

UNIVERSITÉ DE RENNES 1

MASTER 2 CALCUL SCIENTIFIQUE ET MODÉLISATION  
RAPPORT DE STAGE

---

**Etude et développement d'outils  
mathématiques pour estimer, en temps  
réel, le tassage et le volume d'un silo de  
maïs à partir de capteurs embarqués**

---

*Auteur :*

Sébastien HERVIEU

*Tuteur de Stage :*  
Geoffroy ETAIX

*Tuteur Universitaire :*  
Fabrice MAHÉ

1<sup>er</sup> septembre 2018



# Remerciements

Je tiens tout d'abord à remercier très sincèrement monsieur Geoffroy ETAIX, qui m'a accordé sa confiance pour mener ce stage, ainsi que toute l'équipe de Tellus Environment dont l'accueil et la gentillesse a été au delà de toutes mes espérances.

J'adresse mes remerciements respectueux à messieurs Fabrice Mahé et Eric Darrigrand, professeurs de l'Université de Rennes 1, qui m'ont permis de rejoindre cette formation et qui m'ont accompagné lors de la recherche de stage et de son déroulement.

Je remercie le Fongecif Bretagne, dont le soutien m'a permis de réaliser ce congé de formation, ainsi que toute l'équipe d'IDAPPS qui m'a aidé dans le montage du dossier de financement : Stéphane RENGER, Yann GIRARDEAU et surtout Laila BENLARIBI, que je salue très chaleureusement. Qu'elle soit ici félicitée de sa redoutable efficacité !

Je remercie l'équipe du service Formation Continue de l'Université Rennes I, mesdames Lisa GENDREAU et Marie-Gisèle DARAS

Je remercie enfin ma petite famille, mon épouse Céline et mes deux filles Clémentine et Angèle, pour leurs encouragements, leur soutien et leur patience tout au long de ces deux dernières années. Nous sommes sur un beau chemin.

# Table des matières

<b>Remerciements</b>	<b>1</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Tellus Environment - Missions Principales . . . . .	5
1.1.1 Géophysique et cartographie haute-définition . . . . .	5
1.1.2 Traitement des données . . . . .	5
1.1.3 Collecte des données géophysiques . . . . .	6
1.2 Activité R&D . . . . .	7
1.3 Projet Symeter V2 . . . . .	7
1.3.1 Symeter V1 . . . . .	7
1.3.2 Symeter V2 : Objectifs . . . . .	9
<b>2 Capteurs, Modélisation, Contraintes</b>	<b>11</b>
2.1 Capteurs . . . . .	11
2.1.1 LIDAR . . . . .	11
2.1.1.1 Détecteur Actif . . . . .	12
2.1.1.2 Scan de l'environnement . . . . .	12
2.1.1.3 LIDAR Hokuyo UTM-30LX-EW . . . . .	12
2.1.2 IMU . . . . .	13
2.1.2.1 Les grandeurs mesurées . . . . .	14
2.1.2.2 IMU XSense MTi 28A53G35 . . . . .	15
2.1.3 GPS en mode RTK . . . . .	15
2.1.3.1 GPS simple . . . . .	15
2.1.3.2 Principe du mode RTK . . . . .	16
2.1.4 Intérêt du couplage IMU/GPS . . . . .	17
2.2 Environnement de programmation : ROS . . . . .	18
2.3 Modélisation . . . . .	18
2.4 Contraintes de développement . . . . .	18
2.4.1 Capacités de tests en grandeur réelle limitées . . . . .	18
2.4.2 Plateformes de test disponibles . . . . .	19
2.4.2.1 Environnement de simulation Gazebo . . . . .	19
2.4.2.2 Prototype monté sur Camionnette . . . . .	19
2.5 Les grandes phases du stage . . . . .	20
2.6 Synthèse des outils . . . . .	21
<b>3 Simulation d'un tracteur évoluant sur un chantier d'ensilage à l'aide de ROS/Gazebo</b>	<b>22</b>
3.1 Présentation de ROS . . . . .	22

3.1.1	Gestion des transformations . . . . .	23
3.1.2	Gestion des capteurs par ROS . . . . .	24
3.2	Présentation de Gazebo . . . . .	24
3.2.1	Construction d'un robot virtuel . . . . .	25
3.2.2	Vérification de disponibilité des capteurs . . . . .	27
3.3	Contraintes de mise en oeuvre . . . . .	28
3.3.1	Pas d'adhérence au démarrage de la simulation . . . . .	28
3.3.2	Simulation mécanique, frottements, adhérence . . . . .	28
3.3.3	Conclusions sur les contraintes . . . . .	28
3.4	Mise en Oeuvre : simulation d'un environnement de tassage de silo . . . . .	29
3.4.1	Montage d'un tracteur simulé . . . . .	29
3.4.1.1	Modélisation du tracteur . . . . .	29
3.4.1.1.1	Modélisation physique . . . . .	29
3.4.1.1.2	Actuateurs et Contrôleurs . . . . .	29
3.4.1.2	Propulsion et Guidage . . . . .	30
3.4.1.2.1	Algorithme . . . . .	31
3.4.1.2.2	Implémentation sous ROS . . . . .	32
<b>4</b>	<b>Mise en place du processus de localisation</b>	<b>34</b>
4.1	Présentation du problème . . . . .	34
4.1.1	Repères . . . . .	35
4.1.1.1	Repère Inertiel . . . . .	35
4.1.1.2	Repère Centré et Fixe par Rapport à la Terre . . . . .	35
4.1.1.3	Repère de navigation local . . . . .	36
4.1.1.4	Repère du véhicule . . . . .	36
4.2	Filtres de Kalman . . . . .	36
4.2.1	Filtres de Kalman Linéaires . . . . .	36
4.2.1.1	Discrétisation . . . . .	37
4.2.1.2	Boucle de Kalman . . . . .	38
4.2.1.3	Fonctionnement conceptuel du Filtre de Kalman . . . . .	39
4.2.2	Estimation de Pose 3D : Filtres de Kalman Etendus . . . . .	39
4.3	Mise en oeuvre sous ROS. . . . .	40
4.3.1	Module robot_localisation . . . . .	40
4.3.2	navsat_transformation_node . . . . .	42
4.3.3	Configuration des Capteurs . . . . .	42
4.3.4	Tests de localisation sous Gazebo . . . . .	42
<b>5</b>	<b>Exploitation des données LIDAR</b>	<b>44</b>
5.1	Présentation de la chaîne de traitement des données LIDAR . . . . .	44
5.2	Acquisition et mise en forme des données LIDAR . . . . .	44
5.2.1	Transformation trame LIDAR en un nuage de points . . . . .	45
5.2.2	Filtrage de la ligne de point par downsampling . . . . .	45
5.3	Accumulation des nuages de points . . . . .	47
5.3.1	Spécifications pour le point_cloud_aggregator . . . . .	47
5.3.2	Principe de stockage des données 3D . . . . .	47
5.3.3	Mise en oeuvre : octomap . . . . .	48
5.4	Mise en oeuvre sous Gazebo . . . . .	48
5.4.1	Modélisation d'un chantier d'ensilage . . . . .	48

5.4.2	Test sous gazebo . . . . .	49
<b>6</b>	<b>Mise en oeuvre à partir de mesures réelles</b>	<b>51</b>
6.1	Protocole de test . . . . .	51
6.1.1	Instruments d'acquisition . . . . .	51
6.1.2	Exploitation . . . . .	51
6.2	Exploitation des données . . . . .	53
<b>7</b>	<b>Perspectives</b>	<b>54</b>
7.1	Amélioration de l'utilisation de l'IMU . . . . .	54
7.2	Developpement d'un module d'accumulation propre à Symeter V2 . . . . .	54
7.3	Simulation de l'action de tassage . . . . .	55
7.4	Développement des outils de détection de monitoring de tassage . . . . .	55
<b>8</b>	<b>Conclusion</b>	<b>56</b>
<b>A</b>	<b>Représentation de l'Attitude et des Rotations</b>	<b>58</b>
A.1	Angles d'Euler . . . . .	58
A.2	Quaternions Unitaires . . . . .	59
	<b>Bibliographie</b>	<b>61</b>

# Chapitre 1

## Introduction

J'effectue depuis le 3 avril 2018 mon stage de fin d'étude de Master 2 CSM au sein de Tellus Environnement, afin de mener à bien la partie étude et développement du projet appelé en interne "Symeter V2". Le but de ce projet est de permettre le monitoring du tassage d'un silo de maïs en temps réel en utilisant un LIDAR embarqué sur un tracteur.

Cette introduction présente les activités de Tellus Environment et introduit le contexte du projet Symeter V2.

### 1.1 Tellus Environment - Missions Principales

#### 1.1.1 Géophysique et cartographie haute-définition

Tellus Environment a été créée en 2012 et s'est initialement spécialisée dans le traitement de données géophysiques, la cartographie haute définition des sous-sols et des fond-marins. Elle propose à ses clients une offre bout en bout d'acquisition et de traitement des données pour permettre à ses clients d'agir en fonction de ces conclusions.

#### 1.1.2 Traitement des données

Tellus Environment dispose d'une expertise particulière du traitement des données géophysiques et optiques en mettant en oeuvre le procédé MagSalia, procédé dont le brevet est exploité exclusivement par l'entreprise.

Initialement développé par le laboratoire de Mathématiques de L'Université de Bretagne Occidentale, ce procédé peut exploiter des mesures magnétiques, sonar multibeam ou LIDAR. Lorsqu'il est appliqué à des acquisitions magnétiques, il produit une tomographie en 3 dimensions de masses magnétiques ponctuelles et étendues jusqu'à 30 mètres de profondeur.

Dans sa version sonar multifaisceaux ou LIDAR, il permet d'accentuer le relief du modèle numérique en mettant en évidence les anomalies du fond marin ou les perturbations de la sous-couche terrestre. Ceci est particulièrement adapté aux problématiques agricoles, archéologiques et aux divers aménagements qui nécessitent des zones vierges (non exploitées) ou blanches (exploitées mais cartographie imprécise ou absente).



FIGURE 1.1 – Géoradar ZOND



FIGURE 1.2 – Sonar (en cours d'utilisation)

### 1.1.3 Collecte des données géophysiques

Tellus Environment met en oeuvre également ses propres équipements géoradar et magnétomètre pour l'acquisition de données sur le sous-sol, sur des surfaces de quelques mètres-carrés à quelque hectares. Elle est aussi en mesure de planifier et coordonner la mise en oeuvre d'équipements plus lourds - LIDAR, géoradar (fig. 1.1 et 1.3), magnétomètres aéroportés - pour obtenir des données sur des surfaces beaucoup plus importantes, de l'ordre du kilomètre carré.

Tellus Environment peut aussi coordonner le déploiement d'équipements d'acquisition marins - sonar (fig. 1.2 et 1.4), multibeam, etc - pour permettre la cartographie des fonds marins et des sous-sols aquatiques, que ce soit en environnement eau douce - rivières, lacs

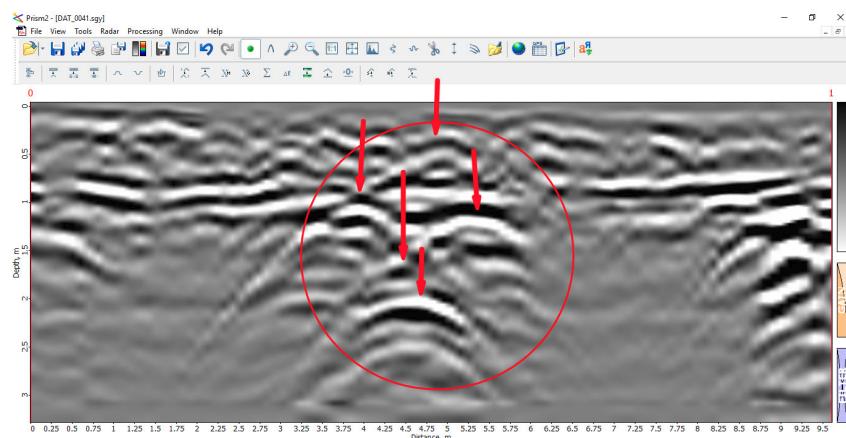


FIGURE 1.3 – Exemple d'un radargramme d'une détection de réseaux enterrés. (Image Tellus Environment)

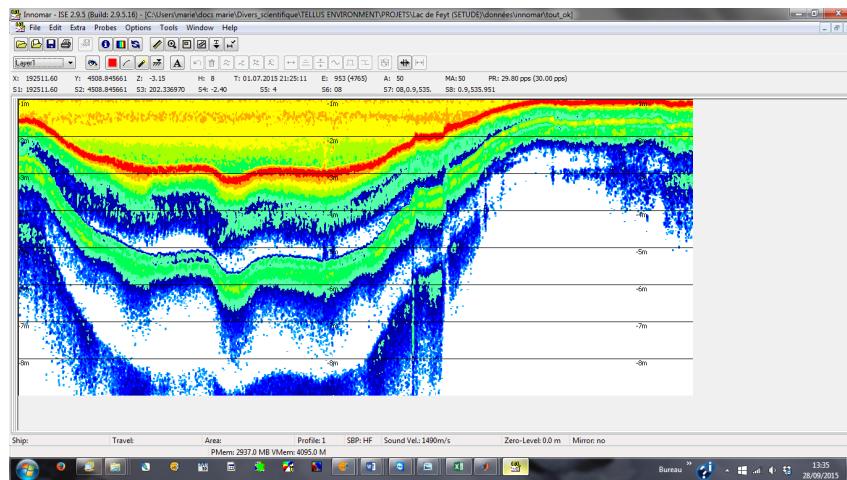


FIGURE 1.4 – Exemple d'un sonargramme généré par un sonar sub-bottom (Image Tellus Envirinment)

étangs - ou marins.

Tellus Environment est spécialiste de la fusion de données (géophysiques et optiques), en ayant accès à de nombreuses bases cartographiques pour extraire l'information utile sur une zone d'intérêt.

## 1.2 Activité R&D

Tellus Environment comprend une activité R&D de cartographie qui met en oeuvre ses expertises en traitement de données, en capteurs liés à la géophysique pour participer au développement de produits innovants.

Le sujet de ce stage se déroule au sein de l'activité R&D de Tellus Environment pour participer au développement du projet Symeter V2.

## 1.3 Projet Symeter V2

Le projet Symeter V2 consiste en la mise en oeuvre de LIDARs montés sur un tracteur de tassage de silo maïs, en vue de mesurer en temps réel la qualité de tassage du maïs. Ce projet prend la suite du Symeter V1 qui est décrit dans la section suivante.

### 1.3.1 Symeter V1

Symeter V1 est un équipement qui permet de mesurer en temps réel l'état de tassage d'un silo de fourrage (maïs, herbe, ...) en vue d'augmenter la qualité des silos générés. Il est basé sur un équipement de mesure laser, le LIDAR, qui permet de collecter très rapidement de nombreux points de mesure.

Ce LIDAR est placé à l'aplomb du silo, au sommet d'une perche de 7m de hauteur, stabilisée par un trépied (voir fig. 1.6). La visualisation des données est transmise au moyen d'une liaison WIFI à un écran embarqué dans le tracteur de tassage (fig. 1.5).

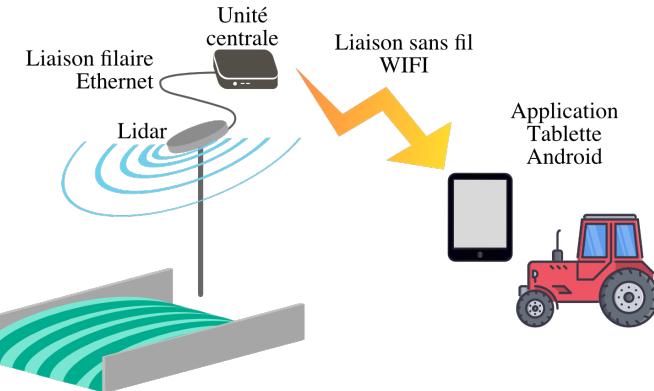


FIGURE 1.5 – Architecture de Symeter V1



FIGURE 1.6 – Le système Symeter V1 en cours d'utilisation

Ce produit a été développé dans le courant de l'année 2017. Son utilisation sur des chantiers d'ensilage réel avait démontré que la mesure de tassage était possible et précise. Mais le retour d'expérience a mis en évidence des points d'amélioration.

Tout d'abord, le déploiement et la configuration de l'installation est plutôt longue et complexe, ce qui n'est adapté pas à la contrainte de mise en oeuvre simple et rapide qui est généralement demandée pour les outils de chantiers d'ensilage.

Ensuite des problèmes de connectivité dans la liaison sans fil entre le module d'acquisition et de traitement d'une part, et la tablette chargée de l'affichage des informations d'autre part, induisent une fiabilité aléatoire du système.

Enfin le positionnement du trépied au plus près du silo en vue d'en obtenir un point de vue le plus à l'aplomb possible (voir fig. 1.6) pose un risque non négligeable de collision entre le tracteur de tassage et le trépied. Cette collision forcerait le re-déploiement du trépied.

### 1.3.2 Symeter V2 : Objectifs

Le projet Symeter V2 a donc été lancé pour améliorer Symeter V1 en prenant en compte des retours d'expérience exposés dans la section 1.3.1. Un choix radical était nécessaire pour permettre à la fois la simplicité de mise en oeuvre, réduire le nombre de composants en interaction et limiter le besoin de liaisons sans fil.

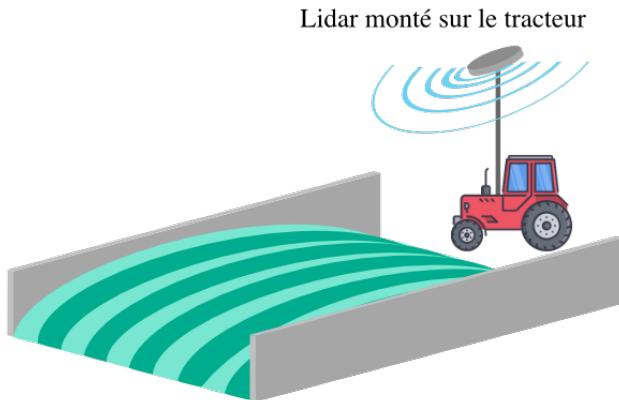


FIGURE 1.7 – Architecture de Symeter V2

Il a donc été décidé de baser Symeter V2 sur le montage de l'équipement directement sur le tracteur de tassage, comme indiqué dans la figure 1.7. Cette configuration a pour avantage qu'elle supprime complètement le besoin d'une liaison sans fil. Tous les composants étant montés sur le tracteur, ils peuvent tous être interconnectés au moyen de connections filaires.

Le fait de monter une bonne fois pour toute l'équipement sur le tracteur permet d'avoir un déploiement sur chantier grandement simplifié, la calibration et la configuration étant effectuée en atelier lors du montage sur le tracteur. La mise en oeuvre de Symeter V2 par un simple appui sur un bouton au début du chantier d'ensilage est donc envisageable.

Bien sûr le fait de monter un LIDAR sur une plateforme entraîne un besoin implicite fort : la *pose* du tracteur doit être connue avec une assez grande précision à chaque instant pour permettre la construction du modèle numérique 3D du chantier d'ensilage. (la *pose* d'un object dans un référentiel donnée est l'union de ses coordonnées Cartésiennes  $x, y, z$  et des coordonnées angulaires de son attitude  $\theta_x, \theta_y, \theta_z$  ).

Ce nouveau besoin force la mise en oeuvre de capteurs et des méthodologies associées permettant de déterminer cette pose. Ces capteurs seront un IMU - une centrale inertielle miniaturisée - et un GPS exploité en mode RTK, appelé aussi "GPS centimétrique".

La mise en oeuvre de ces capteurs pour déterminer la pose du tracteur et permettre la construction du modèle numérique 3D du chantier, à partir de relevés LIDAR issus d'une plateforme mobile représente la grande difficulté de ce projet.

Mais si ces difficultés peuvent être levées, il peut être envisagé que Symeter V2 soit à la fois simple d'emploi, fiable et très utile.

Des outils de simulation robotique seront utilisés pour initier le développement de Symeter V2, pour démarrer rapidement l'implémentation de la chaîne de traitement complète, et potentiellement tester diverses configurations de capteurs avant de monter le prototype en grandeur réelle.

Le reste de ce document décrit d'abord le contexte technique du projet Symeter V2. L'implémentation des processus de localisation, de l'acquisition des données LIDAR et leur utilisation pour générer le modèle numérique 3D du chantier d'ensilage est ensuite discutée. Enfin des tests avec des données acquises en grandeur réelle sont enfin évoqués.

# Chapitre 2

## Capteurs, Modélisation, Contraintes

Ce chapitre présente dans leurs grandes lignes les capteurs cibles du projet Symeter V2, les outils de modélistation qui seront utilisés pour les mettre en oeuvre et les contraintes qui en découlent. Les grandes étapes du projet sont enfin élaborées à la lumière de ces informations.

### 2.1 Capteurs

Les capteurs à mettre en oeuvre seront un ou plusieurs LIDAR, un ou plusieurs IMU et un ou plusieurs GPS, selon un flux de données illustré par la figure ?? . Cette section décrit les fonctionnalités de ces capteurs et les données qu'ils génèrent et comment ils sont mis en oeuvre dans le projet Symeter V2.

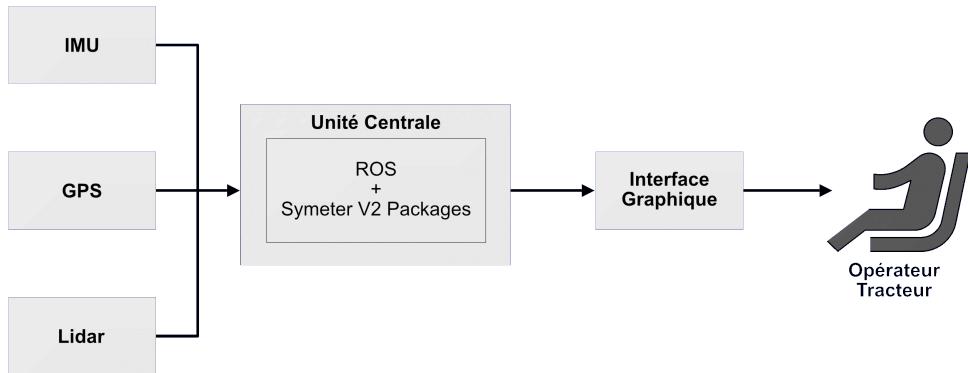


FIGURE 2.1 – Flux de données de système Symeter V2

#### 2.1.1 LIDAR

Un LIDAR est un équipement qui permet de prendre de nombreuses mesures de l'environnement à partir d'un laser. En exploitant les informations d'angle de pointage et en observant les caractéristiques du signal réfléchi tant du point de vu temporel (temps de vol, phase) que du point de vue de la forme du signal (puissance réfléchie, déphasage, distorsion, effet Doppler , ...), il est possible de mesurer de nombreuses caractéristiques de l'environnement.

Le LIDAR est utilisé par Symeter V2 pour détecter la forme précise du chantier d'ensilage.

### 2.1.1.1 DéTECTEUR ACTIF

Le LIDAR est essentiellement un détecteur actif dont le principe est identique à celui du radar : un pulse électromagnétique est émis dans une direction privilégiée par une partie émettrice, constituant un signal incident.

Ce signal incident est réfléchi par un object de l'environnement, générant un signal réfléchi qui retourne en direction du LIDAR. Ce signal réfléchi est reçu par un détecteur.

En comparant le signal émis et le signal reçu nous pouvons en déduire certaines caractéristiques de l'environnement.

**Différents types de mesure :** Il existe deux grands types de LIDAR :

**LIDAR à "temps de vol"** : les LIDARs de ce type déterminent la distance de la cible en mesurant le temps écoulé entre le moment de l'émission du pulse et la réception du signal réfléchi par la cible.

**LIDAR "full waveform"** : les LIDARs de ce type émettent un signal périodique continu et mesurent la distance de l'objet détecté essentiellement par mesure de déphasage du signal réfléchi par rapport au signal émis. Les modifications de la forme du signal réfléchi et de son intensité permettent d'extraire d'autres informations à propos de la cible.

### 2.1.1.2 Scan de l'environnement

En faisant varier la direction du bloc émetteur/récepteur du LIDAR, il est possible de faire plusieurs mesures de l'environnement. Le LIDAR mettant en oeuvre des procédés optiques, il est possible de faire varier très rapidement son pointage et ainsi d'obtenir de très nombreux relevés de l'environnement.

**Modes de scan de l'environnement :** Il existe deux grands modes de scan de l'environnement pour un LIDAR :

**Mode 2D ou "planar"** : Dans ce mode le LIDAR mesure l'environnement dans un plan unique, généralement en faisant tourner le faisceau de manière circulaire autour d'un axe, comme illustré figure 2.2,

**Mode 3D** : Dans ce mode le LIDAR mesure des portions d'espace, généralement en combinant deux rotations simulées du faisceau, soit en faisant tourner sur un même axe de révolution plusieurs faisceaux ayant des inclinaisons différentes (voir fig. 2.3).

### 2.1.1.3 LIDAR Hokuyo UTM-30LX-EW

Le projet Symeter V2 met en oeuvre un LIDAR de type Hokuyo UTM-30-LX-EW (figure), le même que celui utilisé par Symeter V1. C'est un équipement robuste et précis qui est de plus bien connu par Tellus Environment.

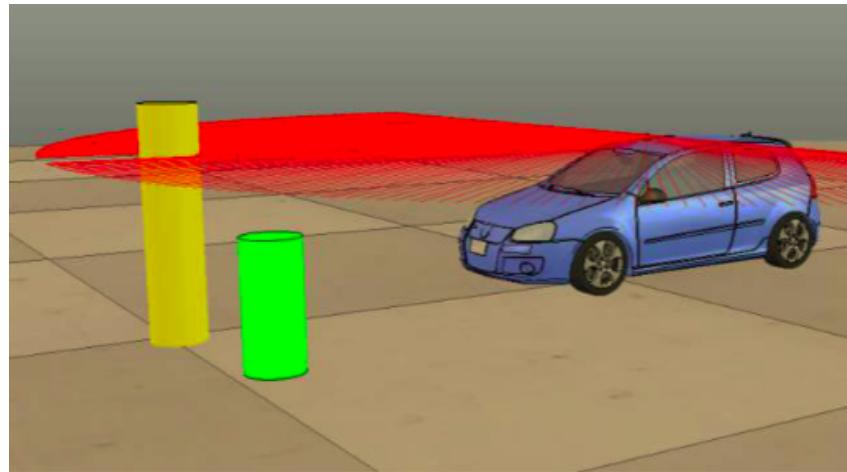


FIGURE 2.2 – Plan de détection d'un lidar 2D ([YSA<sup>+</sup>13])

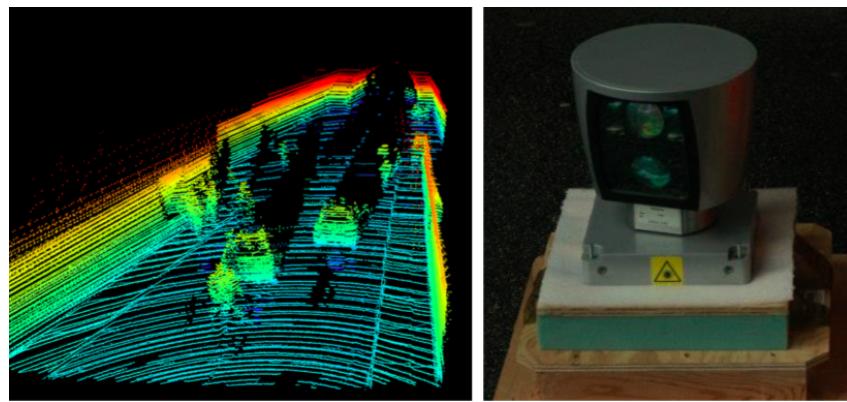


FIGURE 2.3 – Scan laser 3D (gauche) acquis au moyen d'un LIDAR HDL-64E de Velodyne (droite) - Image [Ste14]

Cet équipement comporte un laser infrarouge (longueur d'onde de 905nm) pour scanner un champ semi-circulaire de 270°. C'est un LIDAR de type "planar, temps de vol". Il mesure la distance des objets à sa portée par pas de 0,25°. La distance de détection maximale est de 30m.

Angle de Balayage	270°
Résolution Angulaire	env. 0,25°
Temps de Balayage	25ms/balayage (40 balayages par seconde)
Distance de détection	Portée garantie : 0,1 à 30m

TABLE 2.1 – Caractéristiques principales du Hokuyo UTM-30LX-EW

### 2.1.2 IMU

Un IMU - Inertial Measurement Unit - est un équipement comportant plusieurs capteurs inertIELS, d'accélération et d'angles qui permettent avec un traitement des données approprié d'établir de suivre la pose du véhicule sur lequel il est monté.



FIGURE 2.4 – LIDAR Hokuyo UTM-30LX-EW (roboshop.com)

#### 2.1.2.1 Les grandeurs mesurées

Un IMU comporte généralement 3 accéléromètres linéaires et un gyroscope, et souvent un magnétomètre.

Le **gyroscope** permet de mesurer, avec précision et en continu, l'inclinaison du véhicule en terme de **roulis, tanguage et lacet**.

Les **accéléromètres** mesurent en continu les accélérations linéaires de la partie du véhicule sur lequel l'IMU est fixé, dans trois directions différentes. Ces directions sont généralement dénotées  $x, y, z$ , par analogie avec le repère cartésien. Simplement, les accélérations sont mesurées dans le repère du véhicule, mobile et variable et non dans un repère absolu immobile.

En combinant les variations d'inclinaison et les mesures d'accélérations linéaires dans le repère mobile du véhicule, il est possible de reconstruire avec une certaine certitude la trajectoire du véhicule dans le repère absolu, en utilisant les équations de dynamique Newtonienne (fig 2.5).

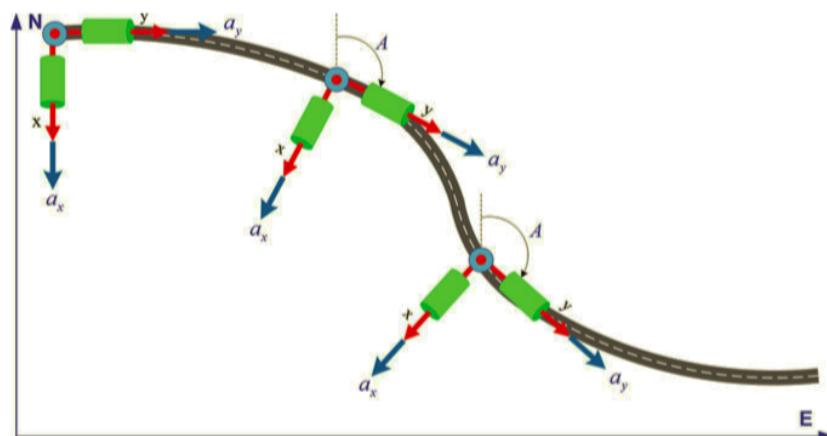


FIGURE 2.5 – Principe de la Navigation Inertielle

Comme indiqué ci-dessus, un IMU possède souvent un **magnétomètre**. Celui-ci permet de mesurer la direction et l'intensité du champ magnétique terrestre par rapport à l'IMU et permet d'en dériver le cap, la direction vers lequel le véhicule pointe sur la surface de la Terre.

Ceci permet, en démarrant l'IMU alors que le véhicule est parfaitement immobile, d'obtenir les conditions initiales du vecteur d'état du véhicule, par rapport au repère odométrique.

Par ailleurs, reconstituer une position à partir d'informations d'accélération implique mathématiquement de procéder à une double intégration numérique. Ce procédé est donc sujet à une dérive dans le temps et doit donc être recalé par une mesure de pose issue d'un autre capteur.

Par contre, un IMU peut fournir des informations avec un très haut taux de rafraîchissement, sans nécessiter d'information extérieure au véhicule. Il peut donc se révéler indispensable lorsqu'aucune autre nouvelle information sur la position du véhicule n'est disponible sur des périodes de temps plus ou moins longues.

### 2.1.2.2 IMU XSense MTi 28A53G35

Le module IMU utilisé par Symeter V2 sera le XSense MTi-28153G35, dont un représentation est montrée figure 2.6. Il peut être branché à une unité de calcul par l'intermédiaire d'une liaison série de type USB.



FIGURE 2.6 – IMU Xsens MTi-28A53G35

### 2.1.3 GPS en mode RTK

Le système Symeter V2 ayant besoin d'une précision centimétrique pour permettre la mesure du tas d'ensilage avec précision du même ordre de grandeur, il mettra en oeuvre d'un GPS en mode RTK. Ce mode fonctionnement particulier du système GPS est en mesure de fournir un position centimétrique du véhicule relativement à une borne GPS (base) située à proximité du chantier.

#### 2.1.3.1 GPS simple

Pour expliquer le principe du GPS en mode RTK, il est nécessaire de comprendre comment le système GPS "simple" fonctionne.

Le système GPS est un système de positionnement par satellite qui, en se basant sur les signaux émis par une constellation de satellites, est en mesure de calculer la position sur la surface de la Terre d'un récepteur avec une précision de l'ordre de la dizaine de mètres, pour une utilisation courante.

Le principe de base est celui de la triangulation : les satellites GPS émettent en permanence un signal comportant une horloge (temps GPS). En connaissant les éphémérides des satellites et en écoutant les signaux de plusieurs de ces satellites, notamment leur signal d'horloge respectif, un récepteur GPS peut calculer avec une grande précision la position de chacun des satellites émetteurs dans le référentiel géocentrique.

Une fois le positionnement des satellites établi, une simple triangulation permet de calculer avec une grande précision du récepteur sur la surface de la Terre (voir figure 2.7).

Pour obtenir une précision suffisante, un récepteur doit écouter au moins 4 satellites en même temps.

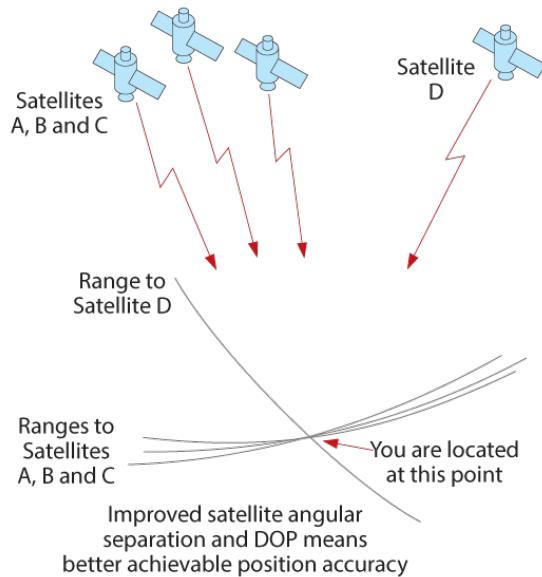


Figure 2.5 Dilution of Precision (improved geometry)

FIGURE 2.7 – Principe de fonctionnement du GPS simple (image [Jef10])

### 2.1.3.2 Principe du mode RTK

Le mode RTK du système GPS est un mode de fonctionnement qui permet de mesurer avec une très grande précision la position relative de deux récepteurs GPS distincts. L'un, dénommé **la base** est un récepteur fixe, dont la position GPS est connue avec une grande précision. L'autre dénommé **le rover** est en communication constante avec la base au moyen d'un liaison radio adaptée (voir figure 2.8)

Les deux récepteurs étant présents sur le même site à la surface de la Terre, ils seront à l'écoute tous les deux des mêmes satellites. En comparant les signaux reçus d'un même satellite par la base et le rover, le rover est en mesure de calculer avec une très grande

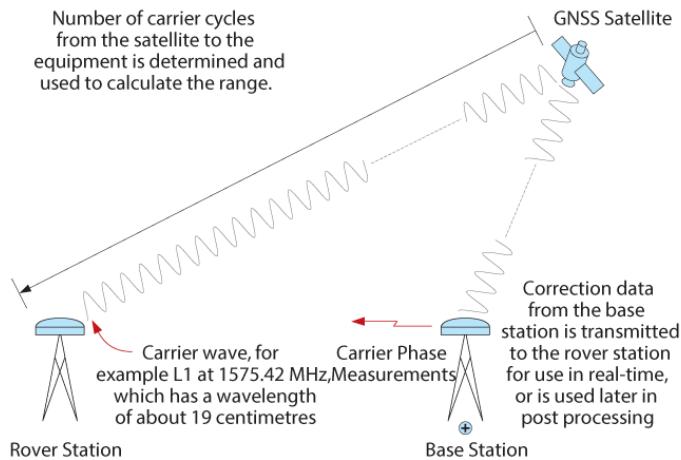


Figure 42 Real-Time Kinematic

FIGURE 2.8 – Principe de fonctionnement GPS RTK (image [Jef10])

précision - de l'ordre du centimètre - la différence de distance entre la base et le satellite d'une part, et le rover et le satellite de l'autre part.

En combinant ces différences de distance à partir de plusieurs satellites, le rover est donc capable de calculer très précisément sa position relativement à la base.

#### 2.1.4 Intérêt du couplage IMU/GPS

L'intérêt du couplage entre le GPS et l'IMU est multiple.

Le GPS en mode RTK est très précis dans le positionnement du véhicule, comme indiqué ci-dessus, de l'ordre de quelques centimètres. Cependant il ne peut mettre à jour cette position qu'à un taux de quelques données par secondes, ce qui est insuffisant pour les besoins du projet Symeter V2. De plus le GPS ne fournit aucune information sur l'attitude du véhicule, alors que cette donnée est indispensable pour Symeter V2. Par ailleurs, il peut arriver que le signal GPS ne soit temporairement pas disponible ou dégradé, soit par indisponibilité temporaire de signal satellite, soit par une perte de connexion entre la base et le rover, généralement par un éloignement excessif. Dans ces cas la fiabilité du GPS se dégraderait considérablement.

De son côté l'IMU fournit bien des informations fiables sur l'attitude du véhicule, peut générer des données à un taux bien plus élevé que le GPS, mais leur exploitation par un module de navigation inertiel (INS) ne permet d'obtenir une position fiable que pour un temps relativement court. Au-delà de ce temps, le module INS doit recalculer son estimation de position avec une source extérieure. Un autre point fort de l'IMU est que c'est un système très robuste dont la disponibilité est généralement excellente.

Le couplage des deux systèmes, GPS et IMU permettrait donc d'obtenir le meilleur de deux équipements, tout en compensant leur faiblesses respectives : le GPS permet de recaler en permanence la position du véhicule issue des données IMU, l'IMU fourni l'attitude du véhicule et peut prendre en charge la navigation à court terme en cas d'indisponibilité ou de mode dégradé du GPS.

## 2.2 Environnement de programmation : ROS

Du fait de son utilisation dans la première version de Symeter, et du fait de ses qualités en terme de modularité, de disponibilité des drivers pour les capteurs et actionneurs, l'environnement logiciel ROS a été choisi avant même le début du stage pour être la base de la partie logicielle de Symeter V2.

## 2.3 Modélisation

Comme indiqué en introduction, le système Symeter V2 sera une plateforme semi-autonome, qui devra être en mesure de fournir une aide à la décision à partir de mesures effectuées depuis une plateforme mobile, et ceux avec une intervention humaine minimale.

Symeter V2 devra donc estimer en temps réel sa pose, afin de pouvoir intégrer une représentation 3D du chantier et effectuer son service.

Les outils de modélisation qui devront être mis en oeuvre sont essentiellement ceux mis en oeuvre dans les systèmes robotiques :

- Les *poses* et les *transformations* : La pose des capteurs et des éléments mécaniques du véhicule les uns par rapport aux autres, et celle du véhicule par rapport à son environnement.
- L'estimation de paramètres à partir d'un ensemble de mesures issues de différents capteurs, au moyen de la *fusion de données*
- Reconstruction d'une scène 3D à partir de *nuages de points*.

Ces outils sont décrits plus en détail dans les chapitres suivants.

## 2.4 Contraintes de développement

Le système Symeter V2 est destiné à évoluer sur un véhicule de type tracteur agricole. Il est de plus destiné à construire une représentation 3D d'un chantier d'ensilage à partir de positions qui vont varier non seulement en *x* et en *y*, mais aussi et surtout en *z*.

### 2.4.1 Capacités de tests en grandeur réelle limitées

Pour développer et tester le système Symeter V2, le développeur ne disposera pas de tracteur en grandeur réelle, ni de chantier d'ensilage, ceux-ci étant des chantiers annuels se déroulant à des moments bien définis dans l'année (printemps et automne).

Par ailleurs, au début du stage il n'est pas encore tout à fait décidé de combien de LIDAR seront nécessaires pour que le système puisse assurer une couverture complète depuis des équipements embarqués sur le tracteur.

Les équipements à intégrer sont relativement couteux et leur mise en oeuvre requiert une certaine expertise. Il est donc nécessaire de pouvoir démarrer le développement du système par le biais de moyens alternatifs.

Ces possibilités sont au nombre de 2 : la simulation d'une part et l'utilisation en grandeur réelle des équipements sur une camionnette pour effectuer des tests simples de reconstitution du terrain.

## 2.4.2 Plateformes de test disponibles

### 2.4.2.1 Environnement de simulation Gazebo

Gazebo est un environnement de simulation très complet qui permet de simuler un environnement physique dans lequel une plateforme semi-robotique simulée peut évoluer.

Cette plateforme simulée peut comporter un grand nombre de capteurs et actuateurs virtuels et ainsi offrir une grande fidélité dans la mise en oeuvre logicielle de la plateforme robotique.

Gazebo est un logiciel intégré à ROS. Sa mise en oeuvre requiert de programmer un véhicule virtuel similaire à la plateforme cible du système (dans notre cas un tracteur avec les capteurs listés dans les sections précédentes), puis de déployer le logiciel ROS pour Symeter V2 sur ce véhicule simulé.

Il est aussi possible de spécifier le "monde" dans lequel le tracteur simulé évoluera, en y ajoutant des murs, du relief, voire une butte en guise de tas d'ensilage.

Il est donc envisageable d'utiliser Gazebo pour simuler un chantier d'ensilage pour tester la localisation, l'acquisition du terrain, la mesure d'un modèle de Silo.

### 2.4.2.2 Prototype monté sur Camionnette

Une simulation n'étant jamais parfaite (comme nous le verrons par la suite), il est quand même nécessaire d'effectuer certains tests avec des instruments réels, lors de tests en grandeur réelle.

Pour la mise en oeuvre en grandeur, un prototype de l'équipement a été monté sur la camionnette de Tellus Environnement, la "Tellus Car".

Le prototype comporte :

- Une unité de calcul
- Un GPS en mode RTK
- Un LIDAR de type Hokuyo UTM-30LX-EW
- Un IMU de type XSense

Ce prototype a été utilisé pour effectuer des captures des trois capteurs alors que la Tellus Car évoluait dans le parking adjacent aux locaux de Tellus Environnement.

## 2.5 Les grandes phases du stage

Le passage de Symeter V1, où le LIDAR est fixe par rapport au chantier, à Symeter V2 où le LIDAR est monté sur le tracteur et est donc mobile par rapport au chantier, remet tout en question vis-à-vis du travail effectué sur Symeter V1.

Le fait d'embarquer le système sur un tracteur pose une contrainte très forte sur le projet en forçant l'ajout d'IMU et d'un GPS qui devront être utilisé pour la localisation.

Donc d'un sujet V1 où nous avions une reconstitution du chantier par une accumulation simple de scans LIDAR rectangulaires, nous avons en V2 deux problèmes complexes interdépendants :

- Localisation à partir de fusion de données IMU et GPS
- Reconstitution du chantier à partir de scans aléatoires

Un problème supplémentaire étant l'absence d'un plateforme matérielle de test pour pourvoir tester les solutions de ces deux problèmes.

Comme il était apparent que sur tous ces sujets nous partions de zéros, il a été décidé très tôt dans le projet de monter une plateforme de tracteur simulée sous Gazebo pour effectuer le développement initial de Symeter V2. Cette plateforme simulée nous permettrait dans un premier temps d'effectuer des tests qualitatifs des solutions choisies, pour valider un certain niveau de faisabilité ainsi que faire des essais sur le choix du nombre de capteur et leur disposition.

A plus long terme, cette plateforme simulée nous permettra d'effectuer des tests quantitatifs de Symeter V2 lorsque de l'implémentation de la mesure de tassage sera complétée, et donc de valider le bon fonctionnement du système avant même de faire des tests sur des chantiers d'ensilage en grandeur.

Le déroulement du stage a donc été le suivant :

1. Vérification des capacités de Gazebo à simuler notre plateforme (capteurs, véhicule)
2. Montage de la plateforme Simulée tracteur et capteurs sous ROS / Gazebo
3. Montage du processus de localisation par fusion IMU + GPS
4. Montage de la chaîne de traitement LIDAR pour reconstitution de la représentation 3D du chantier
5. Tests des éléments développés ci-dessus sur des captures issues du prototype monté sur la Tellus Car.

Les résultats de ces différentes parties du stage sont décrits dans les chapitres suivants de ce rapport.

## 2.6 Synthèse des outils

Monter directement le LIDAR sur le tracteur implique que les mesures ne sont pas prises d'un endroit fixe mais d'une plateforme mobile. Reconstituer un modèle numérique 3D d'un chantier d'ensilage requiert de calculer en permanence la pose précise du tracteur grâce à l'ajout de capteur GPS et IMU.

Cet ajout est une contrainte forte du projet qui remet tout en question. Un système embarqué nous pousse une multitude de nouveaux problèmes !

# Chapitre 3

## Simulation d'un tracteur évoluant sur un chantier d'ensilage à l'aide de ROS/Gazebo

Comme indiqué dans le chapitre précédent, il a été très tôt dans le projet décidé d'utiliser l'environnement logiciel ROS pour implémenter le système Symeter V2. Il avait aussi été décidé de monter un véhicule simulé à l'aide de Gazebo afin d'effectuer le développement initial.

Ce chapitre présente l'environnement logiciel ROS et le logiciel de simulation Gazebo. Cette présentation est illustrée par la mise en oeuvre d'un robot virtuel simple, pour lequel un retour d'expérience est effectué en fin de chapitre suite à cette première expérience. L'implémentation du tracteur de test est ensuite présentée en fin de chapitre.

### 3.1 Présentation de ROS

ROS - Robot OS - est un environnement logiciel destiné au pilotage autonome de systèmes robotiques. Ce logiciel Open Source originellement développé conjointement par Willow Garage et l'université de Standford, a été lancé pour sa version 1.0 en janvier 2010 (voir [www.ros.org](http://www.ros.org)).

ROS en est maintenant à sa 11<sup>ème</sup> distribution, dénommée "ROS Lunar Loggerhead", distribution qui sera utilisée pour Symeter V2.



FIGURE 3.1 – Quelques exemples de robots utilisant ROS (image [MF13])

ROS est un environnement logiciel destiné à la mise en oeuvre de plateforme robotique. Cet environnement très modulaire permet de programmer et de déployer de nombreux modules interdépendants, appelés **packages**, constitués eux de noeuds exécutables appelés **nodes**.

Ces nodes peuvent communiquer entre eux aux moyens de deux mécanismes principaux :

- les services
- les topics

Ces deux mécanismes permettent l'échange de données entre nodes selon des messages de la structure peut être spécifiée par le développeur du système.

Les **services** sont mis à disposition chacun par un node. Ils peuvent être invoqués par d'autre node en effectuant une transaction explicite : le node appelant fourni une donnée d'entrée au node mettant à disposition le service. Ce dernier traite la donnée entrante puis génère une donnée de sortie qui est enfin renvoyée au node, terminant ainsi la transaction.

Les **topics** sont des canaux de diffusion d'information similaire à une autoroute : tout node peut diffuser un message vers un topic et tout node peut écouter un topic pour recevoir et traiter les messages qui y sont émis.

Ces topics pourvoient des données que l'on pourrait qualifier de "sensorielle", dont la durée de vie est courte, telle que les données générées par un capteur fournissant des données en continu.

Ils sont aussi le support de diffusion des **transformations**, qui permettent de décrire l'état de **pose** de chacuns des composants mécaniques d'un véhicule autonome, ainsi que la pose du véhicule dans son environnement.

ROS est un environnement fourni sous une licence Open Source et propose des drivers pour de très nombreux capteurs et actionneurs de tous types. Il intègre aussi des composants logiciels tiers spécialisés tels que, entre autre, la Point Cloud Library pour le traitement des nuages de points et la librairie OpenCV pour le traitement de la vision par ordinateur.

### 3.1.1 Gestion des transformations

La robotique à pour enjeu de permettre la mise en oeuvre de systèmes mécaniques autonomes, mobiles ou non, qui mettent en oeuvre une série de capteurs et d'actionneurs pour agir sur leur environnement avec une interaction humaine très limitées.

La capacité de représenter de manière fine la position des différents éléments d'un tel système les uns par rapport aux autres, que ce soit en position  $x, y, z$ , mais aussi en inclinaison -  $\theta_x$  (tanguage),  $\theta_y$  (roulis),  $\theta_z$  (lacet) - est donc primordiale. L'état de l'élément décrit par ce vecteur  $x, y, z, \theta_x, \theta_y, \theta_z$  est appelé sa **pose**.

Les outils mathématiques de prédilection pour décrire une pose sont bien sur les transformations dans  $\mathbb{R}^3$ , translations et rotations. Les rotations sont gérées nativement par ROS selon 2 méthodes conventionnelles, les *angles d'Euler* et les *quaternions*. Une introduction de ces deux concepts est disponible en annexe A.

Ces poses sont utilisées de manière native par ROS pour deux choses principales. D'une part l'agencement mécanique précis des éléments constituant le robot, notamment ses actuateurs (roues, bras, etc) et ses capteurs, sachant que ces positions relatives sont susceptibles d'évoluer dans le temps. D'autre part les positions des éléments extérieurs aux robot sont décrits selon ce même cadre.

ROS fourni donc un topic spécifique, le topics \tf, dédié à la diffusion constante des transformations des éléments du robot et de la position des objets externes, selon une arborescence structurée. Un exemple de cet arbre est montré ci-dessous :

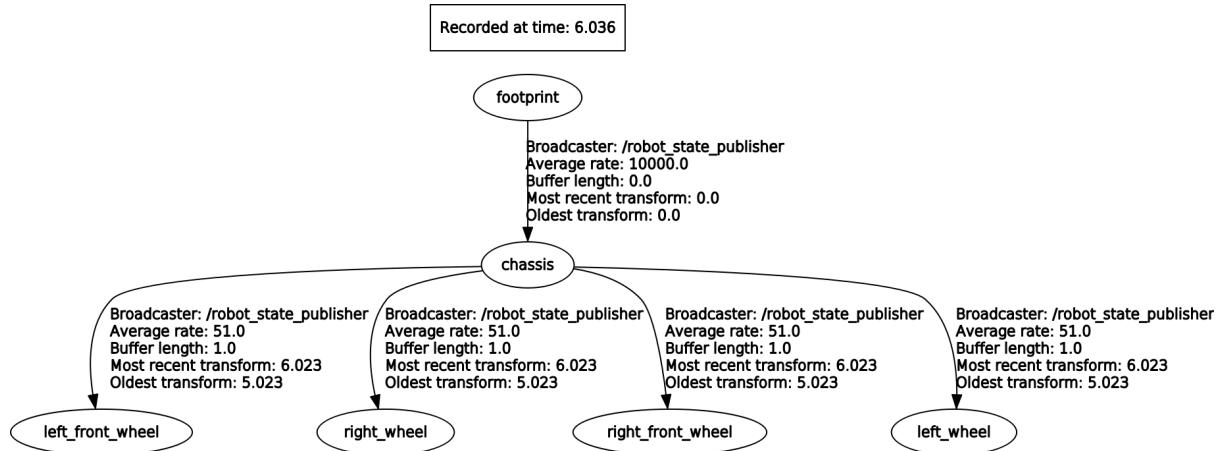


FIGURE 3.2 – Arborescence de transformations du robot mybot

### 3.1.2 Gestion des capteurs par ROS

Les capteurs sont gérés par ROS en permettant aux constructeurs de capteurs de générer simplement les drivers qui injecteront les données mesurées dans ROS. Cette injection est généralement effectuée au moyen de topics dédiés qui permettent de diffuser des messages dont le format est adapté aux grandeurs mesurées par les capteurs (scalaires, angles, distance, pose, laser scan, nuage de point, etc).

Par ailleur chaque message diffusant des données mesurées doit inclure l'identifiant de la transformation du référentiel dans lequel la mesure a été prise. Cela permet de grandement simplifier le traitement des données, en permettant de projeter par exemple une mesure prise dans le référentiel d'un capteur vers un référentiel plus global, en remontant l'arborescence de transformations diffusée par le topic \tf.

## 3.2 Présentation de Gazebo

Gazebo est un logiciel qui permet de simuler un "monde" virtuel 3D, dans lequel il est possible de plonger des véhicules robotiques qui seront alors soumis aux lois physiques

simulées.

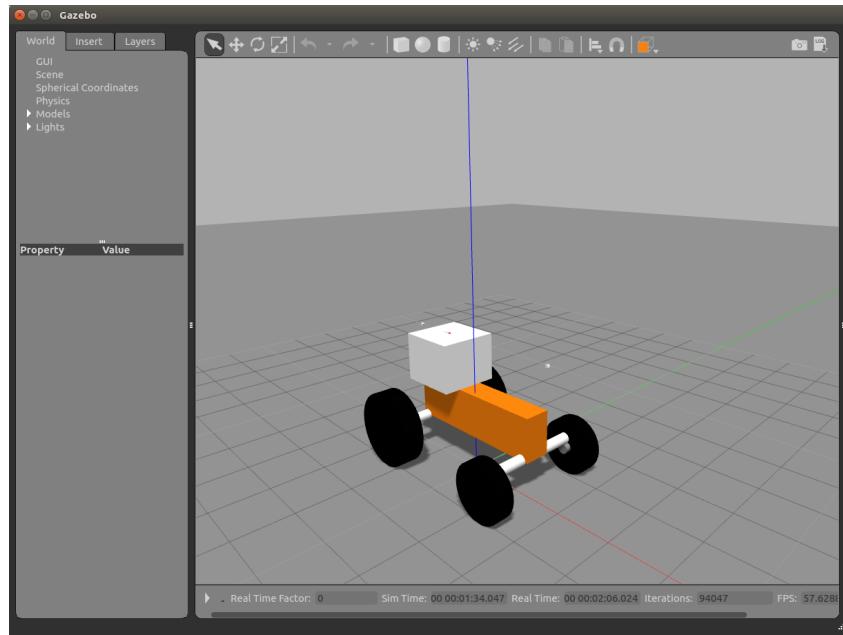


FIGURE 3.3 – Capture d’écran de l’interface graphique de Gazebo

### 3.2.1 Construction d'un robot virtuel

Un robot virtuel sous Gazebo est décrit par un fichier XML au format URDF, préconisé par l'environnement ROS. Ce format permet de spécifier un robot en ses éléments constitutifs simplifiés (chassis, roue, partie de bras, etc) de décrire comment ces éléments sont placés les uns par rapport aux autres.

Chaque élément du robot doit comporter 3 parties différentes :

- un tag `<collision>` qui décrit le modèle 3D avec lequel le simulateur doit calculer les collisions de l'élément avec d'autre objets,
- un tag `<visual>` qui décrit le modèle 3D de l'apparence visuelle de l'élément (pour affichage dans le simulateur),
- un tag `<inertial>` qui décrit les caractéristiques inertielles de l'élément.

```

1 <?xml version="1.0"?>
2 <robot name="mybot" xmlns:xacro="http://www.ros.org/wiki/xacro">
3   <xacro:property name="PI" value="3.1415926535897931"/>
4   <xacro:property name="chassisHeight" value="0.1"/>
5   ...
6   <link name='chassis'>
7     <collision>
8       <origin xyz="0_0_${wheelRadius}" rpy="0_0_0"/>
9       <geometry><box size="..." /></geometry>
10    </collision>
11    <visual>
12      <origin xyz="0_0_${wheelRadius}" rpy="0_0_0"/>
13      <geometry><box size="..." /></geometry>
14      <material name="orange"/>
15    </visual>
16    <inertial>
17      <origin xyz="0_0_${wheelRadius}" rpy="0_0_0"/>
18      <mass value="${chassisMass}"/>
19      <box_inertia .../>
20    </inertial>
21  </link>
22  ...
23  <link name="hokuyo_link">
24    <collision>...</collision>
25    <visual>...</visual>
26    <inertial>...</inertial>
27  </link>
28  ...
29  <joint name="hokuyo_joint" type="fixed">
30    <axis xyz="1_0_0"/>
31    <origin xyz="..." rpy="0_0_0"/>
32    <parent link="chassis"/>
33    <child link="hokuyo_link_1"/>
34  </joint>
35 </robot>

```

Listing 3.1 – Extrait d'une description URDF d'un robot

Les éléments sont positionnés les uns par rapport aux autres en spécifiant des commandes XML `<joint>` qui indiquent comment placer un élément `<child>` par rapport à un élément `<parent>` (voir listing 3.1). Le joint peut être de type "`fixed`", auquel cas les deux éléments sont rigidement liés dans la simulation. Où il peut être de type "`revolute`", "`planar`", "`continuous`", ... pour définir une articulation mobile en un élément parent et un élément enfant. Toute une arborescence d'éléments peut ainsi être agencée pour décrire une simulation de robot avec des parties fixes et des parties mobiles.

Pour plus de détails sur les capacités de description de URDF, voir <http://wiki.ros.org/urdf/XML>.

Le "monde" virtuel dans lequel doit évoluer le robot simulé est spécifié par un fichier de type "`*.world`", fichier qui est interprété par Gazebo pour monter la simulation. Ce fichier est lui écrit en utilisant le format SDF, préconisé par Gazebo. Un exemple d'un tel fichier est montré listing 3.2.

```

1 <?xml version="1.0"?>
2 <sdf version="1.4">
3   <world name="myworld">
4     <include><uri>model://sun</uri></include>
5     <include><uri>model://ground_plane</uri></include>
6     <physics name="ode_200iters" type="ode" default="true">...</physics>
7   </world>
8 </sdf>

```

Listing 3.2 – Exemple d'un fichier "\*.world"

### 3.2.2 Vérification de disponibilité des capteurs

Gazebo propose toute une série de capteurs simulés, disponibles nativement ou par l'intermédiaire de modules tiers. Ces capteurs simulés génèrent des données en cohérence avec l'environnement simulé qui peut être rendu disponible aux modules ROS par l'intermédiaire de plugins adaptés.

Pour les besoins du projet Symeter V2, nous avons vérifié et testé la disponibilité des types de capteurs tels que listés dans le tableau suivant.

Capteur	Plugin Gazebo	Utilisabilité
IMU	<code>libgazebo_ros_imu_sensor.so</code>	oui, mais génère beaucoup de bruit
LIDAR	<code>libgazebo_ros_laser.so</code>	oui, fourni par constructeur
GPS	<code>libhector_gazebo_ros_gps.so</code>	oui, précision configurable

TABLE 3.1 – Plugins utilisables pour émuler les capteurs sous ROS/Gazebo

Pour vérifier la disponibilité et le bon fonctionnement de ces capteurs, nous avons utilisé un petit robot basé sur une implémentation fournie avec ROS et Gazebo, illustré figure 3.4.

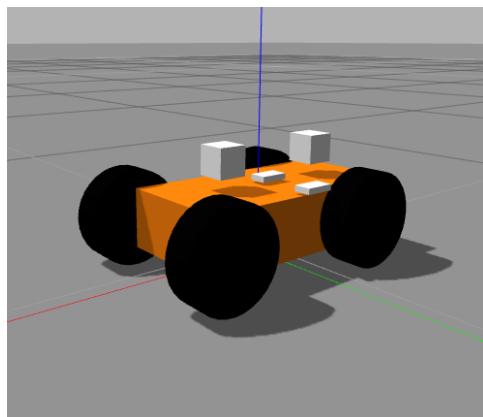


FIGURE 3.4 – Robot basique utilisé pour tester les plugins capteur

### 3.3 Contraintes de mise en oeuvre

La mise en oeuvre de ce petit robot au sein de Gazebo a été riche d'enseignements. Elle a permis notamment de détecter quelques problèmes gênant dans l'utilisation de Gazebo.

#### 3.3.1 Pas d'adhérence au démarrage de la simulation

La mise en oeuvre du robot de base a révélé rapidement révélé que Gazebo comporte quelques problèmes lié à la simulation de l'adhérence des roues avec les objets environnants, notamment le sol.

En particulier, les roues d'un robot ROS plongé dans Gazebo n'auront aucune adhérence sur le sol au démarrage de la simulation. Si la propulsion est mise en oeuvre, nous constatons que les roues tournent mais que le robot ne bouge pas, comme si les roues tournaient dans le vide, ou sur de la glace.

Après avoir changé des réglages dans le moteur physique de Gazebo, dans le réglage du coefficient de frottement des roues et de nombreux autres paramètres sans effet sur le problème, un contournement a été mis en place pour le mitiger : un node ROS a été écrit pour forcer la position du robot à des coordonnées spatiales bien précises, en utilisant le topic `/gazebo/set_model_state`.

Après ce forçage de position, Gazebo prend ensuite bien en compte l'adhérence des roues du robot et le robot bouge quand les roues tournent.

Cela fait que ce script, dénommé `setpose`, doit être invoqué systématiquement après le démarrage de la simulation pour que l'adhérence des roues soit bien prise en compte par le moteur physique de Gazebo.

#### 3.3.2 Simulation mécanique, frottements, adhérence

Simuler un robot avec des points de contact multiples dans Gazebo peut poser parfois problème si les points multiples dérapent. Dans ce cas il est rare que l'adhérence soit récupérée.

Ceci est exacerbé par le fait que les matériaux simulés par Gazebo sont par défaut infiniment rigides et que le moindre choc se répercute dans les points de contact avec le sol.

#### 3.3.3 Conclusions sur les contraintes

Pour conclure avec les limitations et contraintes de Gazebo, la leçon principale est qu'il faut au maximum éviter que le véhicule entre en dérapage : il ne récupère généralement pas, et devient inutilisable : il faut relancer la simulation.

Les raisons pour lesquelles le véhicule peut entrer en dérapage sont nombreuses :

- Mauvaise configuration des paramètres de frottement des éléments en contact avec le sol.
- Vitesses de rotation différentes des roues, soit par une mauvaise consigne de vitesse sur l'une des roues, soit une non prise en compte du différentiel de vitesse entre roues intérieur et extérieur dans un virage, etc
- En virage, les roues intérieures et extérieures sont soumises à des rayons différents. S'ils ne sont pas pris en compte correctement, une ou plusieurs roues sont susceptible de déraper.

## 3.4 Mise en Oeuvre : simulation d'un environnement de tassage de silo

Cette section décrit comment les différents composants de la simulation du chantier d'ensilage sont montés, notamment le tracteur qui a demandé le plus de travail.

### 3.4.1 Montage d'un tracteur simulé

L'intérêt de modéliser un tracteur plutôt que le petit robot utilisé précédemment est double. Tout d'abord cela nous permet de rapprocher la simulation du comportement dynamique du d'un vrai tracteur, ce qui nous permettra de travailler sur le module de localisation à partir d'une dynamique plus réaliste.

La deuxième raison est que ce tracteur permettra dans un étape ultérieure du projet d'étudier différentes configurations d'implantation de capteurs. Il est par exemple envisagé d'équiper Symeter V2 de 2 LIDARS de manière à assurer que l'ensemble du silo sera balayé par au moins un faisceau LIDAR sans que le tracteur n'ai de manœuvre complexe à effectuer. Ce modèle de tracteur permettra de tester de nombreuses configurations différentes, de manière reproductible et avec beaucoup de souplesse.

#### 3.4.1.1 Modélisation du tracteur

**3.4.1.1.1 Modélisation physique** Le tracteur virtuel été modélisé en se basant sur les dimensions générales d'un tracteur Fendt type 512, en utilisant cependant des formes géométriques simplifiées. Le tracteur est donc composé d'un pavé en guise de châssis, d'un cube en guise de cabine, de quelques cylindres allongés en guise d'essieux, et de 4 grands cylindres aplatis en guise de roues, tel qu'illustré dans la figure 3.5.

L'arborescence de transformations entre les différents éléments du tracteur est illustrée figure 3.6

**3.4.1.1.2 Actuateurs et Contrôleurs** Pour permettre une conduite en terrain accidenté, le tracteur simulé sera muni de 4 roues motrices, avec 2 roues directrices à l'avant.

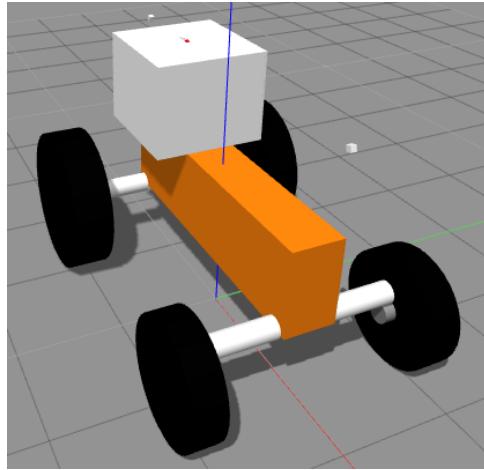


FIGURE 3.5 – Modèle 3D du tracteur simulé

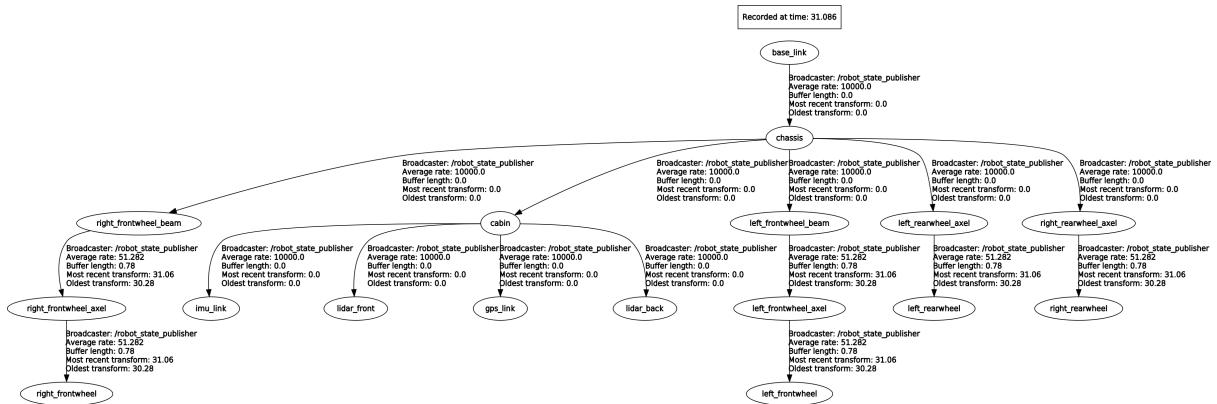


FIGURE 3.6 – Arborescence de transformations du modèle de tracteur simulé

Chacune des roues motrices est commandée en vitesse par son propre topic, telle que décrit dans le tableau suivant, et l’angle de chacune des roues de direction est commandée par un topic dédié.

Nom du contrôleur	Grandeur contrôlée
left_frontwheel_velocity_controller	Vitesse roue avant gauche
right_frontwheel_velocity_controller	Vitesse roue avant droite
left_rearwheel_velocity_controller	Vitesse roue arrière gauche
right_rearwheel_velocity_controller	Vitesse roue arrière droite
right_steer_controller	Angle direction roue avant droite
left_steer_controller	Angle direction roue avant gauche

TABLE 3.2 – Liste des contrôleurs du tracteur simulé

### 3.4.1.2 Propulsion et Guidage

Du fait des contraintes exposées dans le paragraphe 3.3.3, la consigne de vitesse sur chacune des roues doit être cohérente vis-à-vis de la vitesse de consigne, vis-à-vis du rayon de chacune des roues, ainsi qu’avec le rayon de virage imposé. De plus l’angle de chacune des roues directrices doit être réglé de manière différentielle en fonction de l’angle de direction donné en consigne.

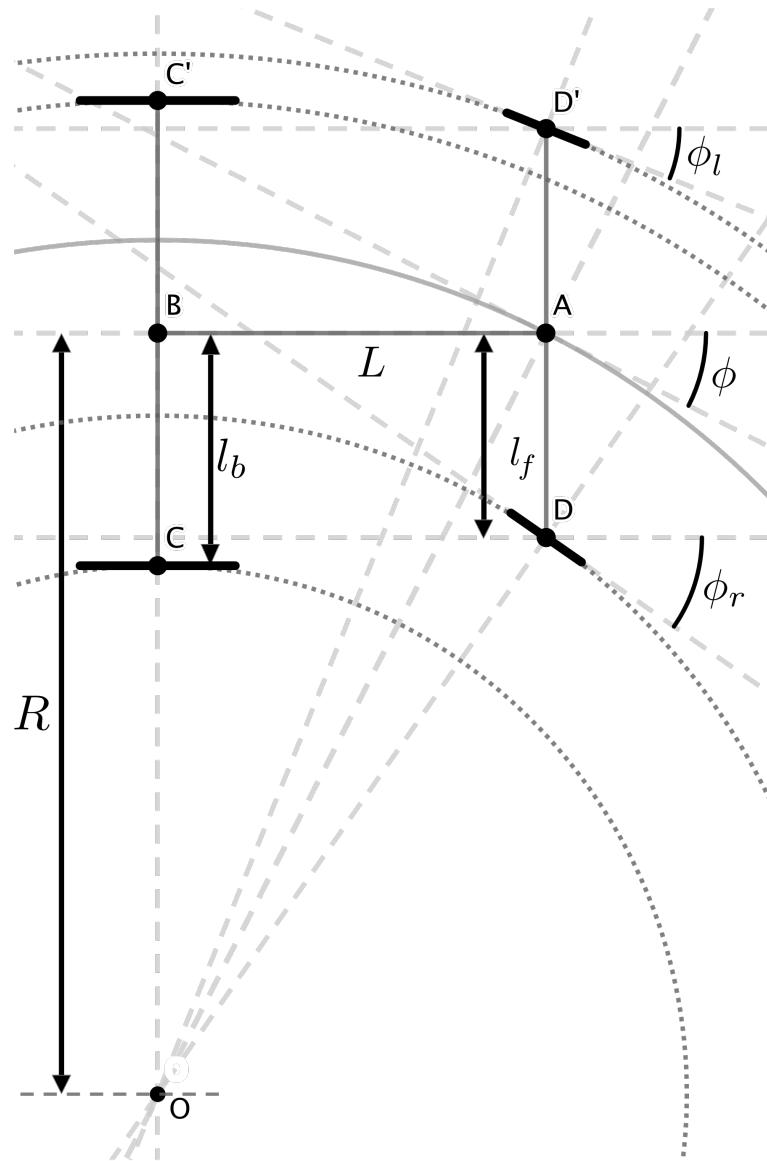


FIGURE 3.7 – Géométrie du différentiel de direction

#### 3.4.1.2.1 Algorithme

La problématique de la direction est illustrée par la figure 3.7 : si l'on donne une consigne d'angle de direction générale  $\phi$ , quels doivent être les angles  $\phi_l$ , pour la roue avant gauche, et  $\phi_r$ , pour la roue avant droite, pour que ces deux roues restent tangentes à leur trajectoire propre ?.

Les données d'entrée sont les suivantes :

- $L$  l'empattement du tracteur, c'est à dire la distance entre les deux essieux,
- $l_f$  la distance entre l'axe avant/arrière du tracteur et le centre d'une roue avant (les roues sont supposées disposées de manière symétrique),
- $\phi$  l'angle de consigne.

Pour déterminer tous ces angles, nous d'abord devons déterminer  $R$  le rayon de courbure de la trajectoire du point central B de l'essieu arrière. Si nous considérons le triangle rectangle (OAB), nous constatons que l'angle (OA,OB) est le même que  $\phi$ . Nous pouvons donc écrire, en posant  $L = AB$  et  $R = OB$

$$\tan \phi = \frac{L}{R} \quad (3.1)$$

d'où

$$R = \frac{L}{\tan \phi} \quad (3.2)$$

Par un raisonnement similaire nous pouvons trouver une formule pour  $\phi_l$  et  $\phi_r$  :

$$\tan \phi_l = \frac{L}{R + l_f} \iff \phi_l = \arctan \frac{L}{R + l_f} \quad (3.3)$$

$$\tan \phi_r = \frac{L}{R - l_f} \iff \phi_r = \arctan \frac{L}{R - l_f} \quad (3.4)$$

En incorporant 3.2 dans 3.3, nous obtenons

$$\phi_l = \arctan \frac{L}{\frac{L}{\tan \phi} + l_f} = \arctan \frac{L}{\frac{L + l_f \tan \phi}{\tan \phi}} = \arctan \frac{L \tan \phi}{L + l_f \tan \phi} \quad (3.5)$$

De manière équivalente,

$$\phi_r = \arctan \frac{L \tan \phi}{L - l_f \tan \phi} \quad (3.6)$$

**3.4.1.2.2 Implémentation sous ROS** L'implémentation sous ROS de cet algorithme est effectué au moyen d'un node dédié, nommé `tracteur_steering.py`, qui fait partie du package `tracteur_control`. Ce node est implémenté en python et prend en entrée le topic `/tracteur/cmd_vel` qui comporte des messages de type `Twist`. Ce message contient deux informations : l'un comporte la consigne de vitesse du tracteur, et l'autre le taux de rotation à gauche ou à droite du tracteur.

Le node `tracteur_steering` calcule sur la base des consignes données 6 nouveau paramètres, sur la base des algorithmes exposés en 3.4.1.2.1 :

- les 4 consignes de vitesse pour chacune des roues
- les 2 consignes de direction pour chacune des roues directionnelle.

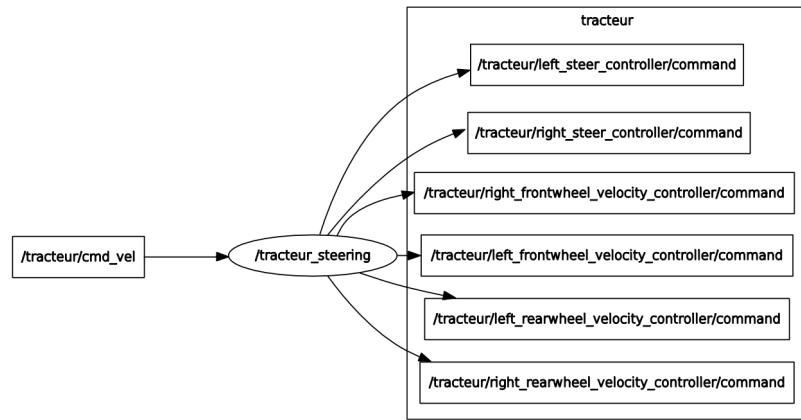


FIGURE 3.8 – Topics reçus et générés par le noeud tracteur \_steering

Ces consignes sont ensuite diffusées chacune vers les topics des contrôleurs de vitesse de rotation des roues et ceux de commande de l'angle roues des roues directionnelles, selon le principe exposé en figure 3.8.

# Chapitre 4

## Mise en place du processus de localisation

Ce chapitre présente la problématique de localisation par fusion de données IMU et GPS. Malgré l'objectif de mettre en oeuvre des composants disponibles "sur étagère" pour monter la version initiale de Symeter V2, il est quand même nécessaire de comprendre en détail les tenants et les aboutissants du problème de la localisation.

Ce problème est complexe et requiert d'en comprendre au moins les bases afin de permettre la configuration correcte des composants.

### 4.1 Présentation du problème

Comme indiqué en introduction, le but du processus de localisation est d'estimer en temps réel pour chaque instant  $t_k$  le vecteur d'état

$$\boldsymbol{x}_k = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \theta_{x_k} \\ \theta_{y_k} \\ \theta_{z_k} \end{bmatrix} \quad (4.1)$$

sur la base d'informations fournies par l'IMU et le GPS en mode RTK, ce vecteur d'état étant exprimé dans un référentiel fixe par rapport au chantier.

L'IMU fourni des informations sur l'inclinaison du véhicule, ainsi que les accélérations linéaires qu'il subit à une fréquence de l'ordre de 30 Hz. Cependant comme indiqué ci-dessus, calculer la position du véhicule à partir de la mesure des accélérations subies requiert une double intégration, sur un signal d'accélération qui est généralement très bruité. Ces éléments impliquent une forte probabilité de dérive dans le temps.

Le GPS en mode RTK peut fournir une mesure de la position du véhicule dans la scène avec une précision de l'ordre de quelques centimètres, mais à un taux de rafraîchissement

beaucoup plus bas, de l'ordre de 2 à 5 Hz. De plus le GPS ne fournit aucune information sur l'assiette du véhicule.

Selon [Gus15], il est possible de mettre en place un procédé qui maintient ce vecteur d'état sur la base de ces deux instruments, communément appelé la "fusion de capteurs". Nous mettrons en oeuvre un tel procédé.

*Note* : Comme le GPS fourni une position absolue dans le référentiel de la scène nous ne sommes pas dans une problématique SLAM (Simultaneous Localization And Mapping), où l'on tente de déterminer de manière imbriquée la position et l'environnement du véhicule. Dans notre cas, les deux problèmes sont décorrélés : d'abord nous déterminons la localisation et ensuite nous mesurons l'environnement.

### 4.1.1 Repères

L'IMU et le GPS peuvent fournir des informations à certains instant  $t_k$  qui permettront de mettre à jour le vecteur d'état pour l'incrément de temps suivant  $t_{k+1}$ . Cependant, les données exposées par les capteurs ne sont pas toutes dans le référentiel fixe par rapport au chantier.

L'IMU par exemple expose des composantes d'accélération linéaire qui sont attachées au référentiel de L'IMU lui-même, supposé fixe par rapport au véhicule, mais qui est donc mobile par rapport au chantier.

Le GPS lui n'est en mesure que de donner des mesures de positions que dans un référentiel lié à la surface de la Terre, en terme de latitude, longitude et altitude ( $\lambda, \phi, h$ ) et il faudra donc transformer cette position géocentrique en coordonnées du référentiel de chantier.

Pour travailler au problème de localisation, nous avons donc besoin d'au moins trois référentiels différents pour maintenir le vecteur d'état  $\mathbf{x}_k$  à partir des mesures de l'IMU et du GPS.

Selon [Gus15], nous pouvons donc distinguer les 4 référentiels suivants

#### 4.1.1.1 Repère Inertiel

Le repère inertiel est dans notre cas le repère dont l'origine se trouve au centre de masse de la Terre. L'axe  $z$  est aligné avec son axe de rotation et les axes  $x$  et  $y$  sont dans le plan équatorial de manière à ce qu'un repère direct soit formé. Ce système ne tourne pas avec la rotation de la Terre.

#### 4.1.1.2 Repère Centré et Fixe par Rapport à la Terre

Ce repère, aussi appelé repère ECEF (Earth-Centered, Earth-Fixed), est similaire au repère inertiel, avec la différence que le repère tourne avec la Terre. Son axe  $z$  passe par le centre de masse la Terre en pointant vers le pôle Nord, l'axe  $x$  passe par l'intersection de l'équateur et du méridien de la position du véhicule. L'axe  $y$  complète le référentiel

direct. Les coordonnées sont généralement exprimées dans ce repère par des coordonnées géodétiques, c'est à dire une longitude, une latitude et une altitude( $\lambda, \varphi, h$ ), et servent à indiquer la position d'un point par rapport à l'ellipsoïde par le standard WGS-84.

#### 4.1.1.3 Repère de navigation local

Le repère de navigation locale sert à mesurer des axes  $x$  et  $y$  tangents à la surface de la Terre. Généralement ROS utilise un repère de navigation local avec l'axe  $x$  pointant vers l'est, l'axe  $y$  pointant vers le nord et l'axe  $z$  pointant radialement vers l'espace. C'est le repère dans lequel la navigation est effectuée dans les applications de localisation où le déplacement du véhicule est négligeable par rapport à la taille de la Terre, et l'origine est placée au point de démarrage du véhicule.

#### 4.1.1.4 Repère du véhicule

Quand des capteurs fixés au châssis d'un véhicule génèrent des mesures, il convient pour exploiter ces mesures de disposer d'un repère attaché directement à ce châssis. L'origine du repère se trouve au centre de masse, l'axe  $z$  pointe vers le haut du véhicule, l'axe  $x$  vers l'avant, et l'axe  $y$  complète le repère direct.

## 4.2 Filtres de Kalman

Selon [Gus15] et [MS16], la fusion de données IMU et GPS peut être effectuée sur la base d'un filtre de Kalman. Cette section s'attache à décrire les principes de fonctionnement d'un tel filtre pour en permettre l'utilisation pour le projet Symeter V2.

Un filtre de Kalman est un algorithme qui permet d'estimer les états d'un système dynamique sur la base de mesures incomplètes ou bruitées. Il a été développé à la fin des années 50 et comprend de très nombreuses applications dans tous les domaines liés à l'instrumentation.

### 4.2.1 Filtres de Kalman Linéaires

Selon [ZM09], les filtres de Kalman sont applicables sur des systèmes dynamiques linéaires modélisés par un ensemble d'équations différentielles, décrites par la relation suivante :

$$\dot{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{Gu} + \mathbf{w} \quad (4.2)$$

où

- $\mathbf{x}$  est le vecteur colonne des états du système,
- $\mathbf{F}$  est la dynamique du système,
- $\mathbf{u}$  un vecteur connu, souvent appelé vecteur de contrôle.
- $\mathbf{w}$  est un bruit blanc, lui aussi exprimé sous forme d'un vecteur.

La matrice de bruit de process  $\mathbf{Q}$  est par définition reliée au vecteur  $\mathbf{w}$  selon la formule :

$$\mathbf{Q} = \mathbf{E}[\mathbf{w}\mathbf{w}^T] \quad (4.3)$$

( $\mathbf{E}[\bullet]$  représente la fonction "espérance de  $[\bullet]$ ".)

Dans notre cas, le système Symeter V2 n'aura pas accès au contrôle, nous choisissons donc l'intégrer la composante  $\mathbf{Gu}$  dans le bruit de process  $\mathbf{w}$ . Le modèle dynamique 4.2 devient donc

$$\dot{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{w} \quad (4.4)$$

Toujours selon [ZM09], l'état du système est estimé à partir de mesures  $\mathbf{z}$  dont la formulation de Kalman impose qu'elles soient linéairement liées aux variables d'état par la relation

$$\mathbf{z} = \mathbf{H}\mathbf{x} + \mathbf{v} \quad (4.5)$$

où

- $\mathbf{z}$  est le vecteur des mesures
- $\mathbf{H}$  est la matrice de transfert des mesures
- $\mathbf{v}$  représente le bruit de mesure.

La matrice de bruit de mesure  $\mathbf{R}$  est reliée au vecteur de bruit de mesure selon la relation :

$$\mathbf{R} = \mathbf{E}[\mathbf{v}\mathbf{v}^T] \quad (4.6)$$

#### 4.2.1.1 Discrétisation

Les filtres de Kalman étant des algorithmes discrets, il convient de discréteriser les relations précédentes afin de pouvoir monter les relations de récurrence des filtres.

Toujours selon [ZM09], si la période de mesure est  $T_s$  nous pouvons discréteriser la dynamique du système en trouvant la matrice fondamentale  $\Phi(y)$  de 2 manières équivalentes

#### Evaluation de $\Phi(t)$ à l'aide de la Transformée de Laplace Inverse

Pour un système dynamique invariant dans le temps décrit par une matrice  $\mathbf{F}$  la matrice fondamentale  $\Phi(t)$  peut être trouvée à l'aide de la formule suivante :

$$\Phi(t) = \mathcal{L}^{-1}[(s\mathbf{I} - \mathbf{F})^{-1}] \quad (4.7)$$

où  $\mathbf{I}$  est la matrice identité,  $\mathbf{F}$  est la dynamique du système et  $\mathcal{L}^{-1}$  est la transformée de Laplace inverse.

#### Evaluation de $\Phi(t)$ par expansion en séries de Taylor

La matrice fondamentale peut aussi être retrouvée en effectuant l'expansion en série de Taylor suivante :

$$\Phi(t) = e^{\mathbf{F}t} = \mathbf{I} + \mathbf{F}t + \frac{(\mathbf{F}t)^2}{2!} + \cdots + \frac{(\mathbf{F}t)^n}{n!} + \cdots \quad (4.8)$$

Nous pouvons enfin calculer la matrice fondamentale  $\Phi_k$  pour une période d'échantillonnage  $T_s$  en prenant :

$$\Phi_k = \Phi(T_s) \quad (4.9)$$

La forme discrète de l'équation des mesures 4.5 devient

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \quad (4.10)$$

et 4.6 devient

$$\mathbf{R}_k = \mathbf{E}(\mathbf{v}_k \mathbf{v}_k^T) \quad (4.11)$$

où  $\mathbf{R}_k$  est une matrice composée des variances de chacune des sources de bruits de mesure.

#### 4.2.1.2 Boucle de Kalman

Le filtre de Kalman pourrait être écrit en une seule équation mais il est généralement conceptualisé en deux phases distinctes, la **prédition** et la **mise à jour**.

L'état du filtre de Kalman un instant donné est composé de

- $\mathbf{x}_k$  l'estimation de l'état à l'instant  $k$
- $\mathbf{P}_k$  la matrice de covariance de l'erreur sur les composantes de  $\mathbf{x}_k$ .

Selon [MS16] La phase de **prédition** utilise l'état estimé de l'instant précédent pour calculer une estimation de l'état actuel, en se basant sur le modèle dynamique. En partant de 4.4, la prédition utilise les formules 4.12 et 4.13 pour déterminer  $\hat{\mathbf{x}}_k$  l'état prédit et  $\hat{\mathbf{P}}_k$  la matrice de covariance prédite.

$$\hat{\mathbf{x}}_k = \Phi_k \mathbf{x}_{k-1} \quad (4.12)$$

$$\hat{\mathbf{P}}_k = \Phi_k \mathbf{P}_{k-1} \Phi_k^T + \mathbf{Q}_k \quad (4.13)$$

Dans la phase de **mise à jour**, les observations de l'état courant (c'est à dire les mesures  $\mathbf{z}_n$ ) sont incorporées pour corriger l'état prédit dans le but d'obtenir une meilleure précision.

Tout d'abord le **gain de Kalman**  $\mathbf{K}$  est calculé selon l'équation 4.14.

$$\mathbf{K} = \hat{\mathbf{P}}_k \mathbf{H}^T (\mathbf{H} \hat{\mathbf{P}}_k \mathbf{H}^T + \mathbf{R}_k)^{-1} \quad (4.14)$$

Ensuite l'état du filtre, représenté par  $\mathbf{x}_k$ , le vecteur d'état et  $\mathbf{P}_k$  la matrice de covariance selon les formules 4.15 et 4.16 respectivement.

$$\mathbf{x}_k = \hat{\mathbf{x}}_k + \mathbf{K}(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k) \quad (4.15)$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}\mathbf{H})\hat{\mathbf{P}}_k(\mathbf{I} - \mathbf{K}\mathbf{H})^T + \mathbf{K}\mathbf{R}\mathbf{K}^T \quad (4.16)$$

#### 4.2.1.3 Fonctionnement conceptuel du Filtre de Kalman

Pour comprendre conceptuellement ce qu'accomplit un filtre de Kalman, il suffit d'analyser les formules 4.14 du gain de Kalman et 4.15 de mise à jour du vecteur d'état.

En effet, dans 4.14, si la matrice de covariance des mesures  $\mathbf{R}_k$  est prépondérante par rapport à  $\hat{\mathbf{P}}_k$ , la matrice de covariance de l'erreur du vecteur d'état, la magnitude de  $\mathbf{K}$  sera petite, l'influence des mesures  $\mathbf{z}_k$  dans 4.15 sera petite, c'est à dire que le filtre fera plus confiance à son estimation du vecteur d'état  $\hat{\mathbf{x}}_n$  qu'aux mesures.

Par contre, si  $\mathbf{R}_k$  est petite par rapport à  $\hat{\mathbf{P}}_k$ , le gain de Kalman aura une grande magnitude, reflétant le fait que la mesure sera plus précise que l'estimation. Et donc  $\mathbf{x}_k$  dans 4.15 sera plus influencée par  $\mathbf{z}_k$  que par  $\hat{\mathbf{x}}_n$ .

Selon [ZM09], la formule du gain de Kalman 4.14 est en fait appelée **Gain de Kalman Optimal** et est obtenue par minimisation la variance de l'erreur d'estimation.

#### 4.2.2 Estimation de Pose 3D : Filtres de Kalman Etendus

Dans la plupart des applications d'estimation d'état, la dynamique des systèmes à estimer n'est pas linéaire et le filtre de Kalman ne peut pas être appliqué en l'état car toute sa formulation se base sur des hypothèses de linéarités fortes.

En particulier dans le cas de Symeter V2, les capteurs IMU et GPS ne peuvent pas être placés au centre de masse du véhicule et le modèle Newtonien dérivé de ce placement introduit immédiatement des non-linéarités dans la dynamique.

Selon [MS16] et [ZM09], il est possible de généraliser les filtres de Kalman pour estimer des systèmes dont la dynamique est non-linéaire et dont la trajectoire n'est pas connue à l'avance.

Selon [MS16], un tel procédé non linéaire peut être décrit par l'équation 4.17 suivante

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1} \quad (4.17)$$

où  $\mathbf{x}_k$  est le vecteur d'état,  $f$  est une fonction de transition d'état non linéaire et  $\mathbf{w}_{k-1}$  est le bruit de processus.

Dans le cas qui nous intéresse où Symeter V2 doit déterminer la pose 3D à 6 degrés de liberté du véhicule,  $\mathbf{x}_k$  représente la dite pose,  $f$  sera un modèle cinématique dérivé de la mécanique Newtonienne.

Selon [ZM09] l'extension des filtres de Kalman à une dynamique non-linéaire est effectuée en calculant à chaque itération  $\mathbf{F}$  le jacobien de  $f$  selon la formule 4.18

$$\mathbf{F} = \left. \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}=\hat{\mathbf{x}}} \quad (4.18)$$

et en discrétilisant ce jacobien en utilisant la formule 4.19

$$\Phi_k = e^{\mathbf{F}t} = \mathbf{I} + \mathbf{F}t + \frac{(\mathbf{F}t)^2}{2!} + \dots + \frac{(\mathbf{F}t)^n}{n!} + \dots \quad (4.19)$$

En pratique l'approximation 4.20 est le plus souvent suffisante

$$\Phi_k \approx \mathbf{I} + \mathbf{F}t \quad (4.20)$$

La boucle de Kalman telle que décrite dans 4.2.1.2 peut ensuite être utilisée en remplaçant  $\Phi$  par  $\Phi_k$  la matrice fondamentale du jacobien de  $f$ .

### 4.3 Mise en oeuvre sous ROS.

Comme les sections précédentes le montrent, l'implémentation d'un système de localisation sur la base mesures par IMU et GPS n'est pas une mince affaire et est plus un sujet de thèse qu'une sous-partie de stage.

L'environnement ROS propose des modules prêts à l'emploi pour implémenter un processus de localisation pour les systèmes de robotique mobile. L'offre est cependant nombreuse et il a fallu vérifier que les solutions proposées répondent bien aux besoins de Symeter V2.

Le besoin le plus important est que la localisation doit être effectuée sur un vecteur d'état compatible avec une scène 3D. Cela veut dire que le vecteur d'état estimé doit être d'au moins 6 dimensions : 3 pour la position spatiale du véhicule et 3 pour son assiette.

Il a donc fallu éliminer tous les modules qui se limitaient à des évolutions 2D du système, d'autre qui n'acceptaient qu'un nombre limité de capteurs, ou alors qui ne supportaient pas le GPS, par exemple.

Le choix final s'est finalement reposé sur le module `robot_localization`, qui est décrit plus en détail dans les sections suivantes.

#### 4.3.1 Module `robot_localization`

Selon le site internet de `robot_localization` (lien), ce module est une collection de nodes dédiés à l'estimation d'état. Chacun de ces nodes implémente un estimateur d'état non linéaire pour des véhicules évoluant dans un espace en 3 dimensions.

Il contient notamment le node `ekf_localization_node` qui est une implémentation d'un filtre de Kalman Etendu, et le node `navsat_transform_node` qui aide à l'intégration des données GPS.

Les caractéristiques principales du node `ekf_localization_node` sont les suivantes :

- Possibilité de fusionner un nombre arbitraire de capteurs
- Possibilité de prendre en compte de nombreux types de données différentes : odométrie, IMU, Pose avec covariance, etc.
- Possibilité de régler la configuration du node capteur par capteur
- Estimation en continu : dès que le node reçoit une mesure, il estime en continu le vecteur d'état, même en l'absence prolongée de nouvelle mesure, en utilisant un modèle interne pour la dynamique du véhicule.

Pour estimer en continu l'état du véhicule, le node `ekf_localization_node` maintient en interne un vecteur d'état à 15 dimensions :

$$[x \ y \ z \ \theta_x \ \theta_y \ \theta_z \ \dot{x} \ \dot{y} \ \dot{z} \ \dot{\theta}_x \ \dot{\theta}_y \ \dot{\theta}_z \ \ddot{x} \ \ddot{y} \ \ddot{z}] \quad (4.21)$$

pour générer le vecteur d'état de sortie à 6 degrés de liberté, sous la forme d'une transformation entre le référentiel de navigation local et le référentiel du véhicule, ayant les coordonnées  $[x \ y \ z \ \theta_x \ \theta_y \ \theta_z]$ , selon le schéma de principe décrit dans la figure 4.1.

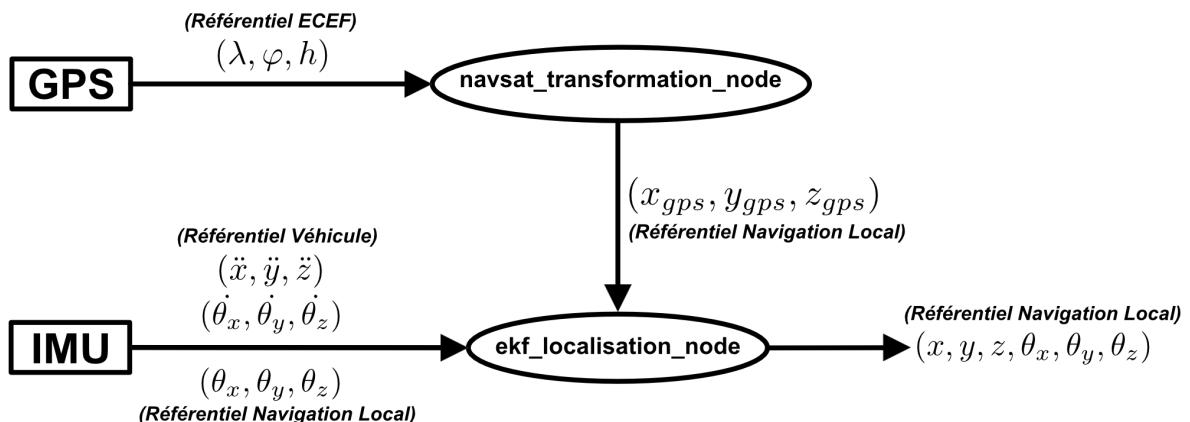


FIGURE 4.1 – Architecture de la fonction Localisation de Symeter V2

L'une des caractéristiques très utiles de `ekf_localization_node` est qu'il est capable de recevoir et traiter des vecteurs de mesure  $z_k$  partielles. Ceci lui permet de prendre en charge de très nombreux types de capteurs, avec des fréquences de mesures très variées, ce qui est un pré-requis dans notre cas l'IMU générant des données à environ 100Hz, alors que le GPS fonctionne à 5 Hz au grand maximum.

Cette caractéristique permet d'envisager l'ajout de capteurs supplémentaires, voir d'un nouveau type (odométrie, triangulation par vision artificielle,...) si nécessaire.

### 4.3.2 navsat\_transformation\_node

Le node `navsat_transformation_node` est le node qui prend en entrée les positions fournies par le GPS dans le repère ECEF, pour les transformer en coordonnées ( $x, y, z$ ) dans le repère local. Il est utilisé de manière concourante avec `ekf_localization_node` lorsque des données GPS doivent être utilisées.

### 4.3.3 Configuration des Capteurs

Pour que `ekf_localization_node` puisse utiliser les données des capteurs à disposition, il faut indiquer pour chacun le topic sur lequel les données sont diffusées, et indiquer quelles grandeurs du vecteur d'état sont susceptible d'être influencées par leurs informations. Ceci permet au node de pouvoir déterminer à chaque nouvelle mesure les grandeurs à soumettre au gain de Kalman quand leur mesure est disponible, et celles qui ne seront gérées que par la prédiction du modèle dynamique.

L'IMU fourni à chaque mesure les informations suivantes :

- Attitude du véhicule par rapport au référentiel de navigation local :  $\theta_x, \theta_y, \theta_z$ ,
- Vitesse angulaire de variation de l'attitude par rapport au référentiel local :  $\dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z$
- Accélérations linéaires par rapport au référentiel du véhicule :  $\ddot{x}, \ddot{y}, \ddot{z}$ .

La sortie du node `navsat_transformation_node` fourni les coordonnées  $x_{gps}, y_{gps}, z_{gps}$  du capteur GPS par rapport au référentiel de navigation local. Le node `ekf_localization-node` utilise ensuite les transformations internes diffusées par le topic `\tf` ainsi que l'estimation de l'attitude du véhicule pour estimer la position du centre de masse du véhicule par rapport au référentiel de navigation local.

Le node `ekf_localization_node` repose aussi fortement sur la valeur de la matrice de bruit de processus  $\mathbf{Q}$  dont la valeur doit être configurée en fonction des caractéristiques des capteurs.

### 4.3.4 Tests de localisation sous Gazebo

Les tests de l'implémentation de la fonction de localisation ont été dans un premier temps effectué uniquement en simulation en utilisant Gazebo. Gazebo permet en effet de fournir nativement les valeurs non-estimées, "réelles" du point de vue de la simulation, du vecteur d'état du véhicule, et donc permet de comparer les valeurs estimées avec ces valeurs "réelles" au sens de la simulation.

Après de nombreux tests, il apparait que l'estimation de l'attitude du véhicule fonctionne très bien lorsque que seules les informations d'attitudes instantanées de l'IMU simulé sont utilisées (voir fig. 4.2 et 4.3).

De même, l'estimateur ne donne des résultats cohérents en position ( $x, y, z$ ) qui si on ignore les accélérations linéaires pour ne prendre en compte que les données GPS pour estimer la position.

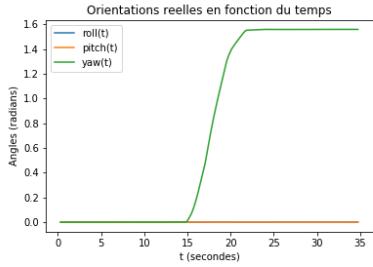


FIGURE 4.2 – Orientations Réelles

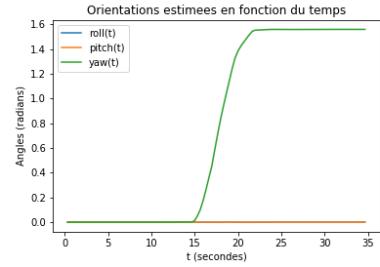


FIGURE 4.3 – Orientation estimées

Il apparaitra plus tard que l’IMU simulé utilisé est de très mauvaise qualité au regard des données d’accélération linéaires et des vitesses angulaires. Le module de localisation devra donc être testé avec des données réelles, non simulées pour une mise au point fine. Ceci sera couvert chapitre 6 quand nous tenterons d’exploiter des données issues d’une mesure réelle.

Lorsque le node `ekf_localization_node` est configuré, dans une simulation Gazebo, pour ne prendre en compte que les angles d’attitude issus de l’IMU d’une part, et les coordonnées du capteur GPS dans le référentiel local de navigation d’autre part, le vecteur d’état estimé est suffisamment précis pour tenter de construire un modèle numérique 3D du chantier à partir des données LIDAR, ce que nous allons évoquer dans le chapitre suivant.

# Chapitre 5

## Exploitation des données LIDAR

Ce chapitre présente la fonction acquisition et exploitation des données LIDAR pour le projet Symeter V2.

Nous nous attachons à mettre en place les principes de traitement des données LIDAR en vue de construire une représentation 3D du chantier, d'abord en montant le pipeline de traitement du flux de données LIDAR, puis en mettant en oeuvre une structure de données adaptée au stockage d'une scène 3D.

### 5.1 Présentation de la chaîne de traitement des données LIDAR

Le LIDAR utilisé par le système Symeter V2 étant de type "2D planar", les points retournés lors d'une unique mesure feront tous partie d'un même plan. L'ensemble des mesures sera donc constitué d'un ensemble de tranches qu'il faudra reconstituer à l'aide d'un traitement adapté.

La chaîne de traitement des données LIDAR est représentée dans la figure 5.1 et comporte les éléments suivants :

- Une source de données de type `laser`, issue des drivers ROS qui pilotent l'équipement LIDAR
- Un module de mise en forme du signal, qui prend en entrée les données `laser`, les filtre de manière à uniformiser la répartition des points de mesure, à réduire la bande passante et transformer les données laser en un nuage de points.
- Un module d'accumulation de nuages de points, qui prend en entrée les nuages de points issus du module de mise en forme en vue de les agrégner et reconstruire la chaîne.

### 5.2 Acquisition et mise en forme des données LIDAR

L'acquisition initiale des données LIDAR est effectuée par un driver approprié, généralement fourni par le constructeur de l'équipement. Les données sont structurées selon un format comportant la distance mesurée jusqu'au premier obstacle pour chaque incrément d'angle du LIDAR.

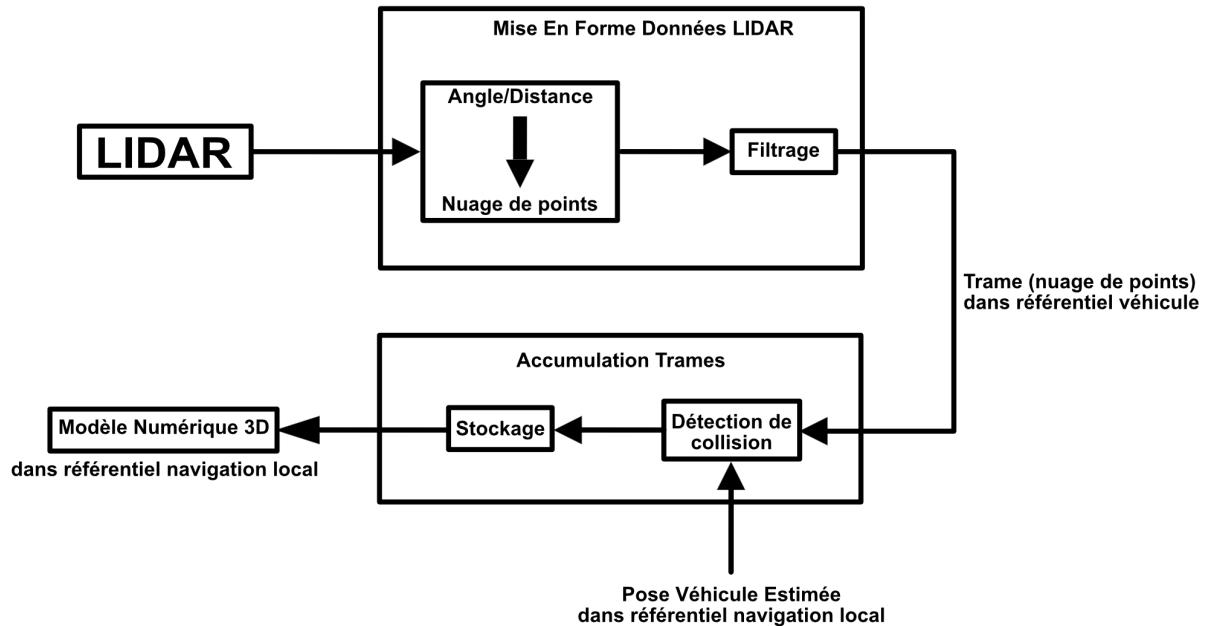


FIGURE 5.1 – Chaîne de traitement des données LIDAR

### 5.2.1 Transformation trame LIDAR en un nuage de points

La première étape du traitement est donc de transformer ces données de type **laser** (au sens de l'environnement ROS) pour générer un nuage de points dans le référentiel du véhicule.

Une trame LIDAR de type planar comporte essentiellement des couples (Angle/Distance) à propos d'une série de point de mesure. Si on connaît la pose du LIDAR dans le repère du véhicule, il est possible de calculer pour chaque couple (Angle,Distance) les coordonnées  $x_p, y_p, z_p$  du point détecté, dans le référentiel du véhicule.

Cette conversion a été implémentée sous ROS en écrivant un node dédié en C++ dénommé `laserscan_filter_node`.

### 5.2.2 Filtrage de la ligne de point par downsampling

A raison de 40 scans par secondes et de 1080 points mesurés par scan, un LIDAR génère 43200 points par secondes, ce qui est beaucoup.

De plus, du fait que la hauteur du LIDAR monté sur un tracteur sera au maximum d'environ 3 à 4 mètres de hauteur, et que l'acquisition des mesures est effectué sur la base d'un incrément d'angles, la densité des points de mesure sera beaucoup plus hétérogène que pour Symeter V1. Cela veut dire que nous aurons beaucoup plus de points de mesure directement sous le LIDAR que sur les côtés.

Nous allons donc effectuer un traitement qui permettra de moyenner les mesures de points autour de positions régulièrement réparties tous les 5 cm (par exemple). Cela

permettra de réduire le nombre de points à incorporer dans le modèle numérique, tout en régularisant la répartition des points dans l'espace.

D'après [ML16], il existe plusieurs méthodes de filtrage de nuages de points. Ceci est effectué en utilisant un procédé basé sur les Voxels

Le principe des Voxels revient à subdiviser le volume contenant un modèle numérique 3D (par exemple, un volume contenant un nuage de point) en cubes de petite taille, constituant chacun un "élément de volume", un "Volume Element" en anglais, contracté en "Voxel" (par analogie à "Pixel" pour "Picture Element").

Le filtrage consiste donc à considérer le sous-ensemble de points du nuage contenus dans l'un de ces voxel, et de calculer le point moyen de ce sous ensemble. Ce point sera le résultat du filtrage, et le nuage filtré sera l'ensemble des points moyens du nuage, un par Voxel. Les voxels ne comportant aucun point du nuage originel restent vides.

Le résultat final de ce traitement est que d'un nuage de points irrégulièrement espacé et avec beaucoup de bande passante nous obtenons un nombre réduit de points régulièrement espacés, idéalement conditionné pour être pris en compte de manière fiable par l'accumulateur.

Traitements a été implémenté pour Symeter V2 en utilisant un node fourni par la Point Cloud Library, le node `voxel_grid`

Le traitement est illustré par les captures d'écran ci-dessous. La figure 5.2 représente le nuage d'acquisition non filtré (ligne derrière le tracteur), alors que la figure 5.3 représente le nuage filtré. Ce nuage est beaucoup moins dense et régulièrement espacé.

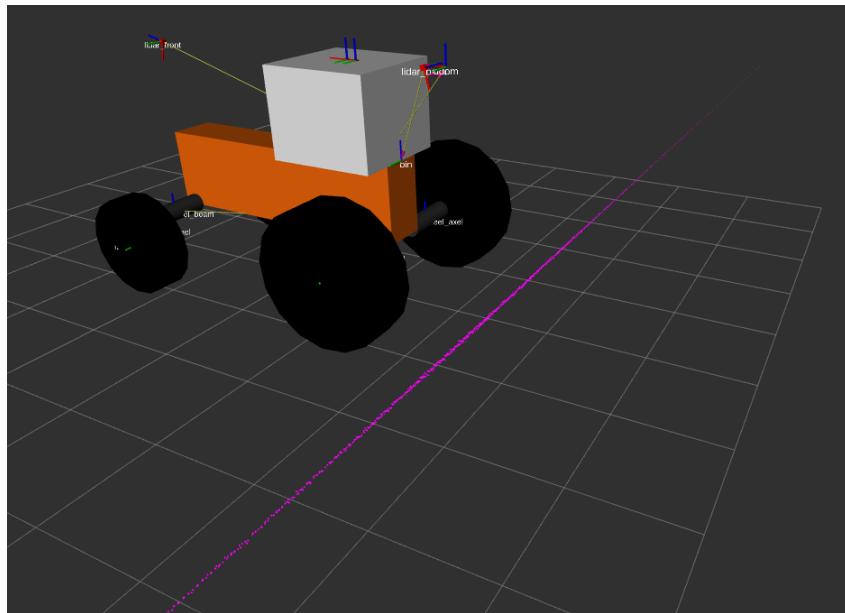


FIGURE 5.2 – Représentation du nuage de points non filtré

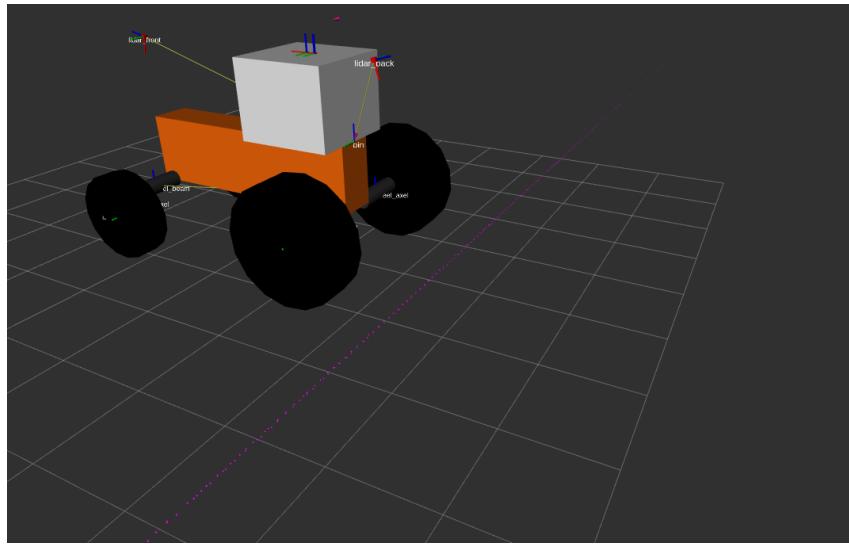


FIGURE 5.3 – Nuage de point filtré par downsampling à base de Voxel Grid.

### 5.3 Accumulation des nuages de points

Une fois les données LIDAR mises en forme, il convient maintenant de les exploiter pour reconstituer la représentation 3D du chantier. Cette tâche est effectuée par un module que l'on appelle le `point_cloud_aggregator`. Comme son nom l'indique ce module agrège et accumule les nuages de points issus de l'aquisition pour générer la représentation 3D du chantier.

### 5.3.1 Spécifications pour le point cloud aggregator

Le `point_cloud_aggregator` dans un premier temps agrège les nuages de points, c'est à dire qu'il prend un nuage de point en entrée, et tente de les ajouter au nuages de points qu'il a déjà en mémoire. Les points sont d'abord transformés du référentiel du véhicule au référentiel local de navigation.

S'il tente d'ajouter un point à une position dans l'espace pour laquelle un point existe déjà en mémoire (collision), ce nouveau point n'est pas ajouté. Cela évite ainsi les points doubles.

Cela met cependant une contrainte forte sur la structure de données stockant les points déjà accumulés : il faut pouvoir identifier de manière unique tous les éléments de volume inclus dans la représentation de la scène afin de pouvoir détecter les collisions.

### 5.3.2 Principe de stockage des données 3D

Il existe des structures de données optimisées qui permettent de stocker des données selon une indexation spatiale 3D. Ces structures reposent toujours sur une discréétisation du volume considéré sous forme de volumes élémentaires (les Voxels), mais permettent en plus de naviguer d'un élément de volume à l'autre de manière relativement simple, et surtout permettent d'ajouter des métadonnées à un élément donné, par exemple pour indiquer la présence ou non d'un objet dans le volume concerné.

La Point Cloud Librairie propose nativement deux structures de données spatiale pour les espaces 3D : les B-trees, qui sont des arbres auto-équilibrés, et les Octrees. Les B-trees sont très adaptés à des structures statiques. L'ajout d'un nouveau point dans un B-tree est très couteux car le ré-équilibrage de l'arborescence de données est complexe.

Les octrees sont eux beaucoup plus souple d'emploi lorsqu'on doit ajouter des données dynamiques. Pour plus de détail sur ces structures de données spatiales, voir [SXZ17].

### 5.3.3 Mise en oeuvre : `octomap`

L'implémentation initiale de ce module d'accumulation et de stockage a été effectuée en utilisant un module open source dénommé `octomap`. Ce module implémente un algorithme de gestion de grille d'occupation 3D. Symeter V2 alimente ce module `octomap` avec les trames de nuages de point issues de l'acquisition LIDAR, et peut ainsi accumuler le modèle numérique 3D du chantier. Pour de plus amples informations sur `octomap`, voir [HWB<sup>+</sup>13].

## 5.4 Mise en oeuvre sous Gazebo

La section précédente décrivait les principes de traitement des données LIDAR en vu de construire une représentation 3D d'un chantier d'ensilage. La présente section va illustrer la mise en oeuvre de ces principes à l'aide la plateforme de simulation décrite dans le chapitre 3 évoluant dans un chantier d'ensilage simulé.

### 5.4.1 Modélisation d'un chantier d'ensilage

Le modèle simulé du chantier d'ensilage est représenté pour 2 murs chacun long de 10 mètres, haut de 3 mètres et d'épaisseur 0,2 mètre, espacés de 9 mètres. Ce modèle est créé en utilisant les primitives XML fournies par le logiciel Gazebo. Le résultat peut être vu figure 5.4.

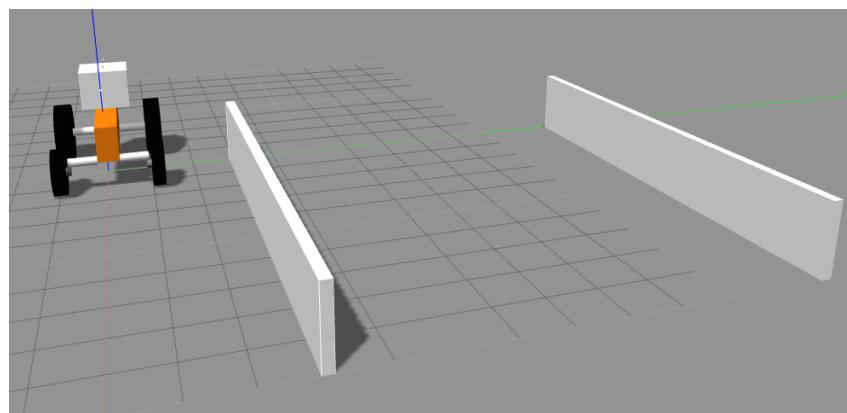


FIGURE 5.4 – Simulation sous Gazebo avec tracteur et silo vide.

Un tas d'ensilage peut aussi être ajouté au chantier. Ce modèle de tas d'ensilage a été construit en utilisant dans un premier temps le logiciel de modélisation 3D **blender**. Ce modèle 3D est ensuite incorporé dans le chantier simulé en l'important dans la description du world 3D de manière appropriée, tel que montré dans la figure 5.5.

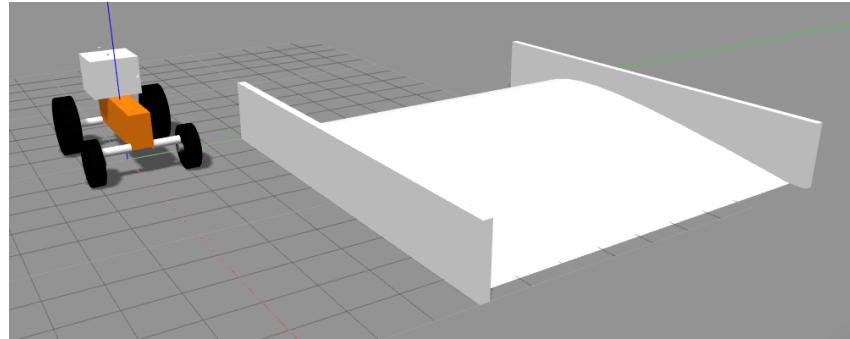


FIGURE 5.5 – Simulation sous Gazebo avec silo rempli.

#### 5.4.2 Test sous gazebo

Une fois la simulation de chantier montée, il est possible de faire évoluer le tracteur simulé dans le chantier. L'opérateur peut ainsi diriger le tracteur de manière à ce que le LIDAR monté à l'arrière du tracteur puisse scanner séquentiellement dans son ensemble le silo (voir figure 5.6).

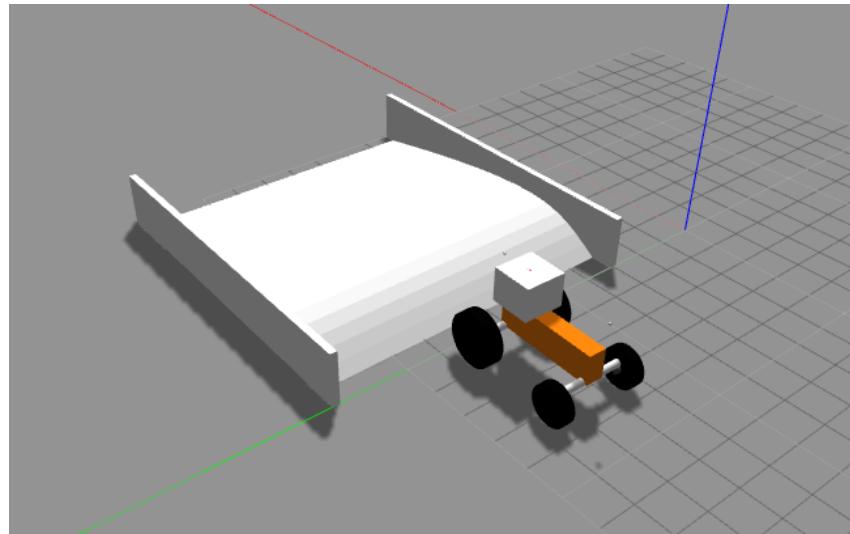


FIGURE 5.6 – Tracteur en train de scanner le silo (vue Gazebo)

Une fois que le tracteur a manœuvré de manière à permettre le faisceau du LIDAR de peindre la surface du silo dans son ensemble, le modèle numérique 3D du chantier est disponible (voir figure 5.7).

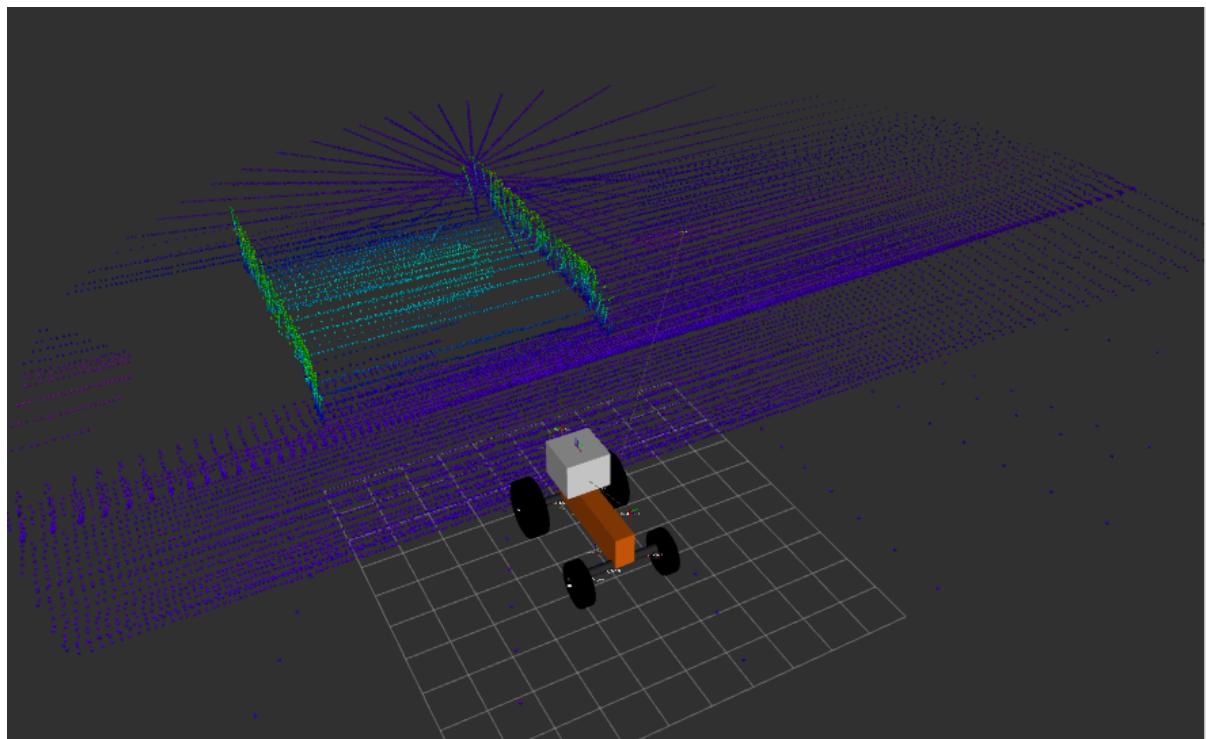


FIGURE 5.7 – Affichage du modèle numérique 3D reconstitué par Symeter V2

# Chapitre 6

## Mise en oeuvre à partir de mesures réelles

Les chapitres précédents ont montré comment le système Symeter V2 a été initialement conçu et monté à l'aide d'outils de simulation. Après avoir obtenu une première série de résultat par ce biais, quelques questions ont émergé concernant la validité de certains aspects de la simulation, et de savoir si la simulation représentait un chemin réaliste de développement.

C'est pour cela qu'une campagne d'acquisition de données réelles, à l'aide des capteurs physiques réels a été effectuée.

### 6.1 Protocole de test

#### 6.1.1 Instruments d'acquisition

Les capteurs (IMU, GPS, Lidar) ont été montés sur la camionnette de Tellus Environnement (affectueusement appelée "TellusCar" par l'équipe) et connectés à un ordinateur disposant de l'environnement ROS (fig. 6.1 et 6.2).

Le but des tests était de faire évoluer la Telluscar ainsi équipée en vue d'enregistrer simultanément les données générées par l'IMU, le GPS et le LIDAR, pour ensuite les rejouer à posteriori et les soumettre à l'implémentation de Symeter V2.

#### 6.1.2 Exploitation

Les données sont enregistrée en utilisant l'outil `rosbag` de ROS qui permet d'enregistrer tous les topics diffusés lors d'une session de fonctionnement. L'outil `rosbag` permet ensuite de rejouer ces captures et d'alimenter des nodes déployés indépendamment et ainsi les tester.

C'est ainsi que l'on peut retrouver la trace des données GPS brutes extraites des captures (figure 6.3), les données générées par l'IMU (figure 6.4), entre autre.



FIGURE 6.1 – Telluscar avec capteurs montés à l’arrière

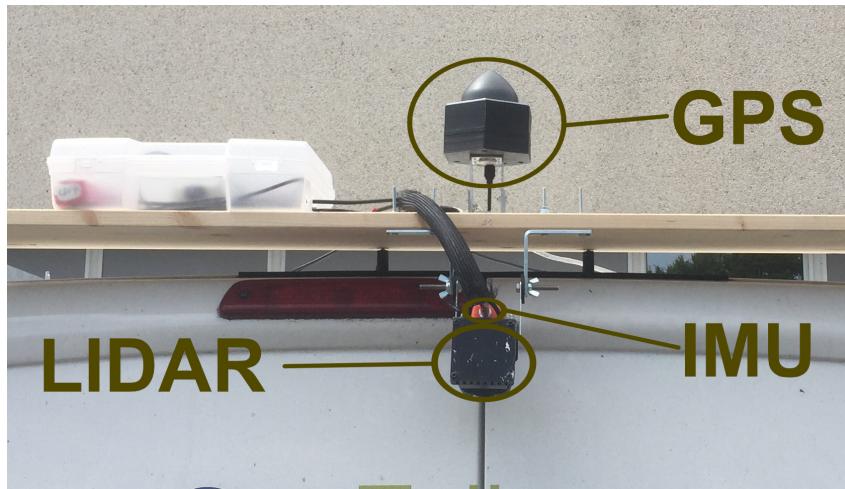


FIGURE 6.2 – Détail des capteurs monté l’arrière de la Telluscar.

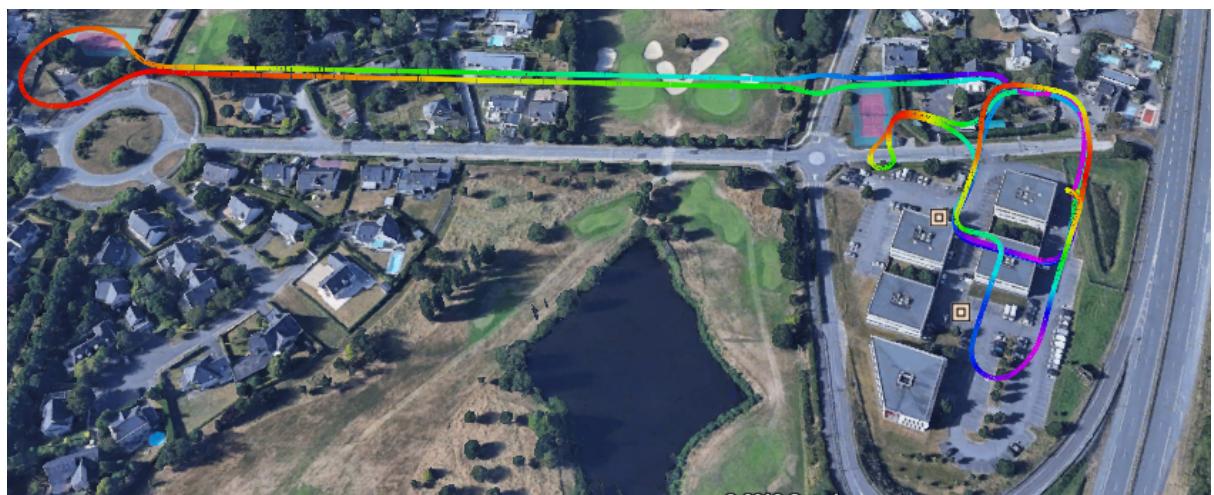


FIGURE 6.3 – Exemple de trajectoire capturée par le GPS lors d’une capture de données

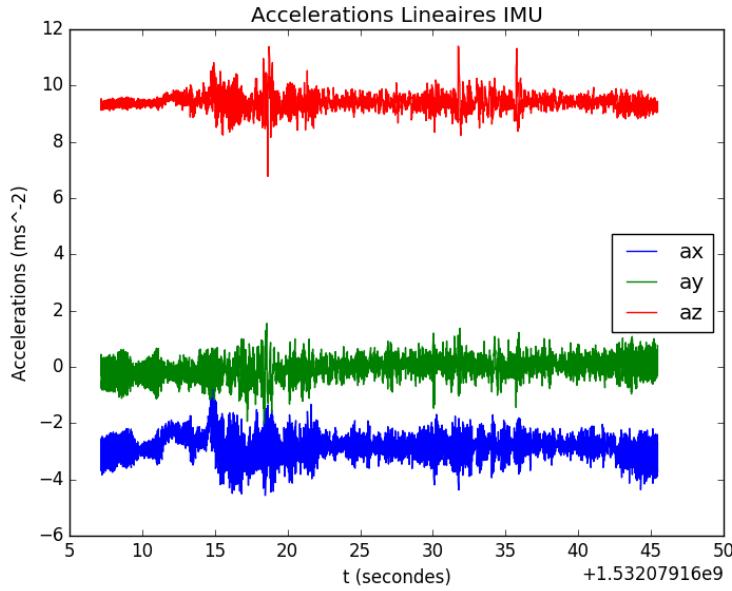


FIGURE 6.4 – Captures des accélérations linéaires issues de l’IMU XSens.

## 6.2 Exploitation des données

L’exploitation des données en vue de faire fonctionner le logiciel Symeter V2 à partir de données GPS, IMU et LIDAR enregistrées est en cours et n’a pas encore donné de résultat concluant.

Deux difficultés sont rencontrées qui devront être levées pour permettre de reconstituer le trajet et l’évolution de la pose du véhicule dans le temps. Tout d’abord la pose de l’IMU par rapport au véhicule à l’air d’avoir une influence très importante sur le comportement du `ekf_localization_node`. Si cette pose n’est pas exactement connue par le système Symeter V2, il est difficile de conserver une estimation fiable de la localisation. Ce problème n’était pas apparu dans la simulation car dans ce cas la pose du l’IMU simulé est implicitement connue avec précision.

Le deuxième problème est que les données d’accélérations générées par l’IMU le sont selon un référentiel qui n’a pas l’air d’être celui attendu par `ekf_localization_node`. Il faut donc rechercher quel référentiel est effectivement utilisé par l’IMU, corriger les données enregistrées de manière à compenser les incohérences d’orientation.

Ce travail d’exploitation est donc encore en cours, mais une fois que ces problèmes auront été contournés, nous avons bon espoir de pouvoir générer un modèle 3D du parking à partir des captures LIDAR.

# Chapitre 7

## Perspectives

Ce chapitre présente les perspectives d'évolution qui pourront être appliquées au projet Symeter V2.

### 7.1 Amélioration de l'utilisation de l'IMU

Comme indiqué dans les chapitres précédents, le processus de localisation n'exploite pas encore pleinement les données d'accélération de l'IMU. L'intégration de ces données dans le `ekf_localization_node` s'avère complexe et requiert une procédure d'exploitation particulière.

L'une des pistes à explorer sera d'intégrer un module INS ("Inertial Navigation System", système de navigation inertiel) qui prendra en compte uniquement les données IMU pour générer son estimation propre de la pose. Cette pose serait ensuite soumise au `ekf_localization_node` pour être fusionnée avec les informations GPS. Cela permettrait de séparer complètement la gestion de l'IMU de la fusion de données, et donc d'y voir plus clair quand à l'état de la localisation.

### 7.2 Développement d'un module d'accumulation propre à Symeter V2

Si le module `octomap` a permis de monter le prototype initial de Symeter V2 pour la simulation sous Gazebo, et de démontrer la possibilité de faire l'acquisition d'un modèle numérique 3D d'un silo à partir d'un LIDAR mobile, ce module comporte quelques caractéristiques qui font qu'il n'est pas complètement adapté aux besoins de Symeter V2.

Notamment, le nuage de point généré par `octomap` représente une information de présence de donnée dans un Voxel de 10x10cm, et le point généré est le barycentre du cube du voxel. Nous perdons donc de la précision dans la mesure d'altitude du tas d'ensilage.

Ceci nous forcera à développer un module d'accumulation et de stockage du modèle numérique 3D qui soit complètement adapté aux besoins de Symeter V2.

## **7.3 Simulation de l'action de tassage**

La simulation peut permettre la mise au point de la localisation, de l'acquisition initiale du chantier d'ensilage. Mais elle n'est pas directement utilisable pour simuler l'opération du tracteur dans le chantier d'ensilage. En effet il n'est pas possible de simuler physiquement le tassage à l'aide de Gazebo.

Il est cependant possible de mettre en oeuvre nos outils de manière à simuler logiquement l'évolution du chantier en implémentant un mécanisme de sauvegarde / restauration de l'état du chantier d'une session de fonctionnement à l'autre. En faisant varier la forme du tas d'ensilage simulé entre deux sessions, il sera possible d'émuler une action de tassage.

## **7.4 Développement des outils de détection de monitoring de tassage**

Une fois les procédés d'acquisition d'un modèle numérique 3D à partir d'un LIDAR mobile finalisés, nous pourrons commencer à mettre en place le monitoring du tassage lui-même.

Ce monitoring devra comparer les modifications du modèle numérique au fur-et-à mesure de la progression du chantier, enregistrer l'évolution au cours du temps du silo, générer les données d'aide à la décision.

Une interface homme/machine permettant de piloter l'intervention du système Symeter V2 sur le chantier devra aussi être développée.

# Chapitre 8

## Conclusion

Au cours de ce stage, nous avons pu confirmer que les outils de simulation proposés par les environnements ROS et Gazebo pouvait permettre le développement du produit Symeter V2, qualitativement et quantitativement.

Nous avons pu implémenter une plateforme de test simulée sous Gazebo et l'environnement physique virtuel associé, permettant ainsi d'émuler le chantier d'ensilage, le tracteur et le système Symeter V2 monté sur celui-ci.

Nous avons mis en place le système de localisation en mettant en oeuvre un composant pré-existant de fusion de capteurs, `ekf_robot_localization`, et ainsi exploiter des signaux de l'IMU et du GPS. Ceci nous permet d'obtenir la pose du véhicule avec une précision compatible avec un reconstitution qualitative du chantier simulé. Nous avons aussi développé les scripts permettant de vérifier le bon fonctionnement de ce procédé.

Nous avons mis en place la chaîne d'acquisition des données Lidar en utilisant principalement les composants disponibles dans Point Cloud Library. Nous avons prétraité ces données en convertissant le format angles/distances en un nuage de points préliminaire, et nous avons filtré ce nuage de points de manière à répartir régulièrement les points de mesure et réduire le flot de données à traiter.

Nous avons enfin mis en oeuvre le composant `OctoMap` pour effectuer l'accumulation des données LIDAR acquises. Nous générions ainsi un nuage de points 3D représentant notre chantier d'ensilage simulé. Nous validons ainsi la faisabilité de Symeter V2.

Nous avons aussi tenté d'utiliser le prototype de Symeter V2 pour exploiter des captures réelles. Ce travail est encore en cours.

Nous avons pu dégager des axes d'amélioration pour augmenter la précision de la localisation, obtenir un nuage de points représentant plus fidèlement le chantier. L'objectif de ces améliorations est d'accéder à un niveau de qualité du procédé qui permette des mesures précises du tassage et du volume.

Nous avons donc à l'issue de ce stage une plateforme fondamentale de simulation qui va permettre à Tellus Environment de développer les fonctionnalités de mesures du tassage de silo et de mesure de volume du système Symeter V2.

Cette plateforme simulée va aussi permettre à Tellus Environment de tester virtuellement diverses configurations de capteurs et ainsi permettre de mieux définir le contour du produit. Cela permet à Tellus Environment d'envisager que les premiers tests en grandeur du système Symeter V2, au cours de vrais chantiers d'ensilage, se dérouleront de manière productive.

L'avenir du projet Symeter V2 est en cours de détermination par ses commanditaires, avec une forte probabilité de continuer pour au moins un an supplémentaire. Cette année sera consacrée au montage d'un prototype fonctionnel et de la conduite des tests en grandeur. Tellus Environment m'a proposé de continuer l'aventure, et nous sommes tombés d'accord pour mener à bien ensemble ce projet dans le cadre d'un CDD d'un an.

# Annexe A

## Représentation de l'Attitude et des Rotations

Selon [Gus15], si nous considérons un véhicule mesurant des mouvements par rapport à son propre repère et un autre système mesurant les positions et vitesses etc du même véhicule par rapport à un repère  $\{e\}$ , et si la navigation est effectuée par rapport à un repère local de navigation  $n$ , il devient très vite apparent qu'un outil pour décrire l'orientation du véhicule par rapport à ces différents repères est nécessaire.

Deux des manières les plus communes de représenter les attitudes et les rotations dans  $\mathbb{R}^3$  sont les *Angles d'Euler* et les *quaternions*.

### A.1 Angles d'Euler

Selon [Nü09], alors que les trois coordonnées Cartesiennes  $x, y$  et  $z$  représentent la position, les trois angles  $\theta_x, \theta_y$  et  $\theta_z$  décrivent l'orientation dans  $\mathbb{R}^3$ . L'union de la position et de l'orientation est appelée la *pose*. L'orientation  $\theta_x, \theta_y$  et  $\theta_z$  est la rotation autour des axes principaux du repère orthonormé, c'est à dire  $(1, 0, 0)$ ,  $(0, 1, 0)$  et  $(0, 0, 1)$ . Les matrices de rotation sont les suivantes :

$$\begin{aligned}\mathbf{R}_x &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x \\ 0 & \sin \theta_x & \cos \theta_x \end{pmatrix} \\ \mathbf{R}_y &= \begin{pmatrix} \cos \theta_y & 0 & \sin \theta_y \\ 0 & 1 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y \end{pmatrix} \\ \mathbf{R}_z &= \begin{pmatrix} \cos \theta_z & -\sin \theta_z & 0 \\ \sin \theta_z & \cos \theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}\end{aligned}$$

La matrice de rotation globale est calculée comme  $\mathbf{R} = \mathbf{R}_{\theta_x, \theta_y, \theta_z} = \mathbf{R}_x \mathbf{R}_y \mathbf{R}_z$ .

Note : La matrice ci-dessus dépend de l'ordre de la multiplication. Des matrices de rotation différentes sont obtenues par  $\mathbf{R} = \mathbf{R}_{\theta_z, \theta_y, \theta_x} = \mathbf{R}_z \mathbf{R}_y \mathbf{R}_x$  ou  $\mathbf{R} = \mathbf{R}_{\theta_y, \theta_x, \theta_z} = \mathbf{R}_y \mathbf{R}_x \mathbf{R}_z$ . De plus un blocage de cardan peut intervenir si les axes

## A.2 Quaternions Unitaires

Toujours selon [Nü09], les quaternions sont un nombre complexe à 4 dimensions qui peuvent être utilisés pour représenter des rotations en 3D. Ils sont générés en postulant une racine additionnelle à -1 nommée  $j$  qui ne soit pas  $i$  ou  $-i$ . Il existe alors nécessairement un élément  $k$  tel que  $ij = k$  qui aussi une racine de -1. Les relations suivantes sont aussi vraies :

$$\begin{aligned} i^2 &= j^2 = k^2 = ijk = -1 \\ ij &= k, \quad ji = -k \\ jk &= i, \quad kj = -i \\ ki &= j, \quad ik = -j \end{aligned}$$

$i, j, k$  et -1 forment la base du quaternion  $\mathbb{H}$ . Tout quaternion  $\dot{\mathbf{q}}$  peut être représenté sous la forme  $\dot{\mathbf{q}} = q_0 + q_x i + q_y j + q_z k$ . Ainsi tout quaternion définit un unique 4-vecteur  $(q_0, q_x, q_y, q_z)^T \in \mathbb{R}^4$ .

Comme pour les nombres complexes, les quaternions possèdent un unique quaternion conjugué. Le quaternion conjugué de  $\dot{\mathbf{q}} = q_0 + q_x i + q_y j + q_z k$  est  $\dot{\mathbf{q}}^* = q_0 - q_x i - q_y j - q_z k$ .

En utilisant le quaternion conjugué et le produit scalaire, nous pouvons calculer la norme du quaternion  $\dot{\mathbf{q}}$  :

$$\|\dot{\mathbf{q}}\| = \sqrt{\langle \dot{\mathbf{q}}, \dot{\mathbf{q}}^* \rangle} = \sqrt{q_0^2 + q_x^2 + q_y^2 + q_z^2}$$

Le sous-ensemble des quaternions unité est important car ses éléments peuvent représenter les rotations dans  $\mathbb{R}^3$ . Un quaternion unité  $\dot{\mathbf{q}}$  vérifie la relation  $\|\dot{\mathbf{q}}\| = 1$ . Un quaternion unité a pour propriété que son inverse  $\dot{\mathbf{q}}^{-1}$  est égal à son conjugué  $\dot{\mathbf{q}}^*$ .

$$\langle \dot{\mathbf{q}}, \dot{\mathbf{q}}^* \rangle = \|\dot{\mathbf{q}}\|^2 = 1$$

Les rotations utilisant les quaternions sont formalisées comme suit : soit  $\dot{\mathbf{q}}$  tel que  $\dot{\mathbf{q}} = (q_0, q_x, q_y, q_z) = (q_0, \mathbf{q})$ . Pour effectuer la rotation du point 3D  $\mathbf{p} = (p_x, p_y, p_z)^T \in \mathbb{R}^3$ , nous écrivons ce point sous la forme d'un quaternion aussi :  $\dot{\mathbf{p}} = (0, p_x, p_y, p_z)^T = (0, \mathbf{p})$ . Alors nous pouvons démontrer que le point  $p_{\text{rot}}$  est le résultat d'une rotation du point  $\mathbf{p}$  par la formule

$$\mathbf{p}_{\text{rot}} = \dot{\mathbf{q}} \dot{\mathbf{p}} \dot{\mathbf{q}}^*$$

[Nü09] montre qu'une rotation d'un angle  $\theta$  autour d'un vecteur  $\mathbf{n}$  sera représenté par le quaternion

$$\dot{\mathbf{q}} = \left( \cos \frac{\theta}{2}, \sin \frac{\theta}{2} \mathbf{n} \right)$$

Pour passer d'une représentation en quaternion en une représentation de rotation autour d'un axe, il suffit de calculer les composantes suivantes :

$$\theta = 2 \arccos q_0$$

$$n_x = \frac{q_x}{\sin \theta}$$

$$n_y = \frac{q_y}{\sin \theta}$$

$$n_z = \frac{q_z}{\sin \theta}$$

# Bibliographie

- [Gus15] Kenneth Gustavsson. UAV Pose Estimation using Sensor Fusion of Inertial, Sonar and Satellite Signals, 2015.
- [HWB<sup>+</sup>13] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap : an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3) :189–206, April 2013.
- [Jef10] Charles Jeffrey. *An introduction to GNSS : GPS, GLONASS, Galileo and other Global Navigation Satellite Systems*. NovAtel, Calgary, 2010. OCLC : 1036065024.
- [KH06] Elliott D. Kaplan and C. Hegarty, editors. *Understanding GPS : principles and applications*. Artech House mobile communications series. Artech House, Boston, 2nd ed edition, 2006. OCLC : ocm62128065.
- [MF13] Aaron Martinez and Enrique Fernández. *Learning ROS for robotics programming : a practical, instructive, and comprehensive guide to introduce yourself to ROS, the top-notch, leading robotics framework*. Community experience distilled. Packt Publ, Birmingham, 2013. OCLC : 861540246.
- [ML16] Carlos Moreno and Ming Li. A Comparative Study of Filtering Methods for Point Clouds in Real-Time Video Streaming. page 5, 2016.
- [MS16] Thomas Moore and Daniel Stouch. A Generalized Extended Kalman Filter Implementation for the Robot Operating System. In Emanuele Menegatti, Nathan Michael, Karsten Berns, and Hiroaki Yamaguchi, editors, *Intelligent Autonomous Systems 13*, volume 302, pages 335–348. Springer International Publishing, Cham, 2016.
- [NKG13] Aboelmagd Noureldin, Tashfeen B. Karamat, and Jacques Georgy. *Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [Nü09] András Nüchter. *3D Robotic Mapping : The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*. Springer, 2009.
- [RS12] TAMAKI RS. Scanning Laser Range Finder UTM-30lx-EW Specification. page 7, December 2012.
- [Ste14] Gerald Steinbauer. TEDUSAR White Book - State of the Art in Search and Rescue Robots, 2014.

- [SXZ17] Shashi Shekhar, Hui Xiong, and Xun Zhou, editors. *Encyclopedia of GIS*. Springer International Publishing, Cham, 2017.
- [YSA<sup>+</sup>13] O. Yalcin, A. Sayar, O.F. Arar, S. Akpinar, and S. Kosunalp. Approaches of Road Boundary and Obstacle Detection Using LIDAR. *IFAC Proceedings Volumes*, 46(25) :211–215, 2013.
- [ZM09] Paul Zarchan and Howard Musoff. *Fundamentals of Kalman filtering : a practical approach*. Number v. 232 in Progress in astronautics and aeronautics. American Institute of Aeronautics and Astronautics, Reston, Va, 3rd ed edition, 2009. OCLC : ocn457170744.