: l'angle formé entre l'horizontale du point de réception une ligne reliant ce point au satellite

FC : Fonction Contrainte

Format ISO 8601: La partie date suit le format suivant YYYY-MM-DD (année-mois-jour).Si une heure ISO est incluse, l'heure et le fuseau horaire sont ajoutés à la date qui suit l'indicatif T , cela donne par exemple : 2016-06-01T14:41:36-08:00.

Formule de Haversine: permet de déterminer la distance du grand cercle entre deux points d'une sphère, à partir de leurs longitudes et latitudes.

FP : Fonction Principale

FS : Fonction Secondaire

GNSS : global navigation satellite systems, donne la position d'un élément partout et en temps réel

GPS : Global Positioning System, un système de positionnement par satellites appartenant au gouvernement fédéral des États-Unis.

NMEA : National Marine & Electronics Association, est une association à but non lucratif fondée par un groupement de professionnels de l'industrie de l'électronique des périphériques marine, conjointement avec des fabricants, des distributeurs, des revendeurs, des institutions d'enseignements. Leur but entre autres, harmoniser et standardiser les équipements de la marine.

Rapport signal à bruit (SNR) : c'est le rapport entre la puissance du signal (l'information utile) et celle du bruit (aucune information). C'est un indicateur de la qualité de la transmission d'une information.

RTK : Real Time Kinetic

timestamp UNIX : une mesure du temps fondée sur le nombre de secondes écoulées depuis le 1er janvier 1970 00:00:00 UTC, hors secondes intercalaires.

U-Blox : une société suisse qui crée des semi-conducteurs et des modules sans fil pour les marchés grand public, automobile et industriel.

ANNEXES

1) Planning final du déroulement du projet :

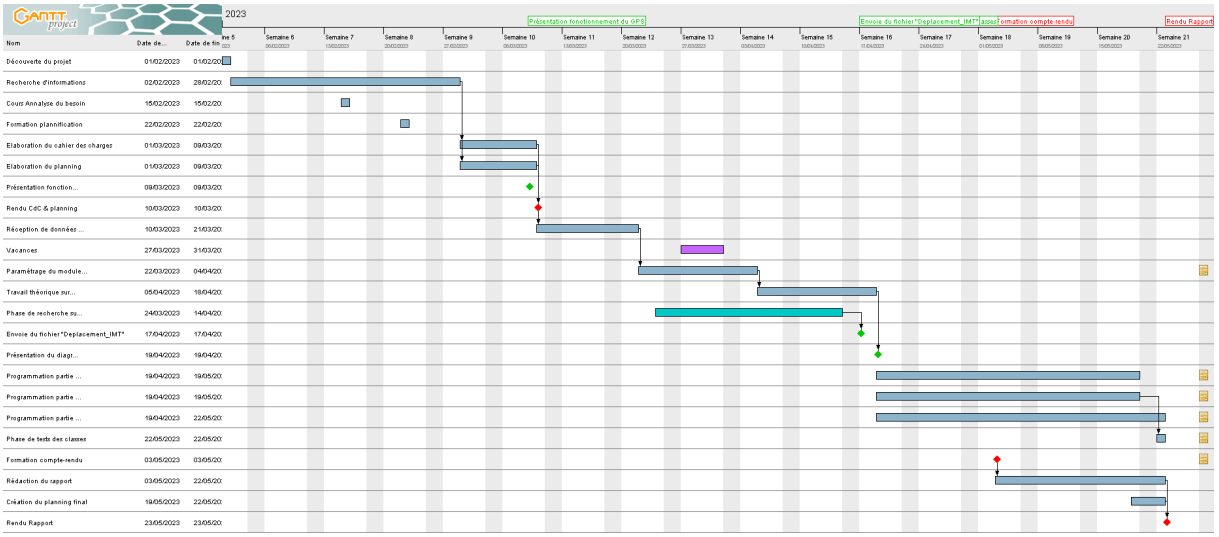

Figure 7 - Planning final

2) Planning initial du déroulement du projet :

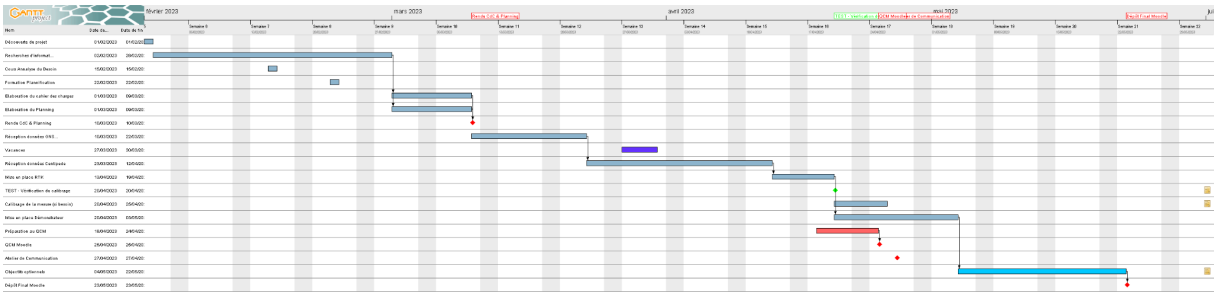

Figure 8 - Planning prévisionnel initial

3) Analyse de l’écart entre les deux plannings :

Nous pouvons remarquer plusieurs différences entre les deux plannings. Premièrement, certaines tâches présentes sur le planning initial (figure n°5) n’apparaissent pas sur le planning final (figure n°4), tandis que d’autres n'ont pas été anticipées. Cela peut s’expliquer par une mauvaise estimation des difficultées qui allaient être rencontrées durant le projet, ainsi que de la méthodologie de travail. Par exemple, nous n’avions pas prévu que nous diviserions le programme en trois grandes parties avant de les mettre en commun. Cela explique la phase de développement en parallèle sur le planning final, alors qu’elle apparaît comme unifiée dans le planning initial. Nous n’avions pas non plus prévu de phase de réflexion sur la conception du programme, alors que cette étape nous a pourtant beaucoup aidé à nous organiser pour coder rapidement et sans encombre. Finalement, nous avions été un peu trop optimistes sur le temps que prendrait la réalisation de chaque tâche, ce qui

explique que nous n’avons pas pu réaliser d’objectif optionnels, comme par exemple effectuer une cartographie du campus à l’aide du démonstrateur et du module U-Blox.