Développement d'un système d'estimation de trajectoire pour activités sportives

BAVOUX Thomas, BELLILI Fouad, JOUANNEAU Maxime,
LENOBLE Louis, ETIENNE Maxime
Encadrants :
M.Condat
M.Vaugeois
Mme.Rogozan

28 mai 2025

Table des matières

Résumé

Ce projet vise à concevoir un système d'acquisition et d'estimation de trajectoire pour activités sportives, dans l'esprit de l'application mobile Strava. L'objectif principal est de permettre à un utilisateur de visualiser son parcours à pied sur une carte, tout en exploitant plusieurs capteurs embarqués pour améliorer la précision de la trajectoire estimée.

Le système repose sur l'utilisation d'une Raspberry Pi couplée à des capteurs tels qu'un GPS, une centrale inertielle (IMU), et un LiDAR. La finalité de ce projet est d'utiliser un ensemble d'outils statistiques ainsi que des connaissances sur les capteurs que nous avons abordés en cours afin de pouvoir calibrer, synchroniser et fusionner au mieux les données issues de ces capteurs.

Par la suite, nous proposerons l'implémentation de fonctionnalités supplémentaires telles que le calcul de distance parcourue, un compteur de pas ou encore des modes de course spécifiques.

Chapitre 1

Introduction

1.1 Contexte

Durant la 3ème année de notre cursus à l'INSA Rouen Normandie, nous devons effectuer un Projet Intégratif (PI). Celui-ci a pour but de mettre en pratique l'ensemble des compétences que nous avons vues en cours. Ici, nous devons mettre en place un système d'acquisition permettant l'estimation de trajectoire d'un parcours. Pour cela, nous pouvons définir un cahier des charges comme suit :

- Les capteurs et les données recupérées doivent être en accord avec la finalité du projet (pas de capteurs inutiles et des capteurs biens calibrés) ;
- Le système doit pouvoir récupérer des données exploitables avec une bonne fréquence d'acquisition ;
- Le système doit effectuer un pré-traitement des données ;
- Le système doit pouvoir synchroniser les données ;
- Le système doit fusioner et exploiter les données ;
- Les résultats doivent pouvoir être enregistrés et affichés sur une carte OpenStreetMap ;
- Le système doit proposer des fonctionnalités supplémentaires utiles ou divertissantes ;

1.2 Organisation du rapport

Afin de vous présenter notre travail, nous découperons ce rapport en plusieurs parties. Dans un premier temps, nous vous présenterons les capteurs que nous avons utilisés ainsi que nos choix pour les paramétrer au mieux. Ensuite, nous vous décrirons le système d'acquisition et vous détaillerons les grandes étapes de la fusion de données. Après cela, nous vous présenterons les analyses des performances que nous avons obtenues ainsi que notre méthode de traitement des données. Pour finir, nous vous présenterons les fonctionnalités supplémentaires que nous avons développées.

Chapitre 2

Présentation des capteurs et traitement des signaux (Partie TP)

2.1 Présentation des capteurs à disposition

2.1.1 Capteur GNSS (TEL0157)

Description et fonctionnement

Le module GNSS TEL0157 de DFRobot est un capteur de positionnement géographique compatible avec les constellations GPS, GALILEO, GLONASS et BEIDOU. Il se connecte à une Raspberry Pi 4 via l'interface I2C, permettant une communication simple et rapide.

Ce capteur émet des trames au format NMEA, un protocole standardisé utilisé dans les systèmes de navigation. Chaque trame débute par un identifiant (comme `GGA` ou `RMC`) et contient des données telles que la latitude, la longitude, l'altitude, l'heure ou encore la qualité de réception.

Dans le cadre du TP, ces trames ont été traitées avec des scripts Python. Les coordonnées extraites ont été affichées en temps réel sur un écran LCD 1602 et enregistrées dans des fichiers CSV, permettant un post-traitement efficace et une visualisation du trajet sur une carte.

Calcul de distance

Le calcul de la distance entre deux points GNSS a été réalisé en utilisant la formule de la distance orthodromique, qui tient compte de la courbure de la Terre. Les coordonnées, exprimées en degrés décimaux, ont été converties en radians pour appliquer la formule suivante :

$$D = R \cdot \arccos\left(\sin(\varphi_A) \cdot \sin(\varphi_B) + \cos(\varphi_A) \cdot \cos(\varphi_B) \cdot \cos(\lambda_B - \lambda_A)\right)$$

où R est le rayon moyen de la Terre (environ 6371 km), φ la latitude et λ la longitude.

Cette méthode a été implémentée dans le script `get_distance.py`, qui permet de calculer la distance entre la position courante du capteur et un point d'intérêt, comme l'entrée d'un bâtiment ou un point de repère extérieur. Les tests effectués avec des coordonnées réelles (New York, Paris, Rouen, etc.) ont permis de vérifier la cohérence des résultats obtenus.

Une expérience a été menée pour se déplacer jusqu'à un point connu, l'arche située à proximité du bâtiment Dumont D'Urville (coordonnées : 49.384210, 1.069372). La distance affichée diminuait progressivement à mesure que nous nous approchions du point cible, avec une précision acceptable compte tenu des limitations du module. Cette expérience a confirmé l'utilité du capteur pour des calculs de distances en vol d'oiseau dans un contexte réel.

Avantages et inconvénients

Le principal atout du TEL0157 est sa compatibilité multi-constellation, qui assure une bonne couverture satellite et une meilleure stabilité des données, surtout en environnement dégagé. Sa connexion I2C le rend facilement intégrable à des systèmes embarqués comme la Raspberry Pi.

Cependant, ce capteur présente des limites. En intérieur ou en présence d'obstacles (bâtiments, végétation), la réception des signaux peut devenir instable, voire inexistante. De plus, les mouvements rapides dégradent la qualité des mesures, rendant certaines positions incohérentes ou bruitées.

Sources de bruit et défauts techniques

Plusieurs phénomènes affectent la précision des mesures GNSS :

— **Propagation atmosphérique** : les signaux traversent l'ionosphère et la troposphère, ce qui introduit des erreurs de délai.
— **Effet multitrajet** : le signal peut être réfléchi par des surfaces (murs, vitres...), créant des interférences.
— **Perturbations électroniques** : un câblage inadéquat ou la proximité d'autres composants électroniques peut générer du bruit.

Ces facteurs sont à prendre en compte lors de l'analyse des données, surtout dans un contexte de mesure fine.

Calibration effectuée

Afin de pouvoir exploiter les données du GPS, nous avons utilisé le protocole I2C. Pour cela, nous avons utilisé la commande `i2cdetect` afin de trouver les périphériques connectés à la carte. Ensuite, un programme s'occupe de lire la trame NMEA afin de récupérer et convertir les coordonnées au format **Degrés Décimaux (DD)**.

Pour calibrer le système nous avons dans un premier temps récupéré des données via ce capteur GPS mais aussi via un téléphone (considéré comme parfait). Ensuite, pour connaitre l'erreur statique de ce capteur, nous avons étudié la différence moyenne entre nos données et les références pour la latitude mais aussi pour la longitude.

Afin de corriger cette erreur, nous soustrayons le résultat obtenu à l'ensemble des futures données.

Choix des paramètres d'acquisition

En ce qui concerne le GPS, nous ne pouvons pas configurer les paramètres d'acqusition. Par défaut le système effectue une acquisition par seconde. Cependant, cela n'empêche pas d'assurer un bon compromis entre précision temporelle et volume de données. En effet, au vu de l'utilité de ce capteur (pour de la marche), une acquisition par seconde reste satisfaisant.

Ensuite, nous avons choisit d'utiliser le format de coordonnées **Degrés Décimaux (DD)**, car il est plus adapté aux calculs de distance et à la cartographie.

Finalement, le format CSV a été utilisé pour l'enregistrement, facilitant ainsi l'analyse postérieure des données.

Filtrage des signaux bruts

Un traitement logiciel a été mis en place pour filtrer les données :

— Seules les trames NMEA valides ont été conservées.

— Les points aberrants (valeurs nulles, sauts de position) ont été écartés. Nous vous expliquerons plus en détail notre méthode dans la partie sur le traitement des données.

Les données ainsi nettoyées peuvent être visualisées sur carte via `folium`. Dans le cadre du TP, lsssssa trajectoire obtenue est globalement fidèle au parcours réel, bien que quelques imprécisions subsistent par endroits.

2.1.2 Accéléromètre et Gyroscope (MinIMU-9 v6)

Description et fonctionnement

Le capteur MinIMU-9 v6 est une centrale inertielle miniature intégrant deux composants : un accéléromètre et un gyroscope **LSM6DSO**, ainsi qu'un magnétomètre **LIS3MDL**. Ces deux éléments communiquent indépendamment via le bus I2C et possèdent donc des adresses distinctes. Dans le cadre de notre TP, nous avons principalement travaillé avec le LSM6DSO, responsable de la mesure de l'accélération linéaire et de la vitesse angulaire.

La communication se fait en lisant et écrivant des valeurs dans une mémoire interne de 256 registres. Chaque registre peut être dédié à la configuration ou à l'acquisition des données. L'identification initiale des périphériques a été réalisée à l'aide du registre WHO_AM_I, permettant de confirmer la connexion des capteurs à la Raspberry Pi via l'outil `i2cdetect` et `i2cget`.

Avantages et inconvénients

Le MinIMU-9 v6 se distingue par son intégration compacte et sa richesse fonctionnelle. Il permet d'obtenir en temps réel des mesures pertinentes sur les mouvements et rotations de l'objet auquel il est fixé. Grâce à sa double interface I2C, il est possible d'interroger séparément l'accéléromètre/gyroscope et le magnétomètre.

En revanche, ce capteur nécessite une configuration logicielle rigoureuse pour fonctionner correctement. En effet, une mauvaise initialisation des registres peut conduire à une mauvaise acquistion voir un disfonctionnement du système. De plus, ce sont des capteurs

très sensibles. Ainsi, le bruit et les erreurs statiques peuvent être relativement élevés, ce qui peut conduire à une mauvaise interpretation des données.

Sources de bruit et défauts techniques

De manière générale, les mesures issues de l'IMU sont impactées par l'environnement dans lequel le capteur est utilisé. En effet, un environnement incluant des vibrations de la plaque d'essai peuvent fausser les données de l'accéléromètre ou le gyroscope. Dans un second temps, les environnement incluant de fortes interférences magnétiques peuvent générer des perturbations sur le magnétomètre.

Calibration effectuée

La calibration d'un IMU est en général plutôt complexe, car il est assez rare d'avoir une référence à disposition. C'est pour cela que nous avons seulement calibré le gyroscope. Bien qu'il soit assez simple de calibrer ce composant, cela reste une étape cruciale car c'est un composant clé dans la fusion de données, en évitant toute dérive.

Pour cela, nous avons fait des acquisition avec le gyroscope de manière statique. Dans la théorie, la vitesse de rotation autour de chaque axe doit alors être nulle. En faisant la moyenne de ces acquisitions, nous avons alors pu trouver l'erreur statique de ce capteur.

Choix des paramètres d'acquisition

Les paramètres d'acquisition sont définis par les registres de configuration `CTRL1_XL` (pour l'accéléromètre) et `CTRL2_G` (pour le gyroscope). Ces registres permettent de choisir la fréquence d'échantillonnage (de 12.5 Hz à 6.66 kHz), ainsi que la plage de mesure (par exemple, ± 2 g, ± 4 g pour l'accélération ; ± 250 dps, ± 500 dps pour la vitesse angulaire).

En modifiant les valeurs écrites dans ces registres, nous pouvons alors adapter le capteur à différents types de mouvements : une faible plage permet une meilleure précision pour des petits gestes, tandis qu'une grande plage est utile pour des mouvements rapides ou brusques. En ce qui concerne la fréquence d'échantillonnage, une grande fréquence permettra d'avoir une grande précision mais prendra plus de mémoire et sera plus demandeur en calculs. En revanche, une faible fréquence d'acquisition sera moins précise, mais nous donnera moins de données à traiter.

Dans le cadre de notre utilisation, il parait évident que nous n'avons pas besoin d'une grande plage de mesure ainsi que d'une grande fréquence d'acquisition. En effet, à l'échelle d'un humain, nous ne ferons jamais de gestes rapides ou brusques. De plus, notre système qui s'occupe de traiter les données n'est pas d'une puissance infinie, ainsi, nous voulons traiter le moins de données possibles.

Pour nous assurer que nous ne faisions pas d'erreur, nous avons tout de même fait des acquisitions statiques avec des fréquences différentes afin de déterminer si le gyroscope est plus ou moins sensible au changement de fréquence. Le résultat de notre analyse nous a permis de confirmer que cela n'avait pas d'impact.

Acquisition et visualisation

Une fois le capteur correctement configuré, des mesures ont été effectuées à l'aide du script `get_imu_data.py`. En laissant la plaque d'essai statique, les lectures étaient stables et proches des valeurs théoriques. En déplaçant modérément la plaque, on a pu observer des variations nettes sur les axes X, Y et Z.

Les acquisitions ont été enregistrées dans des fichiers CSV pour différentes configurations, puis analysées et tracées avec `plot_imu_data.py`.

2.1.3 Capteur LiDAR (RPLiDAR A1M8)

Description et fonctionnement

Le capteur RPLiDAR A1M8 est un capteur de télémétrie laser 2D capable de balayer l'environnement sur 360° dans un plan horizontal. Contrairement à un LiDAR 3D, ce modèle ne fournit qu'un balayage dans un seul plan. Il émet un faisceau laser à faible puissance qui est réfléchi par les objets environnants. Le capteur mesure le temps nécessaire au retour du faisceau pour calculer la distance, en utilisant le principe du temps de vol.

Le module est relié à une Raspberry Pi 4 via un port USB, ce qui simplifie grandement son intégration. Les données sont traitées à l'aide de la bibliothèque `rplidar-roboticia`, qui permet d'interagir avec le capteur, de récupérer les mesures en temps réel, et d'afficher les résultats sous forme graphique.

Avantages et inconvénients

Le LiDAR RPLiDAR A1M8 présente plusieurs atouts. Il permet un balayage complet sur 360°, ce qui le rend très utile pour cartographier un espace ou détecter des objets dans toutes les directions autour de lui. Son format compact, sa compatibilité USB et la disponibilité d'une bibliothèque Python facilitent grandement son utilisation.

Cependant, ses performances restent limitées en comparaison de modèles plus avancés. D'abord, sa portée est restreinte et la résolution angulaire reste faible pour des applications de précision ou à grande distance. Ensuite, comme tout LiDAR basé sur la réflexion, la qualité de mesure peut être perturbée par des surfaces absorbantes ou trop brillantes.

Sources de bruit et défauts techniques

Les mesures du LiDAR peuvent être perturbées par plusieurs facteurs : la réflexion dépend fortement de la nature de la surface (couleur, texture, inclinaison...), et certaines surfaces diffusent ou absorbent le faisceau laser, provoquant une perte de qualité ou une mesure erronée. Les objets en mouvement rapide ou trop fins peuvent aussi être mal détectés. De plus, des points de faible qualité sont parfois retournés par le capteur, notamment lorsqu'aucune surface n'est détectée à l'angle correspondant.

Calibration et configuration

Aucune calibration physique n'est nécessaire. En revanche, une initialisation logicielle permet d'assurer son bon fonctionnement :

— Installation du package `RPi.GPIO` et détection du port USB associé au capteur.
— Définition des constantes de distance minimale et maximale dans le script `lidar.py`, en se basant sur les spécifications du constructeur.
— Test de lecture via le script `single_scan.py`, permettant d'effectuer un balayage complet à 360°.

Chaque mesure du LiDAR contient trois informations principales : l'angle (en degrés), la distance (en millimètres) et un indice de qualité. Les points sont ensuite convertis en coordonnées cartésiennes à l'aide d'une fonction de transformation polaire-plan (vue en BEV).

Choix des paramètres d'acquisition

Bien que nous n'ayons pas eu besoin d'utiliser ce capteur durant ce projet, durant le TP nous avons pu voir que le capteur fonctionne avec une fréquence d'acquisition par défaut de 5.5 Hz et une résolution angulaire moyenne d'environ 1°. La distance minimale mesurable est de l'ordre de 150 mm, tandis que la portée maximale atteint environ 12 mètres. La précision annoncée est d'environ 1 à 2 % de la distance mesurée, mais cela peut varier fortement selon la qualité de la réflexion.

Filtrage et traitement des mesures

Les données issues du LiDAR peuvent être bruitées. Pour améliorer la lisibilité des nuages de points ou permettre de personnaliser l'acquisition de données, nous avons pu mettre en place plusieurs fonctions permettant de :

— **Filtrer selon la qualité du point** : exclusion des mesures ayant un indice de qualité trop faible.
— **Filtrer selon l'angle du LIDAR** : sélection des points compris dans un intervalle d'angles défini par l'utilisateur.
— **Afficher nos mesures** : transformation des mesures (angle, distance) en coordonnées (x, y) dans un repère 2D.

Précision et validation expérimentale

Des expériences ont été réalisées pour vérifier l'exactitude des mesures. Une feuille blanche a été positionnée à exactement 25 cm devant le capteur, et les mesures collectées via le script `measure_distance.py` ont permis d'obtenir une distance estimée très proche de la valeur réelle. L'écart-type observé sur plusieurs mesures permet de confirmer la stabilité du capteur dans des conditions idéales.

Les tests ont été répétés avec d'autres matériaux (carton, métal, vêtements), montrant une dépendance nette entre la nature de la surface et la qualité du signal. Les objets foncés ou absorbants (comme les tissus noirs) génèrent des distances erronées ou incomplètes.

À plus grande distance (1 mètre, 2 mètres...), les mesures restent cohérentes, mais la dispersion augmente légèrement. Pour maximiser la fiabilité des mesures, il est recommandé d'éviter les angles obliques et les surfaces absorbantes.

Conclusion sur l'usage du LiDAR

Bien que le RPLiDAR A1M8 s'avère être un outil performant pour cartographier un espace en 2D, détecter des obstacles et estimer des distances avec une bonne précision à courte portée, son usage n'est pas adapté aux environnements ouverts ou mal éclairés. Dans le cadre de notre projet, il n'apparait donc pas comme une solution afin d'améliorer la précision de notre GPS.

Chapitre 3

Système d'acquisition

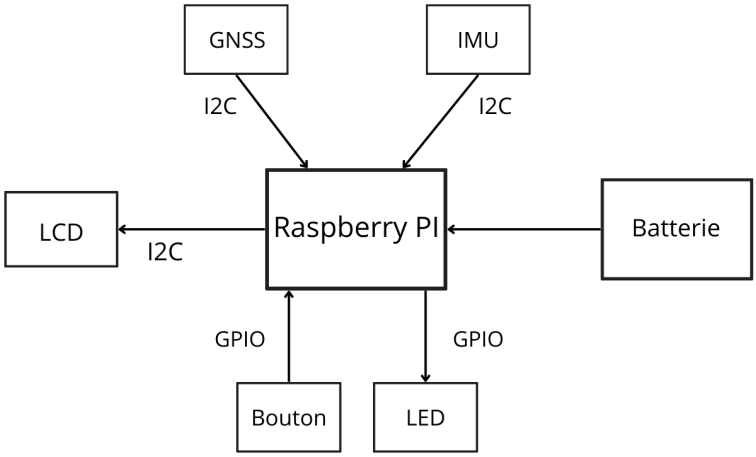

FIGURE 3.1 – Figure 1 : Montage de notre RaspberryPI

3.1 Montage électronique et Description du système

Description globale du système

Tout d'abord, le système est construit autour d'une Raspberry Pi 4 qui centralise les acquisitions des capteurs. Ceux-ci sont connectés via les ports GPIO et I2C, ce qui permet de faciliter la communication. Notre programme permet lancer l'acquisition à l'aide d'un bouton poussoir, une LED confirme alors le bon fonctionnement du système. Les données récupérées sont stockées localement et traitées directement avec la carte. Cela permet d'éviter toute manipulation des données par l'utilisateur. Après que les données aient été traitées, synchronisées, fusionnées et exploitées, le programme affiche la trajectoire sur OpenStreetMap.

Chaine de traitement des données

Afin de comprendre au mieux la façon dont nous avons construit notre système, voici les grandes étapes de notre algorithme :
Pression du Bouton poussoir $\Longrightarrow$ Acquisitions de l'IMU et du GNSS + LED allumée $\Longrightarrow$ Pression du Bouton poussoir $\Longrightarrow$ Prétraitement des valeurs aberrantes (rejets et remplacement) $\Longrightarrow$ Synchronisation temporelle des données du GNSS et de l'IMU $\Longrightarrow$ Conversion de la latitude et la longitude en coordonnées cartésiennes par rapport au centre de la Terre $\Longrightarrow$ Initialisation de la matrice de rotation de l'IMU $\Longrightarrow$ Suppression de l'erreur statique du gyroscope $\Longrightarrow$ Utilisation de la matrice de rotation + des données du gyroscope afin de changer la base de l'IMU $\Longrightarrow$ Utilisation des valeurs converties dans un filtre de Kalman $\Longrightarrow$ Reconversion des valeurs du GNSS en latitude et longitude $\Longrightarrow$ Affichage du trajet post-traitement.

3.2 Choix des paramètres d'acquisition

Comme nous l'avons précisé précédement, nous avons choisi de configurer nos capteurs afin qu'ils ne fournissent pas trop de données à traiter, tout en garantissant une bonne précision. Pour un humain qui marche ou qui court, il ne faut donc pas de grandes valeurs.

Voici la liste des paramètres des capteurs :

— **GNSS** : Fréquence d'acquisition à 1Hz
— **Accéléromètre** : Fréquence d'acquisition à 12.5Hz et plage de mesure à +/-2g.
— **Gyroscope** : Fréquence d'acquisition à 12.5Hz et plage de mesure à +/-2 dps.
— **Magnétomètre** : Fréquence d'acquisition à 40Hz et plage de mesure à 4 Gauss.

Filtrage des signaux bruts

Maintenant que nous avons determiné les paramètres d'acquisition, nous pouvons commencer à récupérer des données. Cependant, toutes les données ne sont pas exploitables. En effet, il est possible qu'il y ait des sauts de position. Pour cela, nous avons mis en place une méthode statistique afin de trier ces valeurs. Nous vous la présenterons alors dans la partie liées aux statistiques.

3.3 Synchronisation des capteurs

Après avoir récupéré nos données filtrées, il faut réussir à les sychroniser. C'est une étape cruciale, car cela nous permettera d'exploiter au mieux les différents types de données à un instant t. Plus précisément, cela nous permettera d'avoir un maximum d'information sur l'état du système à tout moment. Le problème qui se pose est que les capteurs, n'ont pas la même fréquence d'acquisition. Nous ne pouvons donc pas seulement les aligner directement sur une même échelle temporelle sans traitement préalable, car leurs données seraient désynchronisées et pourraient induire des erreurs d'interprétation.

Dans cette partie, nous vous présenterons alors comment nous avons aligner nos données de manière temporelle. En ce qui concerne l'alignement spatial, nous en parlerons plus tard.

Dans un premier temps, il nous a fallu synchroniser les données du GNSS avec celles du téléphone. Pour cela, nous avons travaillé avec le TimeStamp du GPS ainsi que les

heures d'acquisitions du téléphone. Sachant que le GNSS et le téléphone donnent une mesure chaque seconde, il n'était pas difficile de les synchroniser. Nous avons alors rogné l'ensemble des données disponibles selon le min et le max (de la date) de chaque plage de données. Finalement, cette synchronisation nous a permis de comparer les valeurs du GNSS sans traitement avec des valeurs "réelles" et ainsi determiner l'erreur statique.

Dans un second temps, nous avons synchronisé les données du GNSS avec celles de l'IMU. C'est une tâche plus difficile car les fréquences d'acquisitions sont différentes. Nous avons alors séparé ce traitement en deux parties :

- **Réduction des plages de données** : Comme précédemment, nous avons rogné l'ensemble des données disponibles selon le min et le max (du TimeStamp) de chaque plage de données.
- **Interpolation linéaire** : Afin d'avoir un maximum de données exploitables, au lieux de supprimer l'excès de valeurs de l'IMU, nous avons prédit les valeurs associées pour le GNSS.

Finalement, ce traitement nous permettera d'utiliser les données dans la partie fusion de données.

Chapitre 4

Fusion de données

4.1 Définition et utilité

Maintenant que nous avons réussi à configurer notre capteur GNSS et à éliminer les valeurs aberrantes, l'utilisation d'autres capteurs peut sembler superflue. Cependant, il ne faut pas oublier que, bien qu'assez fiable, le GNSS reste sensible à diverses perturbations. En effet, il n'est pas exclu que le capteur perde complètement le signal en raison de l'environnement extérieur, ou qu'il subisse des interférences.

Pour garantir une précision forte dans toutes les situations, nous pouvons alors utiliser la fusion de données. Cette technique consiste à combiner les informations issues de plusieurs capteurs pour obtenir une estimation plus fiable de la position. Dans ce contexte, l'utilisation de l'IMU prend tout son sens. En effet, l'accéléromètre permet de mesurer les variations de vitesse, ce qui permet, à terme, d'estimer la vitesse elle-même. Cette vitesse instantanée peut alors être utilisée pour déterminer les déplacements, et par conséquent, la position.

Comment faire ?

À présent, voyons comment mettre en œuvre cette fusion. Une méthode couramment utilisée est **le filtre de Kalman**, un outil mathématique puissant pour estimer l'état d'un système dynamique, comme la position et la vitesse, à partir de mesures bruitées. Ce filtre repose sur deux grandes étapes, répétées à chaque instant : **une étape de prédiction** puis **une étape de correction**.

Afin de réaliser ces deux étapes, il faut d'abord établir un **modèle d'évolution de l'état** ainsi qu'un **Modèle d'observation**.

4.2 Implémentation

En résumé, nous devons définir chaque élément de ces deux modèles. Ensuite, une fonction python nous permettera d'utiliser l'algorithme de Kalman de manière automatique.

Modèle d'évolution de l'état :

$$x_k = F_k x_{k-1} + G_k u_k + w_k$$

Ici, x_k représente la position et la vitesse réelles, ainsi ce vecteur s'écrit :

$$x_k = \begin{bmatrix} p_{x,k} \\ p_{y,k} \\ p_{z,k} \\ v_{x,k} \\ v_{y,k} \\ v_{z,k} \end{bmatrix}$$

De plus F_k, la matrice de transition d'état permet de passer de la position k-1 à la position k grâce à au vecteur x_{k-1} :

$$F_k = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Ensuite, pour ajouter une information extérieur, nous pouvons définir le veteur u_k (entrée de contrôle). Ici, nous pouvons donner comme information l'accélération observée à l'itération k :

$$u_k = \begin{bmatrix} a_{x,k} \\ a_{y,k} \\ a_{z,k} \end{bmatrix}$$

Pour pouvoir l'utiliser dans notre équation, nous devons définir la matrice de contrôle G_k, tel que :

$$G_k = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 & 0 \\ 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix}$$

Finalement, nous ajoutons du bruit sur la dynamique w_k. Cela nous sert à definir le niveau de confiance que nous attribuons à notre modèle. L'algorithme s'appuiera donc plus ou moins sur cette équation pour approximer la position. Nous définirons ce vecteur de manière empirique.

Modèle d'observation :

$$y_k = H_k x_k + v_k$$

Dans ce modèle, y_k représente le vecteur d'observation, c'est-à-dire les mesures disponibles à l'instant k. La matrice H_k permet de relier l'état interne du système x_k aux variables effectivement observées.

Ici, notre GNSS mesure uniquement la position (et non la vitesse), alors y_k contient seulement $p_{x,k}$, $p_{y,k}$, et $p_{z,k}$, et la matrice H_k s'écrit :

$$y_k = \begin{bmatrix} p_{x,k} \\ p_{y,k} \\ p_{z,k} \end{bmatrix}, \quad H_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Le terme v_k représente le bruit de mesure. Ainsi, ce modèle d'observation permet à l'algorithme de Kalman de confronter l'état prédit par le modèle à ce qui est effectivement observé, et de corriger cet état en conséquence.

4.3 Difficultés rencontrées

Maintenant que nous avons défini nos deux modèles, une dernière étape est nécessaire avant de pouvoir appliquer l'algorithme de Kalman : il s'agit de synchroniser spatialement les données issues de l'IMU et du GNSS. En effet, le GNSS fournit des positions exprimées en latitude et longitude, tandis que l'IMU utilise un repère propre, attaché au mobile. Or, notre modèle travaille dans un repère cartésien. Il est donc essentiel de convertir l'ensemble des données pour qu'elles soient exprimées dans le même système de coordonnées, et ainsi garantir leur cohérence dans les calculs.

Le cas du GNSS :

En ce qui concerne le GNSS nous avons utiliser des formules vues en TD. Cela nous donne :

$$\begin{aligned} \theta &= (90 - \text{latitude}) \cdot \frac{\pi}{180} \\ \phi &= \text{longitude} \cdot \frac{\pi}{180} \\ x &= r \cdot \sin(\theta) \cdot \cos(\phi) \\ y &= r \cdot \sin(\theta) \cdot \sin(\phi) \\ z &= r \cdot \cos(\theta) \end{aligned}$$

Le cas de l'IMU :

Pour l'IMU, nous avons rencontrer bien plus de problèmes. En effet, le fait qu'il utilise un repère attaché au mobile nécessite d'utiliser quelques astuces ainsi que certains outils mathématiques. *Pour résoudre ce problème, nous nous sommes aidé d'un LLM, tout en vérifiant la cohérence des résultats.*

D'abord, nous voulons trouver la matrice de rotation R_{nb} pour passer du repère du mobile au repère NORD-EST-BAS :

$$\vec{v}_{\text{ned}} = R_{nb} \cdot \vec{v}_{\text{body}}$$

Hypothèses et mesures disponibles :

- $\vec{g}_b$: vecteur gravité mesuré par l'accéléromètre (normalisé). Ici, nous faisons l'hypothèse que le système est immobile au lancement du programme.
- $\vec{m}_b$: vecteur champ magnétique mesuré par le magnétomètre (normalisé).

Déduire les axes NED

- $\hat{z}_n$ (Down) : direction opposée de la gravité $\rightarrow -\vec{g}_b$.
- $\hat{x}_n$ (Nord) : projection du champ magnétique sur le plan horizontal, orthogonale à la gravité.

$$\vec{m}_{\text{proj}} = \vec{m}_b - (\vec{m}_b \cdot \vec{g}_b)\vec{g}_b \Rightarrow \hat{x}_n = \frac{\vec{m}_{\text{proj}}}{\|\vec{m}_{\text{proj}}\|}$$

- $\hat{y}_n$ (Est) : produit vectoriel pour compléter la base directe :

$$\hat{y}_n = \hat{z}_n \times \hat{x}_n$$

La matrice de rotation

Chaque vecteur devient une colonne de la matrice R_{nb} (rotation de body vers NED) :

$$R_{nb} = \begin{bmatrix} \hat{x}_n & \hat{y}_n & \hat{z}_n \end{bmatrix}$$

Ensuite, nous voulons trouver la matrice de rotation R_{en} pour passer du repère NORD-EST-BAS au repère centré sur la Terre (ECEF : Earth-Centered, Earth-Fixed) :

$$\vec{v}_{\text{ecef}} = R_{en} \cdot \vec{v}_{\text{ned}}$$

Vecteur Down (D) exprimé en ECEF

Le vecteur Down pointe vers le centre de la Terre, donc dans la direction opposée au vecteur position (local).

Le vecteur position dans ECEF (non normalisé) pour latitude ϕ, longitude λ est :

$$\vec{r} = \begin{bmatrix} \cos(\phi)\cos(\lambda) \\ \cos(\phi)\sin(\lambda) \\ \sin(\phi) \end{bmatrix} \Rightarrow \vec{D} = -\vec{r}$$

Donc :

$$\vec{D}_{\text{ecef}} = \begin{bmatrix} -\cos(\phi)\cos(\lambda) \\ -\cos(\phi)\sin(\lambda) \\ -\sin(\phi) \end{bmatrix}$$

Vecteur Est (E) exprimé en ECEF

Le vecteur Est est tangent à l'équateur local, perpendiculaire à Nord et Down.

$$\vec{E}_{\text{ecef}} = \begin{bmatrix} -\sin(\lambda) \\ \cos(\lambda) \\ 0 \end{bmatrix}$$

Ce vecteur pointe vers l'Est à latitude constante.

Vecteur Nord (N) exprimé en ECEF

Produit vectoriel :

$$\vec{N} = \vec{E} \times \vec{D}$$

Ou bien directement défini :

$$\vec{N}_{\text{ecef}} = \begin{bmatrix} -\sin(\phi)\cos(\lambda) \\ -\sin(\phi)\sin(\lambda) \\ \cos(\phi) \end{bmatrix}$$

Finalement, un autre problème se pose. Cette matrice de rotation n'est valable que pour la position initiale de l'IMU. A partir du moment où ce capteur commence à tourner, les valeurs ne sont plus correctes. Pour résoudre ce problème, nous pouvons utiliser le gyroscope. Celui-ci nous aidera à connaitre la vitesse angulaire de chaque axe du capteur et ainsi pouvoir compenser ces rotation. Nous pouvons alors "revenir" à la position initiale à chaque instant et ensuite réutiliser cette matrice. Après avoir implémenter toutes ces transformations, nous pouvons les appliquer sur nos données et enfin utiliser le filtre de Kalman.

Conclusion sur la fusion des données

Après avoir compris la théorie autour du filtre de Kalman, ainsi que les nombreux obstacles que nous allions rencontrer afin de l'utiliser, nous avons tenté de l'implémenter. Durant nos premières phases de tests, nos résultats montraient des erreurs prochent de 500 mètres autour du point réel. Il nous faudra alors régler les différents paramètres de manière empirique et revoir si notre implémentation est correcte.

Chapitre 5

Analyse des performances

5.1 Méthodologie d'analyse

L'objectif de cette partie est d'analyser des données acquises par les différents capteurs. Cette analyse doit permettre de mettre en place un algorithme de traitement des données lors d'une acquisition réelle de l'application.
Pour cela, et comme demandé, nous allons suivre une approche statistique. L'idée est donc de réaliser différents tests et mesures pour calibrer cet algorithme de traitement.

5.2 Scénarios de tests

Dans cette démarche d'analyse, il faut en premier lieu acquérir des données variées et représentatives pour proposer une analyse pertinente. En effet, l'usage de l'application de suivi d'itinéraire peut se faire dans des cadres variés, et doit donc répondre au mieux à un maximum de cas différent. C'est dans ce but que nous avons défini différents type de parcours pour nos acquisitions.

Dans un premier temps, nous avons réalisé des acquisitions de parcours immobiles. L'objectif de ces parcours est de mesurer l'erreur aléatoire des capteurs.

Nous avons par la suite réalisé de nombreuses acquisitions sur une même boucle, qui présentait les scénario suivant : ligne droite dégagée, proximité avec un mur, passage sous un toit et passage dans la végétation. Pour ce cas, nous avons également fait des acquisitions avec un téléphone pour avoir une référence stable pour le gps. Voici un tracé d'un parcours qui n'a pas été traité :

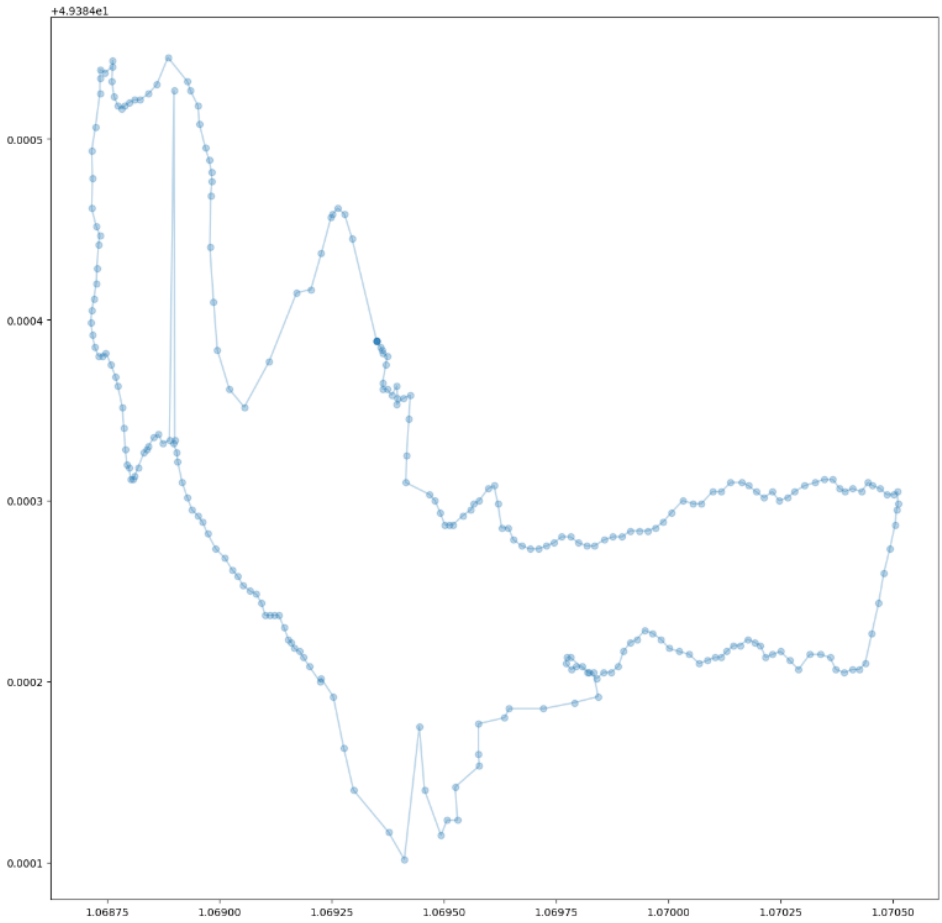

FIGURE 5.1 – Trajet non traité

5.3 Analyse statistique des résultats

5.3.1 Partie GPS

Cette partie traite de l'analyse statistique des données mesurées par le GPS. Dans un premier temps, nous avons analysé les données des acquistions immobiles du GPS. Celles-ci nous fournissent un nuage de point. Grâce à ce dernier, nous avons pu mesurer l'erreur aléatoire du capteur, donnant une idée de sa précision globale. Les erreurs de positionnement ont été analysées statistiquement (moyenne, écart-type, distribution). Le système a montré une erreur moyenne de 1,2 m en extérieur.

Dans un second temps, nous avons traité les données des acquisitions successives d'un même parcours, pour le GPS et la référence. L'objectif de cette partie est de déterminer l'erreur statique de notre GPS, ainsi que de mettre en place un algorithme de traitement des points aberrants.

Nous avons d'abord vérifié que les parcours de références étaient assez précis pour être utilisé comme tel. Pour cela, nous les avons affichés avec matplotlib.pyplot. Cette affichage à montré que la référence n'était pas parfaite, mais nettement supérieur à notre GPS, et qu'il faisait donc sens de l'utiliser.

Avant de réaliser une mesure de l'erreur statique, il nous faut mettre en place l'algoritmhe de traitement des points aberrants, qui pourraient fausser les autres mesures. L'idée est la suivante : Notre GPS a une fréquence d'acquisition d'environ 1Hz. Pour détecter un point aberrant, nous comptions utiliser le fait que la distribution des distances d'un point à l'autre suit une courbe normale. Pour s'assurer que ce modèle est utilisable, nous avons effectué un test de Kolmogorov-Smirnov et calculé la p-value associée à nos échantillons, avec un seuil de rejet fixé à 5%. Ce test nous a comfirmé que la distribution des distances peut être approximée à une loi normale. Une fois cette courbe estimée, les points dont la distance associée ont une probabilité infiérieur à un seuil de n% sont éliminés. Le problème ce cette méthode est que les points aberrants modifient grandement l'estimation de cette loi empirique. De plus, la fréquence d'acquisition n'étant pas constantes, la distance d'un point à l'autre peut plus ou moins varier en fonction d'un délai de mesure plus ou moins grand.

Pour résoudre ces problèmes, nous avons mis en place deux choses. La première est de travailler sur les vitesses et non les distances. En effet, cela permet d'intégrer la variation de la fréquence d'acquisition à la distance mesurée. On calcule donc la vitesse moyenne d'un point à l'autre, en divisant la distance par le delta de Timestamp.

La deuxième étape est d'éliminer les points trop aberrant avant d'estimer notre courbe normale. Pour cela, nous utilisons une boite à moustache sur la norme des vitesses, et éliminons ainsi les points correspondant à une norme trop importante.

Une fois ces points éliminés, nous estimons la moyenne et l'écart type des vitesses empiriques, et estimons une loi normale correspondant. De cette loi, nous calculons les probabilités de chaque vitesses, et donc de chaque point avec un seuil en pourcentage. Les points les plus improbables sont éliminés. Voici une visualisation du résultat pour la vitesse :

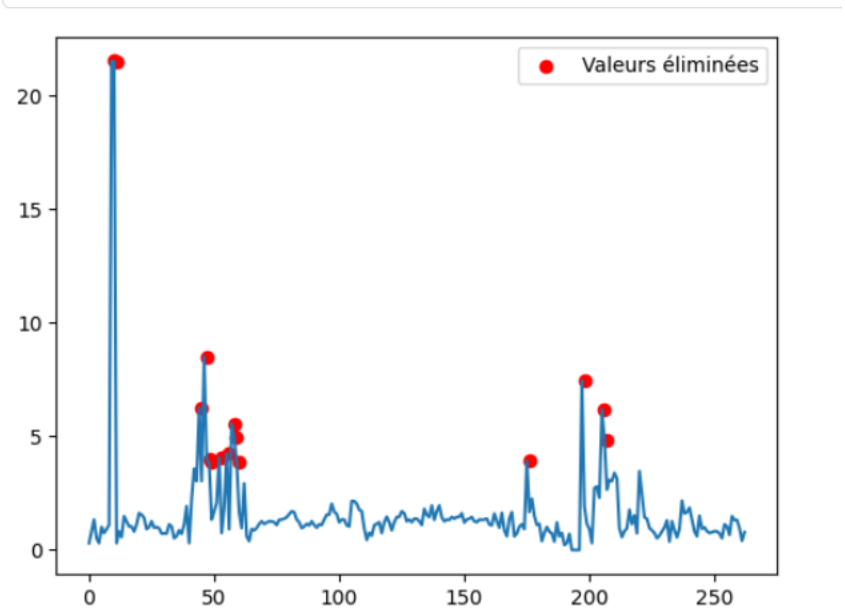

FIGURE 5.2 – Vitesse et points aberrants

Nous pouvons aussi voir les points du trajet liés à ces vitesses aberrantes :

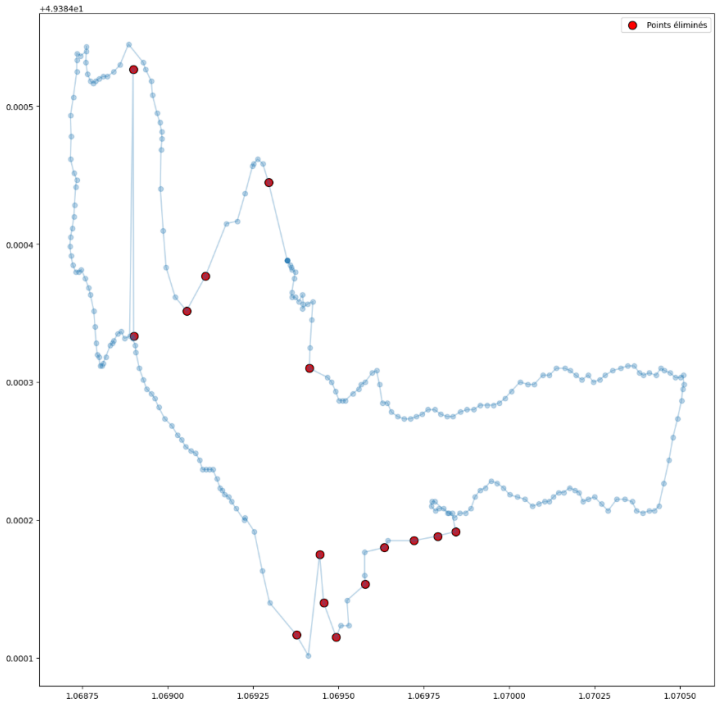

FIGURE 5.3 – Trajet et points aberrants

Cette approche montre des résultats satisfaisants, mais créent une discontinuité dans les points et dans le parcours. Pour palier à ce problème, nous avons décidé de remplacer les points éliminés par de nouveaux points. Pour cela, il semblait judicieux d'utiliser une interpolation linéaire. Cette approche permet de lisser certaines parties du parcours où des points ont été éliminés, tout en gardant une cohérence globale.

Comparaison post traitement

A la fin de ces traitements, nous avons une trajectoire similaire à cette dernière :

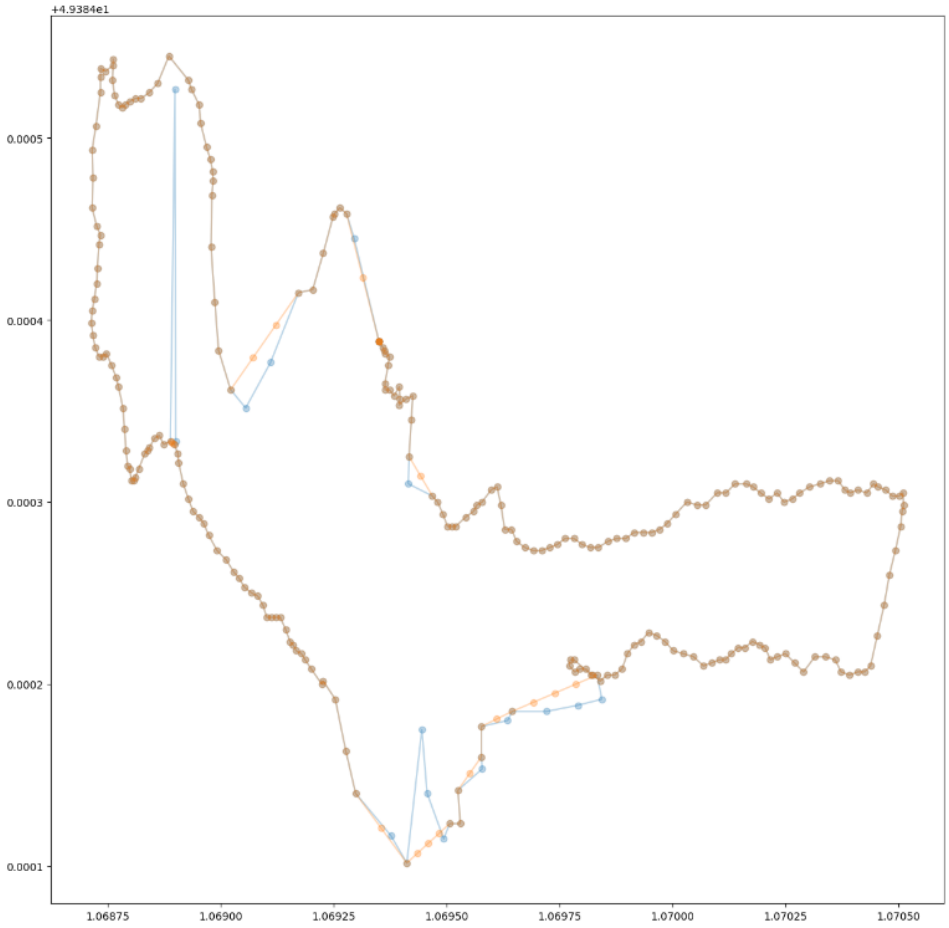

FIGURE 5.4 – Trajet corrigé

Les points bleus représentant les anciens points ont été remplacés par les points oranges, représentant les nouveau points. Nous voyons alors que la trajectoire est beaucoup plus lisse et qu'il n'y a plus de points aberrants.

5.3.2 Partie IMU

Lors de nos analyses, nous avons eu l'occasion de tracer les valeurs captées par l'IMU. Ces dernières semblaient plutôt satisfaisantes. Nous avons alors priorisé l'analyse du GNSS. Nous n'avons malheureusement pas eu le temps d'approfondir le traitement de ces données de la même façon que le GNSS. Ainsi, concernant la partie IMU, nous avons seulement traité l'erreur statique afin de calibrer notre capteur.

Pour cela, nous avions seulement la possibilité d'étudier le gyroscope. En effet, l'accéléromètre et le magnétomètre ont un référentiel propre et des composantes par défauts (accélération gravitationnelle et magnitude du Nord), ce qui rend l'analyse beaucoup plus complexe. Nous avons alors fait des mesures à vide avec le gyroscope afin d'étudier la moyenne empirique. Celle-ci nous indique alors l'erreur statique.

Chapitre 6

Implémentation finale

6.1 Grandes étapes de l'implémentation

Maintenant que nous nous sommes penché sur l'ensemble des parties du projet et que nous avons compris la théorie des acquisition et du traitement des données, nous pouvons tenter de les implémenter afin de rendre le système complètement autonome. Cela passe donc par :

- Demander le nom du fichier de l'utilisateur à créer.
- Attendre que l'utilisateur appuie sur le bouton.
- Lancer les programmes d'acquisition de l'IMU et du GNSS simultanément.
- Allumer la LED indiquant le bon fonctionnement du programme.
- Récuprer les fichiers liés aux acquisition afin de les traiter avec l'ensemble des fonctions codées précédement.
- Afficher et comparer les resultats obtenus

Chapitre 7

Fonctionnalités supplémentaires

Afin de rendre notre système d'acquisition un peu plus interactif, nous avons décidé d'y ajouter quelques fonctionnalités permettant à l'utilisateur de connaitre diverses informations liées à son parcours.

7.1 Calcul de distance et vitesse

Une des fonctionnalités de base est le calcul de distance. Celle-ci est calculée à partir de la somme des distances entre les points GPS que nous avons préalablement corrigés. Ensuite, nous avons calculé la vitesse moyenne liée au trajet. Il nous a suffit de diviser la distance sur le temps total.

7.2 Compteur de pas

Pour ajouter une fonctionnalité un peu plus complète et complexe, nous avons décidé d'implémenter un compteur de pas se fiant à deux mesures. D'abord, nous calculons le nombre de pas grâce à la distance parcourue et la vitesse moyenne. Ensuite, nous estimons le nombre de pas grâce aux valeurs d'accélération.

Distance et vitesse

Pour calculer le nombre de pas à partir de ces données, nous avons considéré le modèle suivant :

En notant que le pas moyen d'un homme est autour 50 à 80 cm, nous commençons par diviser la distance totale du parcours par 65 cm. Cela nous donne une première estimation du nombre de pas. Ensuite, nous considérons que le pas d'un homme varie en fonction de sa vitesse. Pour prendre en compte cette information nous avons multiplier le resultat par un coefficient résultant de la vitesse moyenne de marche d'un homme par sa vitesse moyenne durant l'acquisition. Cela nous donne la formule suivante :

$$\text{vitesse\_marche\_moyenne\_homme} = 4.5$$
$$\text{pas\_distance} = \frac{d}{0.65}$$
$$\text{coef\_vitesse} = \frac{\text{vitesse\_marche\_moyenne\_homme}}{v}$$
$$\text{resultat} = \text{pas\_distance} \times \text{coef\_vitesse}$$

Ce compteur, bien qu'intuitivement intéressant et facile à implémenter, comporte des limites. En effet, ici nous considérons que l'utilisateur va utiliser notre système en marchant (environ 4.5 km/h), or il suffit qu'il commence à courir pour que le coef_vitesse explose et nous donne une approximation absurde. C'est pour cela que nous combinons ce calcul avec un autre bien plus cohérent.

Accélération

Ce compteur de pas est implémenté à partir des pics d'accélérations de l'IMU. Plus précisément, nous commençons par calculer la norme de l'accélération détectée par l'IMU. Ensuite, nous filtrons le signal résultant grâce à un filtre passe bas de Butterworth. A partir de ces données filtrées, nous calculons un seuil donné par :

$$\text{moyenne} = \text{np.mean(norme\_filtrée)}$$
$$\text{std} = \text{np.std(norme\_filtrée)}$$
$$\text{seuil} = \text{moyenne} + 1.0 \times \text{std}$$

Maintenant, nous avons tous les outils nécessaires afin de détecter les pics représentant des pas. Cependant, une dernière caractéristique est à prendre en compte. Il peut arriver que deux valeurs d'accélérations soient très proches. Cela peut donc fausser nos résultats en laissant penser que l'utilisateur a fait deux (ou plus) pas au lieu d'un seul. Nous établissons alors une durée minimale de 300 ms entre la détection de deux pics afin de prendre en compte un pas.

Compteur de pas final

Maintenant que nous avons deux méthode permettant d'approximer nos valeurs de pas, nous pouvons combiner les deux en leur applicant des coefficients en fonction de la confiance que nous leur attribuons. Finalement, nous avons ce modèle :

$$\text{resultat} = (1 - 0.2) \times \text{pas\_imu} + 0.2 \times \text{pas\_distance\_vitesse}$$

Chapitre 8

Conclusion

8.1 Synthèse

Ce projet a permis de mettre en oeuvre un système d'estimation de trajectoire, en exploitant plusieurs capteurs embarqués pour améliorer la précision. Afin d'obtenir une estimation plus fiable de la position il nous a fallu calibrer, synchroniser et fusionner les données des capteurs. Une fois cette étape réalisée, nous avons effectué différent traitement sur ces données pour corriger et améliorer les données en se basant sur des méthodes statistiques et non empiriques. A travers ce projet, nous avons donc approfondi nos connaissances sur les capteurs, tout en apprenant à exploiter les données que nous collections.

Le système a été testé dans différents scénarios, et les résultats ont montré une précision plutôt insatisfaisante. En effet, des difficultés ont été rencontrées lors de la fusion de données, notamment en ce qui concerne la synchronisation spatiale des données issues de l'IMU et du GNSS. Des améliorations ont été apportées pour résoudre ces problèmes, mais des axes d'amélioration restent à explorer.

8.2 Axes d'amélioration

Un des axes principaux d'amélioration est donc la fusion de données : il aurait été non seulement intéressant de trouver une autre façon de garantir une bonne synchronisation des données, mais aussi d'explorer d'autres méthodes de fusion de données, telles que les filtres de Kalman étendus pour améliorer la précision de l'estimation de la position.

Ensuite, nous pourrions nous pencher sur l'optimisation des algorithmes de traitement des données : nous pourrions sûrement améliorer les algorithmes de traitement des données pour réduire le temps de calcul et surtout améliorer la précision des résultats.

De plus, il est évident que nous pourrions tenter d'intégrer d'autres capteurs : multiplier ou intégrer d'autres capteurs, tels qu'un autre GNSS ou bien des caméras, pourraient être une solution afin d'améliorer la précision de notre système et offrir des fonctionnalités supplémentaires.

En ce qui concerne la calibration des capteurs, il pourrait être intéressant de développer d'autres méthodes de calibration plus précises, notamment pour l'IMU, afin de

réduire les différents types d'erreurs.

Finalement, le développement de fonctionnalités supplémentaires pourraient aussi permettre une meilleure immersion durant l'utilisation du système par un utilisateur. Par exemple, cela pourrait être la détection d'obstacles ou la navigation en intérieur, pour rendre le système plus polyvalent et utile dans différents contextes.