

# Travail de Bachelor

## Robot agricole autonome

**Non confidentiel**

**Image / photo**  
**(Facultatif)**

<b>Étudiant :</b>	<b>Arzur Gabriel CATEL TORRES</b>
<b>Travail proposé par :</b>	Etienne Messerli REDS Adresse NPA Ville
<b>Enseignant responsable :</b>	<b>Etienne Messerli</b>
<b>Année académique</b>	2019-2020

Yverdon-les-Bains, le 3 juillet 2020

## Travail de Bachelor 2019-2020

### Robot agricole autonome

---

REDS

#### Résumé publiable

### Enoncé

Nous souhaitons réaliser un robot d'arrosage de plantons dans le but de soutenir une agriculture écologique et durable. Durant les premières semaines, les plantons sont très sensibles à un manque d'eau par le fait d'un enracinement très faible (motte de culture de 4 x 4 cm). Ce stress peut engendrer des pertes de production assez conséquente. Un précédent projet a permis d'identifier différentes problématiques majeures à résoudre. Dès lors le projet a été scindé en plusieurs parties, soit : le déplacement autonome du robot, la détection des plantons et la conception mécanique. Nous avons aussi identifié d'autres fonctionnalités intéressantes pour un robot autonome. Si nous connaissons avec précision l'emplacement des plantons, il est possible de réaliser un désherbage mécanique par le passage d'une griffe entre ceux-ci. Une autre fonctionnalité serait de planter les plantons avec précision et de mémoriser leur position.

L'objectif de ce projet est de réaliser une commande embarquée permettant un déplacement autonome du robot dans un champ. Nous pouvons déjà spécifier que la commande doit permettre au robot de se placer au début d'une ligne de plantation et de pouvoir aller vers un point de remplissage ou de charge. Nous souhaitons aussi pouvoir cartographier la position des plantons avec une précision du centimètre. Il est imaginé que le système de commande sera basé sur une plateforme embarquée Linux, qui dispose de système de géolocalisation (GPS, plateforme inertielle, ...), de moyen de communication et de possibilité de stockage de position ou de déplacements. Le projet comprendra le choix, l'adaptation et la mise en œuvre d'un prototype afin de réaliser des essais réels dans un champ.

Pour les aspects de géolocalisation et de positionnement nous allons collaborer avec le professeur Sébastien Guillaume de l'institut INSIT.

### Cahier des charges

Le but du projet est de réaliser une commande embarquée permettant un déplacement autonome du robot dans un champ. Nous pouvons identifier les étapes suivantes :

Choix du matériel pour réaliser un démonstrateur du robot autonome. Cela comprend le choix d'une plateforme mécanique, d'une carte embarquée et des différents composants pour faire fonctionner l'ensemble.

Choix et utilisation d'un environnement permettant la gestion d'un robot autonome. A priori, le choix se portera sur l'environnement ROS (Robot Operating System). Celui-ci est basé sur l'OS Linux.

Choix des capteurs permettant le positionnement du robot dans un champ. Il est supposé que le positionnement soit assuré par une composition de systèmes d'acquisitions comme par exemple un GPS, une centrale inertielle, etc.

Choix et développement d'une cartographie du champ. Il faudra faire un choix qui soit le plus standard possible afin d'assurer l'évolution du logiciel.

Développement et réalisation du logiciel de commande du déplacement autonome du robot. Cela comprendra l'intégration des différents systèmes d'acquisition, la programmation de séquence de déplacement et la possibilité d'aller à des points fixes.

Intégration des développements sur le démonstrateur du robot autonome. Celui-ci comprendra une commande manuelle sans fils et un écran tactile pour faciliter le développement et les tests.

Réalisation de tests de déplacements autonomes. Qualification de la fiabilité des déplacements, de la réaction lors d'objet mobile et de mesurer de la précision du positionnement.

Étudiant :

Catel Torres Arzur Gabriel

Date et lieu :

.....

Signature :

.....

Enseignant responsable

Date et lieu :

.....

Signature :

.....

Messerli Etienne

Nom de l'entreprise/institution

Date et lieu :

.....

Signature :

.....

Nom prénom de la personne  
confiant l'étude

## Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en **Ingénierie / Economie d'entreprise**.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Nom du doyen

Chef du Département TIC

Yverdon-les-Bains, le 3 juillet 2020

# Authentification

Le soussigné, Arzur Gabriel Catel Torres, atteste par la présente avoir réalisé seul ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Begnins, le 3 juillet 2020

Arzur Gabriel Catel Torres

## Table des matières

Enoncé .....	1
Cahier des charges.....	1
1 Introduction .....	8
1.1 Contexte du projet.....	8
1.2 Etat initial du projet.....	8
1.3 Structure du rapport.....	9
2 Analyse globale .....	10
2.1 Informations à disposition.....	10
2.2 Robot de test .....	10
2.3 Les ressources disponibles en robotique .....	10
2.4 Techniques pour déterminer la position d'un robot .....	11
2.4.1 Odométrie.....	11
2.4.2 Tracking de la puissance des moteurs .....	12
2.4.3 Système GNSS.....	12
2.4.4 Système inertiel .....	13
2.4.5 Correction RTK.....	13
2.4.6 Choix du système pour la navigation.....	14
3 La robotique et ROS .....	15
3.1 Qu'est-ce que ROS ? .....	15
3.2 ROS, un poids important dans les choix .....	15
3.3 Compatibilité de ROS avec les OS.....	15
4 Choix des composants du robot .....	16
4.1 Définir le matériel nécessaire .....	16
4.1.1 Système de déplacement .....	16
4.1.2 Choix du matériel pour la base .....	16
4.2 Carte électronique .....	18
4.2.1 Raspberry pi 3 .....	18
4.2.2 Raspberry pi 4 .....	19
4.2.3 Tinker Board .....	19
4.2.4 Choix de la carte .....	20
4.3 Drivers moteurs .....	20
4.3.1 Roboclaw 2x30A .....	20
4.3.2 HB-25 Parallax.....	21
4.3.3 Cytron 5V-30V DC Motor Driver .....	21
4.3.4 Choix des drivers pour les moteurs .....	22
4.4 Alimentation .....	22

4.5	I2c to PWM .....	23
4.6	Touchscreen.....	24
4.7	Scanner laser .....	24
4.8	IMU / GNSS.....	25
4.9	Caméra.....	25
4.10	Contrôleur.....	26
4.11	Ajout d'une alimentation à part pour la Raspberry Pi 4.....	26
5	Montage du robot.....	28
5.1	Etage 1 .....	28
5.2	Etage 2 .....	29
5.3	Etage 3 .....	31
5.4	Connectique précise .....	32
5.5	Schéma global du robot.....	33
6	Outils logiciel.....	34
6.1	OS.....	34
6.2	ROS .....	34
6.2.1	Langages supportés .....	34
6.2.2	Système de fichier .....	35
6.2.3	Notions de base de ROS (définitions) .....	35
6.3	Catkin.....	35
7	Robot contrôlé manuellement .....	37
7.1	Développement sur la Donkey Car .....	37
7.1.1	Package i2cPWM .....	37
7.1.2	Package joy .....	38
7.1.3	Package ps4-ros .....	39
7.1.4	Package donkey_car .....	39
7.1.5	Schéma des nodes .....	<b>Erreur ! Signet non défini.</b>
7.2	Développement sur le robot démonstrateur .....	40
7.2.1	Fonctionnement des drivers moteurs .....	40
7.2.2	Package diff_drive .....	41
7.2.3	Package robot_demo.....	41
8	Intégration des capteurs dans ROS.....	43
8.1	RPLidar A1.....	43
8.2	Mti-7 DK (IMU /GNSS) .....	44
8.3	D435i Realsense.....	47
9	Système autonome .....	48
9.1	Quelques définitions.....	49

9.1.1	Les frames.....	49
9.1.2	TF package .....	50
9.1.3	Orientation « Quaternion » .....	50
9.1.4	Position d'un robot .....	50
9.1.5	Recovery process .....	51
9.1.6	Clearing process.....	51
9.2	La navigation autonome dans ROS.....	51
9.2.1	Navigation « map-based » .....	51
9.2.2	Navigation réactive .....	51
9.3	Définir les entrées de la Nav Stack .....	51
9.3.1	Odométrie.....	51
9.3.2	Map.....	53
9.3.3	Nuage de points.....	54
9.3.4	Laser Scan .....	54
9.3.5	Algorithme AMCL.....	54
9.4	Conclusion du système autonome .....	55
10	Création d'un GUI .....	56
10.1.1	LVGL.....	56
10.1.2	Kivy.....	57
10.1.3	TKinter .....	57
10.1.4	Pygame .....	58
11	Démarrage automatique .....	59
11.1.1	Init.d.....	59
11.1.2	Systemd .....	59
11.1.3	Rc.local.....	60
11.1.4	Améliorations du lancement du GUI .....	60
12	Conclusion.....	61
12.1	Synthèse .....	61
12.2	Futur du projet.....	61
12.3	Commentaires personnels.....	61
12.4	Remerciements.....	62
13	Bibliographie .....	64
14	Documents supplémentaire .....	66



# 1 Introduction

## 1.1 Contexte du projet

De nos jours, un des plus gros défis de l'humanité est de conserver une terre vivable pour notre génération et pour les suivantes. Pour y parvenir il est impératif de modifier les manières de consommer et de produire de la société.

En tant que discipline scientifique, l'écologie permet de mieux comprendre comment les êtres vivants vivent et interagissent au sein d'un milieu. En tant qu'idée politique et sociale, l'écologie a pour objectif de protéger les écosystèmes, la biodiversité, et l'environnement en général, notamment afin de permettre aux sociétés d'y vivre avec résilience et de façon pérenne. C'est dans ce cadre que ce travail de Bachelor s'inscrit.

L'objectif final du robot est de tendre vers une production plus écologique dans le monde de l'agriculture. Son objectif premier est d'assurer l'arrosage de plantons durant leurs premières semaines de croissance. Ces derniers sont très sensibles à un manque d'eau. Les plantons sont petits et leurs racines peu profondes. De fait, la surface qu'il est utile d'arroser est petite comparée à la taille d'un champ. L'utilisation d'un arroseur classique (automatique ou manuel) permettant d'arroser le champ dans sa totalité est donc une solution peu écologique. Développer un système ciblant précisément les surfaces sensibles et importantes permet de réduire la consommation d'eau sans pour autant détériorer les produits.

Une fois le robot mis en place, il n'est pas impossible (voir intéressant) de réfléchir à des utilisations autre, toujours dans le domaine de l'agriculture, comme par exemple le désherbage ou même la cueillette. Une fois un robot stable développé, toutes les adaptations pour diverses tâches sont envisageables.

La complexité d'un système robotique poussé tel que celui-ci implique de séparer les différents traitements que le robot doit être capable d'effectuer. Le sujet de ce travail de bachelor concerne le déplacement autonome d'un robot dans un champs. C'est donc sur cette partie que la réflexion, la recherche et le développement est dirigé.

Une importance particulière doit être portée aux autres parties à développer en parallèle ou dans un futur proche. Il faut prendre en compte que le reste des fonctionnalités prévoit d'être implémenté, ce n'est pas une partie stand alone. Ce sont des problématiques concernant le matériel (taille, place, branchement...) mais aussi la partie logiciel (charge du processeur, évolution du programme, généricité...).

La mécanique du projet finale est menée par d'autres étudiants dans le cadre de travaux de bachelors dédiés.

Enfin concernant l'alimentation du système, le projet s'inscrivant dans le cadre de l'écologie, il serait intéressant qu'il existe la possibilité de mettre en place une alternative d'alimentation écologique à base de panneaux solaires pour recharger les batteries du robot. Cette réflexion ne s'inscrit pas dans le cadre de ce travail de Bachelor mais c'est une des problématiques du projet final à prendre en compte.

## 1.2 Etat initial du projet

Ce projet a débuté quelques années auparavant et des travaux de Bachelor d'élèves des années précédentes ont déjà soulevés certains problèmes et certaines limites auxquelles il est nécessaire de faire face pour obtenir un système capable de remplir les contraintes du cahier des charges.

Un premier robot a été créé et plusieurs tests concernant le déplacement et de traitement d'image ont été réalisés. Ce robot a montré certaines limites, l'objectif du travail de recherche mené est de proposer un robot plus évolué et plus performant en se basant sur les résultats du premier système.

Le projet mené n'a cependant jamais mis en place un système de déplacement autonome, il est donc nécessaire d'effectuer une analyse et de trouver une solution sans aucune base.

### 1.3 Structure du rapport

Le rapport se décompose en plusieurs parties :

**Chapitre 2 : Analyse globale** : analyse de la problématique et les solutions apportées pour y répondre. Nous trouverons dans ce chapitre les différentes informations et ressources à disposition, et nous listerons les technologies qui permettent de répondre au cahier des charges.

**Chapitre 3 : La robotique et ROS** : explication de l'importance de ROS dans le domaine de la robotique. Explication globale de ROS et conséquences sur le projet.

**Chapitre 4 : Choix des composants du robot** : analyse des avantages et inconvénients des différents composants pour chaque partie du robot et choix de ceux-ci.

**Chapitre 5 : Montage du robot** : présentation des différentes couches du robot et de l'agencement de celui-ci. Des schémas du câblage du matériel sont disponibles pour comprendre le système dans son ensemble. Explications des choix du placement du matériel suivant leur utilité / accessibilité.

**Chapitre 6 : Outils logiciel** : présentation et explication des outils logiciels principaux utilisés pour le développement du robot. Explication de ROS et de son système de build notamment.

**Chapitre 7 Robot contrôlé manuellement** : explication de la mise en place du logiciel permettant de contrôler manuellement le robot. Ce chapitre explique aussi le travail sur les deux robots différents et les adaptations nécessaires pour le fonctionnement du robot démonstrateur une fois celui-ci suffisamment avancé dans le montage.

**Chapitre 8 : Intégration des capteurs dans ROS** : explication de l'intégration des capteurs choisis dans ROS. Une preuve du fonctionnement et de la justesse des informations reçues sera fournie dans la mesure du possible.

**Chapitre 9 : Système autonome** : détails concernant le fonctionnement du système autonome mis en place. Le but de ce chapitre est de comprendre le fonctionnement concret du package mis à disposition par ROS pour arriver à une navigation efficace et fiable.

**Chapitre 10 : Création d'un GUI** : explication concernant le choix du framework utilisé pour le GUI et de sa mise en place dans le système.

**Chapitre 11 : Démarrage automatique** : détails concernant le démarrage automatique du système. Explication des choix concernant les technologies utilisées pour la résolution du problème et améliorations futures possibles.

## 2 Analyse globale

### 2.1 Informations à disposition

Les spécifications du projet ne sont pas des plus prolifiques concernant la navigation et le positionnement. La liberté est totale concernant le développement du projet qui s'inscrit pleinement dans le domaine de la recherche. L'objectif est de trouver des solutions permettant de réaliser un système capable de se positionner au centimètre près dans un champ de plusieurs hectares. Un défi d'envergure pour un système qui se veut « low cost ».

Les travaux réalisés sur ce sujet les années précédentes sont consultables et représentent des sources d'informations précieuses pour les décisions à prendre concernant la direction générale à prendre et pour éviter de reproduire des erreurs ou d'éviter de se heurter aux mêmes limites d'un système.

De plus, d'autres projets de robotiques menés au REDS (drone autonome) sont à disposition concernant l'aspect logiciel, traitement des données et problématique de déplacement. Le sujet d'étude est différent mais certaines contraintes et analyses sont similaires dans la mise en place d'un robot (détection des obstacles, stratégie d'esquive, correction de la direction...) et d'autres diffèrent suivant le cadre du projet (intérieur/extérieur, précision voulue...).

Enfin, la participation à un concours de robotique (Eurobot) nous fournit déjà de nombreuses bases solides pour appréhender correctement l'organisation et la mise en place du logiciel permettant le bon fonctionnement du système. Les sources et le développement mené sur le robot du concours sont consultables et permettent de gagner du temps sur les solutions parfois peu triviales de problématiques complexes.

### 2.2 Robot de test

Pour développer un logiciel permettant le déplacement autonome, il est indispensable d'avoir un robot qui puisse se déplacer. Se contenter des simulations au sein d'environnements virtuels n'est pas suffisant. Il existe des logiciels et interfaces permettant de simuler le comportement d'un robot mais il est indispensable de se confronter aux problématiques matérielles qui vont survenir tout au long du développement.

Le premier objectif est le développement d'un robot comportant les parties nécessaires pour le déplacement autonome. Malgré un travail logiciel non négligeable à réaliser (analyse et conception), une partie importante de mécanique et électronique fera l'objet de recherche, d'analyse et de prise de décision. Il est important de bien définir au préalable les composants nécessaires au développement. Ainsi, le choix du système entier à tester doit être défini rapidement et correctement analysé pour éviter des problèmes de timing sur les commandes de matériel.

Dans un deuxième temps, la partie logicielle sera elle aussi analysée pour déterminer la structure du programme, le traitement et la communication des informations fournies par les différents composants et enfin définir le comportement du robot d'après toutes les situations qu'il peut rencontrer dans un champ.

### 2.3 Les ressources disponibles en robotique

De nombreux systèmes existent aujourd'hui (robot ménager, drones...). Des recherches poussées ont été menées pour répondre aux contraintes générales de la robotique et à des problèmes plus spécifiques à un projet. Un point indispensable est de se renseigner sur les réponses apportées et jusqu'à quel point cette solution permet de répondre aux contraintes de notre propre cahier des charges.

Il en est de même pour le matériel. S'inspirer de structures existantes et fonctionnelles permet de gagner du temps et d'éviter des problèmes de compatibilités entre les composants. Le temps est limité pour un projet de cette envergure et réduire le nombre de problèmes potentiels améliore les chances de succès.

ROS (Robot Operating System) est aujourd'hui un outil reconnu dans le domaine de la robotique. De nombreux projets ont pour base ROS et la communauté est vaste et active. Un système de navigation autonome est présent et répond à de nombreuses problématiques (esquive d'obstacles, navigation vers un objectif...). Cet outil est aujourd'hui incontournable lorsque l'on parle de développement de robot. Il sera expliqué plus en détail dans un chapitre dédié.

## 2.4 Techniques pour déterminer la position d'un robot

De nombreuses méthodes existent pour déterminer la position d'un système autonome. Le problème principal est de cerner les contraintes correctement pour analyser au mieux les besoins et les possibilités.

### Les contraintes :

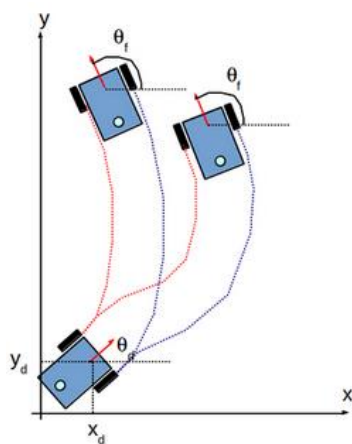
- Un champ est un lieu extérieur.
- Le sol d'un champ est imparfait dans sa topographie (pas parfaitement plat).
- Le cahier des charges demande une précision au centimètre.
- Le robot doit pouvoir se recharger seul (énergie et outils).
- Le système doit être low cost.

Différents systèmes permettant la navigation autonome :

#### 2.4.1 Odométrie

L'odométrie est une technique permettant d'estimer la position d'un véhicule en mouvement.

Cette technique repose sur la mesure individuelle des déplacements des roues pour reconstituer le mouvement global du robot. En partant d'une position initiale connue et en intégrant les déplacements mesurés, on peut ainsi calculer à chaque instant la position courante du véhicule.



Voici un exemple d'utilisation de l'odométrie pour déterminer le déplacement d'un véhicule.

Cet exemple montre que la distance parcourue par chacune des roues est identique, mais la position finale du véhicule est différente. Il est donc nécessaire d'effectuer des traitements supplémentaires pour gérer les courbes.

Cependant si le système ne se déplace qu'en ligne droite et tourne sur lui-même, ce problème n'apparaît plus, ce qui est possible concernant le robot développé dans le travail de recherche mené.

Figure 1 : Exemple d'odométrie faussée

Il existe différentes manières de gérer l'odométrie qu'il est important de comparer. Cependant des capteurs permettant de récupérer directement l'odométrie ne sont pas prévus. Il est possible de la générer de différentes manières (visuel, algorithmes de prédiction...).

La possibilité la plus envisageable est cependant d'ajouter une roue supplémentaire (au milieu du véhicule) qui ne permet pas au véhicule de se déplacer mais dont l'objectif est uniquement de fournir des données de vitesse de rotation pour en déterminer l'odométrie.

#### 2.4.2 Tracking de la puissance des moteurs

Une alternative à l'odométrie est de déterminer un déplacement d'après la puissance des moteurs. En effet, pour qu'un robot avance tout droit, il faut que la puissance envoyée aux roues soit équivalente (éventuellement une gestion de calibration à mettre en place au préalable).

Avec quelques calculs et tests empiriques, il est possible de déterminer la distance  $d$  parcourue par un véhicule en un temps  $t$  d'après la puissance  $p$  des moteurs.

Il est possible d'ajouter des systèmes sur les moteurs pour détecter la vitesse de rotation. Cependant, cela nécessite une électronique de mesure des courants qui est coûteuse et la gestion d'un tel système peut être complexe.

Cette stratégie n'est pas retenue pour le projet, la précision est une contrainte importante et la calibration est complexe et doit être faite de manière très minutieuse pour avoir une précision suffisante.

#### 2.4.3 Système GNSS

Système de positionnement par satellites.

Un système de positionnement par satellites fournit sur un récepteur les coordonnées géographiques en trois dimensions (longitude, latitude, hauteur ellipsoïdale), la vitesse de déplacement et la date / heure à son utilisateur. Ces informations sont calculées à partir des mesures de distance à un instant donné entre le récepteur de l'utilisateur et plusieurs satellites artificiels dont les positions dans l'espace sont connues avec précision.

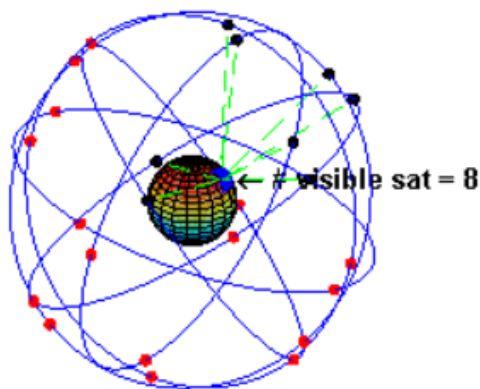


Figure 2 : GNSS schéma constellation/visibilité

En combinant la mesure simultanée de la distance d'au moins quatre satellites, le récepteur est capable par multilatération de fournir la position et l'altitude avec une précision de l'ordre du mètre, la vitesse avec une précision de quelques cm/s et le temps avec une précision atomique.

Cette image montre qu'au point bleu à ce moment précis, il est possible d'avoir au maximum 8 satellites visibles (ce nombre varie suivant l'état de la constellation des satellites et du point de référence). En prenant en compte les bâtiments et les obstacles possibles, ce nombre diminue rapidement, d'où l'importance d'avoir un surplus de satellites par point potentiel permettant de compenser cette problématique.

Le GPS n'est pas utilisable en intérieur et une des contraintes principales est d'avoir un environnement suffisamment dégagé pour assurer le minimum de satellites nécessaires.

La précision obtenue n'est pas satisfaisante.

Les données obtenues utilisées pour un robot au sol se limitent à la longitude et la latitude. Ces informations permettent de créer avec plusieurs points un espace sphérique et pour retranscrire ces informations sur une carte « plate » en utilisant les algorithmes spécifiques à cette conversion.

De nombreux systèmes de GNSS existent. Les principaux sont :

Nom du système GNSS	Pays
GPS	USA
GLONASS	Russie
Galileo	Europe
BeiDou	Chine
QZSS	Japon
IRNSS	Inde

#### 2.4.4 Système inertiel

Une centrale inertielle est un ensemble de capteurs permettant de mesurer le mouvement.

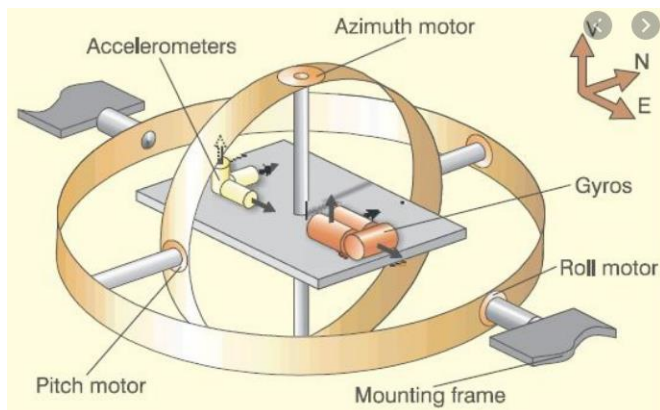


Figure 3: Schéma centrale inertielle

L'accélération du mobile par rapport à un référentiel galiléen est mesurable grâce à des accéléromètres. Il en faut trois, un pour chaque axe de l'espace.

Dans une centrale « à plateforme stabilisée », les accéléromètres sont stabilisés par des gyroscopes, ce qui permet de maintenir les capteurs alignés avec les axes Nord-Sud, Ouest-Est et la verticale.

Dans une centrale « strap-down », les accéléromètres sont fixes par rapport au véhicule, mais la vitesse de rotation est mesurée par trois Gyromètres, le traitement de signal permet ensuite de faire les changements de repères.

Ces mesures ont l'avantage de ne pas dépendre de sources extérieures. Elles ont de plus un rafraîchissement de plusieurs centaines de hertz, nécessaire à la fonction de pilotage. Toutefois, à cause des problèmes de dérive (dégradation de la précision au fil du temps), les données peuvent être corrigées par un recalage avec une source complémentaire, par exemple un système de positionnement par satellites.

#### 2.4.5 Correction RTK

Le Real Time Kinematic (RTK) est une technique utilisée pour améliorer la précision d'un signal GNSS en utilisant une station fixe de référence qui envoie des corrections à un récepteur en mouvement.

En utilisant ces corrections, le système GNSS peut fixer la position de l'antenne à 1 ou 2 cm près (en théorie et suivant le matériel).

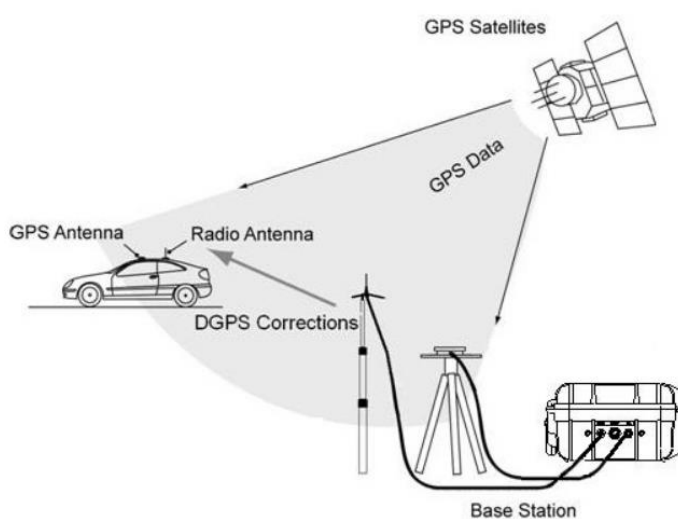


Figure 4 : Schéma correction RTK

La correction RTK se définit comme étant un système d'augmentation de la précision. On parle de SBAS, de correction RTK ou encore de correction PPK.

Le principe de la méthode différentielle est de corriger la position (longitude, latitude, hauteur) du robot d'après une erreur calculée à partir d'un point connu.

Le point connu est une station dont la position exacte est connue.

#### 2.4.6 *Choix du système pour la navigation*

Le système le plus cohérent par rapport aux nécessités et aux contraintes (environnement, cahier des charges et temps pour le projet) et le plus viable, c'est le GNSS. Même seul, il permet de naviguer en extérieur. Cependant, il est intéressant de le coupler avec une IMU pour obtenir une précision plus intéressante dans le cadre du projet.

Des plateformes inertiels avec correction GPS existent, il n'y a donc pas besoin de la composer soi-même. La fréquence du GPS est de l'ordre de 1Hz tandis que la technologie inertielle permet de travailler à plusieurs centaines de Hz.

Enfin, l'odométrie permet d'avoir une base intéressante concernant les déplacements du robot. Il est intéressant de la récupérer et d'utiliser les autres données des capteurs pour corriger les erreurs.

Avec un GNSS basique, la précision est de l'ordre du mètre mais peut être améliorée avec la technologie RTK.

Pour conclure, le plus optimal est d'avoir une combinaison de plusieurs systèmes pour obtenir la précision la plus satisfaisante possible. Il faut tout de même aussi prendre en compte que le système ne doit pas non plus être surchargé uniquement avec la récupération et le traitement des données du déplacement autonome. La balance est fragile et des tests doivent impérativement être prévus.



## 3 La robotique et ROS

### 3.1 Qu'est-ce que ROS ?

Robot Operating System (ROS) est une plateforme de développement logicielle pour robot. Il s'agit d'un méta-système d'exploitation qui peut fonctionner sur un ou plusieurs ordinateurs et qui fournit plusieurs fonctionnalités telles que :

- Abstraction du matériel.
- Contrôle des périphériques de bas niveau.
- Mise en œuvre de fonctionnalités couramment utilisées.
- Transmission de messages entre les processus et gestions des packages installés.

Avant les OS robotiques, chaque concepteur de robot, chaque chercheur en robotique passait un temps non négligeable à concevoir matériellement son robot ainsi que le logiciel embarqué associé. Cela demandait des compétences en mécanique, électronique et programmation embarquée. Généralement, les programmes ainsi conçus correspondaient plus à de la programmation très bas niveau, proche de l'électronique, qu'à de la robotique proprement dite, telle que nous pouvons la rencontrer aujourd'hui dans la robotique de service. La réutilisation des programmes était non triviale car fortement liée au matériel sous-jacent.

L'idée principale d'un OS robotique est d'avoir un outil proposant des fonctionnalités standardisées faisant abstraction du matériel, tout comme un OS classique pour PC, d'où l'analogie de nom.

### 3.2 ROS, un poids important dans les choix

L'utilisation de ce Framework est absolument cruciale pour le choix de la carte car il n'est pas compatible avec tous les OS. L'OS pour lequel il est principalement développé est « Ubuntu ».

Ce framework va aussi impacter le choix de certains composants. En effet, la communauté qui développe sur ROS propose de nombreux packages permettant d'intégrer facilement certains composants dans l'environnement. C'est vers ces composants qu'il faut se diriger pour composer le robot pour éviter de dédier trop de temps pour l'intégration d'un certain matériel dans le système développé.

### 3.3 Compatibilité de ROS avec les OS

Cet outil ne fonctionne actuellement qu'avec des plateformes basées sur Linux. Les tests et les développements sont fait sur Ubuntu et Mac OS X.

ROS n'est pas fonctionnel uniquement sur Ubuntu. Certaines personnes ont développé la possibilité d'utiliser ROS avec Raspbian, Fedora, Arch Linux ou sur des microcontrôleurs par exemple. Il existe malgré tout des limites notamment concernant l'utilisation avec un microcontrôleur, qui ne permet pas d'utiliser tous les outils existants (certains outils de communication synchrones (entre les threads) ne sont pas disponibles). La conséquence est que l'utilisation des packages créés par les utilisateurs va, dans certains cas, être délicate et il sera nécessaire d'adapter le code pour l'utiliser.

Raspbian a cependant une version de ROS plutôt fonctionnelle et stable. Malgré tout, des problèmes surviennent dans certains cas et la résolution de ceux-ci n'est pas toujours évidente. Un autre point négatif est que la mise à jour de l'OS est parfois indispensable, mais cette dernière peut compromettre le fonctionnement de ROS et les modifications faites pour que le framework fonctionne.

Idéalement pour éviter au maximum les problèmes (et éviter une baisse des performances) avec ROS, il est recommandé d'avoir Ubuntu comme système d'exploitation.



## 4 Choix des composants du robot

### 4.1 Définir le matériel nécessaire

Un robot peut se déplacer de différentes manières. Pour commencer, il est important d'analyser les besoins et les contraintes pour définir la base la plus cohérente et la plus adaptée.

Le but du projet est de mettre en place un système permettant de naviguer de manière autonome dans un champ.

#### 4.1.1 *Système de déplacement*

Un robot avec une plateforme rectangulaire et des roues est une base cohérente pour les besoins du projet.

Cependant il existe trois possibilités courantes pour la gestion du déplacement avec des roues :

1. Ackerman : un moteur permettant de contrôler la vitesse de déplacement du robot (avant / arrière) et un autre moteur permettant de tourner les roues (avant en général). C'est le système utilisé pour les véhicules (taille réel ou robot véhicule) en général.
2. Différentiel : des moteurs pour chaque roue permettant de les contrôler de manière complètement indépendante les unes des autres. Pour tourner, il suffit d'avoir les roues d'un côté qui vont dans un sens et les roues de l'autre côté qui vont dans le sens opposé. Cela permet notamment d'effectuer des virages à rayon nul.
3. Mecanum : les roues mecanum, ou roues omnidirectionnelles, permettent à un véhicule de se diriger dans toutes les directions, aussi bien sur le côté que vers l'avant et l'arrière. Chaque roue (4 minimum) est motorisée, commandée en vitesse chacune indépendamment des autres, en fonction du mouvement à produire. Elles sont placées généralement en position évoquant un rectangle. Les axes principaux de rotation des roues sont parallèles.

#### 4.1.2 *Choix du matériel pour la base*

Le châssis doit être suffisamment spacieux pour contenir tous les composants souhaités. Pour cela, une liste du matériel potentiel à utiliser doit être déterminée en amont.

La liste comporte tous les éléments présents sur le robot destinés au développement du système autonome.

Les composants sont les suivants :

- Moteurs, drivers moteurs et batterie.
- Carte électronique ou microcontrôleur.
- Système inertiel / GNSS.
- Caméra.
- Capteur de profondeur (scanner rotatif).
- Voltage converter.
- Board pour envoyer les PWM.
- Ecran tactile.

Tous ces composants et leur taille ne nécessitent pas une plateforme trop imposante. L'ensemble tient sans problème sur une plateforme de 30 x 30 cm en ajoutant des étages.

Une plateforme avec la base existe :



Figure 5 : base du robot (Prowler)

Cette base est suffisante pour mettre en place un système autonome tel qu'il a été défini au préalable.

Les roues de 13.7cm permettent de rouler dans un champ pour faire des tests de déplacement autonome avec la plateforme. Le châssis de 22.8cm x 25.4cm donne suffisamment de liberté pour ajouter des composants tels qu'une caméra, des cartes supplémentaires, des capteurs...

Cette base comporte les roues, le châssis et les 4 moteurs de ce type :



Figure 6 : moteurs DC

Voltage (Nominal)	12V
Voltage Range (Recommended)	6V - 12V
Speed (No Load)*	313 rpm
Current (No Load)*	0.52A
Current (Stall)*	20A
Torque (Stall)*	416.6 oz-in (30 kgf-cm)
Gear Ratio	27:1
Gear Material	brass primary, nylon secondary, steel tertiary
Gearbox Style	Planetary
Motor Type	DC
Motor Brush Type	Graphite
Output Shaft Diameter	6mm (0.236")
Output Shaft Style	D-shaft
Output Shaft Support	Dual Ball Bearings
Electrical Connection	Male Spade Terminal
Operating Temperature	-10°C ~ +60°C
Mounting Screw Size	M3 x 0.5mm
Product Weight	330g (11.64oz)

Les spécifications des moteurs sont indispensables pour déterminer les drivers nécessaires à leur gestion.

La base est complète, il faut à présent passer au choix des composants électroniques présents sur le système. Les choix seront faits en partie d'après le matériel déjà disponible dans l'institut REDS.

## 4.2 Carte électronique

Pour contrôler un robot, il est possible de le faire depuis un microcontrôleur simple ou avec une board un peu plus évoluée capable de supporter un OS (tel que les Raspberry pi, banana pi, tinker...).

Ce projet est complexe, se doit d'être low cost et le temps pour le réaliser est court. Le microcontrôleur est une option très intéressante car il consomme beaucoup moins que l'autre système, ce qui est un gros fort puisque ce projet s'inscrit dans une optique écologique. Cependant, la complexité impose d'avoir un OS qui puisse tourner et permettre d'implémenter des options de manière plus pratique.

Plusieurs cartes sont à disposition :

1. Raspberry pi 3
2. Raspberry pi 4
3. Tinker Board

Ces trois choix sont les plus cohérents pour le projet et chaque choix possède des avantages et des inconvénients à analyser dans l'optique de choisir correctement cet élément.

Un aspect très important pour le choix de la carte pour le projet est la compatibilité et la possibilité d'intégrer et de faire fonctionner ROS dessus.

### 4.2.1 Raspberry pi 3



Figure 7 : Raspberry pi 3

Cette carte est la plus intéressante en termes de ressources et de développement sur internet. Elle est parfaitement compatible avec les dernières versions d'Ubuntu et donc avec ROS.

C'est un choix sûr et une solution viable pour le développement du projet. Aucune prise de risque avec cette carte.

L'institut REDS a de plus beaucoup travaillé dessus (projet SOO, SO3...) et les ingénieurs familiers avec cette carte sont nombreux.

Cependant la puissance de calcul risque d'être insuffisante dans le cadre du projet final. En effet, le système autonome avec les divers capteurs ne posera pas de problèmes mais ajouter des communications externes et un traitement d'image pourrait engendrer une surcharge du processeur.

Pour cela, il serait possible de déléguer le traitement des données ou le calcul de certains algorithmes du système à un système externe. Cela impliquerait donc que le système soit dépendant d'un autre et nécessiterait donc la mise en place d'un système de communication fiable et sécurisé.

#### 4.2.2 Raspberry pi 4



Figure 8 : Raspberry pi 4

Version supérieure de la Raspberry pi 3 :

- USB 3 très intéressant pour le matériel et la vitesse de transfert des données.
- Globalement plus de puissance (plus de mémoire, processeur plus performant).

Le seul point négatif est que cette carte est très récente et ce n'est que depuis peu qu'Ubuntu est portable correctement sur cette dernière. Il y a encore peu de temps, la seule façon de porter Ubuntu était d'adapter à la main les problèmes de compatibilité rencontrés. Aujourd'hui, une version officielle est disponible mais certains problèmes surviennent encore lors du développement. En effet, résoudre des problèmes de portage d'OS n'est pas toujours trivial et requiert du temps.

De nombreuses fonctionnalités sont similaires à la version précédente mais certains aspects ont été modifiés (dans le but d'améliorer les performances), par exemple le wifi et le Bluetooth sont implémentés différemment.

C'est malgré ce défaut un choix très intéressant surtout au niveau des performances qu'elle propose.

#### 4.2.3 Tinker Board



Figure 9 : tinker board

Cette carte en termes de performances est très proche de la Raspberry pi 3 (légèrement plus puissante).

Une version d'Ubuntu est disponible pour la Tinker, ce qui la rend éligible.

Un gros point négatif est la taille de la communauté utilisant cette carte. Elle est très limitée donc le développement et les mises à jour sont très rares. Peu de personnes répondent sur les discussions dans les forums. Lorsqu'un problème survient, il est bien plus difficile de trouver une solution qu'avec les cartes précédemment présentées.

Cependant un point positif important est qu'un ancien élève a développé un robot avec la tinker board et ROS. Donc un système stable avec un développement logiciel sur ROS est disponible (sources consultables).

#### 4.2.4 Choix de la carte

Le choix final se porte sur la Raspberry pi 4.

La version antérieure semblait être un choix idéal mais le gain en puissance de calcul n'est pas négligeable. De plus, les développeurs auront tendance à se diriger vers cette carte dans les projets futurs et les ressources vont donc constamment croître. Cette carte posera peut-être quelques problèmes (interfaçage de matériel, compatibilité avec d'anciens composants...) dans un premier temps mais sera bien plus intéressante d'ici peu au vu de l'activité et de l'intérêt que porte la communauté à cette carte dans le domaine de la robotique et des systèmes embarqués en général.

Pour un développement futur, aucune hésitation c'est cette carte qui a le potentiel le plus prometteur.

### 4.3 Drivers moteurs

Pour contrôler les moteurs, le choix s'est porté sur l'utilisation de valeurs PWM. C'est un choix fait d'après le développement du robot précédent, mais aussi du fait que le développement du robot de ce projet a commencé avec la Donkey Car, qui utilise aussi ce système d'envoi de valeurs PWM pour la gestion des moteurs.

Les moteurs reçoivent du courant. Les valeurs PWM doivent donc être « traduites » en courant, qui doit être envoyé aux moteurs. Ce sont des moteurs DC, le courant est donc continu.

En considérant que des valeurs PWM sont effectivement envoyées à chaque moteur indépendamment, il est indispensable d'avoir des drivers moteurs pour récupérer la valeur et la convertir en courant. Plusieurs drivers moteurs existent et un choix s'impose.

Pour choisir lequel est le plus adapté, tout comme pour la carte, il est nécessaire d'analyser les avantages et inconvénients des différents composants trouvés :

#### 4.3.1 Roboclaw 2x30A

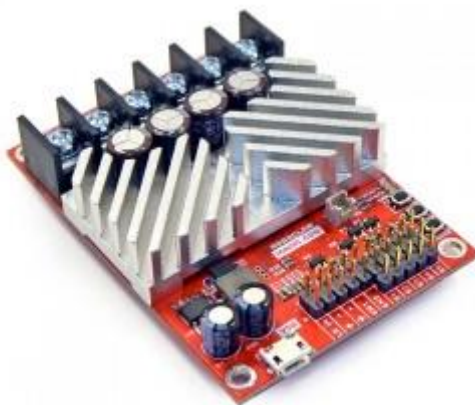


Figure 10 : roboclaw 2x30A

Avantages	Inconvénients
<ul style="list-style-type: none"><li>- Conseillé par le fournisseur des moteurs</li><li>- Facile d'utilisation</li><li>- Très bonne documentation</li><li>- Contrôle deux moteurs</li></ul>	<ul style="list-style-type: none"><li>- Coûteux</li><li>- Trop puissant pour nos besoins</li><li>- Beaucoup d'intelligence « inutile » de base</li><li>- Nombreuses fonctionnalités peu utiles</li></ul>

#### 4.3.2 HB-25 Parallax



Figure 11 : HB-25 Parallax

Avantages	Inconvénients
<ul style="list-style-type: none"> <li>- Utilisé dans le travail de diplôme précédent</li> <li>- Facile d'utilisation</li> <li>- Aucun traitement</li> </ul>	<ul style="list-style-type: none"> <li>- Indisponible en vente</li> </ul>

#### 4.3.3 Cytron 5V-30V DC Motor Driver



Figure 12 : Cytron MD10C

Avantages	Inconvénients
<ul style="list-style-type: none"> <li>- Peu coûteux</li> <li>- Petit</li> <li>- Simple à utiliser</li> </ul>	<ul style="list-style-type: none"> <li>- Inconnu, pas de ressources/avis au sein du REDS</li> <li>- Ne peut pas tenir les moteurs à pleine puissance</li> </ul>



#### 4.3.4 Choix des drivers pour les moteurs

Le driver des moteurs est important mais tous ceux proposés permettent de remplir la fonction demandée, le choix se porte donc sur le plus simple et le moins coûteux : Cytron 5V-30V DC Motor Driver.

Ce composant devrait d'après la documentation et les spécifications parfaitement subvenir aux besoins du projet.

Le fait de ne pas pouvoir garder le maximum de la puissance des moteurs en continu n'est pas un problème, le système n'est pas basé sur la vitesse de déplacement mais sur la précision. Les déplacements sont plutôt lents et permettent aussi de ne pas soumettre le robot à des consommations d'énergie trop importantes.

### 4.4 Alimentation

C'est un robot pour tester le déplacement autonome. Pour ce système, il est intéressant d'avoir une batterie qui puisse tenir au minimum 1 heure en alimentant les moteurs, les drivers des moteurs et la Raspberry pi 4.

Les moteurs sont alimentés avec une puissance de 12V, il faut donc une batterie 12V.

Mon choix s'est donc porté pour être large sur une batterie 12V et 6000 mAh :



Figure 13 : batterie Li-ion

Cependant la Raspberry pi 4 est alimentée en 5V, la batterie ne convient pas telle quelle. Il est nécessaire d'ajouter un voltage converter pour transformer les 12V en 5V et que la Raspberry soit correctement alimentée.

Un élément important à noter est que la Raspberry pi 4 ne dispose d'aucun système de gestion du courant reçu. Elle ne subit aucun dommage si elle est alimentée entre 4.6 et 5.7 Volts. Ces valeurs sont relativement souples et permettent de convertir avec une certaine marge d'erreur non compromettante l'énergie fournie par la batterie.



Matériel : Step Down Board Module Buck

Ce matériel permet de convertir 9-38 V en 5V. Il n'est pas réglable mais ce n'est pas utile pour ce projet.

Figure 14 : voltage converter

Avec ce système, l'alimentation du de la Raspberry pi 4 ne risque pas d'endommager le matériel.

## 4.5 I2c to PWM

Si le système doit envoyer des commandes PWM aux différents moteurs, il est important que ces valeurs soient envoyées à une fréquence suffisamment élevée pour que l'envoi du courant après interprétation des valeurs soit correct. Le maintien de cette fréquence est problématique pour la Raspberry pi 4. En effet, le système va s'encombrer avec l'envoi constant de ces valeurs et ce temps de calcul ne sera pas disponible pour les autres fonctionnalités.

Pour soulager le système, il existe du matériel permettant de s'occuper de l'envoi des valeurs PWM avec une fréquence suffisante.

Cette carte possède une mémoire avec des registres permettant de stocker les valeurs de PWM à envoyer à la fréquence souhaitée. La Raspberry pi 4 n'aura donc plus qu'à modifier les registres lorsqu'un changement de vitesse d'un ou plusieurs moteurs est souhaité.

Le composant choisi pour le projet est dû à la situation exceptionnelle du déroulement de ce semestre qui nous a contraint à réaliser un premier développement avec un robot véhicule (Donkey Car) déjà entièrement monté. Le but était de faciliter le développement du travail de recherche en gardant le même composant déjà compris et fonctionnel d'après l'architecture logicielle développée.

La carte utilisée par le robot AROSSA utilise le même composant, mais avec un PCB différent (Adafruit). Le fonctionnement est cependant le même, seule une légère différence de la fréquence maximum possible est à noter (plus élevée sur l'Adafruit).

Ainsi, c'est ce composant i2c PWM qui a été choisie : PCA9685.



Figure 15 : PCA9685

- 16 canaux PWM 12 bits
- i2c
- 5V compatible

Aucune comparaison n'a été faite pour ce matériel mais une analyse entre les différents i2c to PWM aurait montré que tous sont finalement très similaires que ce soit dans la structure du PCB ou bien dans la façon d'utiliser les registres.

Ce matériel est en général utilisé pour la gestion de servos-moteurs. Mais avec des drivers moteurs spécifiques, la récupération des données en PWM n'est pas un obstacle et permet aussi de gérer des moteurs DC.

Cependant, la seule limite posant un problème est la fréquence maximum d'envoi des données se limitant à 1kHz. En général, un moteur DC fonctionne dans des fréquences comprises entre 10 à 30 KHz. Ce n'est pas un problème pour le développement d'une solution dans le cadre du TB permettant le déplacement autonome, mais c'est une contrainte à prendre en compte pour le projet final.



## 4.6 Touchscreen

Le choix du touchscreen s'est principalement porté sur sa compatibilité avec la Raspberry pi 4 et avec l'OS Ubuntu. Ce composant n'est présent que pour le développement du projet et ne sera pas forcément utilisé (ou pas de la même façon) pour le projet final.

Après avoir cherché un touchscreen d'après ces critères, c'est sur celui-ci que notre choix s'est porté :



Figure 16 : touchscreen

Ce touchscreen fonctionne avec un HDMI (câble micro HDMI / HDMI pour la Raspberry pi 4) et avec un USB pour la partie alimentation et touchscreen (détection d'événements).

## 4.7 Scanner laser

Concernant ce composant, un scanner rotatif était disponible à l'école. Il n'y a pas eu besoin d'en commander un.

Le but de ce scanner est de détecter les obstacles en extérieur pour qu'une politique d'esquive des objets soit possible.

Le scanner disponible est le suivant : RPlidar A1 :



Figure 17 : rplidar A1

- Plage d'activité de 12 mètres (rayon de 6 mètres)
- Mesure par triangulation laser
- Balayage de la plage laser omnidirectionnelle à 360 degrés
- Mesure les données de distance dans plus de 8000 fois / s
- Taux de balayage configurable de 2-10Hz

Ce composant est un plug and play compatible avec la Raspberry pi 4 et interfaçable facilement avec ROS. La configuration est simple et rapide et les données suffisamment précises pour être utilisées pour la navigation d'un robot autonome.

## 4.8 IMU / GNSS

Après une discussion avec une équipe d'ingénieurs travaillant régulièrement avec ce type de matériel et des recherches personnelles, un kit de développement était la solution la plus pratique pour la mise en place d'un robot. Ce type de matériel coûte cher (plus la précision est grande, plus le matériel est coûteux) et les différences de prix sont très élevées. En effet, les kits varient de 300 Francs environ à plus de 10'000 Francs.

Dans le cadre d'un robot agricole low cost, il n'est pas envisageable que le prix ne soit déterminé que par ce composant qui le rendrait peu intéressant sur le marché. Ainsi, c'est un des kits low cost qui a été choisi : mti-7 DK de Xsens :



Figure 18 : mti-7 DK (IMU/GNSS)

- Peut se brancher en USB, SPI, UART
- IMU
- GNSS (avec antenne)
- Bonne documentation
- Communauté active

Les IMU/GNSS low cost sont très utilisés dans le développement de robots autonomes. Les informations permettent de renforcer la précision de localisation du robot avec différents algorithmes traitant les données et fusionnant les résultats.

## 4.9 Caméra

Même principe que pour le scanner 360. Une caméra performante était disponible au REDS. Cette caméra est la D435i de RealSense.



Figure 19 : D435i (caméra RealSense)

- RGB résolution : 1920 x 1080 pixels
- Depth résolution (profondeur) : 1280 x 720 pixels
- Connection USB
- IMU intégrée
- Jusqu'à 90 FPS

Cette caméra est particulièrement adaptée à la navigation autonome des robots et pour la reconnaissance d'objets. Le champ de vision conséquent de la caméra permet d'avoir plus de surface couverte.

C'est une caméra stéréoscopique (permet d'obtenir le relief avec la combinaison de deux images représentant l'œil gauche et l'œil droit) avec des capteurs de profondeurs. Cela permet d'avoir une fonction de perception spatiale intéressante pour la navigation d'un robot (plutôt en intérieur).

## 4.10 Contrôleur

Les premiers pas pour appréhender correctement et comprendre le fonctionnement du système est de passer par le contrôle du robot créé avec un joystick.

Le contrôleur utilisé est le dualshock 4 (manette playstation 4) car il est facile à trouver, est utilisable avec le Bluetooth et de nombreux projets de robotique proposent le contrôle via ce matériel (véhicule RC):



Figure 20 : ds4 controller

## 4.11 Ajout d'une alimentation à part pour la Raspberry Pi 4

Après de nombreux tests avec le matériel commandé, la Raspberry Pi 4 se retrouvait parfois sous-alimentée et le système redémarrait. Pour comprendre d'où venait le problème, il a fallu analyser le matériel et comment se comportait le courant lors de l'activation des 4 moteurs.

Pour commencer, le voltage convertisseur choisi permet de convertir des valeurs comprises entre 9 et 38 Volts en fournissant 5 Volts en sortie. La batterie fournit 12 Volts en sortie, c'est donc suffisant.

Cependant, lorsque les 4 moteurs s'activent ensemble, le système redémarre certaines fois.

Après avoir analysé les chutes de tensions provoquées par l'activation des moteurs à l'oscilloscope, voici ce qui est observable :

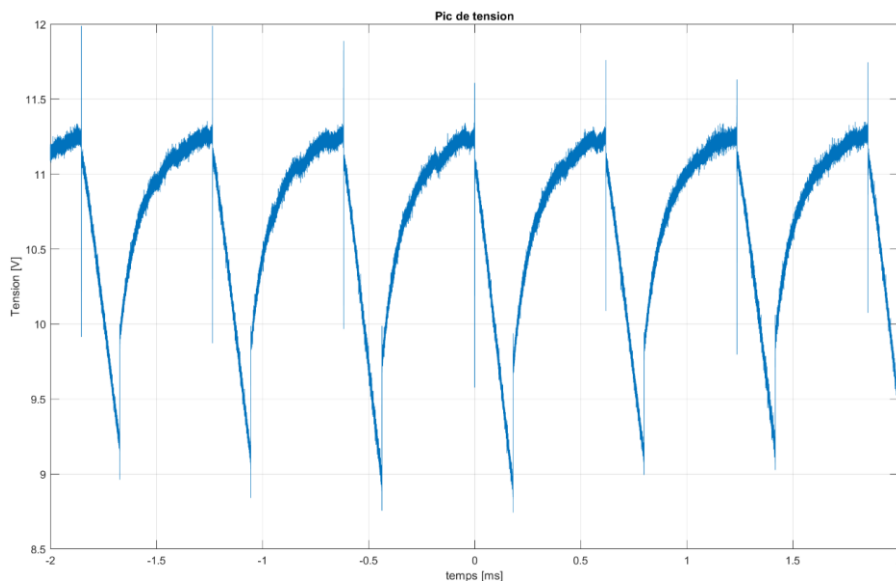


Figure 21 : graphique chute de tension

Voici l'évolution de la tension de la batterie dans le temps lorsqu'une accélération du véhicule est demandée. La tension s'approche dangereusement de la valeur minimum acceptée par le voltage converter choisi... Dans certains cas, la tension sera plus basse et la pi se retrouvera donc non-alimentée pendant un court laps de temps, mais suffisant pour redémarrer le système dans certains cas.

Visiblement, le système actuel n'est pas résistant à des chutes de tensions de ce type. La solution la plus élégante dans ce cas est d'utiliser une batterie supplémentaire pour alimenter la Raspberry Pi 4. C'est une solution intéressante car il sera peut-être nécessaire d'ajouter une Raspberry Pi 4 supplémentaire au système pour assurer la puissance de calcul nécessaire au traitement d'image.

Le composant choisi est donc la batterie Aukey PB-N52 :



*Figure 22 : batterie supplémentaire Aukey PB-N52*

La Raspberry Pi 4 demande une alimentation entre 4.6 et 5.7 Volts pour fonctionner. Ce type de batterie pour téléphone fonctionne sans problème pour ce système embarqué.

## 5 Montage du robot

Le montage du robot prend en compte de nombreux aspects :

- Facilité d'accès à certaines parties pour le développement.
- Changement de la batterie possible (2 batteries en alternance).
- Laser rotatif le plus dégagé possible.
- Touchscreen visible.
- Antenne GPS la plus exposée possible.
- Élégance visuelle du système (si possible).

Ce n'est pas le robot final, ce système n'est monté que pour tester la partie logicielle (pour ce travail le déplacement autonome et par la suite le traitement d'image).

La plateforme mesurant 22.8 x 25.4, il est impossible que tout puisse être fixé sur la seule plateforme fournie. Il est impératif d'ajouter des étages pour que tous les composants puissent tenir.

Pour cela, de nouvelles plaques de mêmes dimensions ont été créées puis fixées par l'atelier mécanique. Ces plaques sont en bois dans un premier temps (question de temps / coût) mais seront dans un matériau similaire à la plaque de base fournie dans le démonstrateur final.

L'agencement proposé se divise en trois étages :

### 5.1 Etage 1

Cet étage est difficilement accessible puisque deux étages seront au-dessus. Il est indispensable d'y mettre les composants auxquels l'accès n'est pas souvent requis lors du développement du système et des essais.

Les moteurs doivent être sur cet étage puisqu'ils sont interfacés aux roues. Les drivers moteurs alimentent les moteurs et peuvent être proches.

La batterie est aussi sur cet étage mais doit être montée de sorte qu'il soit possible de la retirer facilement tout en étant suffisamment stable pour ne pas bouger lors des déplacements du robot.

L'intérêt d'une telle mise en place est d'alterner entre plusieurs batteries pour que la démo puisse être la plus lisse possible et sans interruptions de fonctionnement.

Pour faciliter le câblage d'alimentation depuis la batterie, l'ajout d'un module permettant d'avoir 6 sorties (ou plus) est possible au vu de la place qu'il reste sur cet étage.

Enfin, la place est suffisante pour y ajouter le composant qui envoie les PWM aux différents moteurs. Avoir le PCA9685 sur cet étage permet surtout de se limiter aux câbles de l'I2C connectés depuis la Raspberry pi 4 pour communiquer.

Si le PCA9685 se trouve sur l'étage supérieur près de la Raspberry pi 4, cela implique que tous les câbles des PWM allant vers les différents drivers moteurs sont plus longs et traversent verticalement le système. Pour une simplicité et une élégance de câblage, c'est à cet étage qu'il est le plus intéressant de placer ce composant.

Voici un schéma permettant de visualiser l'organisation finale avec les dimensions des différents composants :

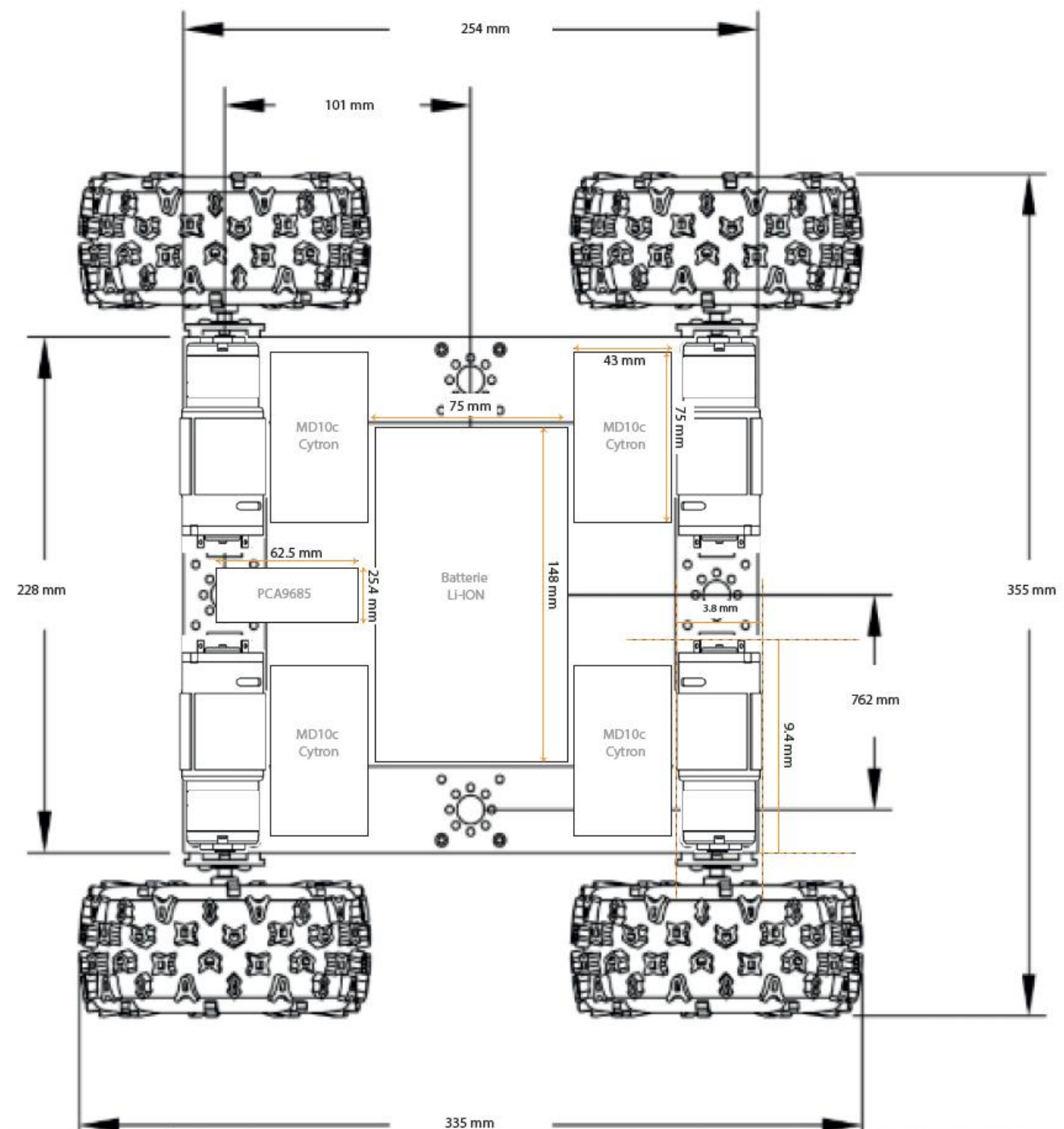


Figure 23 : agencement étage 1

## 5.2 Etage 2

Cet étage contient le laser 360 RPLidar. Il est seul sur son étage pour dégager le plus possible sa vision et renforcer son champ d'action et donc sa précision. Le premier objectif est de trouver un moyen pour supporter l'étage supérieur en ayant la plus petite réduction de visibilité possible.

La solution utilisée est la même que le robot « RosBot » développé par l'entreprise Willow : quatre réglettes les plus fines possible avec un angle correspondant à l'angle du faisceau du Lidar pour augmenter la longueur et assurer un maintien de l'étage avec du matériel dessus tout en gardant la surface minimum de « gêne ».

Ci-dessous le schéma explicatif :

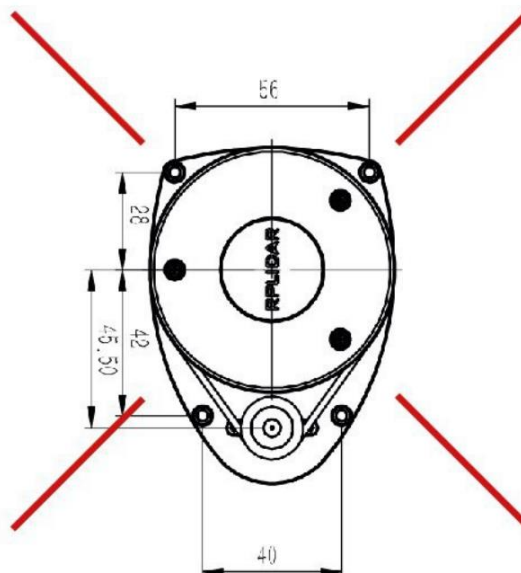


Figure 24 : fixation du Lidar

Les réglettes de soutien (en rouge) sont d'une épaisseur de 2mm.

Pour déterminer la hauteur à laquelle l'étage sera le plus optimisé, il faut prendre en compte que les roues ne doivent pas se trouver dans le champ d'activité du laser.

Le robot doit naviguer en extérieur, des saletés (herbe, terre...) peuvent se trouver dans les roues, prendre une marge pour compenser ces impuretés est nécessaire.

Pour commencer, voici les mesures pour obtenir la hauteur du faisceau par rapport à la plaque sur laquelle est fixé le lidar :

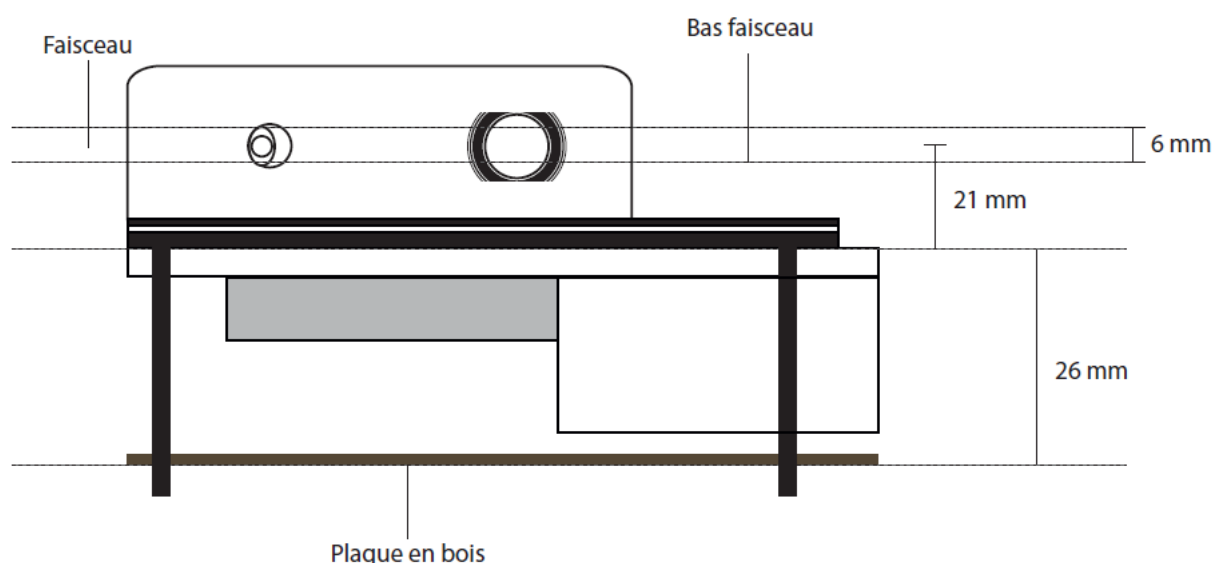


Figure 25 : mesures utiles du Lidar

Le milieu du faisceau se trouve à 47mm de la base. La taille du faisceau est de 6mm, le bas du faisceau est donc à :  $47 - 6 / 2 = 44\text{mm}$ .

A présent, voici les mesures de la hauteur des roues par rapport aux fixations en métal :

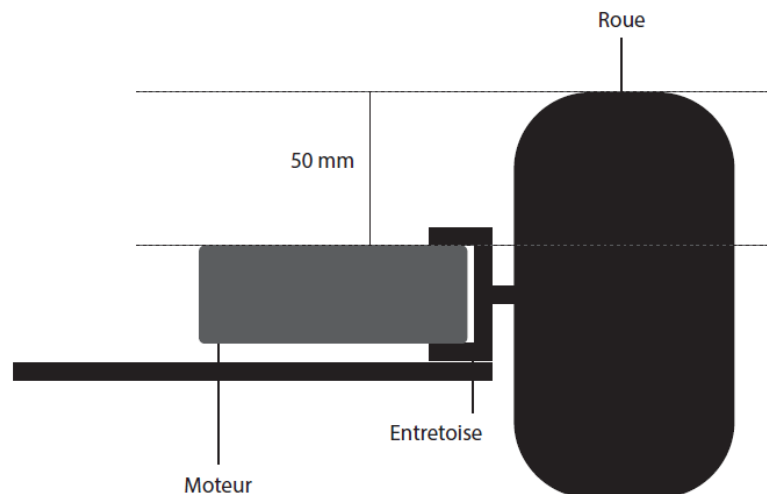


Figure 26 : mesures châssis - Fixation Lidar

D'après les mesures réalisées, il faudrait une entretoise de 6mm pour être tout juste au niveau du haut de la roue. Avec 10mm environ de marge pour ne pas scanner des impuretés, une taille de 15mm est suffisante.

### 5.3 Etage 3

Enfin l'étage le plus haut comportera tout le reste des composants. Notamment la Raspberry pi 4 et sa batterie 5V, le mti-7DK, l'écran tactile et la caméra.

Cet étage sera complété lorsque le robot démonstrateur sera finalisé.



## 5.4 Connectique précise

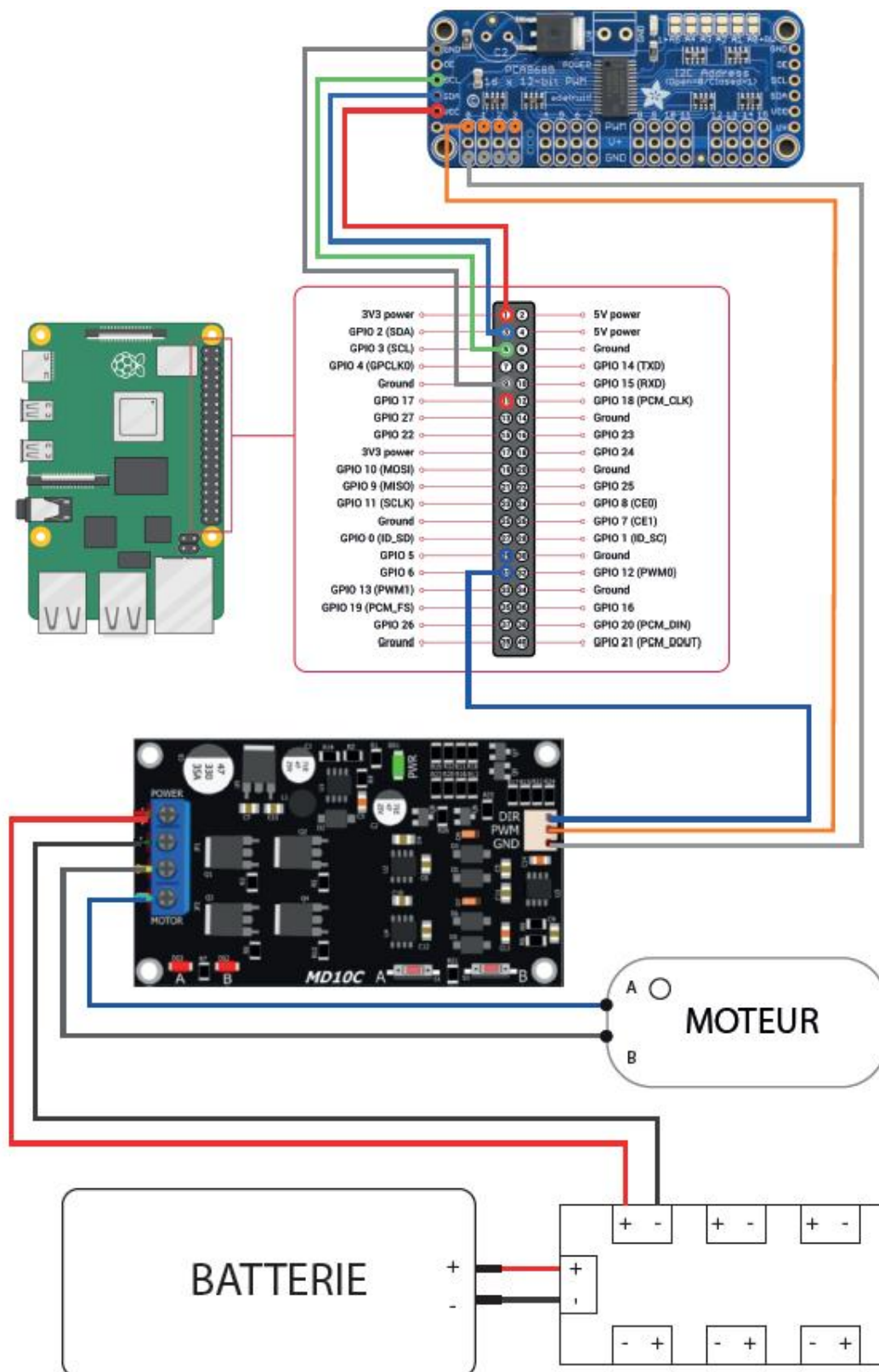


Figure 27 : Schéma de connexion électronique

Pour obtenir le schéma complet et le montage final, se référer aux annexes (A[3]).

Sur ce schéma ne sont montrés les câblages que des éléments pour le fonctionnement basique du système : moteurs, drivers moteurs, batterie, multiplexage de la batterie, board i2c to PWM et Raspberry pi 4.

L'objectif est la reproductibilité du câblage en montrant les liaisons à effectuer.

Il est important d'utiliser les GPIO 5 et 6 de la Raspberry Pi 4 pour la gestion des directions des drivers moteurs. C'est avec le logiciel que la direction est gérée et la direction dépend de la valeur du GPIO. Plus de détails dans les annexes (A[1]).

## 5.5 Schéma global du robot

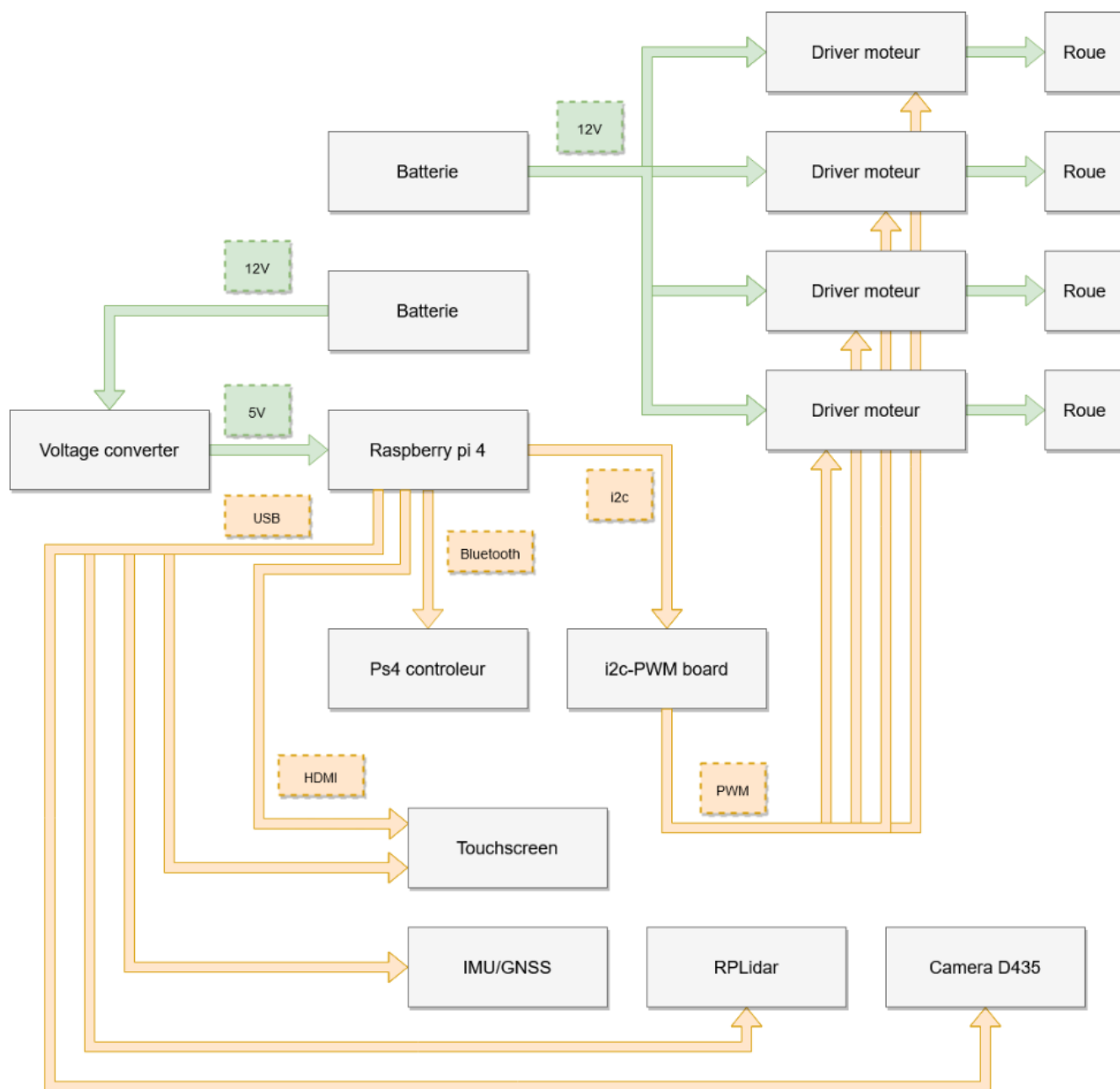


Figure 28 : liaisons système complet

## 6 Outils logiciel

### 6.1 OS

Le choix de l'OS a été défini dans un chapitre précédent. Le but est d'utiliser Ubuntu pour la synergie optimale avec ROS.

Cependant, de nombreuses versions d'Ubuntu existent et il est nécessaire d'en déterminer une.

Pour commencer, ce projet n'as pas besoin de Desktop. C'est donc une version uniquement navigable via des terminaux qui sera utilisée.

La spécificité d'Ubuntu est qu'il existe de nombreuses versions (aujourd'hui la version la plus récente est la 20.04) mais qui ne sont pas toutes supportées dans le temps avec la même assiduité. En effet, certaines versions sont plus « délaissées ». Aujourd'hui, les versions LTS (Long Term Support) sont Ubuntu 18.04 (supportée jusqu'en Avril 2028) et Ubuntu 20.04 (pas encore défini mais version LTS donc long terme aussi).

Ubuntu 18.04 : « Bionic Beaver »

Ubuntu 20.04 : « Focal Fossa »

Le choix pour ce projet se porte sur Ubuntu 18.04. C'est une version LTS aujourd'hui stable. La version 20.04 n'est pas une version très différente donc l'éventuel portage du système sur la version 20.04 ne nécessitera pas des changements trop conséquents.

La version la plus récente est apparue le 23 Avril 2020. C'est encore une version trop récente pour l'utiliser dans un tel projet. Si des erreurs surviennent ayant pour cause la version 20.04, il faudra changer l'OS et passer sur l'autre version LTS 18.04.

### 6.2 ROS

ROS a déjà été conceptuellement présenté dans un chapitre précédent. Cette partie décrit les différents outils mis à disposition par cet ensemble de frameworks pour faciliter l'implémentation de logiciels destinés à la robotique.

#### 6.2.1 Langages supportés

ROS est neutre et peut être programmé en différents langages. La spécification de ROS intervient au niveau de la communication inter-processus et inter-systèmes. Les connexions peer-to-peer sont négociées en XML-RPC qui existe dans un grand nombre de langages.

Principe de peer-to-peer de ROS : Un robot suffisamment complexe est composé de plusieurs ordinateurs ou cartes embarquées reliées par Ethernet ainsi que parfois des ordinateurs externes au robot pour des tâches de calcul intensif. Une architecture peer-to-peer couplée à un système de tampon (buffering) et un système de lookup (un name service appelé master dans ROS), permet à chacun des acteurs de dialoguer en direct avec un autre acteur, de manière synchrone ou asynchrone en fonction des besoins.

Les langages supportés sont :

- C++
- Python
- Lisp

Pour ce projet, les langages utilisés seront C++ (en majorité) et Python. Ce sont aussi les langages les plus utilisés pour le développement de packages ROS. Les ressources sont donc suffisantes pour couvrir l'ensemble des besoins.

La partie C++ est utilisée pour les actions nécessitant une vitesse d'exécution rapide et constante (capteurs) et python pour la partie gestion moins critique (packages intermédiaires pour le contrôle du robot).

### 6.2.2 Système de fichier

1. Les packages : unité principale d'organisation logicielle de ROS. Un package est un répertoire qui contient les nœuds (concept décrit plus tard), les bibliothèques externes, des données, des fichiers de configuration et un fichier de configuration xml nommé manifest.xml.

2. La stack : C'est une collection de packages. Elle propose une agrégation de fonctionnalités telles que la navigation, la localisation... Une stack est un répertoire qui contient les répertoires des packages ainsi qu'un fichier de configuration nommé stack.xml.

### 6.2.3 Notions de base de ROS (définitions)

1. Les nœuds : instance d'un exécutable. Un nœud peut correspondre à un capteur, un moteur, un algorithme de traitement, de surveillance... Chaque nœud qui se lance se déclare au Master. On retrouve ici l'architecture microkernel où chaque ressource est un nœud indépendant.

2. Le master : service de déclaration et d'enregistrement des nœuds qui permet ainsi à des nœuds de se connaître et d'échanger de l'information. Le Master est implémenté via XMLRPC. Il comprend une sous-partie très utilisée qui est le Parameter Server. Celui-ci, également implémenté sous forme de XMLRPC, comme son nom l'indique est une sorte de base de données centralisée dans laquelle les nœuds peuvent stocker de l'information et ainsi partager des paramètres globaux.

3. Les topics : mode de communication asynchrone. Système de transport de l'information basé sur le système de l'abonnement / publication. Un ou plusieurs nœuds pourront publier de l'information sur un topic et un ou plusieurs nœuds pourront lire l'information sur ce topic. Le topic est en quelque sorte un bus d'information asynchrone un peu comme un flux RSS. Le topic est typé, c'est-à-dire que le type d'information qui est publiée (le message) est toujours structuré de la même manière. Les nœuds envoient ou reçoivent des messages sur des topics.

4. Les messages : un message est une structure de donnée composite. Un message est composé d'une combinaison de types primitifs (chaînes de caractères, booléens, entiers, flottants...) et de message (le message est une structure récursive). La description des messages est stockée dans nom\_package/msg/monMessageType.msg. Ce fichier décrit la structure des messages.

5. Les services : communication synchrone entre deux nœuds. Cette notion se rapproche de la notion d'appel de procédure distante (remote procedure call).

6. Les paramètres : variables qui peuvent être globales à tout le système ROS ou visible pour un nœud spécifique. Ces variables sont présentes dans le serveur de paramètres. Ce dernier utilise des types de données XMLRPC pour les valeurs de paramètres (int32, bool, strings, doubles...). Les paramètres peuvent être définis dans un fichier .yaml récupéré lors du lancement d'un nœud (appelant ce fichier de paramètres). Le système de paramètres permet d'adapter le fonctionnement d'un nœud aux besoins du projet (modification de la fréquence, présence de certains capteurs spécifiques dans le système...).

## 6.3 Catkin

Catkin est un système de build. Un build est une version exécutable d'une partie du système ou du système entier.

Un système de build est chargé de générer des "targets" à partir du code source brut qui peuvent être utilisées par un utilisateur final. Ces cibles peuvent être sous la forme de bibliothèques, de programmes exécutables, de scripts générés, d'interfaces exportées (par exemple des fichiers d'en-tête C++) ou de tout autre élément qui n'est pas du code statique. Dans la terminologie ROS, le code source est organisé en "package", chaque package étant généralement constitué d'une ou plusieurs targets lors de sa construction.

ROS a son propre système appelé ROS build. Il se trouve que ROS ayant énormément évolué (au-delà des espérances des développeurs), ROS build n'était plus suffisant pour subvenir aux besoins de la constante évolution du framework. Il aurait fallu repenser le système de build et l'adapter pour suivre l'évolution de ROS.

La solution apportée est donc de passer sur un système de build existant : Catkin.

Je ne vais pas apporter trop d'informations sur Catkin, c'est un outil puissant qui simplifie grandement le building. Il faut cependant veiller à respecter l'architecture et la hiérarchie des informations pour avoir un fonctionnement optimal.

De nombreux tutoriels existent et expliquent très bien tous les points les plus importants pour comprendre correctement ce que fait Catkin et comment l'utiliser pour rester le plus générique et pratique possible.

Dans le tutoriel pas à pas dans les annexes (A[2]), toutes les commandes utilisées et les modifications apportées dans les fichiers CMakeLists.txt sont détaillées.

## 7 Robot contrôlé manuellement

Ce chapitre est séparé en deux parties.

Au vu de la situation de ces derniers mois, les commandes et le montage du robot de démonstration ont été lourdement retardés. De ce fait, c'est sur un autre robot (véhicule aussi) que nous nous sommes familiarisés avec ROS et les concepts de robotique.

Le développement sur la donkey Car a représenté une partie importante du temps alloué pour le travail de diplôme.



Figure 29 : Donkey Car

C'est une RC car basique avec une Raspberry pi 4.

Les différences majeures avec le robot de démonstration sont :

- La différence des moteurs. Ceux de la Donkey Car sont des Servos-moteurs.
- Les drivers moteurs (ESC pour la Donkey Car, Cytron MD10c pour le démonstrateur).
- La gestion de la conduite (Ackerman pour la Donkey Car contre différentielle pour notre système).

Ces différences ont nécessité plusieurs ajustements une fois le montage final du système de démonstration suffisamment avancé permettant de porter le système de contrôle du véhicule avec la manette PS4.

### 7.1 Développement sur la Donkey Car

Le site officiel de la Donkey Car ne propose pas de fonctionnement avec ROS comme base. Toute la mise en place s'effectue avec des scripts et des codes disponibles dans des repo fournis.

Le premier défi était donc de trouver un moyen pour faire fonctionner ce système avec ROS et pouvoir le contrôler avec la manette PS4.

#### 7.1.1 Package i2cPWM

La première manière d'avoir des résultats visibles est d'interfacer au plus vite la board envoyant des commandes PWM aux drivers moteurs pour activer les roues.

Pour cela, partir d'un package existant était la solution la plus pratique. Le nom du package est « ros-i2cpwmboard ». Ce package propose la configuration des registres du PC9685 (board i2c) pour que celle-ci envoie correctement les valeurs de PWM à l'ESC.

Au-delà de cette configuration, le package propose de nombreuses fonctionnalités :

- Trois navigations différentes (Ackerman, Differential et Mecanum).
- Modification des valeurs des PWM à envoyer.
- Modification des valeurs suivant certains modes spécifiques (à explorer...).

Pour les besoins du robot, nous n'utiliserons ce package que pour la configuration initiale des registres et pour la mise à jour des registres contenant les valeurs des PWM des deux moteurs à gérer.

Il est possible rien qu'avec ce package de lancer un nœud permettant de mettre à jour les valeurs et donc de contrôler les deux moteurs.

Pour ce faire, il faut lancer le nœud et publier manuellement des messages dans le topic souhaité (dans ce cas-ci le topic /servos\_absolute).

Ce n'est pas un message fourni par ROS qui est utilisé pour la publication des informations mais un message spécifique.

Le Callback mettant à jour les valeurs attend le message Servos, qui est un tableau de N Servo.

Chaque message Servo est composé de :

- int32 -> ID du PWM (0 à 15)
- float32 -> valeur du PWM souhaitée (entre 0 et 4096)

Si les moteurs s'activent, c'est que la configuration est bonne.

Attention concernant l'ESC de la Donkey Car, la première valeur envoyée après initialisation de ce composant sera la valeur d'arrêt. En dessous, les moteurs s'activeront dans un sens et au-dessus de cette valeur dans l'autre. Pour plus d'information concernant cette configuration, se référer à la doc de l'ESC (D[4]).

### 7.1.2 Package joy

Le contrôleur est connecté en Bluetooth avec le driver ds4drv (permettant de détecter la manette PS4 en Bluetooth et de l'interfacer avec le système).

C'est un package permettant d'utiliser différents contrôleurs, dont celui de la PS4 et de récupérer les informations des différents inputs dans un message pour les traiter comme voulu.

Le type de message reçu est « sensor\_msgs/Joy » et se définit ainsi :

```
# Reports the state of a joysticks axes and buttons.
Header header      # timestamp in the header is the time the data is received from the joystick
float32[] axes     # the axes measurements from a joystick
int32[] buttons    # the buttons measurements from a joystick
```

Ces messages sont des standards dans l'API de ROS.

Lorsque ces messages sont correctement publiés, le nœud permet alors de récupérer tous les inputs de la manette et de les transmettre sous la forme générique.



### 7.1.3 Package *ps4-ros*

Ce package a été mis en place par un utilisateur et correspond relativement bien au fonctionnement souhaité pour le contrôle d'un véhicule.

Il est important de souligner qu'avoir un système qui utilise les messages de l'API de ROS permet d'avoir un logiciel le plus générique possible et qui soit le plus résilient aux changements apportés. Par exemple ici le but est de commencer en contrôlant manuellement le véhicule mais par la suite, celui-ci doit fonctionner de manière autonome. Pour les déplacements, il existe un type de message générique et utilisé par la grande majorité des packages pour faciliter l'utilisation et la mise en place d'un nouveau package.

Les messages pour les déplacements sont les « *geometry\_msgs/Twist* » qui sont définis ainsi :

```
# This expresses velocity in free space broken into its linear and angular parts.  
Vector3 linear  
Vector3 angular
```

Ce message est composé d'autres messages (génériques aussi). On retrouve ici le même principe que la programmation orientée objet, mais avec les messages dans ROS.

Les messages « *Vector3* » sont composés de 3 valeurs floats nommées x, y et z.

Ce nœud va donc récupérer le message envoyé publié sur */joy* et va transformer les valeurs reçues pour les faire correspondre à des valeurs angulaires et linéaires comprises entre -1 et 1 (c'est la convention pour la gestion des vitesses). Cela permet d'avoir un minimum et un maximum et de fonctionner avec un système de facteurs pour les transformations.

Ce nœud va tout d'abord demander un « calibrage » de la manette PS4, puis une fois fait va récupérer les inputs de la manette et les transformer de la manière suivante :

- L2 = avancer
- R2 = reculer
- Joystick gauche = tourner (gauche/droite)

La sensibilité de l'appui sur les deux gâchettes et la manipulation du joystick vont générer des valeurs entre -1 et 1 dans les champs requis.

Pour le déplacement d'un véhicule terrestre (qui avance, recule et tourne), seul la valeur x du « *Vector3 linear* » et la valeur z du « *Vector3 angular* » seront utilisées. En effet, le véhicule ne peut qu'avancer et reculer (x linéaire) et tourner à gauche ou à droite (z angulaire).

### 7.1.4 Package *donkey\_car*

C'est la principale différence avec le système final du démonstrateur. La conduite étant « Ackerman », un moteur gère la vitesse (avancer / reculer) et l'autre la direction (gauche / droite).

Rien de complexe, il suffit d'effectuer des calculs avec des facteurs et les maximum et minimum que l'on souhaite imposer aux moteurs.

Pour plus de précisions se référer aux annexes (A[1]).

Le principe est simple, une conversion est faite avec les valeurs récupérées dans les messages « *geometry\_msgs/Twists* » publiés dans le topic « */cmd\_vel* » pour publier des messages « *Servos* » dans le topic « */servos\_absolute* ». Avec ce package (et le nœud lancé) le système fonctionne parfaitement et la Donkey car est utilisable avec la manette PS4.



## 7.2 Développement sur le robot démonstrateur

Une fois le robot démonstrateur suffisamment avancé pour, au minimum, porter le code précédemment généré, certaines adaptations étaient nécessaires.

### 7.2.1 Fonctionnement des drivers moteurs

Pour commencer, il fallait comprendre le fonctionnement de ces drivers moteurs pour savoir comment fournir les valeurs de PWM.

Les drivers moteurs du robot sont compatibles avec deux types d'opérations PWM :

1. Sign-Magnitude PWM : 2 signaux sont nécessaires, un pour gérer la direction du moteur (1 ou 0, avant ou arrière) et le PWM qui est branché sur le PWM des drivers pour contrôler la vitesse de rotation du moteur.
2. Locked-Antiphase PWM : un seul signal nécessaire, la valeur du PWM. Cependant, cette valeur est connectée à la direction, tandis que le pin de PWM du driver n'est pas utilisé (1 ou 0). Le fonctionnement est modifié : lorsque la valeur de PWM est à 50 % les moteurs s'arrêtent, si la valeur est en dessous de 50%, le moteur tourne en CW (ou CCW suivant la valeur donnée au pin du PWM du driver). Lorsque le signal est en dessus de 50%, le moteur tournera alors dans l'autre sens.

Explication de l'envoi du courant sur une des entrées du moteur :

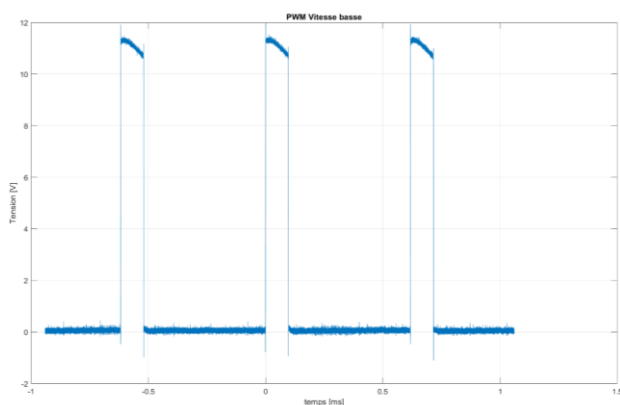


Figure 30 : PWM valeur basse

Voici le courant envoyé à une des entrées du moteur. Dans le cas 1, la vitesse des moteurs est basse et le sens est géré par le logiciel. Dans le cas 2, le driver envoie la période de courant inverse dans l'autre entrée ce qui va donner une vitesse élevée dans un certain sens de rotation du moteur (avant ou arrière selon le positionnement du moteur).

L'important est de comprendre que dans le cas 1, la vitesse est lente et la direction gérée par un autre pin, tandis que dans le cas 2, cela correspond à une vitesse élevée dans un certain sens de rotation.

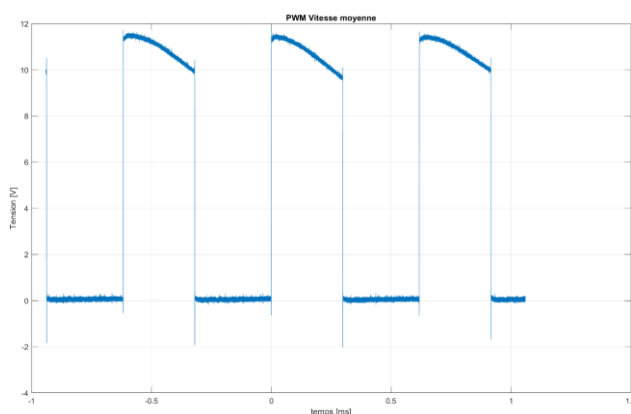


Figure 31 : PWM valeur 50%

En reprenant la même logique appliquée dans l'exemple précédent, cette valeur de PWM correspond à la vitesse de rotation moyenne des moteurs dans le cas 1 et à l'arrêt des moteurs dans le cas 2. La même valeur de courant est envoyée dans les deux sens de rotation des moteurs, ce qui laisse les moteurs à l'arrêt.

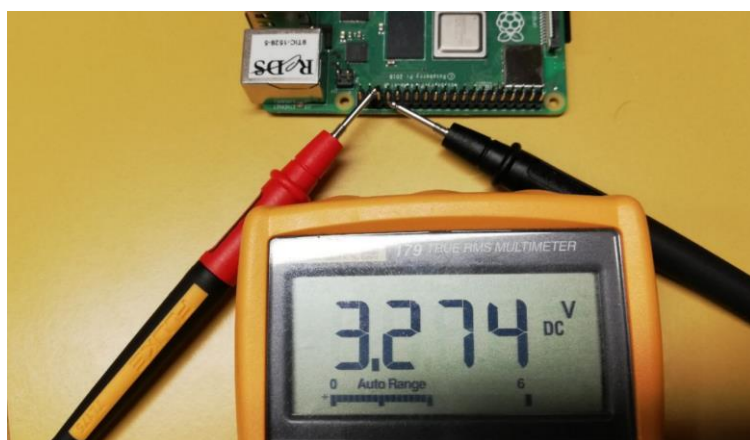
Ce point correspond pour le cas 2 au changement de sens de rotation des moteurs. Ainsi, si la valeur de PWM est supérieure au 50%, la rotation est inversée. La valeur maximum d'un sens est le 0 et le maximum de l'autre sens est le 100% de la valeur des PWM (donc 4096 pour nous).

Pour la prise de ces mesures, nous avons utilisé l'oscilloscope pour récupérer des données Matlab fournies directement par l'oscilloscope, puis générer le graphique sur Matlab pour avoir un design plus propre et une documentation plus agréable à parcourir.

La seconde option était similaire à celle utilisée dans le robot précédent. Cependant, la valeur de 50% demandait d'envoyer la même quantité de courant aux deux sorties de courant, un bruit de sifflement constant provenant des moteurs à l'arrêt nous a fait opter pour l'utilisation de la première solution.

Il a fallu pour cela choisir des GPIO de la Raspberry Pi et les utiliser de sorte que les moteurs tournent dans un sens ou dans un autre suivant les besoins. Les GPIO choisis sont « GPIO5 et 6 » et correspondent au pins 29 et 31 de la board.

Les drivers moteurs supportent, d'après la doc, pour le pin de direction une entrée 3V ou 5V. La Raspberry Pi 4 est censée générer une des deux valeurs en sortie. D'après la doc, c'est du 3.3V qui est en sorti. L'utilisation d'un voltmètre pour mesurer la sortie le confirme :



La sortie est de 3.274V sur le voltmètre. La valeur de sortie a aussi été vérifiée avec un oscilloscope qui donne une mesure plus précise et la sortie est bien de 3.3V.

Figure 32 : mesure sortie GPIO avec voltmètre

### 7.2.2 Package diff\_drive

L'adaptation première du système est la suivante : passer d'une conduite Ackerman à une conduite différentielle.

Un package existe : « diff\_drive ». Mais une adaptation est nécessaire. Les valeurs dans le négatif n'étant pas correctement gérées, le code a été modifié pour répondre à toutes les attentes. Nous avons adapté le code en modifiant les valeurs reçues pour toujours calculer les résultats de PWM dans le positif et les modifier suivant le signe reçu avant de les renvoyer dans les messages.

Le nœud récupère les messages de type « geometry\_msgs/Twists » publiés sur le topic « /cmd\_vel ».

Les messages sont publiés dans deux topics distincts :

- lwheel\_desired\_rate
- rwheel\_desired\_rate

La conduite différentielle regroupe les roues gauches et les roues droites. Ainsi, la gestion s'effectue de la même façon que s'il n'y avait que deux roues.

### 7.2.3 Package robot\_demo

Ce package reprend pour base le package donkey\_car. La différence est qu'il a fallu adapter le code pour intégrer deux callbacks différents permettant de gérer les deux lignes de roues (droite et gauche). Chaque callback récupère

des messages (respectivement des roues gauches et droites) contenant les valeurs de PWM souhaitées (comprises entre 0 et 4096) à envoyer aux moteurs.

Ce nœud permet aussi la gestion de la direction des moteurs. En effet, si la valeur reçue est positive ou négative, la gestion du GPIO contrôlant la direction du moteur sera différente.

Les moteurs sont montés en miroir. Pour avoir un système qui avance, il faudra que la valeur du pin d'une des lignes de roues soit l'opposé de l'autre.

La vitesse des moteurs a été bridée pour éviter que le système se déplace trop vite (et que la précision soit compromise).

Le code est similaire à celui de la Donkey Car concernant les « subscribe » et les « publish » mais change dans le traitement des données reçues.

Avec ces modifications, le système est porté sur le robot démonstrateur. Contrôler le robot avec une conduite différentielle est possible et fonctionnel.

## 8 Intégration des capteurs dans ROS

### 8.1 RPLidar A1

Le scanner et l'odométrie sont des capteurs essentiels du système de navigation stack. N'ayant pas d'odométrie prévue dans le robot actuel, c'est le Lidar qu'il faut commencer par intégrer.

Pour cela il existe un package permettant de fournir les données formatées avec le type de messages de base que prévoit ROS : « sensor\_msgs/LaserScan »

```
# This expresses velocity in free space broken into its linear and angular parts.
Header                                # timestamp in the header is the acquisition time of first ray in the scan.
                                     # in frame frame_id, angles are measured around the positive Z axis
                                     # (counterclockwise, if Z is up) with zero angle being forward along the x axis
float32 angle_min                     # start angle of the scan [rad]
float32 angle_max                     # end angle of the scan [rad]
float32 angle_increment               # angular distance between measurements [rad]
float32 time_increment                # time between measurements [seconds] - if your scanner is moving, this will # be
                                     # used in interpolating position of 3d points
float32 scan_time                     # time between scans [seconds]
float32 range_min                     # minimum range value [m]
float32 range_max                     # maximum range value [m]
float32[] ranges                      # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities                # intensity data [device-specific units]. If your device does not provide intensities,
                                     # please leave the array empty.
```

Un fichier .yaml permet de configurer comme souhaité les paramètres modifiables du RPLidar, mais la configuration de base convient parfaitement pour intégrer et tester le nœud et la publication des messages dans un topic spécifique.

Le nœud publie dans le topic /scan les données récupérées du hardware.

Pour confirmer la justesse des données, il est intéressant de visualiser sous forme de nuage de points les données récupérées. Pour cela, le nœud Rviz est utilisé et le package récupéré permet de lancer la visualisation et la récupération des données simultanément. La visualisation génère une image de ce type :

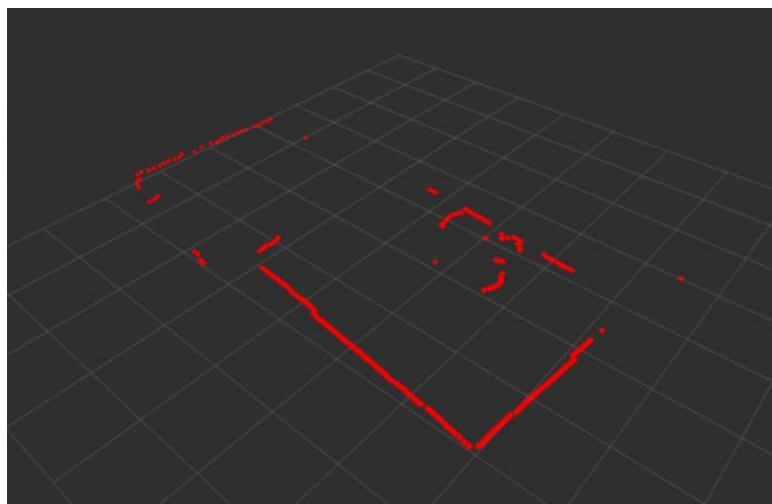


Figure 33 : RPLidar points - Rviz

Le nuage de points visible correspond bien à la pièce dans laquelle le test a été réalisé.

La visualisation n'est pas très précise car elle ne prend que certains points pour reconstituer les données récupérées mais l'aspect d'ensemble est le bon.

Lorsque le RPLidar est déplacé, la base ne change pas mais la pièce se redessine correctement.

## 8.2 Mti-7 DK (IMU /GNSS)

Un tutorial est disponible pour intégrer le dev kit dans le workflow de ROS. ROS étant très populaire dans la robotique, de nombreux fournisseurs de hardware proposent un package ROS déjà utilisable pour éviter à la communauté de devoir le faire. Cela rend le matériel intéressant car le gain de temps pour la récupération des informations est non négligeable.

Cependant, l'installation basique ne fonctionne pas dans le cadre du projet. En effet, les librairies XDA sont fournies compilées pour du x86 et la Raspberry pi 4 est dotée d'une architecture ARM. Les librairies ne sont donc pas compatibles.

Il fallait auparavant utiliser des commandes bas niveau pour récupérer les informations. Cependant, depuis 2019 (automne), Xsense a ouvert au public le code des librairies XDA, ce qui permet de les compiler soi-même et donc de les rendre compatible pour ARM.

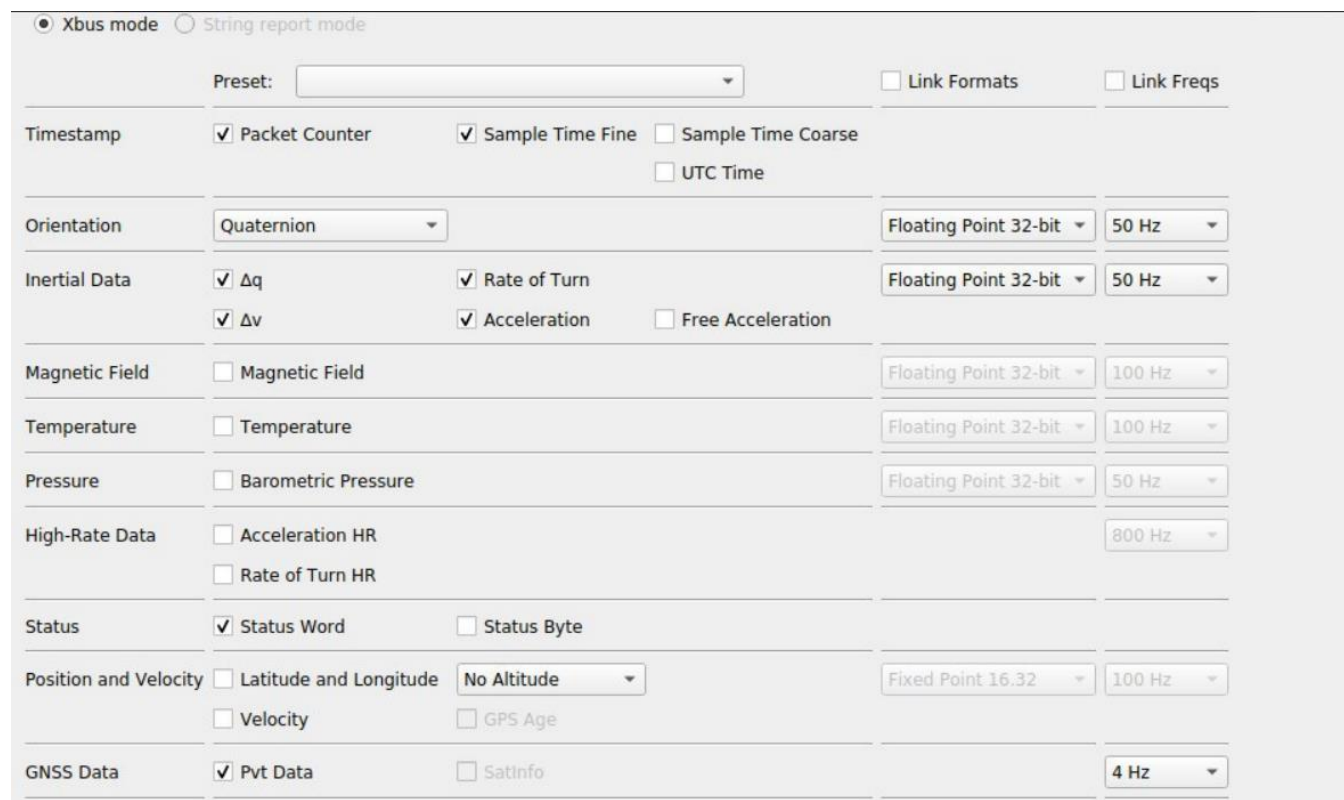
Le plus pratique est de faire les installations basiques sur un ordinateur sous x86, récupérer le package ROS fourni (se trouvant dans le path de la librairie) et de l'intégrer directement dans les sources du projet. En suivant ensuite les instructions permettant de compiler au préalable les librairies, la compilation du projet fonctionne.

Contrairement aux librairies installées de base, la compilation des codes du package ROS fournit des librairies statiques (et non linkés dynamiquement), ce qui permet d'utiliser aussi ce package ROS sur des microcontrôleurs qui ne sont pas dotés d'OS, mais cela ne concerne pas ce projet.

La configuration du matériel est aussi une partie indispensable du processus. Pour cela, il est beaucoup plus simple d'avoir un système sur x86 avec une interface graphique (Desktop) permettant d'accéder au MTmanager. Cet outil est une interface utilisateur permettant de configurer les sorties et la fréquence d'envoi des données du mti-7.

Le kit possède une mémoire permettant de garder la configuration même hors tension, ce qui permet de le configurer depuis une machine et de l'utiliser sur la Raspberry Pi 4 par la suite.

La configuration est la suivante :



The screenshot shows the MTManager configuration window. At the top, there are two radio buttons: 'Xbus mode' (selected) and 'String report mode'. Below them is a 'Preset' dropdown menu. To the right, there are checkboxes for 'Link Formats' and 'Link Freqs'. The main configuration area is divided into several sections, each with a label on the left and a set of options on the right. The sections are: 'Timestamp' (with checkboxes for Packet Counter, Sample Time Fine, Sample Time Coarse, and UTC Time), 'Orientation' (with a Quaternion dropdown and Floating Point 32-bit and 50 Hz options), 'Inertial Data' (with checkboxes for Δq, Rate of Turn, Δv, Acceleration, and Free Acceleration, and Floating Point 32-bit and 50 Hz options), 'Magnetic Field' (with a checkbox for Magnetic Field and Floating Point 32-bit and 100 Hz options), 'Temperature' (with a checkbox for Temperature and Floating Point 32-bit and 100 Hz options), 'Pressure' (with a checkbox for Barometric Pressure and Floating Point 32-bit and 50 Hz options), 'High-Rate Data' (with checkboxes for Acceleration HR and Rate of Turn HR, and 800 Hz options), 'Status' (with checkboxes for Status Word and Status Byte), 'Position and Velocity' (with checkboxes for Latitude and Longitude, No Altitude dropdown, Velocity, and GPS Age, and Fixed Point 16.32 and 100 Hz options), and 'GNSS Data' (with checkboxes for Pvt Data and SatInfo, and 4 Hz options).

Figure 34 : configuration mti-7 DK

Un détail important concernant la configuration : si trop d'informations avec une fréquence trop élevée sont demandées, alors ce message apparaît :

```
05.08.2020 - 09:25:17.192 -> WARNING: Device: 07880A40: Data Overflow detected. Sample times may not be correct. Selected baudrate may be too low.
05.08.2020 - 09:25:17.149 -> WARNING: Device: 07880A40: Data Overflow detected. Sample times may not be correct. Selected baudrate may be too low.
05.08.2020 - 09:25:17.094 -> WARNING: Device: 07880A40: Data Overflow detected. Sample times may not be correct. Selected baudrate may be too low.
```

Il suffit de modifier la fréquence d'envoi des données (ici 50Hz pour un fonctionnement sans Overflow).

De base, le package configure un nœud qui publie des messages différents dans de nombreux topics. Seuls deux types de messages principaux sont intéressants dans le cadre de la navigation autonome :

#### 1. sensor\_msgs/Imu (publié dans le topic /imu/data)

```
# This is a message to hold data from an IMU (Inertial Measurement Unit)
#
# Accelerations should be in m/s^2 (not in g's), and rotational velocity should be in rad/sec
#
# If the covariance of the measurement is known, it should be filled in (if all you know is the variance of each
# measurement, e.g. from the datasheet, just put those along the diagonal). A covariance matrix of all zeros will
# be interpreted as "covariance unknown", and to use the data a covariance will have to be assumed or gotten
# from some other source
#
# If you have no estimate for one of the data elements (e.g. your IMU doesn't produce an orientation
# estimate), please set element 0 of the associated covariance matrix to -1
# If you are interpreting this message, please check for a value of -1 in the first element of each
# covariance matrix, and disregard the associated estimate.

Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance # Row major about x, y, z axes
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance # Row major about x, y, z axes
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance # Row major x, y z
```

#### 2. sensor\_msgs/NavSatFix (publié dans le topic /gnss)

```
uint8 COVARIANCE_TYPE_UNKNOWN=0
uint8 COVARIANCE_TYPE_APPROXIMATED=1
uint8 COVARIANCE_TYPE_DIAGONAL_KNOWN=2
uint8 COVARIANCE_TYPE_KNOWN=3
std_msgs/Header header
sensor_msgs/NavSatStatus status
float64 latitude
float64 longitude
float64 altitude
float64[9] position_covariance
uint8 position_covariance_type
```

De nombreuses explications sont fournies pour ce message, c'est donc la version compacte qui est fournie dans le rapport. Pour plus de précisions, se référer à la documentation de ROS.

Pour tester le bon fonctionnement du matériel, il est possible d'utiliser le mtmanager qui permet la configuration du système, mais aussi la visualisation des données sous forme graphique.



Voici un exemple de visualisation des données :

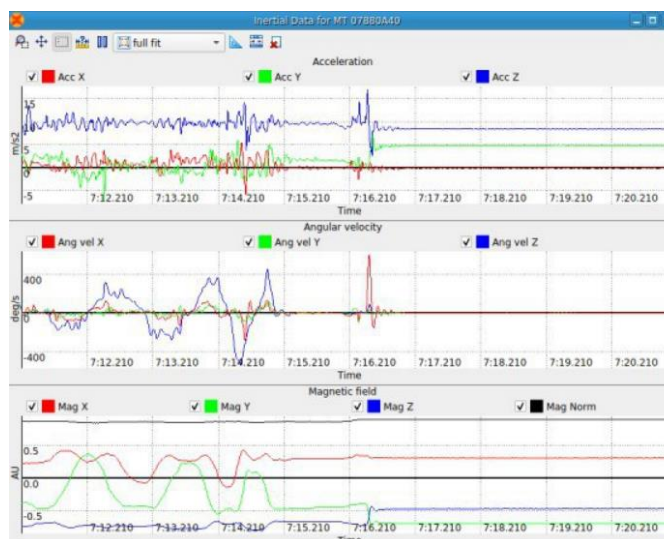


Figure 35: visualisation des accélérations

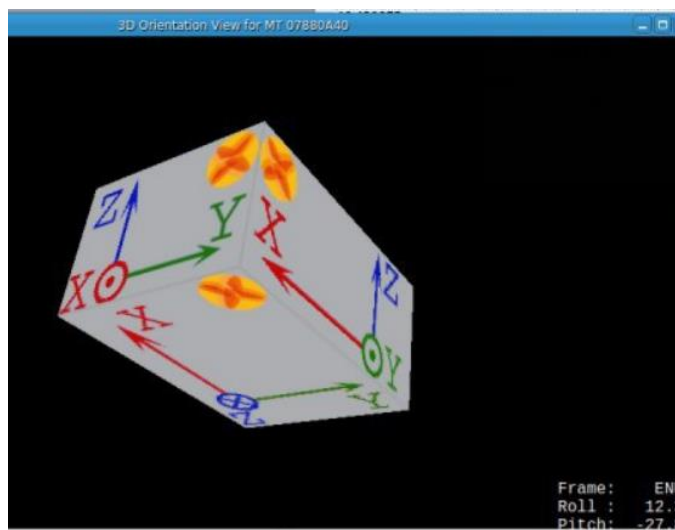


Figure 36 : visualisation de la position angulaire

Après des tests effectués de manière empirique, le délai est très faible et les données sont correctes (pas de vérifications précises des données possible avec le temps/matériel à disposition).

Concernant l'IMU, certaines informations sont importantes.

Premièrement, il existe deux normes différentes pour la récupération et le traitement des informations d'une IMU :

- ENU (east, north, up)
- NED (north, east, down)

C'est ce que l'on appelle les conventions des axes. Pour plus d'informations, se référer à la page Wikipédia car le concept est complexe mais très bien expliqué (voir dans la bibliographie).

L'important est de savoir que ROS fonctionne avec la convention ENU, si les données reçues sont en NED, il existe un package ROS pour les convertir et les formater correctement. La centrale IMU/GNSS mti-7 DK fournit les informations en ENU, donc aucune conversion n'est requise.

Il est important pour le fonctionnement dans ROS d'avoir l'axe X correctement dirigé ou d'utiliser un package pour transformer les axes. Il faut savoir que l'axe X doit être parallèle au déplacement avant/arrière du véhicule.

L'indication des axes est imprimée sur l'IMU :

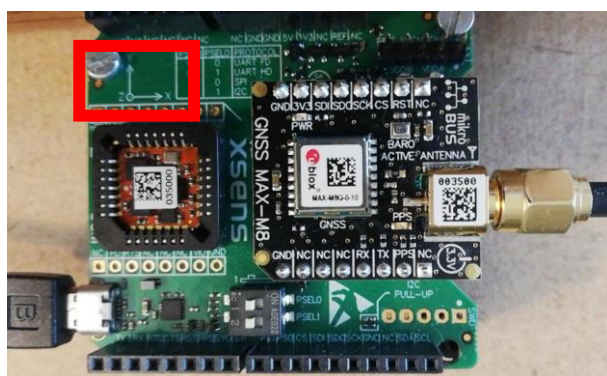


Figure 37 : IMU axes

Le cadre rouge montre l'impression des axes sur le PCB.

L'axe X doit être correctement dirigé ou bien il faut appliquer une transformation avec le package TF (expliqué plus tard).

### 8.3 D435i Realsense

Enfin, l'intégration de la caméra D435i dans le système. Le package ROS utilise la librairie « librealsense2 » qu'il faut installer au préalable dans l'environnement de la Raspberry Pi 4.

Une fois la librairie installée, le package permet de générer plusieurs nœuds différents selon le type de matériel. C'est un package générique permettant de gérer la plupart des caméras librealsense.

Le nœud de base ne fournit que les informations de bases d'une caméra realsense (image raw, profondeur...). La d435i possède un IMU intégré, ce qui permet de récupérer le même type de données que l'IMU intégrée précédemment (mais la précision est moins élevée).

Le plus intéressant que propose cette caméra pour le système est la possibilité d'obtenir une odométrie visuelle qui peut se fusionner avec d'autres types d'odométrie pour améliorer la précision de la navigation autonome.

La caméra étant surtout utilisée dans le cadre d'un travail de diplôme différent (concernant le traitement d'image pour établir un algorithme permettant la détection de divers plantons), nous avons pu récupérer l'intégration de la D435 pour utiliser les parties qui étaient intéressantes pour le fonctionnement du robot.

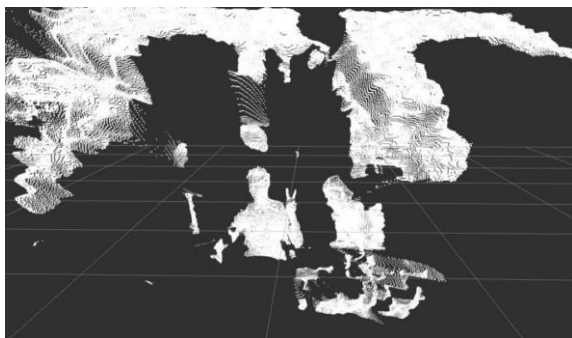


Figure 38 : white point cloud

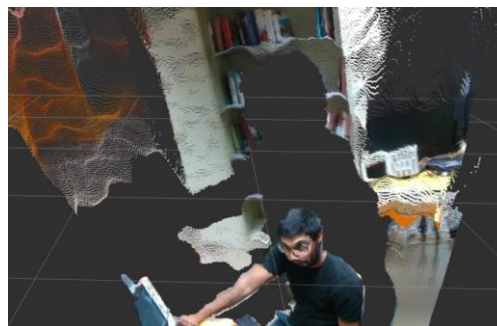


Figure 39 : colored point cloud

Voici les images récupérées avec Rviz permettant la visualisation du nuage de points que détecte la caméra. Cette sortie est assez précise mais fonctionne avec les paramètres des FPS de l'ordre de 3 à 5.

Ces nuages permettent de dessiner une map plus précise en 3D permettant de détecter plus d'obstacles qu'uniquement avec le RPlidar. La caméra fournit de nombreux topics, il serait intéressant de garder le minimum utile pour le projet pour déterminer s'il est possible de l'utiliser ou non avec la Raspberry Pi 4 pour obtenir des informations de précision supplémentaires.



## 9 Système autonome

Le système de navigation autonome dans ROS possède une base intéressante et relativement complète qu'il est possible d'utiliser.

Ce système s'appelle la « navigation stack ». D'un point de vue conceptuel, la nav stack est simple. Elle prend des informations d'après l'odométrie et des capteurs et ressort une commande de vitesse destinée à la base mobile. Cependant, l'utiliser sur un robot autonome est passablement plus compliqué.

Les prérequis logiciels de base sont :

- Robot fonctionne avec ROS.
- Possède un tf (transform tree, concept expliqué par la suite).
- Publication des messages des capteurs correctement formatés avec les messages ROS prévus.
- La navigation doit être configurée correctement pour la forme du robot.

La navigation stack a été créée pour être le plus général possible. Cependant, s'il existe des prérequis logiciels, sont présents également des prérequis matériels :

- Uniquement pour des robots à conduite différentielle (avec holonomie possible).
- Un laser balayage 360° monté sur la base du robot.
- Un robot carré pour de meilleures prévisions.

Ci-dessous un schéma du fonctionnement de la Nav Stack avec les entrées et sorties :

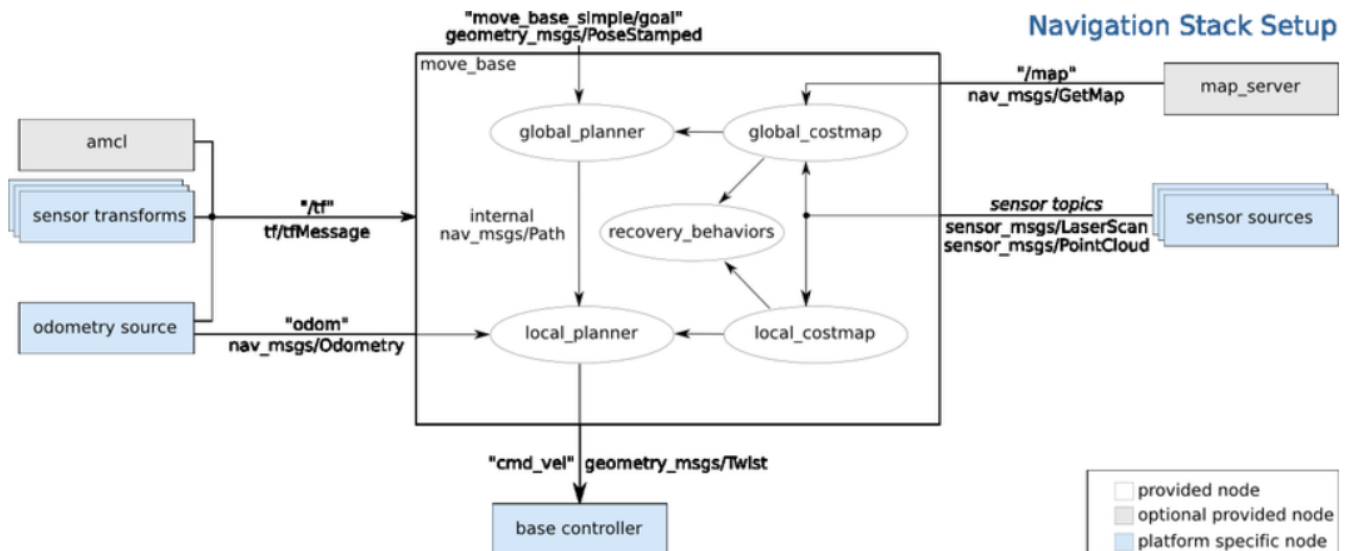


Figure 40 : Schéma navigation stack

Pour commencer, il est important d'interfacer tous les différents capteurs et de les intégrer à ROS pour définir ce qu'il est possible de fournir et de quelle manière.

## 9.1 Quelques définitions

Les concepts expliqués ci-dessous sont tous assez complexes. L'objectif ici est de les vulgariser pour qu'ils soient suffisamment clairs lorsqu'ils interviendront dans l'explication de la navigation autonome.

### 9.1.1 Les frames

Un système est doté de nombreuses frames variables selon les besoins / possibilités.

Les frames sont des références utilisées pour localiser des objets / robots. En effet, il est impossible de déterminer une position et une orientation sans un point de référence. Ce point est une frame.

La frame principale dans ROS est la « reference frame » qui correspond au point  $x=0, y=0$  de la grid (visible sur Rviz). La localisation des éléments seront tous référencés sur cette frame. Voici un exemple :

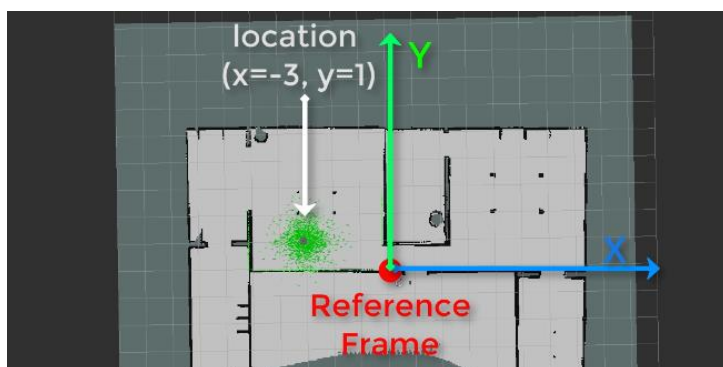


Figure 41 : reference frame

Cette reference frame est en général appelée « map ».

La grille est segmentée d'après la taille des carrés. Chaque pas représente une valeur de 1. La segmentation peut être modifiée selon les besoins.

Ci-dessous un exemple de visualisation des différentes frames d'un projet de robot autonome :

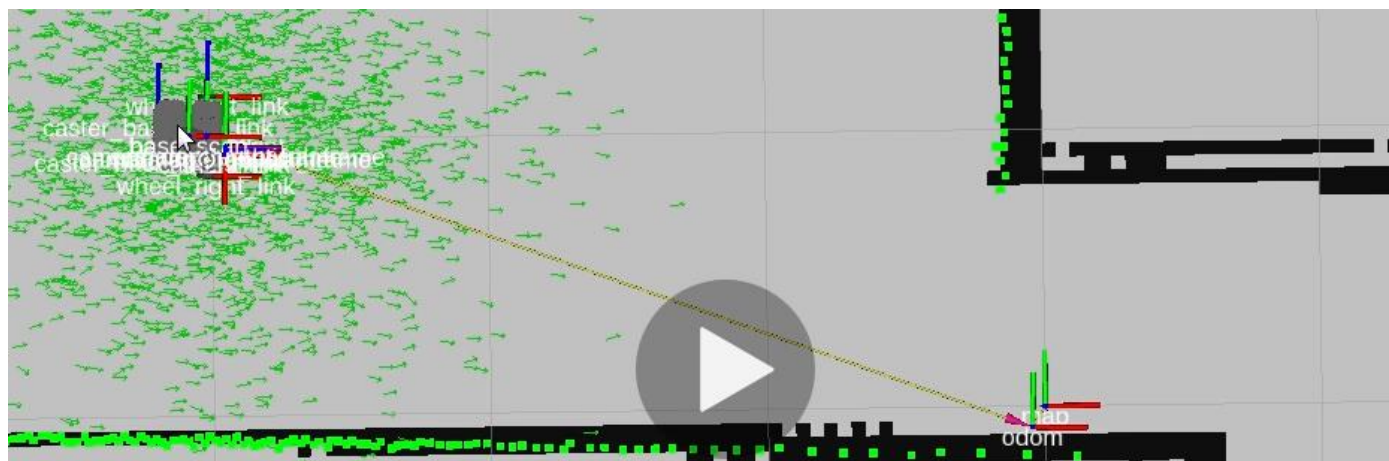


Figure 42 : visualisation des frames

Chaque axes  $x/y/z$  correspond à une frame différente.

Les frames peuvent être utilisées, par exemple, pour détecter la position d'une roue (droite ou gauche, avant ou arrière) pour modifier des calculs d'après les valeurs de position relatives ou absolues.

La position d'un objet est donc relative par rapport au point de référence (frame) pris en compte.

Une des utilisations des frames est la suivante : Si un robot connaît sa position par rapport à la frame statique (map) et connaît la position d'un obstacle, comment déterminer la position absolue de l'obstacle ? C'est à ce moment qu'intervient le concept de « frame transformation » qui sera utilisé dans le projet.

### 9.1.2 TF package

« Transformation Frame » package. C'est le package le plus important de ROS pour la navigation autonome. C'est ce qui permet d'effectuer les transformations nécessaires d'une frame à une autre pour déterminer la position d'un élément spécifique par rapport à la base souhaitée.

Ce package utilise l'équation suivante pour faire les transformations :

$$\begin{bmatrix} {}^w x \\ {}^w y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} * \begin{bmatrix} {}^w x \\ {}^w y \\ 1 \end{bmatrix}$$

Rotation Matrix      Translation Vector

Figure 43 : équation tf

Toutes les transformations s'effectuent avec ce package suivant cette équation.

Ce package possède aussi un nœud permettant de visualiser le tf tree (ensemble des liens entre frames) avec une GUI ou bien en générant un PDF.

### 9.1.3 Orientation « Quaternion »

Il faut retenir de ce concept que c'est la structure utilisée pour représenter l'orientation d'un robot dans ROS.

Cette structure se présente sous cette forme : x, y, z et w. Pour nous, ces valeurs n'ont pas vraiment de sens et les relier à une orientation n'est pas possible. C'est cependant la structure de base de ROS pour les calculs d'orientation. Les humains comprennent les valeurs de type roll, pitch et yaw.

### 9.1.4 Position d'un robot

La position d'un robot sur un plan 2D correspond à des valeurs x et y et une orientation w :

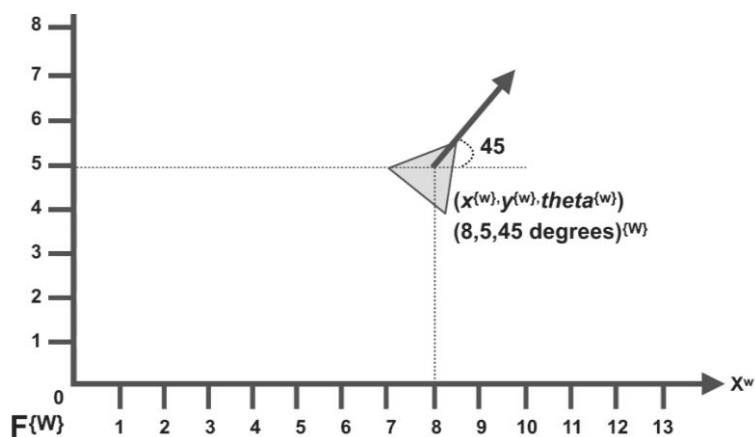


Figure 44 : position d'un objet

Il est possible de définir un robot ou un objet dans un environnement avec ces valeurs.

Plus ces valeurs sont précises, meilleur est le système.

La pose sera différente selon la frame de référence. Les frames peuvent avoir un décalage sur x, y et w. Certaines frames sont statiques et d'autres ne le sont pas. Il est donc important de bien comprendre le concept de frame pour comprendre les poses relatives des objets les uns par rapport aux autres et les poses absolues dans un environnement.

### 9.1.5 Recovery process

Lors de la navigation autonome, le robot va parfois se retrouver face à des obstacles qui apparaissent ou qui n'étaient pas présents lorsque la map a été générée. Le chemin initial prévu n'a donc pas pris en compte ces données et va potentiellement se heurter à ces objets.

Le concept du recovery process est de retourner à un état initial précédent (retour en arrière du véhicule) et de recalculer le passage qui pose un problème. Cet algorithme permet d'éviter des objets et/ou de passer dans des endroits avec peu d'espace pour naviguer.

### 9.1.6 Clearing process

Tandis que le robot se déplace dans son environnement, la map est modifiée en utilisant le scanner laser pour supprimer les obstacles qui sont présents de base sur la map mais que ne le sont plus lors du passage du robot.

## 9.2 La navigation autonome dans ROS

La navigation autonome signifie la capacité de se mouvoir dans l'espace sans percuter des obstacles. Il y a plusieurs concepts majeurs :

### 9.2.1 Navigation « map-based »

Le robot utilise et charge une map de l'environnement (nécessaire à la navigation). Le robot a donc une connaissance globale des obstacles statiques et la map est utilisée pour planifier les mouvements que le robot doit effectuer pour se déplacer d'un point A à un point B.

C'est cette partie de la navigation qui gère le clearing process.

Pour ce faire, il est important de répondre à trois questions principales :

1. Où suis-je ? Le concept de localisation permet de répondre à cette question.
2. Où vais-je ? C'est la map qui permet de le savoir.
3. Comment m'y rendre ? L'algorithme de la navigation stack calcule le chemin possible (motion planning).

### 9.2.2 Navigation réactive

Contrairement au concept précédent, cette navigation n'utilise pas de map, mais uniquement les informations locales du robot observées par les différents capteurs. Ces informations sont traitées et utilisées pour planifier le déplacement du robot.

Cette navigation gère le recovery process.

## 9.3 Définir les entrées de la Nav Stack

Comme expliqué en début de chapitre, la navigation stack a besoin de diverses entrées pour déterminer une vitesse en sortie. L'objectif est d'utiliser tous les capteurs intégrés précédemment pour combler toutes les demandes et obtenir un système de navigation.

### 9.3.1 Odométrie

L'odométrie est une variable extrêmement importante dans ROS pour tout ce qui concerne la navigation autonome. Même si sa précision est approximative, c'est la base utilisée pour tout le système et elle sera corrigée avec les autres capteurs présents si nécessaire.

Le robot ne possède pas des capteurs permettant d'obtenir une odométrie facilement. En effet, ce sont en général des capteurs sur des roues, sur des moteurs ou bien une roue « folle » qui permet de l'obtenir. Il va cependant falloir trouver un moyen d'en générer une avec le système actuel.

Avec le matériel présent sur le robot, l'odométrie peut être générée de trois manières principalement décrites ci-dessous.

### 1. Caméra stéréoscopique

Comme vu précédemment, la caméra stéréoscopique avec capteurs de profondeurs permet de récupérer une odométrie visuelle. Le package de RealSense propose de nombreux fichiers .launch différents adaptés aux diverses caméras. La d435 possède un IMU mais il n'est pas nécessaire de récupérer ces données si le seul objectif est de récupérer l'odométrie.

La D435 possède même un package permettant de générer une map en 3D uniquement avec la caméra. Cela fonctionne très bien en intérieur car de nombreux points de collisions avec les murs, objets ou obstacles divers permettent de générer correctement une map précise pour naviguer.

### 2. Odométrie d'après des scans

L'odométrie peut être générée d'après un algorithme en utilisant le laser du RPLidar. Le package se nomme « rf2o\_laser\_odometry ».

RF2O est une méthode rapide et précise pour estimer le mouvement planaire d'un lidar à partir de balayages de distance consécutifs. Contrairement aux approches traditionnelles, cette méthode ne recherche pas de correspondances mais effectue un alignement de balayage dense basé sur les gradients de balayage, à la manière de l'odométrie visuelle 3D dense. Le problème de minimisation est résolu dans un schéma grossier à fin de faire face aux grands déplacements, et un filtre lisse basé sur la covariance de l'estimation est utilisé pour gérer l'incertitude dans les scénarios sans contrainte (par exemple, les corridors).

Une explication plus détaillée se trouve dans les annexes (A[1]).

Cette méthode seule cependant produit, certes, une estimation très bonne dans des conditions optimales (vitesse lente et changements de directions lents) mais peut se perdre rapidement si les conditions ne sont pas satisfaites.

### 3. Odométrie d'après l'IMU et le GPS

En extérieur, il est possible d'obtenir une odométrie d'après le mti-7 DK. Il existe un package (robot\_localization) qui permet d'utiliser deux nœud différents pour générer une odométrie et la fournir à l'autre nœud. C'est une boucle nécessaire pour estimer une odométrie d'après uniquement ces deux capteurs.

Le filtre de Kalman est l'algorithme utilisé pour générer cette odométrie. C'est un filtre à réponse impulsionnelle infinie qui estime les états d'un système dynamique à partir d'une série de mesures incomplètes ou bruitées. C'est un thème majeur de l'automatique et du traitement du signal. Un exemple d'utilisation peut être la mise à disposition, en continu, d'informations telles que la position ou la vitesse d'un objet à partir d'une série d'observations relatives à sa position, incluant éventuellement des erreurs de mesures.

Pour le robot, c'est l'EKF (Extended Kalman Filter) qui est utilisé. En effet, le filtre de Kalman est limité aux systèmes linéaires. Cependant, la plupart des systèmes physiques sont non linéaires, l'EKF permet de gérer ces cas.

Les trois solutions sont explorées dans la suite du rapport et il est possible que plusieurs de ces solutions soient finalement fusionnées pour obtenir une précision accrue de l'odométrie.

L'odométrie est très sensible et il est complexe d'en obtenir une précise facilement. En effet, il existe des algorithmes permettant de fusionner toutes les données et d'estimer des sorties d'odométrie cohérentes par rapport à la précédente. Ces algorithmes permettent d'éviter que le système ne se perde et rend la sortie d'odométrie plus fiable et plus résistante.

### 9.3.2 Map

Une map est une grille (matrice) de cellules. Suivant la résolution, une cellule peut avoir une taille de 5 à 50 centimètres.

Le concept de map dans ROS repose sur des concepts similaires par rapport au reste des messages. En effet, une map est représentée par un tableau d'uint8. Ce tableau garde tous les points avec une distance donnée en paramètre (choix du développeur). Chaque valeur dans ce tableau représente une « cellule ». Chaque cellule a une valeur entre 0 et 100 représentant la probabilité d'occupation. Si la valeur est -1, c'est que le taux est inconnu.

Ces données peuvent être utilisées par Rviz pour visualiser la map actuelle générée (permettant de vérifier le bon fonctionnement du processus de mapping).

Le message permettant de définir une map est : `nav_msgs/OccupancyGrid` et se définit de la manière suivante :

```
# This represents a 2-D grid map, in which each cell represents the probability of occupancy.
Header
MapMetaData info      #MetaData for the map
int8[] data           # The map data, in row-major order, starting with (0,0). Occupancy probabilities are
                      # in the range [0,100]. Unknown is -1.
```

Ce message est en partie composé d'un autre message : « `nav_msgs/MapMetaData` » formaté ainsi :

```
# This hold basic information about the characterists of the OccupancyGrid
time map_load_time    # The time at which the map was loaded
float32 resolution    # The map resolution [m/cell]
uint32 width          # Map width [cells]
uint32 height         # Map height [cells]
geometry_msgs/Pose origin # The origin of the map [m, m, rad]. This is the real-world pose of the
                      # cell (0,0) in the map.
```

La map est générée par un package nommé « `hector_map` » qui prend en entrée les messages publiés sur le topic `/scan` (donc de type `sensor_msgs/LaserScan`) et qui génère une map d'après un algorithme traitant les messages scannés. L'odométrie permet au système de savoir dans quel état (position et position angulaire) se trouve l'objet et donc comment il se déplace pour ajouter à l'algorithme ces données permettant de corriger le scanner pour que l'algorithme ne se « perde pas ».

La map est générée en continue et peut être visualisée sur Rviz tandis que le robot se déplace dans une pièce.

Voici un exemple de map sur Rviz :

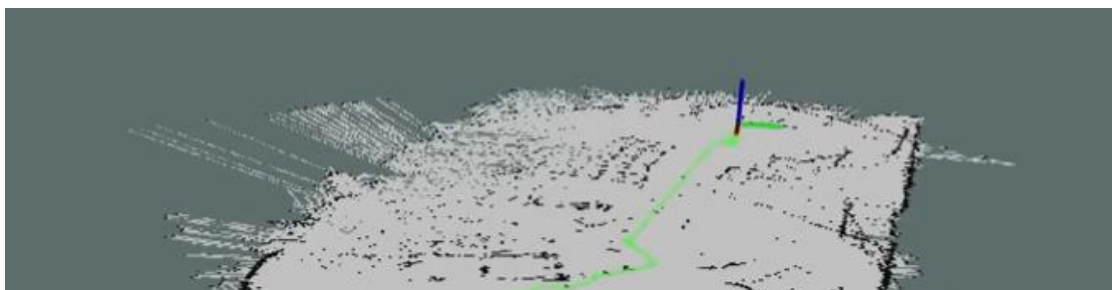


Figure 45 : Rviz map `hector_slam`

La map est donc générée tandis que le robot se déplace. Le trait vert correspond au déplacement du robot.



Concernant la map, il est possible de fournir une map au système en entrée ou bien de la générer depuis le néant. Dans le cadre du projet de diplôme, l'intégration complète du système autonome est l'objectif premier. Nous nous contenterons d'une map générée du néant pour commencer.

Pour sauvegarder une map, un package existe (`map_server` `map_saver`). Ce nœud permet de générer un fichier « \*.pgm » et un fichier « \*.yaml ».

Le fichier .pgm contient les données de la map. Ce fichier correspond à une représentation en grayscale de la map qui peut être ouvert depuis n'importe quel éditeur d'image. Il est possible de modifier directement la map avec l'éditeur. L'image ressemble à cela :

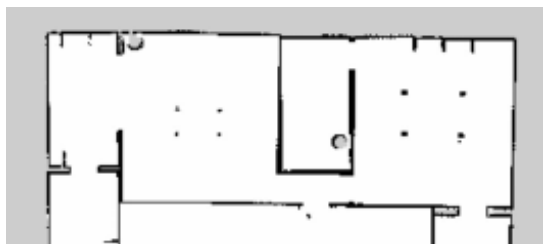


Figure 46 : map ouverte dans un éditeur

Les espaces blancs représentent les espaces inoccupés et les espaces noirs les espaces occupés.

Le fichier .pgm représente les données encodées avec le format P5 (encode en code binaire les informations).

Le fichier .yaml contient les métadatas de la map (la résolution en m/pixel, l'origine, la valeur médiane déterminant l'occupation...).

Pour générer une map, il est important d'avoir trois frames spécifiques : `odom` (odométrie du robot), `map` (référence absolue) et `base_footprint` (position du robot). `Hector_map` fonctionne cependant sans odométrie mais en fournir une rend la génération de la map plus facile et plus juste.

Un paramètre important de la map est le paramètre « delta » qui représente la résolution de la map (0.05 correspond à 5 centimètres).

### 9.3.3 Nuage de points

Le nuage de points peut être fourni par la caméra stéréoscopique D435.

Le message se nomme : `sensor_msgs/PointCloud` et est de la forme :

```
# This message holds a collection of 3d points, plus optional additional information about each point.
Header                                     # Time of sensor data acquisition, coordinate frame ID.
geometry_msgs/Point32[] points            # Array of 3d points. Each Point32 should be interpreted as a 3d point
                                           # in the frame given in the header.
ChannelFloat32[] channels                 # Each channel should have the same number of elements as points array,
                                           # and the data in each channel should correspond 1:1 with each point.
                                           # Channel names in common practice are listed in ChannelFloat32.msg
```

Ce nuage de points n'est pas indispensable mais permet d'être plus précis dans la détection d'obstacles, ce qui rend la navigation plus sûre.

### 9.3.4 Laser Scan

Comme vu précédemment, c'est le Lidar qui fournit cette information. Elle est déjà récupérée par `Hector_slam` et par l'algorithme `rf2o`. La Navigation stack peut aussi s'abonner au topic `/scan` pour récupérer l'information.

### 9.3.5 Algorithme AMCL

AMCL est un système de localisation probabiliste pour un robot se déplaçant en 2D.

Il met en œuvre l'approche de localisation Monte Carlo (telle que décrite par Dieter Fox) adaptative (ou d'échantillonnage KLD), qui utilise un filtre à particules pour suivre la pose d'un robot par rapport à une carte connue.

La base de cet algorithme fonctionne avec les données du scanner laser et de la map du robot. Le but est d'estimer une pose pour confirmer la pose de base déterminée par le reste du système.

## 9.4 Conclusion du système autonome

La « boîte noire » du déplacement autonome se nomme « move\_base » et cette boîte reçoit en entrées les informations expliquées en début de chapitre.

Cette boîte se divise en deux parties principales qui sont la navigation map-based et la navigation réactive. Ces deux parties principales utilisent les informations des divers capteurs pour déterminer un chemin pour arriver à destination.

Chaque partie a un rôle important, la première permet de naviguer dans un environnement statique et la deuxième d'adapter le chemin pour y arriver.

La gestion des paramètres est importante et l'efficacité et la précision de la navigation autonome seront directement impactées par cette gestion.

La mise en place d'un tf tree correct est la priorité absolue du système de déplacement.

Le système de déplacement n'est pas affecté par un système qui doit naviguer en extérieur ou en intérieur, ce sont les capteurs autour et la génération de la map qui seront gérés différemment mais tant que les bonnes informations sont apportées à la navigation stack, la navigation autonome est fonctionnelle.

Le but de ce système fragmenté est d'avoir la possibilité de modifier facilement le fonctionnement entier du robot sans pour autant que la nav stack soit atteinte.



## 10 Création d'un GUI

La création d'une interface utilisateur est intéressante notamment pour la partie développement et test. En effet, le but est d'avoir une interface permettant de choisir le système souhaité après le démarrage. Idéalement il doit être possible de changer de démonstration sans redémarrer la Raspberry Pi 4.

Le touchscreen est un élément intéressant mais pas forcément le plus évident à mettre en place avec une application générant une GUI.

Deux parties principales sont à gérer :

1. L'affichage et le design du GUI (le design importe peu, le fonctionnel est recherché).
2. La récupération des events (zone d'appui du doigt de l'utilisateur).

Plusieurs outils existent pour générer des GUI. Ces derniers sont disponibles dans divers langages mais dans le cadre de ce projet, nous allons nous contenter d'explorer les outils en Python et en C/C++.

Trois outils principaux étaient les plus évidents après des recherches sur la mise en place de GUI avec gestion du touchscreen sur une Raspberry Pi 4.

### 10.1.1 LVGL

Light and Versatile Graphics Library.

Cet outil est intéressant pour des systèmes embarqués. C'est une librairie assez compacte qui ne surcharge pas trop le processeur, ce qui est avantageux dans le cadre d'un projet tel qu'un robot nécessitant de nombreux capteurs pour fonctionner.

Voici un exemple d'interface réalisé avec LVGL :

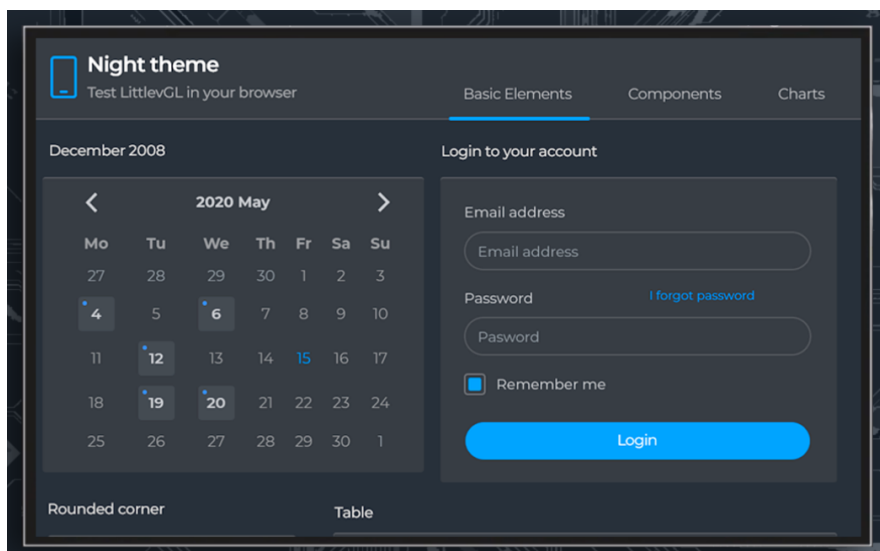


Figure 47 : interface LVGL

Obtenir ce genre de résultat est assez facile avec LVGL. Des repo sur git avec des développements d'interfaces sont trouvable en abondance sur github / gitlab.

Cette librairie fonctionne en C et il existe quelques outils permettant de générer du code (correspondant à l'interface souhaitée) d'après une interface graphique qui permet de la créer. Ces outils n'implémentent en général pas toutes les fonctionnalités de la librairie LVGL mais la plupart des outils nécessaires pour l'application souhaitée.

sont toujours présents. En effet, l'objectif est d'avoir des boutons permettant de réaliser des actions spécifiques (en général l'exécution d'un script).

Malheureusement, le seul point négatif est que nous n'avons pas réussi à récupérer les events (correspondant aux inputs utilisateur avec la pression sur l'écran). L'interface fonctionnait correctement avec la souris cependant.

#### 10.1.2 Kivy

Kivy est une bibliothèque libre et open source pour Python, utile pour créer des applications tactiles pourvues d'une interface utilisateur naturelle.

Le framework contient tous les éléments pour construire des applications et notamment :

- Fonctionnalités de saisie étendues pour la souris, le clavier, les interfaces utilisateurs tangibles, ainsi que les événements multi-touch générés par ces différents matériels.
- Une bibliothèque graphique basée seulement sur OpenGL ES2, et utilisant les Objets Tampons Vertex et les shaders.
- Une large gamme de widgets acceptant le multi-touch.
- Un langage intermédiaire (le Kv3), pour construire facilement des widgets personnalisés.

Cet outil est surtout intéressant pour réaliser des interfaces utilisateurs complexes. Ce système est complexe et gourmand en ressources. C'est un choix intéressant pour la qualité graphique des éléments existants mais utiliser un tel système pour ce projet n'a pas de sens.

#### 10.1.3 Tkinter

Le paquet tkinter (« interface Tk ») est l'interface Python standard de la boîte à outils d'IUG Tk. Tk et tkinter sont disponibles sur la plupart des plates-formes Unix.

C'est un outil léger avec peu de widgets de base. L'API n'est pas la plus intuitive, le résultat avec des codes basiques est en général peu satisfaisant et le « look and feel » n'est pas le plus réussi (un peu ancien).

Voici un exemple d'une interface utilisateur générée par un code utilisant cette librairie.

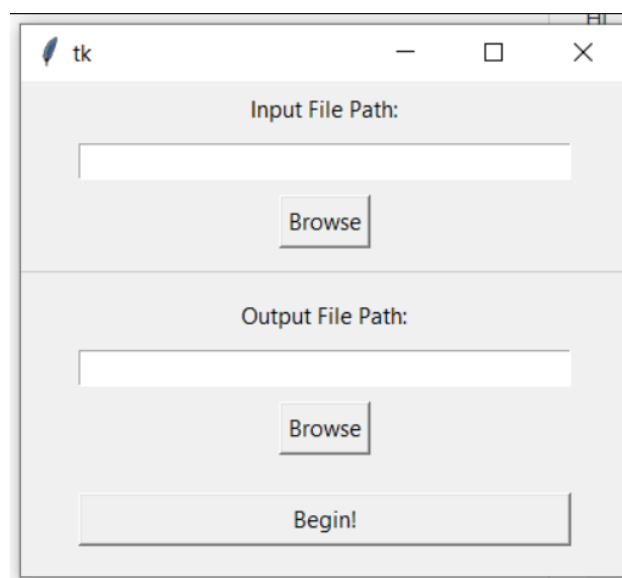


Figure 48 : interface Tkinter3

Tkinter est certainement très adapté pour des développeurs confirmés ayant déjà utilisé cette API, mais n'ayant pas de temps supplémentaire pour analyser et comprendre parfaitement ce que la librairie propose, Tkinter est écarté des choix potentiels.

#### 10.1.4 Pygame

Pygame est une bibliothèque libre multiplateforme qui facilite le développement de jeux vidéos temps réel avec le langage de programmation Python.

Pygame n'est plus utilisée exclusivement pour des jeux vidéos, mais également pour des applications diverses nécessitant du graphisme. La mise en place d'interfaces utilisateurs est possible.

L'intérêt principal de cette librairie est la simplicité pour réaliser une interface rapidement. En effet, quelques lignes de code permettent d'obtenir un fond avec des boutons et des callback liés au boutons.

Cette simplicité permettant le développement d'une GUI en un temps réduit est l'atout majeur de Pygame. Le choix se porte naturellement vers cette librairie du fait que l'interface n'est pas un prérequis pour le produit fini.

Un autre avantage de Pygame est qu'il est possible « d'endormir » l'application qui sera réveillée par un event (pression sur l'écran). Ainsi, la charge sur le processeur est faible si aucune action n'est effectuée sur le touchscreen.

## 11 Démarrage automatique

Ce système a pour but de fonctionner automatiquement lors de la mise sous tension des moteurs et de la Raspberry Pi 4.

Il faut pour cela être capable de lancer tous les scripts et mettre en place les variables d'environnement nécessaires sans utiliser une connexion SSH ou un clavier.

Il existe quelques méthodes pour y parvenir.

### 11.1.1 *Init.d*

C'est un dossier qui se trouve dans `/etc/init.d`.

Init.d ne fait qu'une seule chose, mais il le fait pour tout le système, ce qui le rend très intéressant et important. Il contient un certain nombre de scripts de démarrage / arrêt pour les divers services existants du système. De « acpid » jusqu'à x11-common (gestion console) sont contrôlés à partir de ce répertoire.

Dans le répertoire `/etc` se trouvent en général des répertoires de type `rc#.d` où # correspond à un niveau d'initialisation (allant de 0 à 6). Le fonctionnement précis d'init.d n'est pas l'objet de cette partie, l'intérêt est surtout qu'il est possible d'ajouter son propre script et l'inclure lors du démarrage / arrêt du système.

C'est une option intéressante mais ce n'est pas la plus récente. Elle ne permet pas énormément de contrôle sur les groupes et les utilisateurs.

### 11.1.2 *Systemd*

Systemd est une suite logicielle qui fournit une gamme de composants système pour les systèmes d'exploitation Linux.

Le premier composant de systemd est le système d'initialisation. Il a pour but d'offrir un meilleur cadre pour la gestion des dépendances entre services, de permettre le chargement en parallèle des services au démarrage et de réduire les appels aux scripts shell.

Systemd adopte un nouveau système de log appelé « The Journal », permettant de loguer plus rapidement et plus efficacement les différentes phases de démarrage. Les logs sont authentifiés afin de réduire les chances de corruptions lors d'un piratage. Ils sont portables et l'outil comporte une gestion de saturation de l'espace disque afin de mieux gérer les traces.

Chronologiquement, c'est le dernier outil mis en place pour démarrer automatiquement des scripts au démarrage. C'est aussi un outil plus performant en terme de sécurité et de nombreuses options peuvent être précisées pour personnaliser au mieux le lancement du script :

- Quel utilisateur/groupe exécute le script.
- A quel moment est lancé le script (par rapport aux autres systemd).
- Quel type d'application est lancé (graphique, système...).

Pour la mise en place, il faut créer un nouveau fichier de type `<nom_fichier>.service` et le formater correctement (indentation importante) pour spécifier correctement tous les paramètres souhaités. Pour un exemple de mise en place d'un systemd, se référer au tutoriel d'installation dans les annexes (A[2]).

De plus, il est possible (pour la phase de test du fonctionnement des scripts) de start/stop les différents services créés pour les tester sans avoir à constamment reboot le système.

Il existe aussi un système de statut pour récupérer l'état du service. Cela permet de vérifier si le service est toujours vivant et si l'exécution est celle attendue (sans erreurs).

Enfin, comme mentionné plus tôt, un système de journal existe et permet de récupérer les informations d'exécution. Cela correspond à la sortie Stdout redirigée dans un fichier de log.

Une information importante à prendre en compte lors de la mise en place d'exécution automatique de scripts avec systemd est qu'un service fait l'équivalent du lancement d'un nouveau shell, mais virtuellement. De fait, le script `.bashrc` (script exécuté à l'ouverture d'un nouveau shell) contenant la mise en place de variables d'environnement et de setup de certaines parties locales ne sera pas exécuté automatiquement dans un service. Il n'est pas non plus possible d'exécuter le `.bahrc` depuis le script écrit parce que ce fichier n'existe pas dans le cadre du service.

En conclusion de cette analyse, il est important d'être conscient des variables d'environnement utilisées et nécessaires pour l'application lancée. Ces variables doivent être exportées par l'utilisateur au préalable dans le script pour être présentes dans l'environnement créé par le service.

C'est un outil puissant, sécurisé et efficace pour le lancement de scripts à l'initialisation. Cependant, l'outil n'est pas évident à prendre en main.

Systemd est l'outil principal utilisé dans le projet pour lancer automatiquement le système. Les fichiers ajoutés se trouvent dans `/etc/systemd/system/` et sont de type `rbt_*.service`.

### 11.1.3 *Rc.local*

Ce fichier était très utilisé avant l'introduction des outils `init.d` puis `systemd`. Cependant, le principe est le même. Ce fichier permet d'exécuter des commandes (qui peuvent être l'exécution d'un script...) au démarrage. Cependant, `rc.local` est un peu différent car c'est le dernier script exécuté et il exécute les commandes dans l'ordre renseigné dans le fichier.

Dans les dernières distributions d'Ubuntu, ce fichier n'est pas forcément présent (car moins utilisé). Il faut donc le créer soi-même et le formater correctement pour qu'il soit appelé correctement. Un exemple de `rc.local` est fourni dans le tutorial présent dans les annexes (A[2]).

Toutes les commandes avant « `exit 0` » seront exécutées.

Le `rc.local` se trouve dans `/etc/rc.local`. Ce projet utilise ce fichier pour initialiser le bluetooth et certains droits.

### 11.1.4 *Améliorations du lancement du GUI*

Après avoir mis en place l'initialisation du `rosmaster` (avec la commande `roscore` lancée automatiquement au boot), il ne manquait plus qu'à lancer la GUI permettant de choisir le démarrage de la démo avec contrôleur ou la démo de navigation autonome.

L'utilisation de `systemd` est aussi tout à fait fonctionnelle mais la seule manière de démarrer le GUI est de le lancer avec l'utilisateur/groupe `root`. C'est fonctionnel mais ce n'est pas forcément élégant et c'est une amélioration qui peut être apportée dans le futur.

## 12 Conclusion

### 12.1 Synthèse

L'objectif du projet était de proposer un robot autonome permettant de se déplacer dans un environnement extérieur avec une précision de localisation de l'ordre du centimètre. Aujourd'hui, le système autonome est en place et fonctionnel en intérieur.

Voici l'état actuel du projet :

- Un robot pour tester le déplacement autonome est monté.
- Ce robot peut être commandé avec une manette PS4 pour d'autres démonstrations.
- La structure de base du code ROS est mise en place. Il suffit de modifier les paramètres et les données envoyées pour que le système évolue.
- Le système mis en place est reproductible et un tutoriel complet est fourni pour récupérer un environnement fonctionnel rapidement.
- Tous les capteurs sont intégrés dans le workflow de ROS. Il est possible de modifier les paramètres de lancement des nœuds pour modifier le comportement des capteurs.
- Le robot génère une map ou bien peut récupérer une map existante générée. Cette map est mise à jour continuellement lors des déplacements du robot.
- Une odométrie fictive est générée d'après les capteurs disponibles sur le robot.
- Le robot est capable d'esquiver les obstacles sur son chemin et de recalculer un itinéraire si nécessaire.
- Une interface utilisateur est disponible permettant de démarrer le système sans système tiers.

### 12.2 Futur du projet

Pour le futur du projet, il serait intéressant d'améliorer certains aspects :

- Utilisation plus poussée de la caméra pour récupérer certaines informations supplémentaires utiles à la navigation autonome (à explorer).
- Fusion d'un maximum de données pour augmenter la précision de l'odométrie fournie (filtre EKF, AMCL...).
- Ajouter un système permettant de récupérer l'odométrie directement depuis des capteurs sans en générer une fictive. Même si elle est fautive, la corriger n'est pas un problème.
- Porter le système en extérieur avec l'utilisation du GPS.
- Réflexion sur la génération automatique d'une map d'après des coordonnées GPS données.
- Intégrer au système le traitement d'images permettant de repositionner le robot correctement d'après un planton (fusion avec le TB de Mr. Favre).

### 12.3 Commentaires personnels

Ce travail de diplôme a été pour moi une opportunité de m'immiscer dans le monde fantastique de la robotique. Depuis le concours de robotique proposé par l'école aux élèves d'informatique embarquée auquel j'ai eu la chance de participer, j'ai développé une passion pour ce monde.

En effet, la mise en place de systèmes démontrant une forme d'intelligence et de prise de décision est simplement incroyable.

Ce projet a cependant été très compliqué. Le fait d'avoir défini la mécanique, l'électronique et le logiciel du robot était un défi conséquent à relever. J'ai pu approcher de nombreux concepts encore inconnus pour moi du domaine de l'électronique. La familiarisation avec certains outils importants (oscilloscopes, voltmètres, alimentations...) est une compétence formidable pour ma vie professionnelle à venir dans le monde de l'embarqué.

Avoir pu travailler (plutôt vers la fin du projet) en équipe avec un autre diplômé m'a vraiment motivé, le partage d'idées et de connaissances est vraiment un aspect que je valorise énormément et qui est bénéfique pour tous les membres d'un groupe.

N'ayant eu aucun appui pendant la mise en place du logiciel avec ROS, j'ai expérimenté l'autonomie la plus totale concernant la résolution des problèmes et la création d'un projet conséquent. La formation embarquée proposée par l'HEIG-VD m'a permis de me confronter aux différents problèmes et de les surmonter en les analysant et en imaginant la solution la plus cohérente avec mes besoins.

Enfin, j'ai pris beaucoup de plaisir avec ce système que je pouvais manipuler et tester. Je pense avoir appris de nombreux concepts fondamentaux en robotique mais aussi en informatique embarquée en général. Je pense être prêt pour entamer une carrière dans le monde professionnel au sein d'une équipe.

## 12.4 Remerciements

Pour commencer, je tiens à remercier l'équipe du REDS qui m'a aidé et soutenu au long du projet, à savoir Mr. Etienne Messerli, Mr. Jean-Pierre Miceli et Mme Eliéva Pignat notamment.

Un grand remerciement à l'équipe de la mécanique qui m'a aidé à mettre sur pied le robot démonstrateur (Mr. Steve Maillard, Mr. Christophe Maendly).

Je tiens à remercier les élèves qui m'ont apporté des solutions dans des domaines qui n'étaient pas forcément évidents pour moi (Sugène Pant, Rapha Da Cunha Garcia, Isaia Spinelli).

De plus, ma famille m'ayant permis de reprendre des études et de financer mon cursus a été un point central de ma réussite. Mention spéciale à ma femme et aux efforts qu'elle a fourni durant ces trois années.

Enfin, je tiens à remercier tous les restaurants japonais avec une formule à volonté qui ont nourrit mon cerveau fatigué et mes chats qui ont supporté les mauvaises humeurs (et ont continué de fournir une dose de câlins plus que satisfaisante) lors de certaines périodes de frustration ayant pour cause le non-fonctionnement ou la non compréhension du système.

## Table des figures

Figure 1 : Exemple d'odométrie faussée .....	11
Figure 2 : GNSS schéma constellation/visibilité .....	12
Figure 3: Schéma centrale inertielle .....	13
Figure 4 : Schéma correction RTK .....	13
Figure 5 : base du robot (Prowler).....	17
Figure 6 : moteurs DC .....	17
Figure 7 : Raspberry pi 3 .....	18
Figure 8 : Raspberry pi 4 .....	19
Figure 9 : tinker board .....	19
Figure 10 : roboclaw 2x30A .....	20
Figure 11 : HB-25 Parallax.....	21
Figure 12 : Cytron MD10C .....	21
Figure 13 : batterie Li-ion .....	22
Figure 14 : voltage converter.....	22
Figure 15 : PCA9685.....	23
Figure 16 : touchscreen .....	24
Figure 17 : rplidar A1 .....	24
Figure 18 : mti-7 DK (IMU/GNSS).....	25
Figure 19 : D435i (caméra RealSense) .....	25
Figure 20 : ds4 controller .....	26
Figure 21 : graphique chute de tension.....	26
Figure 22 : batterie supplémentaire Aukey PB-N52 .....	27
Figure 23 : agencement étage 1 .....	29
Figure 24 : fixation du Lidar .....	30
Figure 25 : mesures utiles du Lidar .....	30
Figure 26 : mesures châssis - Fixation Lidar.....	31
Figure 27 : Schéma de connexion électronique .....	32
Figure 28 : liaisons système complet.....	33
Figure 29 : Donkey Car.....	37
Figure 30 : PWM valeur basse .....	40
Figure 31 : PWM valeur 50% .....	40
Figure 32 : mesure sortie GPIO avec voltmètre .....	41
Figure 33 : RPLidar points - Rviz .....	43
Figure 34 : configuration mti-7 DK .....	44
Figure 35: visualisation des accélérations .....	46
Figure 36 : visualisation de la position angulaire .....	46
Figure 37 : IMU axes .....	46
Figure 38 : white point cloud.....	47
Figure 39 : colored point cloud.....	47
Figure 40 : Schéma navigation stack .....	48
Figure 41 : reference frame .....	49
Figure 42 : visualisation des frames .....	49
Figure 43 : équation tf .....	50
Figure 44 : position d'un objet.....	50
Figure 45 : Rviz map hector_slam.....	53
Figure 46 : map ouverte dans un éditeur .....	54
Figure 47 : interface LVGL.....	56
Figure 48 : interface Tkinter3 .....	57



## 13 Bibliographie

- [1] «Explication des concepts principaux de ROS,» [En ligne]. Available: <https://blog.generationrobots.com/fr/ros-robot-operating-system-3/>.
- [2] «Site web et références de l'agriculteur mandataire,» [En ligne]. Available: <https://bovigny.ch/ferme-bio-du-moulin/>.
- [3] «Resource principale regroupant la documentation complète de ROS,» [En ligne]. Available: <https://wiki.ros.org/>.
- [4] «Explications du fonctionnement de la correction RTK,» [En ligne]. Available: <https://www.vboxautomotive.co.uk/index.php/en/how-does-it-work-rtk>.
- [5] «Explication du fonctionnement de l'odométrie,» [En ligne]. Available: <https://fr.wikipedia.org/wiki/Odom%C3%A9trie>.
- [6] «Listes de centrales inertielles low cost pour la robotique,» [En ligne]. Available: <https://damien.douxchamps.net/research/imu/>.
- [7] «Détails du fonctionnement du package RPLidar,» [En ligne]. Available: <http://wiki.ros.org/rplidar>.
- [8] «Détails sur la transformation et la modification des informations du scan,» [En ligne]. Available: [http://wiki.ros.org/action/fullsearch/laser\\_filters?action=fullsearch&context=180&value=linkto%3A%22laser\\_filters%22](http://wiki.ros.org/action/fullsearch/laser_filters?action=fullsearch&context=180&value=linkto%3A%22laser_filters%22).
- [9] «Marche à suivre de l'intégration de la D435i dans ROS,» [En ligne]. Available: <https://github.com/IntelRealSense/realsense-ros/wiki/SLAM-with-D435i>.
- [10] «Détails théoriques de l'implémentation de la Navstack avec IMU/GNSS sans odométrie,» [En ligne]. Available: [https://roscon.ros.org/2018/presentations/ROSCon2018\\_INSNavstack.pdf](https://roscon.ros.org/2018/presentations/ROSCon2018_INSNavstack.pdf).
- [11] «Intégration de l'IMU dans la Navstack,» [En ligne]. Available: [https://cw.felk.cvut.cz/w/\\_media/misc/projects/nifti/sw/ins.pdf](https://cw.felk.cvut.cz/w/_media/misc/projects/nifti/sw/ins.pdf).
- [12] «Package map ROS utilisé pour la navigation avec la d435i,» [En ligne]. Available: [http://wiki.ros.org/rtabmap\\_ros](http://wiki.ros.org/rtabmap_ros).
- [13] «Solution alternative de SLAM avec la d435i,» [En ligne]. Available: <http://wiki.ros.org/rgbdslam>.
- [14] «Détails de la navigation autonome sur ROS,» [En ligne]. Available: <http://wiki.ros.org/navigation>.
- [15] «Cours explicatif de la navigation autonome sur ROS,» [En ligne]. Available: <https://www.udemy.com/course/ros-navigation/learn/lecture/14971586#overview>.
- [16] «Explication des conventions sur les axes,» [En ligne]. Available: [https://en.wikipedia.org/wiki/Local\\_tangent\\_plane\\_coordinates](https://en.wikipedia.org/wiki/Local_tangent_plane_coordinates).

- [17 «Documentation sur le package pour la board PCA9685,» [En ligne]. Available: <http://bradanlane.gitlab.io/ros-i2cpwmboard/>.
- [18 «Package ROS de base pour la conduite différentielle,» [En ligne]. Available: [http://wiki.ros.org/diff\\_drive\\_controller](http://wiki.ros.org/diff_drive_controller).
- [19 «Informations concernant le mti-7 DK IMU/GNSS,» [En ligne]. Available: <https://www.xsens.com/products/mti-1-series>.
- [20 «Informations concernant le Framework permettant de générer une interface utilisateur pour le touchscreen,» [En ligne]. Available: <https://www.pygame.org/news>.
- [21 «Documentation permettant l'utilisation de la simulation avec Gazebo,» [En ligne]. Available: <https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#simulation>.

## 14 Documents supplémentaire

Dans le repo gitlab 2020\_catel se trouvent des documents cités dans le rapport.

La référence aux annexes correspond à : « A[N] ». Les annexes se trouvent dans 2020\_catel/publi/annexes.

La référence aux divers documents (datasheet, manuel d'utilisation...) correspond à : « D[N] ». Les documents se trouvent dans 2020\_catel/doc/doc\_système\_final