Système de localisation et cartographie

Plateforme de développement et algorithme de navigation

Bertrand Wüthrich

Septembre 2022

Master conjoint UNIGE-HES-SO en développement territorial
Orientation Ingénierie géomatique

Direction : Prof. Adrien Gressin, Dr. Michela Thiémard-Spada
Expert : Dr. Matthieu Deveau

Mémoire n° 1008

UNIVERSITÉ DE GENÈVE

Hes·so
Haute Ecole Spécialisée de Suisse occidentale

Table des matières

Liste des figures et des tableaux

1 Introduction

1.1 Présentation du projet

Ce travail s'intègre dans un projet en cours dont l'objectif est de suivre la croissance de la vigne grâce à un système permettant d'extraire et géolocaliser des informations sur la production et l'état de santé du vignoble [1]. Cela passe par la mise au point d'éléments matériels (caméra multispectrale, antenne GNSS, centrale inertielle) et logiciels (navigation, télédétection, assemblage de nuages de points, intelligence artificielle).

Parmi cet ensemble d'éléments à mettre en place, le travail qui est présenté ici consiste en deux parties distinctes. D'une part, il s'agit de développer une application permettant d'opérer le système de manière efficace, de rendre possible son extension et d'interagir facilement avec lui. Cela passe par la migration des briques de programme déjà disponibles vers le système ROS, pour Robotic Operation System, une solution logicielle spécialisée dans les interactions entre composants. Ainsi que la mise en œuvre d'une interface web pour la configuration et le pilotage de l'ensemble.

Le second volet consiste à développer un système de navigation qui s'intègre dans le cadre mis en place. L'objectif est d'obtenir à haute fréquence des informations de position et d'orientation en se basant sur les données fournies par des capteurs inertiels et des observations GNSS. L'idée est ensuite de pouvoir lier ces informations avec les images capturées par les caméras pour pouvoir les traiter par photogrammétrie.

1.2 La navigation inertielle

L'objectif de la navigation inertielle est de déterminer la position et l'orientation d'un système sans l'intervention de sources externes par exemple GNSS. Dans ce cas, la navigation se base uniquement sur les données de capteurs embarqués et autonomes, principalement des accéléromètres et des gyroscopes qui fournissent des informations d'accélérations et de vitesse angulaire dans plusieurs directions.

Le principal avantage de ce type de système est justement son indépendance vis-à-vis d'autres infrastructures. Il permet donc de prendre le relais en cas d'absence momentanée de signal GNSS ou de s'en affranchir totalement. C'est cette capacité qui est recherchée pour des applications telles que l'aviation civile, le guidage de drones, mais surtout dans le domaine militaire où il est clé de ne pas dépendre d'un signal qui peut être brouillé.

Un autre point fort est la disponibilité des données. Si une fréquence de 10 Hz sur des observations GNSS est déjà excellente, il est classique de disposer de capteurs inertiels permettant de connaitre l'état de son système à plus de 100 Hz. Dans le cas qui nous occupe ici, c'est plutôt cette capacité qui est intéressante. En effet, il est primordial d'estimer de manière fine la trajectoire du système afin de connaitre précisément l'orientation du capteur photographique et sa localisation au moment de la prise de vue.

Le point faible de ce système et que, quelle que soit la technologie employée, une dérive plus ou moins marquée vient entacher la position et l'orientation estimée. Pour des gyroscopes, le Tableau 1 permet de se faire une idée de la dérive moyenne en fonction de la technologie.

Les capteurs embarqués dans la centrale inertielle utilisée dans ce projet sont de type MEMS, il est donc irréaliste d'imaginer obtenir des données précises pendant une longue période. La solution dans ce cas-là est d'utiliser des données GNSS à intervalles réguliers pour corriger la trajectoire estimée. La fusion en temps réel des données pondérées en fonction de leur précision est réalisée par un filtre de Kalman.

Tableau 1 : Dérive de différentes technologies de gyroscope [7][4]

Technologie	Coût	Dérive
MEMS Microectromechanical system	\$ 1'000	$\backsim 3\ ^\circ/h$
FOG Fiber Optic Gyroscope	\$ 25'000	$\backsim 1\ ^\circ/h$
RLG Ring Laser Gyroscope	\$ 100'000	$\backsim 0.5\ ^\circ/h$
HRG Hemispherical Resonator Gyroscope	\$ 1'000'000 +	$\backsim 0.005\ ^\circ/h$

La centrale inertielle équipant le système propose également de fournir des informations de position et d'orientation calculées selon un algorithme basé là aussi sur un filtre de Kalman. L'idée étant ensuite de comparer ces deux trajectoires. Ne disposant pas de vérité terrain permettant de juger de la qualité de l'une ou l'autre des prédictions, l'objectif ici est donc plutôt de fournir un système fonctionnel sur lequel pourrait éventuellement venir se greffer d'autres systèmes. Il offre également l'avantage d'être complètement transparent et ajustable selon les besoins. La solution clé en main offerte par la centrale inertielle est sans doute plus aboutie mais ne laisse aucune possibilité de réglage et d'extension.

2 Bases théoriques

La majorité des notions présentées dans ce chapitre sont tirées de [8]

Dans les chapitres qui suivent, les conventions de notation suivantes sont appliquées :

- Pour un vecteur, est indiqué en exposant le référentiel dans lequel est exprimée la grandeur en question. Ainsi f^b dénote le vecteur des forces spécifiques exprimé dans le repère INS (b)
- Pour un vecteur, est indiqué en indice soit :
  - La composante de la grandeur en question. Ainsi, v_e dénote la composante est du vecteur vitesse.
  - Le référentiel de référence et celui en rotation dans le cas de vitesses angulaires par exemple. Ainsi, ω_{ie}^e représente le vecteur des vitesses angulaires du référentiel géocentrique fixe (e) par rapport au référentiel géocentrique inertiel (i), exprimé dans le référentiel géocentrique fixe (e).
  - L'instant considéré ou l'étape d'un algorithme dénoté par un indice t ou k.
- La matrice de rotation permettant par exemple de passer du référentiel local (l) au référentiel INS (b), est notée R_l^b.

2.1 Système de coordonnées

Dans cette section sont décrits les différents systèmes de coordonnées qui interviennent dans la cadre du projet. La Figure 1 présente comment chacun des référentiels se situe par rapport aux autres.

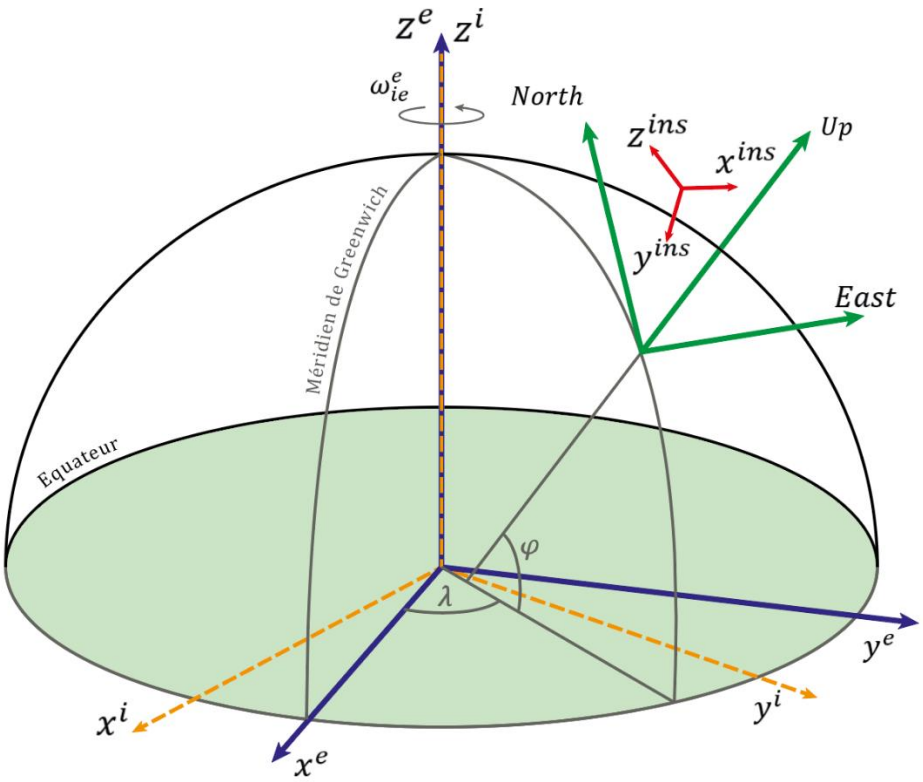


Figure 1 : Systèmes de coordonnées – Référentiel géocentrique inertiel en jaune, référentiel géocentrique fixe en bleu, référentiel local en vert et référentiel INS en rouge

1. Le référentiel géocentrique inertiel a son origine au centre des masses de la Terre, l'axe z selon l'axe de rotation dans la direction du pôle Nord, l'axe x dans la direction du point vernal et l'axe y est de manière à former un repère direct. L'exposant i est utilisé dans la notation.
2. Le référentiel géocentrique fixe a son origine au centre des masses de la Terre, l'axe z selon l'axe de rotation dans la direction du pôle Nord, l'axe x passe par l'intersection du plan de l'équateur et du méridien de Greenwich et l'axe y est de manière à former un repère direct. L'exposant e est utilisé dans la notation.
3. Le référentiel local est un repère topocentrique ellipsoïdal. Il a son origine qui coïncide avec l'origine du système de navigation employé, l'axe y point en direction du Nord géographique, l'axe x pointe vers l'Est et l'axe z est de manière à former un repère direct. Il point donc vers le haut. L'exposant l est utilisé dans la notation.
4. Le référentiel de la Xsens est illustré sur la Figure 2. Son origine est localisée comme indiqué sur la Figure 3. Le roulis est sur l'axe x, le tangage sur l'axe y et le lacet sur l'axe z. Bien qu'il soit possible de la modifier, c'est cette configuration qui a été utilisée pour l'acquisition des mesures.
5. Le référentiel INS a son origine au même endroit que le repère Xsens. L'axe y est l'axe du roulis, il est aligné avec l'axe x du repère Xsens. L'axe x est l'axe du tangage, il est opposé à l'axe y du repère Xsens. L'axe z est l'axe du lacet, il est aligné avec l'axe z du repère Xsens. C'est dans ce repère et non dans celui de la Xsens que l'orientation du système a été modélisée. L'exposant b est utilisé dans la notation.

Figure 2 : Repère Xsens

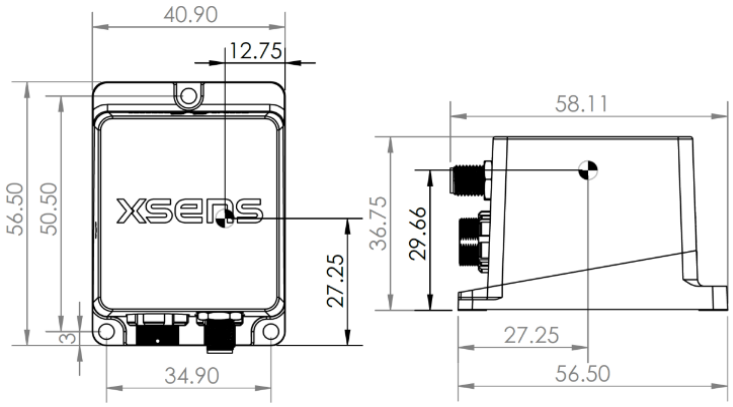


Figure 3 : Dimension et localisation de l'origine des mesures [14]

2.2 Navigation inertielle

2.2.1 Principes

Un système de navigation inertielle et un système permettant de déduire la position, la vitesse et l'orientation à partir de données collectées par des capteurs inertiels, typiquement des gyroscopes et accéléromètres. Le principe est que connaissant l'état initial du système, les états suivants sont obtenus par intégrations des accélérations et des vitesses angulaires.

$$\Delta p(t) = \int \Delta v(t)\, dt \; = \iint a(t)\, dt^2$$

Avec Δp l'incrément de position, Δv l'incrément de vitesse et a l'accélération à l'instant t

Les données d'accélération sont donc employées pour prédire la position et la vitesse. Les informations de vitesse angulaire permettent de déduire l'orientation du système et ainsi de connaitre dans le référentiel local les composantes de l'accélération mesurée dans le référentiel inertiel. En effet dans le système dont il est question ici, la centrale inertielle peut occuper n'importe quelle orientation par rapport à la direction de déplacement.

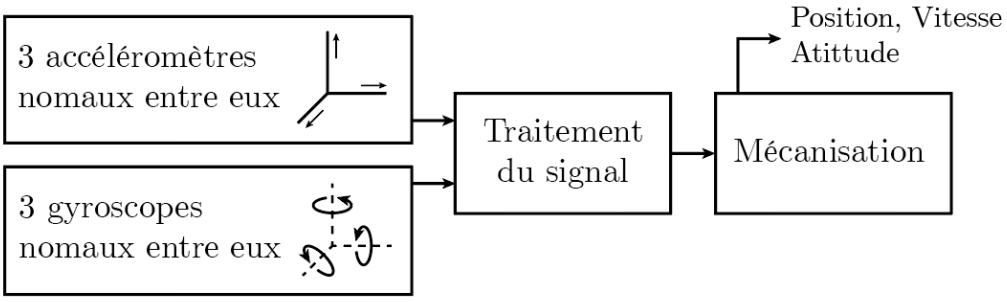


Figure 4 : Navigation inertielle – Principe - Tiré de [8]

Les signaux issus des capteurs inertiels sont prétraités pour filtrer certains effets indésirables avant la mécanisation, qui est le processus d'intégration permettant de connaitre la position, la vitesse et l'orientation dans le référentiel local (Figure 4). Pour connaître l'état du système dans les 3 dimensions, il est nécessaire de disposer de 3 accéléromètres et de 3 gyroscopes normaux entre eux ou au moins sur 3 axes indépendants.

2.2.2 Capteurs

La navigation inertielle repose sur les mesures de plusieurs capteurs dont les principaux sont décrits dans la section suivante.

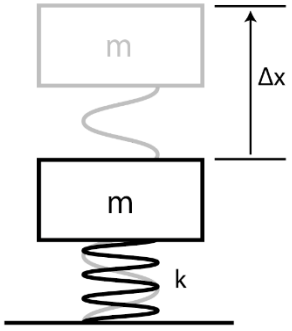
2.2.2.1 Accéléromètres

Schématiquement, un accéléromètre est constitué d'une masse de référence m connecté à un ressort et, ainsi, pouvant osciller le long d'un axe. La position de la masse permet de déduire l'intensité de l'accélération. Pour ce faire l'accéléromètre est calibré pour qu'au repos la position neutre de la masse corresponde à une accélération nulle [2]. Un facteur d'échelle est employé pour convertir la position le long de l'axe de mesure en une accélération. En effet, à l'aide du principe fondamental de la dynamique (2^{e} loi de Newton) il est possible de montrer que dans un système non amorti, l'accélération est proportionnelle au déplacement Δx [6]. Il entre notamment en jeu la constante de rigidité k du ressort.

Un accéléromètre mesure la force spécifique ou accélération propre f. C'est-à-dire la somme de toutes les forces non gravitationnelles divisée par la masse de référence.

$$f = \frac{1}{m}\sum F^a - \frac{1}{m}\sum F^a_{grav} = \ddot{x}^a - \frac{1}{m}\sum F^a_{grav}$$

Avec $\sum F^a$ l'ensemble des forces réelles dans le système accéléré, $\sum F^a_{grav}$ l'ensemble des forces gravitationnelles et $\ddot{x}^a$ l'accélération dans le système accéléré.



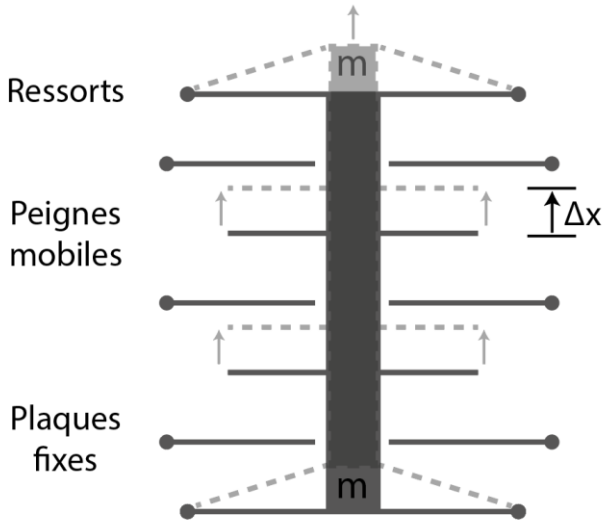


Figure 5 : Accéléromètre : ci-dessus principe physique. Ci-contre, exemple de fonctionnement d'un accéléromètre MEMS. Le déplacement des peignes mobiles induit une variation de capacité entre les plaques fixes de laquelle est déduit le déplacement de la masse.

Dans le cadre de la navigation inertielle, c'est l'accélération $\ddot{x}^a$ qui est employée pour déduire la position par double intégration. Le système accéléré étant non-inertiel, le calcul de $\ddot{x}^a$ fait intervenir les forces d'inertie :

$$\ddot{x}^a = f + \frac{1}{m}\sum F^a_{grav} + a_{Centrifuge} + a_{Coriolis} + a_{Euler}$$

Dans ce projet les référentiels d'inertie et accéléré sont choisi de façon que les accélérations centrifuges et de Euler soient nulles. Les mesures faites par l'accéléromètre sont donc corrigées de la gravité et de l'accélération de Coriolis qui apparait lorsque le système accéléré est en rotation par rapport au système inertiel et que l'accéléromètre est en mouvement par rapport au système accéléré.

2.2.2.2 Gyroscopes

Un gyroscope est un capteur qui mesure la vitesse angulaire par rapport au référentiel inertiel. Un gyroscope peut être de différent type selon le principe physique qui est utilisé. Ainsi, contrairement aux gyroscopes conventionnels avec une masse en rotation, et s'appuyant sur la conservation du moment cinétique, la majorité des systèmes miniaturisés, y compris celui employé dans le cadre de ce projet, utilise une structure vibrante dont la rotation par rapport à son support produit un couple normal au plan de rotation par effet de Coriolis [5]. C'est ce couple qui est mesuré et converti en une vitesse angulaire.

Pour obtenir la vitesse de rotation du référentiel INS par rapport au référentiel local, la mesure effectuée par rapport au référentiel inertiel doit être corrigée de :

- La vitesse de rotation du référentiel local par rapport à la Terre
- La vitesse de rotation de la Terre par rapport au référentiel inertiel

En plus de ces deux capteurs inertiels, d'autres capteurs peuvent être employés pour augmenter la précision et limiter la dérive de systèmes de navigation inertielle. Parmi eux les

magnétomètres et baromètres. Les premiers sont employés pour obtenir en tout temps la direction du nord magnétique et ainsi améliorer l'estimation de l'attitude du système. Le baromètre en mesurant les variations de pression atmosphérique permet de fiabiliser l'estimation des déplacements verticaux. Leur intégration dans le calcul de l'état du système se fait généralement via un filtre de Kalman dont le principe est décrit plus loin.

2.2.2.3 Sources d'erreurs

La mesure de tous ces capteurs peut être sujette à plusieurs types erreurs dont quelques-unes sont décrites ici. Elles sont la source de la dérive qui apparait lors de navigation purement inertielle.

- Un décalage du signal de sortie produit une mesure qui est décalée par rapport à la valeur vraie (biais).
- Une erreur sur le facteur d'échelle permettant d'obtenir la mesure provoque une erreur qui est proportionnelle à la valeur mesurée.
- Une non-orthogonalité entre les axes du capteur peut apparaitre lors de la fabrication.
- Un mauvais alignement entre la centrale inertielle et le système dont on cherche à connaitre la trajectoire et l'orientation

Toutes ces erreurs sont systématiques, elles peuvent être quantifiées et faire l'objet d'une calibration. Certaines erreurs sont aléatoires et doivent donc être modélisées puis compensées pendant l'utilisation du capteur. Il s'agit par exemple :

- Une variation du biais entre plusieurs utilisations du capteur
- Une variation du biais en cours d'utilisation
- Une variation du facteur d'échelle en cours d'utilisation

De plus, tout capteur est sujet au bruit de mesure, c'est-à-dire a une variation aléatoire du signal de sortie lorsque le signal d'entrée est constant. Comme le bruit varie avec la fréquence d'échantillonnage, plutôt qu'un simple écart-type, c'est en général la densité de bruit qui est spécifiée par le fabricant. En $°/s/\sqrt{Hz}$ pour un gyroscope ou en $m/s^2/\sqrt{Hz}$ pour un accéléromètre.

2.2.3 Mécanisation

La mécanisation est le processus de conversion des données brutes, mesurées dans le référentiel inertiel, issues des accéléromètres et gyroscopes vers des informations de position, vitesse et orientation. Ces données peuvent être délivrées dans n'importe référentiel, mais ici c'est le référentiel local qui a été choisi. En effet, pour notre application il a l'avantage de présenter une solution facilement interprétable, notamment en ce qui concerne l'attitude. La mécanisation est un processus récurrent qui, basé sur des conditions initiales, itère en continu pour calculer les données de sortie.

L'algorithme peut être divisé en une série d'étapes :

1. Spécification des conditions initiales

2. Obtention et correction des données d'accélération et de vitesse angulaire dans le référentiel INS par rapport au référentiel inertiel.
3. Calcul de l'attitude (roulis, tangage, lacet) sur la base des vitesses angulaires mesurées
4. Calcul de la vitesse dans le référentiel local par intégration des accélérations tournées dans le référentiel local et compensées des effets de la gravitation et des forces de Coriolis.
5. Calcul de la position dans le référentiel local par intégration de la vitesse.

Cette manière de procéder permet d'obtenir la position au format :

φ : Latitude λ : Longitude h : Altitude ellipsoïdale,

Ainsi que les vitesses dans les directions de l'Est, du Nord et du zénith local (v_e, v_n, v_u) et l'attitude sous forme d'angle de roulis, de tangage et de lacet

2.2.3.1 Traitement des données brutes

Les données fournies par les accéléromètres et les gyroscopes doivent être converties en des incréments d'angle et de vitesse ainsi que corrigées des effets de biais et de facteur d'échelle. Ces incréments corrigés sont donnés par :

$$\Delta\theta_{ib}^{b} = \frac{\widetilde{\omega}_{ib}^{b} - b_{gyro}}{1 + s_{gyro}}\Delta t \qquad \Delta v^{b} = \frac{\tilde{f}^{b} - b_{acc}}{1 + s_{acc}}\Delta t$$

Avec $\widetilde{\omega}_{ib}^{b}$ les vitesses de rotation mesurée par les gyroscopes, $\tilde{f}^{b}$ les forces spécifiques mesurées par les accéléromètres, b_{gyro} et b_{acc} les biais des gyroscopes et des accéléromètres, s_{gyro} et s_{acc} les facteurs d'échelle des gyroscopes et des accéléromètres, Δt l'incrément de temps et finalement $\Delta\theta_{ib}^{b}$ et Δv^{b} les incréments d'angles et de vitesses corrigés.

2.2.3.2 Calcul de l'attitude

Le calcul de l'attitude est basé sur les données des gyroscopes corrigées de la vitesse de rotation de la Terre par rapport au référentiel inertiel ($\omega^{e} \cong 15°/h$) et de la vitesse de rotation du référentiel local par rapport à la Terre. Ces corrections qui correspondent à la vitesse de rotation du référentiel local par rapport au référentiel inertiel sont données par :

$$\omega_{il}^{l} = \begin{bmatrix} 0 \\ \omega_e cos\varphi \\ \omega_e sin\varphi \end{bmatrix} + \begin{bmatrix} -v_n/(R_M + h) \\ v_e/(R_N + h) \\ v_e \tan\varphi/(R_N + h) \end{bmatrix} \qquad \text{puis} \qquad \omega_{il}^{b} = R_{l}^{b}\omega_{il}^{l}$$

Avec R_N le rayon de courbure de la section normale à l'ellipsoïde, R_M le rayon de courbure moyen de l'ellipsoïde et R_l^b la matrice de rotation entre le référentiel local et le référentiel INS. L'incrément d'angle entre le référentiel INS et le référentiel local est obtenu par intégration sur Δt :

$$\Delta\theta_{lb}^{b} = \Delta\theta_{ib}^{b} - \omega_{il}^{b}\Delta t$$

Cet incrément d'angle est utilisé pour mettre à jour la matrice de rotation R_b^l. Il existent plusieurs méthodes pour la paramétrisation et la mise à jour de cette matrice. Ici c'est la méthode des quaternions qui a été employée. Elle présente les avantages d'être efficace en matière de calculs et d'éviter les problèmes de singularité qui peuvent apparaitre avec d'autres paramétrisations.

$$q_t = q_t + f\left(q_t, \Delta\theta_{lb}^b\right) \quad \Rightarrow \quad R_b^l = f(q_t) \quad \Rightarrow \quad \begin{matrix} roulis = f(R_b^l) \\ tangage = f(R_b^l) \\ lacet = f(R_b^l) \end{matrix}$$

Le détail de la méthode est présenté dans [8]. Le principe est d'utiliser l'incrément d'angle $\Delta\theta_{lb}^b$ pour mettre à jour un quaternion représentant l'orientation du référentiel INS par rapport au référentiel local. Ce quaternion permet de calculer la matrice de rotation R_b^l de laquelle peut être déduite l'attitude du système.

2.2.3.3 Calcul de la vitesse

La vitesse est calculée par intégration de l'accélération mesurée, corrigée de la gravité. Les effets des forces d'inertie sont à nouveau à prendre en compte. L'incrément de vitesse dans le référentiel local est alors donné par :

$$\Delta v^l = R_b^l \Delta v^b - \left(2\Omega_{ie}^l + \Omega_{el}^l\right) v^l \Delta t + g^l \Delta t$$

Avec Ω_{ie}^l la matrice anti symétrique associée à la rotation de la Terre par rapport au référentiel inertiel, Ω_{el}^l la matrice antisymétrique associée à la rotation du référentiel local par rapport à la Terre et g^l le vecteur de gravité dans le référentiel local. La gravité est calculée en fonction de la latitude et de la hauteur ellipsoïdale par la formule de Somigliana [8]. La vitesse à l'instant considéré est donnée par intégration :

$$v_{t+1} = v_t + \frac{1}{2}(\Delta v_t^l + \Delta v_{t+1}^l)$$

2.2.3.4 Calcul de la position

Pour connaitre la position au format voulu, il est nécessaire de convertir le vecteur des vitesses obtenu précédemment :

$$(v_e, v_n, v_u) \quad \Rightarrow \quad (\dot{\varphi}, \dot{\lambda}, \dot{h})$$

Cette relation est donnée par :

$$\dot{\varphi} = \frac{v_n}{R_N + h} \qquad \dot{\lambda} = \frac{v_e}{(R_N + h)cos\varphi} \qquad \dot{h} = v_u$$

L'intégrale permet d'obtenir la nouvelle position :

$$\varphi_{t+1} = \varphi_t + \frac{1}{2}\frac{v_{n,t} + v_{n,t+1}}{R_N + h}\Delta t \qquad \lambda_{t+1} = \lambda_t + \frac{1}{2}\frac{v_{e,t} + v_{e,t+1}}{(R_N + h)cos\varphi}\Delta t$$

$$h_{t+1} = h_t + \frac{1}{2}(v_{u,t} + v_{u,t+1})\Delta t$$

2.3 Filtre de Kalman

2.3.1 Présentation

En présence de mesures issues de différentes sources et de précisions variables, l'emploi d'un filtre de Kalman permet d'estimer de manière optimale l'état d'un système. Il permet de combiner les mesures entre elles en les pondérant de manière adéquate. De plus, à chaque étape de l'algorithme, des indicateurs statistiques sont fournis, ce qui permet de quantifier la précision de l'estimation. Cette approche permet d'opérer en temps réel et de connaitre l'état d'un système à une fréquence choisie, indépendamment de la fréquence d'acquisition des mesures.

L'algorithme procède en deux étapes. D'abord une phase de prédiction durant laquelle l'état suivant est estimé sur la base de l'état précédent et de la dynamique du système. Puis une phase de mise à jour qui, lorsque des mesures nouvelles sont disponibles, ajuste l'état prédit en fonction de la précision de ces mesures et du bruit dans le système. Formellement, l'algorithme fait intervenir les paramètres décrits ci-dessous.

Tableau 2 : Paramètres principaux d'un filtre de Kalman

F	Propagateur : Matrice décrivant la dynamique du système
x	Vecteur d'état du système
G	Diffuseur : Matrice décrivant comment le bruit w se propage dans le système
Q	Matrice de covariance du bruit du système
w	Bruit blanc du système
R	Matrice de covariance du bruit de mesure
η	Bruit de mesure
H	Modèle fonctionnel : Matrice décrivant comment l'état du système et relié aux observations
z	Vecteur des observations
P	Matrice de covariance de l'estimation de l'état
K	Gain de Kalman

Pour que l'estimation soit optimale, il est nécessaire que les mesures et la dynamique du système puissent être décrites à l'aide de relations linéaires. De plus, le bruit du système et le bruit sur les mesures doivent être indépendant et être un bruit blanc. La formulation suivante s'applique pour un filtre de Kalman linéaire discrétisé sur un pas de temps Δt :

Modèle dynamique : $x_t = (I + F\Delta t)x_{t-1} + G\,\Delta t\, w_{t-1}$

Modèle des observations : $z_t = H_t x_t + \eta_t$

2.3.2 Description de l'algorithme

Les étapes de l'algorithme sont décrites ci-dessous :

1. Initialisation du filtre : Doivent être fournis le vecteur d'état initial x_0 et son degré d'incertitude P_0 ainsi que les matrices de covariance Q et R qui traduisent respectivement le bruit sur les données issues de la mécanisation et sur les mesures GNSS. Initialement, P_0 est typiquement choisi avec des variances élevées, conséquence de la méconnaissance de l'état initial du système.

2. Prédiction : Sur la base de l'état et l'incertitude au moment $t-1$, ces paramètres sont prédits pour l'instant t.

$$x_t^{pred} = F\, x_{t-1} \qquad \text{et} \qquad P_t^{pred} = F\, P_{t-1} F + G_{t-1} Q_{t-1} G_{t-1}^T$$

3. Mise à jour : Dans cette phase, une nouvelle mesure est prise en compte pour mettre à jour l'état du système. Si le filtre est exécuté à une fréquence plus élevée que la fréquence d'acquisition des mesures, cette étape peut être soit évitée. Auquel cas une nouvelle prédiction est réalisée mais avec une mesure fictive ayant un très bas niveau de confiance. Dans les deux cas, l'incertitude sur l'état du système augmentera. Pour prendre en compte une nouvelle mesure, il faut d'abord calculer le gain de Kalman :

$$K_t = P_t^{pred} H_t^T \left(H_t P_t^{pred} H_t^T + R_t\right)^{-1}$$

   C'est au travers du gain de Kalman que la pondération entre la confiance faite aux mesures et la confiance en l'état du système est réalisée. En effet, une valeur élevée de R_t par rapport à P_t^{pred} tend à donner plus de poids aux observations et corrige l'état en conséquence. C'est la situation où par exemple, une observation GNSS est disponible. Alors que dans le cas inverse, en cas de mesures fortement dégradées ou lors de mise à jour sans mesures, le gain de Kalman est relativement faible et une plus haute confiance est donnée aux prédictions.

4. Une fois le gain calculé, le vecteur d'état et sa matrice de covariance sont également mis à jour en intégrant les observations pondérées :

$$x_t = x_t^{pred} + K_t\left(z_t - H_t x_t^{pred}\right) \qquad \text{et} \qquad P_t = P_t^{pred} - K_t H_t P_t^{pred}$$

   Le nouveau niveau de confiance en l'état du système dépend ainsi de l'ancien, du gain de Kalman et donc indirectement de la précision des observations. Ainsi, la précision des nouvelles observations est disponible.

5. L'estimation de l'époque t est alors terminée, l'algorithme reprend ensuite à l'étape 2.

2.3.3 Systèmes non linéaires

La majorité des systèmes ne répondant pas à l'hypothèse de linéarité, il est possible d'employer un filtre de Kalman pour un système non linéaire à condition de le linéariser. Dans un tel cas, l'état estimé ne correspond plus à la position, la vitesse ou l'orientation d'un système, mais à la différence entre un état de référence et l'état actuel.

L'état de référence peut être de différente nature. S'il s'agit d'une trajectoire connue autour de laquelle le système est linéarisé, il s'agit d'un filtre de Kalman linéarisé [6]. En revanche si

la trajectoire de référence n'est pas connue en avance, comme c'est le cas ici, c'est la trajectoire estimée par mécanisation et corrigée des erreurs estimées par le filtre qui est employée. Il s'agit donc de la différence entre la trajectoire calculée sur la base des données inertielle et les observations GNSS. Dans ce cas, il est fait appel à un filtre de Kalman étendu.

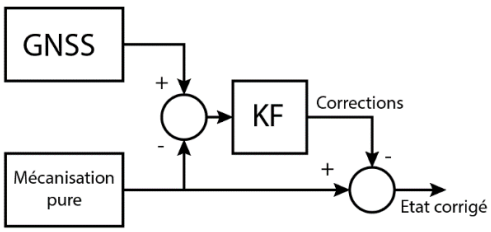


Figure 6 : Système en boucle ouverte, adapté pour un filtre de Kalman linéarisé

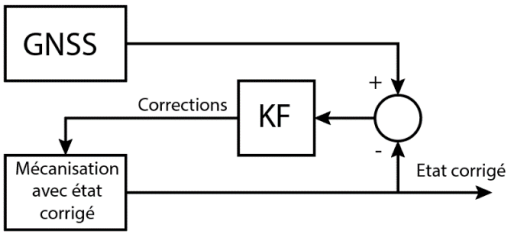


Figure 7 : Système en boucle fermée, prévu pour un filtre de Kalman étendu. L'état employé pour la mécanisation est corrigé des erreurs estimées.

2.3.4 Modèle dynamique

Le vecteur d'état regroupe les erreurs sur la position, sur la vitesse et sur l'attitude ainsi que les biais des gyroscopes et des accéléromètres :

$$\delta x = \left[\delta\varphi,\ \delta\lambda,\ \delta h,\ \delta v_e,\ \delta v_n,\ \delta v_u,\ \delta p,\ \delta r,\ \delta y,\ \delta w_x,\ \delta w_y,\ \delta w_z,\ \delta f_x,\ \delta f_y,\ \delta f_z\right]^T$$

Ou sous une forme condensée : $\delta x = [\delta pos_{1x3},\ \delta v_{1x3},\ \delta a_{1x3},\ \delta w_{1x3},\ \delta f_{1x3}]^T$

Le modèle dynamique consiste à décrire comment une erreur sur chacun de ces paramètres se propage dans le système. Or, dans un système dynamique, les erreurs varient avec le temps et sont donc décrites par des équations différentielles. Ainsi, pour un état x, celui-ci évolue avec le temps selon une relation du type $\dot{x} = f(x,t)$. En la linéarisant autour d'une référence (ici le résultat de la mécanisation) avec un développement de Tylor du 1^{e} ordre, apparait comment l'erreur par rapport à la référence évolue dans le temps :

$$\delta\dot{x} = \frac{\partial f}{\partial x}\delta x = F(t)\delta x$$

F est alors le propagateur présenté plus haut. Les modèles d'erreur présentés ci-dessous sont tirés de [8]. Pour satisfaire à l'énoncé du modèle dynamique tel qu'écrit plus haut, ils ont été intégrés sur un pas de temps Δt.

$$\int_{t-1}^{t} \delta\dot{x}\, dt = \delta x_{t-1} + F(t)\ \Delta t\ \delta x_t$$

Erreur sur la position :

$$\begin{bmatrix}\delta\varphi\\ \delta\lambda\\ \delta h\end{bmatrix}_t = \begin{bmatrix}\delta\varphi\\ \delta\lambda\\ \delta h\end{bmatrix}_{t-1} + \Delta t \begin{bmatrix}0 & \frac{1}{(R_M+h)} & 0\\ \frac{1}{(R_M+h)cos\varphi} & 0 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\delta v_e\\ \delta v_n\\ \delta v_u\end{bmatrix}_{t-1} = \delta pos_{t-1} + F_r\,\Delta t\,\delta v_{t-1}$$

Erreur sur la vitesse :

$$\begin{bmatrix}\delta v_e\\ \delta v_n\\ \delta v_u\end{bmatrix}_t = \begin{bmatrix}\delta v_e\\ \delta v_n\\ \delta v_u\end{bmatrix}_{t-1} + \Delta t \begin{bmatrix}0 & f_u & -f_n\\ -f_u & 0 & f_e\\ f_n & -f_e & 0\end{bmatrix}\begin{bmatrix}\delta p\\ \delta r\\ \delta y\end{bmatrix}_{t-1} + R_b^l \begin{bmatrix}\delta f_x\\ \delta f_y\\ \delta f_z\end{bmatrix}_{t-1}$$

$$= \delta v_{t-1} + F_v\, \Delta t\, \delta a_{t-1} + R_b^l\ \delta f_{t-1}$$

Erreur sur l'attitude

$$\begin{bmatrix}\delta p\\ \delta r\\ \delta y\end{bmatrix}_t = \begin{bmatrix}\delta p\\ \delta r\\ \delta y\end{bmatrix}_{t-1} + \Delta t \begin{bmatrix}0 & \frac{1}{(R_M+h)} & 0\\ \frac{-1}{(R_N+h)} & 0 & 0\\ \frac{-tan\varphi}{(R_N+h)} & 0 & 0\end{bmatrix}\begin{bmatrix}\delta v_e\\ \delta v_n\\ \delta v_u\end{bmatrix}_{t-1} + R_b^l \Delta t \begin{bmatrix}\delta w_x\\ \delta w_y\\ \delta w_z\end{bmatrix}_{t-1}$$

$$= \delta a_{t-1} + F_e\ \Delta t\, \delta v_{t-1} + R_b^l \Delta t\, \delta w_{t-1}$$

Biais des accéléromètres

$$\begin{bmatrix}\delta f_x\\ \delta f_y\\ \delta f_z\end{bmatrix}_t = \begin{bmatrix}\delta f_x\\ \delta f_y\\ \delta f_z\end{bmatrix}_{t-1} + \Delta t \begin{bmatrix}-\beta_{fx} & 0 & 0\\ 0 & -\beta_{fy} & 0\\ 0 & 0 & -\beta_{fz}\end{bmatrix}\begin{bmatrix}\delta f_x\\ \delta f_y\\ \delta f_z\end{bmatrix} = \delta f_{t-1} + F_f\ \Delta t\, \delta f_{t-1}$$

Biais des gyroscopes

$$\begin{bmatrix}\delta w_x\\ \delta w_y\\ \delta w_z\end{bmatrix}_t = \begin{bmatrix}\delta w_x\\ \delta w_y\\ \delta w_z\end{bmatrix}_{t-1} + \Delta t \begin{bmatrix}-\beta_{wx} & 0 & 0\\ 0 & -\beta_{wy} & 0\\ 0 & 0 & -\beta_{wz}\end{bmatrix}\begin{bmatrix}\delta w_x\\ \delta w_y\\ \delta w_z\end{bmatrix} = \delta w_{t-1} + F_w\, \Delta t\, \delta w_{t-1}$$

Où β_f et β_w sont les paramètres d'un modèle de Gauss-Markov utilisé habituellement pour prendre en compte les composantes non déterministes des erreurs de mesures des accéléromètres et gyroscopes [8]. En effet, si une partie des erreurs peuvent être déterminées puis éliminées par un processus de calibration, une partie est aléatoire et doit donc faire l'objet d'un modèle stochastique.

Le bruit blanc du système a été modélisé comme étant composé d'erreurs sur accélération, sur la vitesse angulaire, sur le taux de variation du biais des accéléromètres et des gyroscopes :

$$w = \left[\delta a_e,\ \delta a_n,\ \delta a_u,\ \delta \dot{p},\ \delta \dot{r},\ \delta \dot{y},\ \delta \dot{w}_x,\ \delta \dot{w}_y,\ \delta \dot{w}_z,\ \delta \dot{f}_x,\ \delta \dot{f}_y,\ \delta \dot{f}_z\right]$$

La matrice G de distribution du bruit dans le système décrit les liens entre ces sources de bruit et le vecteur d'état. Finalement, le modèle dynamique linéarisé discret s'écrit alors :

$$\delta x_t = \begin{bmatrix}I & F_r\Delta t & 0 & 0 & 0\\ 0 & I & F_v\Delta t & 0 & R_b^l\Delta t\\ 0 & F_e\Delta t & I & R_b^l\Delta t & 0\\ 0 & 0 & 0 & I + F_w\Delta t & 0\\ 0 & 0 & 0 & 0 & I + F_f\Delta t\end{bmatrix}\delta x_{t-1} + \begin{bmatrix}\Delta t^2/2\ F_r & 0 & 0 & 0\\ I\ \Delta t & 0 & 0 & 0\\ 0 & I\ \Delta t & 0 & 0\\ 0 & 0 & G_w\ \Delta t & 0\\ 0 & 0 & 0 & G_f\ \Delta t\end{bmatrix} w_{t-1}$$

Où G_w et G_f sont des matrices fonction de β_w et β_f complétant le modèle stochastique de Gauss-Markov.

2.3.5 Modèle des observations

Le modèle des observations décrit comment les observations sont liées au vecteur d'état. Dans le cas présent, les observations sont constituées de la différence entre la position et la vitesse issues de la mécanisation et les mesures GNSS :

$$\delta z = \begin{bmatrix} \varphi_{INS} - \varphi_{GNSS} \\ \lambda_{INS} - \lambda_{GNSS} \\ h_{INS} - h_{GNSS} \\ v_{eINS} - v_{eGNSS} \\ v_{nINS} - v_{nGNSS} \\ v_{uINS} - v_{uGNSS} \end{bmatrix}$$

Il a été important à cette étape de bien vérifier la cohérence des données. En effet, le GNSS a fourni des altitudes en millimètre et utilise un repère NED, c'est-à-dire avec des vitesses positives lorsque h décroit. Les signes et facteurs d'échelle devant être ajustés en conséquence. Dans notre cas les grandeurs mesurées sont toutes présentes dans le vecteur d'état. Ainsi, le lien, constitué par la matrice H est direct :

$$\delta z = H\delta x_t + \eta_t = \begin{bmatrix} I_{6x6} & 0_{6x9} \end{bmatrix} \delta x_t + \eta_t$$

2.4 Intégration des observations GNSS

Comme décrit précédemment, la mécanisation permet d'obtenir des informations de position, vitesse et orientation sur la base des mesures d'accéléromètres et de gyroscopes. Ces mesures comportent des erreurs qui, à long terme, altèrent la précision des données calculées sur cette base seule. La stratégie adoptée pour obtenir à haute fréquence des informations de position et d'orientation précises consiste à combiner des mesures inertielles ayant une bonne précision à court terme et disponible à haute fréquence avec des mesures GNSS présentant une précision absolue élevée mais disponible à basse fréquence. C'est au travers du filtre de Kalman que les données GNSS sont intégrées aux données issues de la mécanisation.

A chaque itération du filtre, un état de référence est calculé sur la base des données inertielles prétraitées. C'est la mécanisation. Le filtre de Kalman est utilisé ensuite pour prédire l'état suivant et intégrer les écarts sur la position et la vitesse. Cette dernière est calculée en faisant la différence entre les données issues de la mécanisation et les observations GNSS. Le filtre prédit donc bien l'erreur sur les différents paramètres du vecteur d'état. L'état de référence est ensuite corrigé des écarts estimés. Deux approches sont possibles :

1. Boucle ouverte : Dans ce cas-là, l'état utilisé pour la mécanisation n'est pas corrigé et le vecteur d'état représentant l'erreur entre la trajectoire estimée et la trajectoire de référence, grandit en continu. Le problème principal étant que la linéarisation du système se faisant dans un voisinage autour de l'état de référence, ces erreurs croissantes menacent la validité de l'hypothèse de linéarité. L'estimateur s'écarte ainsi de l'optimalité et peut rencontrer des problèmes de performance [8].

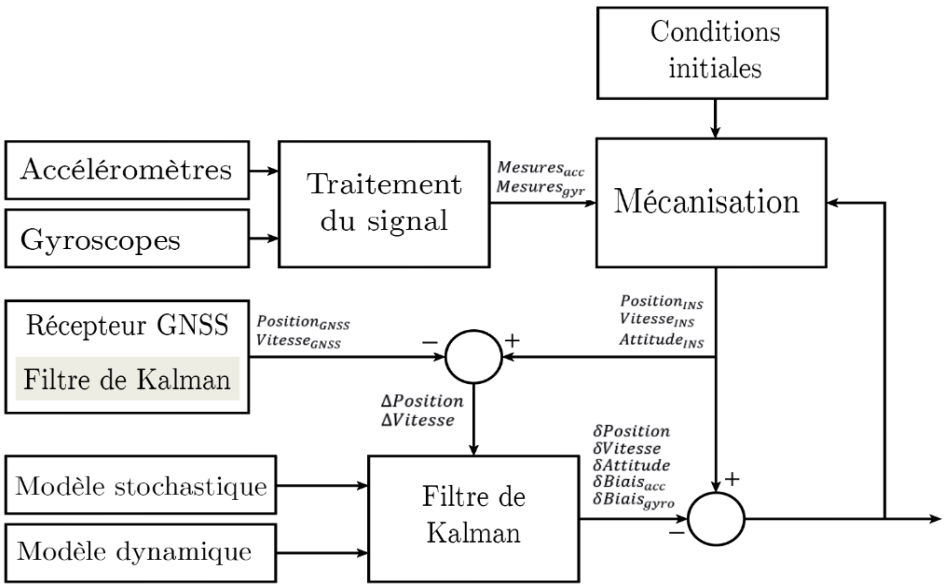


Figure 8 : Fusion des données inertielle et GNSS [8]

2. Boucle fermée : C'est l'état corrigé qui est utilisé à l'instant suivant pour la mécanisation. Dans ce cas, le vecteur d'état doit être réinitialisé après chaque itération du filtre. Cette méthode à l'avantage de garantir que l'erreur estimée reste relativement faible. En revanche, il n'est pas possible d'avoir accès à la trajectoire issue des mesures inertielles seulement. C'est cette méthode qui bien que plus complexe a été employée dans le cadre de ce projet.

Lorsqu'aucune observation GNSS n'est disponible, l'utilisation du filtre reste nécessaire pour tenir à jour la matrice de covariance du système. En effet, en l'absence d'observations, l'état prédit ne diffère pas de l'état estimé à l'instant précédent du fait de la réinitialisation du vecteur d'état :

$$x_t^{pred} = F\, x_{t-1} = \vec{0} \qquad \text{car} \qquad x_{t-1} = \vec{0}$$

L'erreur estimée reste ainsi nulle tant qu'une observation n'est pas disponible. Dans les faits, au lieu de ne pas faire de mise à jour en l'absence de données GNSS, il a été choisi de réaliser cette étape, mais en donnant un poids très faible à des données GNSS fictives. En revanche, la procédure doit être suivie pour estimer comment le degré de précision du système se dégrade et ainsi calculer un gain de Kalman adéquat lors de nouvelles observations GNSS.

3 Composants

Dans cette section sont présentés les différents éléments qui composent le système ainsi que son architecture.

3.1 Système de navigation

3.1.1 Description

La plateforme MTi-680G fabriquée par Xsens constitue le cœur du système de navigation. Il s'agit d'une centrale inertielle couplée à un récepteur GNSS-RTK. Elle inclut un accéléromètre 3 axes, un gyroscope 3 axes, un magnétomètre 3 axes et un capteur de pression. Les données brutes de chacun de ces capteurs sont traitées par un microcontrôleur qui s'occupe de les synchroniser et de les combiner pour fournir la position, la vitesse et l'orientation de la plateforme (Figure 9). Le récepteur GNSS-RTK intégré permet d'accéder à une position absolue de précision centimétrique si l'appareil est connecté à un service de correction NTRIP. Il met également à disposition un signal PPS pouvant être utile à la synchronisation de la plateforme avec des dispositifs externes.

Tableau 3 : Précision en position vitesse et orientation

Donnée	Précision (RMS)
Roulis / Tangage	0.5°
Lacet	1°
Position horizontale/verticale	1.0 m / 2.0 m
Vitesse	0.05 m/s
Position – avec RTK	0.05 m / 0.1 m
Vitesse – avec RTK	0.05 m/s

Trois interfaces sont disponibles. L'interface hôte est utilisée pour la configuration du système et la transmission des données dans le format RS232. Elle inclut un système RTS/CTS de contrôle du flux. Les corrections RTCM sont communiquées dans le format RS232 via un port dédié. Finalement, un connecteur SMA permet de relier l'antenne GNSS. Plusieurs types de données sont disponibles et peuvent être fournies dans différents formats. Le Tableau 4 en présente quelques-unes des plus communes.

Tableau 4 : Exemple de données pouvant être fournies par la plateforme Xsens

Types de donnée	Format	Unité	Fréquence maximum
Orientation	Angle de Euler	degrés	400 Hz
	Quaternion	-	
	Matrice de rotation	-	
Position	Longitude / Latitude	degrés	400 Hz
	Altitude ellipsoïdale	mm	
Vitesse	Vitesse	mm/s	400 Hz
Vitesse angulaire	Vitesse angulaire	rad/s	400 Hz
	Vitesse angulaire « high rate »	rad/s	1600 Hz

Accélération	Accélération	m/s^2	400 Hz
	Accélération « high rate »	m/s^2	2000Hz

Comme vu dans la section 3, le repère Xsens diffère du repère INS par une rotation antihoraire autour de l'axe Z. Xsens utilise la convention « East-North-Up » avec le roulis et le tangage mesurés sur les axes X et Y. Or, ils sont modélisés sur les axes Y et X. De plus, l'indexation du lacet n'est pas identique, 0° avec l'axe de roulis pointant à l'Est pour la Xsens contre 0° avec l'axe de roulis pointant au nord dans le modèle. Les quantités mesurées doivent donc être converties pour passer du repère de la Xsens vers le repère INS utilisé pour la modélisation du système. A noter finalement que par l'usage de magnétomètres, c'est le nord magnétique et non géographique qui est utilisé comme référence pour le lacet.

Tableau 5 : Spécifications des capteurs

Appareil	Stabilité du Biais	Densité de bruit	Ecart-type à 50 Hz
Accéléromètres	$10\ \mu g\ en\ x, y$ $15\ \mu g\ en\ z$	$60\ \frac{\mu g}{\sqrt{Hz}}$	$420\ \mu g$
Gyroscopes	$8°/h$	$1.2 \cdot 10^{-4}\ \frac{rad}{s \cdot \sqrt{Hz}}$	$8.7 \cdot 10^{-4}\ \frac{rad}{s}$

3.1.2 Traitement des données

La plateforme dispose d'un algorithme de fusion lui permettant de fournir des informations de position, vitesse et orientation à fréquence fixe et indépendamment de la disponibilité des données GNSS utilisées pour affiner l'estimation. Le type d'algorithme utilisé n'est pas décrit en détail mais il s'agit certainement d'un filtre type « filtre de Kalman » [14]. En l'absence de données GNSS, l'état de la plateforme est estimé de la manière suivante :

- Tangage et roulis : L'estimation se base sur l'hypothèse que les accéléromètres ne mesurent que les effets de la gravitation et d'une accélération constante. En connaissant le champ de gravité, le roulis et le tangage peuvent être estimé. La limite est que pour un système avec des accélérations qui varient en raison d'un mouvement non uniformément accéléré, cette hypothèse n'est plus vérifiée et l'estimation de ces deux angles dérive. Une fois satisfaite à nouveau, l'algorithme converge vers une solution optimale
- Lacet : Le signal des magnétomètres permet de connaitre la direction du nord magnétique, utilisé comme référence pour l'estimation du lacet. La présence de potentielles perturbations dans le champ magnétique est prise en compte dans l'estimation.
- Position et vitesse : Les données issues de la mécanisation sont employées et éventuellement complétées par des observations GNSS.
- Biais : Le modèle utilisé pour l'estimation des biais des gyroscopes n'est pas décrit, mais se base sur la mesure de la gravité et la déviation par rapport au nord magnétique. Le modèle utilisé déclenche automatiquement un algorithme de mise à jour « CZRU », pour « Continuous Zero Rotation Update », lorsqu'aucune rotation

n'est mesurée pendant un certain laps de temps. Elle permet efficacement d'estimer les biais des capteurs inertiels. Rien n'est dit en particulier sur l'estimation des biais des accéléromètres.

L'appareil dispose finalement d'une option « Smoother » permettant de lisser le signal de sortie ainsi que trois profils de filtre selon si le baromètre est utilisé ou non pour estimer l'altitude et si le magnétomètre intervient ou non dans le calcul du lacet. Les paramètres de calibrations des différents capteurs (biais, alignement, gain) sont déterminés en usine et stockés dans la mémoire de l'appareil. Il est possible d'accéder à ces paramètres via le logiciel MT Manager fourni avec la plateforme.

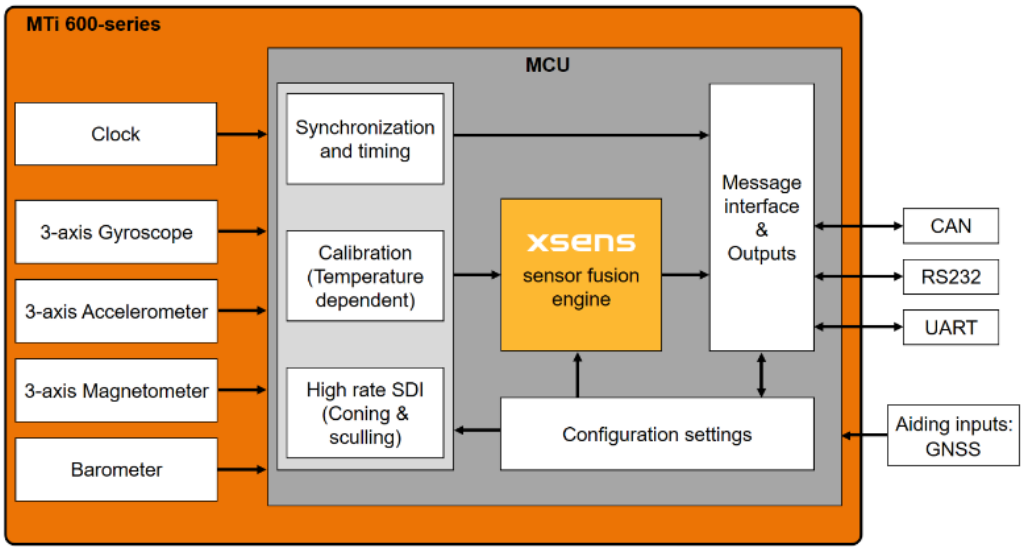


Figure 9 : Différents composants de la plateforme Xsens

Remarque : Contrairement à ce qui est indiqué sur Figure 9, le modèle employé ici ne dispose pas du port UART comme les autres appareils de la gamme MTi.

3.2 Ordinateurs

Deux Raspberry Pi 4 ont été employés comme ordinateurs pour faire fonctionner l'entier du système. Cette solution a été retenue au vu des performances requises pour pouvoir exécuter efficacement l'entier du système. Selon la fréquence désirée, la capture des images par exemple est particulièrement gourmande en ressources. Quelques caractéristiques de cet appareil sont présentées dans le Tableau 6.

Tableau 6 : Caractéristiques du Raspberry Pi 4 [15]

Processeur	ARM Cortex-A72 1,5 GHz	Système d'exploitation	Ubuntu
Mémoire	4 Go	Environnement	Linux
Connectique	4 ports USB / Ethernet / microHDMI / Wifi / Bluetooth		

Etant limité en puissance de calcul pour certaines applications, un ordinateur portable classique a été employé pour effectuer certain des tests présentés dans ce document. Il s'agit d'un Lenovo X1 Carbon.

Tableau 7 : Caractéristiques du Lenovo X1 Carbon

Processeur	Intel Core i7-7600U 2.80GHz	Système d'exploitation	Windows 10
Mémoire	16 Go	Environnement	Windows

Lors des mesures sur le terrain, une batterie « RealPower PB-20k » a été employée. Selon ses caractéristiques, elle est adaptée pour alimenter le Raspberry Pi.

Tableau 8 : Alimentation Raspberry vs RealPower

Alimentation	Courant en sortie	Puissance nominale
Raspberry Pi 4	3 A	15.3 W
RealPower PB-20k	3 A	18 W

3.3 Caméras

Pour la prise de photos, quatre caméras sont utilisées afin de disposer d'images dans différentes bandes spectrales. En effet, étant dotées d'un capteur monochrome, c'est par le filtre disposé devant l'objectif que la plage de longueur d'onde captée est choisie. Les quatre caméras sont identiques, il s'agit du modèle DMK 37BUX264 fabriqué par The Imaging Source. Les principales caractéristiques du capteur et de l'objectif sont présentées dans le tableau ci-dessous.

Tableau 9 : Caractéristique des appareils photographiques

Résolution	2,448×2,048	Interface	USB 3.1
Frame rate	38 fps	Focale	12 mm
Obturateur	Global	Ouverture	F1.6 – F16
Pixel size	3.45 x 3.45 µm	Distance minimum	10 cm

Pour cette application, il a été nécessaire de choisir un boîtier ayant la possibilité d'être déclenché à distance via un port dédié. Les précédentes expériences montrent également que l'usage d'un obturateur global évite l'effet de « rolling shutter » qui peut apparaitre lors de prises de vue en mouvement ou d'objets mobiles. Seul réglage manuel, le degré d'ouverture du diaphragme devrait être identique sur chaque boîtier pour avoir une profondeur de champ cohérente entre les images.

3.4 Système

3.4.1 Architecture

La Figure 10 illustre l'architecture générale du système en présentant comment les différents composants sont connectés entre eux et via quelles interfaces. Le Raspberry I est la pièce maitresse du système. Il s'occupe de réceptionner les données de la Xsens via le câble Host et de lui envoyer les corrections RTCM par un canal dédié. Il se charge de coordonner le déclenchement des caméras et héberge le serveur web servant d'interface utilisateur. Le Raspberry II pour sa part gère la communication avec les caméras et collecte les données qui en sont issues.

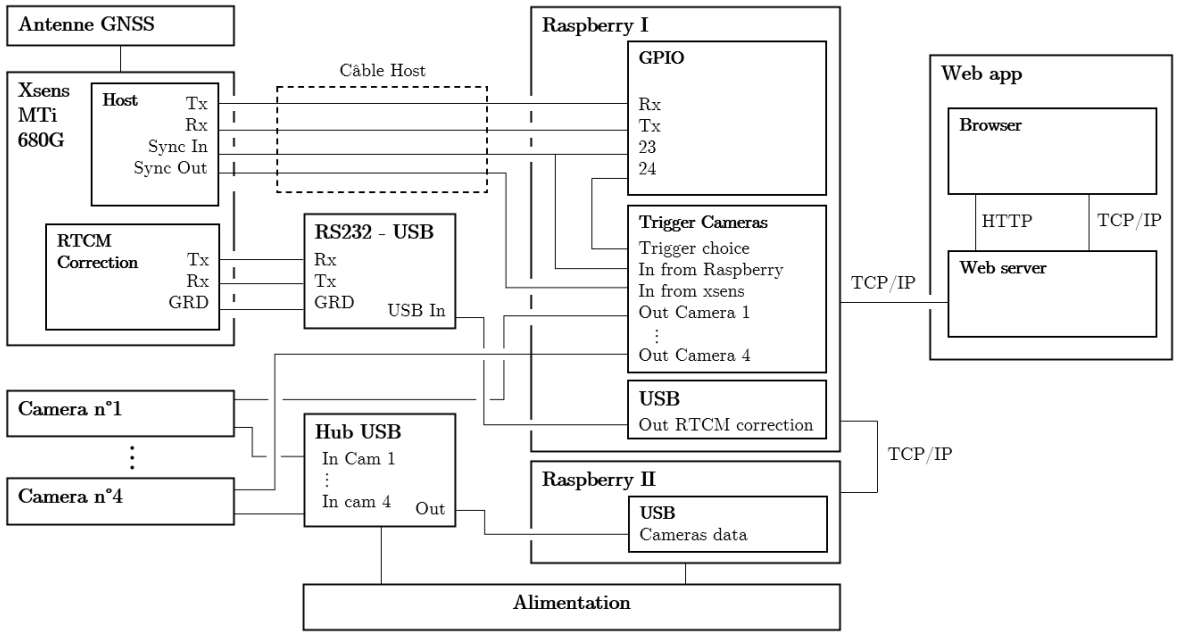


Figure 10 : Connexions entre les différents composants du système

Pour pouvoir communiquer entre eux, il est nécessaire que les deux Raspberry soient sur le même réseau.

3.4.2 Interfaces

Dans certains cas, l'interface entre deux composants a fait l'objet de développements particuliers. Quelques informations sont fournies ici sur ces cas spécifiques.

3.4.2.1 Extension du Raspberry

Le Raspberry I est équipé d'une extension lui permettant de communiquer avec la plateforme Xsens et de piloter le déclenchement des caméras. Elle intègre :

- Un port « LEMO » 12 pins pour connecter le câble Host de la Xsens
- Quatre ports « LEMO » pour le déclenchement de chaque caméra
- Un composant pour convertir le signal RS232 de la Xsens vers le UART du Raspberry
- Un composant permettant de choisir (via un port GPIO) le mode de déclenchement des caméras :

- Soit depuis la Xsens via un port dédié du câble Host
- Soit depuis le Raspberry I via un port GPIO. Dans ce mode, une impulsion doit également parvenir à la Xsens via le port SyncIn du câble Host pour qu'une mesure soit synchronisée avec la capture des images.

L'extension vient se brancher sur l'ensemble des ports GPIO du Raspberry I. La Figure 11 représente schématiquement le fonctionnement de cette extension.

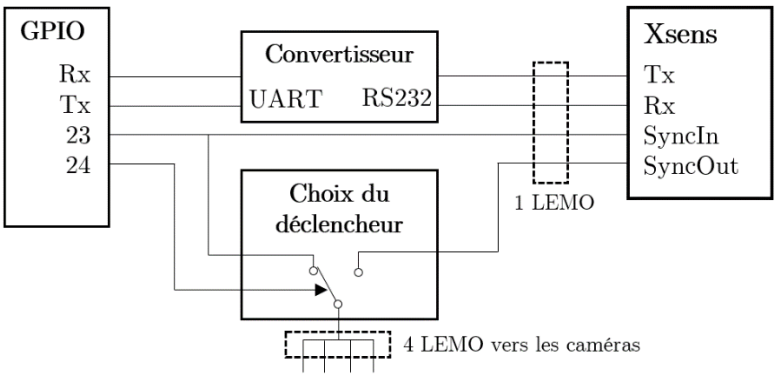
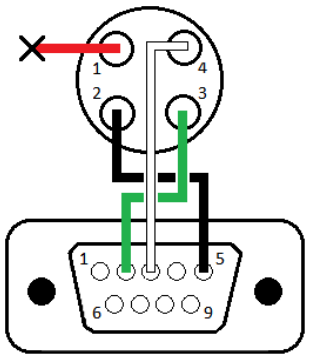


Figure 11 : Schéma de principe de l'extension du Raspberry I

3.4.2.2 Corrections RTCM

Initialement, il a été choisi de faire transiter les corrections RTCM via le câble Host. Le désavantage est que les mesures envoyées vers le Raspberry entrent en conflit avec les corrections envoyées vers la Xsens. Ainsi à chaque mise à jour RTCM, le flux de mesures est interrompu. Pour remédier à ce désagrément, un port dédié aux corrections est disponible sur la plateforme. Pour pouvoir les envoyer depuis le Raspberry via USB, un convertisseur RS232-USB a été employé. La manière de connecter le câble dédié et le convertisseur est illustrée sur la Figure 12.



Connexion câble RTCM – convertisseur RS232-USB

n°	Fonction du port	n°	Fonction du port
3	RTCM TxD	2	RTCM RxD
4	RTCM RxD	3	RTCM TxD
2	Terre	5	Terre

Figure 12 : Schéma de connexion RTCM – Raspberry via convertisseur RS232-USB

3.4.2.3 Hub USB

Les quatre caméras sont connectées à un hub USB qui est lui-même connecté par USB au Raspberry II. Sur ce modèle de Raspberry, le courant maximum à disposition pour l'ensemble des quatre ports USB est de 1.2 A. La consommation de chaque caméra à 5V est approximativement de 360 mA. Le courant fourni ne suffisant pas, pour éviter des problèmes de fonctionnement, il a été choisi d'alimenter les caméras via le hub USB.

4 Application

4.1 Développement

Dans cette section, il est décrit les principaux outils informatiques qui ont été utilisés pour le développement du système. Après une partie introductive, il est expliqué comment ils ont été mis en œuvre dans le cadre du projet

4.1.1 Docker

4.1.1.1 Introduction

Docker est un outil qui permet d'empaqueter une application et ses dépendances à l'intérieur d'un conteneur. Le but étant de pouvoir facilement transporter cette application et l'exécuter dans différents environnements. Souvent confondu avec une machine virtuelle, un conteneur ne comprend pas de système d'exploitation, il s'appuie sur celui de la machine hôte. Cela le rend plus léger et donc plus facile à partager et à entretenir [10].

Un conteneur Docker est créé sur la base d'une image. Cette dernière peut avoir été récupérée depuis un registre, par exemple *dockerhub* [22], ou construite avec un Dockerfile. L'image est exécutée par un contrôleur nommé *Docker Engine* pour créer un conteneur.

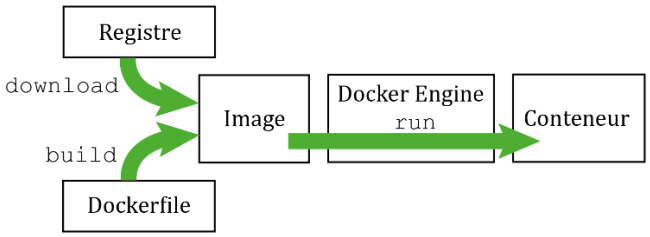


Figure 13 : Création d'un conteneur Docker

Dans le cadre de ce projet, il a été choisi d'utiliser Docker pour éviter certaines complications liées à l'installation de ROS sur un Raspberry Pi, mais surtout pour pouvoir aisément migrer l'application vers un autre environnement si nécessaire.

4.1.1.2 Dockerfile

L'image utilisée pour ce projet a été construite avec un Dockerfile. Elle est basée sur une image ROS disponible en ligne, associée à toute une série de dépendances nécessaires à la bonne marche de l'application. Le Dockerfile a été complété au fil du développement pour tenir compte des différentes librairies qui ont été ajoutées. La commande suivante lance la construction d'une image sur la base d'un Dockerfile :

```
docker build -t [nom de l'image]:[tag] .
```

l'option `-t` permet de taguer l'image par exemple avec un numéro de version. A chaque exécution d'une image, c'est un nouveau conteneur qui est créé avec à l'intérieur ce qui a été prévu dans le Dockerfile et d'éventuelles données enregistrées auparavant par exemple dans un volume. Tout changement à l'intérieur d'un conteneur qui ne fait pas l'objet d'une mise à jour du Dockerfile ou d'un enregistrement dans un volume est perdu à l'arrêt du conteneur.

La même image a été employée sur les deux Raspberry, bien que moins de dépendances soient requises pour le Raspberry II. A noter finalement que l'installation de ROS est sourcée lors de la création de l'image. Nul besoin de le faire lors du démarrage du conteneur.

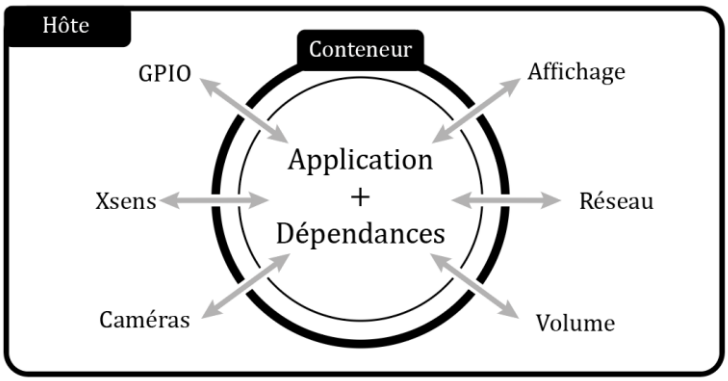


Figure 14 : Interactions avec l'extérieur du conteneur

4.1.1.3 Démarrage d'une image

Une image est exécutée avec la syntaxe suivante :

```
Docker run [options] [nom de l'image]:[tag]
```

Plusieurs options ont été employées, notamment pour pouvoir accéder depuis le conteneur aux ressources connectées à l'ordinateur hôte (Figure 14). Elles sont décrites dans le tableau ci-dessous.

Tableau 10 : Options employées au lancement de l'image Docker.

Option	Description	sur Raspberry		
		I	II	I+II
`-it`	Requis pour l'exécution dans une console			X
`--name dev_ros`	Permet de donner un nom au conteneur			X
`--env="DISPLAY"` `-v /tmp/.X11-unix/:/tmp/.X11-unix`	Requis pour accéder à l'affichage			X
`--net=host` `--pid=host`	Permet à ROS de communiquer entre deux conteneurs situés sur deux machines sur le même réseau			X

`--device=/dev/video0`
…
`--device=/dev/video7`
	Rend les caméras visibles depuis le conteneur		X	

`--device=/dev/ttyS0`	Rend la plateforme Xsens visible depuis le conteneur	X		
`--device=/dev/ttyUSB0`	Permet de se connecter au convertisseur RS232 – USB	X		

`--device=/dev/mem:/dev/mem`
`--`
`device=/dev/gpiomem:/dev/gpiomem`
	Permet d'accéder aux port GPIO	X		

`-p 8000:8000`	Expose le port 8000 pour pouvoir accéder à l'interface web	X		

`-p 9090:9090`	Expose le port 9090 pour le Webserveur ros_bridge	X		
`-v ros_vol:/app`	Permet d'accéder au volume ros_vol depuis le conteneur			X
`--rm`	Supprime le conteneur à sa sortie			X

L'utilisation d'un volume, créé au préalable, permet de retrouver ses fichiers entre deux démarrages de l'image. Ici, tout ce qui se situe dans le répertoire /app du conteneur est sauvé dans le volume ros_vol. Le fait de supprimer le conteneur à sa sortie, oblige une certaine rigueur sur la tenue du Dockerfile et l'enregistrement des données. Cela permet d'éviter de mauvaises surprises en cas de redémarrage du conteneur.

Concernant ROS, il n'y a pas besoin de configurer quoi que ce soit pour faire communiquer deux machines si elles sont situées sur le même réseau. Ici, la difficulté réside dans le fait que ROS est exécuté dans deux conteneur totalement isolés tant que les deux options présentées ci-dessus ne sont pas utilisées.

Finalement, une fois l'image lancée, il peut être utile d'accéder au conteneur depuis plus qu'un terminal. Cela est possible grâce à la commande suivante :

```
docker exec -it [nom ou Id du conteneur] bash
```

4.1.2 ROS

4.1.2.1 Introduction

Nous présentons dans cette section le logiciel ROS sur lequel est basé une bonne partie du système dont il est question ici. ROS, pour *« Robot Operating System »* est une plateforme permettant de développer des systèmes pour la robotique. Souvent appelé métasystème d'exploitation ROS permet de mettre en relation des applications fonctionnant sur différents ordinateurs, d'interagir avec des périphériques physiques et virtuels et de mettre en relation chaque composant via un système de messages, de service et d'actions.

Deux APIs Python et C++ ont été développées pour interagir avec ROS. Dans l'une comme dans l'autre, l'unité fondamentale de l'architecture ROS est le nœud. Associé à un processus, à un périphérique ou purement virtuel, il s'exécute continuellement. Un nœud peut envoyer ou recevoir des données en publiant ou en souscrivant à des *« topics »*. Un nœud peut également requérir l'exécution d'une tâche (client) auprès d'un autre nœud (serveur) au travers de services, pour des tâches courtes. L'utilisation d'actions est recommandée lorsqu'un retour en continu sur le statut de la tâche est souhaitable.

Au vu de la complexité du système dont il est question dans ce projet, l'utilisation de ROS a semblé particulièrement adaptée. En effet, différents capteurs et appareils sont amenés à communiquer ensemble, à s'échanger des données et à voir le cours de leur exécution liée à d'autres composants. ROS permet de coordonner toutes ces interactions de manière optimale.

Il a été choisi d'utiliser la nouvelle version de ROS, c'est-à-dire ROS2 et sa distribution Foxy Fitzroy sortie le 20 juin 2020. Le choix n'a pas été direct, car plusieurs éléments sont à prendre en compte notamment la compatibilité de certaines librairies et modules développés sous ROS. Toutefois, le fait que la communauté soit résolument tournée vers ROS2 et que la

dernière distribution de ROS1 soit bientôt plus maintenue nous a poussés vers la nouvelle version.

Avant de se lancer dans le développement du système, une recherche a été faite sur les différents drivers existant pour les composants du système. Seuls quelques candidats pour piloter la Xsens ont été trouvés :

Tableau 11 : Driver Xsens disponible

Développement	Distribution	API	Date de sortie	Repo GitHub
ETHZ	Melodic – ROS1	Python	2019	[16]
BlueSpace AI	Foxy – ROS2	C++	2020	[17]
Xsens	Noetic – ROS1	C++	2022	[18]

Aucun de ces drivers n'a été retenu soit en raison de l'API C++ qui n'est pas ce qui était voulu, soit en raison de la version de ROS. Il a donc été choisi de développer un driver ROS2 écrit en Python, s'inspirant largement du travail de l'ETHZ. L'architecture générale et une grande partie du code en sont donc tirés.

4.1.2.2 Les packages

La Figure 15 présente comment s'organise l'application ROS. A la base se trouve l'espace de travail (dev_ws), c'est le répertoire depuis lequel le programme sera compilé puis exécuté. Dans le dossier /src sont placés les différents packages. C'est sous la forme d'un ou plusieurs packages que l'application peut être transmise et installée. Un package Python est créé avec la commande suivante :

```
ros2 pkg create --build-type ament_python <package_name>
```

Chaque package présente la même architecture :

- Un fichier *package.xml* avec les métadonnées sur le package et une liste des dépendances requises.
- Un fichier *setup.py* avec les instructions d'installation du *package*. Doivent notamment y figurer les éventuels fichiers de configuration ou de lancement (*launchfile*) et un point d'entrée pour chaque exécutable du package.
- Un fichier *setup.cfg* requis si un package contient des exécutables. Pour ce projet, il n'a pas été nécessaire de modifier ces fichiers.
- Un dossier qui contient les exécutables sous forme de fichier Python

4.1.2.3 Les exécutables

Un exécutable type contient un nœud, c'est-à-dire une classe qui hérite de la classe *Node*, et une fonction *main* qui régit l'exécution du nœud. Lorsqu'un package est démarré, ce sont les fonctions *main* de ces exécutables qui sont appelées.

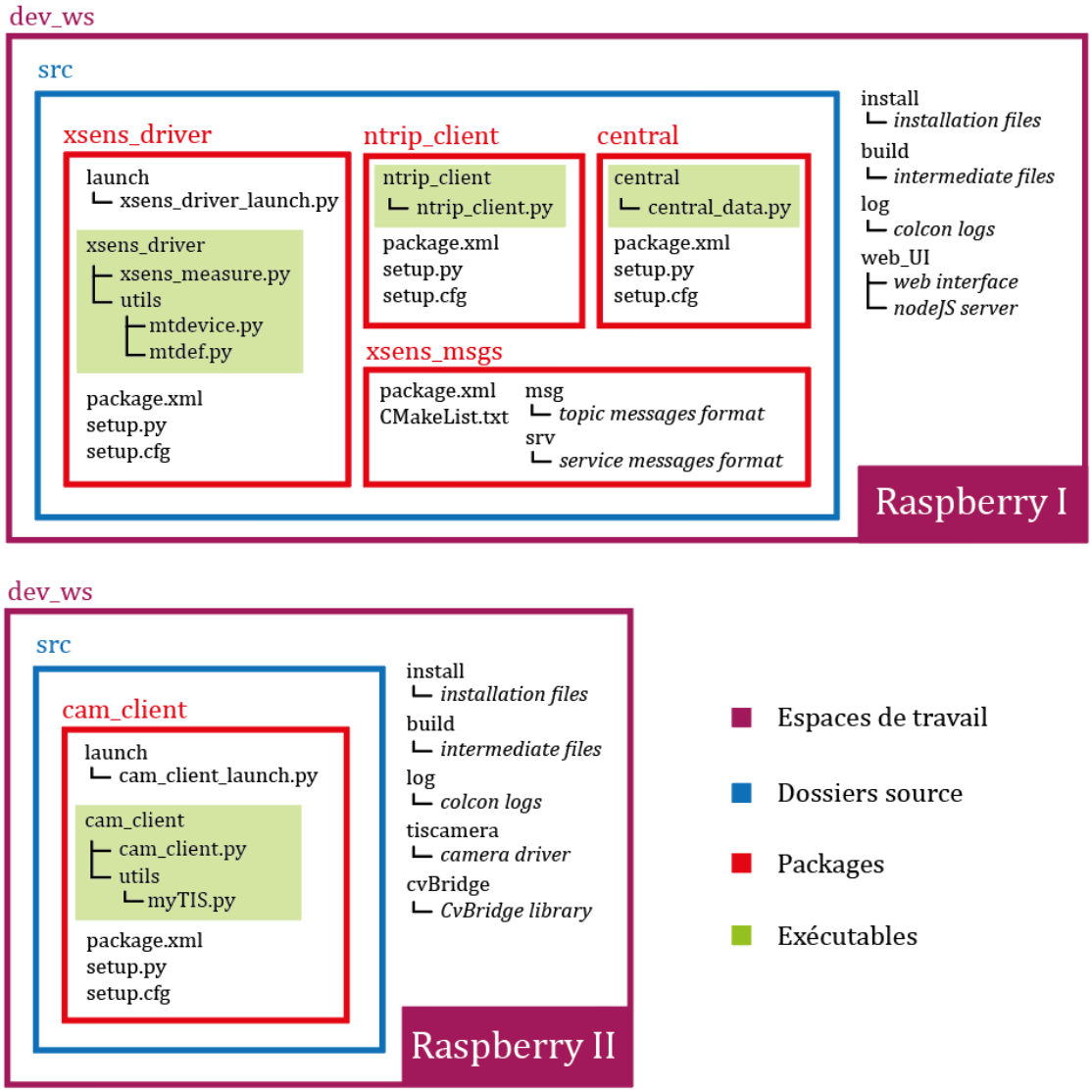


Figure 15 : Arborescence et organisation de l'application ROS

Tant que le nœud « tourne » il va publier, lire et exécuter ces fonctions callback. Avec la fonction `spin` le nœud s'exécute tant que l'exécutable est actif. La fonction spin_once s'assure que le nœud s'exécute une fois. Afin de pouvoir donner au nœud différent statut et attendre sur d'autres exécutables, il a souvent été choisi d'utiliser une boucle `while` avec une condition sur le statut du nœud, à l'intérieur de laquelle une fonction `spin_once` lance le nœud.

C'est dans le nœud que sont créés les *publishers* qui permettent d'émettre sur un topic et les *subscribers* qui permettent d'accéder à ce qui est publié. C'est là aussi que sont définis d'éventuels services ou actions.

Lorsqu'un *publisher* est créé, le nom du topic sur lequel il va publier doit être définit tout comme le type de message qu'il va employer. De la même manière, lorsqu'un *subscriber* est instancié, il faut lui spécifier quel type de message il doit lire, sur quel topic le chercher et quelle fonction doit être appelée à la réception d'un message. Cette fonction callback prend en argument le contenu du message et permet ainsi de le traiter.

4.1.2.4 Les messages

Au début de l'exécutable, tous les formats de message employé doivent être importés. Ils peuvent être de deux types. Soit c'est un format prédéfini présent dans l'installation de ROS, soit c'est un format de message personnalisé qui est spécifié dans un package. Pour cette application, presque tous les formats employés ont été créés sur mesure dans le package xsens_msg. Les formats de message peuvent être généré que dans des packages CMake. La commande pour en créer un est la suivante :

```
ros2 pkg create --build-type ament_cmake tutorial_interfaces
```

Dans ce cas, le fichier package.xml est le même que dans Python, les fichiers *setup* sont remplacés par le fichier *CMakeLists.txt* dans lequel doit être spécifié le nom de tous les fichiers de message créés. Le format de message précise le type et le nom de chacune des données à transmettre. A noter que depuis ROS2 des règles strictes régissent la façon de nommer ses données, notamment l'usage des majuscules.

```
string output_config
uint32 baudrate
string port_name
uint16 rtcm_refresh_dist
uint32[] sync_config
float64 dt_freq
```

Message de configuration de la Xsens. Publié par le nœud web_app et souscrit par xsens_data. Il s'agit donc d'un message destiné à des échanges sur un topic.

```
float64 longitude
float64 latitude
float64 alti_ell
---
```

Message de service. Les 3 premières lignes concernent le message de requête (la position pour une mise à jour des corrections RTCM). Sous les tirets est spécifié le format de la réponse. Dans le cas présent, rien n'est envoyé une fois le service terminé.

4.1.2.5 Installation et lancement

L'installation de l'application se fait depuis l'espace de travail. Au travers de la commande `colcon build`. Celle-ci va créer trois dossiers, /install, /build et /log. Exécutée comme telle, cette commande procède à l'installation de tous les packages présents dans /src. Une fois terminée, il est nécessaire d'ajouter les exécutables installés à l'environnement avec la commande : `. install/setup.bash`

La commande `colcon build` a souvent été employée avec l'option `--packages-select` qui permet de sélectionner quel package doit être installé / réinstallé. En effet, à chaque modification du code source d'un package, il est nécessaire de réinstaller l'entier du package pour le tester dans l'application. Au moment d'écrire ces lignes, l'option `--symlink-install` est portée à l'attention de l'auteur ! En répercutant directement les modifications du code source dans l'installation, elle pourrait bien être la solution à ces incessantes réinstallations.

Pour lancer le programme, plutôt que de démarrer chaque exécutable individuellement, il est possible de les lancer de manière coordonnée. Pour ce faire, il faut écrire un *launchfile* précisant quels exécutables sont concernés. Ce fichier doit se situer dans le dossier /launch d'un des packages. La syntaxe pour le lancer est la suivante :

```
ros2 launch <package_name> <launch_file_name>
```

Dans le cas présent, trois *launchfiles* ont été nécessaires :

- xsens_driver_launch.py sur le Raspberry I, lance les packages xsens_driver, central_data, ntrip_client
- rosbridge_websocket_launch.xml sur le Raspberry I, lance le package rosbridge_server
- cam_client_launch.py sur le Raspberry II, lance le package cam_client

Il serait théoriquement possible de n'avoir qu'un seul fichier de lancement sur le Raspberry I. Toutefois, le *launchfile* du package rosbridge est relativement complexe et écrit au format XML au lieu de Python. Par conséquent, la fusion des deux fichiers n'est pas aisée.

4.1.3 Dépendances

La majorité des dépendances nécessaire à la bonne exécution de l'application sont spécifiées dans le Dockerfile et donc préinstallées dans l'image employée pour ce projet. Leur liste est disponible en annexe.

Seuls deux packages n'ont pas pu faire l'objet d'une préinstallation. Il s'agit du programme de pilotage des caméras « tiscamera » et du package de conversion des images openCV vers ROS, « CvBridge ». Ces deux éléments doivent donc être réinstallés lors du portage de l'application vers une autre machine.

4.2 Description des nœuds

Dans cette partie, le rôle de chaque nœud est présenté. Il ne s'agit pas d'une description exhaustive de tous les éléments du programme, seuls les points d'intérêt font l'objet d'un développement.

4.2.1 web_app

C'est via ce nœud que l'utilisateur peut interagir avec le système et visualiser les données mesurées. Le nœud s'occupe de l'envoi des configurations à chacun des autres nœuds en relation avec un service (ntrip_client) ou du matériel (xsens_meaure, cam_client). Il récupère les données publiées pour les afficher sur l'application web et informe sur le statut des différents composants du système.

Publications	Suscription
· /xsens_config	· /xsens_status
· /ntrip_config	· /ntrip_status
· /cam_config	· /cam_status
	· /xsens_data

Comme le nœud est hébergé sur un serveur web, la communication avec les autres nœuds de l'application doit se faire via une passerelle [9]. Cette intermédiaire c'est le « méta-package » ros_bridge qui fournit un protocole d'échange entre ROS et n'importe quel client externe capable d'émettre des messages au format JSON. Plusieurs packages composent la suite ros_bridge. Les deux les principaux sont [19]:

- rosbridge_library contient une API Python qui reçoit les données au format JSON et contrôle les publishers, subscribers et service pour les transmettre dans ROS.
- rosbridge_server contient le serveur Websocket qui donne accès à ros_bridge_library

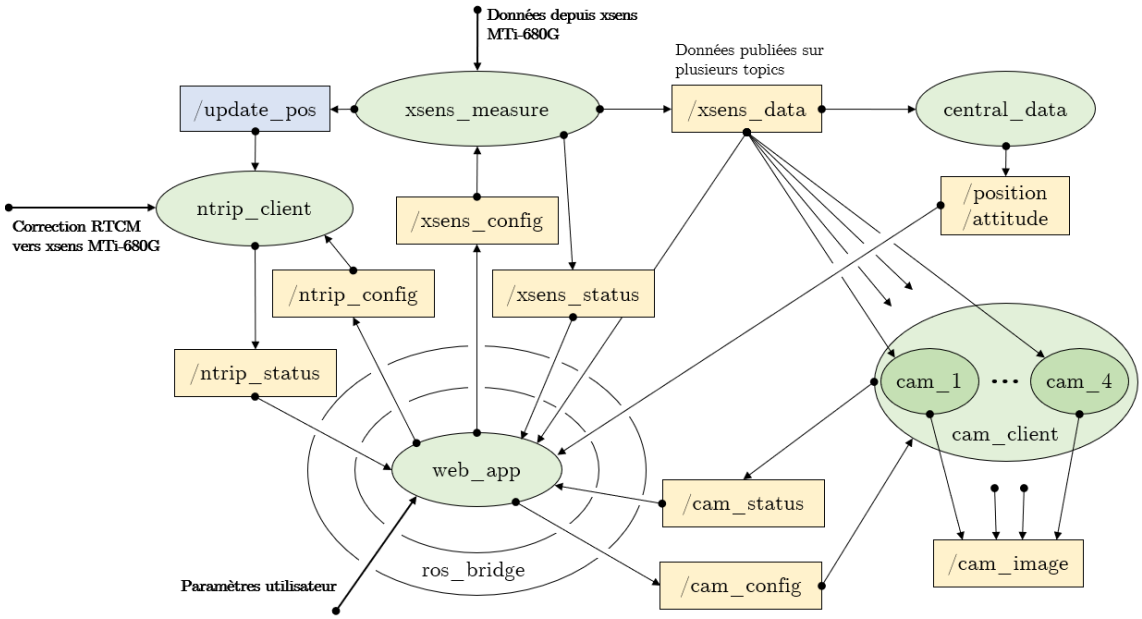


Figure 16 : Architecture ROS

4.2.2 xsens_measure

Ce nœud s'occupe de la communication avec la centrale inertielle Xsens MTi-680G. Initialement, le nœud est « inactif » et attend son initialisation avec le démarrage du nœud *web_app*. Une fois initialisé, le statut passe à « déconnecté », dans l'attente de sa configuration. La configuration reçue lui permet de se connecter à la centrale inertielle, spécifier les paramètres de synchronisation ainsi que les données à mesurer et leur format. Le statut passe ensuite à « actif » et le nœud commence à publier les données reçues. Il se charge également de demander une mise à jour de la position utilisée auprès du service NTRIP lors de déplacement au-delà d'un seuil choisi.

Publications	Suscription	Service
· /xsens_data · /xsens_status	· /xsens_config	· /update_pos (client)

4.2.2.1 Configuration

La configuration de la plateforme Xsens consiste principalement à spécifier les paramètres de connexion, indiquer quelles données obtenir et le mode de synchronisation. Pour établir le lien avec la Xsens, doivent être spécifiés le baud et le nom du port employé. Si aucun port n'est spécifié, une recherche automatique est effectuée.

Le choix des données à fournir en sortie est spécifié par une ligne de caractère comprenant le nom du groupe de donnée, le type, la fréquence désirée et éventuellement des indications de format. Par exemple : « *oe10e, aa100, iu* » permet d'obtenir les angles d'Euler à 10 Hz dans

la convention ENU, l'accélération à 10 Hz et un timestamp en UTC time pour chaque message. La liste complète est fournie en annexe.

4.2.2.2 Déclenchement des caméras

Pour le déclenchement des caméras, une connexion GPIO est établie sur les canaux 23 et 24. Deux cas de figure peuvent se présenter. En effet, comme dit précédemment, les caméras peuvent être déclenchées :

- depuis le Raspberry. Dans ce cas, la sortie du port 24 est mise sur « HIGH » pour indiquer que c'est le Raspberry qui déclenche. Un *timer* est créé avec pour fonction d'envoyer une impulsion sur le port 23 à l'intervalle fixé. La Xsens est configurée pour recevoir le même pulse sur sa ligne SyncIn. La fonction « Trigger Indication » est choisie pour qu'une mesure soit prise lors du déclenchement des caméras.

- depuis la plateforme Xsens via un port dédié. Dans ce cas, la sortie du port 24 est mise sur « LOW » pour indiquer que c'est la Xsens qui déclenche. Cette dernière est configurée pour envoyer une impulsion à l'intervalle choisi via sa ligne SyncOut.

La configuration des options de synchronisation consiste en une suite de chiffres se rapportant au mode choisi pour différents paramètres. Par exemple : « 3,2,1,0,0,0,10,0 » correspond à la fonction « Trigger Indication », sur la ligne SynIn1, avec une polarité positive, avec plusieurs pulses et différentes options concernant le pulse lui-même. L'ensemble des codes peuvent être retrouvés dans la documentation [13] p.26. A noter qu'il semble indispensable d'envoyer les deux configurations suivantes avant de pouvoir spécifier sa propre configuration :

(ClockBiasEstimation, 10, 1, 0, 0, 0, 0, 1000) et (OnePpsTimePulse, 10, 1, 0, 0, 0, 500, 0)

4.2.2.3 Réception des mesures

A chaque itération de la boucle « while » correspondant au statut « Active » du nœud, les fonctions callback sont exécutées via la commande *spin_once* puis une fonction vient lire sur le port Host les nouveaux messages en provenance de la Xsens. Le message est ensuite traité puis publié dans le format spécifié dans *xsens_msg*. Des données désirées à même fréquence seront envoyées dans le même message avec le même timestamp.

Si initialement la fréquence maximum de réception des données était de 25 Hz, il a été possible d'augmenter cette valeur en faisant passer le baud de 115200 bps à 460800 bps. Malgré cela, il faut choisir le nombre de données à fournir et leur fréquence avec soin. En cas de requêtes trop ambitieuses, le système émet l'erreur suivante :

MTException("Ack (0x%02X) expected, MID 0x%02X received

Cette dernière n'a pas pu être résolue et requiert une réinitialisation du système pour pouvoir à nouveau communiquer avec la plateforme Xsens. Le Tableau 12 présente quelques cas de figures avec les fréquences maximums qui ont pu être atteintes et le temps mis pour lire sur le port. Il s'agit de données acquises avec les nœuds xsens_measure et nptrip_client actifs, mais sans calcul de trajectoires en simultané.

Remarques : Il faut être prudent lors de la modification du baud. En effet, seules quelques valeurs spécifiées dans la documentation peuvent être employées. En cas d'utilisation d'une valeur erronée, il n'est plus possible de se connecter à la Xsens et une réinitialisation selon une méthode particulière est nécessaire.

Quelques questions restent toutefois en suspens par rapport au fonctionnement du flux de donnée entre la Xsens et le nœud ROS. En particulier, lors de l'acquisition sur le terrain la configuration correspondait à la dernière ligne du Tableau 12 (cinq données souhaitées dont GNSS et enregistrement via `ros2 bag record`). Or les fréquences indiquées ici n'ont pas pu être atteintes (voir Tableau 14). En effet, à cette occasion, il a été constaté qu'au-delà d'une certaine fréquence, le taux d'acquisition des données n'est pas constant et peut chuter drastiquement après un laps de temps relativement court.

Tableau 12 : Fréquences moyennes atteintes en fonction du nombre de données demandée

Fréquence souhaitée [Hz]	Fréquence obtenue [Hz]	Ecart-type fréquence [Hz]	Temps de lecture moyen [ms]	Données souhaitées
500	424.8	36.97	1.41	2 (Acc.+ Time)
400	310.13	17.59	1.70	5 (Acc.+ Time + Pos + Gyr + Ori.)
400	306.88	22.32	1.85	5 (Acc.+ Time + Pos + Gyr + GNSS.)
300	233.34	55.39	0.89	5 (Acc.+ Time + Pos + Gyr + GNSS.) + Enregistrement des données

Le cas représenté sur la Figure 17 correspond à cette situation où, au cours de l'utilisation, le temps de lecture sur le port passe d'environ 70 millisecondes à 1 seconde voir plus. Après plusieurs tests, il apparait que ce type de comportement se produit lorsque peu de mémoire est disponible sur le Raspberry. En effet, lorsque le système est lancé après un redémarrage de l'ordinateur, tout se déroule normalement, le processeur « tourne » à environ 85% de sa capacité et laisse 1.3 Go de mémoire libre pour d'autres processus. Lors de la collecte des donnée présentées ci-dessous, seuls 380 Mo de mémoire étaient disponibles et le processeur ne « tournait » plus qu'à 65% de sa capacité. Il serait intéressant de chercher à comprendre pourquoi la mémoire se rempli de la sorte et comment l'éviter.

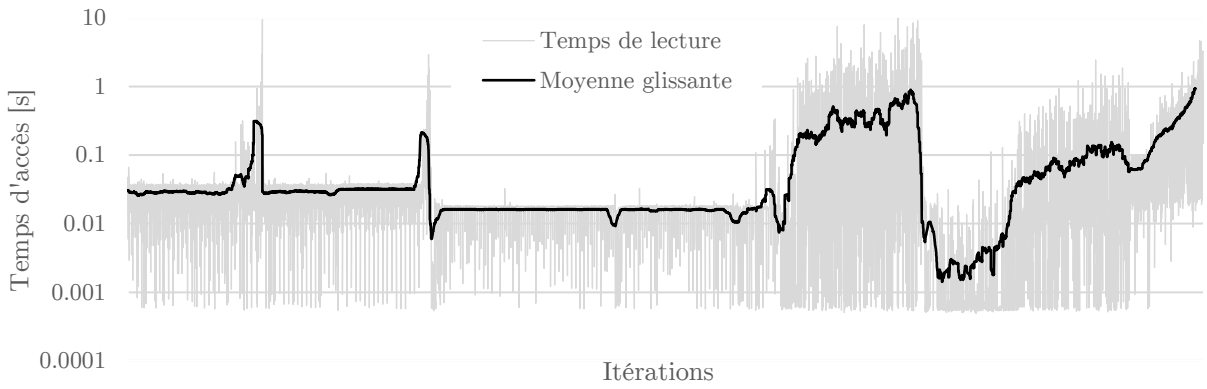


Figure 17 : Temps de lecture d'un message de mesure en provenance de la Xsens.

Dans le standard RS232 employé pour l'envoi des mesures, il est commun d'avoir un protocole de gestion de flux nommé RTS/CTS (Request to Send / Clear to Send). Ce dernier permet d'indiquer que l'émetteur est prêt à transmettre (RTS) et de s'assurer que le récepteur est prêt à recevoir (CTS). Comme le problème de transmission fait penser à des données qui s'accumuleraient sans pouvoir être lues, une autre piste serait de regarder si ce protocole pourrait en être la cause.

4.2.2.4 Requête NTRIP

Lors de la publication de nouvelles données de positions, un test est réalisé sur la distance séparant le point actuel avec le point de la dernière correction RTCM. Au-delà du seuil fixé, une requête de mise à jour est envoyée au nœud ntrip_client avec les coordonnées du nouveau point.

4.2.3 ntrip_client

Ce nœud s'occupe de la transmission des corrections RTCM. Le nœud est initialement « inactif » jusqu'au démarrage du nœud *web_app*. Il s'initialise ensuite et passe au statut « déconnecté » jusqu'à réception de ses données de configuration. Celles-ci lui permettent d'établir une connexion avec le service NTRIP choisi et de se connecter au port dédié à l'envoi des corrections RTCM sur la centrale inertielle. Cette étape franchie, le nœud passe au statut « actif ». Dans ce mode, le nœud envoie en continu les corrections RTCM reçue et, si demandé, s'occupe d'ajuster la position utilisée pour les requêtes auprès du service NTRIP.

Publications	Suscription	Service
· /ntrip_status	· /ntrip_config	· /update_pos (serveur)

4.2.4 central_data

Le traitement des données reçues de la centrale inertielle est réalisé dans ce nœud. Il s'agit principalement de la mécanisation et de l'estimation de la position, de la vitesse et l'attitude du système via un filtre de Kalman.

Le nœud étant abonné aux topics accélération, vitesse angulaire et données GNSS, il a été choisi que ça soit la disponibilité d'une mesure des accéléromètres et des gyroscopes qui déclenche l'algorithme d'estimation de la trajectoire du système. Ensuite, deux options ont été testées pour définir le rythme de l'algorithme :

1. Un *timer* est lancé au terme de l'initialisation du filtre. A une fréquence choisie, il exécute une fonction *callback* qui suit l'algorithme décrit sur la Figure 18. Cette méthode à l'avantage du choix de la fréquence, ce qui permet d'obtenir des données au rythme voulu. En revanche, la gestion du temps est plus complexe. Il est nécessaire de prévoir un compteur basé sur le temps machine pour calculer le laps de temps écoulé entre deux itérations dans le cas où aucune nouvelle donnée inertielle n'est disponible. De plus, nous avons vu qu'en itérant plus rapidement que la fréquence des données inertielles, en raison de la réinitialisation du vecteur d'état, le nouvel état ne différera pas du précédent (2.4).

2. A l'intérieur d'une boucle, tous les *callbacks* sont régulièrement testés à l'aide de la commande `spin_once` de ROS. Lorsqu'un nouveau message contenant des données inertielles est reçu, une nouvelle itération est lancée. La première partie de l'algorithme décrit sur la Figure 18 n'est donc plus applicable. En effet, à chaque étape la mécanisation est réalisée avec un Δt basé sur le timestamp des messages. C'est cette méthode qui a été choisie pour les raisons évoquées au point 1 en plus du fait qu'elle est potentiellement moins gourmande en ressources.

Il faut préciser que dans l'état actuel il est nécessaire que les données d'accélération et de vitesse angulaire soient disponibles simultanément. Il n'est donc pas possible de choisir des fréquences différentes pour ce deux topics. Ainsi une avarie sur un des deux capteurs provoque une erreur dans le système.

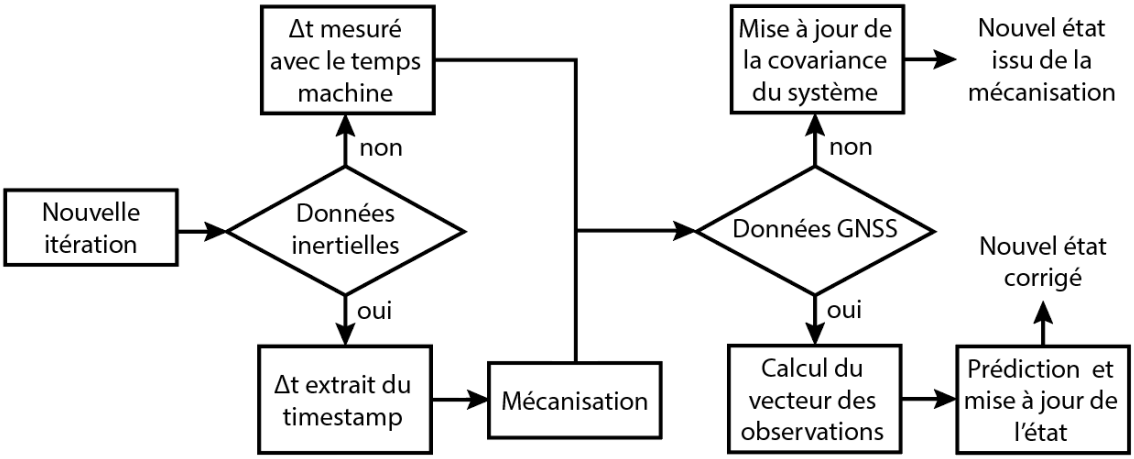


Figure 18 : Algorithme en fonction de la disponibilité des données inertielles et GNSS

Dans l'état actuel, les conditions initiales de l'algorithme doivent être entrées directement dans le code source du nœud. Cela n'étant pas idéal, l'idée à terme serait de les récupérer depuis l'application web comme pour la configuration des autres nœuds.

Publications	Suscription
· /position	· /xsens_data
· /attitude	

4.2.5 cam_client

Le nœud cam_client s'occupe de la gestion des caméras. Il s'assure de la connexion et de la configuration de chacune des caméras selon les paramètres publiés par le nœud web_app. Puis de la capture et de la publication des images. Il est en fait constitué d'un nœud principal et de quatre « sous nœuds », en charge de la publication des images de chaque caméra.

Publications	Suscription
· /cam_image	· /cam_config
· /cam_status	· /xsens_data

4.2.5.1 Configuration

Une fois la configuration reçue, le nœud recherche toutes les caméras et, le cas échéant, attend que les quatre appareils soient détectés. Il instancie ensuite 4 objets de la classe *Camera* en spécifiant pour chacun la résolution et le framerate voulu. D'autres propriétés pourraient être spécifiées à ce moment-là comme par exemple, le temps d'exposition ou le gain.

4.2.5.2 Capture des images

Lors de la création de chaque *Camera* une fonction callback lui ait ajoutée. Cette fonction est exécutée lors du déclenchement de la caméra. Le mode de déclenchement doit lui aussi être activé selon la procédure spécifiée par le fabricant [21].

Lorsqu'un signal de déclenchement est envoyé par la Xsens ou le Raspberry, via le callback, la caméra passe au statut « occupée » et une image est saisie au format openCV. Libre ensuite d'enregistrer cette image en local ou de la publier.

Pour que chaque image puisse être associée à une position / orientation du système, le nœud récupère le timestamp de la dernière donnée publiée par la Xsens et, partant de là, compte en temps machine le temps jusqu'à la capture de l'image.

4.2.5.3 Publication des images

Il a été choisi de publier les images capturées bien que dans l'état actuel du projet, il n'existe pas de nœuds responsables du traitement des images. Si un jour un tel nœud est développé, il n'aura qu'à souscrire au topic /cam_image pour recevoir un flux de photo en provenance des caméras.

Il été essayer de publier les images dans un format de message ROS personnalisé, mais cela n'est pas aisé avec une image au format openCV. Par conséquent, la librairie CvBridge issue de la communauté ROS a été préférée. Celle-ci rend possible la conversion d'images openCV vers le format de message ROS sensor_msgs/Image et inversement. Si plusieurs formats d'encodage sont disponibles, ici il a été choisi de publier en échelle de gris sur 8 bits [20].

4.3 Interface web

4.3.1 Description

L'interface web est une application qui permet de configurer et contrôler l'entier du système. Elle est basée sur la plateforme *Node.js* et le framework *Vite* qui met à disposition un serveur de développement simple et rapide.

Les librairies suivantes sont nécessaires à l'affichage de la page :

- roslibjs est une API JavaScript qui, du côté client, permet de communiquer avec le ros_bridge au format JSON.
- plotly.js permet entre autres d'afficher des données graphiques de manière dynamique.
- bootstrap permet de mettre en formes les différents éléments d'une page web de manière simple grâce à une grande bibliothèque d'objets.

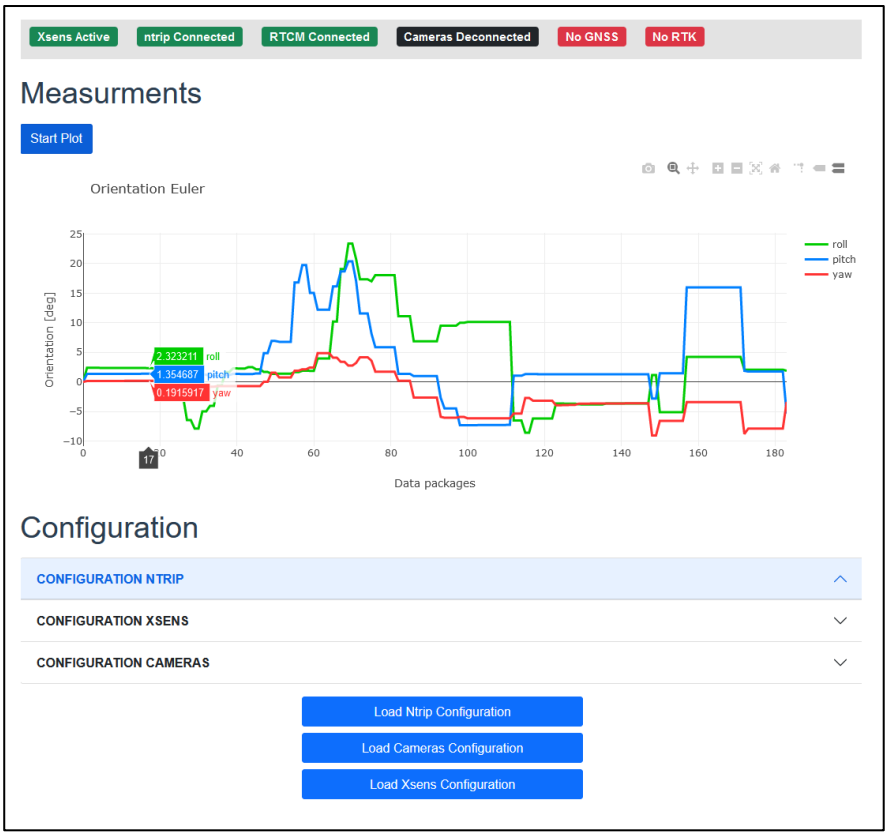
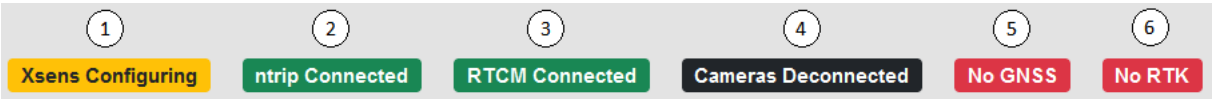


Figure 19 : Interface utilisateur

La commande `npm run dev -- --port 8000 -host` permet de lancer le serveur Vite. Il est possible d'y accéder localement à htpp://localhost :8000/ ou depuis une machine située sur le même réseau à l'adresse htpp://[adresse IP du Raspberry I]:8000/

4.3.2 Composition de l'interface

L'interface est composée d'un bandeau dans lequel est affiché le statut des différents composants du système.



1	Etat de la connexion avec la Xsens	4	Etat de la connexion des caméra
2	Etat de la connexion avec le service NTRIP	5	Indique la présence ou nom d'un signal GNSS
3	Etat de la connexion pour l'envoi des corrections RTCM	6	Indique si un calcul de phase à lieu et si la solution est flottante ou entière

Ensuite, une zone permet de visualiser graphiquement un flux de donnée. Actuellement, c'est l'orientation calculée par la Xsens ainsi que les données de positions qui sont montrée. Il est bien sûr possible de changer le flux de données affiché en souscrivant à d'autres topics. Pour le moment, cela demande une intervention dans le code source de la page. Une fois démarré,

il n'est pas possible de suspendre ou stopper l'affichage des données. Plusieurs outils pour interagir avec le graphique sont proposés par plotly.js.

Trois écrans déroulants servent à la configuration du service NTRIP, de la Xsens et des caméras.

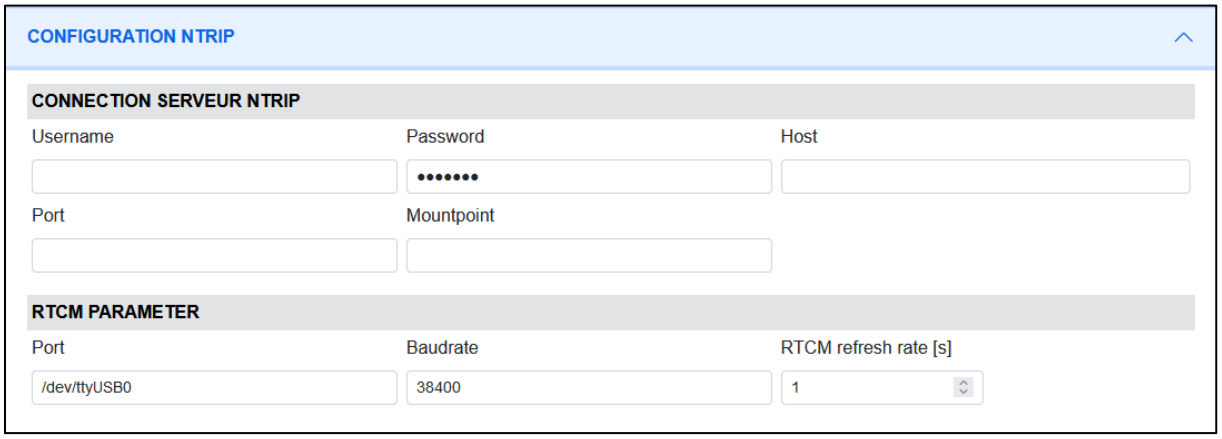


Figure 20 : Configuration du service NTRIP

La Figure 20 correspond au masque de configuration pour le service NTRIP et l'envoi des corrections RTCM. Cette configuration est envoyée au nœud ntrip_client. On y trouve les informations de connexion au serveur NTRIP, les caractéristiques du port employé pour envoyer les corrections RTCM ainsi que la fréquence voulu pour l'envoi des corrections vers la plateforme Xsens.

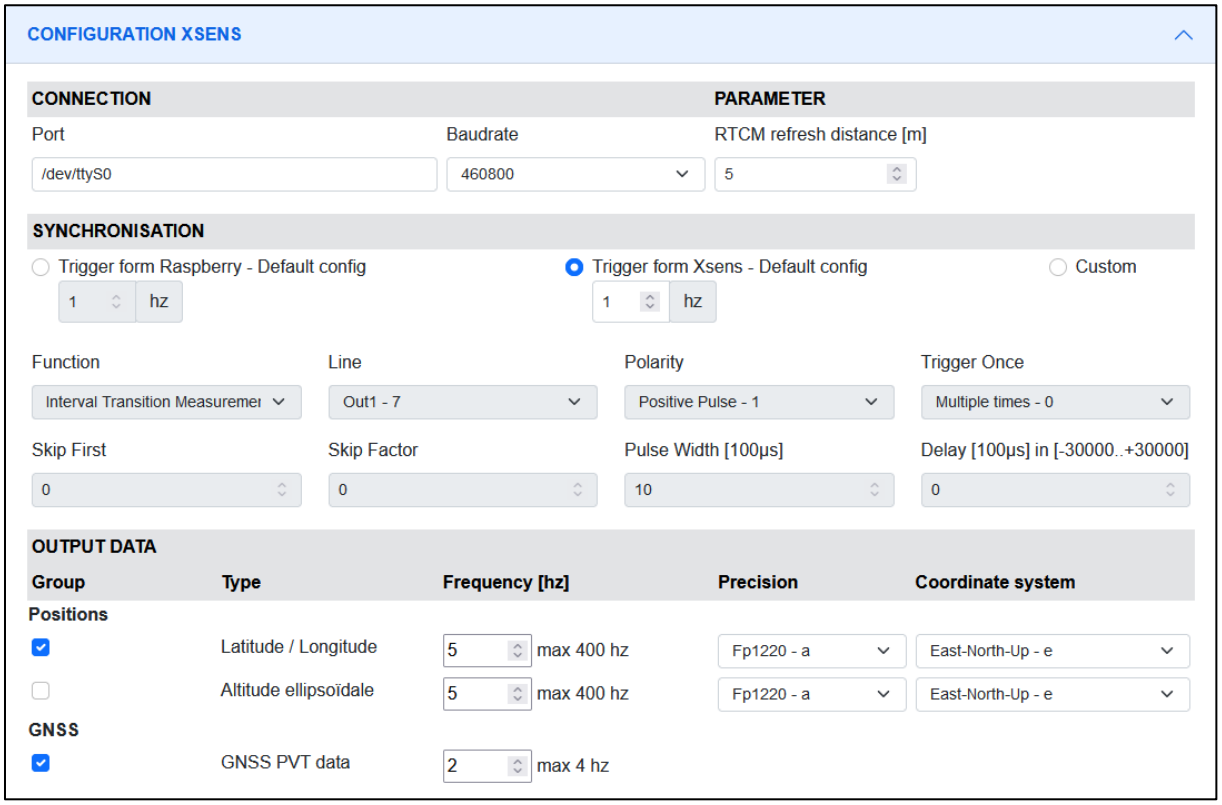


Figure 21 : Masque de configuration de la Xsens

La Figure 21 présente les nombreuses options de configuration de la plateforme Xsens. En premier lieu les caractéristiques de la connexion et à partir de quel déplacement une nouvelle correction RTCM doit être requise. Ensuite viennent les options de synchronisation avec deux préréglages et la possibilité de choisir une configuration personnalisée. Finalement, le choix, la fréquence et le format des données à fournir peuvent être précisés. Pour réduire la taille de la figure, seules 3 données sur les 11 disponibles sont montrées. Pour toutes les données proposées par la Xsens mais absentes de l'interface, un format de message et un publisher doivent être définis dans xsens_msg et xsens_driver. Un travail à faire si par exemple les données des magnétomètres étaient souhaitées.

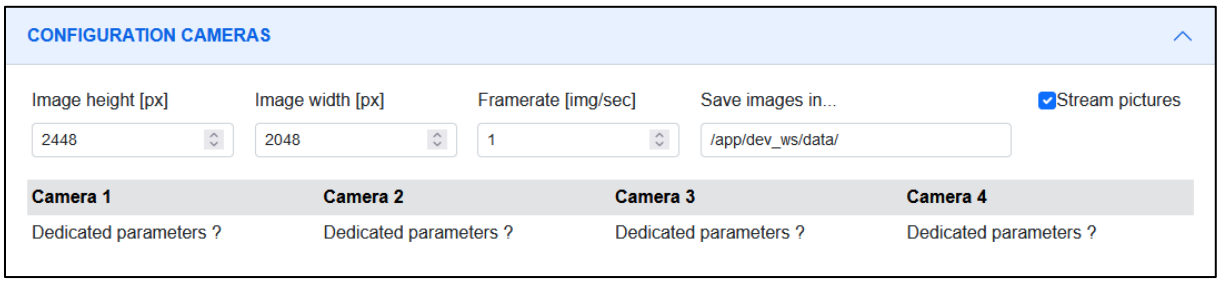


Figure 22 : Masque de configuration des caméras

Le masque de configuration des caméras (Figure 22) permet de choisir la résolution des images, le dossier de sortie si les images devaient être stockées sur un disque et la fréquence de déclenchement qui serait utilisée en l'absence de déclenchement automatique. Il a été prévu un espace pour spécifier des paramètres individuellement pour chaque caméra. Si cette option est utilisée, il faudra réfléchir à la manière de les publier. Soit comme maintenant c'est à dire envoyer un message avec toutes les configurations, soit publier sur quatre topics différents. Dans le même esprit, la coche « Stream pictures » n'est pas active pour le moment. Elle a été ajoutée avec l'idée de récupérer les images publiées sur /cam_images pour les afficher dans l'interface.

Dernier élément de l'interface, trois boutons permettent de publier chacune des configurations afin d'initialiser les nœuds respectifs ou changer leurs réglages en cours d'exécution.

4.4 Emploi du système

4.4.1 Installation

Pour les deux Raspberry la procédure est identique, à l'exception du point 5 qui est requis uniquement pour le Raspberry II.

1. Disposer de l'image Docker de l'application
2. Décider comment les fichiers créés à l'intérieur du conteneur seront récupérés entre deux démarrages (utilisation d'un volume, ne pas supprimer le conteneur à sa fermeture, autre).
3. Suivre les étapes 1 à 5 de la procédure de démarrage ci-dessous.
4. Télécharger l'application depuis le répertoire [23]
5. Installer les packages « tiscamera » et « CvBridge » [21] et [20]

Remarque : Depuis le 1 juin 2022, une nouvelle version de « tiscamera » est disponible.

4.4.2 Démarrage

La procédure de démarrage et d'emploi du système est détaillée ici point après point :

1. Le réseau servant à la communication des Raspberry entre eux doit être disponible
2. Démarrage des Raspberry I et II (RI et RII) et accès à leur bureau.
3. Dans un terminal, désactiver le contrôle d'accès avec `xhost +` pour avoir accès au serveur X depuis le conteneur Docker
4. Démarrage des conteneurs :

   RI : `docker run -it --name dev_ros --env="DISPLAY" --net=host --pid=host --device=/dev/ttyS0 --device=/dev/ttyUSB0 --device=/dev/mem:/dev/mem --device=/dev/gpiomem:/dev/gpiomem -p 8000:8000 -p 9090:9090 -v /tmp/.X11-unix/:/tmp/.X11-unix -v ros_vol:/app --rm ros_app:11.0`

   RII : `docker run -it --name dev_ros --env="DISPLAY" --net=host --pid=host --device=/dev/video0 --device=/dev/video1 --device=/dev/video2 --device=/dev/video3 --device=/dev/video4 --device=/dev/video5 --device=/dev/video6 --device=/dev/video7 --device=/dev/mem:/dev/mem --device=/dev/gpiomem:/dev/gpiomem -p 8000:8000 -p 9090:9090 -v /tmp/.X11-unix/:/tmp/.X11-unix -v ros_vol:/app --rm ros_app:11.0`

5. Si nécessaire, se connecter au conteneur depuis d'autres terminaux avec :
   `docker exec -it dev_ros bash`
6. Sourcer l'application et le package tiscamera :
   RI : `source tiscamera/build/env.sh`
   RII : `source tiscamera/build/env.sh et source install/local_setup.bash`
7. Démarrer le serveur de développement Vite et se connecter à l'interface :
   Depuis le dossier webUI : `npm run dev -- --port 8000 –host`
   Dans un navigateur : http://[IP RI]:8000/
8. Démarrage des packages ROS:
   RI : `ros2 launch rosbridge_server rosbridge_websocket_launch.xml`
   RI : `ros2 launch xsens_driver xsens_driver_launch.py`
   RII : `ros2 launch cam_client cam_client_launch.py`
9. Configurer le système
10. Charger la configuration NTRIP puis attendre l'établissement de la connexion au service et pour les corrections RTCM.
11. Charger la configuration des caméras.
12. Charger la configuration Xsens. L'estimation de la position et de l'orientation du système commence. Lors de l'envoi de signaux de déclenchement, des images sont capturées.

Pour arrêter le système, l'exécution des packages xsens_driver et cam_client doit être stoppée depuis leurs terminaux respectifs.

5 Algorithme de navigation

5.1 Introduction

Comme décrit précédemment, la plateforme MTi-680G de Xsens est capable de fournir des informations de position et d'orientation à haute fréquence (400 Hz) sur la base des données collectées par ces différents capteurs. Toutefois, l'algorithme de fusion n'est que sommairement décrit, aucun indicateur statistique permettant de connaitre la fiabilité de l'estimation n'est disponible et il n'est pas possible d'intervenir dans l'algorithme si de nouveaux capteurs devaient être intégrés au système. Pour toutes ces raisons, il a été décidé de recalculer la position et l'attitude du système, en fonction des données brutes, issue, dans un premier temps, des accéléromètres et des gyroscopes de la Xsens. Cela implique donc le développement des algorithmes de mécanisation, d'estimation et d'intégration. Les données GNSS brutes n'étant pas disponibles, ce sont les observations fournies par la Xsens qui seront employées.

5.2 Méthode de développement

Dans cette section, il est décrit l'approche qui a été choisie pour aboutir à un algorithme capable de prédire la position, la vitesse et l'orientation d'un système à partir de données inertielles et GNSS.

5.2.1 Trajectoires simulées

Des jeux de données simulées ont été employés afin de pouvoir disposer de données « parfaites ». Du bruit, des biais et des facteurs d'échelles ont ensuite été ajoutés pour obtenir des jeux de données simulant le signal des capteurs inertiels. Disposer de données « parfaites » et de données « bruitées » permet d'évaluer l'efficacité de l'algorithme d'estimation par rapport à une trajectoire de référence. Un exercice qu'il est bien plus compliqué à mettre sur pied sur le terrain, avec des données réelles. Un autre avantage étant de pouvoir « jouer » ces données au rythme désiré, ce qui est moins aisé avec des données enregistrées dans ROS et évidemment impossible lors d'estimation en temps réel.

La méthode suivante a été suivie pour le calcul des données :

<u>Accélérations dans le repère local</u>

1. Dessin de la trajectoire voulue, discrétisation et export des coordonnées. Le format MN95 a été utilisé afin d'avoir des positions spécifiées en mètres. Chaque point correspondant à une donnée de l'IMU, la discrétisation a été faite de manière à obtenir une fréquence d'environ 50 Hz si la trajectoire est parcourue à la vitesse du pas. Plusieurs trajectoires différentes ont été employées dont deux exemples sont illustrés sur la Figure 23. Afin de rester concis, seront présentés uniquement les résultats de la trajectoire de droite qui, avec des changements de direction soudains, représente un plus grand défi.

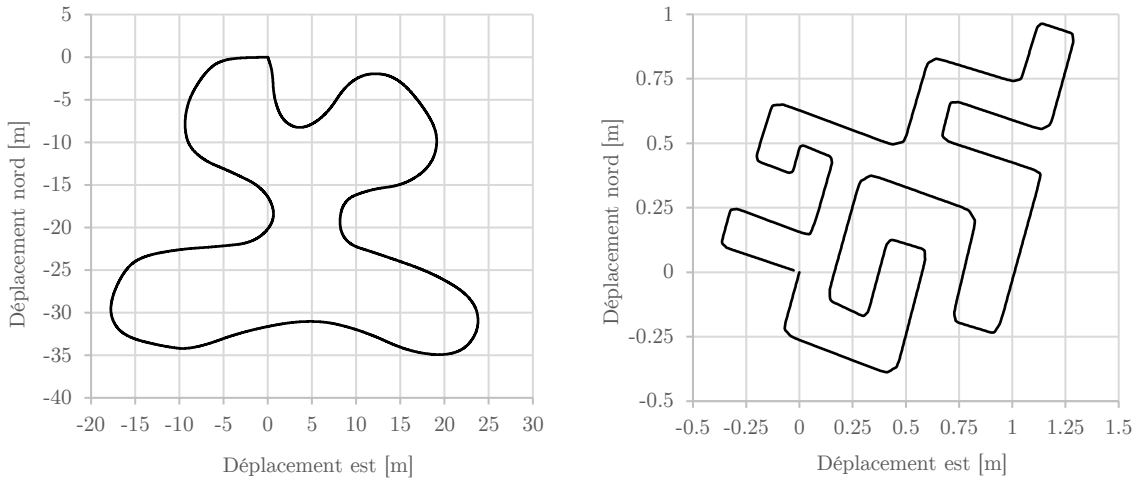


Figure 23 : Trajectoires de test

2. Dessin d'un profil d'altitude.
3. Dessin d'un profil temporel spécifiant l'instant du passage en chaque point.

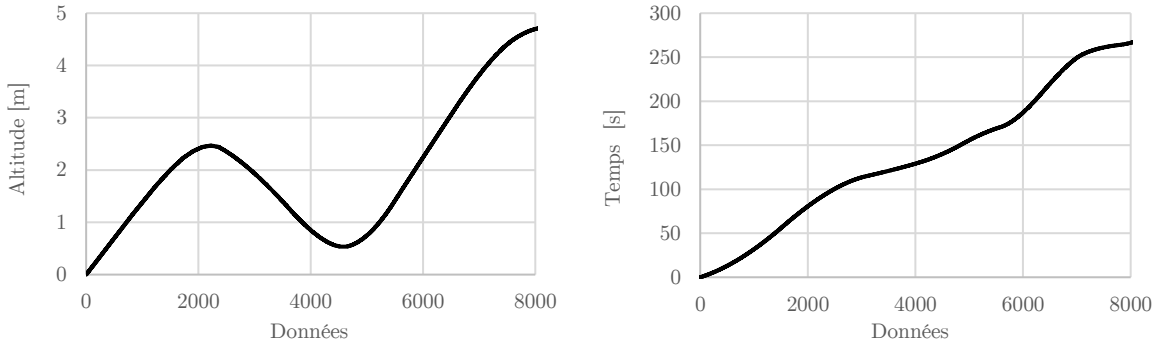


Figure 24 : Profils d'altitude à gauche et temporel à droite

4. Calcul des variations de position en (Est, Nord, Altitude) subies par le système et des différences de temps entre mesures.
5. Calcul des vitesses v_E, v_N, v_H puis des accélérations a_E, a_N, a_H avec l'hypothèse que les vitesses et accélérations initiales sont nulles

Vitesses angulaires dans le repère INS

6. Un premier jeu de donnée a été généré sans vitesses angulaires, avec l'hypothèse que le repère INS est aligné sur le repère local (Figure 25).
7. Dans le cas simple où le système subit une rotation selon un axe uniquement, il a été possible de définir un profil de rotation, ici sur le lacet, et de déduire simplement par intégration, la vitesse angulaire mesurée par le gyroscope sur cet axe. Les vitesses sur les deux autres axes étant nulles (Figure 26)

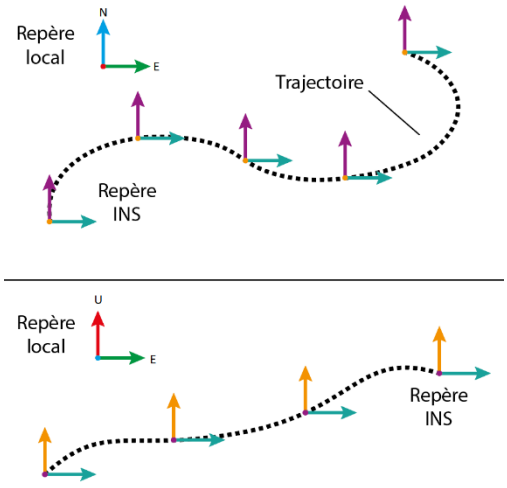


Figure 25 : Pas de rotation du système. Les repères INS et local sont alignés

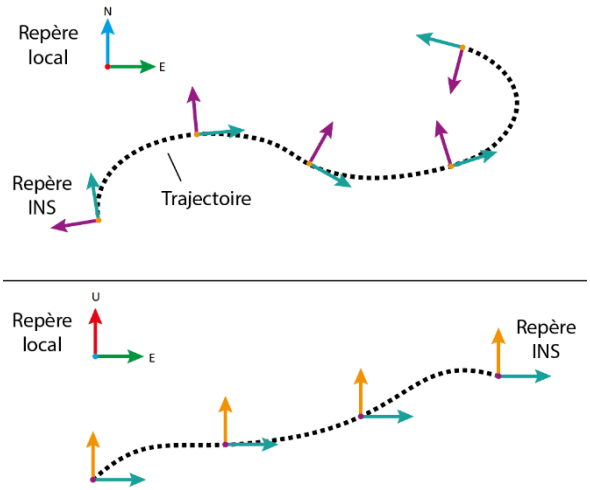


Figure 26 : Rotation en lacet seulement. La vitesse angulaire peut être déduite depuis l'angle choisi.

8. Dans le cas général où le système se déplace le long de la trajectoire avec une orientation qui varie selon les trois axes simultanément, il n'est plus possible d'intégrer sur les 3 axes des incréments d'angle qui auraient été choisis. Afin de simplifier le calcul, il a été choisi de fixer non pas l'orientation à chaque instant, mais directement les trois vitesses angulaires mesurées par les gyroscopes. L'orientation du système étant alors une conséquence de ces rotations. Arbitrairement, il a été choisi que l'axe X de la Xsens pointe en continu dans la direction du mouvement, raison pour laquelle sur la Figure 28 la vitesse angulaire w_z est plus bruitée que les deux autres.

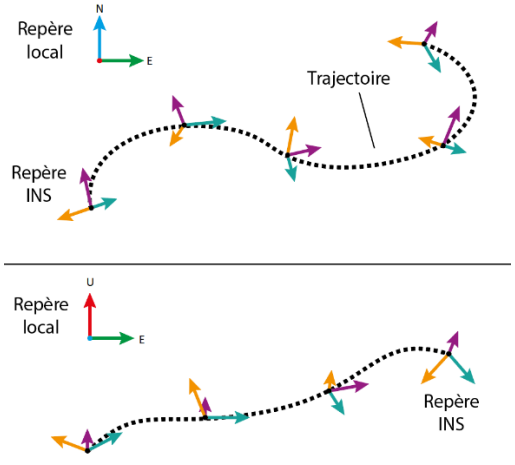


Figure 27 : Cas général, l'orientation du système varie selon ses trois axes

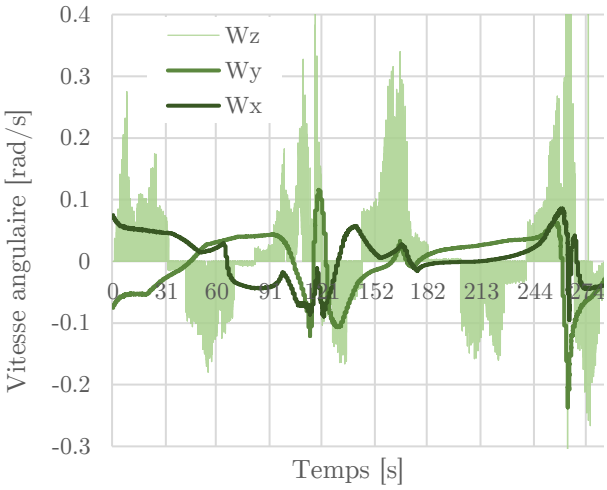


Figure 28 : Profils de vitesses angulaires dans le repère INS

Accélérations dans le repère INS

9. Choix d'une orientation initiale et calcul du quaternion et de la matrice de rotation initiale
10. A chaque pas de temps : intégration des vitesses angulaires, mise à jour du quaternion, calcul de la nouvelle matrice de rotation entre repères local et INS et finalement calcul des accélérations dans le repère INS en « tournant » les accélérations exprimées dans le repère local.

Ces données idéales s_i ont été utilisées pour connaitre la trajectoire de référence. En revanche, pour simuler les signaux de sortie s_c des capteurs, du bruit w et un biais b ont été ajouté :

$$s_c = s_i + w + b$$

Avec w un bruit suivant une loi normale centrée en 0 et d'un écart-type choisi pour être en accord avec l'état de l'art. Les valeurs utilisées sont présentées dans le Tableau 13. Les Figure 29 et Figure 30 donnent pour les valeurs d'accélération mesurées selon l'axe X du repère INS.

Tableau 13 : Valeurs de biais et de bruit pour la génération des observations

	Ecart-type	Biais
Accélérations	0.001 m/s^2	0.001 m/s^2
Vitesses angulaires	0.01 rad/s	0.001 rad/s

Il a été choisi un ratio 1/10 entre la fréquence des observations GNSS et celle des données inertielles.

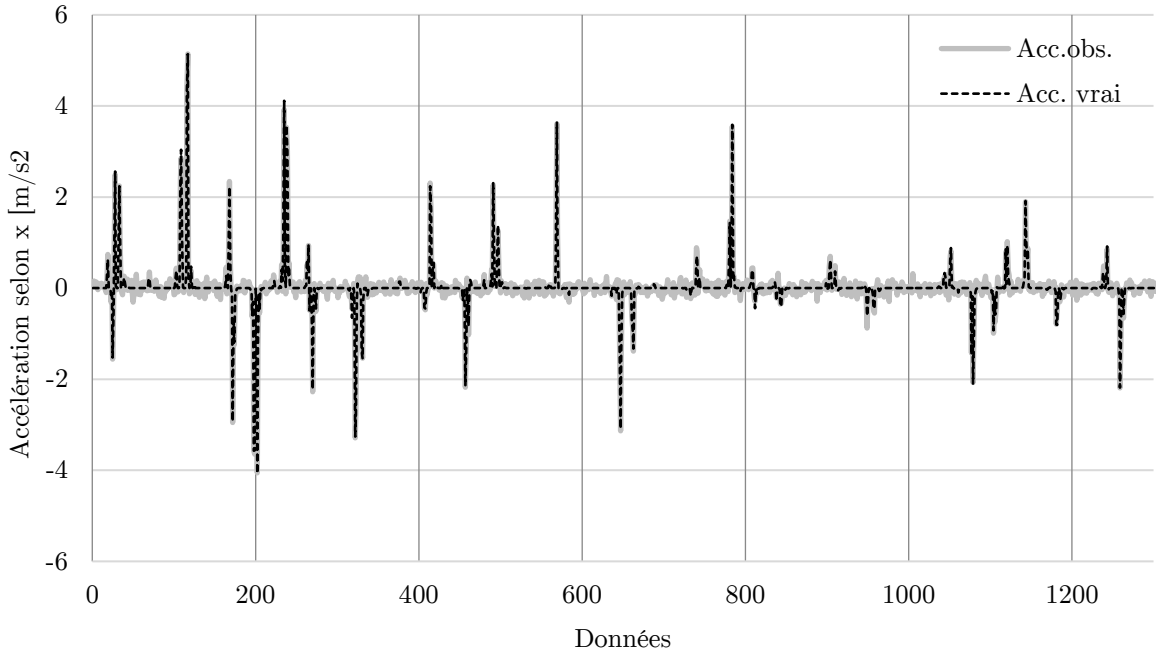


Figure 29 : Accélération selon l'axe X. Données vraies et les observations générées avec un biais de 0.001 m/s^2 et un écart-type de 0.001 m/s^2

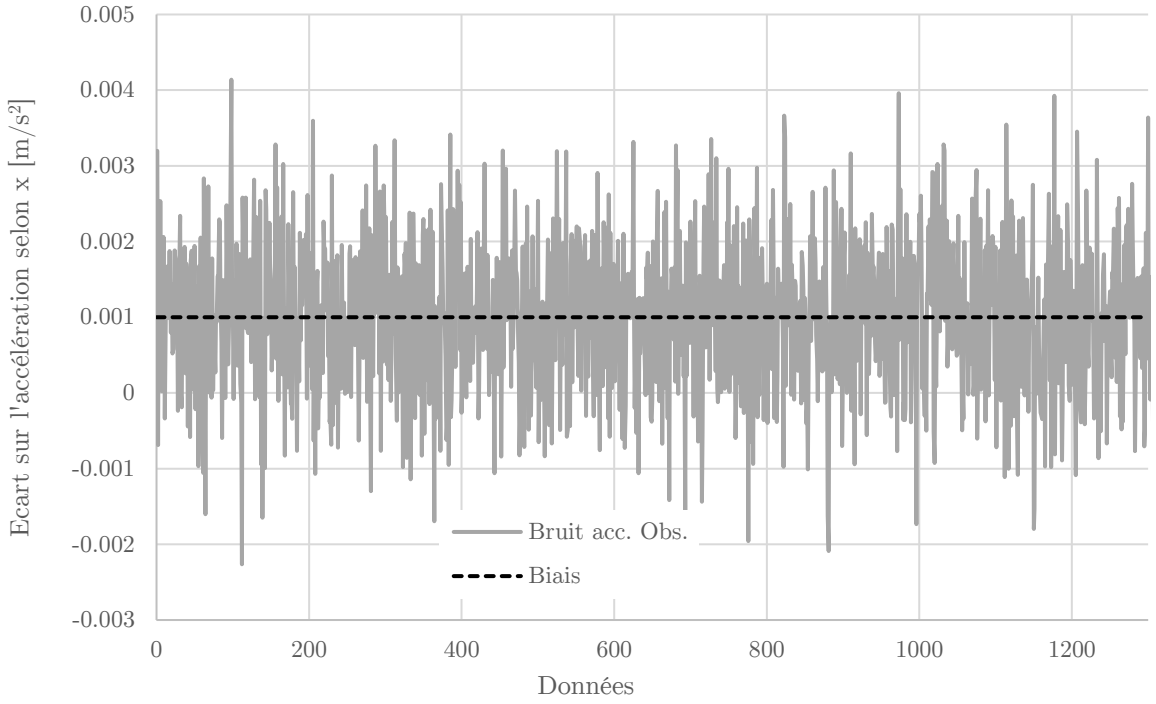


Figure 30 : Ecarts sur l'accélération selon l'axe x. Observations générées avec un biais de 0.001 m/s^2 et un écart-type de 0.001 m/s^2

Un processus par étapes a été suivi afin d'intégrer au fur et à mesure les différentes fonctionnalités. Le résumé des étapes et les résultats attendus sont décrits ci-dessous :

1. Mécanisation seulement, avec les données non-bruitée, repère INS aligné sur le repère local, pas de rotation. La trajectoire calculée doit correspondre à la trajectoire dessinée.
2. Mécanisation seulement, avec les données non bruitée, repère INS aligné sur le repère local, rotation en lacet seulement. La trajectoire calculée doit correspondre à la trajectoire dessinée, le lacet calculé doit correspondre à l'angle de lacet choisi.
3. Mécanisation seulement, avec les données non bruitée, rotation selon les trois axes avec les vitesses spécifiées. La trajectoire calculée doit correspondre à la trajectoire dessinée. La vérification de ce critère garanti que l'orientation calculée est-elle aussi correcte. En effet, cela signifie que les données d'accéléromètres sont bien transformées entre le repère INS et le repère local.
4. Mécanisation + filtre de Kalman sans modélisation du biais, avec les données bruitées mais sans biais. La trajectoire et l'orientation estimée suivent le cas idéal mais dérivent sensiblement
5. Mécanisation + filtre de Kalman sans modélisation du biais, avec les données bruitées mais sans biais. Simulation d'observations GNSS. La trajectoire et l'orientation estimée suivent le cas idéal, dérivent sensiblement et sont corrigées lorsqu'une observation GNSS est disponible
6. Mécanisation + filtre de Kalman avec modélisation du biais, avec les données bruitées et biaisées. La trajectoire et l'orientation estimée suivent le cas idéal.

7. Finalisation : prise en compte du bras de levier entre l'antenne GNSS et le lieu des mesures dans l'IMU, contrôle de la symétrie des matrices de covariances.

Les corrections liées à la gravité, aux forces de Coriolis et à la rotation de la Terre par rapport au repère inertiel ont aussi été ajoutées pour être testées dans la perspective d'utiliser l'algorithme avec des données acquises en situation réelle. Ces effets n'ayant pas été pris en compte lors de la création des données, leur correction en simulation péjore l'estimation de la trajectoire et de l'orientation.

5.2.2 Trajectoire terrain

L'utilisation de ROS pour opérer l'entier du système permet de faire appel à une fonctionnalité de stockage des mesures. Cela permet de rejouer une série de données acquises en situation réelle. Pour effectuer les tests prévus, il était important de disposer des données d'accélération, de vitesse angulaire et GNSS, mais également la position et l'orientation calculées par la plateforme Xsens afin de pouvoir confronter les données entre elles. La configuration suivante a donc été utilisée :

Tableau 14 : Configuration choisie pour l'acquisition des données

Paramètre	Fréquence désirée	Fréquence obtenue
Accélération	50 Hz	49.32 Hz
Vitesse angulaire	50 Hz	49.32 Hz
Position / Vitesse GNSS	4 Hz	4.00 Hz
Position Xsens	10 Hz	9.89 Hz
Orientation Xsens	5 Hz	4.95 Hz

Etant limité en termes de fréquence par les performances du Raspberry Pi, il a été choisi de favoriser les données inertielles, desquelles la qualité de l'estimation est fortement tributaire par rapport aux données d'orientation et de position fournie par la plateforme Xsens. Toutefois, la fréquence obtenue pour ces dernières devrait rester suffisante pour réaliser une bonne comparaison entre les deux trajectoires.

Une boucle avec différents points de passage connus en planimétrie et altimétrie a été définie à l'avance afin de servir de référence pour les données GNSS, la trajectoire calculée par la plateforme Xsens et la trajectoire calculée par l'algorithme dont il est question ici. Il faut relever qu'en procédant de la sorte un contrôle de la position peut être fait, mais ne permet pas de valider l'orientation prédite. Pour ce faire, d'autres méthodes doivent être imaginées. Par exemple, une orientation de référence obtenue avec un ensemble d'antennes GNSS ou placer le système sur un bras robotisé lui faisant subir des rotations prédéfinies.

5.2.3 Plateforme de test

Trois options ont été envisagées pour tester l'algorithme de navigation complet :

1. Jouer les données en temps réel sur le Raspberry Pi employé pour opérer le système. Il faut alors de lancer le nœud de calcul, à savoir central_data pour qu'il s'exécute en continu dans l'attente de la publication d'un message sur le topic `mti/acceleration`.

Cela peut être simulé avec la commande `ros2 bag play bagfile_name` qui permet de rejouer des données enregistrées au préalable avec `ros2 bag record`.

Cette option s'est avérée inadaptée en raison des performances limitées du Raspberry Pi.

A noter que grâce à l'option `-rate` il est possible rejouer les données à une vitesse réduite et ainsi se départir de ce problème de fréquence. Toutefois, il aurait fallu réduire fortement le débit de données pour être sûr de n'en manquer aucune. Cela aurait fortement augmenté le temps de simulation, ce qui est problématique en phase développement.

2. Jouer les données selon la méthode précédente, mais sur un ordinateur plus puissant que le Raspberry Pi.

Le portage de tout le système d'exploitation du Raspberry vers le LENOVO n'aurait pas dû poser de problème étant donné qu'il suffisait de déplacer l'image Docker. Toutefois, dans les faits, le transfert de l'image n'a pas été si facile si bien qu'il a été choisi de reconstruire localement une nouvelle image à l'aide du `dockerfile`. Une fois cette nouvelle image démarrée et tous les fichiers requis transférés depuis Github, des problèmes d'affichage sont survenus, surement causés par l'absence de serveurX. Au vu de ces difficultés, cette option a été abandonnée. De plus, il n'est pas garanti que les données aient véritablement pu être jouées en temps réel.

3. Exporter au format CSV les données enregistrées avec `ros2 bag record` pour pouvoir les lire au rythme voulu, sur l'ordinateur de son choix. Etant enregistré dans une base de données SQL au format BLOB, il n'a pas été possible d'exporter directement le contenu de la base. Sur le Raspberry Pi, un script Python a donc été employé pour écrire dans des fichiers CSV toutes les données publiées sur les topics voulus. L'option -rate a été employée ici pour ralentir le débit et s'assurer que toutes les données publiées aient bien été écrites. Ces fichiers ont ensuite pu être transférés.

```
0000  00 01 00 00 04 00 00 00 0b 00 00 00 21 00 00 00  ............!...
0010  1f 00 00 00 26 1e 98 33 00 00 00 00 00 00 00 40  ....&..3.......@
0020  30 50 6b 3f 00 00 00 00 30 06 6f 3f 00 00 00 e0  0Pk?....0.o?....
0030  bf 1e 67 bf                                      ..g.
```

Figure 31 : Un message ROS au format BLOB affiché en mode binaire tel qu'il est stocké dans la base de données créée par `ros2 bag record`

C'est cette option qui a été retenue, en employant le LENOVO. En plus des raisons évoquées plus haut, lorsqu'il s'agit de traiter des données et développer un algorithme tel qui celui dont il est question ici, cette option a l'avantage d'offrir un environnement de travail plus optimal qu'en lignes de commande à l'intérieur d'un conteneur Docker. Cette approche permet aussi de plus facilement simuler, par exemple une perte momentanée du signal GNSS.

5.3 Résultats et analyse

Les résultats des tests réalisés sur les différents jeux de données sont présentés dans cette section.

5.3.1 Trajectoires simulées

Les figures ci-dessous présentent les trajectoires et orientations prédites avec et sans utilisation du filtre de Kalman. Ainsi est représentée en rouge la trajectoire idéale calculée par mécanisation avec les données non bruitées et en pointillés la trajectoire estimée.

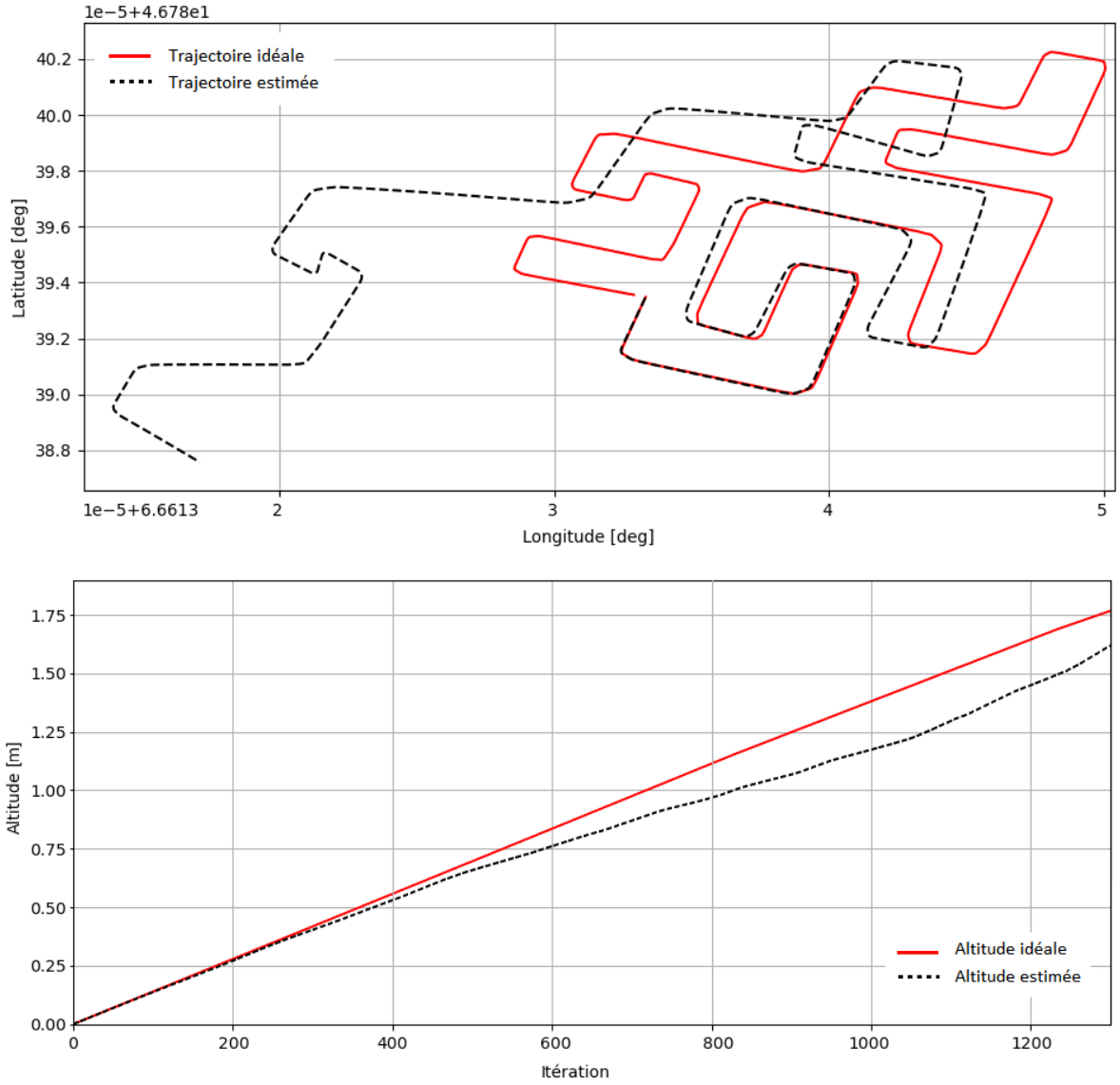


Figure 32 : Trajectoire et profil d'altitude estimée par mécanisation seulement

Les Figure 32 et Figure 33 correspondent au cas où l'état est estimé uniquement par mécanisation donc sans utilisation de filtre. Une dérive apparait tant en position qu'en orientation. Elle est le fruit du biais et du bruit introduit dans les données. L'écart est plus important sur la position, notamment car les données d'accélération sont basées sur les données d'orientation. Ainsi les erreurs se cumulent. Les Figure 34, Figure 35 et Figure 36 illustrent l'estimation de l'état avec les mêmes données bruitées mais cette fois à l'aide du filtre de Kalman décrit précédemment.

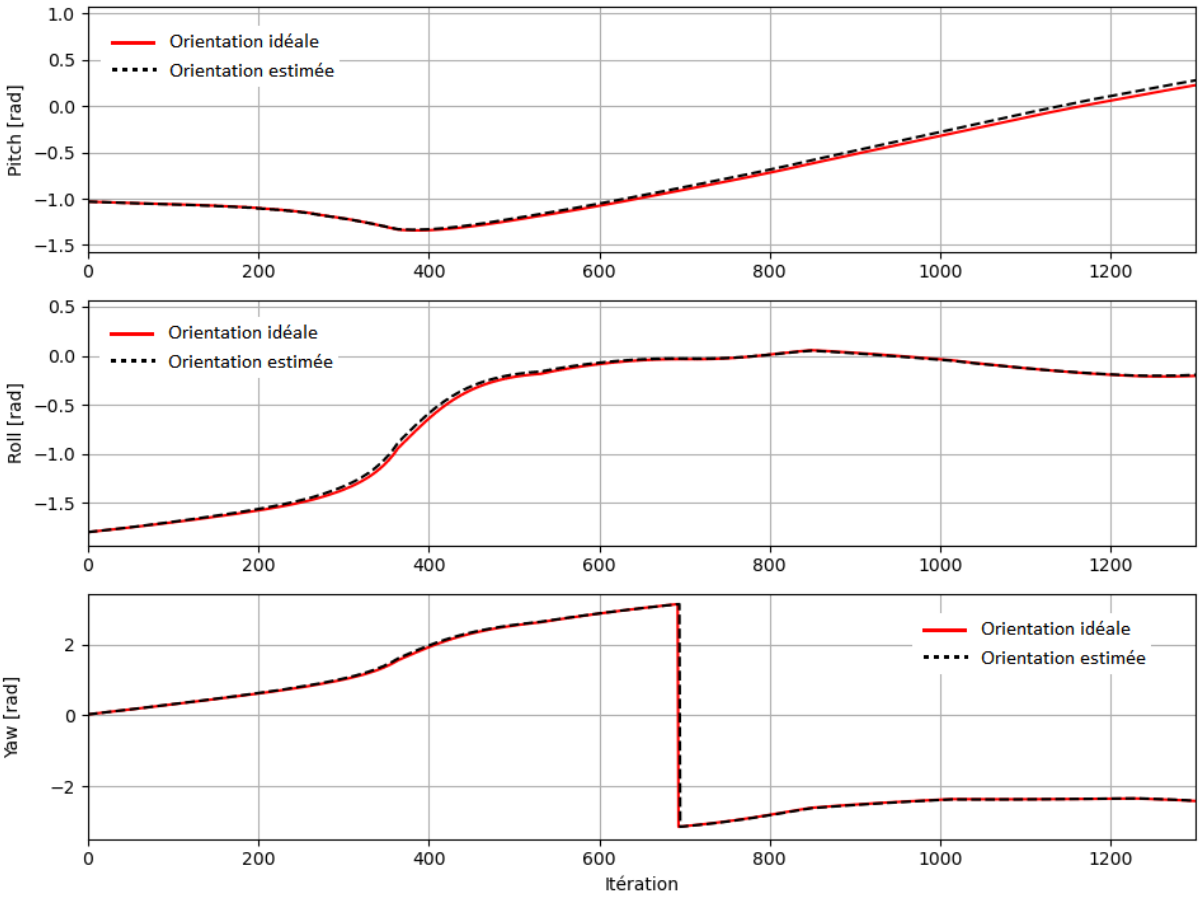


Figure 33 : Orientations estimées par mécanisation seulement

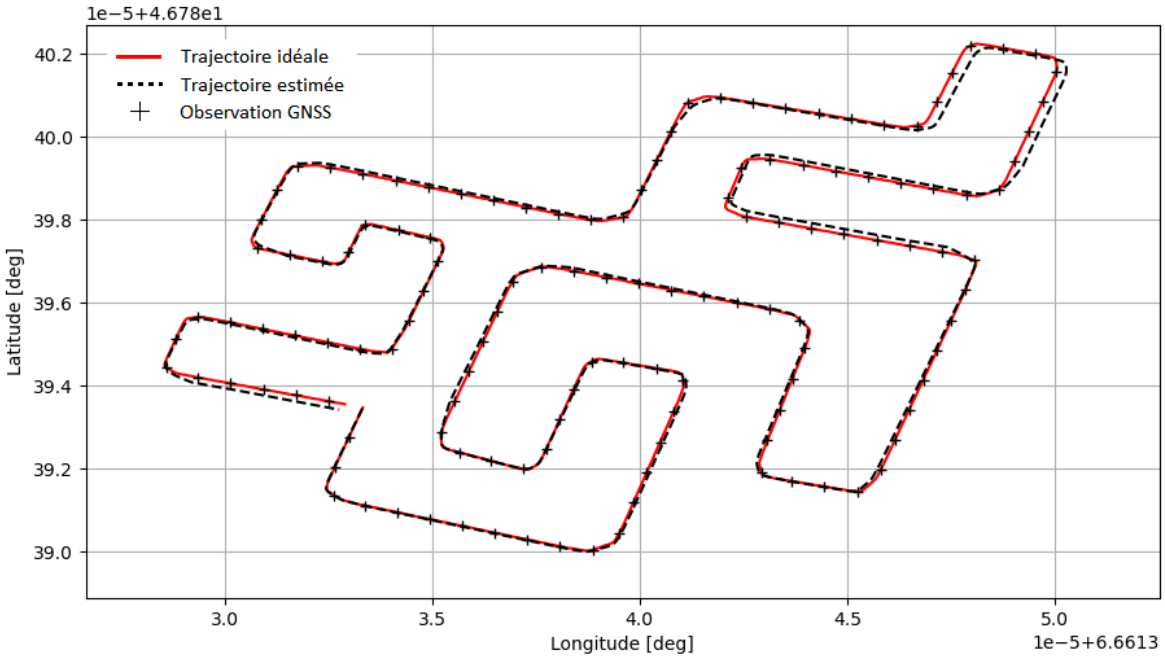


Figure 34 : Trajectoire estimée à l'aide du filtre de Kalman

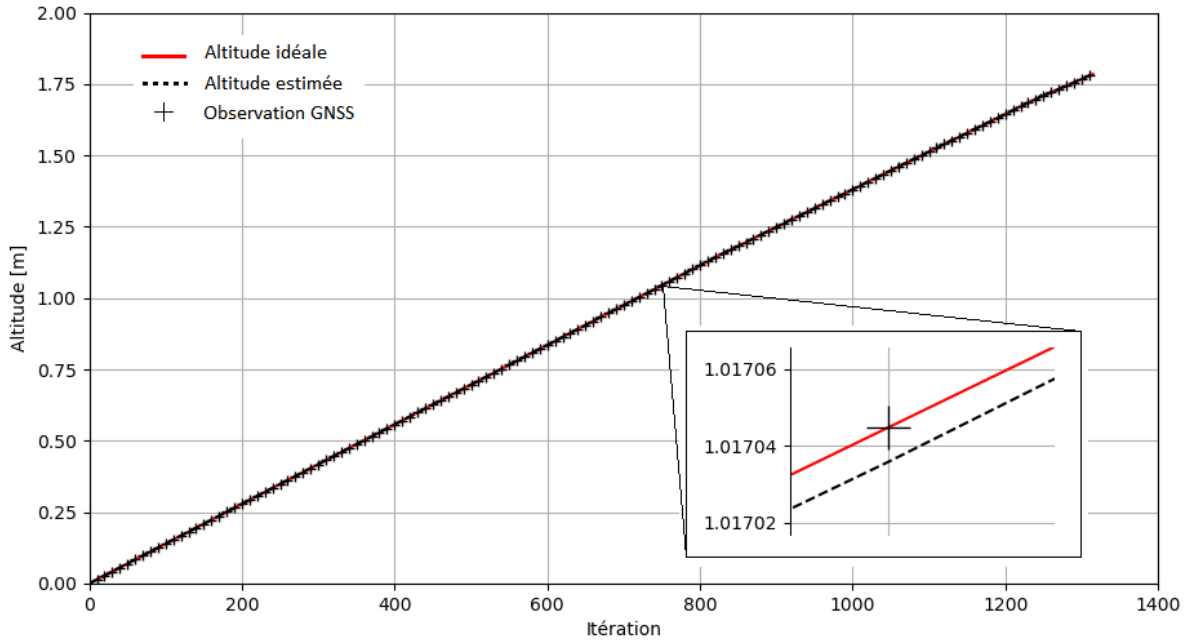


Figure 35 : Altitude estimée à l'aide du filtre de Kalman

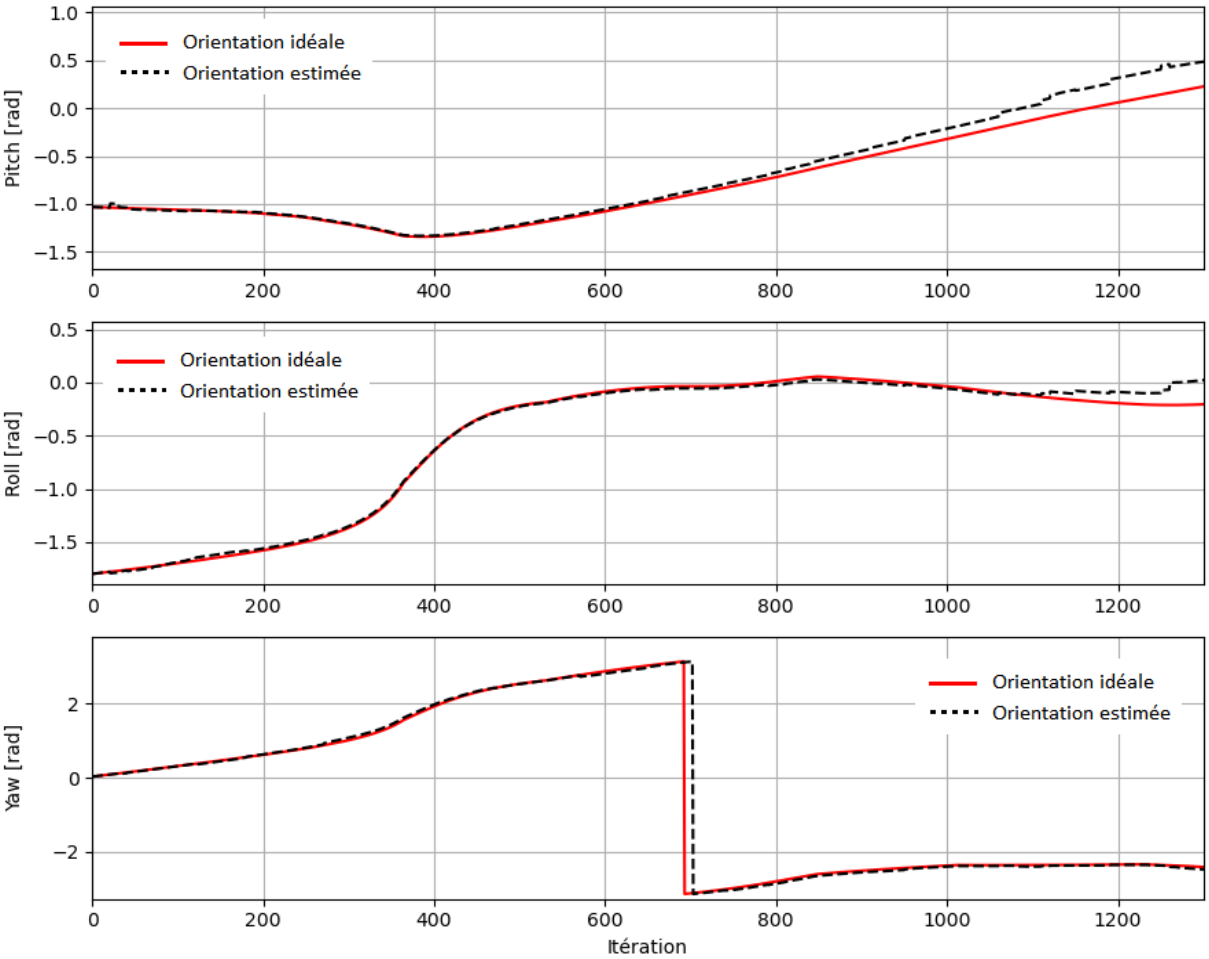


Figure 36 : Orientations estimées à l'aide du filtre de Kalman

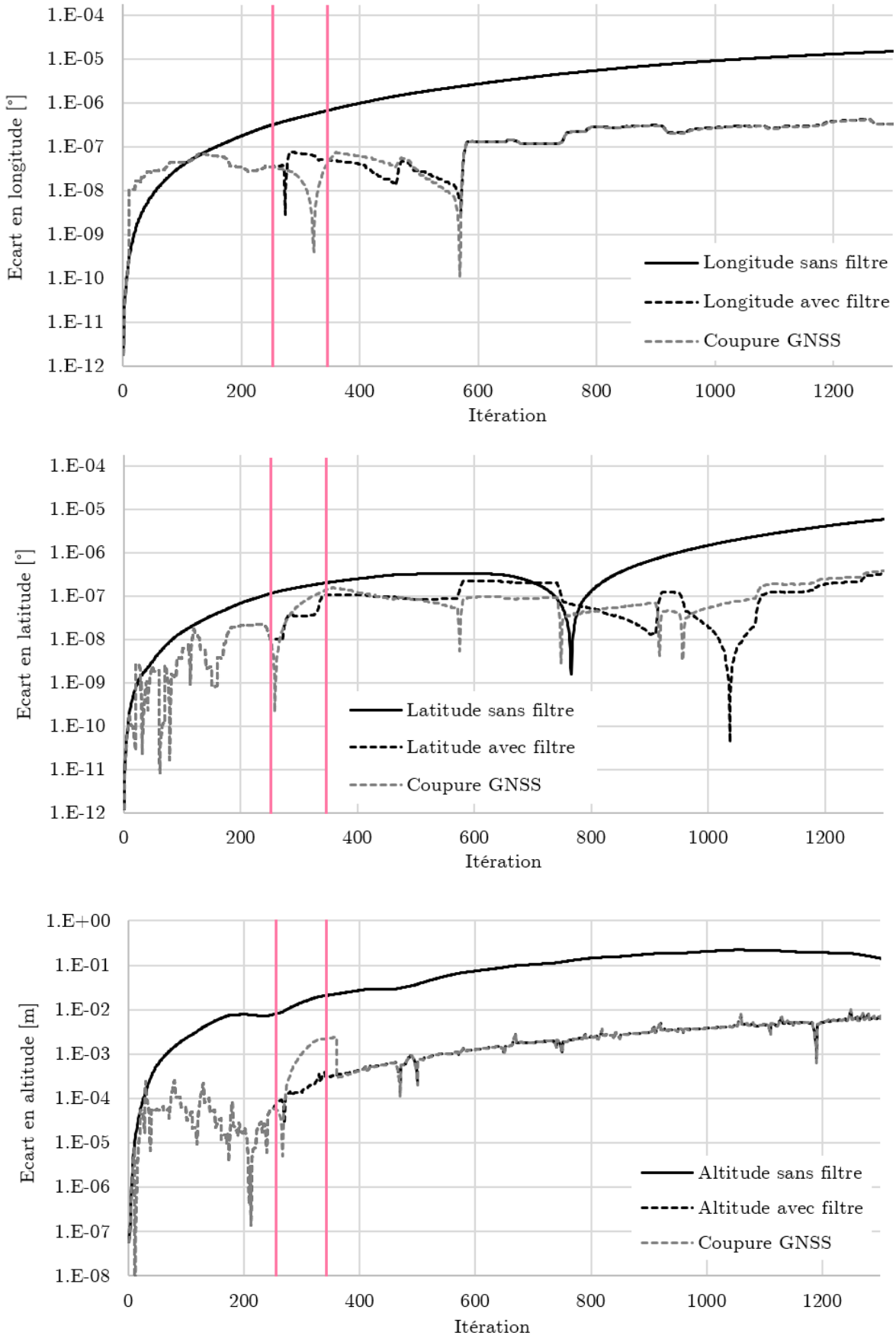


Figure 37 : Ecart en position et altitude avec et sans filtre par rapport à la trajectoire idéale. En rose la période de coupure GNSS.

Sur la Figure 37, le gain en termes d'estimation de la position apparait clairement. En effet, après une phase de mise en place du filtre durant laquelle l'algorithme converge vers les valeurs de biais et affine le degré d'incertitude de chaque variable d'état, l'estimateur se stabilise et réduit l'écart avec la trajectoire idéale d'environ un ordre de grandeur par rapport à l'estimation sans filtre.

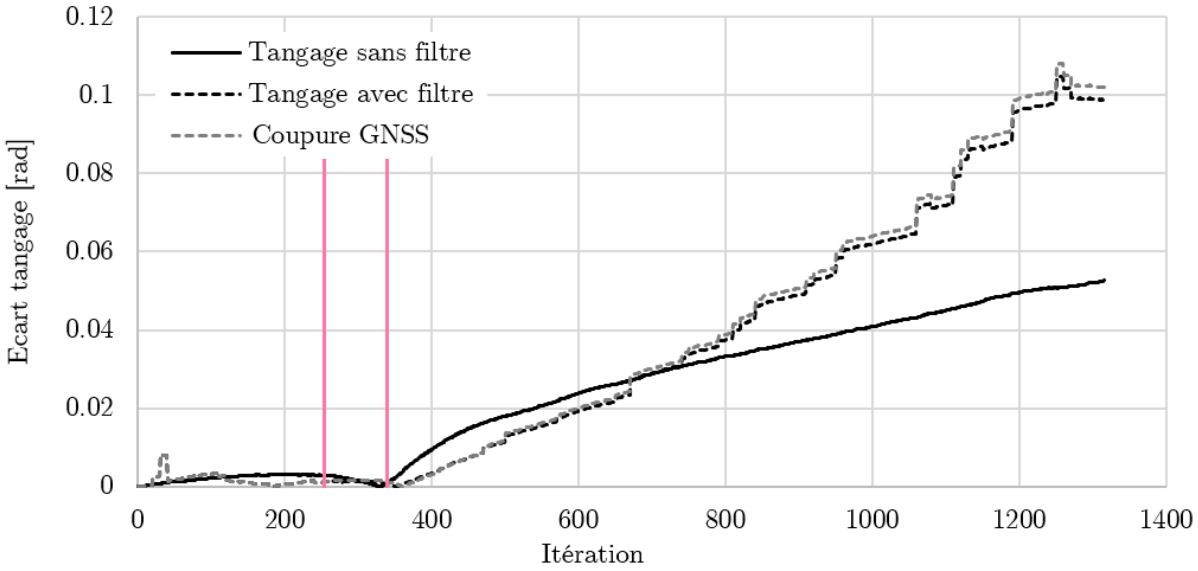


Figure 38 : Ecart sur le tangage avec et sans filtre par rapport à la trajectoire idéale. En rose la période de coupure GNSS.

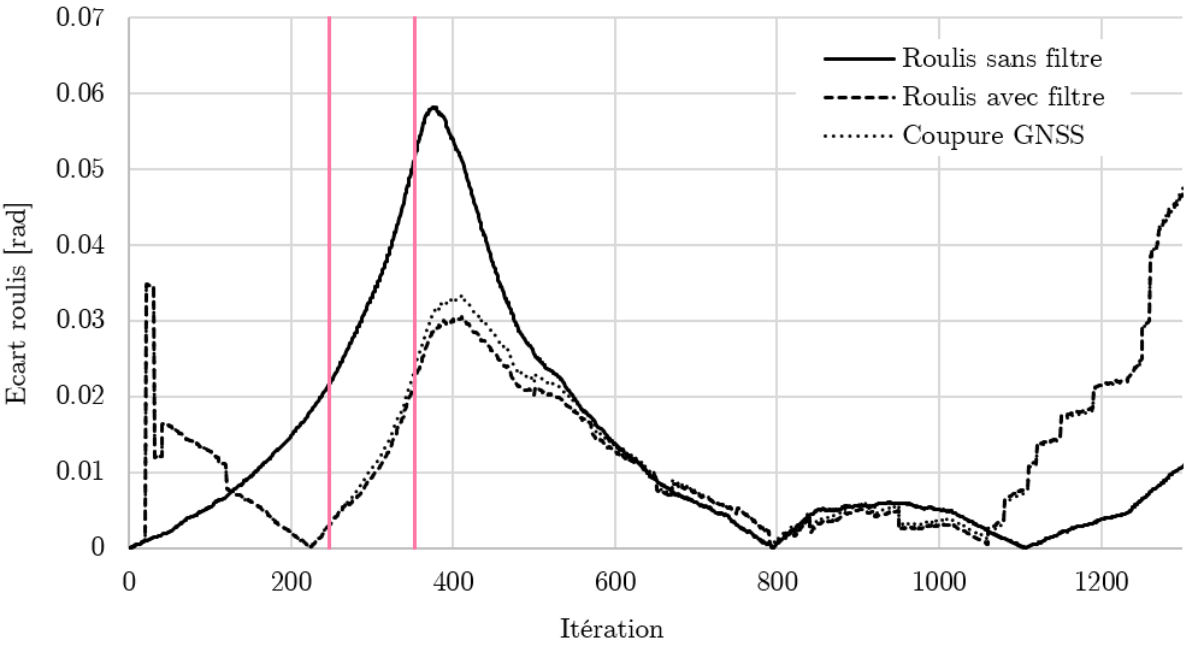


Figure 39 : Ecart sur le roulis avec et sans filtre par rapport à la trajectoire idéale. En rose la période de coupure GNSS.

L'apport du filtre sur l'estimation de l'orientation est plus mitigé. La répercussion des observations GNSS sur l'estimation des angles dépend fortement de différents paramètres du système et notamment les facteurs β_w et β_f du modèle de Gauss-Markov. Normalement obtenus par calibration, une étude de sensibilité sur ces 6 paramètres n'a pas permis

d'obtenir des résultats plus probants que ceux présentés ici. Cependant, au vu de la haute sensibilité du système à ces valeurs il est probable qu'une meilleure solution existe.

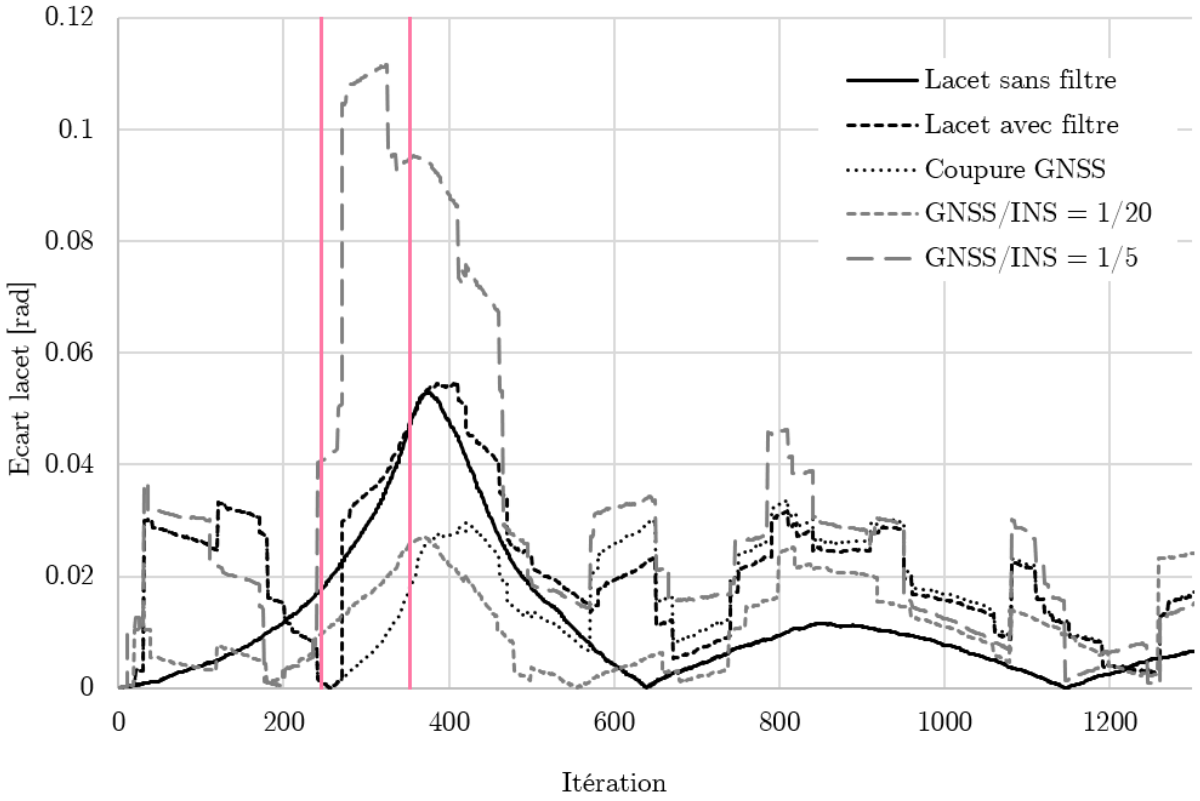


Figure 40 : Ecart sur le lacet avec et sans filtre par rapport à la trajectoire idéale

En l'état, l'utilisation du filtre pour intégrer des observations GNSS ne permet pas d'améliorer l'estimation de l'orientation. Cet effet est particulièrement visible sur l'estimation du lacet (Figure 40) où la situation se dégrade lorsque la fréquence des observations augmente et au contraire devient meilleur en cas de coupure du GNSS ou de baisse de la fréquence. Bien qu'existant, l'impact est moins marqué sur les angles de tangage (Figure 38) et de roulis (Figure 39).

5.3.2 Trajectoire terrain

Contrairement à la trajectoire simulée pour laquelle une référence est disponible, ici la comparaison a principalement été faite par rapport aux données issues de la Xsens. Ces données ne sont bien entendu pas absolues et comportent donc des erreurs. La Figure 41 présente la trajectoire terrain telle qu'estimée par la plateforme Xsens ainsi que les observations GNSS collectées sur le parcours.

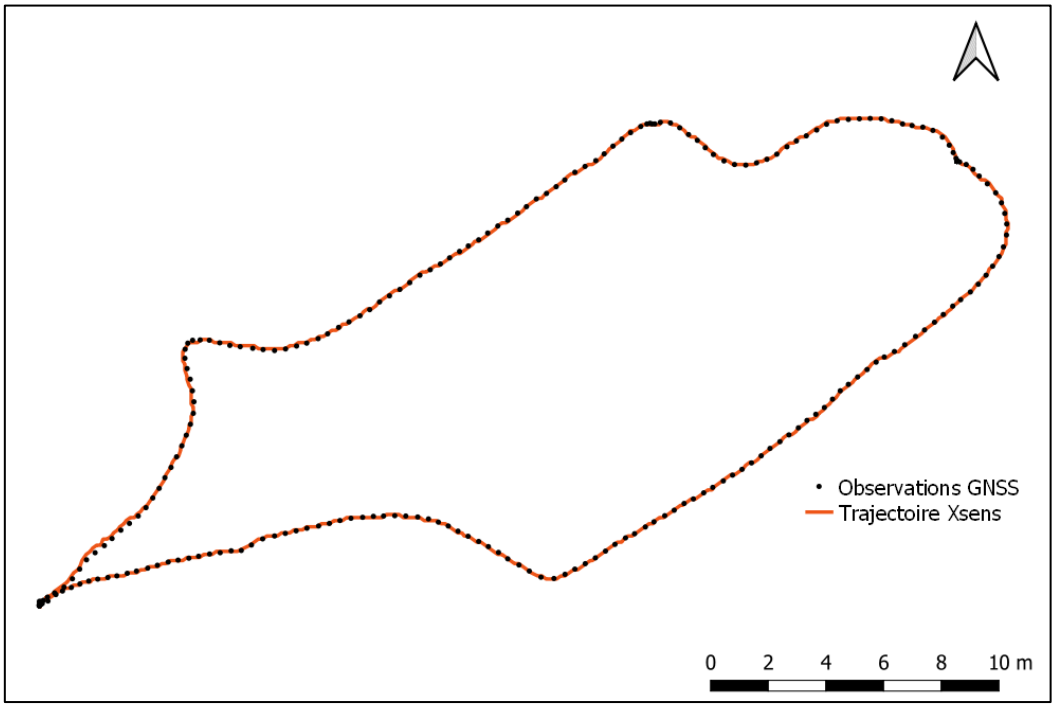


Figure 41 : Observation GNSS et trajectoire telle que calculée par la plateforme Xsens

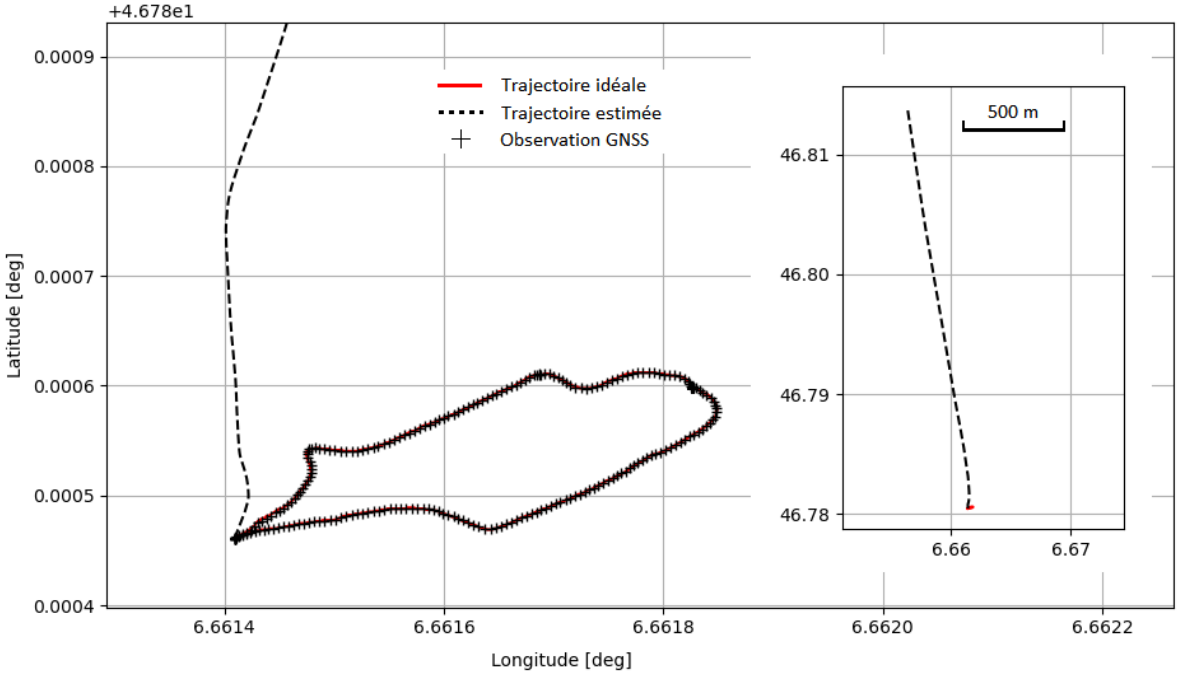


Figure 42 : Position estimée par mécanisation uniquement

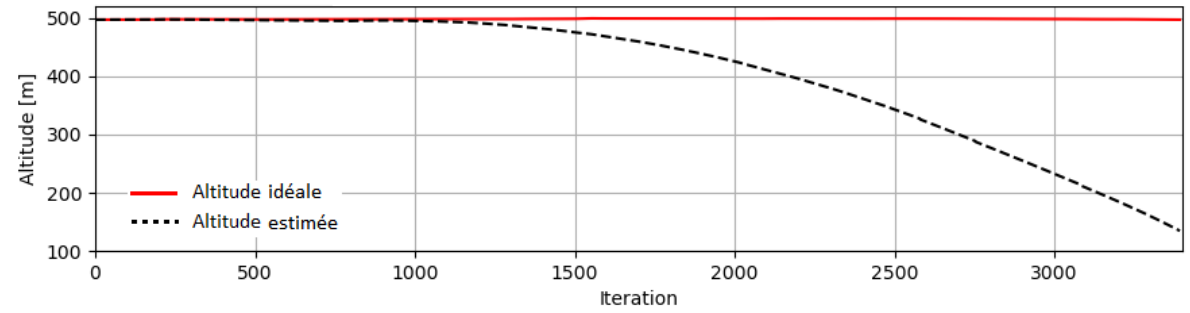


Figure 43 : Altitude estimée par mécanisation seulement

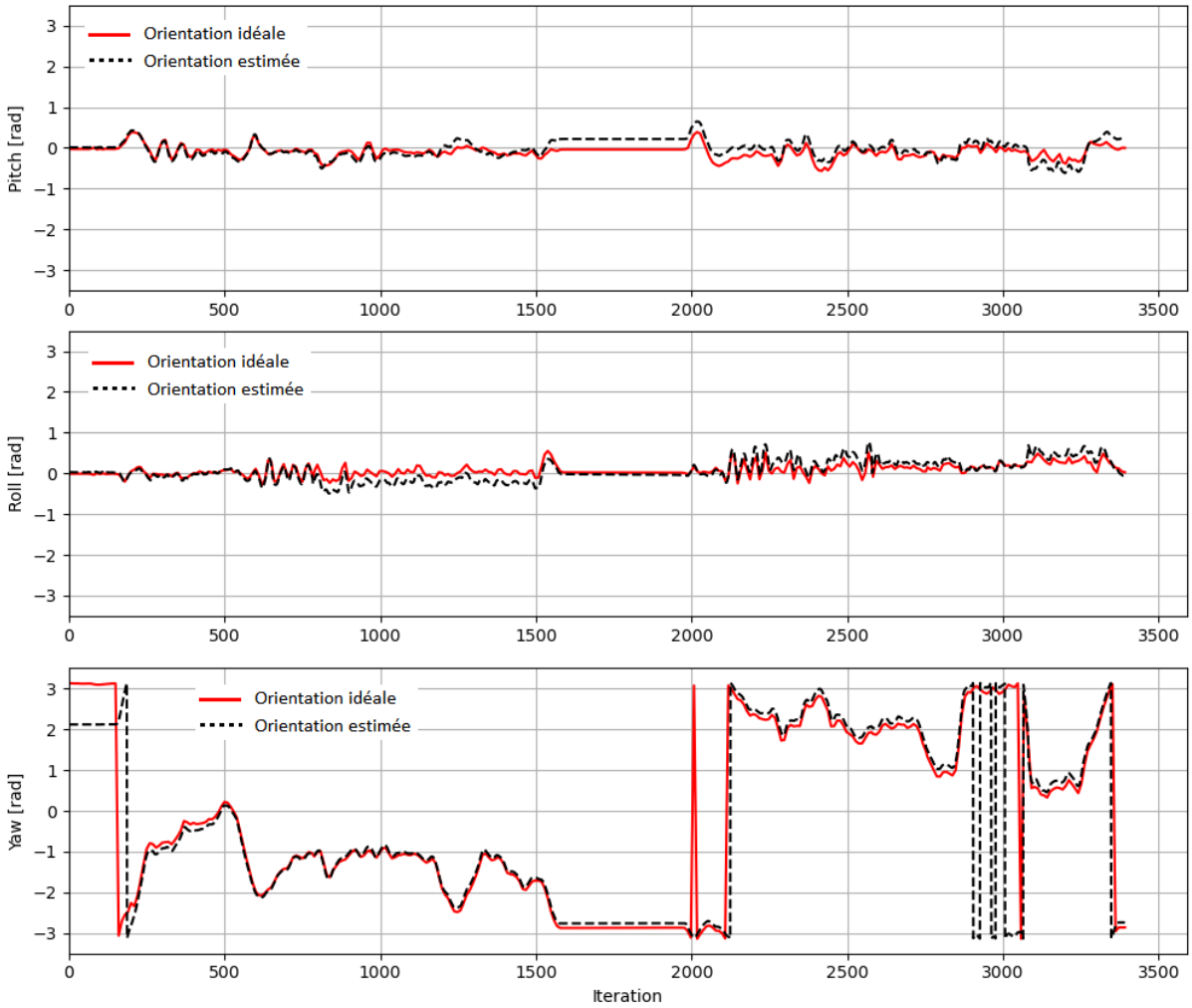


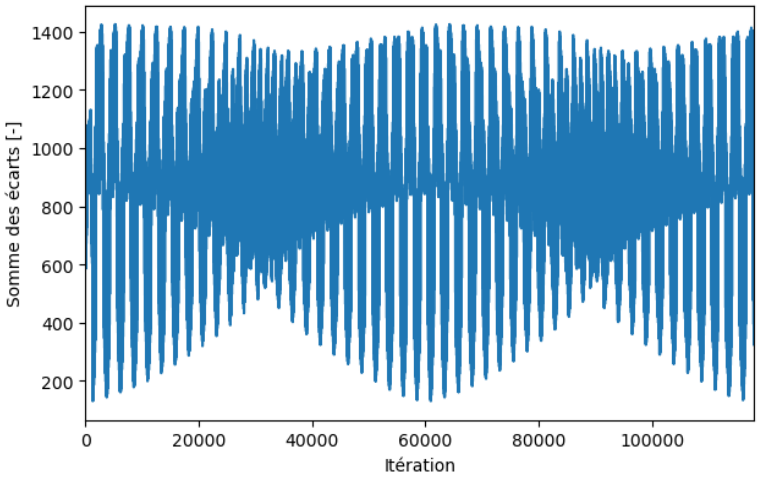
Figure 44 : Orientations estimées à l'aide du filtre de Kalman

L'analyse des données issues de la mécanisation seule (Figure 42, Figure 43, Figure 44) indique que si l'orientation est prédite en cohérence avec la Xsens, les données de position sont entachées d'une erreur importante. De tels écarts sont difficilement explicables. Deux pistes ont été explorée pour essayer d'obtenir des résultats plus concluants :

1. Une hypothèse testée est que l'attitude initiale spécifiée dans le programme n'est pas la même que celle utilisée par la plateforme Xsens. Ainsi les accélérations ne sont pas tournées de la bonne manière, notamment la gravité qui mal corrigée pourrait

provoquer des déplacements conséquents. La forte perte d'altitude plaide également pour un problème lié au vecteur gravité.

Il a donc été recherché une pose initiale « optimale » en faisant varier les trois angles d'orientation sur une plage de valeur pertinente. La trajectoire prédite a, à chaque fois, été comparée à la trajectoire Xsens. Le trio d'angle offrant l'écart minimal ayant été retenu.



Solution optimale :

Roulis : -0.01745 rad

Tangage : -0.03490 rad

Lacet : -0.3316 rad

Figure 45 : Somme des écarts pour chaque combinaison

La principale différence est d'environ 30° sur le lacet. Il aurait été intéressant de déceler des écarts sur les valeurs initiales du roulis et du tangage, mais comme pour les valeurs utilisées, l'optimum correspond environ à une pose horizontale. Cette démarche n'a donc pas été concluante comme le montre d'ailleurs la Figure 46.

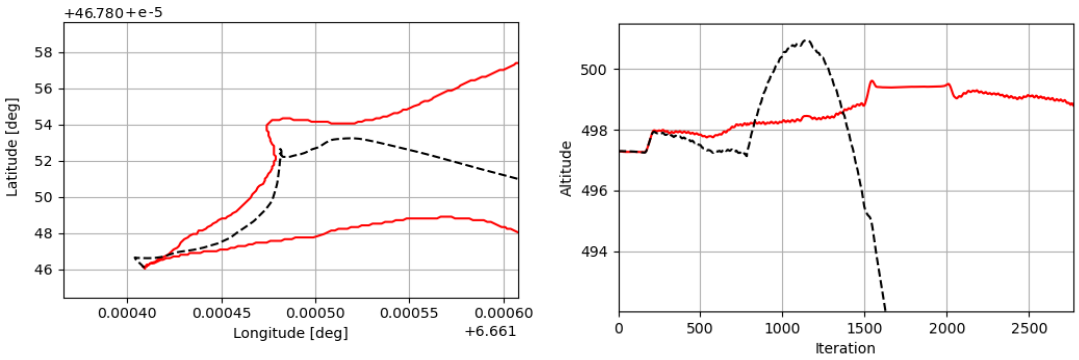


Figure 46 : Première partie des trajectoires estimées avec les angles initiaux « optimaux »

2. Une démarche similaire a été menée sur les facteurs d'échelle sur la sortie des capteurs. Une recherche du facteur optimal a donc été menée en faisant varier, chacun à leur tour, les six facteurs et comparant une nouvelle fois les données estimées et celles de la Xsens. Malheureusement, là aussi, l'analyse n'a pas permis de mettre en évidence un problème d'échelle.

C'est sur cette base que le filtre de Kalman a ensuite été appliqué. Compte tenu des résultats de la mécanisation, une variance élevée a été choisie pour les erreurs sur les accélérations ainsi que sur les vitesses de variation des biais. En effet, les paramètres du modèle de Gauss-

Markov n'ayant pas été mesurés, il est important de laisser une certaine marge de manœuvre au filtre pour converger. Une confiance plus élevée a été choisie pour les données de vitesse angulaire et les observations GNSS.

Les graphiques ci-dessous représentent les écarts en position et en orientation avec les données de la Xsens. Une moyenne glissante sur 20 données a été utilisée pour rendre les figures plus lisibles.

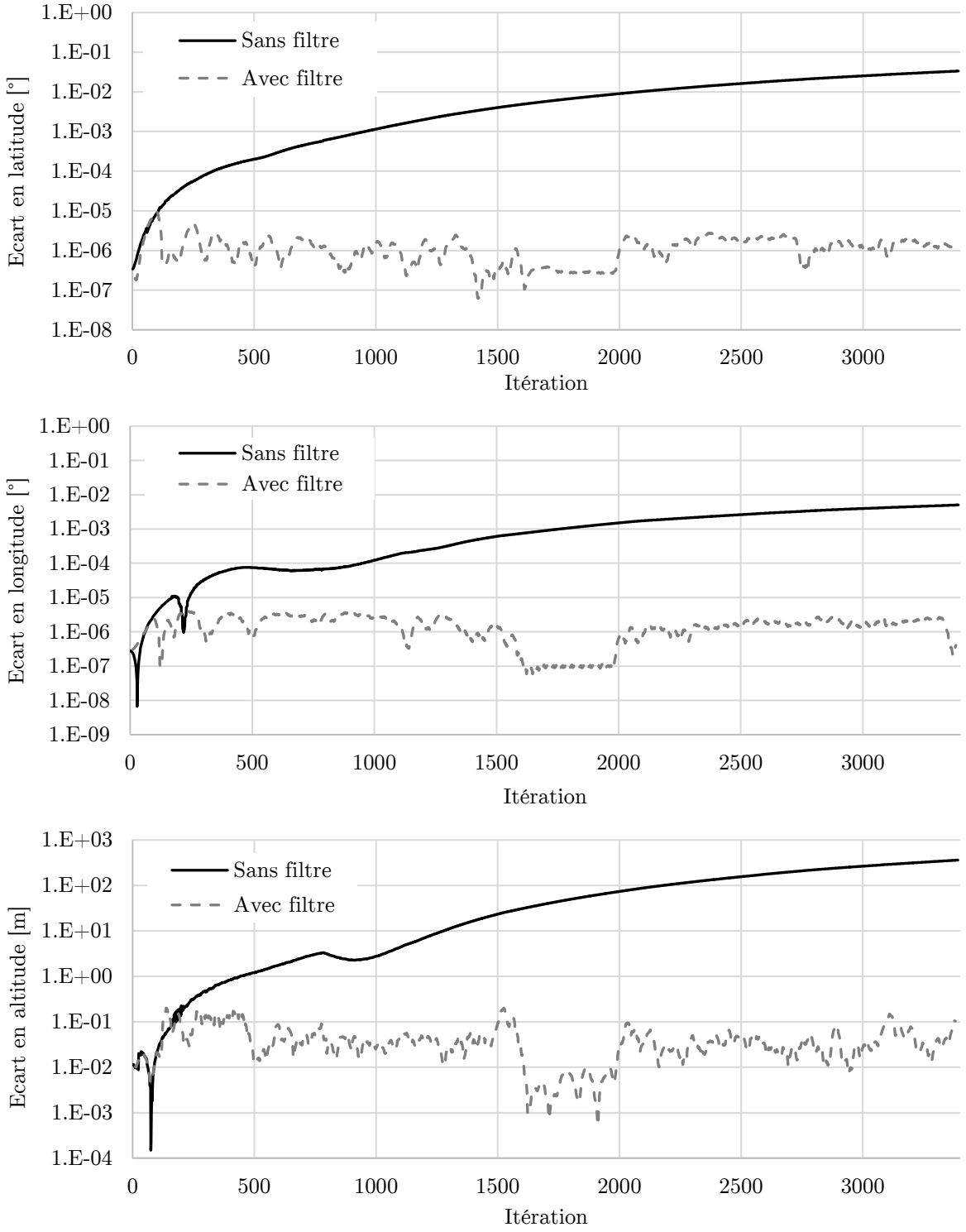


Figure 47 : Ecart en position et en altitude avec la trajectoire Xsens

Il n'est pas très intéressant d'étudier l'apport du filtre sur la position au vu du résultat de la mécanisation. Après une phase d'initialisation où les écarts sont assez importants (Figure 48), ceux-ci se stabilisent autour d'environ 10 cm en planimétrie et entre 1 et 10 cm en altimétrie.

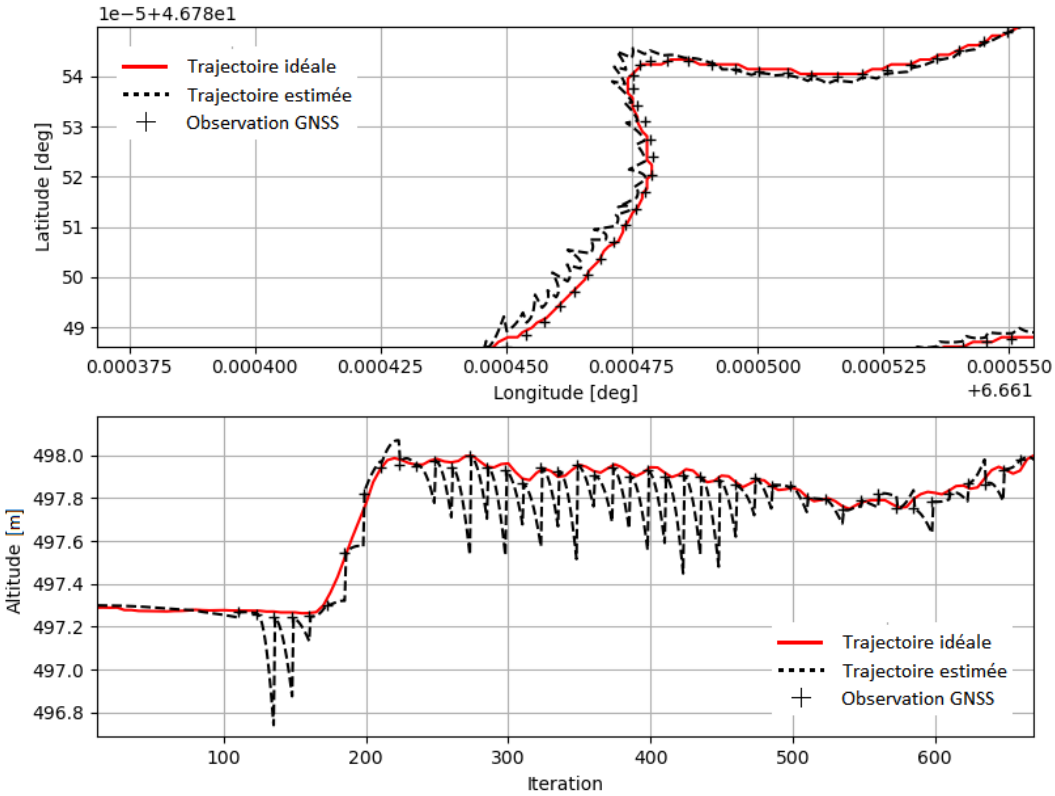
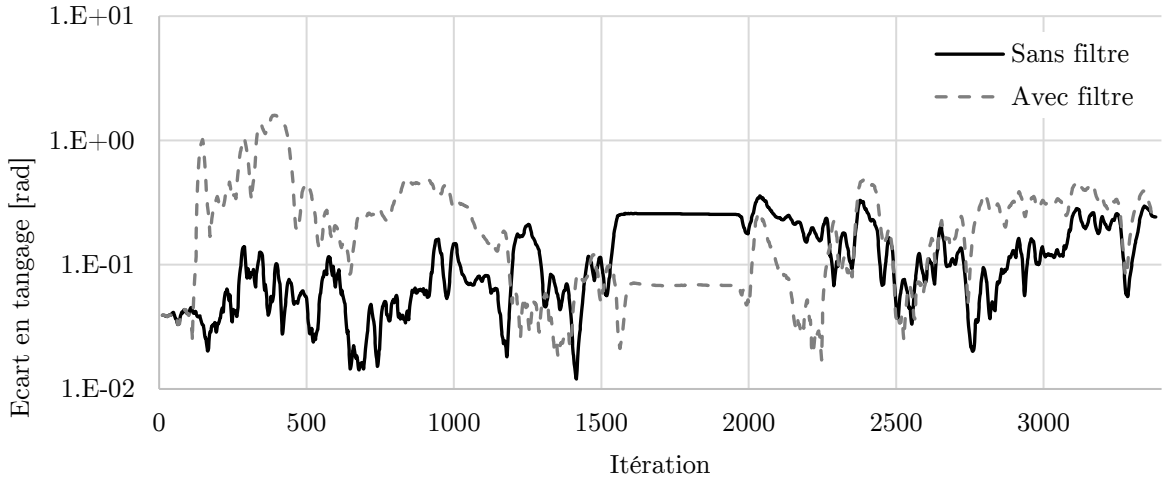


Figure 48 : Phase de convergence du filtre

Comme dans le cas des données simulées, l'apport du filtre sur l'estimation de l'orientation n'est pas flagrant. Après une phase d'initialisation qui dure environ 600 itérations (Figure 51), le filtre converge et fournit une estimation de qualité similaire à celle produite sans filtre. Cette longue phase de convergence du filtre vers des valeurs de biais optimales, est due à la grande incertitude sur la modélisation des biais provoquée par l'ignorance des valeurs de β_w et β_f. Un autre élément est le peu de mouvements dynamiques imposés en début de parcours.



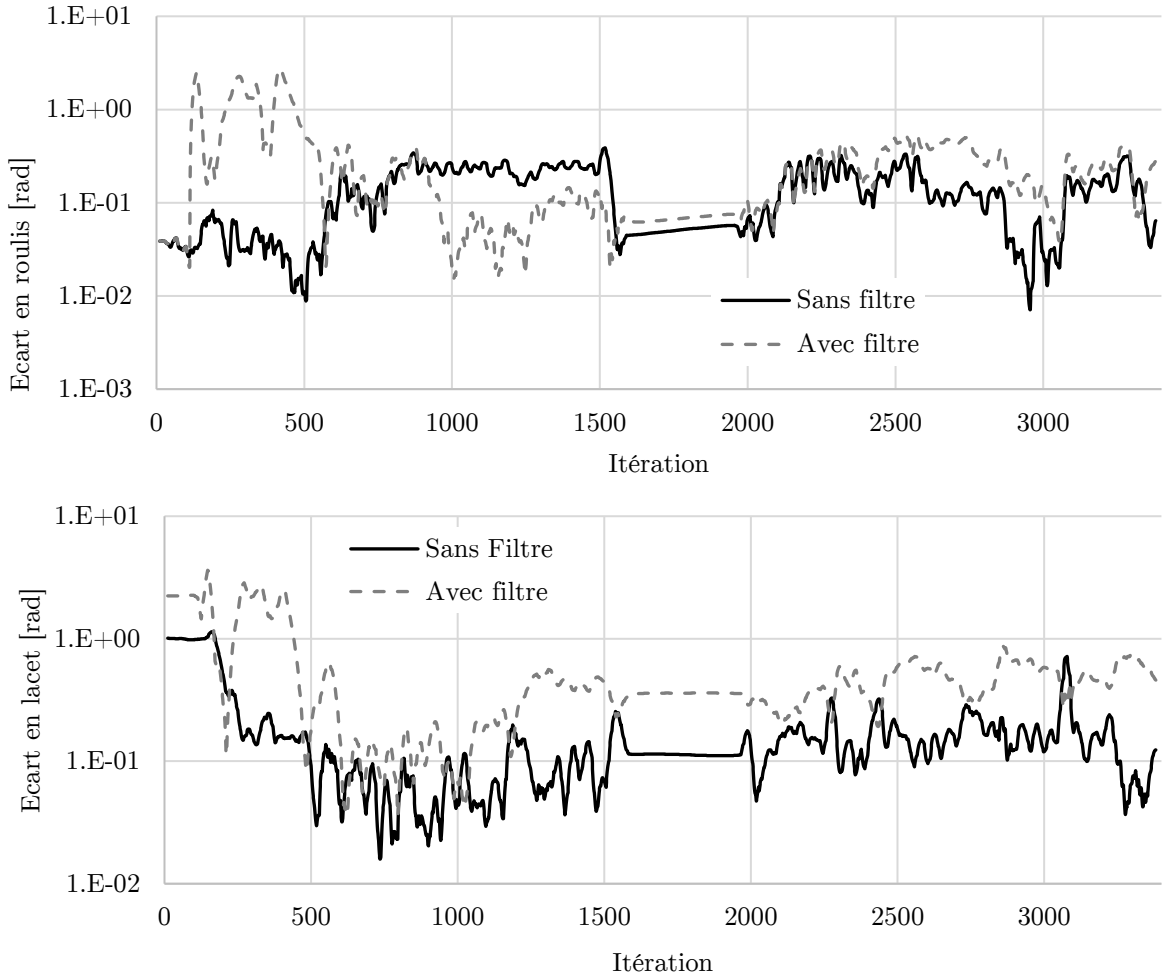


Figure 49 : Ecart en position et en altitude avec la trajectoire Xsens

L'ajustement du filtre sur la prédiction du lacet n'a pas été aisé. Une certaine forme de symétrie entre les corrections appliquées dans chacun des deux cas (Figure 49) interroge. Il est certain qu'une connaissance précise des conditions initiales est nécessaire pour obtenir de bons résultats. Comme illustré sur la Figure 50, selon les conditions initiales choisies, le filtre peut converger vers une situation opposée à la réalité.

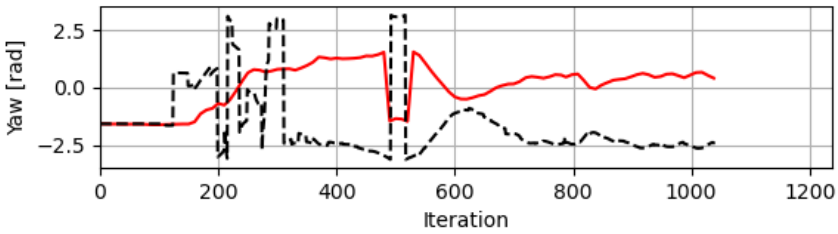


Figure 50 : Symétrie entre prédictions

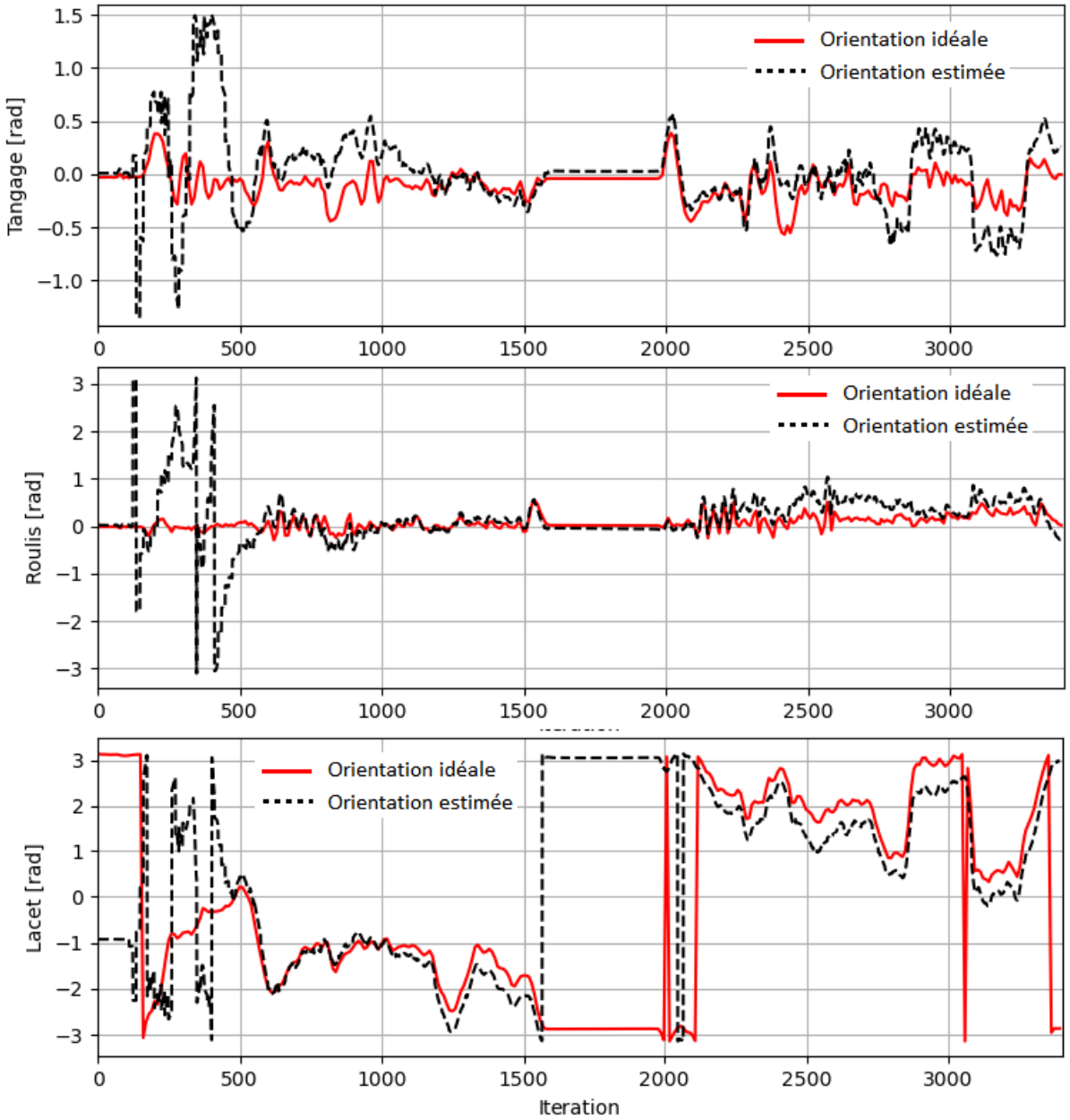


Figure 51 : Estimation de l'attitude

En raison des résultats mitigés de la mécanisation et par conséquent de la nécessité d'avoir des données GNSS fréquente pour aboutir à une estimation correcte, il n'a pas été tenté de simuler une perte de signal GNSS. Dans l'état actuel, la prédiction faite par la centrale Xsens est bien supérieure à celle issue de l'algorithme. Il n'a donc pas été juger pertinent de comparer les deux trajectoires par rapport aux points de passage. A ce stade de développement, il est difficile d'avoir une approche quantitative.

Au vu des nombreux enseignements apportés par l'analyse de ces données, il aurait été intéressant d'acquérir une nouvelle série de mesure, notamment pour évaluer si le problème qui apparait lors de la mécanisation est systématique ou non, et de chercher à mieux connaitre les caractéristiques des capteurs pour réduire le nombre de paramètres à ajuster et, par conséquent, réduire la variance qui s'y rapporte.

Le fait d'utiliser les données issues d'un autre algorithme n'est pas non plus idéal dans la mesure où les traitements appliqués aux données ne sont pas connus. Impossible par exemple de savoir quelles influences ont les magnétomètres sur la correction de la dérive du lacet ou de savoir si une procédure CZRU a été lancée lors des périodes de haltes.

6 Perspectives

Différentes pistes d'amélioration de l'application et du système de navigation sont évoquées dans cette partie.

6.1 Amélioration de l'application

Un point qui nécessite d'être traité est la question du temps de lecture sur le port de la Xsens à l'arrivée des mesures. Le fait que ce temps augmente au cours des mesures peut rendre le système inopérant. Quelques pistes ont été évoquées, notamment l'état de la mémoire au moment du lancement de l'application. Il est toutefois difficile de reproduire cette erreur et par conséquent de la traiter.

En l'état, l'interface utilisateur offre le strict minimum pour piloter l'ensemble. Un point prioritaire serait de pouvoir stopper proprement le programme, par exemple via les topics de configuration afin de laisser à chaque nœud l'occasion de compléter la tâche en cours. De cette manière, il devrait être possible de laisser les packages s'exécuter sur les Raspberry et ne plus avoir besoin de revenir aux terminaux une fois le système démarré. A noter que ROS prévoit un véritable système de gestion du cycle de vie des nœuds (lifecyle node) mais implémenté pour le moment uniquement dans l'API C++. La partie graphique de l'interface doit aussi être améliorée avec au moins la possibilité de suspendre ou arrêter la mise à jour du graphique. La possibilité de pouvoir choisir la donnée affichée serait un plus.

La publication des images pourrait être améliorée. Selon la façon de les traiter par la suite, il pourrait être plus judicieux de publier un ensemble de 4 images prises simultanément plutôt que chaque image dans un ordre qui peut varier.

Finalement, pour améliorer la portabilité de l'application il serait intéressant d'essayer d'intégrer l'installation de « tiscamera » et de « CvBridge » à l'image Docker. Actuellement, ces packages nécessitent une réinstallation après chaque portage.

6.2 Amélioration de l'algorithme de navigation

En l'état, l'algorithme produit est une bonne base de travail et fournit des données de navigation cohérente. Il est toutefois prématuré de vouloir le comparer avec celui embarqué dans la Xsens. En effet, pour pouvoir s'aligner il faudrait inclure les données de pression issues du baromètre afin d'améliorer l'estimation de l'altitude et prendre en compte les magnétomètres pour corriger la dérive du lacet. Pour éviter la dérive du système, plusieurs procédures peuvent être prévues :

- CZRU, pour « Continuous Zero Rotation Update » qui permet d'estimer le biais sur les gyroscopes
- ZUPT, pour « Zero velocity Update » qui permet de réduire l'erreur sur la vitesse. Cette dernière se propageant directement à la position et à l'attitude.
- CUPT, pour « Coordinate Update » qui permet d'estimer l'erreur sur la position en stationnant en des points connus en coordonnées.

La Xsens prend également en compte le bras de levier entre l'origine des mesures et l'antenne pour corriger à la fois la position et la vitesse GNSS. Cette fonctionnalité n'a pas été implémentée.

Pour estimer réellement l'efficacité de l'algorithme développé, il est nécessaire de pouvoir se baser sur une vérité terrain. Le passage par des points connus en coordonnées permet une vérification ponctuelle. Il serait également possible de mesurer au théodolite la trajectoire tridimensionnelle en suivant un prisme fixé sur l'appareil. Finalement, la validation des données d'orientation pourrait se faire avec l'utilisation d'un bras robotisé auquel on impose des rotations connues.

Le parcours prévu pour la collecte des données terrain n'était pas optimal dans le sens où il ne permet pas une convergence rapide du filtre. Il aurait été plus judicieux de commencer avec un mouvement plus dynamique et notamment des rotations selon les différents axes. Xsens recommande des rotations de minimum 30° pendant au moins 10 secondes.

Un élément influençant fortement la qualité de l'estimation et la connaissance du niveau de bruit sur chaque axe des capteurs ainsi que la valeur des facteurs β_w et β_f. L'algorithme étant relativement sensible à ces valeurs, en l'absence d'une qualification propre de chaque capteur, ces propriétés ne peuvent être qu'estimées.

Finalement, l'utilisation de deux repères différents, entre celui dans lequel les mesures sont prises et celui utilisé pour la modélisation du système, nécessite des transformations superflues. Une option serait d'aligner le repère Xsens sur le repère INS pour obtenir des mesures directement exploitables ou de récrire les équations de la mécanisation en inversant l'axe du roulis et l'axe du tangage.

7 Conclusion

Le travail présenté dans ce document avait pour objectif premier de proposer une solution adéquate pour opérer un dispositif alliant des caméras multispectrales, une centrale inertielle et un récepteur GNSS. Sur cette base, un système de navigation a dû être conçu afin de connaitre à haute fréquence la position et l'orientation du dispositif.

L'application développée répond au premier objectif en mettant à disposition une interface utilisateur simple et complète. Elle permet de piloter le système et d'intervenir en cours d'utilisation pour ajuster sa configuration.

L'utilisation de ROS rend aisée l'intégration de nouveaux composants comme, par exemple, une source lidar permettant d'aider à la localisation et à la cartographie de l'environnement considéré. La puissance de ROS se déploie pleinement lorsque l'application s'exécute sur plusieurs machines. C'est déjà le cas maintenant avec une gestion décentralisée des caméras mais il est possible d'imaginer devoir étendre ce principe si l'objectif est de rester dans une application temps réel. En effet, dans l'état actuel, avec deux Raspberry, le système commence à montrer ses limites lorsqu'une certaine fréquence est demandée sur la capture des photos ou sur le calcul de la position. Ajouter des analyses photogrammétriques et le traitement de données lidar semble compliqué.

Remplacer les Raspberry ou employer une autre machine est rendu particulièrement aisé grâce à l'implémentation de l'application à l'intérieur d'un conteneur Docker. Le développement peut être sensiblement plus compliqué en raison de certaines difficultés d'accès aux ressources de l'hôte et à un environnement sans interface graphique mais offre une solution vraiment élégante pour pouvoir migrer le programme facilement.

Le développement d'un système de navigation dédié permet de doter l'application d'un algorithme transparent et complétement ajustable en fonction d'éventuelles évolutions. De conception plus simple, il n'est pour le moment pas question de concurrencer la solution intégrée au module Xsens. Cependant, les résultats fournis par les deux systèmes sont cohérents et montrent que l'algorithme fonctionne. Certains éléments importants peuvent encore être implémentés, tels que la prise en compte du bras de levier entre les mesures INS et GNSS ainsi que certaines procédures permettant de limiter la dérive caractéristique des capteurs inertiels. Une caractérisation plus complète de ces capteurs semble être un point essentiel pour améliorer les performances de l'estimateur.

8 Sources et bibliographie

1. Beniaouf, S. (2021). *Suivi des Vignes par télédétection de proximité : le deep learning au service de l'agriculture de précision* [Travail de Master]. Yverdon-les-Bains : HEIG-VD
2. Bernstein, J. (2003). *An Overview of MEMS Inertial Sensing Technology*. Sensors Weekly.
3. Clausen, P. (2019). *Calibration Aspects of INS Navigation* [Thèse de doctorat]. Zürich: ETH.
4. Delhaye, F. (2018). *HRG by Safran - The game-changing technology*. Safran Electronics & Defense, Boulogne-Billancourt, France.
5. Eck, C. (2001). *Navigation Algorithms with Applications to Unmanned Helicopters* [Thèse de doctorat]. Zürich: ETH.
6. Farrel, J. A. (2008). *Aided Navigation*. McGraw-Hill
7. Goodall, C., Carmichae, S., Scannell, B. (2013). *The Battle Between MEMS and FOGs for Precision Guidance*. Analog Devices.
8. Noureldin, A., Karamat T. B., Georgy J. (2013). *Fundamentals of Inertial Navigation, Satellite-Based Positioning and their Integration*. Berlin : Springer
9. Profanter, S. (2014). *Implementation and Evaluation of multimodal input/output channels for task-based industrial robot programming* [Travail de Master]. Munich: TUM.
10. Quigley, M. et al. (2009). *ROS: an open-source Robot Operating System*. Stanford : Stanford University
11. Rehak, M. (2007). *Integrated Sensor Orientation on Micro Aerial Vehicules* [Thèse de doctorat]. Zürich: ETH.
12. Wägli, A. (2009). *Trajectory Determination and Analysis in Sports by Satellite and Inertial Navigation* [Thèse de doctorat]. Zürich: ETH.

Documentation

13. Xsens (2020). *MT Low Level Communication Protocol Documentation* https://www.xsens.com/hubfs/Downloads/Manuals/MT_Low-Level_Documentation.pdf
14. Xsens (2020). *MTi 600-series Datasheet* https://www.xsens.com/hubfs/Downloads/Leaflets/MTi%20600-series%20Datasheet.pdf
15. Raspberry (2021). *Raspberry Pi Hardware* https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

Répertoires GitHub

16. ETH. *ROS Driver for XSens MT/MTi/MTi-G devices*. (consulté en juin 2022)

https://github.com/ethz-asl/ethzasl_xsens_driver

17. Xsens. *Xsens MTi driver for ROS*. (Consulté en juin 2022)
    https://github.com/nobleo/xsens_mti_driver
18. BlueSpace *AI. Xsens MTi driver for ROS 2.0*. (Consulté en mai 2022)
    https://github.com/bluespace-ai/bluespace_ai_xsens_ros_mti_driver
19. *RobotWebTools. rosbridge_suite*. (Consulté en mai 2022)
    https://github.com/RobotWebTools/rosbridge_suite
20. ROS perception. *vision_opencv*. (Consulté en mai 2022)
    https://github.com/ros-perception/vision_opencv
21. TheImagingSource. *Linux SDK for TheImagingSource cameras*. (Consulté en août 2022)
    https://github.com/TheImagingSource/tiscamera
22. Docker. Docker hub. (Consulté en avril 2022)
    https://hub.docker.com/
23. https://github.com/BWuthrich/Travail-de-Master/settings

9 Annexes

A.1 Codes de configuration de la sortie de la Xsens

Groupe	Code	Type	Code	Fréquence max [Hz]
Temperature	t	Temperature	t	1
Timestamp	i	UTC time	u	2000
		Packet counter	p	
		Sample time fine	f	
		Sample time coarse	c	
Orientation	o	Quaternion	q	400
		Rotation matrix	m	
		Euler angles	e	
Pressure	b	Pressure	p	50
Acceleration	a	Delta v	d	2000
		Acceleration	a	
		Free acceleration	f	
		Acceleration HR	h	
Position	p	Altitude ellipsoid	a	400
		Position ECEF	p	
		Latitude / Longitude	l	
GNSS	n	PVT data	p	4
		Satellites info	s	
		PVT pulse	u	
Angular velocity	w	Rate of turn	r	2000
		Delta q	d	
		Rate of turn HR	h	
Sensor readout	r	Acc, Gyr, Mag – temperature	r	2000
		Gyr temperature	t	
Magnetic	m	Magnetic field	f	100
Velocity	v	Velocity	v	400
Status	s	Status byte	b	2000
		Status word	w	
		Device Id	d	
		Location Id	l	

Code	Description
f	Single precision floating number (32 bit) (default)
d	Double precision floating number (64 bit)
a	Fixed point 12.20 number (32 bit)
b	Fixed point 16.32 number (48 bit)
e	East-North-Up coordinate system (default)
n	North-East-Down coordinate system
w	North-West-Up coordinate system

A.2 Liste des dépendances

curl	libglib2.0-0	pymongo	ros-foxy-rviz2
fim	libgstreamer1.0-0	pyserial	ros-foxy-tf
gedit	libpcap0.8	python3-gi	ros-foxy-tf2-tools
gstreamer1.0-plugins-bad	libudev1	python3-gst-1.0	ros-foxy-turtlesim
gstreamer1.0-plugins-base	libusb-1.0-0	python3-pip	tifffile
gstreamer1.0-plugins-good	libuuid1	python3-pip	tornado
gstreamer1.0-plugins-ugly	libxml2	python3-pyqt5	transforms3d
haversine	libzip5	python3-rpi.gpio	wget
libboost-python-dev	nodejs	python3-venv	tiscamera
libgirepository-1.0-1	nros-foxy-rqt*	ros-foxy-gps-msgs	CvBridge

A.3 Configurations disponibles

Format	Largeur de trame [pixels]	Hauteur de trame [pixels]	Framerate [images/seconde]
GRAY8	640	480	300, 5000000/20833, 120, 60, 30, 15, 5, 1
GRAY8	1216	1024	149, 120, 60, 30, 15, 5, 1
GRAY8	1920	1080	90, 60, 30, 15, 5, 1
GRAY8	2048	2048	45, 30, 15, 5, 1
GRAY8	2448	2048	35, 30, 15, 5, 1
GRAY16	640	480	370, 5000000/20833, 120, 60, 30, 15, 5, 1
GRAY16	1216	1024	149, 120, 60, 30, 15, 5, 1
GRAY16	1920	1080	90, 60, 30, 15, 5, 1
GRAY16	2048	2048	40, 30, 15, 5, 1
GRAY16	2448	2048	35, 30, 15, 5, 1