

Projet Héron

DOCUMENTATION GLOBALE

# HERON

# Le Robot

ISEN-Lille

Robotique Mobile

## Documentation globale



1	Introduction.....	7
1.1	Contexte de projet .....	7
1.2	Objectifs du projet.....	7
1.2.1	Cahier des charges.....	8
2	Description du démonstrateur d'industrie 4.0.....	10
2.1	Fonctionnalités .....	10
2.2	Spécification globale.....	11
2.2.1	Présentation Niryo.....	12
2.2.2	Présentation Yumi .....	14
2.2.3	Présentation Banc d'assemblage .....	15
2.2.4	Présentation Robot Héron.....	16
2.2.5	Présentation PC central.....	17
3	Conception et réalisation du robot Héron .....	18
3.1	Introduction du robot Héron.....	18
3.1.1	Fonctionnalités attendues de la part du robot .....	18
3.1.2	Schéma d'interaction.....	19
3.1.3	Schéma Hardware .....	20
3.1.4	Organigramme Software .....	21
3.2	Conception mécanique.....	22
3.2.1	Principales étapes de constructions de la base mobile.....	24
3.2.2	Principales étapes de construction de la glissière.....	25
3.2.3	Principales étapes de l'ajout des carters.....	27
3.2.4	Composition du robot du point de vue global .....	31
3.3	Alimentation du robot.....	32
3.3.1	Batterie et chargeur .....	32
3.3.2	Distribution principale et allumage du robot .....	34
3.3.3	Distribution secondaire .....	35
3.3.4	Bilan de consommation.....	36
3.4	Contrôle bas niveau du robot.....	37
3.4.1	Présentation des composants et de l'électronique assurant l'asservissement de la base mobile du robot.....	37
3.4.1.1	Roues .....	37
3.4.1.2	Moteurs .....	37
3.4.1.3	Encodeurs.....	38
3.4.1.4	Drivers Roboteq.....	39
3.4.2	Développement asservissement .....	40

3.4.2.1	Régulateur bas niveau du PID.....	40
3.4.2.1.1	Modélisation du moteur .....	40
3.4.2.1.2	Choix du régulateur .....	40
3.4.3	Asservissement de la glissière .....	41
3.4.3.1	Présentation des composants et de l'électronique assurant l'asservissement de la base mobile du robot .....	41
3.4.3.2	Développement .....	45
3.4.3.3	Tuto .....	47
3.5	Contrôle haut niveau.....	48
3.5.1	Environnement du contrôle haut niveau .....	48
3.5.2	Composants sur le robot permettant les applications haut niveau .....	49
3.5.2.1	Carte mère Jetson.....	49
3.5.2.2	Capteurs .....	50
3.5.2.2.1	Justification des capteurs .....	50
3.5.2.2.2	Placement des capteurs .....	51
3.5.2.2.3	Organisation des programmes et modularité .....	52
3.5.2.2.4	Lidar .....	53
3.5.2.2.5	IMU .....	56
3.5.2.2.6	Capteurs Infrarouge .....	57
3.5.2.2.7	Camera .....	59
3.5.2.2.8	Capteur de Coulomb .....	59
3.5.2.2.9	Ecran.....	61
3.5.3	Applications Haut Niveau du Robot Héron .....	61
3.5.3.1	Serveur ROS .....	61
3.5.3.2	IHM de contrôle et organigramme Software associés .....	62
3.5.3.3	Découverte des différents modes et fonctionnalités du robot.....	66
3.5.3.3.1	Mode « Remote control » .....	66
3.5.3.3.2	Mode « Mapping » : .....	72
3.5.3.3.3	Mode « Take Key Position » .....	76
3.5.3.3.4	Mode « Edit Key Positions » : .....	82
3.5.3.3.5	Mode « Navigation » : .....	84
3.5.3.4	Détails des programmes haut niveau.....	92
3.5.3.4.1	Bringup.launch .....	92
3.5.3.4.2	Lidar.launch .....	94
3.5.3.4.3	Lpms_imu .....	95
3.5.3.4.4	Tf_broadcaster .....	95

3.5.3.4.5	Navigation.launch.....	97
3.5.3.5	Digital Twin.....	99
3.5.3.5.1	Fichier URDF .....	100
3.5.3.5.2	URDF du robot Héron.....	101
3.5.3.5.3	Fichier display.launch .....	107
3.5.3.6	Bouton d'arrêt d'urgence .....	111
3.5.3.6.1	Développement.....	112
3.5.3.6.2	Tutoriel .....	112
3.5.3.7	Détection de fin de stock.....	114
3.5.3.8	Tutoriels pour assurer la commande haut-niveau .....	115
3.5.3.8.1	Installation Ubuntu 16.04.....	115
3.5.3.8.2	Installer ROS Kinetic .....	115
3.5.3.8.3	Intégration IMU dans Jetson TX2 .....	116
3.5.3.8.4	Untégration du convertisseur CAN to USB pour les IR.....	116
3.5.3.8.5	Intégrer Manette Xbox.....	117
3.5.3.8.6	Fixation port USB pour les périphériques USB.....	120
4	Serveur central et communications .....	122
4.1	Fonctionnalités .....	123
4.2	Spécifications globales .....	123
4.2.1	Spécifications serveur et tests.....	124
4.2.1.1	Spécifications.....	124
4.2.1.2	Développement.....	125
4.2.1.2.1	Initialisation.....	125
4.2.1.2.2	Base de données .....	127
4.2.1.3	Test .....	131
4.2.2	Spécifications clients et tests .....	133
4.2.2.1	Spécification .....	133
4.2.2.2	Développement .....	134
4.2.2.2.1	Initialisation.....	134
4.2.2.2.2	Utilisation .....	134
4.2.2.2.3	Installation.....	136
4.2.2.2.4	Tutoriel créer un client.....	136
5	Gestion de projet.....	139
5.1	Composition de l'équipe .....	139
5.2	Organisation interne et gestion de projet.....	139
5.2.1	Planning .....	140

5.2.2	Gestion Prévisionnelle des Emplois et des Compétences (GPEC) .....	141
5.2.3	Cycle en V et Planning de Production Documentaire (PPD).....	141
5.2.4	Graphe de performance .....	143
5.2.5	Animation du projet (réunion d'équipe, réunion 5 min).....	144
5.2.6	Outils de partage, communication et échange .....	144

## **GITHUB**

<https://github.com/yncreahdf-robotics>

## **DRIVE**

[https://yncrea-my.sharepoint.com/personal/gilles\\_tagne\\_yncrea\\_fr/\\_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fgilles%5Ftagne%5Fyncrea%5Ffr%2FDocuments%2FProjetM2%5FRobotique%5F2019%5F2020&ct=1581582174852&or=OWA-NT&cid=aa291869-d839-db98-6d12-6b50d54aeb07&originalPath=aHR0cHM6Ly95bmNyZWEtbXkuc2hhcmVwb2ludC5jb20vOmY6L2cvcGVyc29uYWwvZ2lsbGVzX3RhZ25IX3luY3JIYV9mci9FcIg3TWxka2JKaE1qekwxR2RZU0FOVUJQMDIBamF0QXVFb2RQeExCano5VEVBP3J0aW1IPWI Bczg3VjJ3MTBn](https://yncrea-my.sharepoint.com/personal/gilles_tagne_yncrea_fr/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fgilles%5Ftagne%5Fyncrea%5Ffr%2FDocuments%2FProjetM2%5FRobotique%5F2019%5F2020&ct=1581582174852&or=OWA-NT&cid=aa291869-d839-db98-6d12-6b50d54aeb07&originalPath=aHR0cHM6Ly95bmNyZWEtbXkuc2hhcmVwb2ludC5jb20vOmY6L2cvcGVyc29uYWwvZ2lsbGVzX3RhZ25IX3luY3JIYV9mci9FcIg3TWxka2JKaE1qekwxR2RZU0FOVUJQMDIBamF0QXVFb2RQeExCano5VEVBP3J0aW1IPWI Bczg3VjJ3MTBn)

# 1 Introduction

## 1.1 Contexte de projet

Actuellement en pleine évolution technologique, il est aisément de constater que l'implémentation et l'intégration de robots dans notre vie quotidienne est une chose de plus en plus commune : voitures autonomes, automates ... Une des dernières nouveautés est de chercher à développer les « industries du futur ». Ces industries chercheraient à faire collaborer les robots entre eux mais également avec les humains afin de d'apporter une aide dans certaines tâches répétitives ou pouvant être dangereuses pour la santé des humains.

Ce projet d'industrie du futur est actuellement en pleine évolution et cela même dans les locaux d'YNCREA (HEI, ISA, ISEN). En effet l'association Yncréa souhaite développer leur propre démonstrateur d'industrie 4.0 Yncréa. Pour cela, Yncréa a su obtenir un local réservé pour ce futur démonstrateur ainsi que de nombreux robots tel que Yumi, Niryo, un banc mécanique etc.

L'objectif est donc le même qu'évoqué précédemment : faire travailler les différents robots en collaboration sans forcément nécessiter d'intervention humaine.

## 1.2 Objectifs du projet

Construction d'un robot mobile qui s'intégrera au démonstrateur d'industrie 4.0 d'Yncréa Hauts De France. Construire et intégrer notre propre base mobile afin de développer l'idée d'industrie du futur en développant à petit échelle un démonstrateur de plusieurs robots reproduisant un cycle de production « type usine ».

Développer un réseau de communications permettant l'échange d'informations entre les différentes entités du démonstrateur.

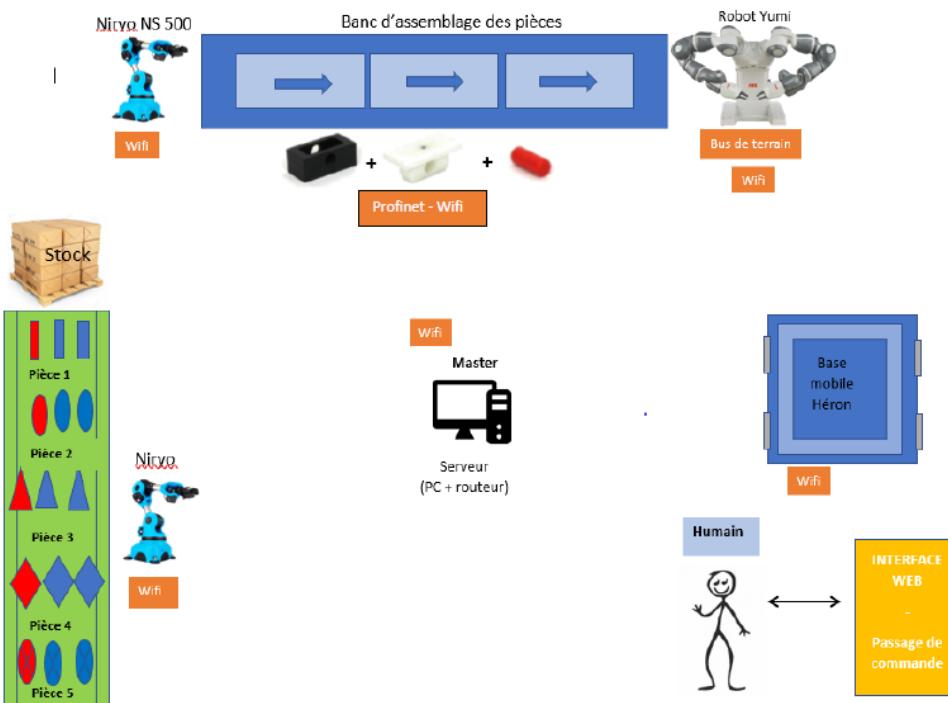


Figure 1 Schéma démonstrateur industrie 4.0 Yncréa HDF

### 1.2.1 Cahier des charges

Vous trouverez ci-dessous les fonctionnalités attendues et fixées lors de l'élaboration du cahier des charges. Celles-ci concernent le développement du robot Héron en lui-même ainsi que le développement du serveur de communication.



Figure 2 Fonctionnalités attendues robot Héron

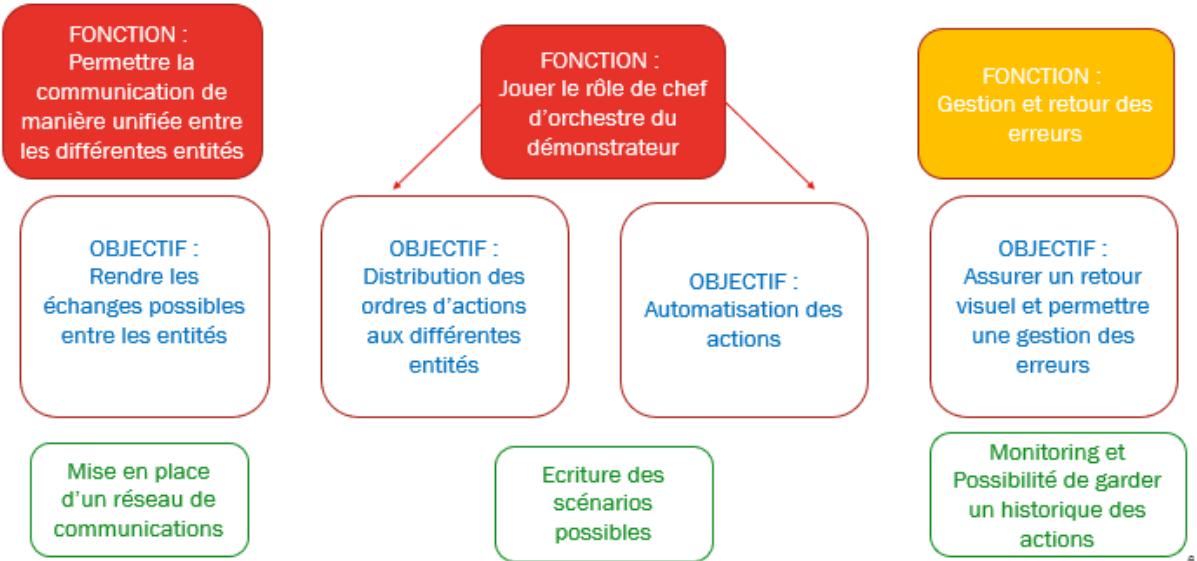


Figure 3 Fonctionnalités attendue serveur de communication

Plusieurs contraintes ont été énoncées en début de projet et qui régiront la suite de notre projet. Voici la liste de ces contraintes :

- Autonomie évaluée autour de 5h avant rechargement
- Recharge manuelle
- Attention Architecture électronique du robot, réalisation d'un robot à but industriel → PAS D'ARDUINO → Utilisation de Raspberry, cartes et contrôleurs existants
- Moteurs et commandes bas niveaux réutilisables pour la coupe de France de robotique
- Utilisation et intégration de plusieurs capteurs :
  - \_ Camera (vision, traitement d'image)
  - \_ IMU (position, orientation du robot)
  - \_ Odomètre (Vitesse du robot)
  - \_ Lidar (Détection longue portée)
  - \_ US, IR (Détection faible portée)
  - \_ Radar
- Adaptation du plateau qui viendra recueillir le matériel transmis.
- Modularité Hardware et Software afin de garantir une adaptation facile face aux demandes du client
- Environnement Linux – Ubuntu 16.04
- Utilisation ROS Kinetic pour la gestion des capteurs et leurs données
- Mise en place de notre propre réseau de communication
- Charge minimale à porter : 10 kg
- Communication non-filaire entre les robots et le serveur

**Une fois ces contraintes posées, certains accords ont été trouvés quant à l'environnement dans lequel nous serions amenés à travailler tout au long du projet mais également à propos du champ de responsabilités de l'équipe Héron.**

\_ Entités présentes dans le démonstrateur : 2 robots NIRYO, 1 robot YUMI, 1 banc mécanique, 1 robot Héron, 1 interface Web.

\_ Scénario et communications :

1) L'équipe en charge de créer la base mobile Héron est libre de proposer ses propres solutions quant à l'élaboration des différents scénarios possibles et donc de déterminer les communications et les types de messages qui transiteront entre les entités.

Cela implique donc une liberté quant à la manière d'approvisionner le banc de production, de gérer les déplacements de notre robot.

2) L'équipe Héron n'a pas à intervenir directement sur les Niryo, Yumi ou sur le banc mécanique. Cependant nous devons nous assurer que les messages que nous souhaitons envoyer/recevoir depuis/vers ces entités soient correctes et exploitables.

Pour cela, au cours du déroulement de nos scénarios test, nous confirmerons cela en simulant le fonctionnement de ces entités.

## 2 Description du démonstrateur d'industrie 4.0

### 2.1 Fonctionnalités

#### Numérisation de l'usine 4.0

L'Industrie 4.0 correspond en quelque sorte à la numérisation de l'usine. À travers le recours à l'internet des objets et aux systèmes robotisés, l'usine intelligente se caractérise par une communication continue et instantanée entre les différents outils et postes de travail intégrés dans les chaînes de production et d'approvisionnement. L'utilisation de capteurs communicants apporte à l'outil de production une capacité d'auto-diagnostic et permet ainsi son contrôle à distance tout comme sa meilleure intégration dans le système productif global.

#### Flexibilité de l'usine 4.0 et personnalisation de la production

En proposant des sites de production composés d'objets intelligents, communicants et liés dans un réseau lui-même relié à l'extérieur, la flexibilité de la production peut être améliorée. Il est donc possible de proposer une production à la fois à grande échelle et personnalisée.

#### Gestion des ressources et des matières premières

L'Industrie 4.0 représente aussi une volonté de répondre aux problématiques actuelles de la gestion des ressources et de l'énergie. Avec un système organisé selon un réseau de communication et d'échange instantané et permanent, on est à même de rendre cette gestion plus efficace en coordonnant les besoins et disponibilités de chaque élément du système de la façon la plus efficiente possible, alimentant par-là de nouveaux gains de productivité.

## 2.2 Spécification globale

Le démonstrateur d'industrie Yncréa HDF est situé au niveau des bâtiments de l'école d'HEI, face à l'ISA et à seulement 2 minutes à pied de l'ISEN.



Figure 4 Local du démonstrateur d'industrie 4.0 Yncréa HDF

Bien que notre objectif soit de développer une base mobile, nommée Héron, que l'on viendra intégrer à un démonstrateur d'industrie du futur, nous pouvons constater que celui-ci devra interagir avec de nombreuses entités.

Nous pouvons constater que le démonstrateur sera composé de divers robots :

- \_ 1 robot Niryo situé à l'entrepôt de stockage qui sera chargé de fournir les pièces nécessaires à la production.
- \_ 1 robot Niryo situé au niveau de la ligne de production de pièces avec pour objectif de décharger une base mobile et de charger le banc mécanique.
- \_ 1 banc mécanique qui assurera l'assemblage des pièces
- \_ 1 robot Yumi qui sera chargé d'emballer les produits assemblés et finis
- \_ 1 robot mobile Héron chargé de transporter les pièces d'un poste à un autre
- \_ Une interface Web par laquelle le client pourra personnaliser sa commande
- \_ Un PC-serveur qui assurera la bonne communication entre toutes ces entités et la gestion automatique d'un cycle de production complet.

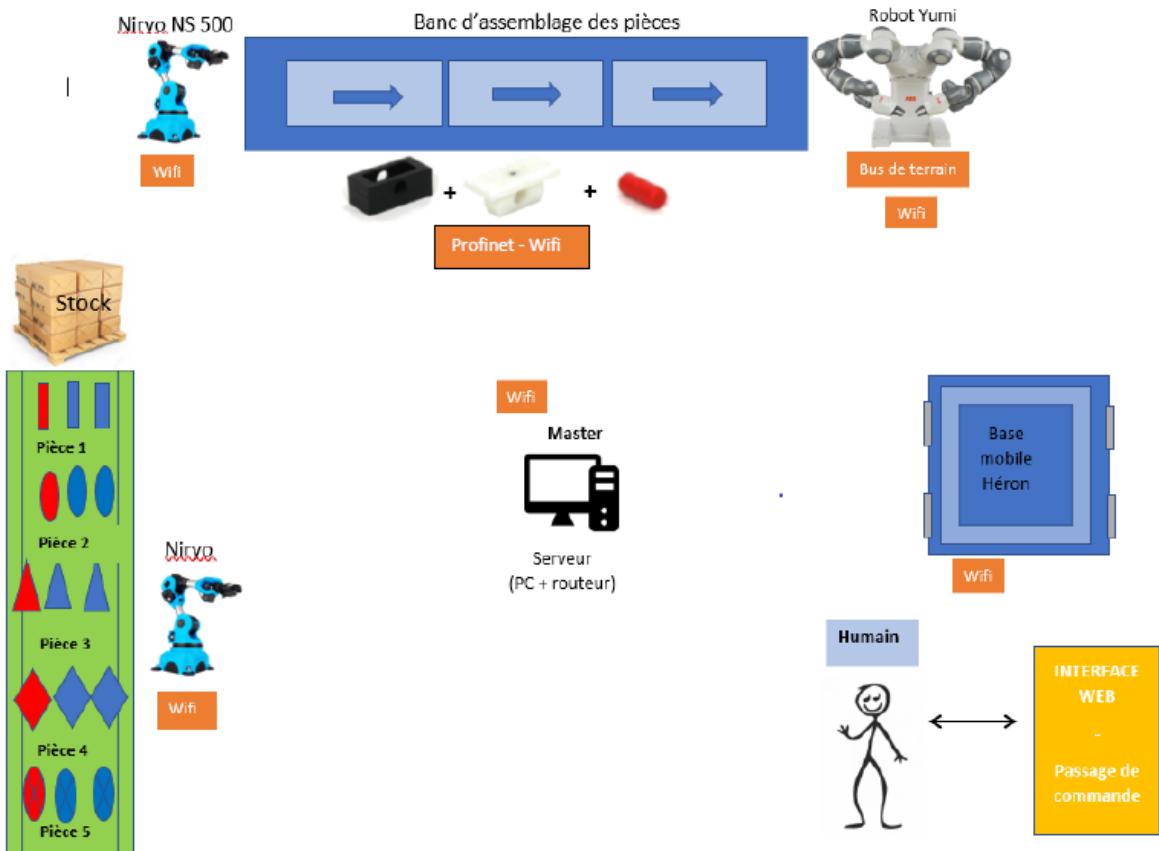


Figure 5 Schéma démonstrateur industrie 4.0 Yncréa HDF

### 2.2.1 Présentation Niryo

Niryo est un bras robotique collaboratif open-source développé par deux anciens étudiants de l'ISEN.

**Poids du robot** : 6kg

**Nombre d'axes et degrés de liberté** : 6 axes

**Matériaux** : Aluminium, PLA (impression 3D)

**Systèmes électroniques** : Arduino, Raspberry

**Système annexe** : Robot Operating System (ROS)

**Communication** : Wifi, Ethernet, USB Port

**Interface de programmation** : Interface visuelle Niryo Black de type Blocky (similaire au Scratch)

**Motorisation** : servomoteurs Dynamixel, moteurs pas à pas

**Service apporté** : Prise de pièces de poids de 500g max, dans un rayon d'action de 40cm.

**Caractéristiques** : Prise de différentes pièces grâce à l'existence de plusieurs types de pinces.

**Système de contrôle** : Manuel, via manette de XBOX ou de Playstation, via une application WEB/Mobile



Figure 6 Robot Niryo

En quelques mots, ces robots Niryo sont de parfait manipulateur d'objets (de petite taille pour l'instant). Ils auront pour but d'assurer une dépose ou un retrait de pièces situées dans une étagère de stockage ou alors directement sur le robot mobile Héron.

Pour obtenir plus d'informations à propos du robot Niryo, rendez-vous directement sur leur site <https://niryo.com/fr/>.

Vous trouverez ci-dessous certaines spécifications du bras Niryo.

1 - Base	Orange
2 - Shoulder	Yellow
3 - Arm	Green
4 - Elbow	Turquoise
5 - Forearm	Blue
6 - Wrist	Purple
7 - Hand	Red

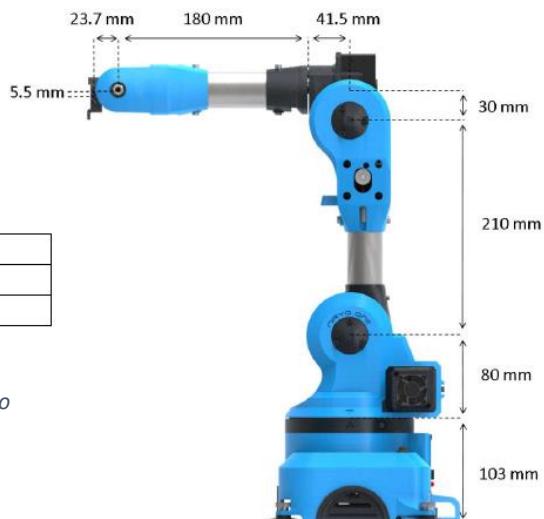


Figure 7 Spécifications robot Niryo

## 2.2.2 Présentation Yumi

Présentation officielle du robot par le constructeur ABB.

« Un robot à deux bras, innovant et convivial, possédant des fonctionnalités avancées, qui offre un large potentiel de fonctions d'automatisation supplémentaires à l'industrie. YuMi® ouvre une nouvelle ère d'automatisation, par exemple dans l'assemblage de petites pièces, où hommes et robots travaillent main dans la main et d'exécuter des tâches identiques tout en garantissant leur sécurité. La sécurité est au cœur même de sa conception, ce qui lui permet de travailler sans cage. »

Le robot Yumi est donc présenté comme une toute nouvelle génération de Cobot, doté de 2 bras et de mains flexibles. Celui-ci est capable de manipuler et assembler des pièces entre elle grâce à sa forte mobilité et précision. Il est doté d'un système universel d'alimentation en pièces, d'une vision intégrée ou encore d'une gestion de trajectoire à la pointe de la technologie, YUMI est le parfait robot manipulateur pour réaliser du travail de précision.



Figure 8 Robot Yumi

Pour obtenir plus d'informations à propos du robot Niryo, rendez-vous directement sur leur site <https://new.abb.com/products/robotics/fr/robots-industriels/yumi>.

**Poids :** 35kg

**Nombre d'axes et degrés de liberté :** 14 axes

**Composition du robot :** Bras en magnésium, léger et rembourré

**Vitesse d'exécution :** 1500mm par seconde

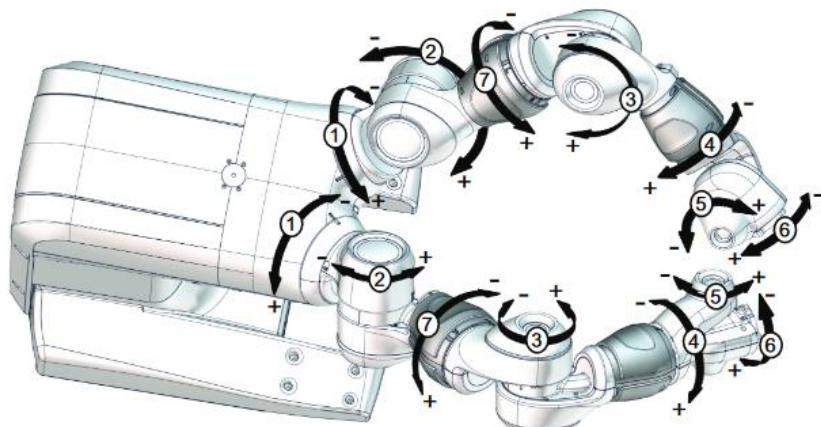
**Communication :** Ethernet, Profibus, USB port

**Interface de programmation :** Interface visuelle type Blockly (similaire au Scratch)

**Service apporté :** Prise de pièces de poids de 500g max sur un rayon de 50cm / Assemblage de pièces pour les industries de précision

**Caractéristiques :** Prise de différentes pièces grâce à l'existence de différents types de pinces, système de prise (Yumi est capable d'enfiler un fil dans une aiguille)

Vous trouverez ci-dessous certaines spécifications du robot YUMI.



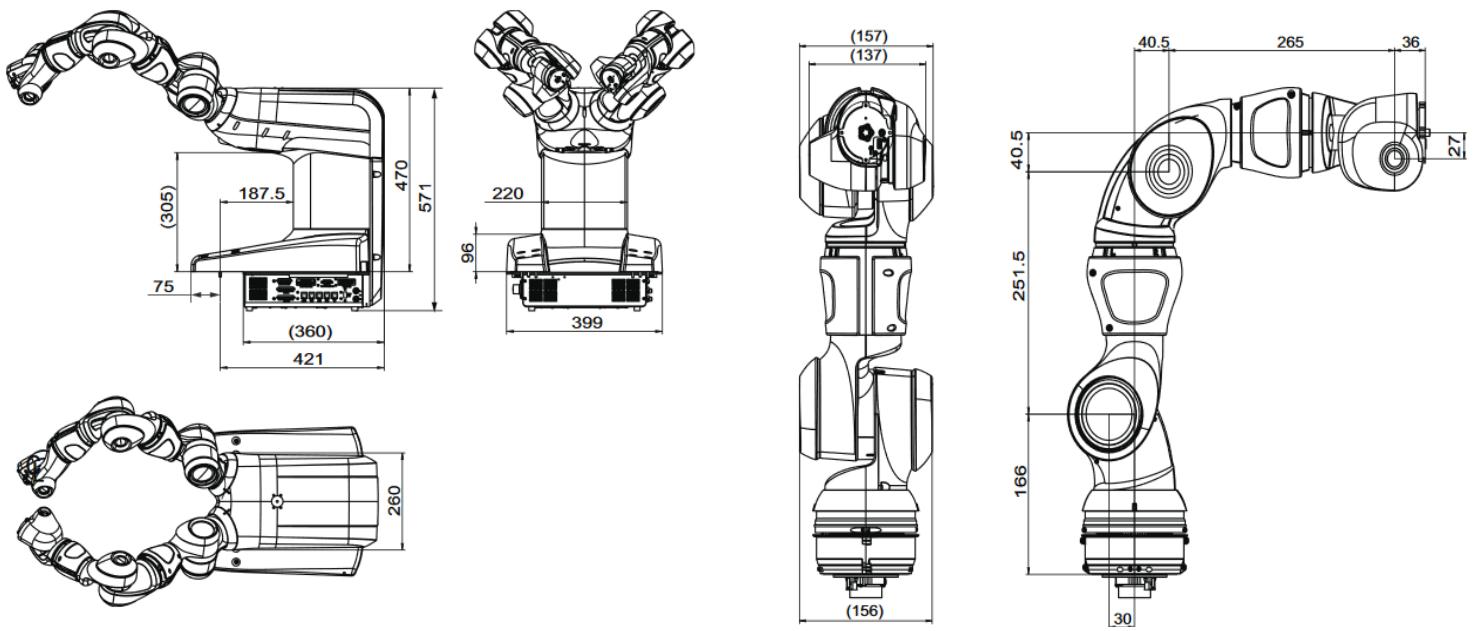


Figure 9 Spécifications robot Yumi

### 2.2.3 Présentation Banc d'assemblage

Description officielle du constructeur :

*« Module mécatronique de base entraîné par un motoréducteur 24 V à vitesse variable et complet avec capteurs de fin de course et module PROFIBUS DP intégré. Conçu pour des expériences de base sur un système de convoyeur ou pour l'incorporation dans un système mécatronique complexe pour contrôler le flux de matériaux. La bande transporteuse transporte les pièces sur les supports et peut être utilisée pour relier des sous-systèmes individuels. »*

*Il est conçu pour être connecté à un système de contrôle PLC. Il peut être combiné avec d'autres bandes transporteuses, unités « courbes » ou jonctions de transfert. Les stations IMS peuvent être connectées directement à la bande et contrôlées conjointement via PROFIBUS. »*

En quelques mots, le banc d'assemblage, comme son nom l'indique, aura pour mission d'assembler les pièces entre elles. Son mode de communication est le « PROFIBUS ».

**Dimensions :** Longueur = 600mm ; Largeur = 160mm ; Hauteur = 12mm

**Motorisation :** Gear motor, 24V DC

**Communication :** Profibus



Figure 10 Banc d'assemblage

## 2.2.4 Présentation Robot Héron

Le robot Héron est un robot mobile, motorisé et composé de multiples capteurs qui assurera des missions de navigation d'un point à un autre. Ce robot aura donc pour mission de se déplacer de manière autonome au sein du démonstrateur. Ses principaux déplacements seront de se présenter devant l'étagère de stockage où Héron viendra récupérer des pièces de l'entrepôt, il se présentera également devant le banc d'assemblage pour alimenter celui-ci en pièces ou encore en fin de ligne pour récupérer les produits finis. Evidemment, le robot Héron a été développé de manière à pouvoir modifier/ajouter/supprimer toutes ces positions qu'il peut atteindre, lui permettant d'atteindre n'importe quelle zone du démonstrateur.

Celui-ci est doté de multiples capteurs lui permettant d'analyser en temps réel son environnement et de pouvoir réagir face aux obstacles imprévus le rendant ainsi plus sécurisé.

**Poids du robot :** 25kg environ

**Dimensions :** Longueur = 60cm ; Largeur 50 cm ; Hauteur = 70 cm à 107cm

**Matériaux :** Aluminium, Forex, PLA, PETG(impression 3D)

**Systèmes électroniques :** Jetson TX2

**Système d'exploitation :** Ubuntu 16.06 LTS, Robot Operating System (ROS Kinetic)

**Autonomie :** 18h

**Communication :** Wifi, Bluetooth

**Langage de programmation :** Python, C++

**Motorisation :** 4 moteurs DC avec balais

**Vitesse du robot :** 44 cm/sec

**Charge maximale en traction :** 60kg estimée

**Charge maximale sur plateau :** 10kg

**Service apporté :** Déplacement au sein du démonstrateur

**Système de contrôle :** Manuel via manette de XBOX, contrôle autonome

**Précision d'estimation de l'odométrie :** Erreur (environ) de 1mm pour une distance parcourue de 1m (soit 0,001% d'erreur)

Ce robot aura donc pour mission d'apporter de la mobilité au sein du démonstrateur. La suite du document aura pour but de vous expliquer en détail la construction et le fonctionnement de ce robot.



Figure 11 Robot Héron

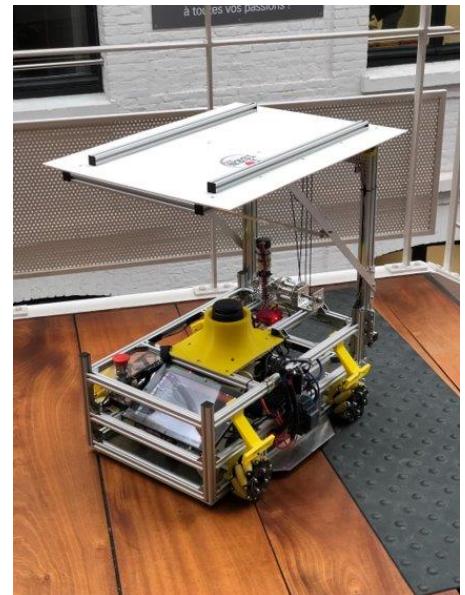


Figure 12 Robot Héron

### *2.2.5 Présentation PC central*

Le PC central a été configurer spécialement pour le projet via le site [www.topachat.com](http://www.topachat.com).



Figure 13 PC Central

Ce PC est, entre autres composé :

- D'un processeur AMD Ryzen 5 3600 (3.6 GHz)
- D'une carte graphique Gigabyte GeForce RTX 2060 OC, 6 Go
- De 16 Go de RAM

Ce PC à maintenant une adresse IP fixe sur le réseau YncreaLab (10.224.0.52).

L'OS installé est Ubuntu 16.04 LTS, le PC possède la distribution ROS Kinetic.

Un serveur MySQL est également présent sur ce PC (l'utilisation de cette base de données est détaillée ultérieurement)

Un serveur FTP est disponible sur ce PC, et pour les phases de développement, un serveur ssh est également disponible sur ce PC (via l'ID centralheron).

### 3 Conception et réalisation du robot Héron

#### 3.1 Introduction du robot Héron

##### 3.1.1 Fonctionnalités attendues de la part du robot

La liste des fonctionnalités ci-dessous nous a guidé tout au long de notre projet et c'est celle-ci qui nous a permis d'avancer étape par étape et de nous assurer que nous répondions toujours au cahier des charges que nous nous étions fixés en début de projet.

###### **Initialiser**

Le robot est capable d'être allumé par un interrupteur ON/OFF, allumant alors tous les dispositifs de celui-ci. La phase d'initialisation consiste à attendre que chaque dispositif de mette en route une fois l'interrupteur fermé (écran, drivers, jetson,...). Sa mise hors tension se fait en pressant un bouton poussoir afin d'éteindre la Jetson, puis en appuyant sur l'interrupteur pour couper l'alimentation.

###### **Recharger**

Le robot est capable d'être rechargeé simplement en branchant son chargeur (intégré au robot) à une alimentation secteur. Le câble de charge est standard. La recharge est rapide et se fait en quelques heures.

###### **Déplacer**

Le robot peut se déplacer de façon Holonomique pour accéder facilement aux différents points stratégiques du démonstrateur. C'est-à-dire que ses mouvements en translation sont à la fois possibles en  $x$  et en  $y$ . Il peut se déplacer, à la même vitesse aussi bien latéralement que d'avant en arrière. Le robot est aussi capable d'effectuer des rotations dans les deux sens.

###### **Elever**

Le robot dispose d'un plateau élévateur lui permettant d'accéder à différentes hauteurs de table. Ce plateau est commandé par un treuil mécanique commandé de manière automatique avec des distances préenregistrées.

###### **Localiser**

Le robot est capable d'analyser son environnement et identifier les obstacles. Il peut se repérer dans son environnement et l'enregistrer via l'utilisation et interprétation de ses différents capteurs. Il peut emprunter un chemin sécurisé pour atteindre une position précise.

###### **Communiquer**

Le robot est esclave dans son environnement de déploiement et se contente d'exécuter les ordres qui lui sont envoyés depuis le PC central. Il envoie des informations concernant l'état dans lequel il se trouve et reçoit des informations qu'il interprète ensuite.

###### **Afficher**

Le robot est capable d'afficher son écran d'accueil (bureau Linux) physiquement via un écran led, et aussi d'afficher les informations relatives à son état via interfaces graphiques (existantes ou créées pour l'occasion).

## Résister

Le robot est conçu de façon à résister aux contraintes de son environnement et du temps. Il est solide et résiste bien aux chocs qui pourraient lui arriver. Sa structure est en métal et lui permet d'avoir une bonne stabilité.

## Porter

Le robot est capable de transporter des pièces d'un point à un autre sur son plateau élévateur. Ces pièces peuvent être de simples produits à assembler tout comme un bras robotique. Des barres parallèles fixées sur son plateau sont destinées à pouvoir fixer des choses sur le robot si besoin ou amélioration future. Sa charge maximale est de plusieurs kilos.

## Déetecter fin de stock

Le robot est capable de détecter une rupture de stock dans l'entrepôt de stockage via traitement d'image et caméra. Cela permet d'alerter et d'anticiper des ruptures de stock afin de faciliter le travail de stockage.

### 3.1.2 Schéma d'interaction

Le robot est doté de divers moyens de communication, notamment en WIFI ou en Bluetooth. Aujourd'hui, le robot est capable d'échanger des données via Bluetooth lorsque l'on décide de le contrôler avec une manette dans le cas d'un contrôle manuel, mais également en WIFI avec le PC Central lorsque l'on décide de placer le robot en mode autonome.



Figure 14 Schéma des interactions possibles de Héron

### 3.1.3 Schéma Hardware

Le schéma hardware ci-dessous représente les différents blocs qui composent notre robot. Vous trouverez toutes les informations concernant l'alimentation de chaque bloc, la manière dont il transfère les données d'un bloc à l'autre ainsi que les liens entre chaque bloc. Ce schéma donne une première vision de la constitution et de l'architecture de notre robot

## SCHEMA HARDWARE HERON

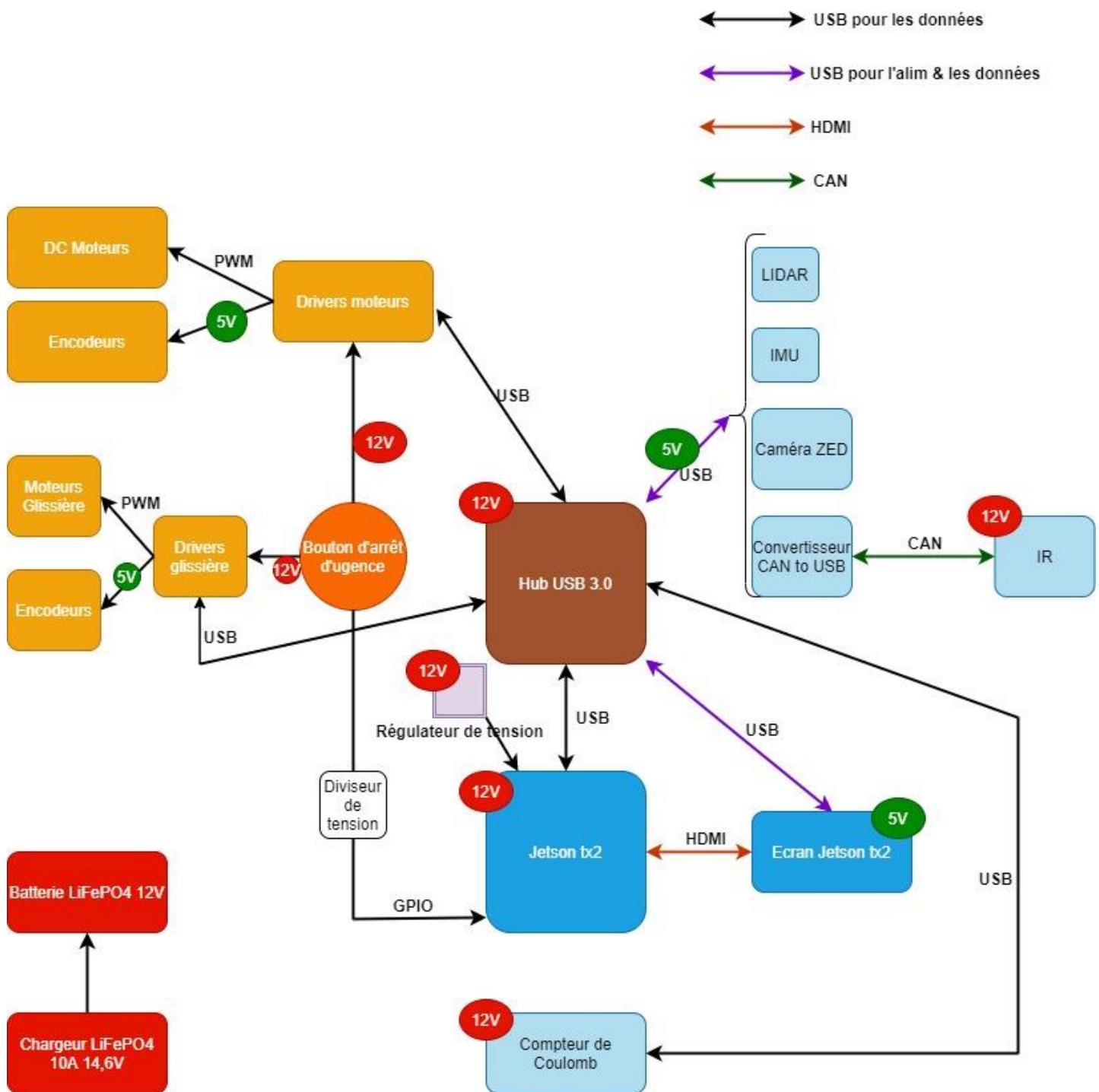
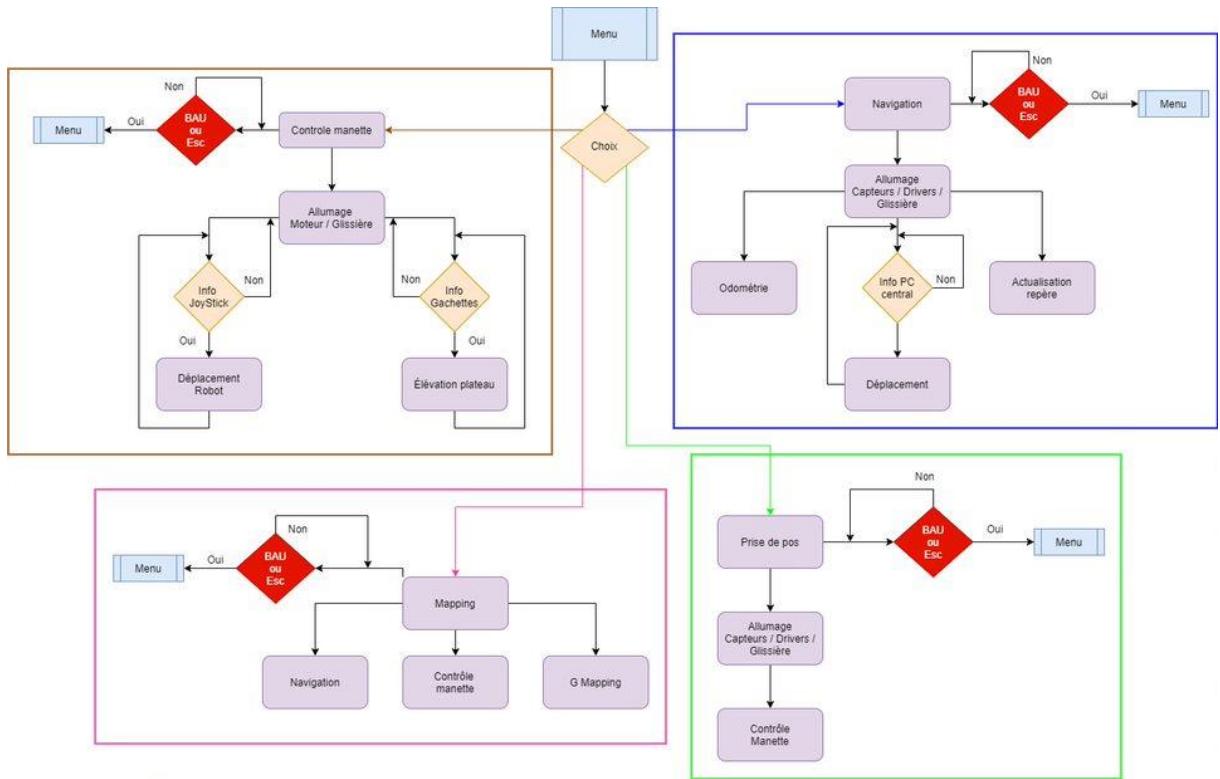


Figure 15 Schéma Hardware Héron

### *3.1.4 Organigramme Software*

Après avoir présenté la constitution hardware du robot, vous pouvez à présent trouver toutes les informations à propos de l'architecture software du robot. Ce schéma fournit une représentation graphique normalisée de l'enchainement des opérations et des décisions effectuées par le robot lors de son fonctionnement.



*Figure 17 Organigramme Software Robot Héron*

Ceux-ci seront expliqués plus en détail à la suite de ce document.

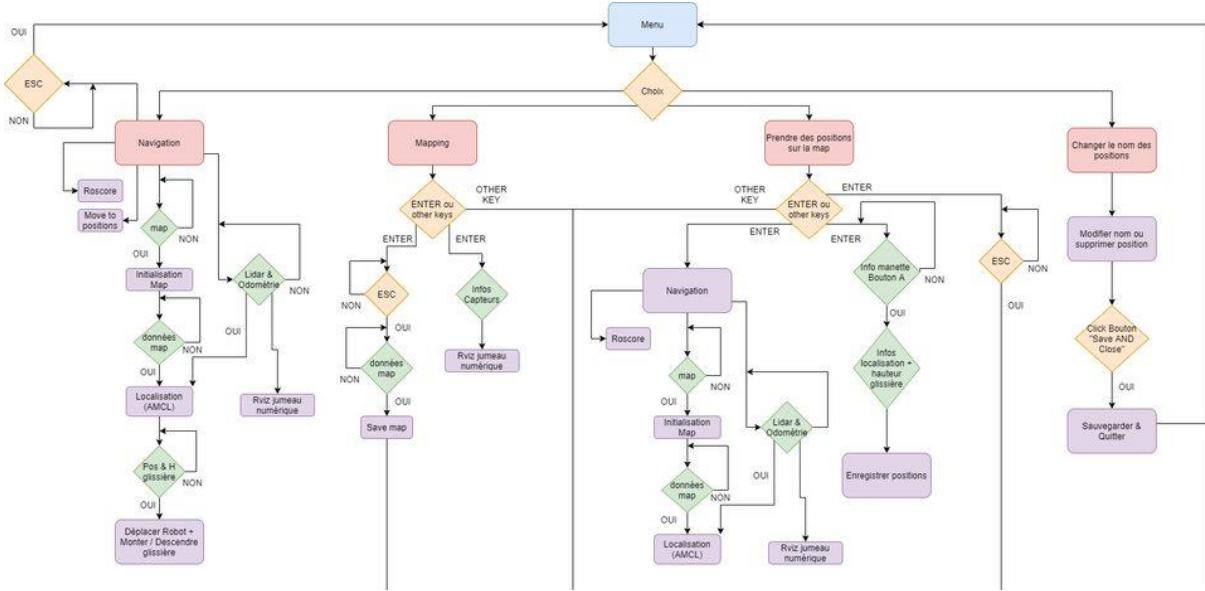
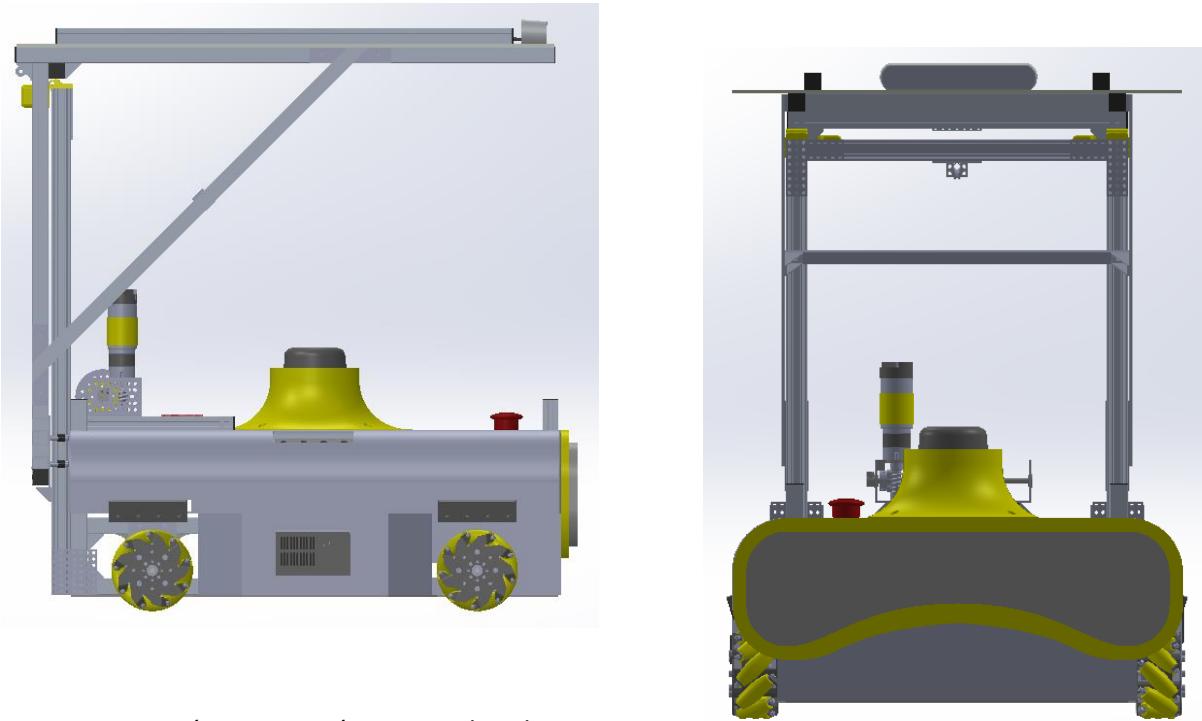


Figure 18 Organigramme Software PC Central

### 3.2 Conception mécanique

Dans cette partie nous allons aborder l'aspect mécanique du robot. Tout d'abord, voici quelques photos de la simulation du robot réalisé sur SolidWorks.



Caractéristiques mécaniques du robot

- Hauteur : 77cm à 107cm

- Largeur : 50cm
- Longueur : 60cm
- Poids : 25kg

Matériaux	Quantité
Profilé 20x20 aluminium : 560mm	6
340mm	12
240mm	4
500mm	1
412mm	1
662mm	2
376mm	1
336mm	3
560mm	2
55mm	4
Plaque 3x300x500 aluminium	2
1 bobine de fils PETG pour impression 3D	1
(Soit 1kg)	
Plaque de PLA 3x500x660	1

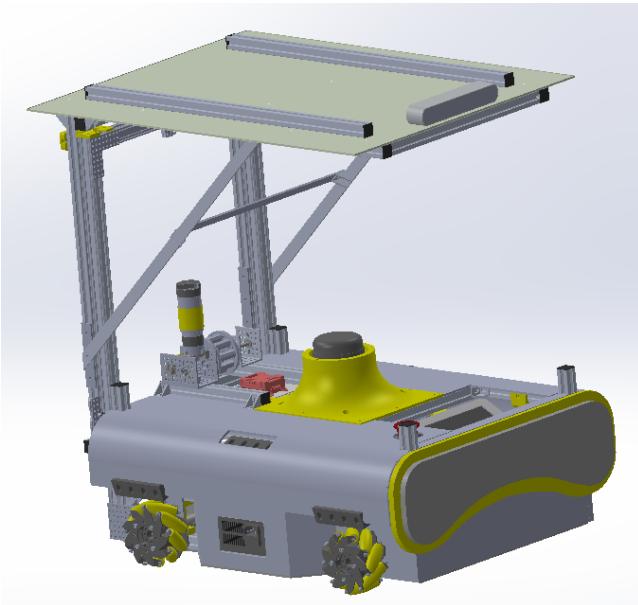


Figure 19 Représentation robot Héron Solidworks

- Charge maximale à porter : 10kg

Voici maintenant un listing des différents matériaux nécessaire à la fabrication du robot ainsi que leurs quantités.

Nous remarquons donc qu'il faut environ 15 mètres de profilé en aluminium 20x20 mm, une quantité importante à ne pas négliger. Pour découper les différentes tailles de profilé nous sommes allés au Fablab d'HEI pour utiliser une machine adaptée et précise. Ensuite pour ce qui est de l'impression 3D nous avons à notre disposition une imprimante Prusa i3 MK3 ainsi qu'une découpe laser au Fablab de l'ISEN.

Pour la réalisation des carters nous nous sommes rapprochés d'une entreprise qui nous a permis de concevoir nos pièces qui viennent recouvrir notre robot. Nous avions tout d'abord pensé à réaliser les pièces en thermoformage ou en injection plastique mais cela était hors de prix.

Nous avons donc réalisé un partenariat avec l'entreprise Pilotes PLV afin de réaliser nos carters. En effet, l'entreprise possède des machines spéciales afin de travailler le plastique. Nous avons donc pu bénéficier de ces installations.

L'aspect esthétique est très important, il apporte une plus-value au projet et embellie notre robot. Les designers de Pilotes PLV nous ont accompagnés afin de donner un côté moderne et industriel.

Pour visualiser notre simulation sur SolidWorks nous vous invitons à télécharger notre dossier complet qui est présent sur notre drive dans la partie SolidWorks.

### 3.2.1 Principales étapes de constructions de la base mobile

- Montage de l'étage 0 en profilé, lors de cette étape il faut bien faire attention de bien serrer les équerres qui vont tenir les profilés entre eux.
- Découpe des plaques en aluminium afin de réaliser le socle du robot. Il faut également penser à tarauder les trous pour les vis afin d'éviter que la tête de vis ne dépasse pas sous le robot.



Figure 20 Equipe Héron en action

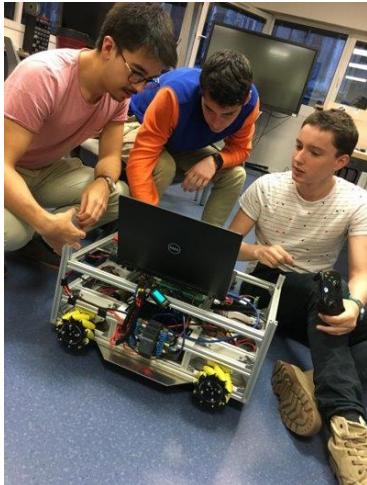


Figure 21 Equipe Héron en action (2)

- Ensuite, il faut intégrer les roues, les moteurs ainsi que la source d'alimentation. Il est primordial de bien fixer les moteurs afin qu'ils soient dans une position adéquate. (Vous retrouverez sur notre SolidWorks les différentes attaches et fixations afin de maintenir fermement le tout.)
- Viens ensuite l'étape d'intégration de tous les capteurs (IMU, IR, Lidar, Coulomb), les boutons d'allumages et aussi la carte-mère (Jetson).
- Une fois ces étapes réalisées il ne reste plus qu'à monter la glissière.

### 3.2.2 Principales étapes de construction de la glissière



Figure 22 Robot Héron sur Solidworks

La partie glissière concerne tout le module qui vient se fixer à l'arrière du robot en plus de la partie « treuil » située sur le dessus du robot à l'arrière. L'idée de conception était ici de faire quelque chose de modulable si besoin. Ainsi, si l'on souhaite retirer la partie glissière il suffira de détacher les fixations du treuil et de la partie arrière pour récupérer une base mobile fonctionnelle. La structure de la base mobile reste ainsi inchangée et est utilisable.

Le plateau élévateur est composé d'un treuil mécanique motorisé avec vis sans fin qui va permettre de tirer un câble dans un système de poulie. Le plateau peut se mouvoir de **695mm à 1070mm**

La glissière est donc composée de deux parties principales : l'une fixe solidement attachée à la base du robot, et l'autre amovible composée de la seconde partie du rail, et du plateau fixé à dessus avec ses renforts.

La partie amovible est en profilés 20mm pour, ses renforts sont des tiges d'aluminium coupées et percées selon besoin (cf. SolidWorks).

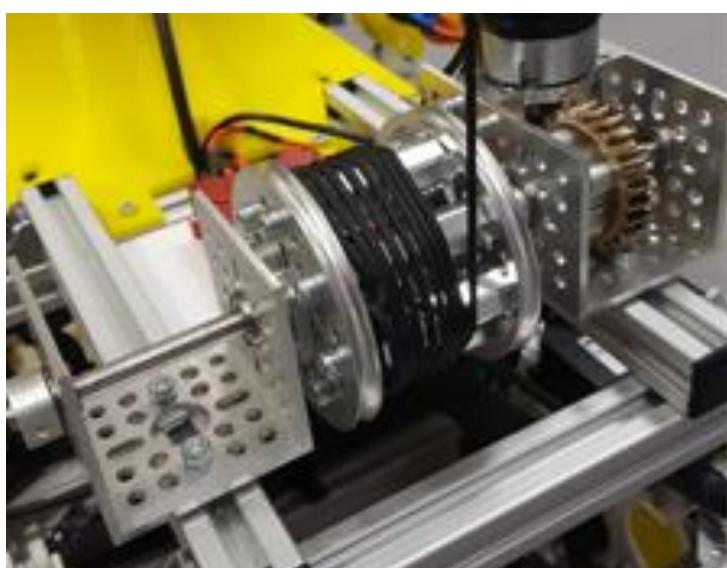
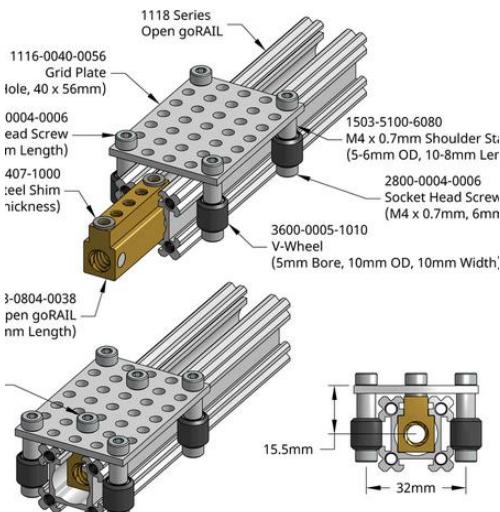


Figure 23 Système de glissière

La partie fixe est composée de rails et profilés différents issus du site goBILDA. Ceux-ci ont une métrique particulière (ex : briques Lego) dans tout leur magasin de pièces ce qui permet d'assembler des pièces avec précision et facilité tout en conservant la solidité.



Le système de levée se fait grâce à un système de poulie, ce qui permet de diviser par 2 la force nécessaire pour soulever le plateau.

La corde utilisée est une corde de 3mm type parachute pouvant supporter jusqu'à 90kg.

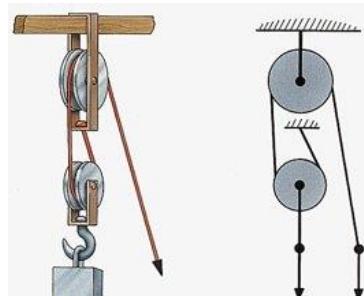
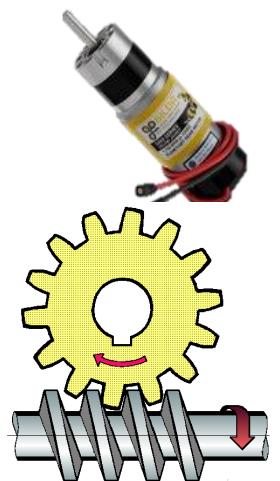


Figure 24 Explication système de poulie

Le principe utilisé de vis sans fin permet le blocage de la hauteur du plateau si le treuil ne tourne plus. Le pignon sans fin fait tourner l'engrenage qui enroule la corde dans un sens ou l'autre. Ainsi en s'enroulant, le câble va permettre d'élever ou d'abaisser le plateau.

Le mouvement se fait grâce à un moteur goBILDA 1150 RPM (specs ci-contre). Sa vitesse est diminuée par l'engrenage (ration 24 :1).



Le moteur fait 145.6 tics d'encodeur pour un tour. Donc un tour d'axe de treuil =  $24 * 145.6 = 3494.4$  tics/tr.

La vitesse du moteur en tics est de 2775 tics/sec maximum à 100% de sa puissance. On appellera QPPS la vitesse du moteur. Celle-ci sera bloquée à 1700 tics/sec

Une fois la glissière ajoutée sur la partie mobile du robot, voici une première représentation du robot Héron.

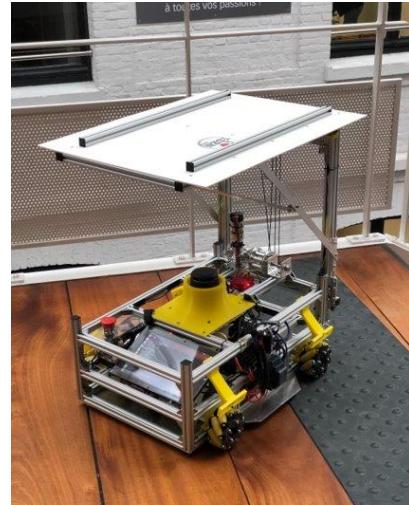


Figure 25 Robot Héron sans les carters

#### SPECS

Motor Size	RS-555
Motor Type	Brushed DC
Nominal Voltage	12VDC
Output Shaft	6mm D
Gearbox Style	Planetary
Gear Ratio	5.2:1
Gear Material	Casing: Steel Ring Gear Stage 1: Steel Sun Gear, Steel Orbital Gears
No-Load Speed @ 12VDC	1,150 RPM
No-Load Current @12VDC	0.25A
Stall Current @12VDC	9.2A
Stall Torque @12VDC	7.9 kg.cm (109 oz-in)
Wire Length	470mm (including connectors)
Wire Gauge	16AWG
Motor Connector Type	3.5mm FH-MC Bullet Connectors
Encoder Connector Type	4-Pin JST XH [FH-MC]
Encoder Type	Relative, Quadrature
Encoder Sensor Type	Magnetic (Hall Effect)
Encoder Voltage Range	3.3 - 5VDC
Encoder Cycles Per Revolution (Encoder Shaft)	7 (Rises of Ch A)
Encoder Cycles Per Revolution (Output Shaft)	36.4 (Rises of Ch A)
Encoder Countable Events Per Revolution (Encoder Shaft)	28 (Rises & Falls of Ch A & B)
Encoder Countable Events Per Revolution (Output Shaft)	145.6 (Rises & Falls of Ch A & B)
Weight	387g

Figure 26: caractéristiques du moteur de la glissière

### 3.2.3 Principales étapes de l'ajout des carters

Dans le but de finaliser la conception de notre robot Héron, nous avons décidé de revêtir celui-ci avec différents carters.



Figure 27 Robot Héron avec les carters

- ➔ Un carter frontal (tête)
- ➔ Un carter arrière
- ➔ Deux carters latéraux

Les carters apportent une vraie plus-value au robot Héron, de l'esthétisme, du professionnalisme et une réduction du bruit des moteurs.

La réalisation de ces carters a été faite en collaboration avec l'entreprise Pilotes PLV ([www.pilotesplv.fr](http://www.pilotesplv.fr)). L'entreprise nous a aider notamment en nous prêtant le matériel nécessaire à la conception de ces pièces.

**PILOTES**

Nous avons donc discuté avec des designers et des experts en prototypage lors de nombreuses réunions afin de se mettre d'accord sur le design final du robot et de pouvoir réaliser les carters.

Tout d'abord, il a fallu s'adapter à la structure du robot en elle-même, c'est-à-dire la forme, la disposition des différents capteurs autour du robot, le BAU, le chargeur et les boutons d'allumages.

Une fois que nous avons trouvé un accord avec les designers, nous avons modélisé sur SolidWorks la version finale des carters.

#### Réalisation :

- Choix du matériau, le Forex (3mm) : le forex est une matière plastique en PVC expansé qui est particulièrement adapté à l'impression directe. Il est facile à découper, à manipuler, à la fois léger et résistant, il est également ignifugé (pas inflammable) ce qui est un vrai plus pour notre projet.
- Découpe du patron (à plat) grâce à la Procut ZUND.
- Pliage fait en chauffant la pièce en question au bon endroit avec une résistance qui vient chauffer le plastique, on peut ainsi lui donner un angle.

- Collage des différentes pièces (si nécessaire, cela dépend des carters)
- Finitions au cutter, pour les petits détails.

Nous avons ramené le robot Héron directement dans les locaux de Pilotes PLV afin de faire les premiers tests et faire les dernières modifications du carter.



Figure 29 Découpe Procunt ZUND



Figure 28 Fabrication des carters



Une fois les carters parfaitement adaptés au robot Héron, nous avons peint les pièces en noir mat (à la bombe) et nous avons imprimé notre logo sur des petites plaques en aluminium (3mm) pour une finition encore plus propre.

Pour cela on a utilisé une imprimante numérique Arizona CANON.



#### *Système d'affichage :*

Afin que l'accès aux différents boutons soit assez simple, nous avons créé une plaque avec les boutons (et l'écran de la batterie) adapté directement dessus, qui vient sous le carter.



Figure 32 Vue avec carter



Figure 30 Vue sans carter



Figure 31 Plaque non fixée pour accéder à l'électronique

#### Montage :

- Système de fixation intelligent, il faut seulement 8 vis type T-nuts pour pouvoir attacher les 4 carters sur la structure du robot. (Soit 2 vis par carter)
- Première étape du montage : Positionner les vis dans chaque trou des différents carters
- Deuxième étape : Positionner les carters avant et arrière en premier. Il faut les glisser au niveau des axes des roues et doivent venir « bord à bord » avec la plaque du bas du robot.
- Troisième étape : Glisser les 2 carters latéraux en faisant attention aux capteurs. Il faut commencer par glisser la partie supérieure du carter (le haut) afin de bien emboiter le bas du carter.
- Quatrième étape : Visser toutes les vis sans forcer et ne pas oublier une rondelle pour ne pas forcer le plastique (risque de casser).

#### Résultat final :



Figure 33 Robot Héron avec les carters

### 3.2.4 Composition du robot du point de vue global

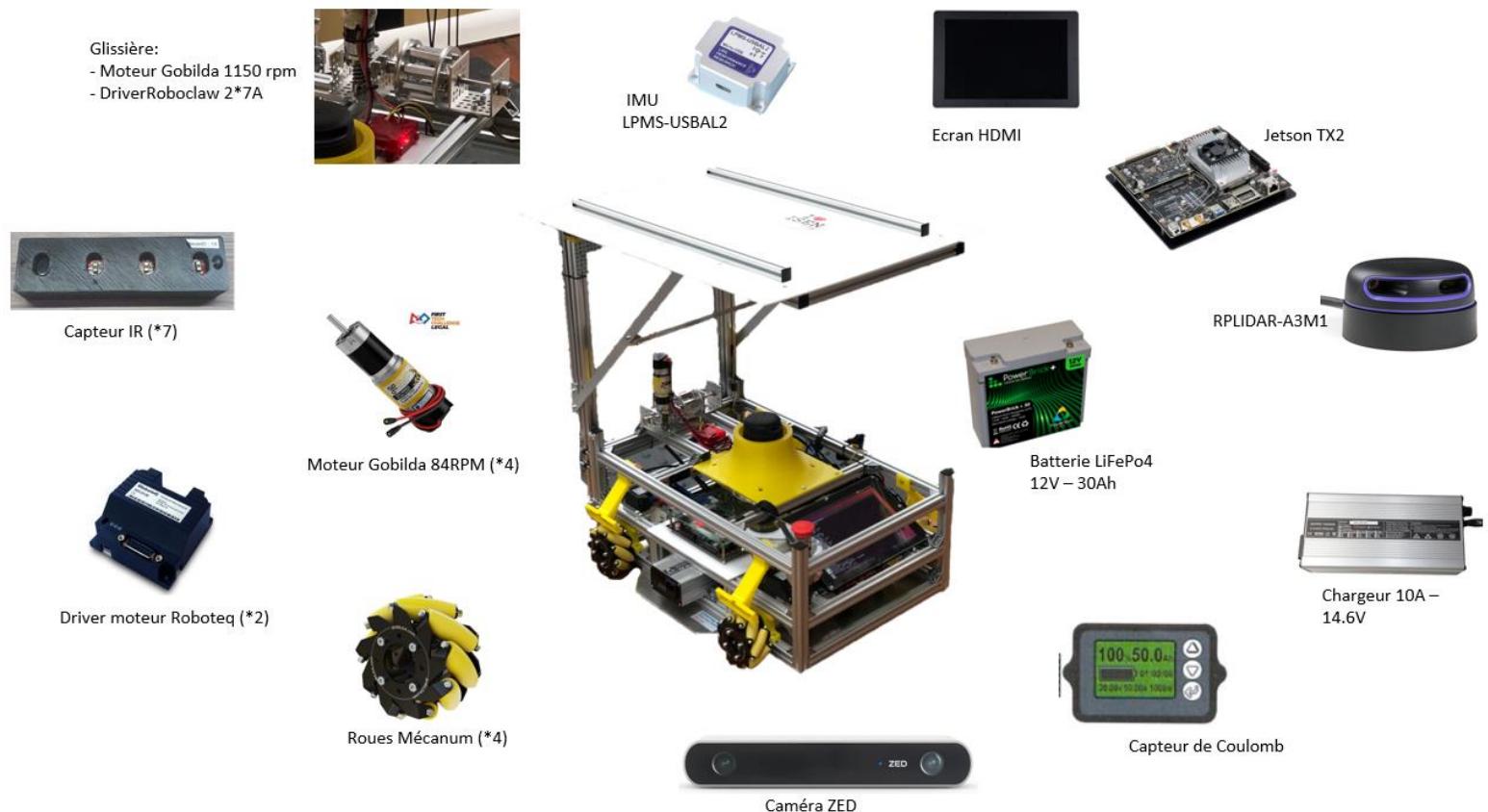


Figure 34 Représentation du robot et de ses composants

Le robot est évidemment composé de multiples capteurs et actionneurs. Ci-dessous une liste de tous les composants du robot.

- \_ 1 batterie
- \_ 1 chargeur
- \_ 4 roues Mécanum
- \_ 4 moteurs
- \_ 2 drivers moteurs
- \_ 1 carte Jetson TX2
- \_ 1 Lidar
- \_ 7 capteurs IR
- \_ 1 caméra ZED
- \_ 1 centrale inertielles (IMU)
- \_ 1 compteur de coulomb
- \_ 1 glissière :
  - \_ 1 moteur
  - \_ 1 driver moteur
- \_ 1 écran HDMI

### 3.3 Alimentation du robot

#### 3.3.1 Batterie et chargeur

Afin d'alimenter le robot et de garantir une autonomie de minimum 10h, nous avons réalisé plusieurs calculs d'estimation afin de choisir une batterie qui répondrait à nos besoins.

La Batterie choisie utilise la technologie LiFePo4, lithium fer phosphate. La tension nominale de 12V permet l'utilisation de tous les périphériques sans avoir à augmenter celle-ci. Les 30Ah permettent une grande autonomie au robot de plusieurs heures. Ce type de batterie tient très longtemps en termes de cycle de recharge, elles se rechargent en quelques heures (3h15 ici).

##### Pack Lithium Ferro-Phosphate (LiFePO4) - 12V - 30Ah

- Durée de vie très élevée : de 3000 cycles à plusieurs milliers (voir abaque)
- Décharges profondes possibles (jusqu'à 100 %)
- Technologie **Lithium Fer Phosphate** complètement sécurisée (pas de risque d'explosion, ni d'auto-inflammation)
- **BMS** (Battery Management System) intégré au boîtier : maximise la durée de vie ET sécurise la batterie
- Très faible toxicité pour l'environnement
- Durée de vie calendaire > 10 ans
- Excellente tenue en température (-20 °C à +60 °C)
- Système flexible : jusqu'à 10 packs en parallèle et 4 en série
- Puissance constante durant toute la décharge (faible résistance interne)
- Très faibles pertes de Peukert (rendement énergétique >98 %)
- Très faible auto-décharge (<3 % par mois)
- Pas de plomb, pas de terres rares, pas d'acide, pas de dégazage
- Gain de poids > 50 % et encombrement diminué de 40 % par rapport aux batteries au Plomb
- Certification : CE, RoHS, UN 38.3, UL

##### Spécifications techniques

	Électriques	Charge standard	Décharge standard	Environnement	Mécanique
Tension nominale	12.8V	14.4V ± 0.2V	45A (576 W)	0°C à +50°C	Cellules et assemblage
Capacité nominale	30 Ah	CC/CV : Constant Current / Constant Voltage	70 A (896 W)	-20°C à +60°C	Boîtier
Energie	384 Wh	15 A (Maxi continu : 30A)	10V	0°C à +50°C @60±25% d'humidité relative	Dimensions
Résistance interne	≤ 50mΩ	Tension de coupure BMS	IP 66	Protection IP	Poids
Nombre de cycles	>3000 cycles (voir abaque)	14.8V ± 0.1V	26650 - 4S6P		Terminal
Autodécharge	< 3% par mois				
Rendement énergétique	98% - 99% @1C				

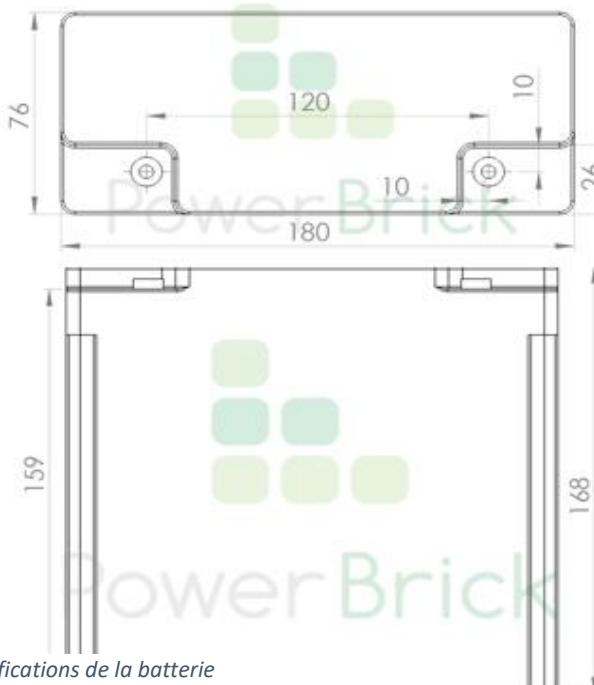
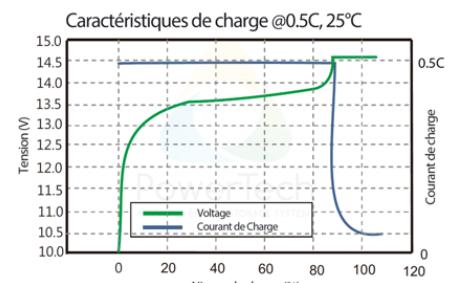
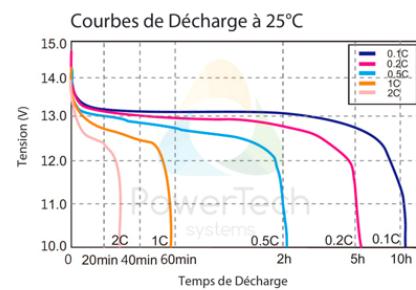


Figure 35 Spécifications de la batterie

<https://www.mylithiumbattery.com/fr/shop/batteries-lithium-12v/batteries-lithium-12v-lifepo4/batterie-lithium-12v-30ah-life-lifepo4-powerbrick/>



Le chargeur est directement intégré au robot et déjà branché. Il suffit juste de brancher le câble d'alimentation standard sur le côté du robot au secteur. Le chargeur fournit 10A ce qui permet une charge plus rapide. Pour charger le robot il faut allumer le bouton rouge, tous les autres éteints, et brancher le câble d'alimentation.

240W-10A Charger for 12V Lithium Iron Phosphate batteries	
Rated input voltage	230 VAC
Input voltage range	100-240VAC
AC input voltage frequency	47-63Hz
Nominal charge voltage	12.8V (4 cells LiFePO4)
Charge voltage	14.6V +- 0.1 V (4 cells LiFePO4)
Constant current	10 +- 0.5 A
Power efficiency	>85%
Maximum output power	240W
End of charge condition	0.12A < Charge current < 0.30A
Output voltage range	9-14.6 V
Output over voltage protection	16V
Over voltage protection	The charger software limits the maximum output voltage to a level suitable for the connected battery system
Thermal cutback	The internal temperature monitor reduces the charger output power in extreme operational temperature to prevent damage
Output short circuit protection	12A
Short circuit protection at the output terminals. Automatic recovery after restoring to normal conditions	
Reverse battery protection	The charger is protected against reversed battery connection.
Dimensions	180*100*55mm
Weight	0.820 kg
Operating temperatures (max/min)	+40°C / -10°C
Storage temperatures (max/min)	60°C / -20°C (allow 2 hours to recover to normal temperature)

Figure 36 Spécifications du chargeur

<https://www.mylithiumbattery.com/fr/shop/chargeur-de-batterie-lithium-ion/chargeurs-pour-batteries-lithium-12v/chargeur-240w-10a-pour-batterie-12v-lithium-fer-phosphate/>

### 3.3.2 Distribution principale et allumage du robot

L'alimentation principale du robot distribue directement la tension de la batterie soit 12,8V nominal.

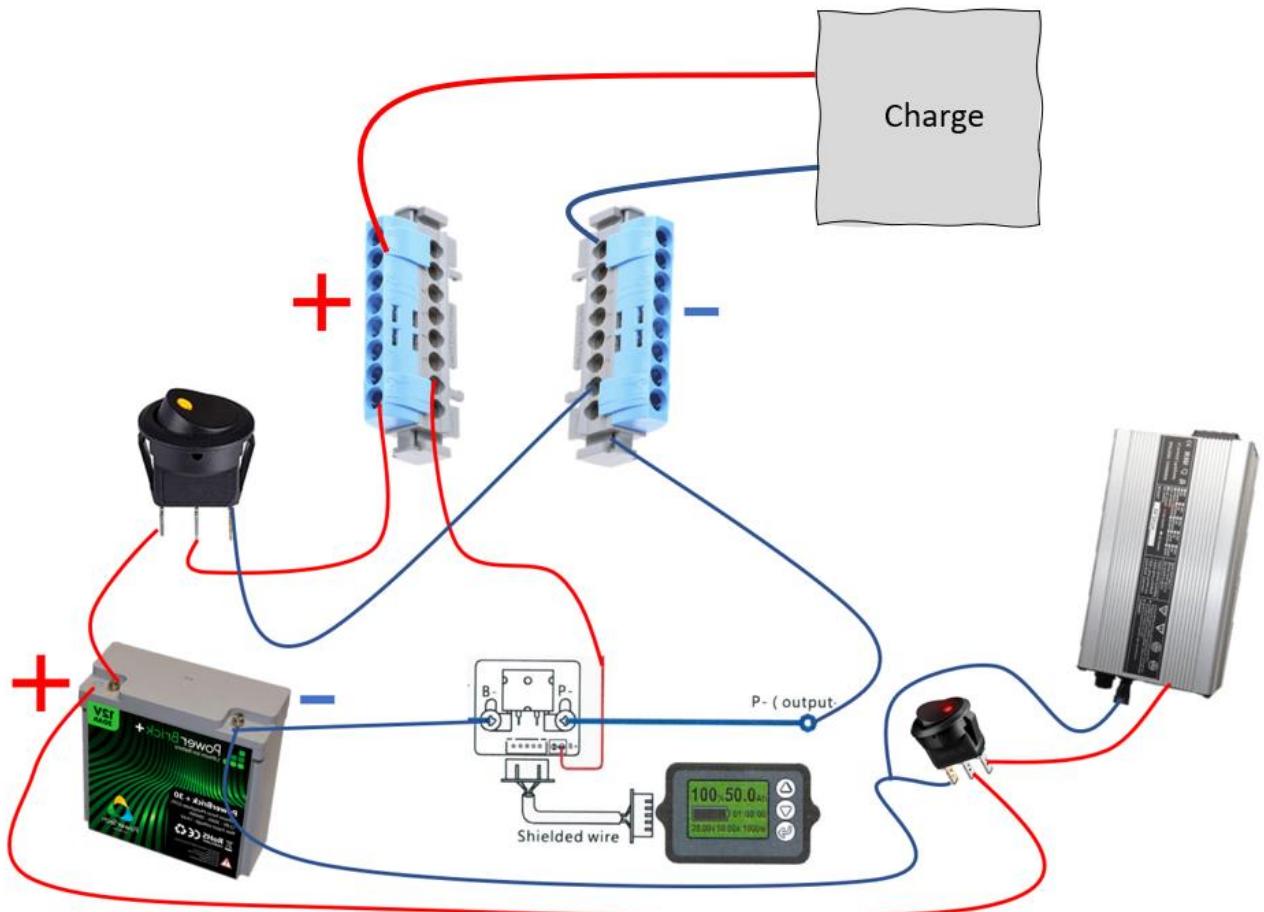


Figure 37 Distribution principale du robot Héron

Un interrupteur « ON/OFF » (jaune) permet d'allumer le robot et de fermer le circuit. C'est l'interrupteur principal. Un autre « CHARGE » (rouge) permet de fermer le circuit côté chargeur lorsque l'on souhaite recharger le robot. Celui-ci doit être fermé (led rouge allumée) que lorsque qu'on utilise le chargeur, l'alimentation générale (jaune) doit être coupé pour la recharge (la laisser allumée n'est pas dangereux mais déconseillé lors de la charge). Pour être clair, une partie de l'énergie du chargeur sera utilisée pour alimenter les périphériques du robot si celui-ci est allumé, au lieu de servir à la recharge de la batterie.

Si tous les interrupteurs sont coupés le robot ne consomme rien. La **charge** attachée est détaillée dans la prochaine partie.

Pour charger le robot, lorsqu'on va allumer le bouton rouge et brancher l'alimentation au secteur, un bruit de ventilation va commencer et durer quelques secondes. Une fois la charge lancée il faut attendre au maximum 3h30 (pleine charge) pour que la batterie soit chargée de 0 à 100% et donc moins de temps si la batterie n'était pas complètement déchargée.

### 3.3.3 Distribution secondaire

La distribution secondaire se fait depuis le bornier, les périphériques demandant 12V sont alimentés, avec les protections nécessaires, un Hub USB est branché pour alimenter en 5V les autres périphériques et communiquer avec le PC. Un convertisseur de tensions est mis avant l'alimentation de la Jetson pour lisser la tension à 12V, celle-ci ne supporte pas la moindre variation de tension.

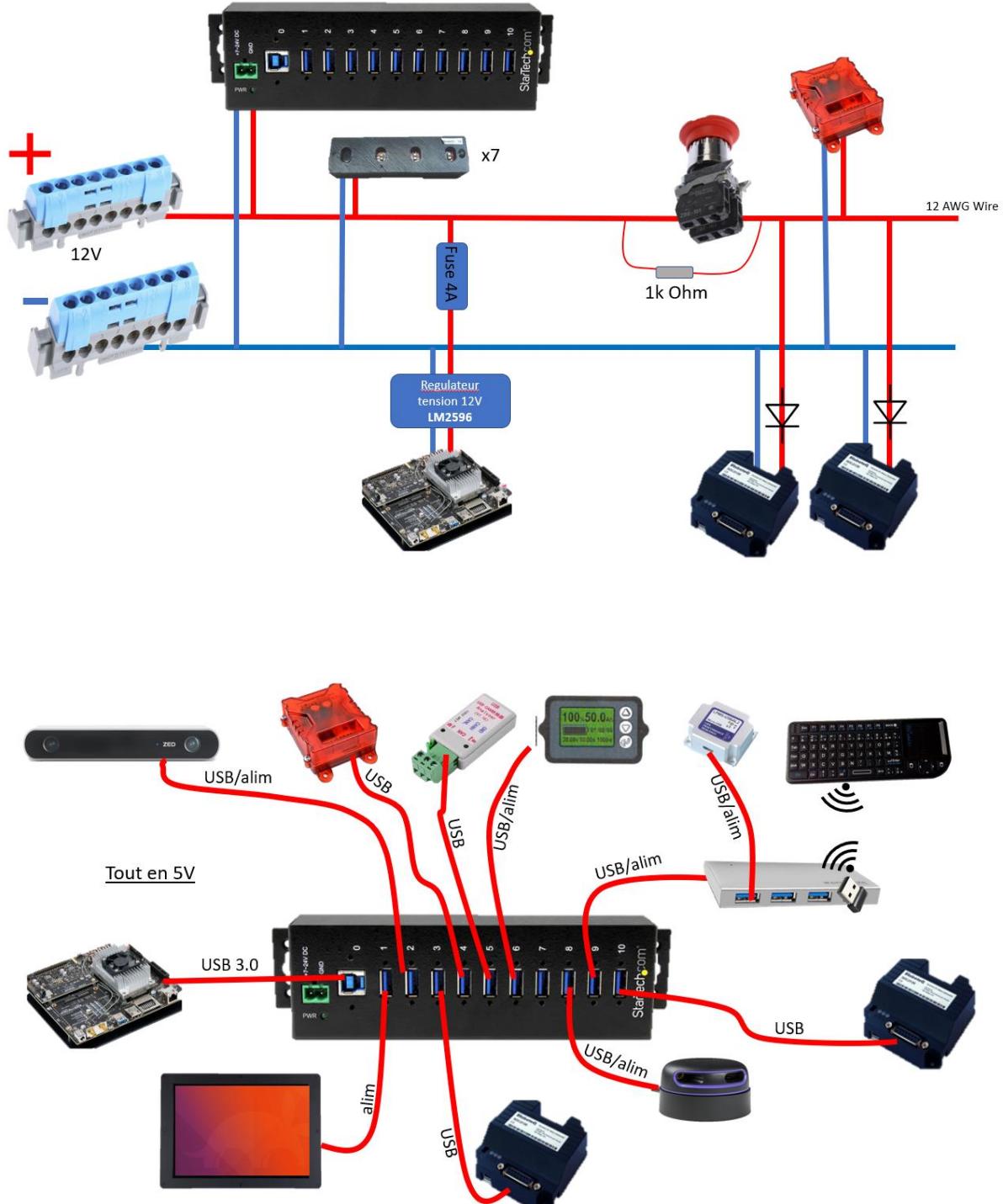


Figure 38 Distribution secondaire du robot

### 3.3.4 Bilan de consommation

Composants	Energie
Jetson TX2	$0,5\text{Ah} \times 12\text{V} = 6\text{Wh}$
Moteur (X4)	$0,52\text{Ah} \times 12\text{V} \times 4 = 25\text{Wh}$ Actif 30% du temps (18mins) => $25 \times 0,3 = 7,5\text{Wh}$
Driver moteur (X2)	$0,3\text{Ah} \times 12\text{V} \times 2 = 7\text{Wh}$ Actif 30% du temps (18mins) => $7 \times 0,3 = 2,1\text{Wh}$ A vide => $0,5\text{Wh}$
Lidar	$0,3\text{Ah} \times 5\text{V} = 1,5\text{Wh}$
Capteur IR (X5)	$0,1\text{Ah} \times 12\text{V} \times 5 = 6\text{Wh}$
IMU	$0,0545\text{Ah} \times 5\text{V} = 2,725\text{Wh}$
Camera ZED	$0,38\text{Ah} \times 5\text{V} = 1,9\text{Wh}$
Moteur glissière	$3\text{Ah} \times 12\text{V} = 36\text{Wh}$ Actif 5% du temps (3mins) => $0,05 \times 36 = 1,8\text{Wh}$
Hub USB	$(1,9\text{W} + 0,18\text{W} + 2\text{W}) \times 0,2 = 0,816\text{Wh}$
Radar	$2,5\text{Ah} \times 5\text{V} = 12,5\text{Wh}$
Capteur US (X2)	$0,1\text{Ah} \times 12\text{V} \times 2 = 2,4\text{Wh}$
Ecran Jetson	$0,48\text{Ah} \times 5\text{V} = 2,4\text{Wh}$

Energie totale = **70,041Wh** en fonctionnement

- Temps de charge 0 [Symbole] 100% = **3h10 ( chargeur 10A)**
- Energie totale batterie :  $12,8\text{V} \times 30\text{Ah} = 384\text{Wh}$
- Energie totale des composants en continu : **70,041Wh** = environ **5h30** d'autonomie théorique
- Energie actif 30% du temps : **47,641Wh** = environ **8h** d'autonomie théorique
- Utilisation en démonstration au forum : **15-20h** d'autonomie réel
- A l'arrêt : **11,528Wh** = environ **33h** d'autonomie théorique

### 3.4 Contrôle bas niveau du robot

#### 3.4.1 Présentation des composants et de l'électronique assurant l'asservissement de la base mobile du robot

##### 3.4.1.1 Roues

Les roues que nous utilisons sont des roues **Mécanum** qui permettent au robot de naviguer dans toutes les directions grâce à leurs rouleaux placés autour de la roue à un angle de 45°. Elles ont un rayon de 10cm mais c'est une valeur à utiliser avec précaution, en effet pour l'odométrie du robot un papier stipule que les calculs basés sur ce rayon ne sont pas corrects et introduit donc une solution prenant en compte un rayon différent pour les translations. Lien du papier :

[https://www.researchgate.net/publication/326283867\\_Influence\\_of\\_mecanum\\_wheels\\_construction\\_on\\_accuracy\\_of\\_the\\_omnidirectional\\_platform\\_navigation\\_on\\_example\\_of\\_KUKA\\_youbot\\_robot](https://www.researchgate.net/publication/326283867_Influence_of_mecanum_wheels_construction_on_accuracy_of_the_omnidirectional_platform_navigation_on_example_of_KUKA_youbot_robot)

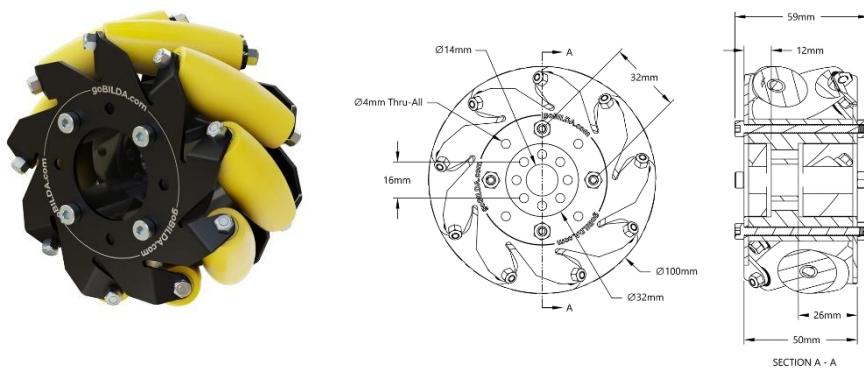


Figure 39 Roue Mécanum

Les roues utilisées proviennent du site GoBILDA : <https://www.gobilda.com/3606-series-mecanum-wheel-left-slant-bearing-supported-rollers-100mm-diameter/>

##### 3.4.1.2 Moteurs

Les moteurs que nous utilisons pour les 4 roues du robot sont des moteurs à courant continu avec réducteur à engrenages planétaires et encodeurs intégrés. Ils sont alimentés en 12V et leur courant maximum est 9.2A.

$$(84/60)*2*\pi*r = \text{vitesse robot mm.s}^{-1} = 44 \text{ cm.s}^{-1}$$

Spécifications moteurs	
Vitesse	84 RPM
Ratio réducteur	71.2 :1
Couple	93.6 kg.cm

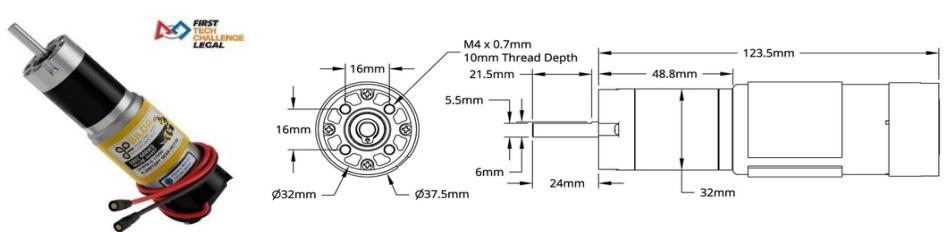


Figure 40 Moteurs DC 84RPM

En ligne droite le robot aura une vitesse **maximale** de 0,44 m.s<sup>-1</sup>.

Les moteurs utilisés proviennent du site goBILDA : <https://www.gobilda.com/5202-series-yellow-jacket-planetary-gear-motor-71-2-1-ratio-84-rpm-3-3-5v-encoder/>

### 3.4.1.3 Encodeurs



Figure 42 Encodeurs magnétiques à effet Hall

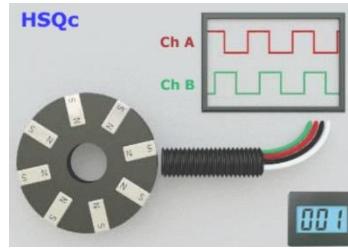


Figure 41 Encodeur effet Hall Quadrature Principe

Les moteurs sont équipés d'encodeurs magnétiques à effet Hall en quadrature. Ce qui veut dire que l'on a deux canaux de données A et B qui nous permettent de compter les événements ainsi que de connaître le sens de rotation du moteur. Ils sont alimentés de 3.3V à 5V.

On les connecte aux drivers en passant par un adaptateur Dsub 15 vers connecteurs à visser.



Spécifications encodeurs	
Cycles par révolution (sortie moteur)	7 (montée A)
Cycles par révolution (sortie bloc)	498,4 (montée A)
Nombre d'événements par révolution (sortie moteur)	28 (montée et descente A et B)
Nombre d'événements par révolution (sortie bloc)	1993,6 (montée et descente 1 et B)

On a donc une résolution de 1993,6 pour un tour de roue. Nos roues ont un diamètre de 100mm donc on obtient une résolution au sol de 0,16 mm.

$$2 * \pi * r / 1993,6 = \text{résolution au sol}$$

### 3.4.1.4 Drivers Roboteq



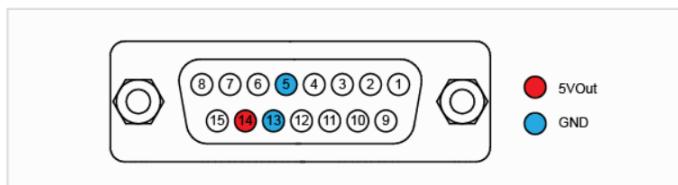
Figure 43 Driver de moteurs  
RoboteQ SDC2160

Les Drivers que nous utilisons pour les 4 moteurs des roues du robot sont des RoboteQ SDC2160. Ils permettent de contrôler deux moteurs chacun, nous en avons donc deux : avant et arrière. La transmission d'information avec la carte mère se fait via USB mais les drivers supportent aussi les communications suivantes : CAN, RS232, Analog, RC Pulse. Ils intègrent la possibilité d'utiliser des régulateurs PID en position, en vitesse et un mode mixte. Nous utilisons une régulation en vitesse.

Les drivers se configurent depuis cette interface. Elle permet de renseigner les spécifications des encodeurs, les moyens de communications, les entrées sorties ainsi que d'uploader des scripts microbasics qui peuvent être exécutés sur les drivers.

On branche les encodeurs sur les drivers par l'intermédiaire du connecteur cité précédemment. Les branchements sont les suivants. Ces informations sont aussi disponibles dans l'interface Roborun+.

Pin	1	2	3	4	6	7	8
Digital In				DIN1			DIN2
Pulse In				RC1			RC2
Ana In <v2.0							ANA2
Ana In >v2.0				ANA1			ANA2
Digital Out	DOUT1						
Encoder				ENC1A			ENC1B
Com		TxDATA	RxDATA		CANL	CANH	



Pin	9	10	11	12	15		
Digital In		DIN5	DIN4	DIN3	DIN6		
Pulse In		RC5	RC4	RC3	RC6		
Ana In <v2.0		ANA1	ANA4	ANA3			
Ana In >v2.0		ANA5	ANA4	ANA3	ANA6		
Digital Out	DOUT2						
Encoder		ENC2A			ENC2B		

Figure 44 Pinout RoboteQ branchement encodeurs

Les drivers génèrent des PWM pour la commande de vitesse des moteurs. C'est la structure de pont en H à l'intérieur des drivers qui permet de les générer. Les moteurs à courant continu se commandent de cette manière car ils interprètent ce signal comme une tension moyenne, ce qui permet donc de faire varier leur vitesse.

La consigne de vitesse des moteurs est entière et comprise dans [-1000 ;1000], donc 1000 niveaux de vitesses différents plus le sens.

La carte mère bénéficie d'une API en c++ fournie par le constructeur pour dialoguer avec les drivers. Nous avons modifié celle-ci pour l'utiliser avec ROS.

### 3.4.2 Développement asservissement

Le module asservissement se compose de **4 moteurs DC** pilotés par **2 drivers** de moteur Roboteq SDC2160. Les drivers assurent la régulation en vitesse des moteurs grâce au feedback des **encodeurs**. La commande en vitesse des moteurs se fait à partir de la **carte mère** qui communique via Serial par l'intermédiaire d'un Hub USB qui accueille les drivers.

On commande les moteurs par des PWM ce qui nous permet de contrôler le sens et la vitesse de rotation. Ces PWM sont produites par les Drivers de moteurs. La vitesse des moteurs est régulée en utilisant le feedback des encodeurs présents sur les moteurs. Cette régulation se fait par le driver avec un PID dont les ont été déterminés en fonction de notre cahier des charges. Les réglages du régulateur se font sur Roborun+ qui est l'interface de configuration des drivers. Le driver qui fournit la puissance aux moteurs est directement alimenté par la batterie.

#### 3.4.2.1 Régulateur bas niveau du PID

##### 3.4.2.1.1 Modélisation du moteur

En utilisant nos moteurs pilotés par nos drivers.



Figure 45 Graphique acquisition

On effectue des relevés de la vitesse de rotation des moteurs en fonction de la consigne appliquée, puis on extrait la courbe de tendance en faisant l'hypothèse d'avoir un système du premier ordre. Ensuite on étudie la réponse à un échelon du système et on en déduit sa fonction de transfert.

##### 3.4.2.1.2 Choix du régulateur

Sur Matlab Simulink on utilise PID Tuner pour obtenir nos gains en fonction du temps de réponse et de l'erreur souhaitée. Pour cela on modélise l'ensemble de notre système régulé.

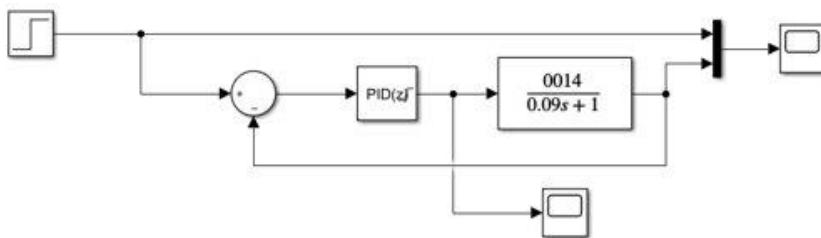


Figure 46 Schéma bloc Matlab Simulink Système et son régulateur PID

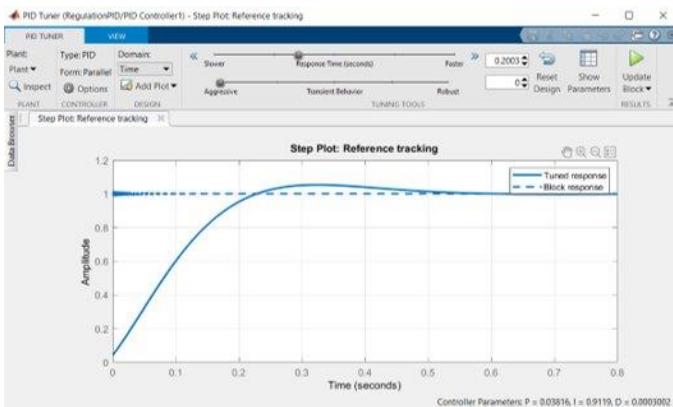


Figure 47 PID Tuner Matlab Simulink

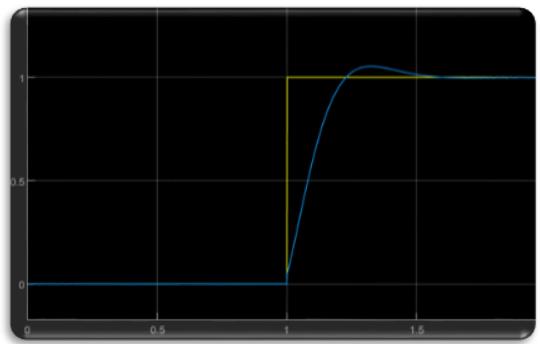


Figure 48 Réponse de notre système régulé

A noter que nos drivers ont un taux de rafraîchissement de 1000Hz pour le calcul des PIDs, ce paramètre est à prendre en compte car il influence les résultats. Il ne faut pas oublier non plus de prendre en compte la saturation de la commande ou encore la précision des encodeurs.

On choisit d'avoir un temps de réponse de notre système de 0.2 secondes et on obtient nos gains (arrondis car le driver accepte des entiers) : P = 0, I = 1, D = 0.

(Le fichier de configuration des drivers est disponible sur le drive.)

### 3.4.3 Asservissement de la glissière

#### 3.4.3.1 Présentation des composants et de l'électronique assurant l'asservissement de la base mobile du robot

Le driver servant à piloter la glissière est un Roboclaw 2\*7A. Il possède une interface Windows pour le paramétrage, les paramètres sont chargés dans sa carte. Toutes les informations et logiciel sont sur le site de Basicmicro. [https://www.basicmicro.com/Roboclaw-2x7A-Motor-Controller\\_p\\_55.html](https://www.basicmicro.com/Roboclaw-2x7A-Motor-Controller_p_55.html)

L'avantage de ce driver est qu'il embarque et qu'il gère des pins permettant d'ajouter des capteurs de fin de course. Un est placé en bas, c'est le Home switch, l'autre en haut pour limiter le mouvement. L'alimentation provient directement de la batterie, celle-ci peut être coupée si un problème survient simplement avec le bouton vert situé sur un des côtés du robot.



Le Driver Roboclaw est bien fourni en documentation et support (forum) et dispose de nombreuses fonctionnalités. Une des fonctionnalités utilisée dans notre cas est celle des capteurs de fin de course intégrés. Des pins sont à disposition sur le driver, certains pour brancher les encodeurs moteurs (voir doc Roboclaw) d'autres pour les capteurs ou autres usages. Dans notre cas ce sont les pins S4 et S5, capteurs bas et haut respectivement, qui sont branchés.

Le driver va s'occuper lui-même de gérer l'arrêt du moteur si jamais celui-ci veut aller plus loin que la limite réelle. Concrètement, si on est déjà au minimum, aller encore plus en bas sera bloqué. Si on est déjà au maximum, aller encore plus en haut sera bloqué. Ce blocage est indiqué par une led rouge s'allumant sur le driver (vert si normal), c'est une sécurité physique. Une

sécurité software limite aussi la glissière en position (voir paramètres plus bas en positon). Si le moteur veut aller plus haut que 14330 tics celui-ci sera bloqué et ne pourra aller plus loin. S4 est en mode **Home(user)** qui signifie que le driver sait que le capteur bas correspond à la position « de base » de la glissière. « User » veut dire que c'est à l'utilisateur de programmer le retour de la glissière au Home.

Il existe un mode « auto » qui fait descendre la glissière dès l'allumage du driver mais engendre quelques bugs, elle est donc désactivée. S5 est en mode **limit(fwd)**, qui a simplement pour rôle de limiter la glissière comme expliqué avant.

Le driver peut gérer 2 moteurs indépendamment, ici nous avons « fusionné » les deux canaux pour augmenter la puissance disponible pour 1 moteur. Ceci est fait en dédoublant physiquement les câbles d'alim, mais aussi en selectionnant le paramètre « single channel mode » dans l'interface.

Le but du « homing », retour à la base automatique, est que lorsqu'on lance le programme, il est impossible de savoir à quel niveau la glissière était quand on a éteint le robot. Il faut donc calibrer le moteur grâce aux capteurs à l'instar des imprimantes 3D.



Figure 49 Partie haute ou basse atteinte led rouge

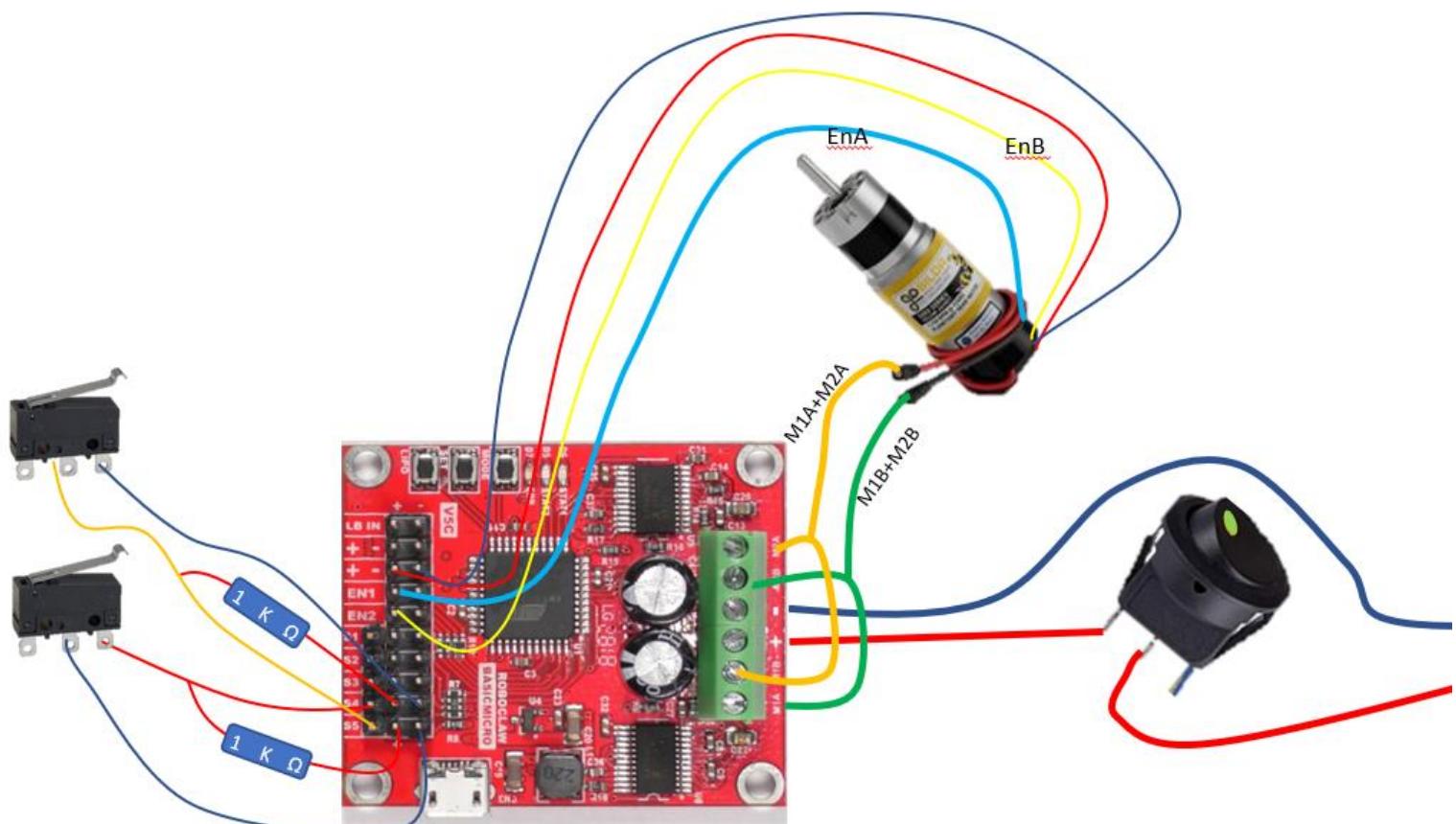
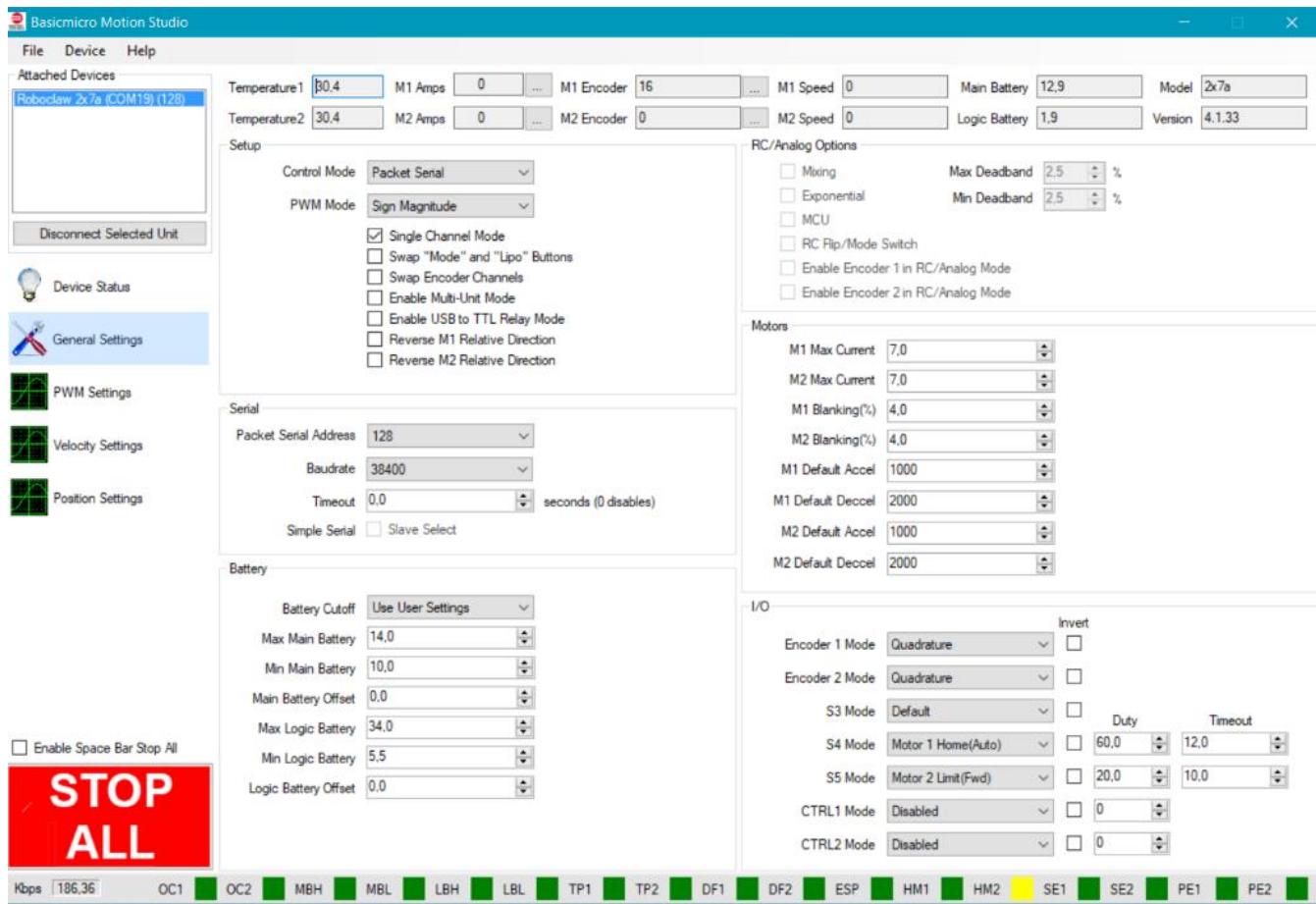
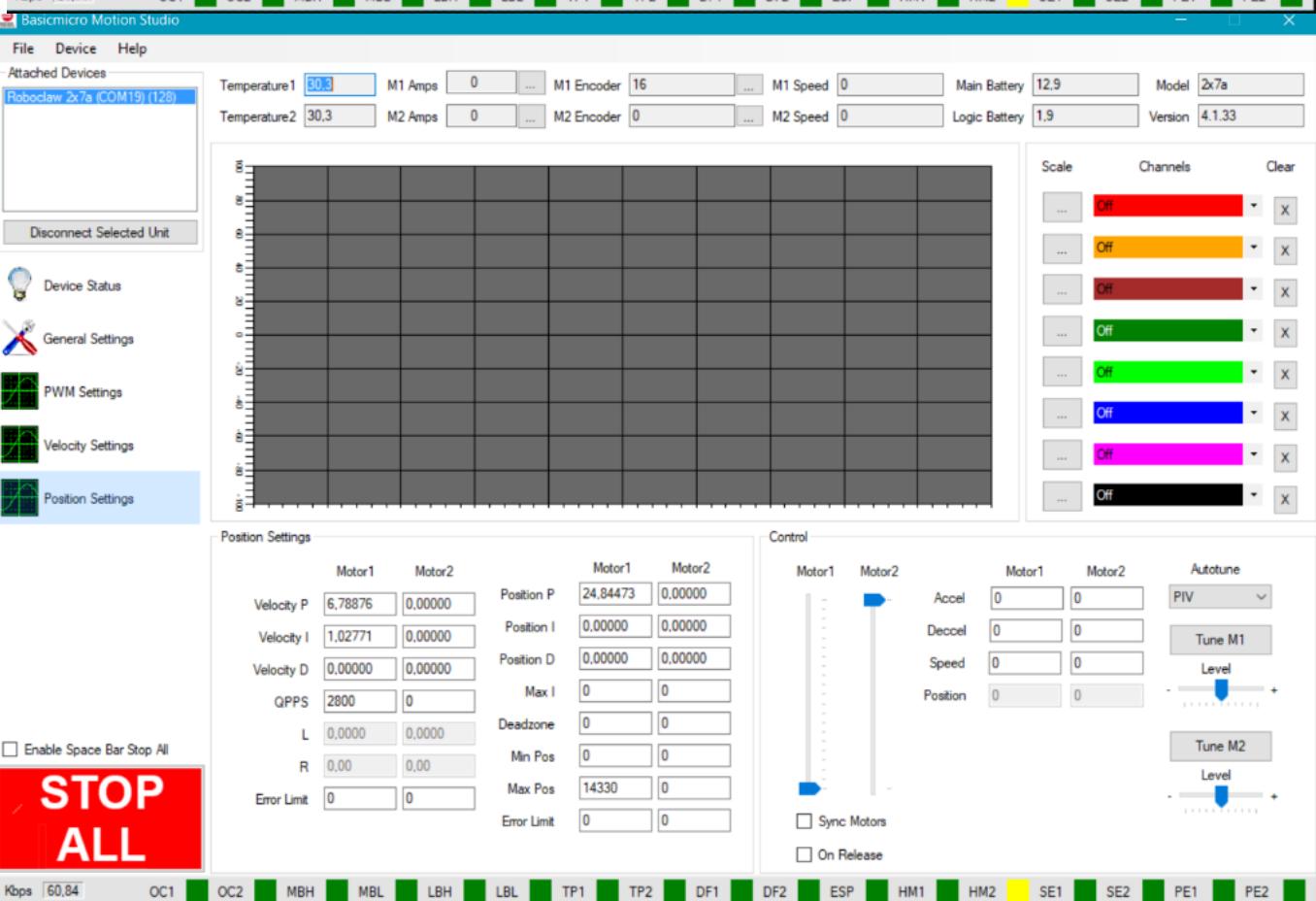
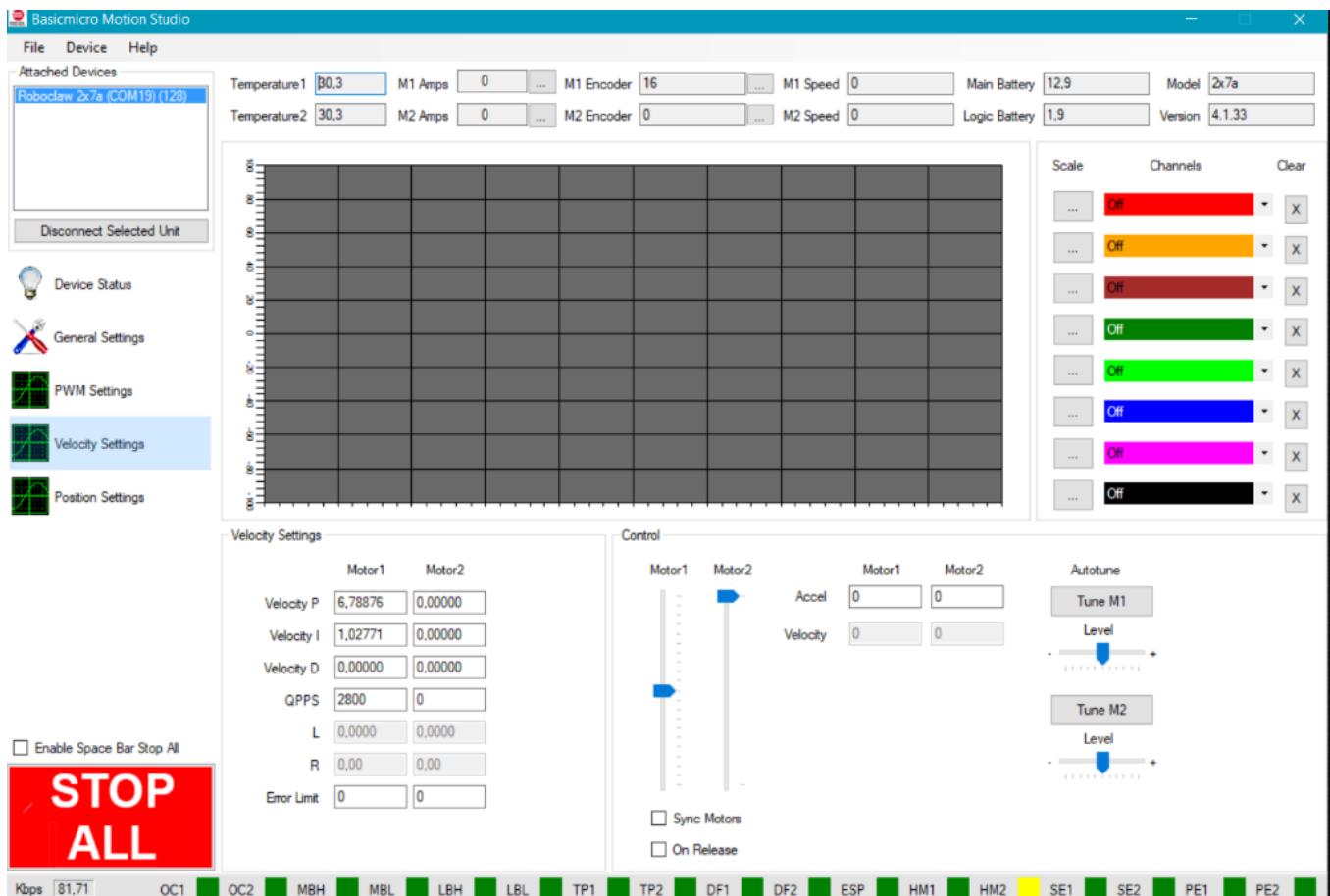


Figure 50 Schéma d'alimentation de la glissière

Ci-dessous l'interface avec les paramètres qui sont déjà dans le driver.





### 3.4.3.2 Développement

[https://github.com/yncreahdf-robotics/heron\\_software/tree/master/src/nodes/winch](https://github.com/yncreahdf-robotics/heron_software/tree/master/src/nodes/winch)

Les programmes qui font fonctionner la glissière sont `winch.py` et `winch_specs.py`

Lorsqu'on lance le programme winch.py le moteur va se recalibrer pour initialiser la position de la glissière à 0. Tant que cette opération n'est pas terminée le driver n'est pas disponible en commande.

En **mode manette**, celle-ci publie sur le topic *cmd\_vel\_winch* pour faire bouger la glissière, elle est commandée en vitesse et est bloquée par les limit switchs.

En mode **automatique**, on va pouvoir demander à la glissière d'aller à une position précise au millimètre. On publie un *Float32* correspondant à une hauteur entre 0,695 et 1,071m sur le topic *cmd\_pos\_winch*. La glissière exécutera alors la commande en allant à la position voulue.

Lorsque celle-ci bouge, sa hauteur instantanée est publiée sur le topic `winch_Height`.

```
rostopic echo /Heron0
heightTicks: 205.0
...
height: 0.700394749641
heightTicks: 205.0
...
MAXHEIGHT = 655 #mm
MAXHEIGHT = 1071 #mm

MAXSPEEDTICKS = 1700 #ticks
MAXTICKS = 14330 #ticks Max height in ticks

TICKSPERMM = MAXTICKS/(MAXHEIGHT-MINHEIGHT) #ticks for 1 mm 38,1117
MAXSPEED_M_S = (MAXSPEEDTICKS/TICKSPERMM)*0.001 # m.s-1

ACCELTICKS = 2000 #ticks
DECELTICKS = 2000#ticks
```

Une fois les fichiers .launch lancés, la commande suivante donne le résultat ci-contre en publiant la hauteur du plateau.

```
rostopic echo /Heron01/winch Height
```

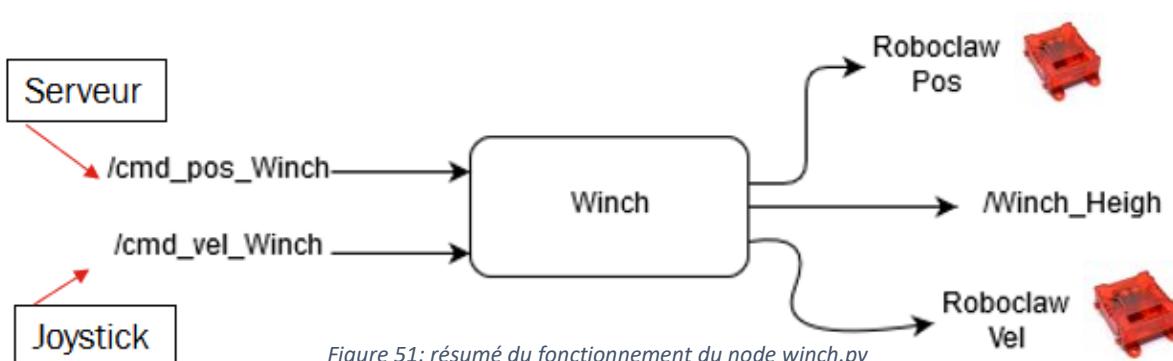
Le type de commande qu'on envoie est issu de fonctions disponibles depuis la bibliothèque du driver (roboclaw\_driver.py).

*Ex : Driver moteur à l'adresse 65 pour contrôler le moteur selon une position à atteindre avec une vitesse, accélération et décélération.*

```
SpeedAccelDeccelPositionM1(adress, accel, speed,  
deccel, position, buffer)
```

Les caractéristiques et constantes sont sauvegardées dans le fichier `winch_specs.py`

Elles sont séparées du reste du code pour pouvoir inclure le fichier dans d'autres programmes si besoin.



*Figure 51: résumé du fonctionnement du node winch.py*

*Winch\_Height* est un type de message custom ROS, il donne comme information la hauteur en mètres (height) et la distance en Ticks (heightTicks), celle-ci est utile pour débugger car on connaît la distance max parcourue en nombre de Ticks par le moteur (14330), au-delà, le driver va automatiquement bloquer la rotation du moteur dans le sens (voir settings dans l'interface).

#### Démarrage type :

Le programme *winch.py* est appelé dans *heron heronController.launch* et *heron bringUp.launch*. Le deuxième programme correspond à l'utilisation automatique de la glissière par ligne de commande, le premier à l'utilisation manuelle.

```
roslaunch heron heronController.launch
```



```
process[Her0n01/xbox_controller-3]: started with pid [3967]
process[Her0n01/controller-4]: started with pid [3968]

Initialize Drivers

RoboteQ Motor Drivers setup :
-----
Opening port: '/dev/roboteq_front'...succeeded.
Initializing port.....done.
Detecting device version...v1.8.
Opening port: '/dev/roboteq_back'...succeeded.
Initializing port.....done.
Detecting device version...v1.8.
[ERROR] [1579176627.251872282]: Couldn't open joystick /dev/v every second.
[INFO] [1579176627.366655]: Connecting to roboclaw
[INFO] [1579176627.405726]: Connected to roboclaw
[INFO] [1579176627.410804]: 4194304 0
[INFO] [1579176627.412957]: initialization : going to home
[INFO] [1579176633.680148]: winch initialized
```

Figure 52 Terminal indiquant la bonne initialisation de la glissière

le code. Si le code affiché à la place du 0 est « 8388608 » (0x80) c'est que la glissière est déjà à sa hauteur max au lancement. Un autre code indiquerait alors une erreur venant du driver.

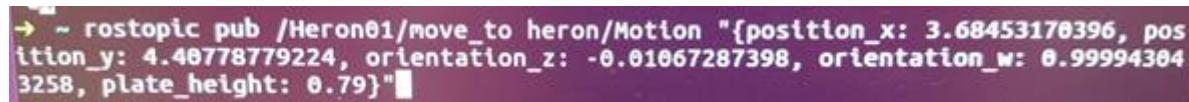
De plus, un code couleur permet de voir sur le driver si les capteurs sont déclenchés ou non. La led sera verte si les capteurs ne sont pas déclenchés, et rouge si l'un ou l'autre l'est comme ci-dessous.

Si jamais la glissière n'arrive pas à se connecter à la Jetson, une erreur est renvoyée « *global name 'port' is not defined* », le programme *winch.py* se fermera alors correctement. Et il faudra le relancer.

Si on veut envoyer une commande de hauteur (lorsqu'on est en mode automatique) :

```
rostopic pub /Heron01/cmd_pos_winch std_msgs/Float32 "data: 0.9"
```

Dans le programme de navigation l'utilisation de la glissière est encore plus automatisée, en effet si on décide d'envoyer le robot à certain goal, on devra envoyer les coordonnées ainsi que la hauteur de plateau désirée. Si l'on rentre des valeurs qui ne se situent pas dans la plage de hauteurs possibles, le max ou min seront alors appliqués.



```
→ ~ rostopic pub /Heron01/move_to heron/Motion "{position_x: 3.68453178396, position_y: 4.48778779224, orientation_z: -0.01067287398, orientation_w: 0.999943843258, plate_height: 0.79}"
```

Figure 53 Exemple de ligne de commande pour contrôler la glissière

### 3.4.3.3 Tuto

**Comment contrôler la glissière :** Lancer le programme de contrôle manuel via manette Xbox, le robot peut normalement bouger via joysticks. Pour déverrouiller la glissière il faut appuyer simultanément sur les deux gâchettes basses de la manette, ensuite il suffira d'appuyer plus ou moins fort sur l'un ou l'autre pour la faire fonctionner dans les deux sens.

Pour contrôler sa position de façon précise, il faut lancer le code de glissière (winch.py) et publier sur le topic *cmd\_pos\_wch* la hauteur voulue en mètre. On peut récupérer sa position en écoutant sur le topic *winch\_Height*.

/ ! \ Si la glissière ne détecte pas le home switch, elle va continuer de dérouler le câble, ce qui va poser un problème et elle ne pourra plus se réinitialiser car le câble sera trop déroulé.

Si tel est le cas :

- **Eteignez le driver avec le bouton situé en haut à l'intérieur du robot (interrupteur vert).**
- **Rallumez-le**
- Lancer le programme de contrôle via manette. Via le menu principal
- La glissière va vouloir descendre jusqu'à déclencher le home switch pour s'initialiser, assurez-vous qu'elle puisse le faire.
- Il sera normalement possible de recommander la glissière avec la manette. (Appuyez sur RT LT en même temps pour déverrouiller les commandes)
- La glissière est à sa plus basse position, montez-la un peu avec la manette (touche RT)
- Redescendez jusqu'à la position la plus basse (pour re initialiser LT)
- Montez ensuite la glissière à sa position la plus haute.
- La glissière est maintenant fonctionnelle et le câble enroulé.

/ ! \ Si la glissière ne répond plus aux commandes de la manette au bout d'un certain temps, relancer le programme. Source du problème non identifiée.

### 3.5 Contrôle haut niveau

#### 3.5.1 Environnement du contrôle haut niveau

Nous avons choisi d'utiliser ROS pour assurer le contrôle haut niveau car il permet énormément d'applications. Il est utilisé dans les robots Niryo et peut s'utiliser sur n'importe quel robot. Il y a des applications pour tout, par exemple, contrôler le robot avec une manette, calculer une trajectoire etc. Il possède aussi des outils de simulation puissants. De plus, on peut facilement intégrer ce que l'on a fait dans le robot dans un nouveau et en contrôler plusieurs.

Le choix de Kinetic est dû au fait que l'on a énormément de documentation sur celui-ci ainsi qu'une plus grande compatibilité au niveau des capteurs.

Voir documentation sur ROS ici : <https://www.ros.org>.

Pour savoir installer ROS Kinetic sur la Jetson TX2, suivre le tuto situé dans la spécification détaillée de la carte Jetson.



Figure 54 Illustration ROS Kinetic

### *3.5.2 Composants sur le robot permettant les applications haut niveau*

#### *3.5.2.1 Carte mère Jetson*

Pour la carte principale, nous avons choisi la Jetson TX2 8 go.

Cette carte est le cœur de notre robot qui assurera une (énorme) partie du traitement des données en bas et haut niveau.

Une grande partie des programmes informatiques assurant le fonctionnement du robot ont donc été développé sur ce robot.

Nous voulions utiliser la Jetson Nano car plus petite et moins chère mais celle-ci ne tourne que sous Ubuntu 18.04 ce qui est un problème et nous verrons pourquoi dans la suite. Le choix de la Jetson s'est fait car celle-ci est une nouveauté pour nous, elle peut tourner sous Ubuntu de plus elle a un énorme potentiel en termes de capacités graphiques, ce qui peut permettre de faire les calculs de traitement d'images et d'IA directement sur le robot.

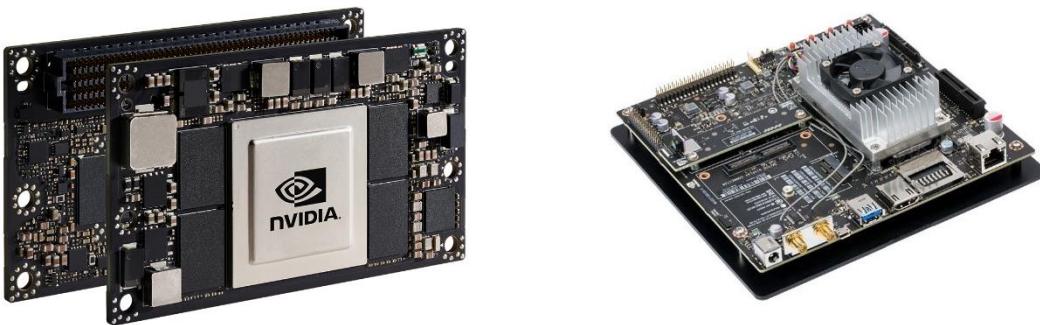
De plus, la carte possède le Wi-Fi, le Bluetooth ainsi que des GPIO, ce qui en fait une carte pratique. Elle peut aussi être connectée en Ethernet et possède une HDMI. Les seuls points négatifs sont que la carte est grande et qu'elle ne possède qu'un port USB, cependant nous allons connecter un HUB USB à celui-ci.

Elle sera alimentée directement via la batterie, on peut lui donner en entrée de 5,5 à 19V pour l'alimenter.

Spécifications Jetson TX2 :

GPU	NVIDIA Pascal™ architecture with 256 NVIDIA CUDA cores
CPU	Processeur dual-core Denver 2 64 bits et processeur quad-core ARM A57
Mémoire	8 Go 128-bit LPDDR4
Stockage	32 Go eMMC 5.1
Encodage vidéo	3x 4K @ 30 (HEVC)
Décodage vidéo	4x 4K @ 30 (HEVC)
Connectivité	Wi-Fi & Gigabit ethernet
Caméra	12 canaux MIPI CSI-2, D-PHY 1.2 (30 Gb/s)
Affichage	HDMI 2.0 / eDP 1.4 / 2x DSI / 2x DP 1.2
UPHY	Gen 2   1x4 + 1x1 OR 2x1 + 1x2, 1 port USB 3.0/USB 2.0
Dimensions	87 mm x 50 mm
Spécifications mécaniques	400-pin connector with Thermal Transfer Plate (TTP)

Figure 55 Carte Jetson TX2



Nous avons choisi d'utiliser Ubuntu 16.04 car celui-ci permet d'intégrer ROS Kinetic. De plus, Ubuntu 16.04 est une version finale d'Ubuntu, et elle n'est plus en expansion, on retrouve alors beaucoup de documentation. L'OS utilisé sur la carte est une version dérivée que l'on appelle L4T 28.2.1 (Linux For Tegra) avec JetPack 3.3.

### 3.5.2.2 Capteurs

#### 3.5.2.2.1 Justification des capteurs



Figure 56 Ensemble des capteurs utilisés

**Lidar** : Mapper l'environnement du robot et le localiser (avec fusion des données de l'IMU et de l'odométrie).

**IMU** : Localiser le robot (avec fusion des données du Lidar et de l'odométrie).

**NB** : cf. pôle asservissement pour l'odométrie.

**Caméra** : Distinguer la couleur de pastille des pièces (détection de fin de stock) et aider à la connaissance de l'environnement.

→ Un tableau de comparaison Excel est disponible dans le dossier respectif de chacun de ces capteurs.

**Compteur de Coulomb** : Surveiller l'état de la batterie en temps réel.

→ Fournit avec la batterie (cf. pôle Alimentation).

**Node IR** : Déetecter des obstacles empêchant de légers déplacements latéraux et la marche arrière.  
**Node IR-US** : Déetecter les obstacles non visibles par le Lidar gênant le plateau (eg. Tables, ...).  
**Radar** : Déetecter l'ensemble des obstacles qui feront face au robot.  
 → Récupérés du projet EDUCAT.

**NB** : Afin de simplifier et de standardiser les communications, nous avons choisi l'USB comme support principal. Toutefois, la communication entre les capteurs IR et IR-US étant nativement le bus CAN, nous avons utilisé un convertisseur CAN-USB pour les intégrer.

**ATTENTION** : Les capteur IR-US n'ont pas pu être fournis à temps à la suite de délais de fabrications trop long. Ils ont donc été momentanément remplacé par des capteurs IR.

### 3.5.2.2.2 Placement des capteurs

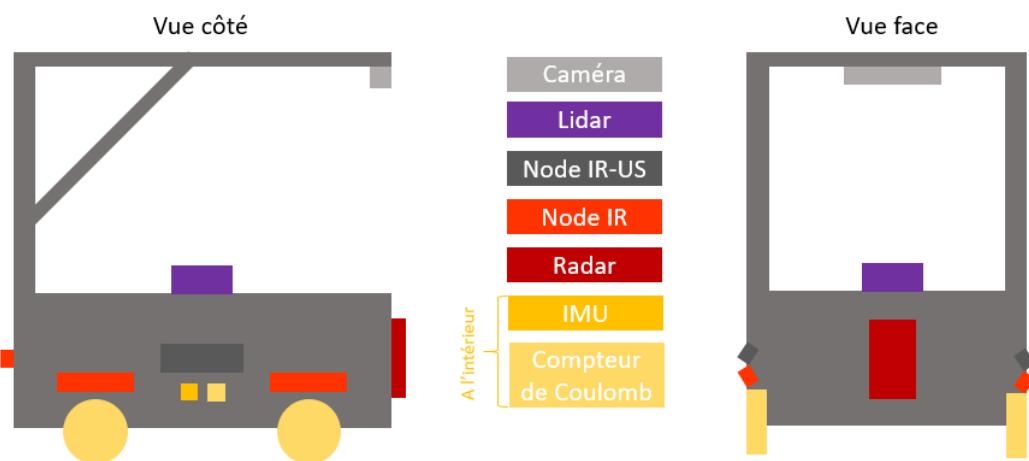


Figure 57 Schéma représentant les positions des capteurs

**Caméra** : Placé sur le dessous du plateau, à l'avant, horizontalement, pointé vers l'avant.  
 → Permet de voir l'environnement face au robot.  
 → Permet de distinguer les pastilles des pièces stockés sur les tables (mise en dessous du plateau pour ne pas gêner la mise en place des plateaux modulables).

**Lidar** : Sur le dessus de l'étage 1, au centre, orienté vers l'avant, avec un champ de vision horizontal dégagé.  
 → Obtenir une connaissance optimale de l'environnement entourant le robot (Distinguer les principaux composants empêchant son déplacement et éviter au plus les siens).  
 → Associer le repère du robot à celui du lidar sur le plan horizontal dans les simulations.

#### Nodes IR-US → Nodes IR :

- 2 capteurs sur les faces latérales (1 par face) :  
 Placé , bord supérieur, 22°.

#### Nodes IR :

4 capteurs sur les faces latérales (2 par face).

- Placé au-dessus de chaque roue, avec un angle de 25° vers le bas, qui permet de voir à 10 cm de la roue, centré avec la face, 5.5 cm.

- 1 capteur sur la face arrière :  
Placé au centre de l'étage 0, horizontalement, pointé vers l'arrière.

**Radar** : Placé sur la face avant, au centre de l'étage 1, horizontalement, pointé vers l'avant

**IMU** : Placé au centre de l'étage 1, orienté vers l'avant.

**Compteur de Coulomb** : Placé au plus près de la batterie, avec l'écran ramené sur le côté latéral le plus proche.

### 3.5.2.2.3 Organisation des programmes et modularité

L'une des contraintes de ce projet était de le rendre le plus modulable possible. Nous avons donc réfléchi à une architecture logicielle permettant :

D'ajouter ou de retirer facilement les différentes composantes du robot.

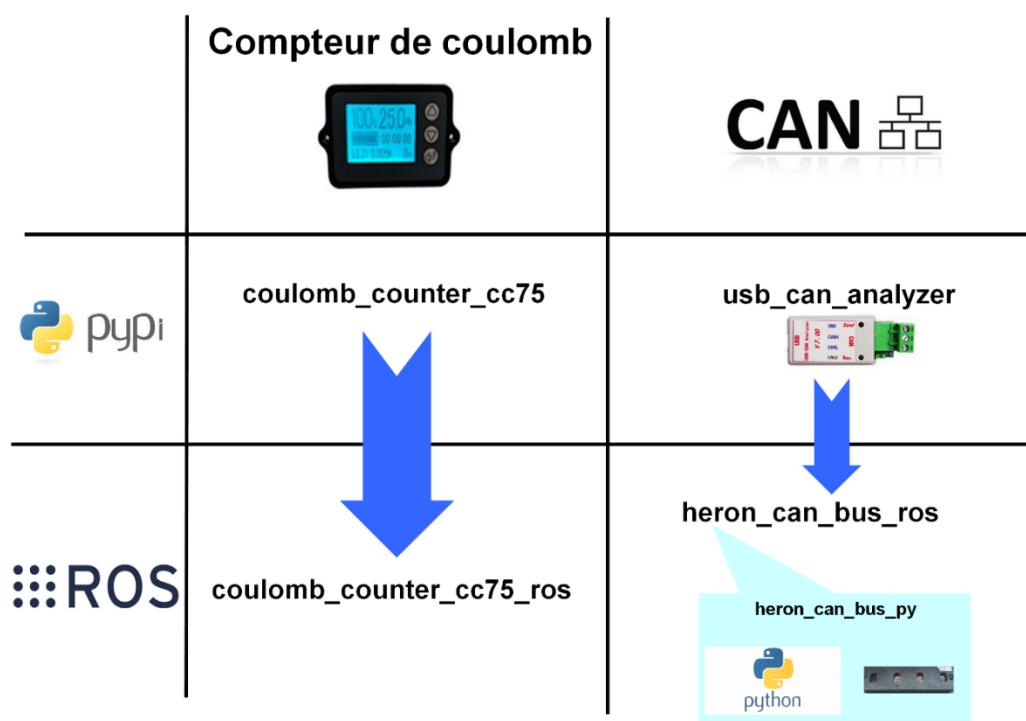
De récupérer facilement le software lié à l'une de ces composantes pour d'autres projets. Cette contrainte est l'une des raisons pour lesquelles nous avons choisi de développer le software avec ROS.

En ce qui concerne les capteurs, nous pouvons les distinguer en 2 catégories :

Ceux qui disposent déjà d'un package ROS qu'il a fallu intégrer et adapter à nos besoins (le LIDAR, l'IMU et le RADAR)

Ceux qui ne disposaient pas d'un software et donc pour lesquelles nous avons dû en développer un (le compteur de coulomb, les capteurs IR et IR-US et le convertisseur CAN-USB).

Voici comment nous avons organisé les softwares qu'il nous a fallu développer :



Pour permettre la réutilisation des capteurs ci-dessus dans d'autre projet, nous avons d'une part créée des bibliothèques Python et d'autre part créé des packages ROS s'appuyant sur les -dites bibliothèques. Ainsi, les bibliothèques Python « coulomb\_counter\_cc75 » et « usb\_can\_analyzer » sont disponibles via le gestionnaire de paquet 'pip' afin de faciliter leur réutilisation respective. Ces bibliothèques python n'implémentent aucun aspect de ROS (nœud, topic, etc...), les rendant donc utilisables tel quel pour diverses applications.

De plus, 2 packages ROS reprenant les bibliothèques python citées ci-dessus, ont été créés :

- L'un pour le compteur de coulomb, réutilisant la bibliothèque python et implémentant les aspects de ROS dans le but de rendre les données de la batterie accessible sur les topics ROS.
- L'autre, quant à lui, reprend le fonctionnement global du bus CAN, implémentant donc l'utilisation du convertisseur USB-CAN ainsi que le fonctionnement des capteurs IR et IR-US. D'une part, ce package réutilisera donc la bibliothèque python du convertisseur et, d'autre part, une bibliothèque python a été créée à l'intérieur de ce même package afin d'implémenter les aspects de ROS et le traitement des données des capteurs IR et IR-US.

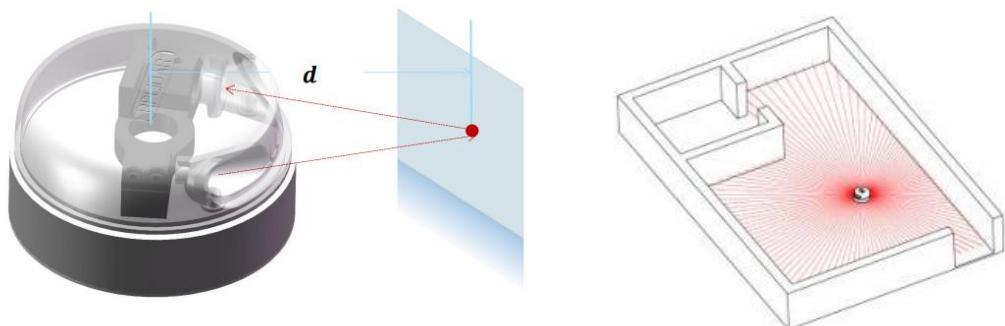
→ Cette bibliothèque a été volontairement mise à l'écart de 'pip' car les capteurs IR et IR-US sont spécifiques au projet, il n'y a donc pas de raison de rendre ce code accessible en dehors du package ROS.

Plus de détail sur les packages ROS cités ci-dessus sont donnés respectivement dans les dossiers « Capteur de coulomb » et « Bus CAN ».

#### 3.5.2.2.4 Lidar



Figure 58 RPLidar A3



Le lidar est un capteur permettant de connaître les différents obstacles autour du robot. Il servira aussi pour le « mapping » et la détection dynamique durant les déplacements du robot.

Le lidar tourne sur lui-même afin de mesurer la distance entre le lidar et l'objet le plus proche. Connaissant la vitesse maximale du robot il est nécessaire que le lidar tourne suffisamment vite afin de détecter les objets.

Le « mapping » se fait à l'aide du lidar, une fois en route, il faut faire se déplacer le robot et enregistrer tous les points mesurer. Cela va permettre de créer une carte de l'environnement dans lequel le robot s'est déplacé.

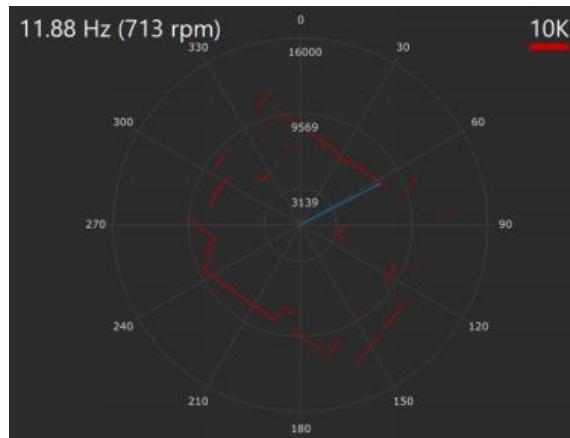


Figure 59 Mapping d'une pièce



Figure 60 Branchement du Lidar

Le LIDAR est alimenté en 5V grâce au HUB USB qui est relié à la carte JETSON

Référence : RPLIDAR-A3M1

Type de communication : USB

Balayage : 5 à 20 Hz

Plage angulaire : 360°

Dimension : 76 mm de diamètre et 41 mm de hauteur

Poids : 190 g

Package ROS : [https://github.com/Slamtec/rplidar\\_ros](https://github.com/Slamtec/rplidar_ros)

Datasheet :

[http://bucket.download.slamtec.com/aaf96dddba2f6a9baa03261628c01af9fc2f866c/LD310\\_SLAMTEC\\_rplidar\\_datasheet\\_A3M1\\_v1.0\\_en.pdf](http://bucket.download.slamtec.com/aaf96dddba2f6a9baa03261628c01af9fc2f866c/LD310_SLAMTEC_rplidar_datasheet_A3M1_v1.0_en.pdf)

Branchement : Le câble du lidar doit être branché à l'adaptateur. L'adaptateur doit être branché sur un port USB avec le câble prévu à cet effet. Le câble d'alimentation peut être branché pour avoir une alimentation externe, dans le cas contraire, l'alimentation passe par le câble USB. (cf. Figure 46)

Récupération des données sur l'ordinateur, prérequis :

Pour utiliser le lidar, il est nécessaire d'avoir installé ROS et catkin sous ubuntu 16.04 puis d'avoir créé le catkin workspace. Enfin, il faut télécharger le package « rplidar\_ros » dans votre catkin\_ws/src/.

Utilisation du package :

La première étape consiste à build le nouveau package ros grâce à la commande « catkin build » ou « catkin\_make » depuis votre dossier catkin\_ws/.

Vous pouvez maintenant lancer la commande « roslaunch rplidar\_ros view\_rplidar\_a3.launch » qui ouvrira automatiquement RViz ce qui permettra de visualiser facilement les données du lidar et donc, les obstacles qui l'entourent

Filtrage des données

Le lidar ayant besoin de rester à une hauteur fixe, nous ne pouvions pas le mettre sur le plateau. Il a donc été nécessaire de le placer sur la base mobile et d'ignorer certaines données qui sont des parties du robot détectées par le capteur (ex : les barres verticales de la glissière pour le plateau). Pour cela nous avons pu utiliser le package « laser\_filters\_ » de ROS permettant de créer des filtres aisément et ignorer les câbles/composants lus dans le robot.

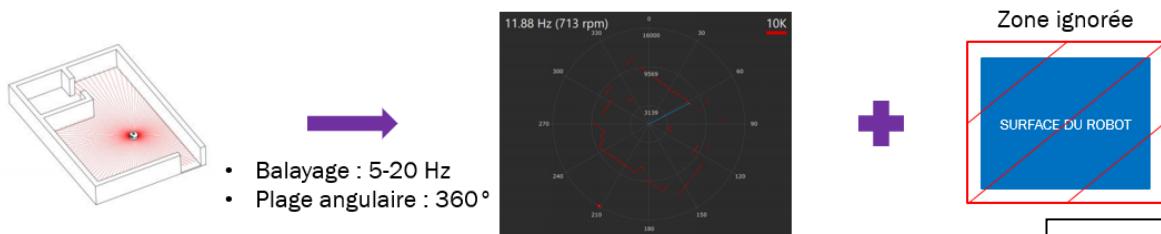


Figure 61 Schématisation du filtrage des données

Ce cadre rouge représente le filtre mis en place. Toutes les valeurs lues par le lidar contenues à l'intérieur de ce rectangle sont ignorées.

### 3.5.2.2.5 IMU

L'IMU permet de mesurer l'accélération, la rotation et le champ magnétique. Avec ces données on peut calculer la vitesse de déplacement et la position du robot.

Grâce à un filtre de Kalman intégré les mesures de l'IMU sont très précise et permettront d'améliorer les estimations de position en fusionnant les données de l'odométrie et de l'IMU.



Figure 62 Centrale Inertielle (IMU)



Figure 63 Branchement de l'IMU

Caractéristiques techniques et alimentation :

Wired interface	USB 2.0
Max. baudrate	921.6kbit/s
Communication protocol	LpBus
Size	40 x 34 x 17 mm
Weight	36 g
Orientation range	Roll: ±180°; Pitch: ±90°; Yaw: ±180°
Resolution	< 0.01°
Accuracy	< 0.5° (static), < 2° RMS (dynamic)
Accelerometer	3-axis, ±2 / ±4 / ±8 / ±16 g, 16 bits
Gyroscope	3-axis, ±125 / ±245 / ±500 / ±1000 / ±2000 dps, 16 bits
Magnetometer	3-axis, ± 4 / ± 8 / ± 12 / ± 16 gauss, 16 bits
Pressure sensor	300-1100 hPa
Data output format	Raw data / Euler angle / Quaternion
Data output rate	up to 400Hz
Power consumption	< 182 mW @ 3.3V
Power supply	5 V DC
Connector	Micro USB, type B
Case material	Aluminum
Temperature range	- 40 ~ +80°C
Software	C++ library for Windows, LpmsControl software and Open Motion Analysis Toolkit (OpenMAT) for Windows.

/ ! \ Attention !! Pour pouvoir l'utiliser sur la Jetson TX2, veuillez suivre le tutoriel présent dans la partie « 5.3.5.3.4 Intégration IMU dans Jetson TX2 ». Cette prise en main ne convenant qu'à un PC ayant Ubuntu.

Pour commencer il faut télécharger la librairie LpSensor disponible sur le lien suivant : <https://lp-research.com/support/>

Puis installer la librairie **LpSensor**

```
tar xvzf ~/Downloads/LpSensor-1.3.5-Linux-x86-64.tar.gz  
sudo apt-get install libbluetooth-dev  
sudo dpkg -i LpSensor-1.3.5-Linux-x86-64/liblpsensor-1.3.5-Linux.deb  
dpkg -L liblpsensor
```

Ensute il faut mettre en place un environnement de travail de type **catkin** (optionnel si un environnement catkin est déjà en place)

```
mkdir -p ~/catkin_ws/src  
cd !:2  
catkin_init_workspace
```

Maintenant il faut **télécharger** puis **compiler** le driver pour l'IMU LPMS

```
cd ~/catkin_ws/src  
git clone https://github.com/larics/timesync\_ros.git  
git clone https://github.com/larics/lpms\_imu.git
```

```
cd ~/catkin_ws  
source devel/setup.bash  
catkin build
```

Il ne reste plus qu'à lancer les commandes ROS suivantes pour visualiser les informations de l'IMU  
`rosrun lpms_imu lpms_imu_node`

(`< rosrun rqt_plot rqt_plot >` permet juste la visualisation des donnes sur un graphique)  
(`< rostopic echo /imu >` permet de voir les données brutes sur un terminal)

Plus d'information sont disponibles avec le lien suivant : <https://lp-research.com/ros-and-lp-research-imus-simple/>

### 3.5.2.2.6 Capteurs Infrarouge



Figure 64 Node IR

La Node IR comporte 4 capteurs IR qui permettent de détecter des obstacles à courte distance dans un angle restreint.



Figure 65 Branchement Node IR

Référence : EDUCAT Distance Node D2 (4 IR)

Nombre de capteurs IR : 4

Dimension : 9.5 x 2.5 x 1 cm

Angle de détection : 27° (pour chaque capteur, 2.5 cm entre chacun d'eux)

Branchement : CAN (VCC – GND – CANH – CANL)

Alimentation : 12V – 0.03A

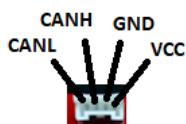
Type de communication : CAN

Fréquence d'envoie des données : 500 000 bps

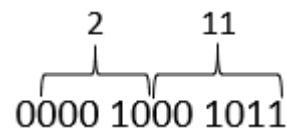
Branchement

Les nodes sont tous reliés entre elles via un bus CAN pour la communication et l'alimentation, avec au bout une partie allant se relier au système général d'alimentation du robot et une autre allant vers le convertisseur USB-CAN afin de transmettre les données. Le type de connecteur présent sur les nodes est le JST PA - 4 pôles - 2 mm - 1 rangée.

Les données transmises



L'ID des messages est de type standard, donc codé sur 11 bits. Toutefois, celui-ci est composé en 2 parties :



La première, codée sur 6 bits, correspond au numéro de la node tandis que la seconde, codée sur 5 bits, correspond au service souhaitée / à la nature du message.

La longueur et le contenu du payload dépendra du service indiqué précédemment, mais dans notre cas, celui-ci sera principalement du service 2, ce service communiquant les informations mesurées par la node.

Dans le cas de ce service, un message de type « remote » avec aucun payload devra être envoyé à la node dont on souhaite recevoir les valeurs, et, en retour, un message de type « data » avec le même ID et un payload sera transmis. Ce payload sera composé de 6 octets. Les 2 premiers octets correspondant à la valeur la plus petite mesurée tandis que chacun des 4 octets suivants correspondra à la mesure de chaque capteur de la node. Les valeurs indiquées sont en centimètres.

### 3.5.2.2.7 Camera

La caméra ZED permet de filmer en 3D grâce à ses 2 capteurs optique afin de reconnaître la distance entre la caméra et l'objet filmé avec un Frame Rate pouvant aller jusqu'à 100 images par seconde.



Figure 66 Caméra ZED

Référence : ZED Stéréo caméra

Type de communication : USB

Sortie vidéo : 2.2K, 1080p, 720p et WVGA

Image par secondes (respectivement) : 15, 30, 60 et 100

Résolution vidéo (respectivement) : 4416x1242, 3840x1080, 2560x720 et 1344x376

Distance vidéo : 0.3 – 25 mètres

Dimension : 175 mm de largeur, 30 mm de hauteur et 33 mm d'épaisseur

Poids : 135 g

Alimentation : USB 5V / 380mA

La caméra ZED permet de récupérer les images de la vision du robot, ces images seront ensuite traitées sur la JETSON avec la librairie OpenCV.

Pour commencer et installer la caméra sur un PC ou sur la Jetson TX2, suivre les instructions suivantes : <https://www.stereolabs.com/docs/installation/>. Attention, pour notre carte, veuillez installer la version « 2.8.5 ZED SDK for Jetpack 3.3 » car nous utilisons Ubuntu 16.04 disponible dans Jetpack 3.3 (le SDK de la Jetson [https://developer.nvidia.com/embedded/jetpack-3\\_3](https://developer.nvidia.com/embedded/jetpack-3_3)). SDK disponible ici : <https://www.stereolabs.com/developers/release/2.8/>.

Utilisation avec ROS : <https://www.stereolabs.com/docs/ros/>.

### 3.5.2.2.8 Capteur de Coulomb

Le capteur de coulomb CC-75 sert à acquérir les différents paramètres de la batterie en temps réel. C'est un petit écran indiquant divers chiffres tels que tension, puissance instantanée, courant instantané...



Figure 67 Capteur de Coulomb

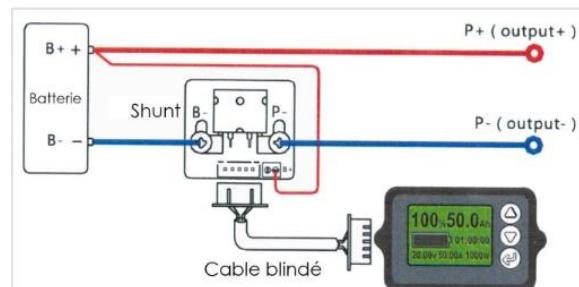


Figure 68 Connexion du capteur de Coulomb

Etat de charge (%) Capacité restante (Ah) Temps restant (HH: MN: SS) Puissance instantanée (W)  
 Courant (A) Tension (V) Vue graphique de l'état de charge de la batterie (SoC)

<https://www.mylithiumbattery.com/fr/shop/accessoires/accessoires-fr/compteur-de-coulomb/> Celui-ci est accompagné d'un module permettant de convertir l'information du capteur en données USB pour communiquer avec la carte mère. Son utilisation sera détaillée par la suite expliquant comment récupérer les informations du capteur via ROS.

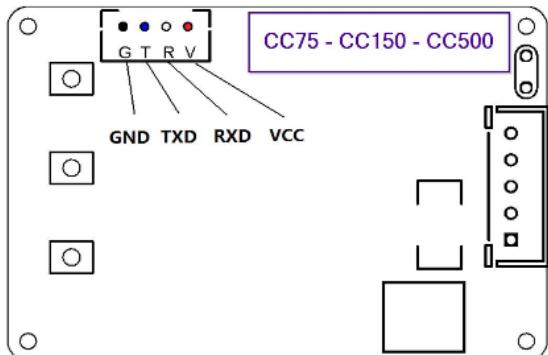
<https://www.mylithiumbattery.com/fr/shop/accessoires/accessoires-fr/kit-de-communication-pour-compteur-de-coulomb-cc75-150-et-500-2/>

### CC75/CC150/CC500 TTL serial communication |



Baud rate :	19200Kbps
Bytes :	17 bytes each time
Interval :	One second Only send when coulometer working(Backlight on)
Interface level :	TTL level 3V Disconnect VCC is better, the max current is 5mA
Isolation method :	In order to protect Coulomb counter,you'd better connect an optocoupler to the send pin.
Check sum:	8 bit cumulative sum from the first byte to fifteenth byte

Byte num:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Meaning:	First bytes	Capacity percentage	Total battery voltage (V)	Real time battery capacity (mAh)	Current (mA)	Remaining time (S)	Check sum										
Hex:	xx	02	3E	05	87	0A	00	00	05	24	00	00	11	94	00	00	xx
Example:	xx	0x02=2%	3E05=0x053E=1342+13.42V	87 0A 00 00=0xA87=2695mAh=2.695Ah	05 24 00 00=0x2405=9221mA=9.221A	11 94 00 00=0x9411=379055=10H:31M:45S											
Range:	xx	0 - 100%	0 - 500.00V	0.1Ah - 5000Ah	-750000mA - +750000mA(Negative value is complement)	00.00:00 - 99:59:59											



RXD : TK15 receive pin  
 TXD : TK15 send pin  
 GND: Ground  
 VCC : 3V power supply, max 5mA

Note: The interface is not RS-232. So it can't be connected directly. It must be TTL level.  
 Usually only connect TXD and GND is OK.

Figure 69 Spécification de communication du capteur de Coulomb

### 3.5.2.2.9 Ecran

L'écran est un simple display sur lequel apparaîtra le menu permettant le contrôle du robot.

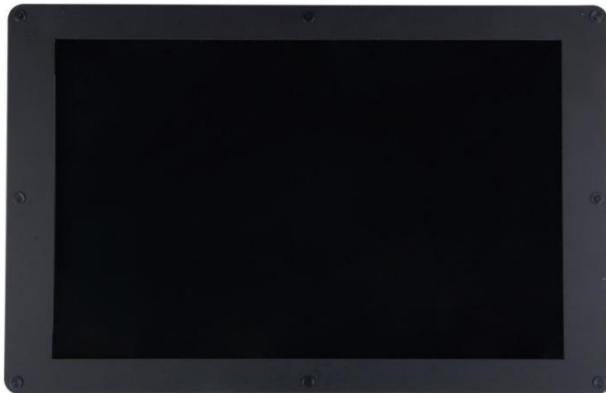


Figure 70 Ecran 10.1"

- Résolution: 1 980 x 1 200 px
- Taux de rafraîchissement: 60 HZ
- Type de panneau: Panneau ISP (angle de vision double 178 °)
- Puissance: interface standard micro usb
- Courant de travail: 5 V 600 mA
- Plateforme de support: Raspberry Pi, Windows, etc. Interface HDMI
- Interface: HDMI 2.0 (compatible avec HDMI1.4)
- Matériaux du cadre: Plaque acrylique colorée

- Pièce jointe: câble HDMI 2.0, câble micro USB, support arrière, vis de montage framboise pi
- Autres fonctions: interrupteur d'alimentation du rétroéclairage, réglage de la luminosité du rétroéclairage

### 3.5.3 Applications Haut Niveau du Robot Héron

Maintenant l'ensemble des composants présentés, leur prise en main individuelle et leur mode de communication expliqués, nous allons à présent détailler les différentes applications qualifiées de haut niveau réalisées par notre robot tel que la commande par manette du robot Héron, la réalisation d'une session de mapping ou encore la navigation autonome du robot.

Cependant toutes ses applications sont évidemment réalisées par le robot Héron, mais elles sont assistées et dirigées par le serveur ROS, présent sur le PC Central, que nous n'avons pas encore eu l'occasion de présenter.

#### 3.5.3.1 Serveur ROS

Nous avons choisi d'utiliser ROS car il permet énormément d'applications. Il est utilisé dans les robots Niryo et peut s'utiliser sur n'importe quel robot. Il y a des applications pour tout, par exemple, contrôler le robot avec une manette, calculer une trajectoire etc. Il possède aussi des outils de simulation puissants. De plus, on peut facilement intégrer ce que l'on a fait dans le robot dans un nouveau et en contrôler plusieurs.

Le choix de Kinetic est dû au fait que l'on a énormément de documentation sur celui-ci ainsi qu'une plus grande compatibilité au niveau des capteurs.

Voir documentation sur ROS ici : <https://www.ros.org>.

Le serveur ROS a pour but de piloter une flotte de robot. Et dans notre cas notre robot Héron. C'est ce serveur qui stock la map (représentation 2D) de l'environnement, localise le robot sur celle-ci, génère la trajectoire du robot et le pilote pour qu'il atteigne son goal. Ce qui fait de lui l'interface entre le serveur central qui ordonne le déplacement du robot et le robot lui-même.

En quelques mots, Le serveur ROS ou autrement dit ROS MASTER est le marionnettiste et Héron la marionnette.

Nous avons pris comme décision de déporter ce serveur ROS sur le PC central pour des raisons de capacité de calcul, la Jetson TX2 ayant évidemment des capacités de calcul inférieure à notre PC central.

On peut donc affirmer que le robot Héron et le serveur ROS sont en permanence en train d'échanger des informations.

Le robot Héron envoie constamment les données lues par ses capteurs vers le serveur ROS, et selon les applications souhaitées, le serveur ROS envoie certaines commandes au robot Héron (comme se déplacer tout droit, changer de trajectoire si un obstacle se présente devant le robot pendant sa navigation autonome).

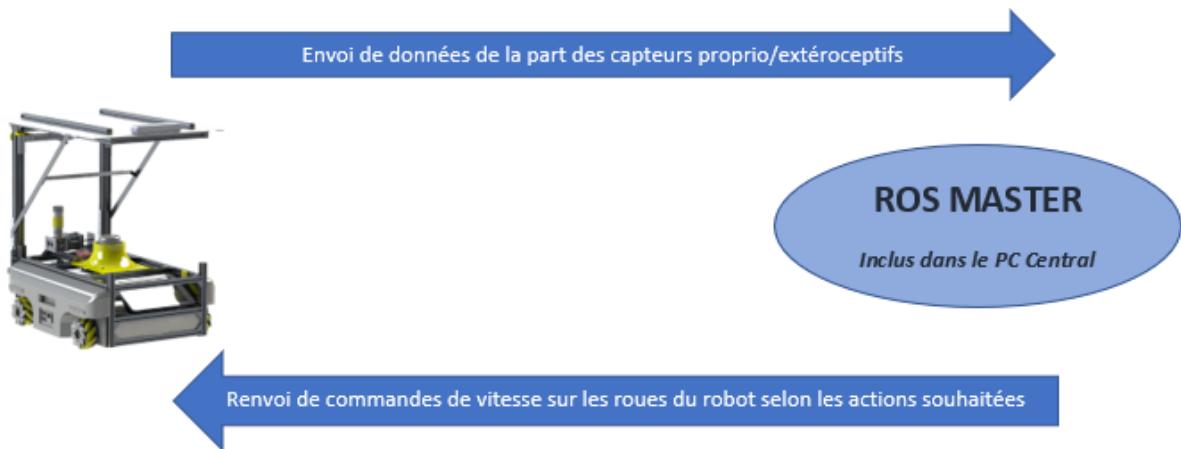


Figure 71 Schématisation des échanges entre Héron et le ROS Master

### 3.5.3.2 IHM de contrôle et organigramme Software associés

Afin de pouvoir activer les différents modes possibles concernant le robot, nous avons développé des interfaces graphiques de manière à simplifier au maximum la prise en main de celui-ci.

Vous allez constater par la suite qu'il existe 2 interfaces graphiques, très similaires.

En effet, une première développée sur le robot et une seconde développée sur le ROS Master (et donc sur le PC Central).

En effet, lorsque l'on souhaite entrer dans un mode avec le robot (par exemple le mode « Navigation autonome »), il est nécessaire de lancer certains programmes sur le robot Héron lui-même (via sa propre interface), mais également certains programmes sur le ROS Master (via sa propre interface). C'est pourquoi dans certains modes, il sera nécessaire pour le bon fonctionnement, de se rendre sur les 2 interfaces graphiques et de choisir les modes adaptés.

En plus de vous proposer ici une vision des interfaces graphiques, vous pouvez également trouver un organigramme logique dans lequel vous pourrez retrouver ce qui se passe lorsque l'on choisit tel ou tel mode.

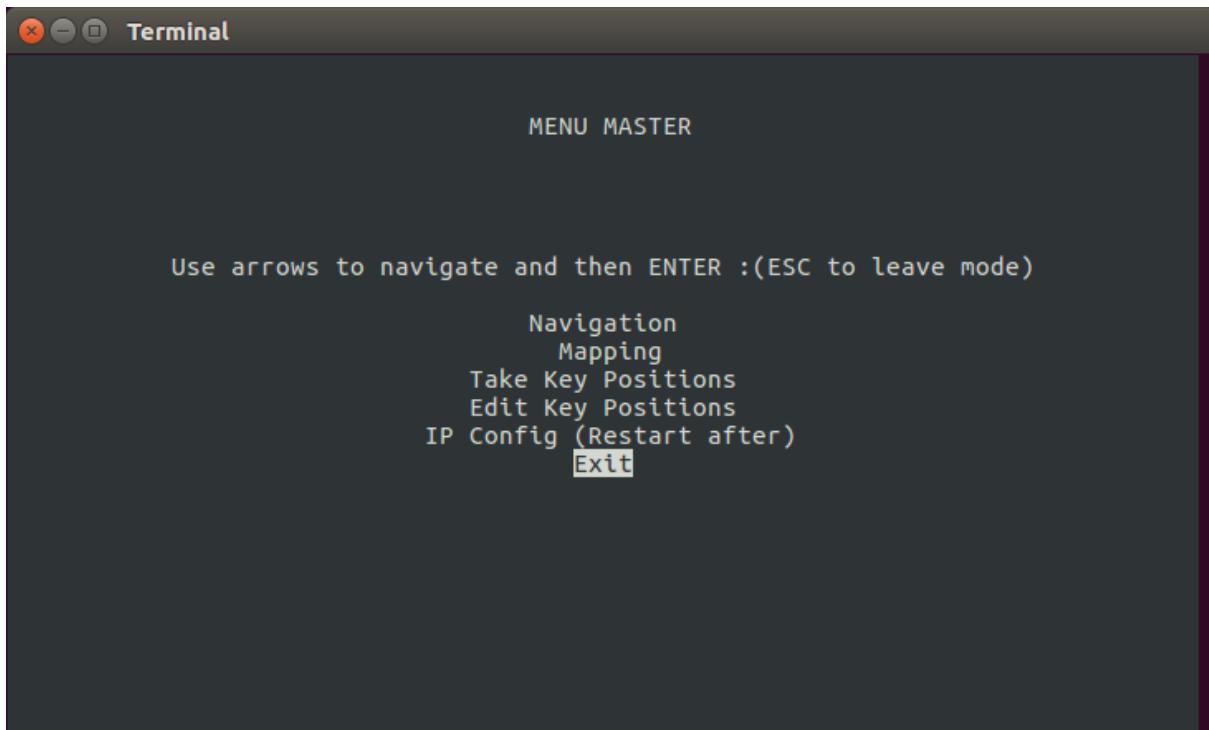


Figure 72 IHM sur PC Central

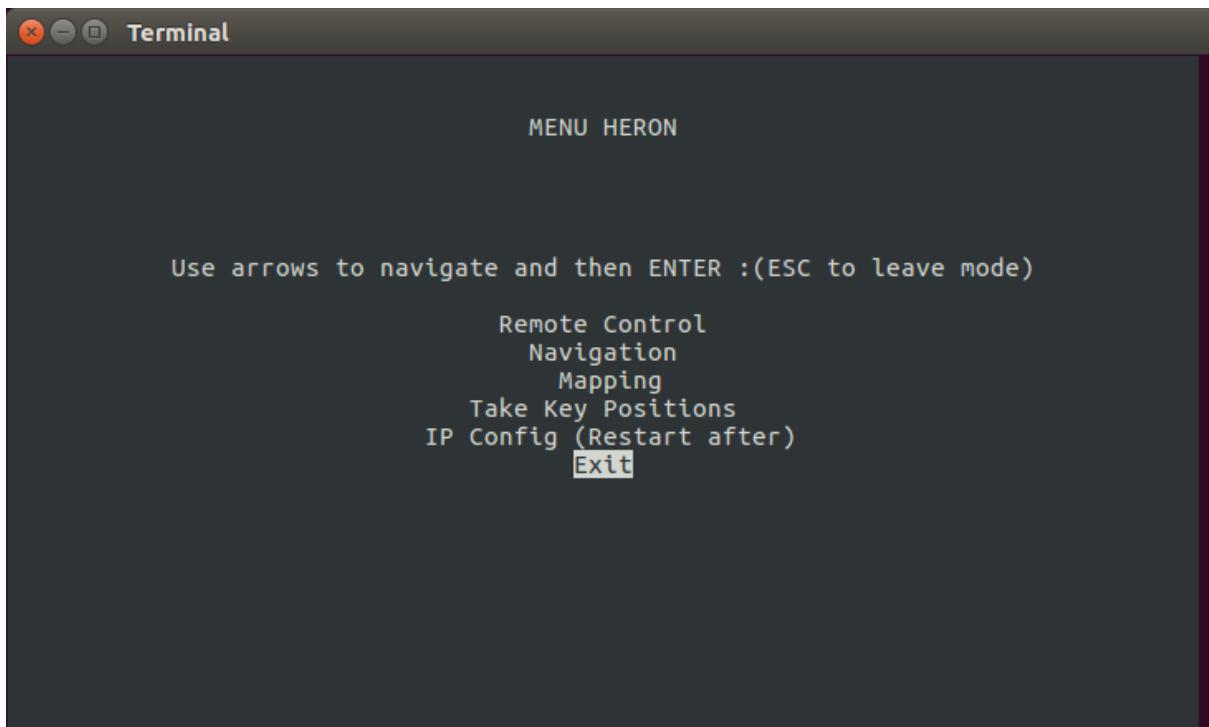


Figure 73 IHM sur Héron

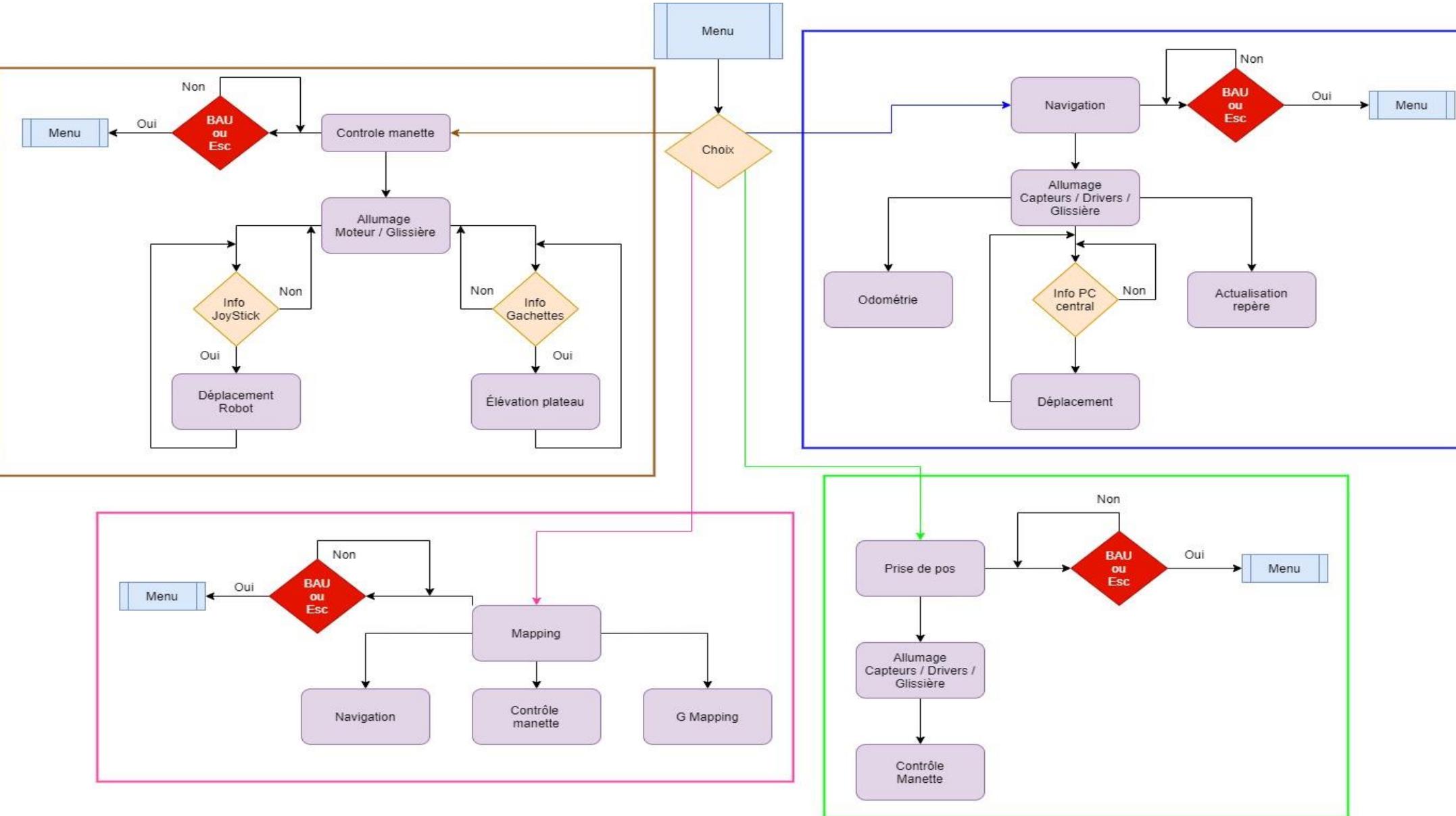


Figure 74 Organigramme software Héron

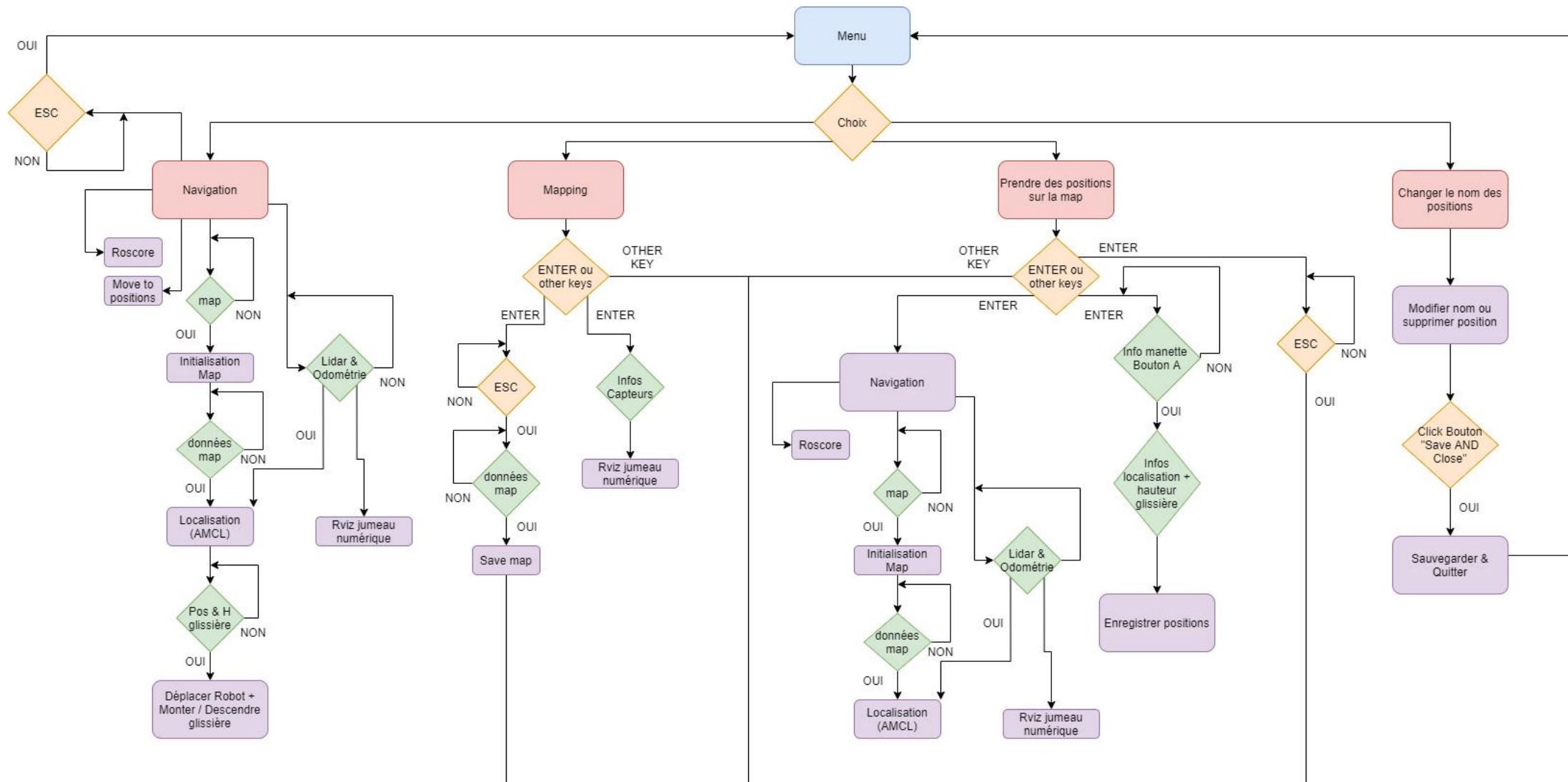


Figure 75 Organigramme software PC Central

### 3.5.3.3 Découverte des différents modes et fonctionnalités du robot

Pour la suite de cette partie, nous allons nous servir des interfaces graphiques du robot Héron et du ROS master pour aborder l'ensemble des fonctionnalités possibles avec le robot avec un ordre bien précis pour faciliter la compréhension.

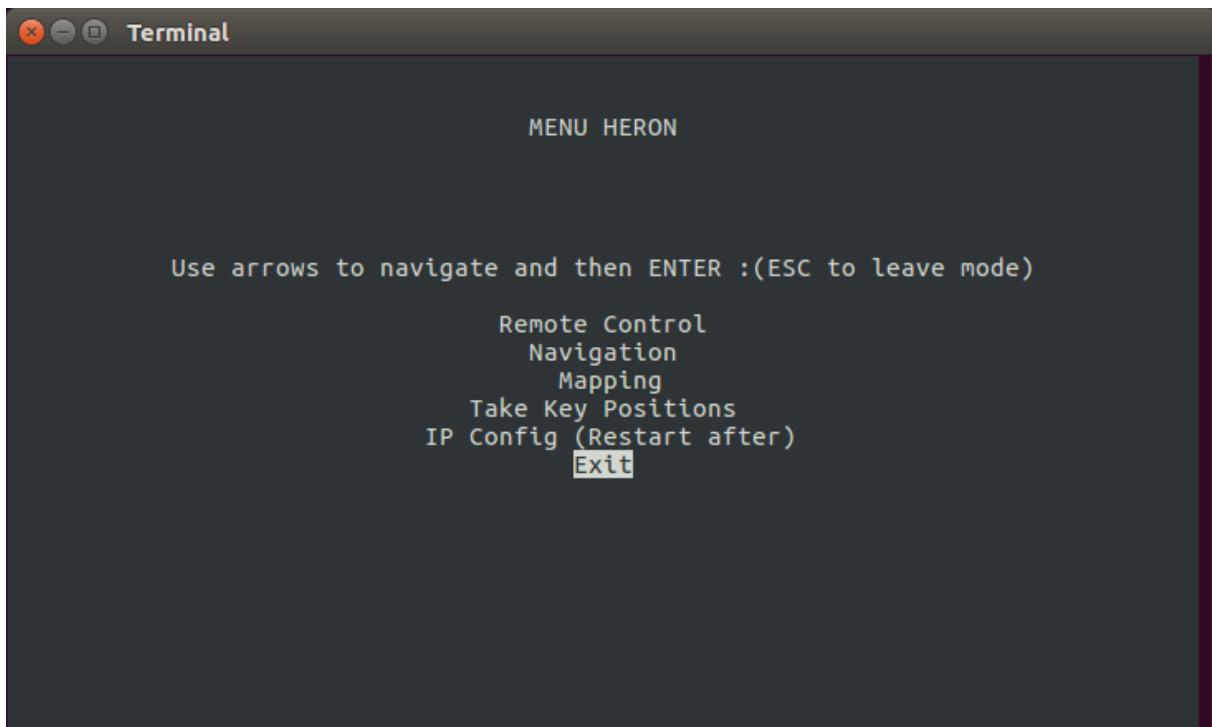
Pour chaque mode, une explication sera apportée sur l'objectif de celui-ci, les capteurs/actionneurs/algorithmes utilisés ainsi que les programmes lancés en arrière-plan.

#### 3.5.3.3.1 Mode « Remote control »

##### 3.5.3.3.1.1 Interface - Héron

Objectif de ce mode : Permettre à l'utilisateur d'avoir un contrôle total sur les déplacements du robot grâce à une manette de XBOX, via communication bluetooth. (Cette application est très utile dans le cadre de journées portes ouvertes pour réaliser une simple démonstration des déplacements possibles du robot).

Afin d'activer ce mode, il suffit uniquement de se rendre sur l'interface graphique du robot Héron et de choisir le mode « Contrôle par manette ». En effet, nous n'avons nullement besoin du ROS Master ici pour assurer ce type d'application.



Ce mode ne fait appel à aucun capteur particulier, uniquement aux actionneurs et l'asservissement développé sur le robot.

Information importante concernant l'asservissement mis en place pour le robot.

Celui-ci est étant équipé de roues mecanums qui lui permettent de rouler dans toutes les directions, sa commande se fait selon les lois suivantes :

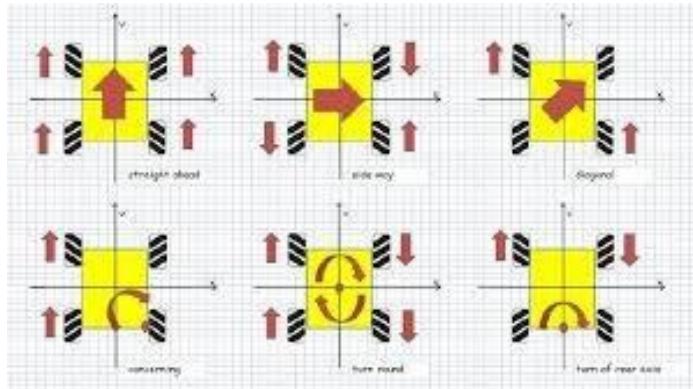


Figure 77 Mouvement du robot en fonction de la commande des moteurs.

$$\begin{cases} \omega_1 = \frac{1}{r}(v_x - v_y - (l_x + l_y)\omega), \\ \omega_2 = \frac{1}{r}(v_x + v_y + (l_x + l_y)\omega), \\ \omega_3 = \frac{1}{r}(v_x + v_y - (l_x + l_y)\omega), \\ \omega_4 = \frac{1}{r}(v_x - v_y + (l_x + l_y)\omega). \end{cases}$$

Figure 76 Equations de commande du robot

```

275 {
276     //Init the ROS node named "drive"
277     ros::init(argc, argv, "drive");
278
279     Driver drivers;
280
281     drivers.connect();
282
283
284     ros::Rate r(100);
285     while (ros::ok() && drivers.getStatus())
286     {
287         ros::spinOnce();
288
289         drivers.pubEncoders();
290
291         r.sleep();
292     }
293
294     drivers.disconnect();
295
296     return 0;
297 }
```

Pour la commande du robot que ce soit en mode manuel ou autonome sur ROS cela passe par le type de message Twist. Nous allons donc détailler le programme qui prend ce message en entrée et en sortie fait bouger le robot.

Rappel : Twist = Vector : Linear(x,y,z), Vector : angular(x,y,z)

drive.cpp

Comme précédemment expliqué ce programme fait bouger le robot. En effet il crée la connexion avec les drivers ce qui permet de leur envoyer des consignes à appliquer aux moteurs. Mais cela permet aussi de récupérer les données des encodeurs afin de les publier pour l'odométrie. Tout ceci est géré par la classe Drivers.

Nous allons voir la commande des moteurs :

```
void setCommands(const geometry_msgs::Twist& msg)
{
    float frontRightSpeed;
    float frontLeftSpeed;
    float backRightSpeed;
    float backLeftSpeed;

    float thetad = msg.angular.z;

    //Equations for mecanum wheeled robot

    frontLeftSpeed = saturate((1 / WHEEL_RADIUS) * (msg.linear.x - msg.linear.y - thetad * (WTOW_LENGTH + WTO_WIDTH)) *
    frontRightSpeed = saturate((1 / WHEEL_RADIUS) * (msg.linear.x + msg.linear.y + thetad * (WTOW_LENGTH + WTO_WIDTH)) *
    backLeftSpeed = saturate((1 / WHEEL_RADIUS) * (msg.linear.x - msg.linear.y + thetad * (WTOW_LENGTH + WTO_WIDTH)) *
    backRightSpeed = saturate((1 / WHEEL_RADIUS) * (msg.linear.x + msg.linear.y - thetad * (WTOW_LENGTH + WTO_WIDTH)) *

    // usefull for debugging purpose
    // cout << "FL motor speed :" << frontLeftSpeed << " FR motor speed :" << frontRightSpeed << endl;
    // cout << "BL motor speed :" << backLeftSpeed << " BR motor speed :" << backRightSpeed << endl;

    //Send computed speeds to the front driver
    frontDriver.SetCommand(_GO, 1, frontRightSpeed);
    frontDriver.SetCommand(_GO, 2, frontLeftSpeed);

    //Send computed speeds to the back driver
    backDriver.SetCommand(_GO, 1, backRightSpeed);
    backDriver.SetCommand(_GO, 2, backLeftSpeed);

}
```

setCommands est la fonction la plus importante, en entrée elle prend un message twist et calcule la vitesse de rotation à appliquer à chaque roue pour que le robot aille à la bonne vitesse dans la bonne direction. Les équations utilisées prennent compte du diamètre des roues et aussi des dimensions du robot.

Ce programme publie aussi les données des encodeurs en faisant la différence en rotation depuis la dernière mesure de valeurs.

#### 3.5.3.3.1.2.2 Odométrie

##### 3.5.3.3.1.2.1 Estimation de l'odométrie

L'estimation de l'odométrie du robot se fait par le relevé des encodeurs présents sur chaque moteur, puis par les étapes de calculs suivantes :

Rotation effectuée par chaque roue en tr/s.

Vitesse moyenne de translation et de rotation du robot sur le temps écoulé depuis la dernière mesure.

Déplacement du robot sur cette même période de temps.

(Voir programme *odom.cpp* dans le package *ROS src/nodes/MotorControl/odom.cpp*)

### 3.5.3.3.1.2.2 Précision de l'odométrie et résolution des problèmes rencontrés

Nous avons remarqué sur rviz que notre odométrie subissait des "sauts" dans l'espace, impossibles physiquement pour le robot. C'est pourquoi l'ajout de filtres supprimant les valeurs erronées à chaque étape précédemment citée a été nécessaire.

L'expérimentation ayant montré que l'erreur était proportionnelle nous avons concentré nos recherches sur les caractéristiques précises des roues Mécanum. Nous avons trouvé cette publication décrivant le calcul du rayon de la roue en fonction du mouvement du robot. Ce papier (<https://www.researchgate.net/publication/326283867> Influence of mecanum wheels construction on accuracy of the omnidirectional platform navigation on example of KUKA youBot robot) propose une nouvelle implémentation du modèle mathématique d'un robot Mécanum avec des équations revues pour améliorer l'odométrie.

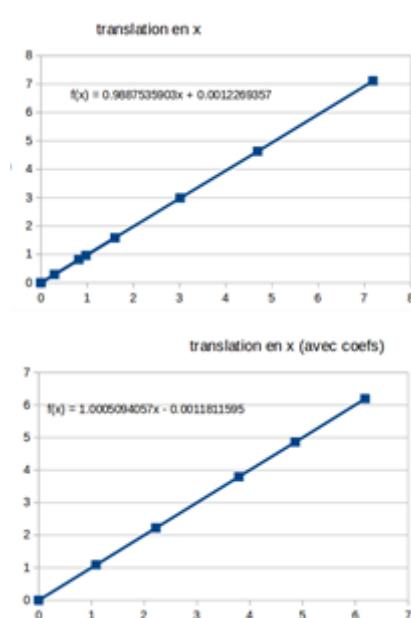


Figure 78 Graphiques : Relation entre distance réelle parcourue et distance estimée parcourue

Pour vérifier le bon fonctionnement de cet ajout nous avons réitéré l'expérience après modification. On remarque sur le second graphique que le coefficient directeur se rapproche fortement de 1 ce qui valide notre modèle. De même pour les translations selon y.

L'ajout de ces coefficients nous a permis de réduire l'erreur de localisation.

Ce premier graphique représente la relation entre la distance réelle parcourue par le robot et la distance estimée parcourue par le robot pour une translation en x (avant/arrière). On obtient une droite dont le coefficient directeur est le coefficient à appliquer aux équations d'estimation de l'odométrie. La même méthode est appliquée aux translations selon y (droite/gauche).

On obtient donc maintenant une précision de l'estimation de l'odométrie de 0,50941 mm en x et 3,732021 mm en y.

Les relevés de toutes ces données et les graphiques ci-dessus sont disponibles dans le classeur `mecanum_coefs.ods` ou en annexe (ATTENTION NE PAS OUBLIER D'AJOUTER LE DOC EN ANNEXE).

### 3.5.3.3.1.2.3 Odométrie et programmation

Voir <http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom>

Deux nodes sont utilisés pour l'odométrie. Le premier est évidemment drive.cpp pour permettre au robot de se mouvoir, il recueille aussi le retour des encodeurs et rends ces informations disponibles sur le topic encs\_sensors. Le second est odom.cpp qui lit ces données et estime l'odométrie du robot comme précédemment expliqué et publie sa transformée expliquée ci-après.

Nous allons décrire le fonctionnement du node odom.cpp qui prend en entrée les données sortantes de drive dans la partie précédente.

```
void callback(const heron::Encoders& data)
{
    current_time = ros::Time::now();
    delta_poses.dt = (current_time - last_time).toSec();

    // calculate the rotation made by each wheel
    diff_encs.Fl = -(data.EncFl / ENCODERS_COUNTABLE_EVENTS_OUTPUT_SHAFT) / delta_poses.dt;      // tr/s
    diff_encs.Fr = -(data.EncFr / ENCODERS_COUNTABLE_EVENTS_OUTPUT_SHAFT) / delta_poses.dt;
    diff_encs.Bl = -(data.EncBl / ENCODERS_COUNTABLE_EVENTS_OUTPUT_SHAFT) / delta_poses.dt;
    diff_encs.Br = -(data.EncBr / ENCODERS_COUNTABLE_EVENTS_OUTPUT_SHAFT) / delta_poses.dt;

    speed.vx = (1 / MECANUM_X) * (2*M_PI*WHEEL_RADIUS) * (diff_encs.Fl + diff_encs.Fr + diff_encs.Bl + diff_encs.Br)/4;      // m/s
    speed.vy = (1 / MECANUM_Y) * (2*M_PI*WHEEL_RADIUS) * (- diff_encs.Fl + diff_encs.Fr - diff_encs.Bl + diff_encs.Br)/4;      // m/s
    speed.vth = - 2*M_PI*WHEEL_RADIUS * (+ diff_encs.Fl - diff_encs.Fr - diff_encs.Bl + diff_encs.Br) / (4*(WTOW_LENGTH + WTO_WIDTH));
```

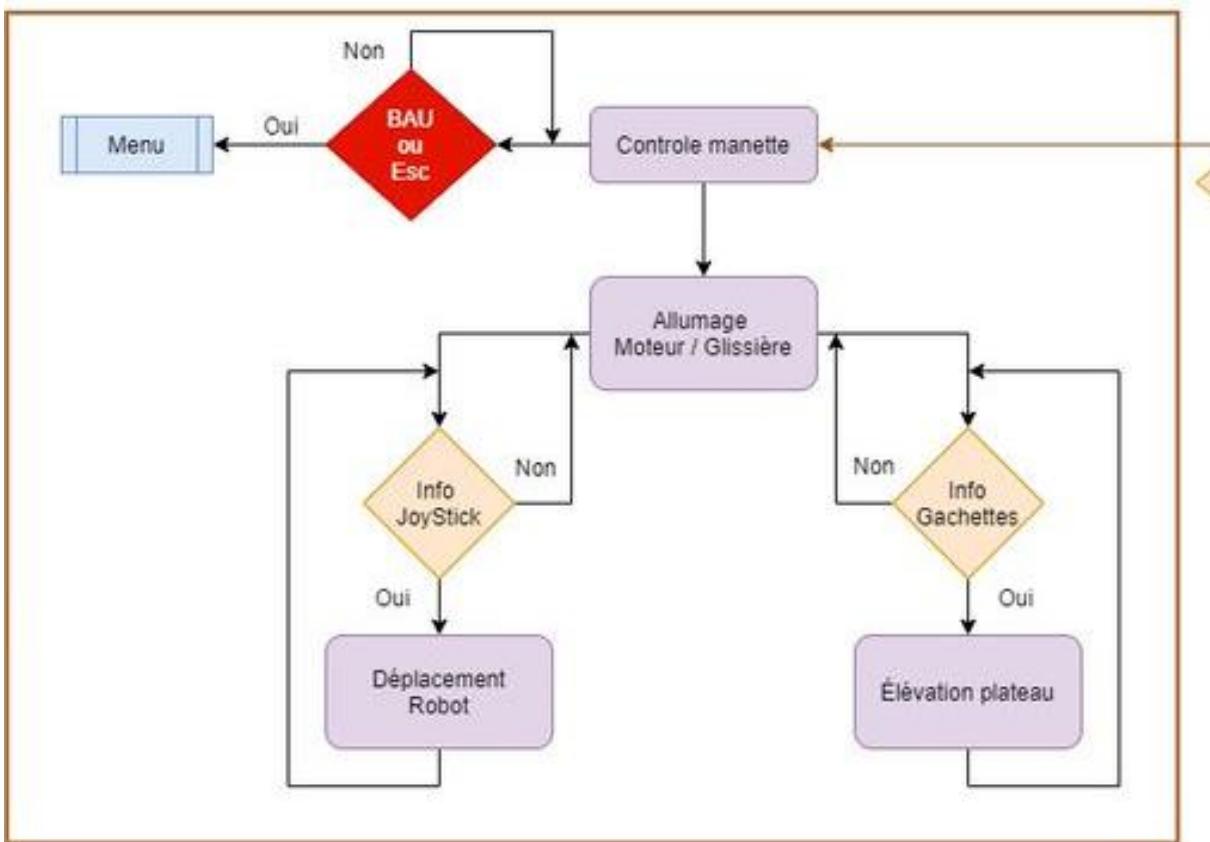
Ici on a le calcul de la rotation effectuée par chaque roue depuis la dernière mesure de valeurs. Puis en utilisant le modèle du robot mais inverse cette fois on calcule la vitesse moyenne du robot sur cette même période.

Pour finir on filtre les valeurs erronées, on calcule la distance parcourue par le robot en roulant à sa vitesse moyenne et on publie la transformée entre le repère du robot et le repère du sol.

### 3.5.3.3.1.2.4 Odométrie et TF

Une fois le calcul de l'odométrie fait, nous voulons le traduire en translations et rotations réelles du robot. Sur ROS cela se fait avec les tf (Voir <http://wiki.ros.org/tf>). En ce qui concerne l'odométrie, le lien fait entre les repères est base\_link (le centre du robot) et odom (le repère fixé à la position d'initialisation du robot).

NB : Cette dernière partie est fortement inspirée du tutoriel d'odométrie de ROS. Voir <http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom>



Les programmes utilisés pour ce mode « Remote control » :

**Mode : Remote Control**

roscore

heronController.launch *Contrôle du robot à la manette.*

Winch.py *Traduit les données de la manette en consignes de vitesse pour le plateau.*

Drive *Créé la liaison avec les drivers de moteurs et leur envoie des consignes de vitesses pour chacun de leurs moteurs.*

xboxController.launch

Joy *Rend les données de la manette disponibles.*

Controller.py *Traduit les données de la manette en consignes de vitesse pour le robot.*

### 3.5.3.3.2 Mode « Mapping » :

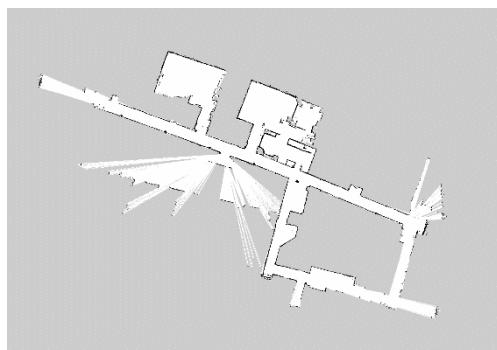
Objectif de ce mode : Permettre à un utilisateur de réaliser une cartographie de son environnement et donc créer une carte en 2D de celui-ci.

Ce mode est à réaliser (obligatoire !) pour que le robot puisse naviguer de manière autonome dans ce même environnement.

Pour activer ce mode, nous avons ici besoin de lancer des programmes sur le robot Héron mais également sur le ROS Master. Nous allons donc interagir sur les 2 interfaces graphiques.

Pour générer une carte de l'environnement nous avons recourt à un mappeur qui par lecture des capteurs sur le robot enregistre la présence d'obstacles. Ensuite un server qui rendra cette carte disponible (map\_server)

Pour le mapping nous utilisons un algorithme dont l'implémentation est déjà existante sur ROS et qui s'appelle gmapping.



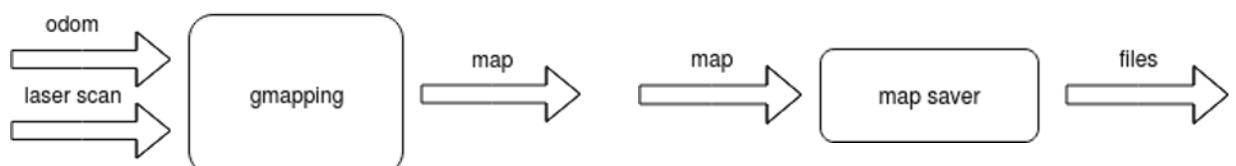
*Map :*  
*Résolution en pixels*  
*Noir = obstacle*  
*Gris = inconnu*  
*Blanc = libre*

Figure 79 Carte de l'étage de robotique

Voir <http://wiki.ros.org/gmapping>

Gmapping est un algorithme permettant de cartographier l'environnement du robot. Les prérequis de l'utilisation de cet algorithme sont d'avoir un robot de forme ronde, carrée ou rectangulaire, d'avoir une source d'**odométrie** précise et d'avoir un **laser** monté quelque part sur le robot.

Le mapping consiste à lancer le programme de Gmapping et de faire naviguer le robot dans son environnement afin qu'il le découvre à l'aide de son lidar. Ensuite map\_saver, un autre programme se charge de sauvegarder sous forme de fichier ces informations.



### 3.5.3.3.2.1 Interface - Héron

Choisir le mode mapping sur le robot Héron va nous permettre d'activer les mêmes actionneurs que lors de l'activation du mode « Remote control » mais également d'utiliser le lidar ainsi que l'algorithme de mapping permettant de construire notre carte en 2D.

```

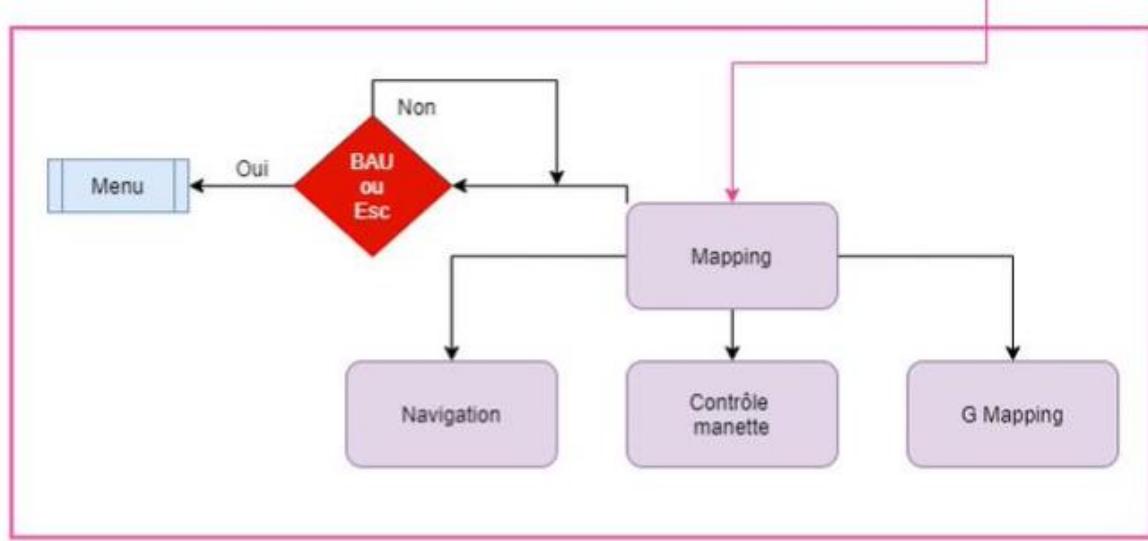
Terminal
MENU HERON

Use arrows to navigate and then ENTER :(ESC to leave mode)

      Remote Control
      Navigation
      Mapping
      Take Key Positions
      IP Config (Restart after)
      Exit

```

3.5.3.3.2.2 Programmes et organigramme logiciel - Héron



Lors de l'activation de ce mode, nous allons appeler différents programmes :

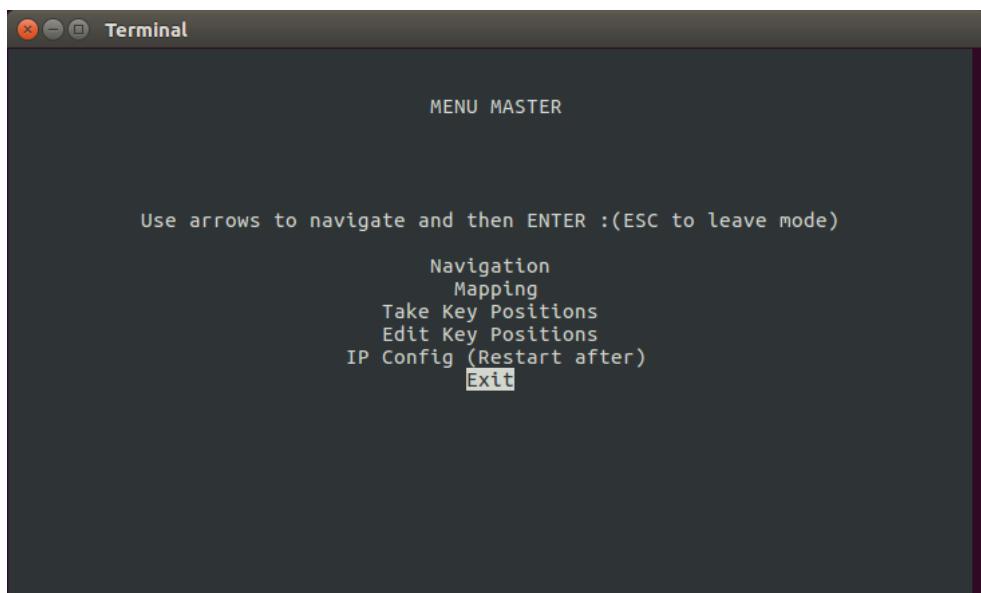
**Mode : Mapping**

gmapping.launch

gmapping\_thing

take\_pos.launch

### 3.5.3.3.2.3 Interface – PC Central



Choisir le mode mapping sur le PC Central (en parallèle du robot Héron) va nous permettre d'ouvrir l'interface graphique RVIZ dans laquelle nous pourrons observer le comportement de notre robot grâce à un digital twin (voir parties suivantes). Grâce à cela nous avons un retour visuel sur les données renvoyées par le lidar, sur la position du robot mais surtout un retour en temps réel de la création de la carte 2D.

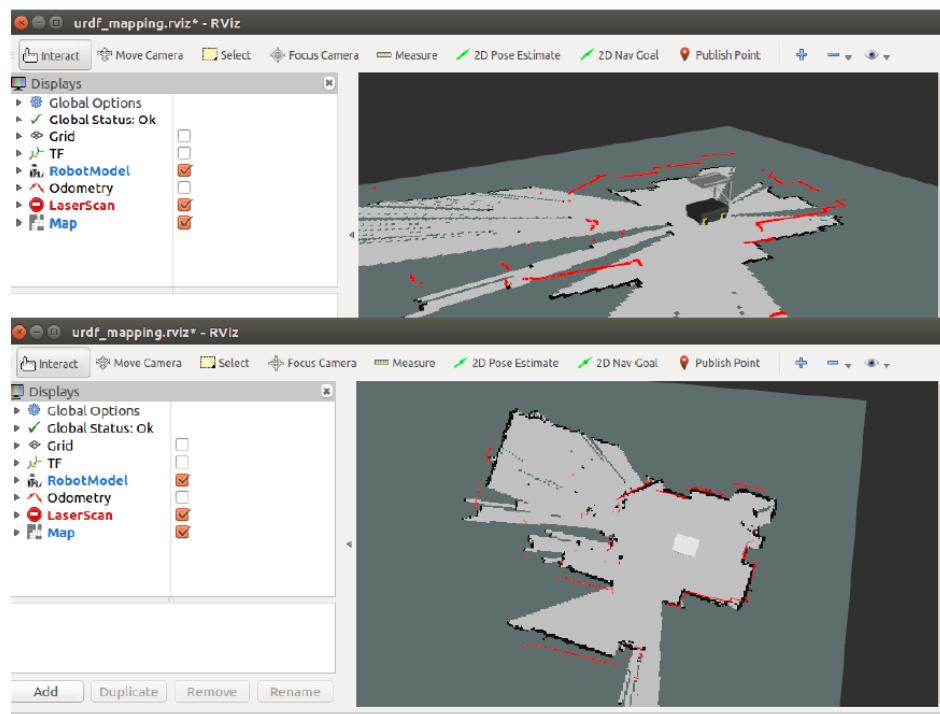
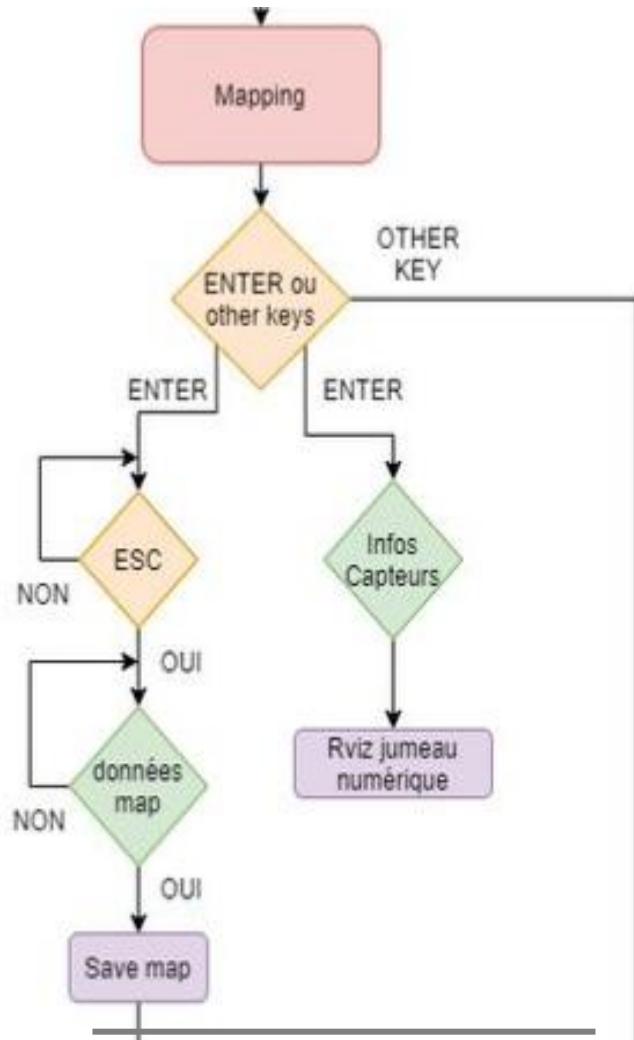


Figure 80 Représentation de la carte en construction sur RVIZ



Lors de l'activation de ce mode nous allons appeler plusieurs programmes :

#### Mode : Mapping

```

display_mapping.launch
joint_state_publisher
robot_state_publisher
tf_robot
rviz
  
```

Une fois le mode « Mapping » lancé sur le robot Héron ainsi que sur le PC Central, vous n'avez plus qu'à saisir la manette de XBOX et faire naviguer le robot dans l'environnement que vous souhaitez cartographier.

Une fois le processus de mapping terminé et après avoir suivi le manuel d'utilisation expliquant les manipulations à réaliser pour la partie « Mapping », nous aurons alors générée une carte 2D. Cette carte de l'environnement, nécessaire pour la future navigation autonome du robot, sera alors enregistrée au format « map.png » (représentant la carte du point de vue visuel) et « map.yaml » (regroupe des informations sur la carte en elle-même telle que la taille de celle-ci etc. ...). Cet enregistrement se réalise automatiquement lors de la fermeture du mode lancé sur le PC Central. Cet enregistrement est géré par le package ROS « map\_saver ».

#### 3.5.3.3.3 Mode « Take Key Position »

Objectif de ce mode : Maintenant que la carte de l'environnement a pu être générée, il faut à présent « faire apprendre au robot Héron » des positions bien précises qu'il pourra atteindre lors de sa navigation autonome.

Auparavant cette étape était quelque peu complexe et nécessitait de multiples actions de la part de l'utilisateur, mais nous avons développé certains outils de manière à faciliter tout cela. Cette étape est toute aussi importante que la précédente, car même si nous avons créé une map au préalable, nous devons à présent faire apprendre des positions au robot pour qu'il puisse les atteindre dans la navigation autonome.

Pour réaliser cette étape d'apprentissage nous allons utiliser le package ROS « AMCL » (Adaptive Monte Carlo Localization)

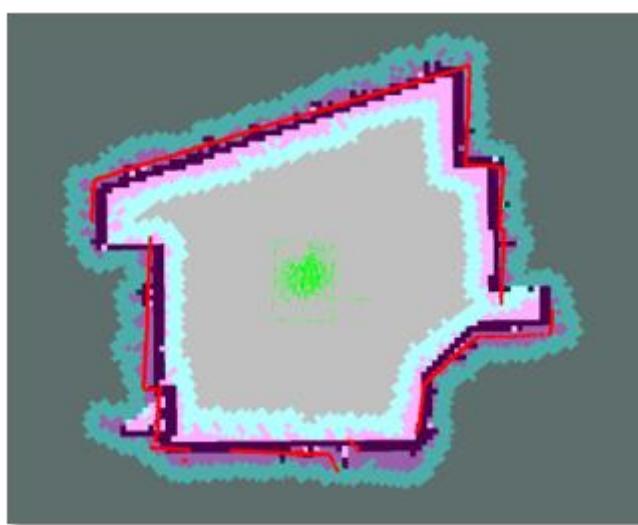


Figure 81 exemple utilisation AMCL

Voir <http://wiki.ros.org/amcl>

Cet algorithme consiste, à partir d'une carte de l'environnement comme expliquée précédemment, à estimer la position et l'orientation du robot au fur et à mesure qu'il se déplace et capte son environnement. L'algorithme utilise un filtre à particule pour représenter la distribution des états possibles du robot avec chaque particule correspondant à un état. Ces particules sont représentées par les flèches vertes sur la Figure 82 exemple utilisation AMCL. L'algorithme démarre avec une distribution aléatoire uniforme des particules. Quand le robot bouge, ces particules se rassemblent petit à petit vers l'emplacement où la probabilité de présence et d'orientation du robot est la plus forte. Le point de convergence des particules donne alors la position relative du robot.

probabilité de présence et d'orientation du robot est la plus forte. Le point de convergence des particules donne alors la position relative du robot.

Le but de cette étape étant de positionner grâce à la manette le robot aux positions qui nous intéressent, presser un bouton sur la manette XBOX afin d'enregistrer automatiquement la pose du robot (x,y,theta,h) où x et y correspondent aux coordonnées du robot dans sa carte, théta correspond à son orientation et h correspond à la hauteur à laquelle la glissière se trouvée lors de l'appui du bouton. Ces positions seront automatiquement inscrites dans un fichier texte qui sera réexploité par la suite.

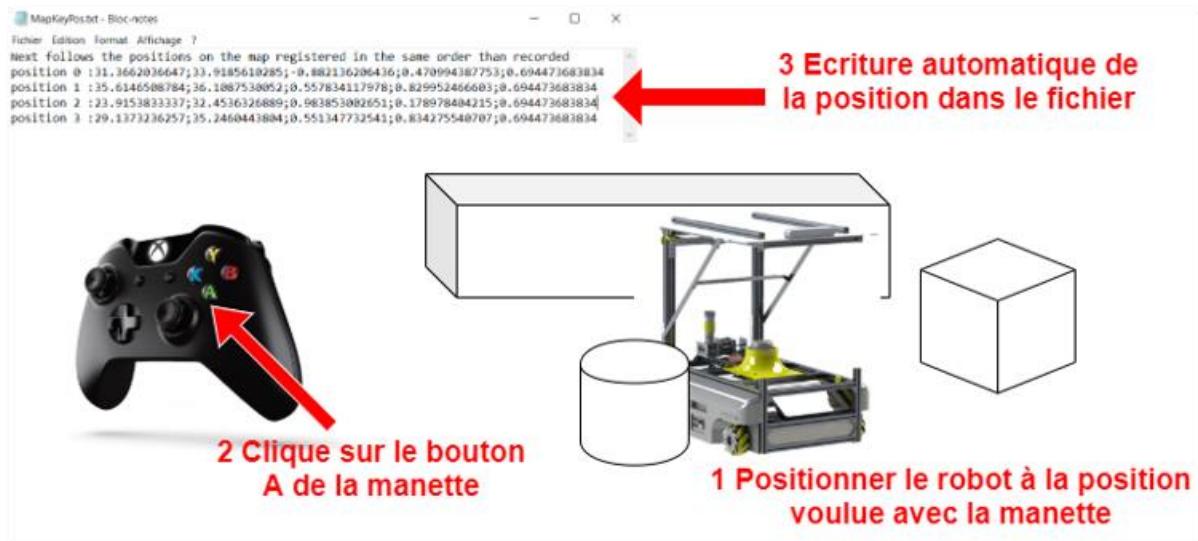
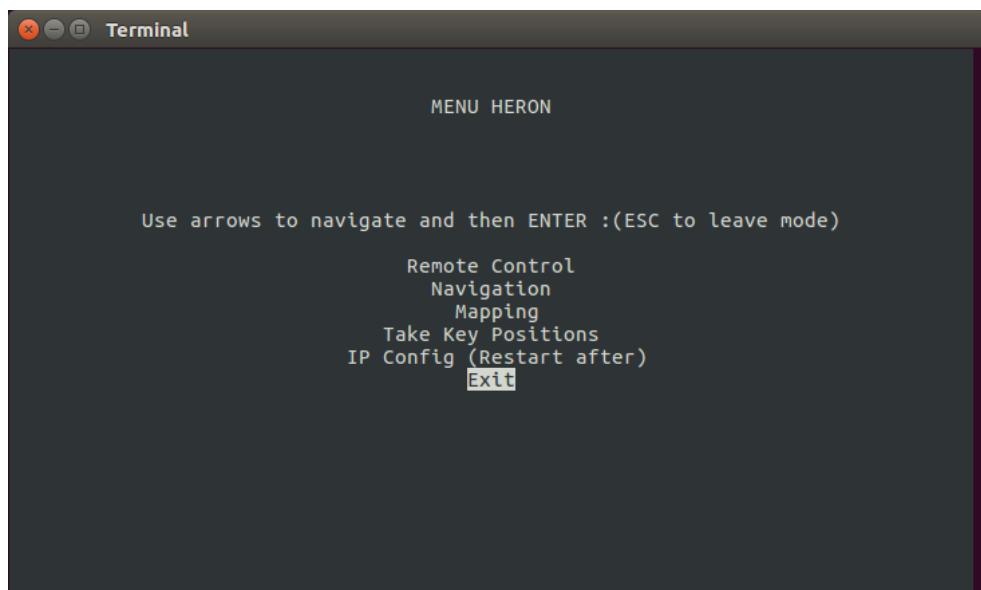
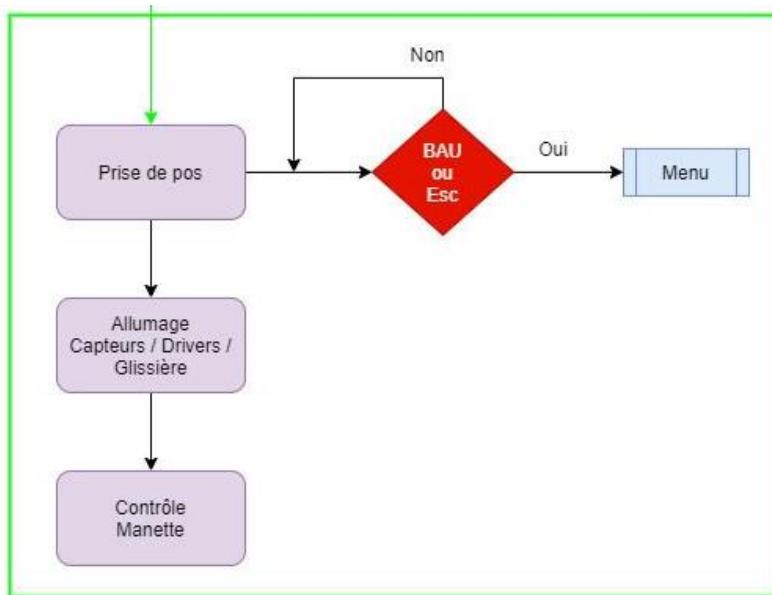


Figure 82 Illustration de la manipulation pour l'enregistrement des positions clés.

#### 3.5.3.3.3.1 Interface – Héron





Lors de l'activation de ce mode nous allons appeler plusieurs programmes :

#### Mode : Take Key Positions

`take_pos.launch`

*Drive* Crée la liaison avec les drivers de moteurs et leur envoie des consignes de vitesses pour chacun de leurs moteurs.

*Odom* Estime la pose du robot avec les données des encodeurs reçues par drive.

*Winch* Traduit les données de la manette en consignes de vitesse pour le plateau.

*lpmis\_imu* Rend les données de l'IMU disponibles.

`lidar.launch`

*RplidarNode* Rend les données du lidar disponibles.

*laser\_filter* Filtre les objets internes au robot (support du plateau et son moteur).

`tf_broadcaster`

`robot_state_publisher`

`xboxController.launch`

*Joy* Rend les données de la manette disponibles.

**Controller.py** Traduit les données de la manette en consignes de vitesse pour le robot.

#### 3.5.3.3.3.3 Interface – PC Central

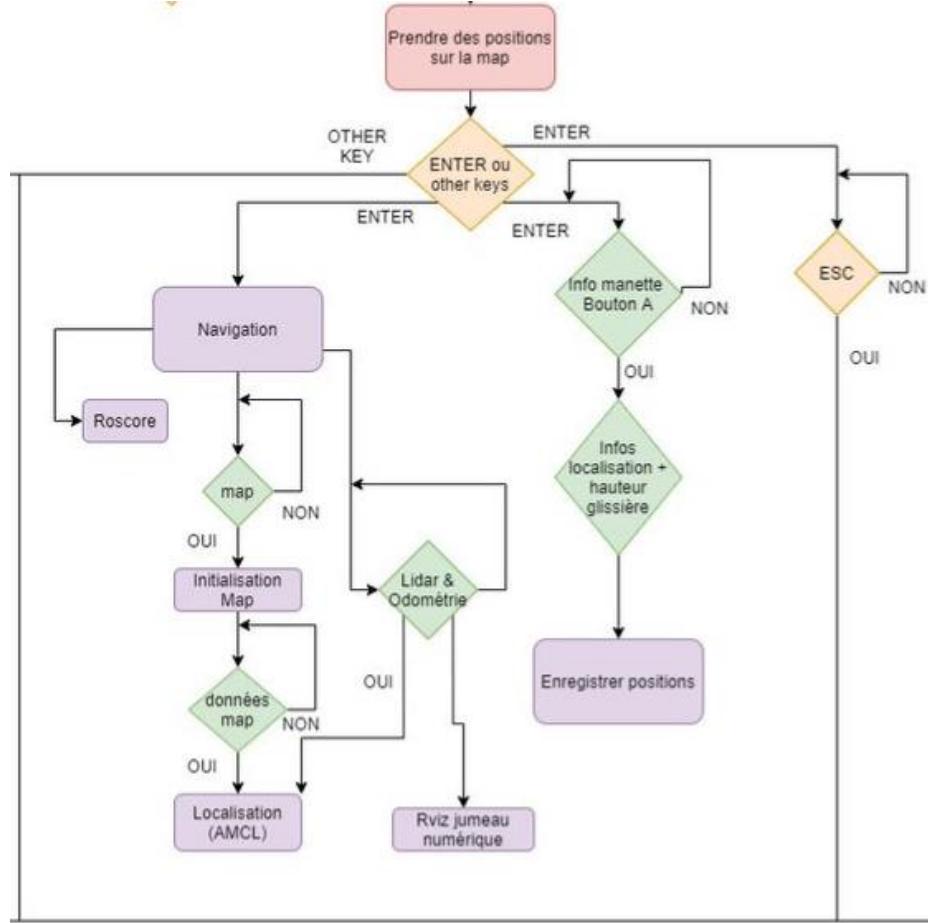
```

Terminal
MENU MASTER

Use arrows to navigate and then ENTER :(ESC to leave mode)

Navigation
Mapping
Take Key Positions
Edit Key Positions
IP Config (Restart after)
Exit
  
```

#### 3.5.3.3.3.4 Programmes et organigramme logiciel – PC Central



Lors de l'activation de ce mode nous allons appeler plusieurs programmes :

#### Mode : Take Key Positions

`take_key_position.launch` Enregistrement de positions clés : pose et de hauteur du plateau. Pour pouvoir y envoyer le robot dans la phase de navigation.

`map_server` Fournit la map de l'environnement sous une forme exploitable pour les algo.

`recordPositions.py` Permet un enregistrement de positions clés en appuyant sur le bouton A de la manette. Cette action à pour effet d'enregistrer la pose du robot et la hauteur du plateau.

`display_navigation.launch`

`joint_state_publisher`

`robot_state_publisher`

`tf_robot`

`rviz`

`amcl.launch`

`amcl + params` Localisation du robot dans son environnement par superposition des données du lidar et de la map à sa position la plus probable.

Après avoir enregistré toutes vos positions souhaitées en ayant suivi le manuel d'utilisation dans la partie « Enregistrer les positions », vous allez à présent pouvoir passer à la prochaine étape.

Node `recordPosition.py` qui scrute l'état du bouton A de la manette de xBox, il lit la position estimée du robot et la hauteur du plateau. Pour déclencher l'enregistrement de ces informations à la pression du bouton. Pour cela le node souscrit au topics `joy`, `amcl_pose` et `winchHeight`. Les données sont écrites dans un fichier text `MapKeyPos.txt` dans home sur le PC centrale.

Description programme `recordPositions.py`

```

34  if __name__ == '__main__':
35      file = open("../MapKeyPos.txt", "w")
36      file.write("Next follows the positions on the map registered in the same order than recorded \n")
37      file.close()
38      # starts the node
39      rospy.init_node('remote')
40
41      # subscribed to joystick inputs on topic "joy"
42      rospy.Subscriber("joy", Joy, callback)
43
44      rospy.Subscriber("amcl_pose", PoseWithCovarianceStamped, getPos)
45
46      rospy.Subscriber("winch_Height", MsgWinch, getHeight)
47
48      rospy.spin()

```

Le programme ouvre le fichier MapKeyPos.txt où les positions clés sont stockées. Il souscrit à la manette, à l'estimation de pose de l'amcl et à la hauteur du plateau.

```

12  def callback(data):
13      global index
14      global tmp_pos
15      # left menu button
16      if(data.buttons[0]):
17          if(tmp_pos != positionMsg):
18              file = open("../MapKeyPos.txt", "a")
19              file.write("position " + str(index) + " ")
20              file.write(positionMsg)
21              print("writing Position" + str(index) + " into the file.")
22              file.close()
23              index += 1
24          tmp_pos = positionMsg

```

A chaque information reçue de la manette le programme exécute cette fonction callback. Elle scrute le bouton A, et s'il est pressé il ouvre le fichier, écrit dans le fichier la position du robot et la hauteur du plateau et ferme le fichier.

```

26  def getPos(data):
27      global positionMsg
28      positionMsg = str(data.pose.pose.position.x) + ";" + str(data.pose.pose.position.y) + ";" + str(data.pose.p

```

A chaque publication de position de l'amcl on sauvegarde la position.

```

30  def getHeight(data):
31      global heightMsg
32      heightMsg = str(data.height)

```

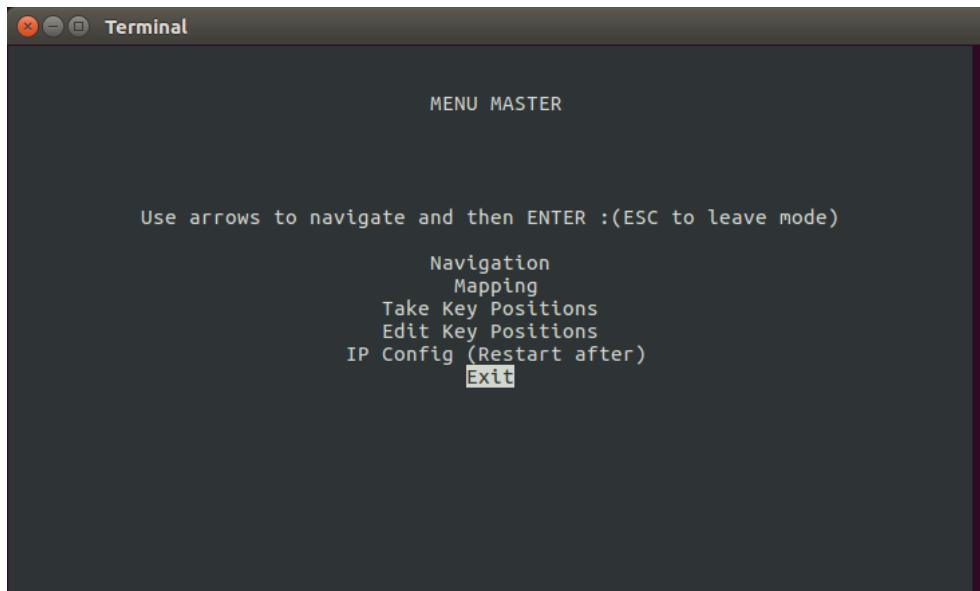
A chaque publication de hauteur du plateau on sauvegarde la hauteur.

#### 3.5.3.3.4 Mode « Edit Key Positions » :

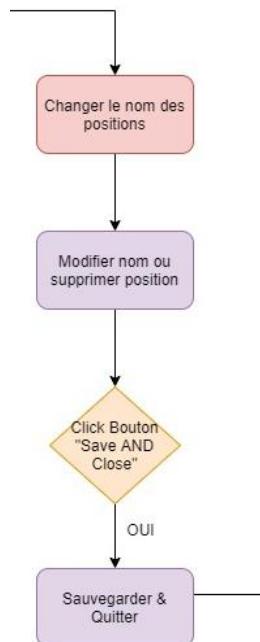
Objectif de ce mode : Une fois l'étape précédente terminée, vous avez pu enregistrer des positions bien précises grâce à la manette, cependant vous devez à présent rendre ces étapes exploitables par le ROS Master pour la future navigation autonome.

Vous allez donc pouvoir les rendre utilisables par le biais de quelques manipulations via une interface graphique développée par nos propres soins pour vous faciliter la tâche. Cette interface graphique n'est à ouvrir que sur le PC Central.

##### 3.5.3.3.4.1 Interface – PC Central



##### 3.5.3.3.4.2 Programmes et organigramme logiciel – PC Central



Lors de l'activation de ce mode nous appelons un seul programme :

### Mode : Edit Key Positions

#### Interface modifPositions.py

Cette interface permet de modifier le nom des positions clés enregistrées et d'en supprimer si voulu.

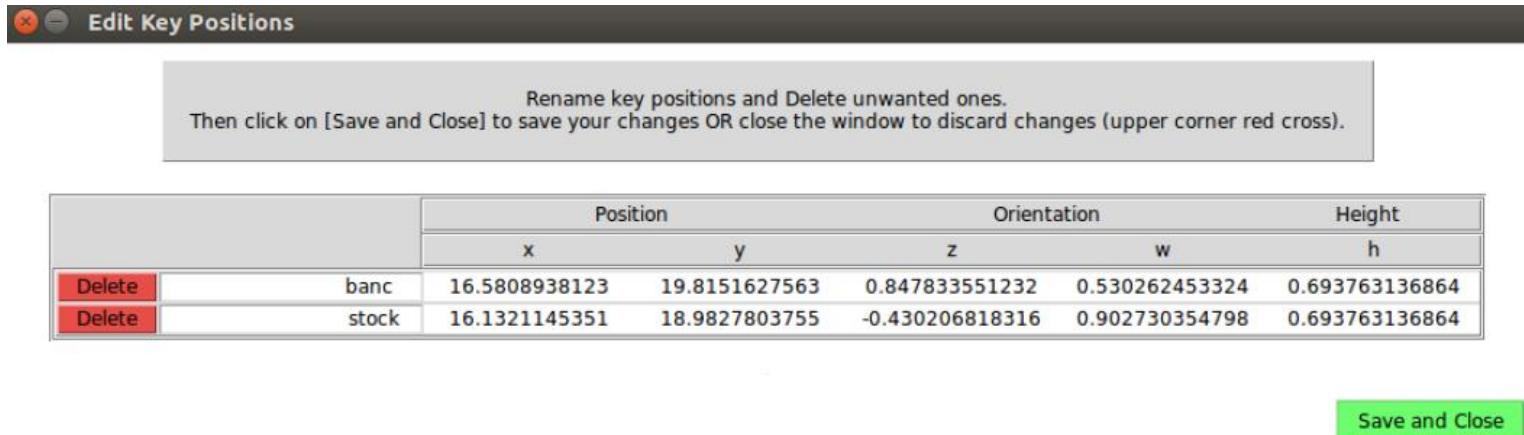


Figure 83 Interface graphique permettant d'éditer les positions pré-enregistrées

Vous pourrez retrouver dans cette interface graphique toutes les positions que vous avez pu enregistrer lors de l'étape précédente.

Celle-ci sont affichées dans un ordre chronologique, la 1<sup>ère</sup> position affichée sur l'interface graphique (1<sup>ère</sup> ligne) correspond à la première position que vous avez enregistré à l'aide de votre manette.

L'objectif de cette interface est de venir renommer les positions que vous avez enregistré. Dans l'exemple, nous avons renommé nos positions « banc » et « stock ».

Cette étape de renommée est importante également, car lorsque vous souhaiterai automatiser les déplacements de votre robot dans votre environnement, vous ne demanderez plus au robot de « se déplacer en [x=16,5 ; y=19,8 ; z = 0,84 ; w = 0,5 ; h=0 ,69] » mais vous demanderez au robot « déplace toi au « banc » ».

Cela vous facilitera la tâche dans vos prochaines étapes de navigation autonome, veillez donc à choisir des noms clairs, courts et décrivant clairement la position à atteindre.

L'interface graphique vous permet donc de :

- \_ supprimer des positions que vous avez pu enregistrer par maladresse grâce au bouton « Delete »
- \_ modifier le nom des positions que vous avez pu enregistrer
- \_ sauvegarder vos modifications grâce au bouton « Save and Close »

### 3.5.3.3.5 Mode « Navigation » :

Objectif de ce mode : Faire naviguer le robot en mode navigation autonome, en lui faisant par exemple atteindre les positions au préalable via une simple interface graphique.

Pour réaliser cette partie de navigation, nous allons à présent utiliser la stack de navigation de ROS.

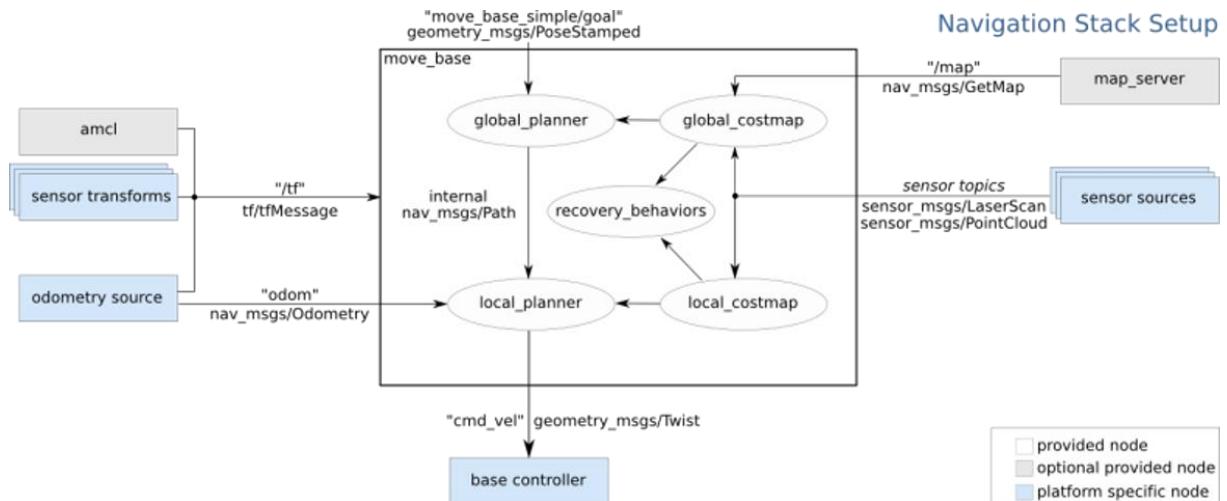


Figure 84 Schéma résumant la stack de navigation ROS

Ce schéma résume parfaitement comment se déroule la navigation autonome de notre robot, à une exception près que nous allons expliquer.

Si l'on se concentre sur le cœur du schéma, le robot utilise bel et bien le package « move\_base » pour générer le tracé que le robot doit suivre pour atteindre un goal.

Le global planner correspond à « NavfnRos ». Ce planner est le planner par défaut proposé par ROS. Nous ne l'avons pas changé car les services offerts par ce planner correspondaient avec nos attentes. Le global planner a pour mission de générer le tracé en se basant sur la map générale que l'on a créée, aussi appelée « global\_costmap ».

La « global\_costmap » représente la map que nous avons construite au préalable, sur laquelle nous avons ajouté des limites que le robot ne peut pas atteindre. Par exemple, nous avons configuré notre global costmap de manière à empêcher le robot de se rapprocher à moins de 10cm d'un mur représenté sur cette map.

Le local planner lui va venir générer également le trajet mais en se basant sur la « local\_costmap ». La « local\_costmap » est générée à partir des données fournies par le lidar. Celle-ci permet de prendre en compte les obstacles imprévus et qui n'ont pas été enregistrés lors de la phase de mapping. Le local planner choisi pour notre stack de navigation est le DWA planner.

Cet algorithme consiste à :

- Effectuer un échantillonnage discret des trajectoires possibles du robot,
- Simuler le résultat pour chaque échantillon,
- Appliquer un score pour chaque trajectoire (en fonction de la vitesse possible du robot, la proximité avec les obstacles et la possibilité d'atteindre l'objectif),

Sélectionner la trajectoire possédant le meilleur score.

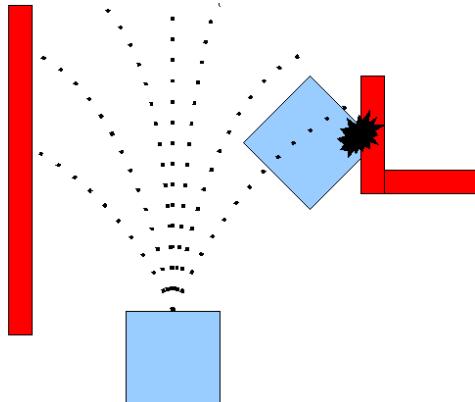
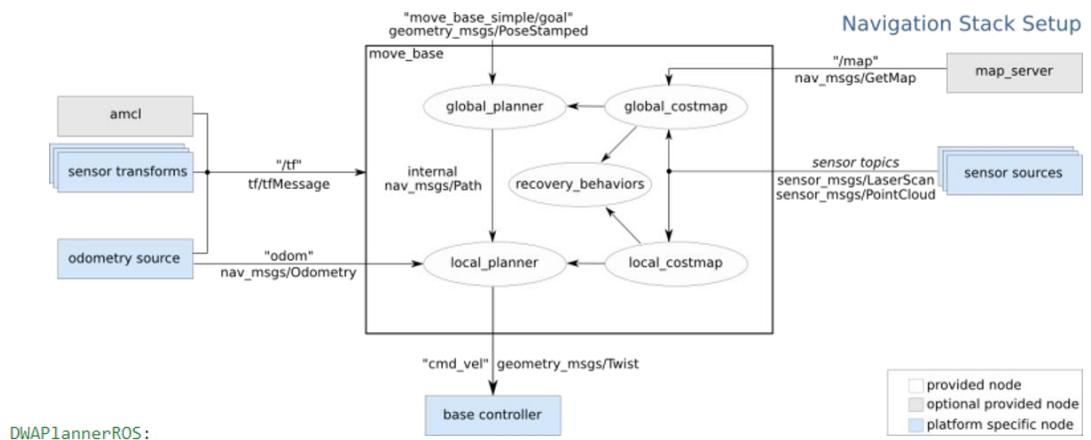


Figure 86 Illustration DWA source : ros-wiki

Attention : Le paramétrage du local planner reste à revoir afin d'améliorer l'évitement d'obstacles dynamiques.

Vous trouverez ci-dessous les différents paramétrages du local\_planner, ainsi que des local et global costmap.



DWAPlannerROS:  
max\_trans\_vel: 0.4

max\_vel\_x: 0.4  
min\_vel\_x: 0.01

max\_vel\_y: 0.4  
min\_vel\_y: 0.01

max\_rot\_vel: 1.0  
min\_rot\_vel: 0.005

acc\_lim\_th: 1.0  
acc\_lim\_x: 0.8  
acc\_lim\_y: 0.8

# goal tolerance  
xy\_goal\_tolerance: 0.05  
yaw\_goal\_tolerance: 0.01

Base\_local\_planner\_params.yaml

Dans ce fichier on spécifie les paramètres relatifs aux caractéristiques du robot, à savoir sa vitesse max et son accélération. Mais aussi la précision que l'on souhaite pour que le robot atteigne son goal.

#### costmap\_common\_params.yaml

```
obstacle_range: 10 #updates obstacles at 10m from the robot
raytrace_range: 10
footprint: [[0.3, 0.25], [-0.364, 0.25], [-0.364, -0.25], [0.3, -0.25]]
#robot_radius: ir_of_robot
inflation_radius: 0.5

observation_sources: laser_scan_sensor

laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan, topic: scan, marking: true, clearing: true}

recovery_behaviors:
- name: 'costmap_reset_conservative'
  type: 'clear_costmap_recovery/ClearCostmapRecovery'
- name: 'aggressive_reset'
  type: 'clear_costmap_recovery/ClearCostmapRecovery'
costmap_reset_conservative:
  reset_distance: 0.01
  layer_names: ["obstacle_layer"]

aggressive_reset:
  reset_distance: 0.0
  layer_names: ['obstacle_layer']
```

Ici on renseigne la distance des obstacles à prendre en compte, l'empreinte du robot (son encombrement) ainsi que le rayon d'inflation autour des obstacles (c'est la zone dans laquelle le robot ne doit pas se rendre). On fournit aussi la ou les sources des capteurs à utiliser pour créer la map et/ou se localiser.

#### global\_costmap\_params.yaml

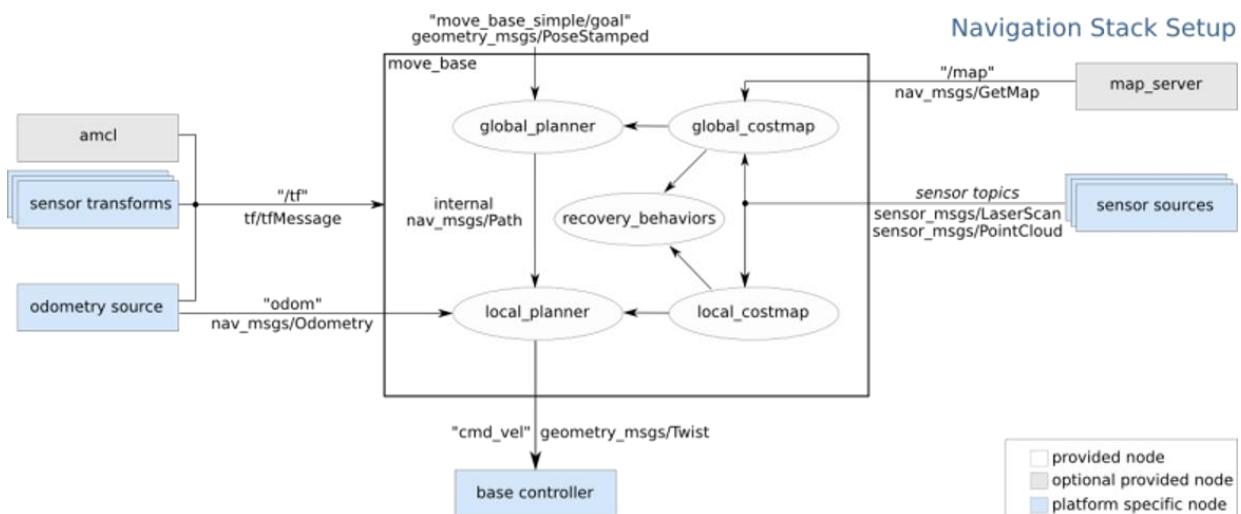
```
global_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 10.0
  static_map: true
```

Quels repères vas-t-on utiliser pour le sol (la map) et pour le robot et à quelle fréquence vas-t-on rafraîchir ce lien.

#### local\_costmap\_params.yaml

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 10
  publish_frequency: 10
  static_map: true
  rolling_window: true
  width: 2.0
  height: 2.0
  resolution: 0.05
```

Quel lien doit être utilisé pour l'odométrie et à quelle fréquence doit-on l'afficher.



Il en est de même pour les « entrées » de ce package `move_base`. Nous avons bien notre source d'odométrie ainsi que celle du lidar en entrée de ce package. L'entrée « `map_server` » est respectée lorsque l'on passe notre carte au préalablement construite en entrée de ce package. Il ne reste plus qu'à « envoyer » des goals à notre robot (« `move_base_simple/goal` » de type « `geometry_msgs/PoseStamped` ») pour pouvoir vérifier le bon fonctionnement de notre stack de navigation.

Et c'est ici que nous avons apporté une petite particularité à cette stack de navigation. Au lieu d'envoyer un goal en ligne de commande (compliqué et source d'erreur) ou alors via l'interface RVIZ, nous avons développé notre propre outil permettant d'envoyer le robot aux positions au préalable enregistrées et cela en un seul clic.

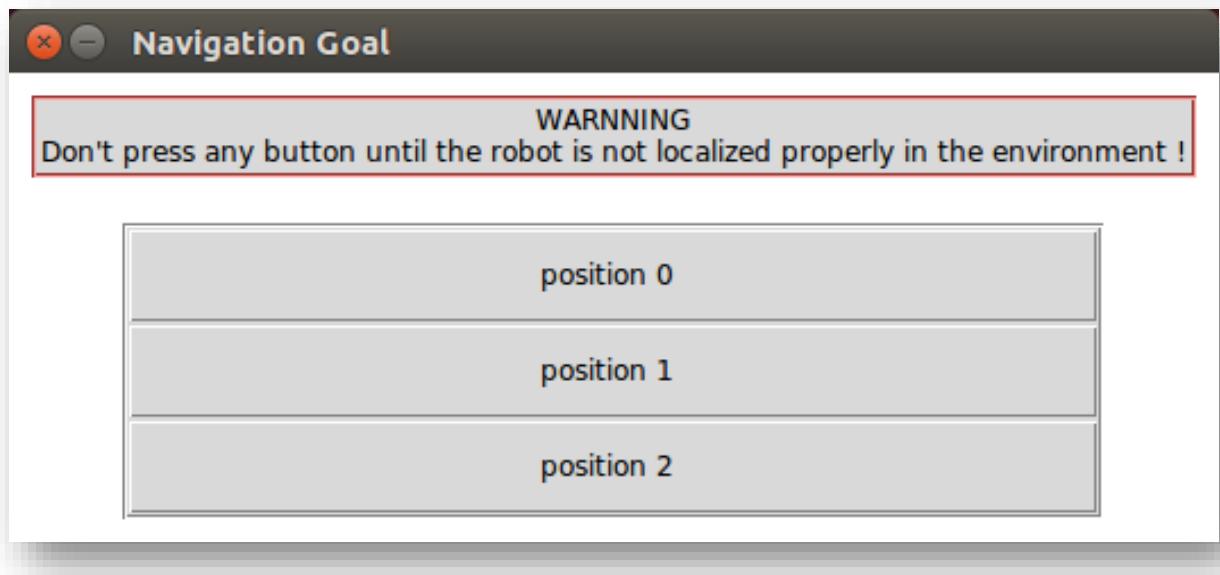


Figure 87 Interface graphique permettant d'envoyer Héron aux positions enregistrées

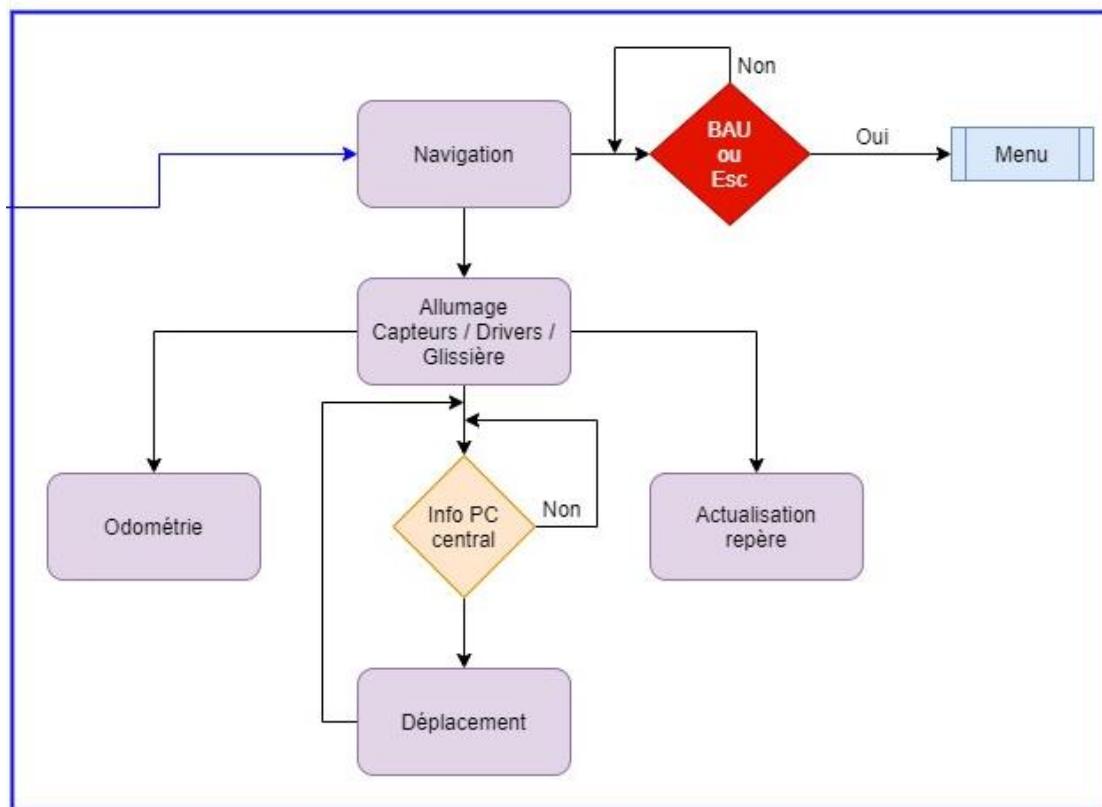
Grâce à cette interface, vous allez pouvoir faire naviguer de manière autonome votre robot aux positions précédemment enregistrées. Cela vous permet de vérifier le bon fonctionnement de la stack de navigation mais également la pertinence des positions enregistrées.

### 3.5.3.3.5.1 Interface – Héron

The screenshot shows a terminal window titled "Terminal". Inside, the text "MENU HERON" is displayed. Below it, instructions say "Use arrows to navigate and then ENTER :(ESC to leave mode)". A list of menu options follows:

- Remote Control
- Navigation
- Mapping
- Take Key Positions
- IP Config (Restart after)
- Exit

### 3.5.3.3.5.2 Programmes et organigramme logiciel – Héron



Lors de l'activation de ce mode nous allons appeler plusieurs programmes :

#### Mode : Navigation

`BringUp.launch` Allume tous les capteurs et actionneurs.

`Drive` Crée la liaison avec les drivers de moteurs et leur envoie des consignes de vitesses pour chacun de leurs moteurs.

`odom` Estime la pose du robot avec les données des encodeurs reçues par drive.

`Winch` Traduit les données de la manette en consignes de vitesse pour le plateau.

`lpms_imu` Rend les données de l'IMU disponibles.

`lidar.launch`

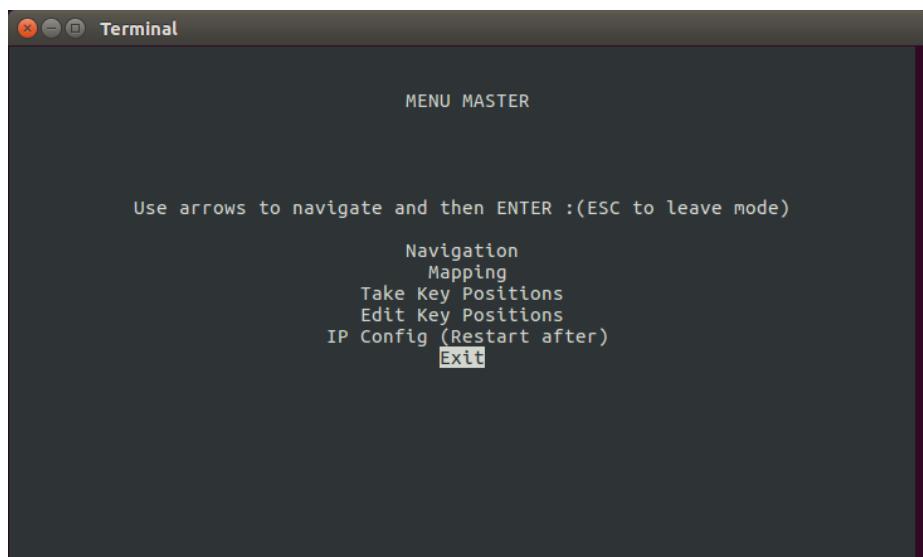
`RplidarNode` Rend les données du lidar disponibles.

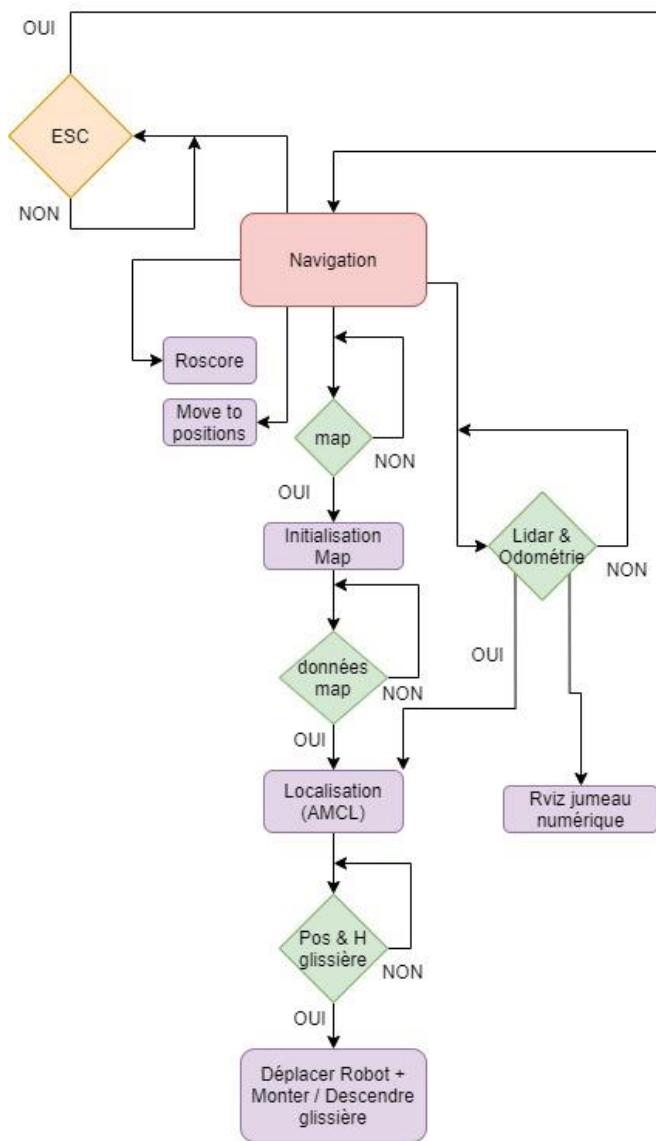
`laser_filter` Filtre les objets internes au robot (support du plateau et son moteur).

`tf_broadcaster` Publie les transformées des données de chaque capteurs.

`robot_state_publisher`

3.5.3.3.5.3 Interface – PC Central





Lors de l'activation de ce mode nous allons appeler plusieurs programmes :

**Mode : Navigation**

master\_navigation.launch

navigation.launch

one\_robot.launch

*map\_server Fournit la map de l'environnement sous une forme exploitable pour les algo.*

*simple\_navigation\_goal Action client pour fournir un goal de navigation.*

move\_base.launch

move\_base

costmap\_common\_params.yaml

local\_costmap\_params.yaml

global\_costmap\_params.yaml

base\_local\_planner\_params.yaml

amcl.launch

*amcl + params Localisation du robot dans son environnement par superposition des données du lidar et de la map à sa position la plus probable.*

display\_navigation.launch

joint\_state\_publisher

robot\_state\_publisher

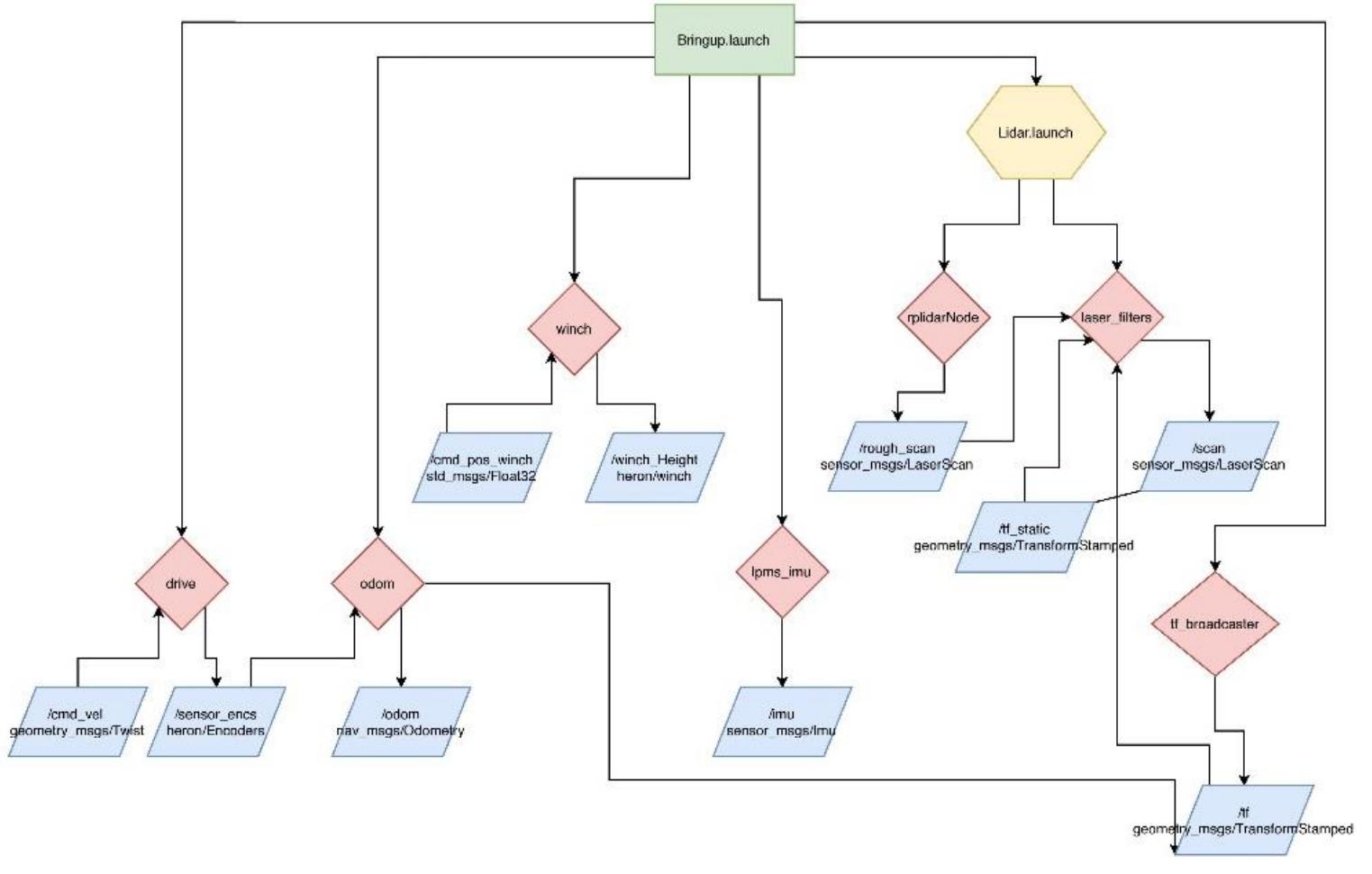
tf\_robot

rviz

*Interface navigationGoal.py Boutons pour chaque Key Position qui permettent de donner un Goal à la navigation.*

### 3.5.3.4 Détails des programmes haut niveau

#### 3.5.3.4.1 Bringup.launch



`drive` :  
Connexion physique  
drivers  
Transforme vitesse du  
robot en vitesse des  
roues  
Remonte l'information  
des tics

`odom` :  
Renvoie l'odométrie et  
s'occupe de la  
transformée par  
rapport au robot

`winch` :  
Contrôle la glissière  
Renvoie le nombre de  
tics et la hauteur du  
plateau

`lpms_imu`:  
Renvoie les données  
de l'IMU

`rplidarNode` :  
Renvoie données du  
lidar  
`laser_filters` :  
Renvoie données du  
lidar filtrées (sans le  
robot)

`tf_broadcaster` :  
Fais les transformées  
de l'IMU et le LIDAR  
par rapport au robot

#### 3.5.3.4.1.1 Drive

- `/cmd_vel`, message de type `geometry_msgs/Twist`
  - `geometry_msgs/Vector3 linear`
    - `float64 x`
    - `float64 y`
    - `float64 z`
  - `geometry_msgs/Vector3 angular`
    - `float64 x`
    - `float64 y`
    - `float64 z`

- `/sensor_encs`, message de type `heron/Encoders`
  - `int64 EncFl`
  - `int64 EncFr`
  - `int64 EncBl`
  - `int64 EncBr`

#### 3.5.3.4.1.2 Odom

- `/odom` de type `nav_msgs/Odometry`
  - `std_msgs/Header header`
  - `uint32 seq`
  - `time stamp`
  - `string frame_id`
  - `string child_frame_id`
  - `geometry_msgs/PoseWithCovariance pose`
  - `geometry_msgs/Pose pose`
  - `geometry_msgs/Point position`
    - `float64 x`
    - `float64 y`
    - `float64 z`
  - `geometry_msgs/Quaternion orientation`
    - `float64 x`
    - `float64 y`
    - `float64 z`
    - `float64 w`
    - `float64[36] covariance`
  - `geometry_msgs/TwistWithCovariance twist`
  - `geometry_msgs/Twist twist`
  - `geometry_msgs/Vector3 linear`
    - `float64 x`
    - `float64 y`
    - `float64 z`
  - `geometry_msgs/Vector3 angular`
    - `float64 x`
    - `float64 y`
    - `float64 z`
    - `float64[36] covariance`

#### 3.5.3.4.1.3 Winch

- `/cmd_pos_winch : std_msgs/Float32`
- `/cmd_vel_winch : std_msgs/Float32`
- `/winch_Height` : message de type `heron/winch` (personnalisé)
  - `float32 height`
  - `float32 heightTicks`

### 3.5.3.4.2 Lidar.launch

#### 3.5.3.4.2.1 RplidarNode

- o /rough\_scan, message de type `sensor_msgs/LaserScan` :

- o `std_msgs/Header` header
  - o `float32 angle_min`
  - o `float32 angle_max`
  - o `float32 angle_increment`
  - o `float32 time_increment`
  - o `float32 scan_time`
  - o `float32 range_min`
  - o `float32 range_max`
  - o `float32[] ranges`
  - o `float32[] intensities`

#### 3.5.3.4.2.2 Laser\_filters

- o /tf\_static, message de type `geometry_msgs/TrasformStamped` :

- o `std_msgs/Header` header
  - o `string child_frame_id`
- o `geometry_msgs/Transform` transform
- o `geometry_msgs/Vector3` translation
  - o `float64 x`
  - o `float64 y`
  - o `float64 z`
- o `geometry_msgs/Quaternion` rotation
  - o `float64 x`
  - o `float64 y`
  - o `float64 z`
  - o `float64 w`

- o /scan, message de type `sensor_msgs/LaserScan` :

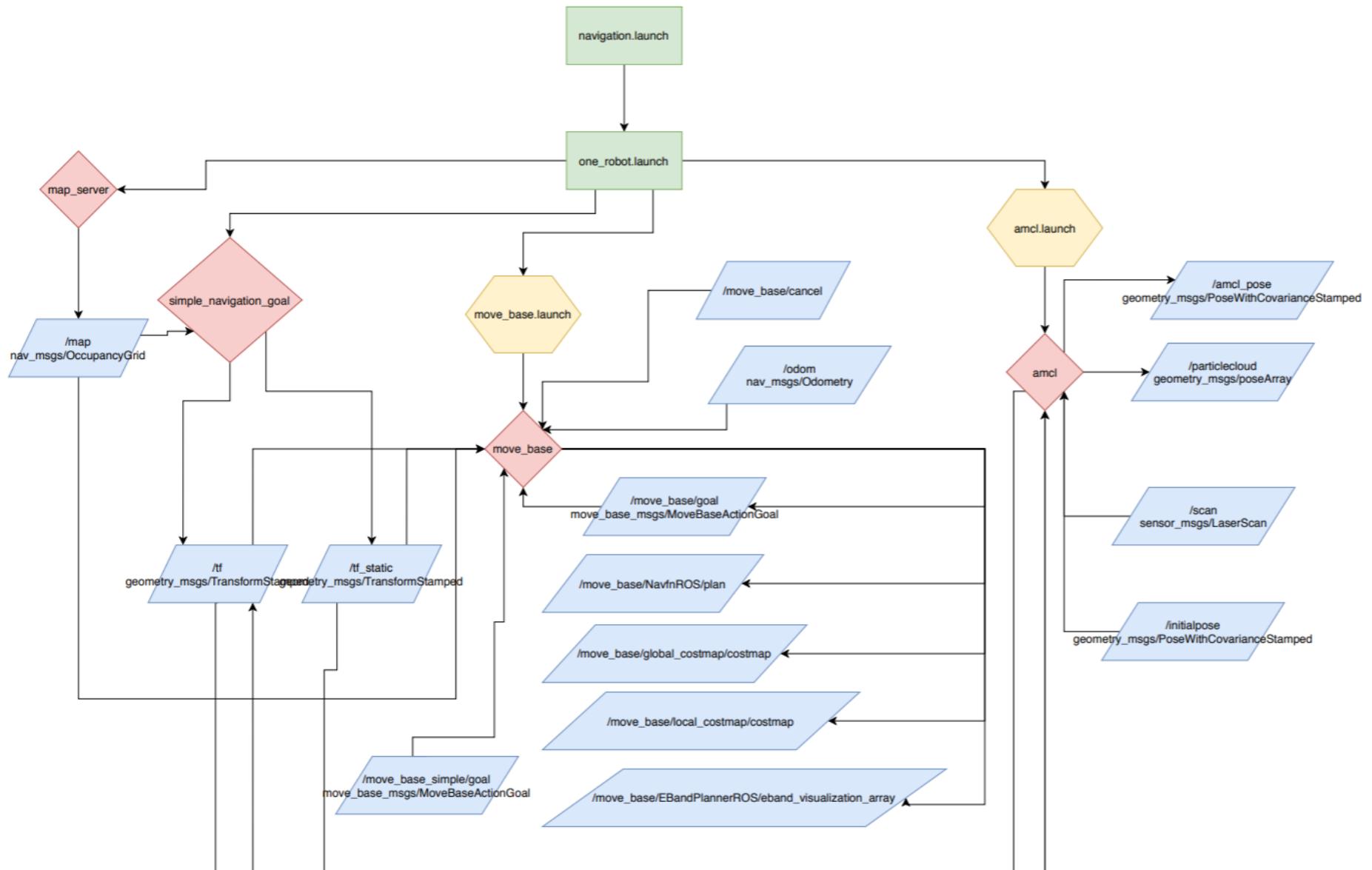
- o `std_msgs/Header` header
  - o `float32 angle_min`
  - o `float32 angle_max`
  - o `float32 angle_increment`
  - o `float32 time_increment`
  - o `float32 scan_time`
  - o `float32 range_min`
  - o `float32 range_max`
  - o `float32[] ranges`
  - o `float32[] intensities`

#### *3.5.3.4.3 Lpms\_imu*

- imu, message de type **sensor\_msgs/Imu** (packages : lpms\_imu, timesync\_ros)
  - **std\_msgs/Header** header
    - uint32 seq
    - time stamp
    - string frame\_id
  - **geometry\_msgs/Quaternion** orientation
    - float64 x
    - float64 y
    - float64 z
    - float64 w
    - float64[9] orientation\_covariance
  - **geometry\_msgs/Vector3** angular\_velocity
    - float64 x
    - float64 y
    - float64 z
    - float64[9] angular\_velocity\_covariance
  - **geometry\_msgs/Vector3** linear\_acceleration
    - float64 x
    - float64 y
    - float64 z
    - float64[9] linear\_acceleration\_covariance

#### *3.5.3.4.4 Tf\_broadcaster*

- **/tf**, message de type **geometry\_msgs/TransformStamped** :
  - **std\_msgs/Header** header
    - string child\_frame\_id
  - **geometry\_msgs/Transform** transform
  - **geometry\_msgs/Vector3** translation
    - float64 x
    - float64 y
    - float64 z
  - **geometry\_msgs/Quaternion** rotation
    - float64 x
    - float64 y
    - float64 z
    - float64 w



#### 3.5.3.4.5 *Navigation.launch*

Ce qui va suivre est le détail du fichier « navigation.launch ». Avec les nodes et les launches, puis sur quels **topics** ceux-ci publient avec les **types de message** qui transitent sur ces **topics**.

##### 3.5.3.4.5.1 One\_robot.launch

###### 3.5.3.4.5.1.1 *Map\_server*

- **/map**, message de type **nav\_msgs/OccupancyGrid** :
  - **std\_msgs/Header** header
  - **nav\_msgs/MapMetaData** info
    - time map\_load\_time
    - float32 resolution
    - uint32 width
    - uint32 height
  - **geometry\_msgs/Pose** origin
  - **geometry\_msgs/Point** position
    - float64 x
    - float64 y
    - float64 z
  - **geometry\_msgs/Quaternion** orientation
    - float64 x
    - float64 y
    - float64 z
    - float64 w
    - int8[] data

###### 3.5.3.4.5.1.2 *Simple\_navigation\_goal*

- **/tf**
- **/tf\_static**

##### 3.5.3.4.5.2 Move\_base.launch

*Move\_base* :

- **/odom**
- **/move\_base/goal**, message de type **move\_base\_msgs/MoveBaseActionGoal** :
  - **std\_msgs/Header** header
  - **actionlib\_msgs/GoalID** goal\_id
    - time stamp
    - string id
- **move\_base\_msgs/MoveBaseGoal** goal
- **geometry\_msgs/PoseStamped** target\_pose
- **std\_msgs/Header** header
- **geometry\_msgs/Pose** pose
- **geometry\_msgs/Point** position
  - float64 x
  - float64 y
  - float64 z

- **geometry\_msgs/Quaternion** orientation
  - float64 x
  - float64 y
  - float64 z
  - float64 w
- **/move\_base\_simple/goal**, message de type **move\_base\_msgs/MoveBaseActionGoal** :
  - **geometry\_msgs/PoseStamped** target\_pose

3.5.3.4.5.3 Amcl.launch

Amcl :

- **/amcl\_pose**, message de type **geometry\_msgs/PoseWithCovarianceStamped** :
  - **geometry\_msgs/PoseWithCovariance** pose
  - **geometry\_msgs/Pose** pose
  - **geometry\_msgs/Point** position
    - float64 x
    - float64 y
    - float64 z
  - **geometry\_msgs/Quaternion** orientation
    - float64 x
    - float64 y
    - float64 z
    - float64 w
    - float64[36] covariance
- **/particlecloud**, message de type **geometry\_msgs/PoseArray** :
- **std\_msgs/Header** header
- **geometry\_msgs/Pose[]** poses
- **geometry\_msgs/Point** position
- **geometry\_msgs/Quaternion** orientation
- **/scan**
- **/initialpose** (pareil que **/amcl\_pose**)

### 3.5.3.5 Digital Twin

Comme vous avez pu le constater, nous avons évoqué à plusieurs reprises l'utilisation de RVIZ pour visualiser les données renvoyées par les capteurs du robot Héron. En plus de cela nous avons créé un « jumeau numérique » de notre robot Héron (c'est-à-dire une représentation du robot Héron au format numérique) qui reproduira les mêmes actions réalisées par Héron dans la réalité. Il permettra de mieux se repérer dans l'espace et de situer le robot par rapport aux données des capteurs. RVIZ étant notre logiciel de simulation, permettant d'avoir un retour graphique de toutes les données des capteurs, ainsi que de savoir où sont placés les éléments du robot. Il fallait qu'il puisse être dans le même sens que le réel, qu'il puisse se déplacer en même temps que le réel et que son centre soit le centre de la partie basse du robot.

Voici l'apparence de ce digital twin :

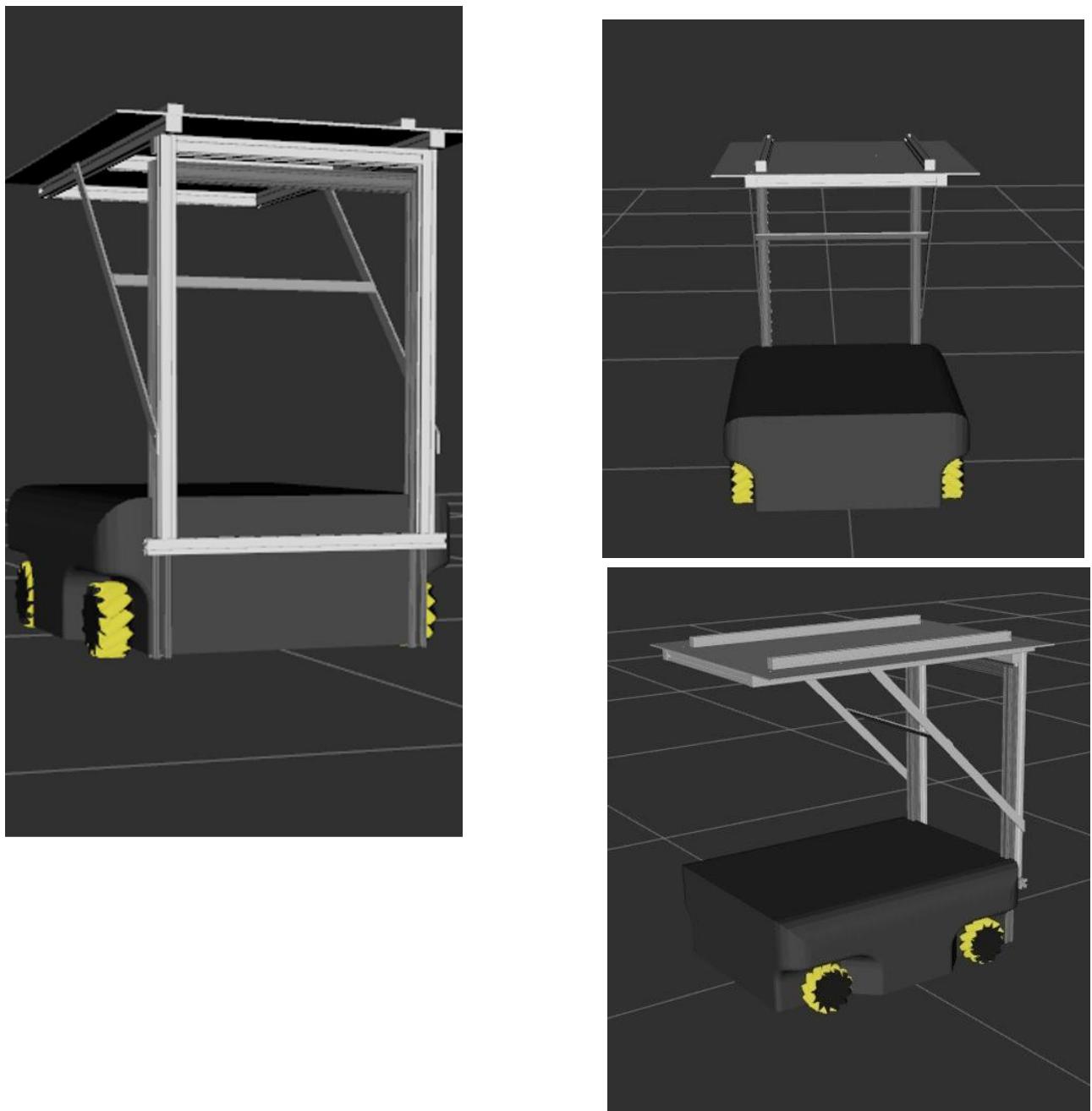


Figure 88 Héron Digital Twin

### 3.5.3.5.1 Fichier URDF

Pour créer un modèle du robot dans ROS, il faut créer un fichier URDF du robot.

Qu'est-ce qu'un fichier URDF ?

Un fichier URDF est un fichier qui peut être lu par les logiciels de simulation dans ROS tel que RVIZ et GAZEBO. Il va reprendre le modèle du robot que l'on veut mettre dans la simulation. Il reprendra les pièces de l'assemblage (appelés link dans l'URDF) et va assembler les pièces selon le type de liaison (appelé joint ici). Chaque joint aura une spécification selon l'axe sur lequel il fonctionne.

Les caractéristiques des links devront être spécifiées dans l'URDF : Aspect, taille, poids, placement, orientation, type de liaison mécanique, origine de liaison, simulation de transmission par le moteur.

L'aspect des links est créé dans l'URDF, soit par des fonctionnalités de base, soit par une balise "mesh" qui va reprendre un fichier STL ou DAE, et va donc reprendre une pièce qui vient de solidworks.

Il va falloir aussi spécifier le joint dans le fichier par un type de liaison, un pivot, une glissière, fixe ainsi que le link parent du joint, et le link child.

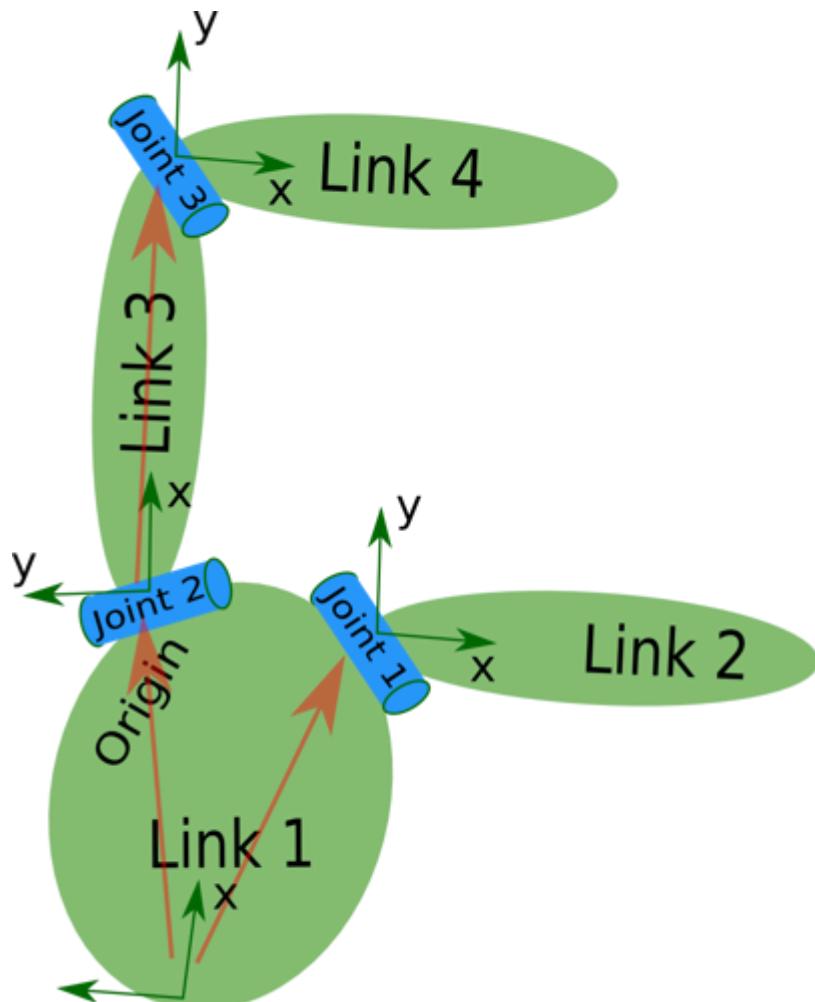


Figure 89 Schéma représentant les "link" et les "joints"

On peut le créer directement comme dans le tutoriel suivant :

<http://wiki.ros.org/fr/urdf/Tutorials/Create%20your%20own%20urdf%20file> ou on peut utiliser une fonction de solidworks qui est solidworks urdf exporter ([http://wiki.ros.org/sw\\_urdf\\_exporter](http://wiki.ros.org/sw_urdf_exporter)) en faisant avec notre modèle solidworks (Il vaut mieux utiliser un modèle simplifié car sinon il sera trop compliqué à charger) comme dans le tutoriel suivant:

[http://wiki.ros.org/sw\\_urdf\\_exporter/Tutorials/Export%20an%20Assembly](http://wiki.ros.org/sw_urdf_exporter/Tutorials/Export%20an%20Assembly).

### 3.5.3.5.2 URDF du robot Héron

#### 3.5.3.5.2.1 Base\_link

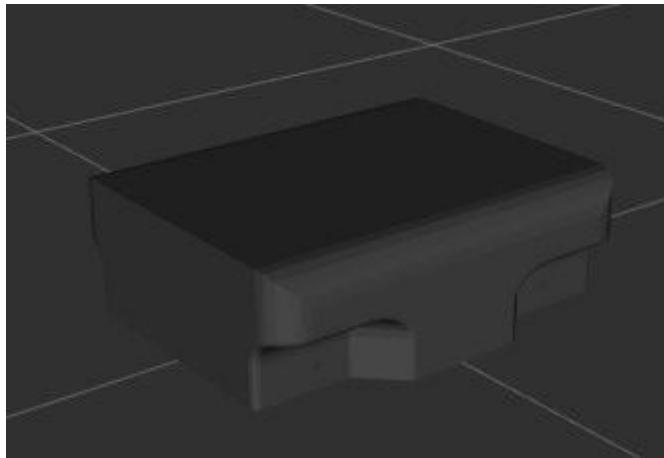


Figure 91 Aperçu de 'base\_link'

Base\_link est la partie basse de notre robot. Dans notre link, auquel on aura spécifié un nom. On retrouve deux parties essentielles, "inertial" et "visual". Inertial est la partie où l'on spécifie toutes les particularités physiques du robot, avec l'origine physique du robot (son centre de gravité), la masse et sa matrice d'inertie. La partie visual permet de voir l'aspect de la partie dans le logiciel, avec l'origine qui est le centre du robot, mesh qui reprend le fichier STL de la pièce, les matériaux, les couleurs..

```
<link
  name="base_link">
  <inertial>
    <origin
      xyz="-0.15 0.025 0.0"
      rpy="0 0 1.57079" />
    <mass
      value="25" />
    <inertia
      ixx="4.73345232681242"
      ixy="-6.13913566072873E-06"
      ixz="-1.89872416704455E-05"
      iyy="6.95282353462315"
      iyz="-8.07624704791103E-06"
      izz="3.18752499734018" />
  </inertial>
  <visual>
    <origin
      xyz="-0.15 0.025 0.0"
      rpy="0 0 1.57079" />
    <geometry>
      <mesh
        filename="package://heron/meshes/base_link.STL"/>
    </geometry>
    <material
      name="">
      <color
        rgba="0.3 0.3 0.3 1" />
    </material>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://heron/meshes/base_link.STL" />
    </geometry>
  </collision>
</link>
```

Figure 90 URDF de 'base\_link'

### 3.5.3.5.2.2 Base\_footprint

```
<robot
  name="heron_urdf">

  <link name="base_footprint">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.001 0.001 0.001"/>
      </geometry>
    </visual>
  </link>
  <joint name="base_footprint" type="fixed">
    <origin xyz="0 0 0.10" rpy="0 0 0"/>
    <parent link="base_footprint"/>
    <child link="base_link"/>
  </joint>
```

Figure 92 URDF de 'base\_footprint'

Base\_footprint représente le sol, il permettra au robot d'être bien placé selon z par rapport au sol. Ici on a la balise "box" qui permet de créer une boite (ici non visuelle). Ici, on spécifie une liaison entre base\_footprint et base\_link, elle est fixe et on retrouve base\_link à 0.10 m de base\_footprint (spécifié grâce à origin).

### 3.5.3.5.2.3 Wheel

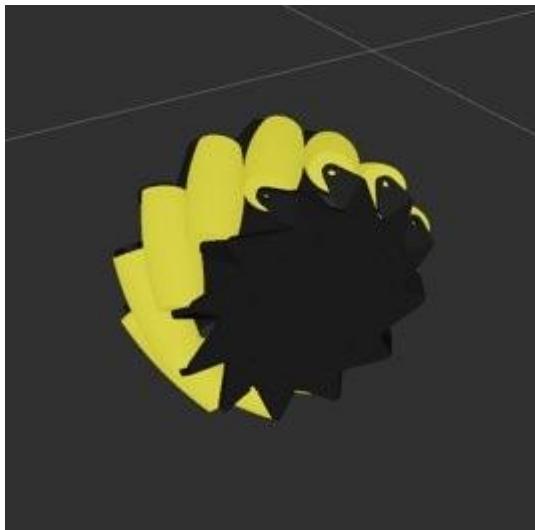


Figure 93 Aperçu de 'wheel'

Ici, on retrouve les mêmes types de balises dans joint que les autres, mais ici dans mesh, on redimensionne la roue grâce à scale.

```
<link
  name="wheel_FL">
  <inertial>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <mass
      value="0.389968906998643" />
    <inertia
      ixx="0.000322138046158454"
      ixy="-6.65852858947508E-37"
      ixz="-1.86618052242594E-37"
      iyy="0.00048299257130799"
      iyz="5.80382085092595E-22"
      izz="0.000322138046158454" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://heron/meshes/wheel_FL.dae" scale="0.393700787 0.5 0.393700787" />
    </geometry>
    <material
      name="" />
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://heron/meshes/wheel_FL.dae" scale="0.393700787 0.5 0.393700787" />
    </geometry>
  </collision>
</link>
```

Figure 94 URDF de 'wheel' (partie link)

```

<joint
  name="wheel_FL"
  type="continuous">
  <origin
    xyz="0.2 0.216235 -0.06"
    rpy="0 0 0" />
  <parent
    link="base_link" />
  <child
    link="wheel_FL" />
  <axis
    xyz="0 1 0" />
  <safety_controller
    k_velocity="0" />
  <limit effort="9.1201845" velocity="8.79645942"/>
</joint>
<!-- Transmission is important to link the joints and the controller (see summit_xl con
<transmission name="wheel_FL_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="wheel_FL">
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
  </joint>
  <actuator name="wheel_FL_motor">
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
<gazebo reference="wheel_FL">
  <mu1 value="1.0"/>
  <mu2 value="1.0"/>
  <kp value="10000000.0" />
  <kd value="1.0" />
  <fdirl value="1 0 0"/>
    <turnGravityOff>false</turnGravityOff>
</gazebo>

```

Figure 95 URDF de 'wheel' (partie joints)

La partie joint est différente :

- Joint de type "revolute"
- Placement de la roue via la section "joint origin"
- Parent : base\_link
- Child : wheel
- Axis : Axe du pivot (y)
- Limit: Limites réelles du robot en N.m et Rad/s
- Transmission : Permet la transmission de puissance sur le sol des roues via un moteur que l'on simule
- Gazebo : Section pour le logiciel robot, paramètres pour roues

Transmission et gazebo ne sont utilisables que dans GAZEBO, un autre logiciel de simulation, mais où l'on peut simuler un couple appliqué aux moteurs.

### 3.5.3.5.2.4 Support

Support est le support de la glissière, il relie le plateau et la partie basse du robot. On ne retrouve rien d'inconnu par rapport aux précédents link.

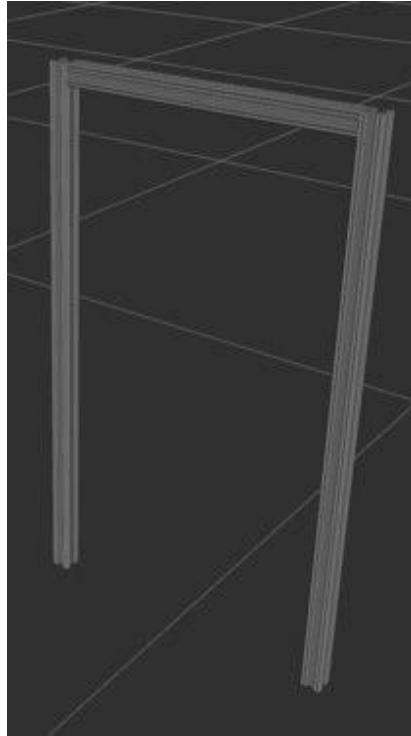


Figure 96 Aperçu de support

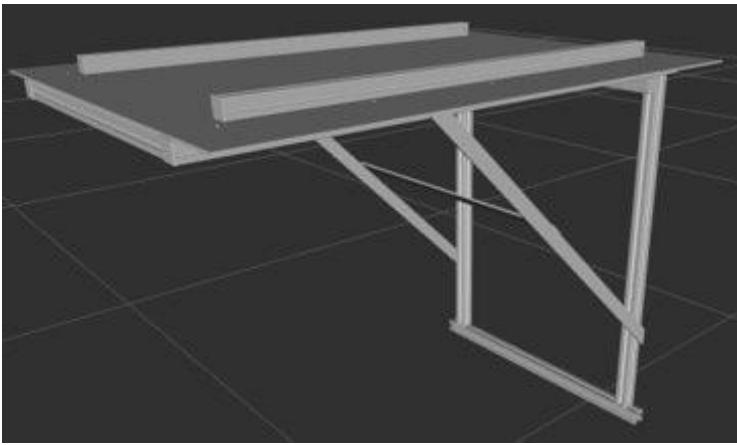
```
<joint
  name="support"
  type="fixed">
<origin
  xyz="-0.3 0 0"
  rpy="0 0 0" />
<parent
  link="base_link" />
<child
  link="support" />
<safety_controller
  k_velocity="0" />
</joint>
```

Figure 97 URDF de 'support' (partie joints)

```
<link
  name="support">
<inertial>
<origin
  xyz="0 0 0"
  rpy="0 0 0" />
<mass
  value="0.389968906998643" />
<inertia
  ixx="0.20"
  ixy="0"
  ixz="0"
  iyy="0.10"
  iyz="-0.08"
  izz="0.14" />
</inertial>
<visual>
<origin
  xyz="-0.015 -0.19 -0.1"
  rpy="1.5707963267949 0 1.5707963267949" />
<geometry>
<mesh
  filename="package://heron/meshes/support.STL" />
</geometry>
<material
  name="">
<color
  rgba="0.5 0.5 0.5 1" />
</material>
</visual>
<collision>
<origin
  xyz="0 0 0"
  rpy="0 0 0" />
<geometry>
<mesh
  filename="package://heron/meshes/support.STL" />
</geometry>
</collision>
</link>
```

Figure 98 URDF de 'support' (partie link)

### 3.5.3.5.2.5 Plate



```

<link
  name="plate">
  <inertial>
    <origin
      xyz="-0.325 -0.25 -0.1"
      rpy="1.5707963267949 0 1.5707963267949" />
    <mass
      value="0.389968906998643" />
    <inertia
      ixx="1.47"
      ixy="0.09"
      ixz="-0.31"
      iyy="1.46"
      iyz="0.39"
      izz="0.35" />
  </inertial>
  <visual>
    <origin
      xyz="-0.33 -0.25 -0.1"
      rpy="1.5707963267949 0 1.5707963267949" />
    <geometry>
      <mesh
        filename="package://heron/meshes/plate.STL" />
    </geometry>
    <material
      name="">
      <color
        rgba="0.9 0.9 0.9 1" />
    </material>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://heron/meshes/plate.STL" />
    </geometry>
  </collision>
</link>
```

```

<joint
  name="plate"
  type="prismatic">
  <origin
    xyz="-0.32 0 0.13"
    rpy="0 0 0" />
  <parent
    link="base_link" />
  <child
    link="plate" />
  <axis
    xyz="0 0 1" />
  <safety_controller
    k_velocity="0" />
  <limit effort="147" lower="0.0"
    upper="0.386" velocity="0.039"/>
</joint>
```

Plate est le plateau du robot. Les caractéristiques sont les suivantes :

- Plateau du robot : Plate
- Joint de type "prismatic" (glissière)
- Parent : base\_link
- Child : plate
- Axis : Axe de la glissière (z)
- Limit : Limites réelles de la glissière en N et m/s
- Upper : Delta le plus haut de la glissière
- Lower : Delta le plus bas

Figure 99 Aperçu de 'plate' / URDF de plate (partie joints) / URDF de plate (partie link)

### 3.5.3.5.3 Fichier display.launch

Le fichier display.launch est le fichier de lancement du digital twin. Il va lancer tous les programmes nécessaires ainsi que charger l'URDF.

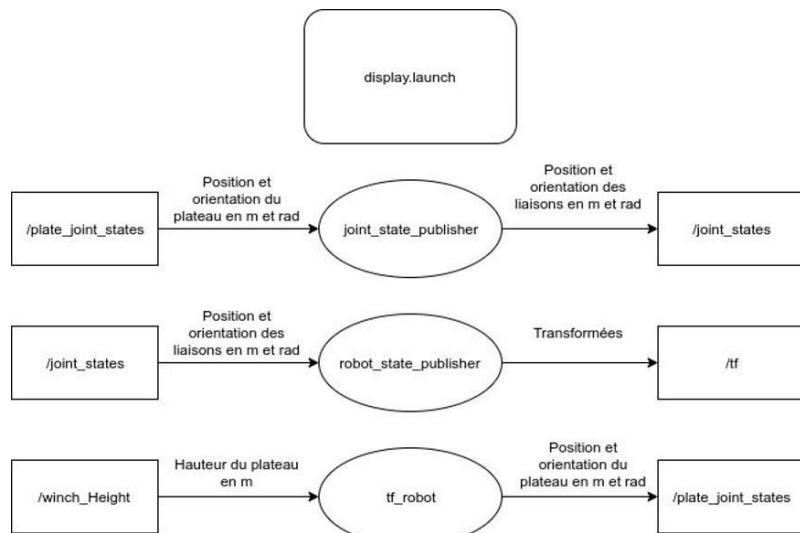
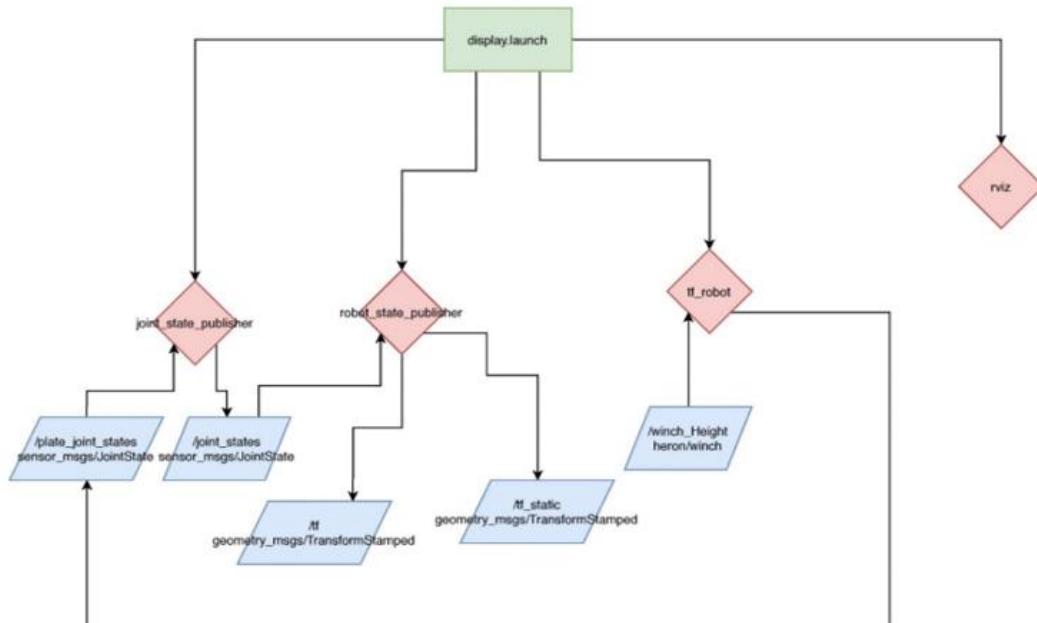


Figure 100 Schéma entrées sorties de « display.launch »



<b>joint_state_publisher :</b> Récupère dans l'urdf (le fichier du robot de la simulation), les états des joints (entre les roues et la base par exemple) et les publie	<b>robot_state_publisher :</b> Récupère les joint_states et fait les transformées de chaque joint sur tf et tf_static	<b>tf_robot :</b> Récupère la hauteur du plateau depuis winch_Height puis publie sur plate_joint_states pour que joint_states récupère la hauteur du plateau et la mette sur la simulation
--	--	---

Figure 101 Schéma du fichier « display.launch »

```

<launch>

<group ns="$(optenv HERON_ID Heron00)">

    <arg
        name="model" />
    <arg
        name="gui"
        default="False" />
    <param
        name="robot_description"
        textfile="$(find heron)/urdf/heron_urdf.urdf" />
    <param name="tf_prefix" value="$(optenv HERON_TF Heron00)" />
    <param
        name="use_gui"
        value="$(arg gui)" />
    <node name="joint_state_publisher"
        pkg="joint_state_publisher"
        type="joint_state_publisher">
        <rosparam param="source_list">["plate_joint_states"]</rosparam>
        <param name="tf_prefix" value="$(optenv HERON_TF Heron00)" />
    </node>
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher">
        <param name="tf_prefix" value="$(optenv HERON_TF Heron00)" />
    </node>

    <node name="tf_robot" pkg="heron" type="tf_robot" respawn="true" output="screen" />
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find heron)/urdf.rviz" >
        <remap from="/initialpose" to="initialpose"/>
        <remap from="/move_base_simple/goal" to="move_base_simple/goal"/>
    </node>
</group>
</launch>

```

Figure 102 Fichier « display.launch »

- Paramètre : robot\_description pour aller chercher le fichier urdf
- Différents nodes de lancés :
  - Joint\_state\_publisher
  - Robot\_state\_publisher
  - Tf\_robot
  - Rviz

#### 3.5.3.5.3.1 Display.launch détaillé

Ce qui va suivre est le détail du fichier « display.launch ». Avec les nodes et les launches, puis sur quels topics ceux-ci publient avec les types de message qui transitent sur ces topics.

- o Joint\_state\_publisher :
  - o /plate\_joint\_states, message de type sensor\_msgs/JointState :
    - o std\_msgs/Header header
      - o string[] name
      - o float64[] position
      - o float64[] velocity
      - o float64[] effort
  - o /joint\_states, message de type sensor\_msgs/JointState :
    - o std\_msgs/Header header
      - o string[] name
      - o float64[] position
      - o float64[] velocity
      - o float64[] effort
- o Robot\_state\_publisher :

- /tf
- /tf\_static
- Tf\_robot :
  - /winch\_Height
  - Rviz

#### 3.5.3.5.3.1.1 Joint\_state\_publisher

```
<node name="joint_state_publisher"
      pkg="joint_state_publisher"
      type="joint_state_publisher" >
  <rosparam param="source_list">["plate_joint_states"]</rosparam>
  <param name="tf_prefix" value="$(optenv HERON_TF Heron00)" />
</node>
```

Figure 103 Node « joint\_state\_publisher » dans le « display.launch »

```
header:
  seq: 4103
  stamp:
    secs: 1576763812
    nsecs: 341048955
  frame_id: ''
name: [wheel_FL, wheel_FR, wheel_BL, wheel_BR, plate]
position: [0.0, 0.0, 0.0, 0.0, 0.0]
velocity: []
effort: []
---
```

Figure 104 Type de message sur « /joint\_states »

Joint\_state publisher est un node qui est déjà intégré par ROS, il va publier dans le topic “/joint\_states” l’état des “joint” selon les paramètres que l’on peut changer (par exemple, la glissière, pour pouvoir la faire lever, on vient changer le paramètre z en mètres, les roues la rotation en z en radians etc..).

Ici, le node souscrit au topic “/plate\_joint\_states”, les données qui seront publiées sur ce topic vont être récupérées et fusionnées dans le topic “/joint\_states”

#### 3.5.3.5.3.1.2 Robot\_state\_publisher

```
<node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher">
  <param name="tf_prefix" value="$(optenv HERON_TF Heron00)" />
</node>
```

Figure 105 « robot\_state\_publisher » dans le « display.launch »

- Node intégré à ROS
- Souscrit au topic “/joint\_states”
- Récupère dans l'URDF les parents et enfants
- Fait les transformées et les publie dans “/tf”
- Souscrit au topic “/joint\_states” pour qu'il fasse la transformée s'il voit une donnée pour un joint de changé

### 3.5.3.5.3.1.3 Tf\_robot

```
header:  
  seq: 0  
  stamp:  
    secs: 1576763883  
    nsecs: 840984106  
  frame_id: "Heron00/base_link"  
child_frame_id: "Heron00/wheel_FR"  
transform:  
  translation:  
    x: 0.2  
    y: -0.216235  
    z: -0.06  
  rotation:  
    x: 0.0  
    y: 0.0  
    z: 0.0  
    w: 1.0
```

- Permet de lever la glissière dans la simulation
- Souscrit à "/winch\_Height"
- Récupère la hauteur actuelle du plateau
- La publie sur "/plate\_joint\_states"
- Simulation :
- joint\_state\_publisher récupère
- robot\_state\_publisher fait la transformée sur la simulation

Figure 106 Type de message sur « /tf »

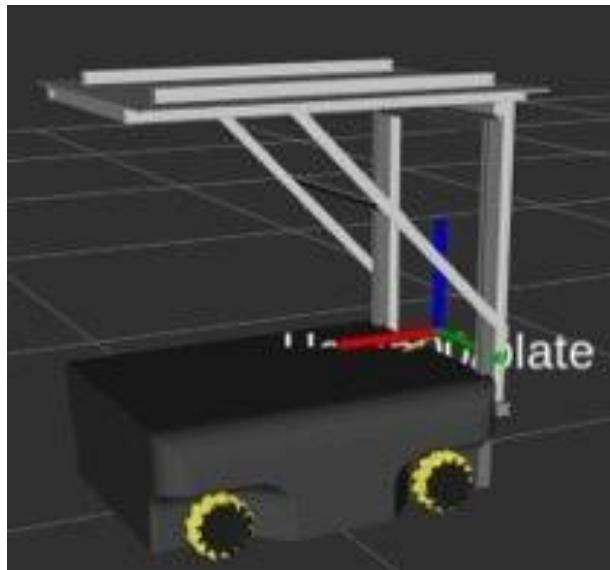


Figure 107 Glissière en position basse sur le digital twin

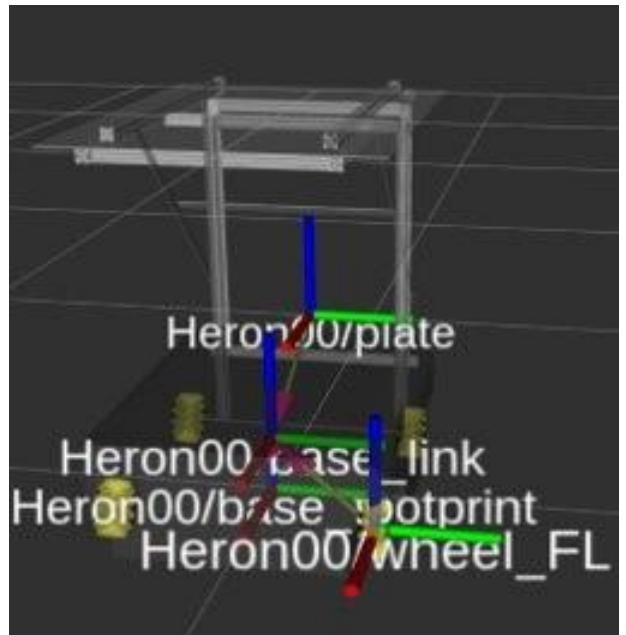


Figure 108 TF des joints du robot

### 3.5.3.6 Bouton d'arrêt d'urgence

Le bouton d'arrêt d'urgence sert à couper toute source de mouvement du robot dans le cas où celui-ci irait à un endroit non prévu ou pouvant l'endommager. Il est situé sur le côté avant gauche du robot pour rester accessible.

Si l'on appui dessus il se mettra en position bloquée circuit OUVERT (NF), si on le débloque, cela fermera le circuit et le courant passera. Ce bouton est fourni avec 2 interrupteurs un NF-(Normalement Fermé) et un NO-(Normalement Ouvert). Le 2<sup>e</sup> ne sert pas car il faut appuyer sur le bouton pour fermer le circuit, ce que l'on ne souhaite pas. Seul le 1<sup>er</sup> (NF) est utilisé dans le robot.

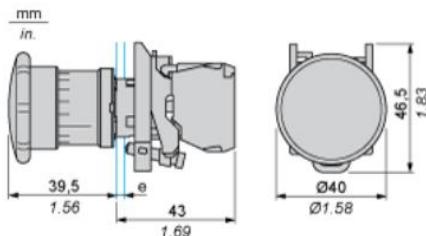
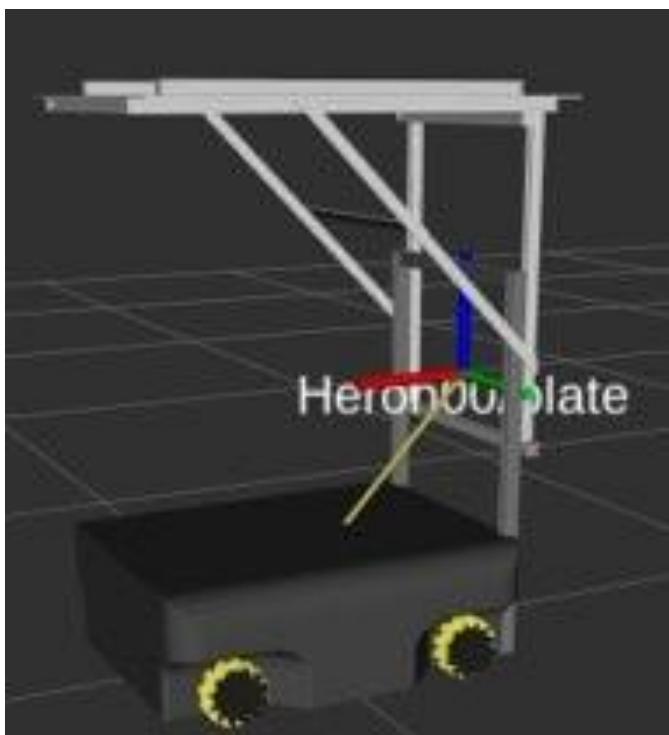


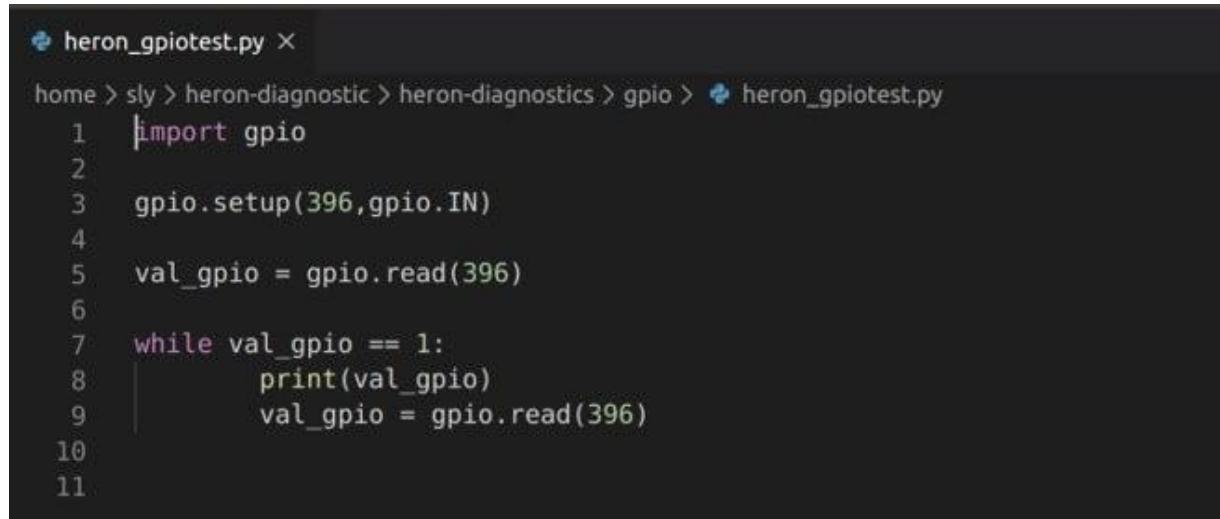
Figure 109 Représentation bouton d'arrêt d'urgence

Pour éviter que les programmes liés aux drivers crashent lorsque le bouton d'arrêt d'urgence est enclenché et qu'il faille relancer les programmes à la main. Il faut pouvoir détecter si le bouton d'arrêt d'urgence a été détecté. C'est pourquoi nous l'avons relié au gpio396 de la Jetson. On va pouvoir savoir si le bouton d'arrêt d'urgence est à l'état haut ou à l'état bas grâce à une bibliothèque python. Tout ceci sera expliqué dans la partie développement et tutoriel.



### 3.5.3.6.1 Développement

Un code test sous python qui permettra de lire le gpio396, celui-ci va lire “1” lorsque le bouton ne sera pas enclenché et va s’arrêter lorsque le bouton sera enclenché, c'est-à-dire qu'il lira “0”.



```
heron_gpiotest.py ×
home > sly > heron-diagnostic > heron-diagnostics > gpio > heron_gpiotest.py
1 import gpio
2
3 gpio.setup(396,gpio.IN)
4
5 val_gpio = gpio.read(396)
6
7 while val_gpio == 1:
8     print(val_gpio)
9     val_gpio = gpio.read(396)
10
11
```

Figure 110 Code source « heron\_gpiotest.py »

### 3.5.3.6.2 Tutoriel

Dans Ubuntu, nous sommes capables de détecter si un arrêt d'urgence a été enclenché ou non. Pour cela, il a fallu connecter le bouton d'arrêt d'urgence au gpio396 sur la PIN 7 du header J21 (4ème PIN du bas en partant de la gauche, voir image ci-dessus).

Pour plus d'information et installer la bibliothèque gpio de python : (J'ai suivi le tutoriel de joel.reindel qui explique comment faire et comment l'utiliser)

: <https://devtalk.nvidia.com/default/topic/1030443/jetson-tx2/using-gpio-on-nvidia-jetson-tx2/>.

La bibliothèque qu'il utilise : <https://github.com/vitiral/gpio>.

@manuelospina  
Hey there, Im guessing you figured this out by now but I thought I would add a little tutorial for those who might need help in the future. I don't know why the readme from git hub doesn't have the install instructions... here is what I did.

In the terminal:

```
1. $ cd
2. $ git clone https://github.com/vitiral/gpio.git
3. $ cd gpio/
4. $ sudo python3 setup.py build
5. $ sudo python3 setup.py install
```

This installs the library to /sys/class/gpio (in case your are looking for it). I suppose this could be different for you. Check it by running python3 term:  
"import gpio; gpio.GPIO\_root"

After that run the gpio test script.

```
1. $ cd tests/
2. $ python3 test_gpio.py
```

I'm not totally sure what the script actually does, seems like all it does is import some stuff. If the install is successful there will be no output.

The readme includes the commands you can run. Here is an example.  
In a bash term enter: "sudo cat /sys/kernel/debug/gpio" to see the currently enabled sysfs GPIOs. The last section "...tegra-gpio:" is where the gpios we want will show up.  
To run some of the commands you need to have root privileges, so start a terminal with "sudo python3":

```
1. import gpio
2. >>> gpio.setup(396,gpio.OUT) #this will setup J21 pin 7 as an output pin. If you rerun "sudo cat /sys/kernel/debug/gpio" you will now see gpio-396 listed as 'out hi' and j21
3. >>> gpio.set(396,1) # rerun "sudo cat /sys/kernel/debug/gpio" you will now see gpio-396 listed as "out hi" and j21
4. >>> gpio.output(396,0) # this also works to drive pins
5. >>> gpio.mode(396) # this shows the current pin mode 'out' right now
6. >>> gpio.setup(396,gpio.IN) # set the pin to input mode
7. >>> gpio.mode(396) # confirms this
8. >>> gpio.read(396) # reads the pin value, I don't think this library does anything with pullup or pull down resistor
9. >>> gpio.input(396) # does the same thing
10. >>> val = gpio.input(396)
11. >>> type(val) # <class 'int'> so you need to use typecasting if you need something else.
12. >>> gpio.cleanup(396) # closes and unexports the pin. Returning it to the system.
```

Figure 111 Tutoriel pour gpio sur la Jetson TX2

### 3.5.3.7 Détection de fin de stock

L'industrie 4.0 et toutes les technologies qui la composent ont aussi pour mission d'épauler les techniciens travaillant dans cet environnement.

Une de nos idées était d'utiliser la caméra ZED positionnée sur le robot Héron afin d'analyser l'entrepôt de stockage, checker l'ensemble du stockage et relever les fins de stock afin de prévenir un technicien. Celui-ci pourra ainsi anticiper la réalimentation de l'entrepôt de stockage. Cette partie n'ayant pas pu être développée au cours de nos différents sprints, nous avons tout de même réfléchi à un concept pouvant être mis en place pour mettre en place cette nouvelle fonctionnalité.

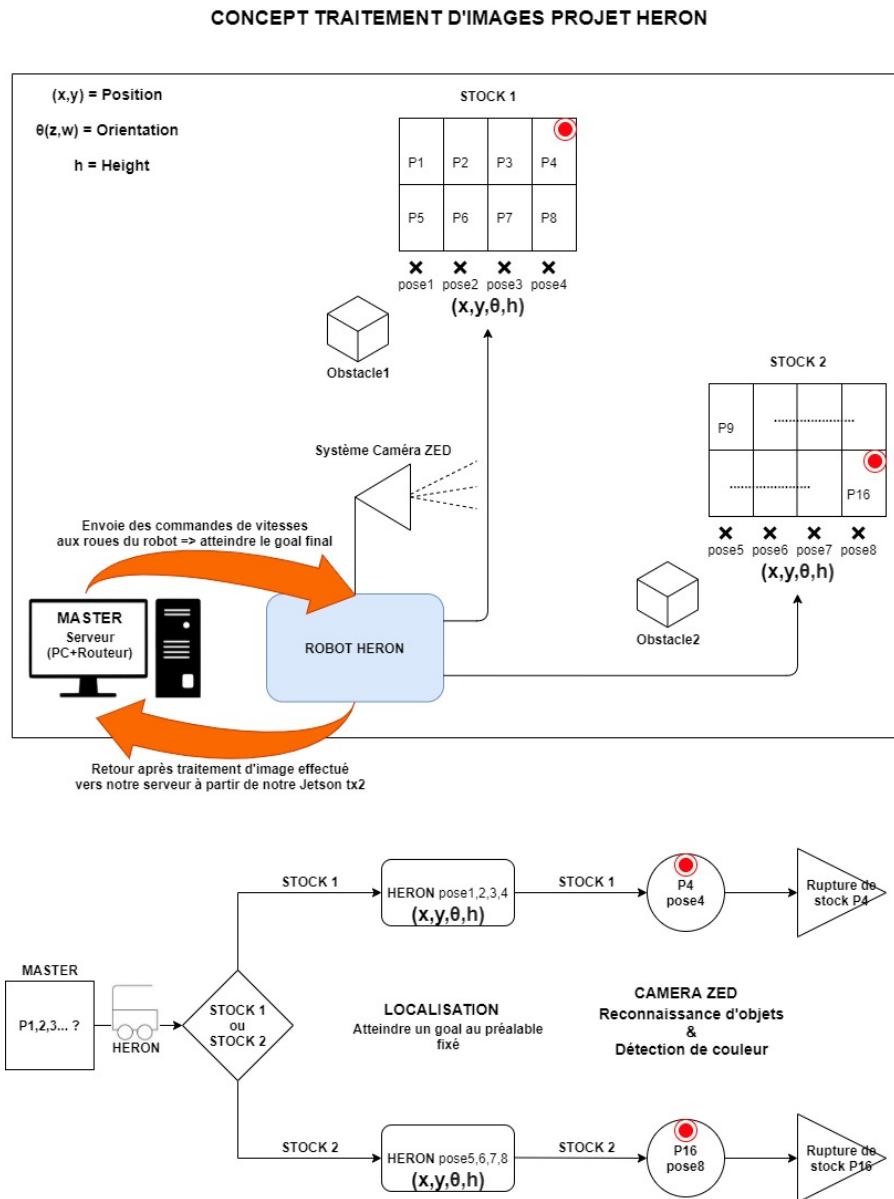


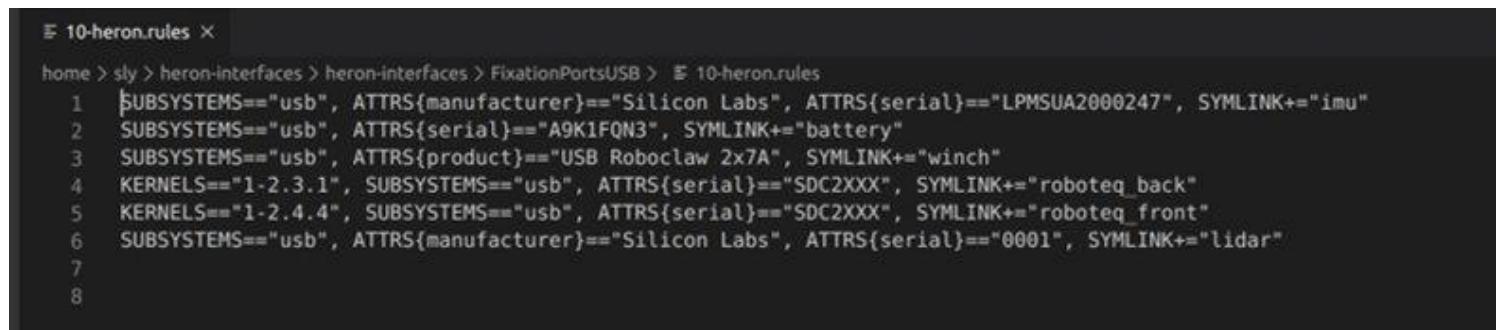
Figure 112 Concept de détection de fin de stock

### 3.5.3.8 Tutoriels pour assurer la commande haut-niveau

#### 3.5.3.8.1 Installation Ubuntu 16.04

!!!! ATTENTION !!!! : L'installation de Ubuntu sur la Jetson TX2 n'est pas conventionnelle, il faut suivre la démarche suivante :

Télécharger Jetpack 3.3 à l'adresse suivante : <https://developer.nvidia.com/embedded/jetpack->



```
home > sly > heron-interfaces > heron-interfaces > FixationPortsUSB > 10-heron.rules
1 #SUBSYSTEMS=="usb", ATTRS{manufacturer}=="Silicon Labs", ATTRS{serial}=="LPM5UA2000247", SYMLINK+="imu"
2 #SUBSYSTEMS=="usb", ATTRS{serial}=="A9K1FQN3", SYMLINK+="battery"
3 #SUBSYSTEMS=="usb", ATTRS{product}=="USB Roboclaw 2x7A", SYMLINK+="winch"
4 KERNELS=="1-2.3.1", SUBSYSTEMS=="usb", ATTRS{serial}=="SDC2XXX", SYMLINK+="roboteq_back"
5 KERNELS=="1-2.4.4", SUBSYSTEMS=="usb", ATTRS{serial}=="SDC2XXX", SYMLINK+="roboteq_front"
6 #SUBSYSTEMS=="usb", ATTRS{manufacturer}=="Silicon Labs", ATTRS{serial}=="0001", SYMLINK+="lidar"
7
8
```

Figure 113 Fichier « 10-heron.rules »

#### 3.3 (se créer un compte NVIDIA Developer)

Suivre la vidéo tutoriel suivante pour installer Ubuntu 16.04 sur la Jetson TX2: <https://www.youtube.com/watch?v=9uMvXqhjxaQ>

#### 3.5.3.8.2 Installer ROS Kinetic

Ouvrir un terminal (Ctrl + Alt + T) et taper :

```
git clone https://github.com/jetsonhacks/installROSTX2.git
cd installROSTX2
./installROS.sh -p ros-kinetic-desktop-full
```

Puis suivre le tutoriel “3.Créer un Workspace ROS” à la page suivante <http://wiki.ros.org/fr/ROS/Tutorials/InstallingandConfiguringROSEnvironment> (Utiliser la commande “catkin build” au lieu de “catkin\_make”, c'est préférable).

A la fin, faire dans l'ordre :

```
cd
nano .bashrc
```

Aller à la fin du fichier et écrire les choses suivantes :

```
source /opt/ros/kinetic/setup.bash
source catkin_ws/devel/setup.bash
export ROS_MASTER_URI=http://10.224.0.52 :11311
export ROS_HOSTNAME=localhost
```

Faire “Ctrl + X” puis “y” puis “Entrée”

Création d'un package ROS : <http://wiki.ros.org/fr/ROS/Tutorials/CreatingPackage> (Cependant pour le robot nous avons déjà des packages, voir dans la partie “Architecture logicielle” dans la suite).

Tous les capteurs seront branchés au HUB, nous pourrons les reconnaître sur les différents ports qui ont été fixés. Voici comment les fixer et la liste des ports utilisés :

Pour ce qui est propre au robot, aller dans le Github suivant : <https://github.com/yncreahdf-robotics/heron-interfaces/tree/Dev> et copier le fichier “10-heron.rules” situé dans FixationPortsUSB dans le dossier “/lib/udev/rules.d/” situé dans le système UBUNTU, puis redémarrer le système pour faire la MAJ des ports.

Pour écrire sa propre règle :

<http://guidella.free.fr/General/ecrireReglePourUdev.html#sec10>

#### 3.5.3.8.3 *Intégration IMU dans Jetson TX2*

Pour intégrer l’IMU dans la Jetson TX2, il faut suivre le tutoriel suivant : <https://lp-research.com/ros-and-lp-research-imus-simple/>

Cependant, il faut télécharger LpSensor pour Jetson TX1 et non Ubuntu sur <https://lp-research.com/support/>

Mais il faut modifier “LpSensor-1.3.5-Linux-x86-64” par “LpSensor-1.3.5-Linux-arm64” et aussi “catkin\_make” par “catkin build” lors de l’exécution des commandes.

Mais AVANT de “catkin build”,  
télécharger pugixml ici <https://archlinuxarm.org/packages/aarch64/pugixml>, extraire le fichier puis copier tous les fichiers présents dans les dossiers de pugixml/usr dans le dossier /usr/de ubuntu dans les dossiers correspondants.

Pour se faire, ouvrir un terminal puis faire les commandes suivantes :

```
sudo cp CHEMIN_PUGIXML/usr/CHEMIN_DOSSIER(S)/NOM_DU_FICHIER /usr/CHEMIN_DOSSIER
sudo cp Downloads/pugixml-1.10-1-aarch64.pkg/usr/include/. /usr/include
sudo      cp      -R      Downloads/pugixml-1.10-1-aarch64.pkg/usr/lib/cmake/pugixml/
/usr/lib/cmake/
sudo cp -R Downloads/pugixml-1.10-1-aarch64.pkg/usr/lib/pkgconfig/ /usr/lib/
sudo cp Downloads/pugixml-1.10-1-aarch64.pkg/usr/lib/libpugixml.so /usr/lib/
sudo cp Downloads/pugixml-1.10-1-aarch64.pkg/usr/lib/libpugixml.so.1/ /usr/lib/
sudo cp Downloads/pugixml-1.10-1-aarch64.pkg/usr/lib/libpugixml.so.1.10/ /usr/lib/
sudo cp -R Downloads/pugixml-1.10-1-aarch64.pkg/usr/share/licenses/ /usr/share/
```

Suite à cela, à la ligne 50 dans catkin\_ws/src/lpms\_imu/src/lpms\_imu\_node.cpp, remplacer « /dev/ttyUSBO » par « /dev imu ».

#### 3.5.3.8.4 *Intégration du convertisseur CAN to USB pour les IR*

Pour intégrer le convertisseur CAN to USB sur la Jetson TX2, veuillez suivre le tutoriel disponible à la page suivante (le premier message de boyuan99) :

/ ! \ Il ne faut pas faire l'étape suivante qui se trouve dans le tutoriel, sinon le convertisseur ne sera pas reconnu :

Next, find the line CONFIG\_LOCALVERSION= ,  
make it CONFIG\_LOCALVERSION="-tegra"  
This step is to avoid the version magic error.

<https://devtalk.nvidia.com/default/topic/1032862/jetson-tx2/a-guide-to-solve-usb-serial-driver-problems-on-tx2/>

Ce tutoriel permettra de mettre à jour le kernel permettant au convertisseur d'être reconnu et assigné à un port « /dev ».

### 3.5.3.8.5 Intégrer Manette Xbox

#### 3.5.3.8.5.1 Spécifications

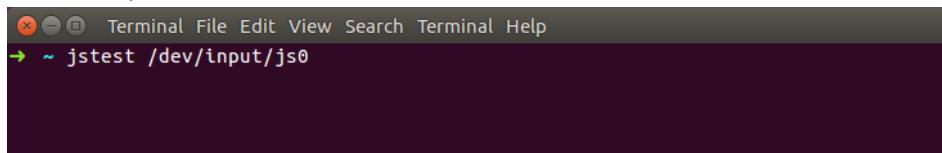


Utile pour contrôler le robot. Joystick de gauche : translations. Joystick de droite : rotations.  
Gachettes R2 et L2 hauteur du plateau (appui sur les deux en même temps pour l'initialiser).

Figure 114 Manette XBOX360

### 3.5.3.8.5.2 Développement

Tout d'abord, afin de pouvoir utiliser la manette XBox one en Bluetooth avec la carte mère Jetson il faut installer xpadneo. Ensuite vous pourrez appairer votre manette avec votre machine. Pour vérifier que cela fonctionne tapez dans un terminal :



A screenshot of a terminal window. The title bar says "Terminal". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The command line shows a green arrow icon followed by the text "jstest /dev/input/js0". The terminal window has a dark background and a light gray border.

Ps : Js0 étant le nom de votre manette, ce paramètre peut changer, pour vérifier tapez la commande ls dev/input/js\*.

Vous pourrez observer à l'écran l'état des joysticks et boutons.

Pour lire les données de la manette dans ROS, on utilise un node du package joy appelé « joy\_node.py ». A la manière de jstest il rend disponible à la lecture les données de la manette sur le topic /joy. Il est donc indispensable pour exploiter la manette.

On retrouve ce node dans le launch du robot s'appelant « xBoxController.launch » qui traduit les mouvements des joysticks en consignes de vitesses pour le robot (déplacement et élévation du plateau). Pour cela il appelle le node « joy\_node.py » ainsi que le node « controller.py » (Figure 116 node controller.py du package heron).

Package : heron – Node : controller.py

```
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Joy
from std_msgs.msg import Float32
from math import pi, atan2, sqrt
import os

import sys
sys.path.append(str(os.getcwd()[:-4])+"catkin_ws/src/heron_software/src/nodes/winch")
import winch_specs
counter = 0

def callback(data):
    twist = Twist()
    twist.linear.x = data.axes[1] * 0.40                      # Indice 1 Joystick gauche axe vertical
    twist.linear.y = data.axes[0] * 0.40                      # Indice 0 Joystick gauche axe horizontal
    twist.angular.z = data.axes[3] * (pi/2)                   # Indice 3 Joystick droit axe horizontal

    cmd_winch = Float32()
    global counter

    # you need to press both LT and RT from Xbox controller to initialize
    if(counter == 0):
        if ((data.axes[2] == 0) and (data.axes[5]== 0)):

            counter=0
        elif ((data.axes[2] == -1) and (data.axes[5]== -1)) :
            counter+=1
    else:
        cmd_winch.data = ((data.axes[2] - data.axes[5])* (winch_specs.MAXSPEED_M_S/2) )

    pub.publish(twist)
    pubWinch.publish(cmd_winch)

# Intializes everything
def start():
    # starts the node
    rospy.init_node('controller')

    # publishing to "Heron/cmd_vel" to control Heron
    global pub
    global pubWinch
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    pubWinch = rospy.Publisher('cmd_vel_winch',Float32, queue_size = 1)

    # subscribe to joystick inputs on topic "joy"
    rospy.Subscriber("joy", Joy, callback)
    rospy.spin()

if __name__ == '__main__':
    start()
```

Figure 115 node controller.py du package heron

xBoxController.launch est lui-même utilisé dans le launch « heronController.launch » qui permet de contrôler le robot à la manette. Ce dernier en plus d'inclure xBoxController inclut aussi le node « drive » et le node « winch.py ». Il va permettre notamment de contrôler les déplacements du robot avec « drive » ainsi que la hauteur du plateau avec « winch.py ».

#### 3.5.3.8.5.2.1 *XBoxController.launch*

Ce qui va suivre est le détail du fichier « xBoxController.launch ». Avec les nodes et les launches, puis sur quels **topics** ceux-ci publient avec les **types de message** qui transitent sur ces **topics**.

- **/joy**, message de type `sensor_msgs/Joy`
  - `std_msgs/Header` header
    - `uint32 seq`
    - `time stamp`
    - `string frame_id`
    - `float32[] axes`
    - `int32[] buttons`

#### 3.5.3.8.6 *Fixation port USB pour les périphériques USB*

Pour communiquer avec les capteurs et les drivers via nos programmes, il faut indiquer le port sur lequel se trouve le périphérique. La plupart du temps, ceux-ci sont du type “/dev/ttyUSB” ou “/dev/ttyACM”. Seulement, lorsque l'on connecte un périphérique, celui-ci se connecte à un port de manière aléatoire. Ce qui fait que nous devons à chaque fois, changer dans nos programmes le nom

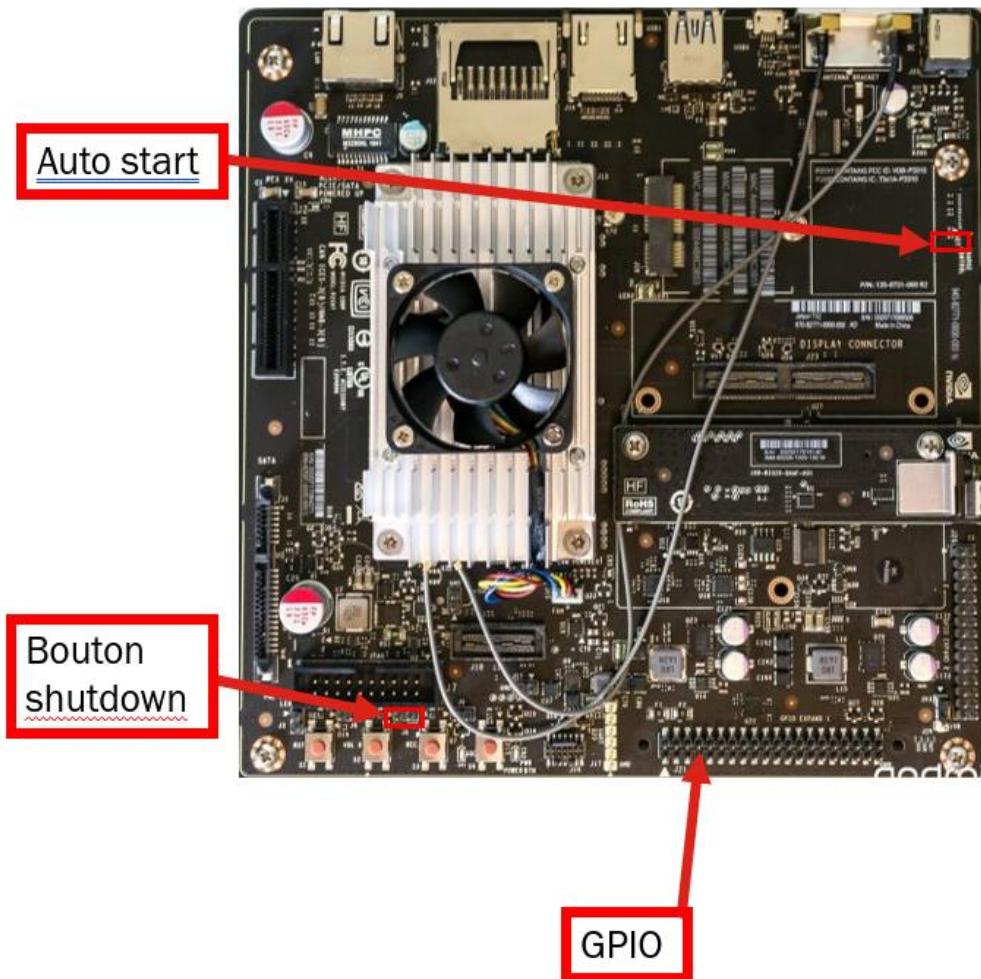


Figure 116 Placement sur la Jetson des fonctionnalités

du port. Nous avons donc dû réfléchir à un moyen d'éviter cela. Il a donc fallu écrire un fichier de règles, pour que la Jetson reconnaisse le périphérique et l'associe à un port que nous connaitrons. Le fichier de règles sera “10-heron.rules” et se trouvera dans le dossier “/lib/udev/rules.d/” de ubuntu.

Les ports seront les suivants :

- IMU : “/dev imu”
- Driver Moteur Roboteq front : “/dev/roboteq\_front” SEULEMENT BRANCHER SUR LE HUB USB PORT 10
- Driver Moteur Roboteq back : “/dev/roboteq\_back” SEULEMENT BRANCHER SUR LE HUB USB PORT 3
- Driver Moteur glissière : “/dev/winch”
- Capteur batterie : “/dev/battery”
- Lidar : “/dev/lidar”

ATTR sont les attributs des périphériques, ce qui va permettre de différencier les périphériques

SYMLINK fait le lien entre le port sur lequel est branché le périphérique et le nom que l'on veut donner à ce port

KERNELS est relié au port USB physique

## 4 Serveur central et communications

Le système possède donc un PC Central. Ce PC central gère les communications entre chaque équipement. Chaque équipement n'a donc besoin de connaître qu'une adresse IP, celle du PC Central (qui comme évoqué précédemment, a été fixé sur le réseau YncreaLab à 10.224.0.52). Avec celle-ci, le système devient alors capable de communiquer avec l'intégralité des systèmes présents.

Les communications entre chaque équipement et le PC Central sont bidirectionnels. Des conventions de communications ont été établies afin de faciliter la compatibilité de chaque équipement.

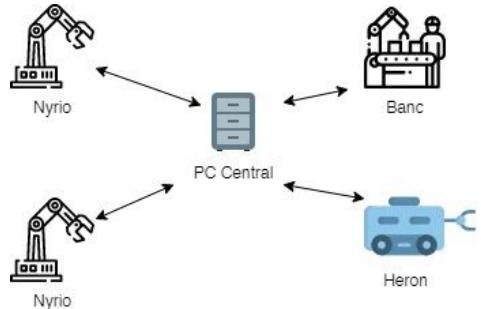


Figure 117 Schématisation des communications

Pour synthétiser, on compte parmi les points importants du développement les faits suivants :

- Les clients ont besoin de connaître uniquement l'IP du PC Central pour fonctionner sur le réseau
- L'IP du PC Central a été fixé sur les réseaux à utiliser (10.224.0.52 pour YncreaLab).
- Chaque équipement a accès (via une base de données) au statut de chaque équipement connecté au réseau (disponible, indisponible, déconnecté ...) ainsi qu'aux fonctions proposées par chaque équipement.

## 4.1 Fonctionnalités

Le PC Central, offre, à tout équipement, la possibilité de :

- Se connecter à se serveur via une phase d'initialisation (sécurisée, légère et performante)
- Lister les différentes fonctionnalités que ce dernier met à disposition de l'environnement (par exemple, pour héron, se déplacer)
- Passer des commandes aux différents équipement présent dans l'environnement
- Connaître l'état de chaque équipement dans l'environnement
- Se voir attribuer des commandes et rendre compte de l'état d'avancement de ces commandes.
- Diviser une tâche commande complexe nécessitant l'intervention de plusieurs équipements en plusieurs sous-commande, spécifique à un équipement à chaque fois.
- Synchroniser des équipements lors de tâches faisant intervenir plusieurs robots (lâcher d'une pièce sur un plateau, par exemple).

## 4.2 Spécifications globales

Pour répondre à ces exigences fonctionnelles, nous avons comparé une dizaine de solutions différentes, basées sur 5 technologies distinctes, parmi lesquelles on compte :

- Un broker MQTT
- Une base de données, gérée par requête SQL
- Une solution développée par nos soins en TCP / IP
- Une exploitation poussée de la solution PROFIBUS

Notre choix s'est orienté vers une solution combinant une base de données gérée par requêtes SQL et une solution développée par nos soins, en TCP / IP, visant à ajouter une couche de sécurité au système implémenté en SQL.

Le choix d'utiliser des requêtes SQL a été motivé par :

- La facilité d'implémentation d'un client SQL (léger, multiplateforme...)
- La simplicité d'utilisation du langage, sa documentation fournie, et la communauté réactive qu'on trouve sur les forums d'entre-aide liés à cette technologie.
- Le fait que cette technologie évitait (contrairement à un broker MQTT, par exemple) certaines confusions dans le développement.
- La large présence de cette solution dans le milieu industriel (idéal pour un démonstrateur de l'industrie du futur)
- Possibilité de conserver des fichiers de logs, ensuite exploitables en simulation

## *4.2.1 Spécifications serveur et tests*

### *4.2.1.1 Spécifications*

Le serveur a été développé en python (2.7) pour la phase d'initialisation (communication TCP/IP).

Les commandes sont gérées via une base de données, des bibliothèques python ont été développées et sont mises à disposition sur le git afin de monitorer la base de données et, éventuellement, interfaçer facilement un nouveau robot avec cette même base de données. (Un exemple d'exploitation de ces outils sera donné ultérieurement).

Le serveur, une fois opérationnel, offre un retour d'informations techniques qu'il est possible d'observer à travers le terminal. Dans ce lot d'information, on compte, entre autres :

- Les tentatives de connexions (IP et nom demandé)
- Les mots de passe demandés et les réponses obtenues
- Les identifiants attribués

Concernant la mise en service du serveur :

Lancer via un terminal (ou en configurant vos fichiers de boot) la commande :

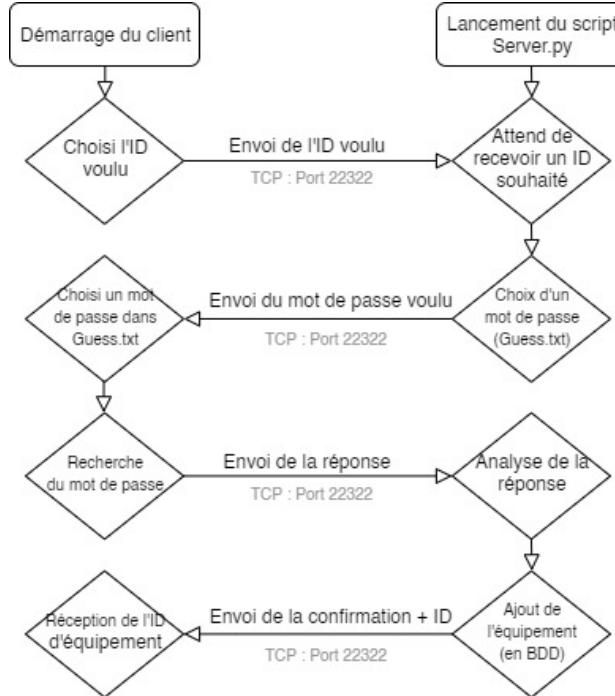
```
python server.py  
(Adaptez si besoin le chemin afin d'atteindre le script)
```

Pour les phases suivantes le serveur sera utilisé (en autre) comme serveur SQL (MySQL). Cependant, aucune opération n'est à effectuer pour cette partie car la configuration proposée permet un démarrage automatique de cette partie.

#### 4.2.1.2 Développement

##### 4.2.1.2.1 Initialisation

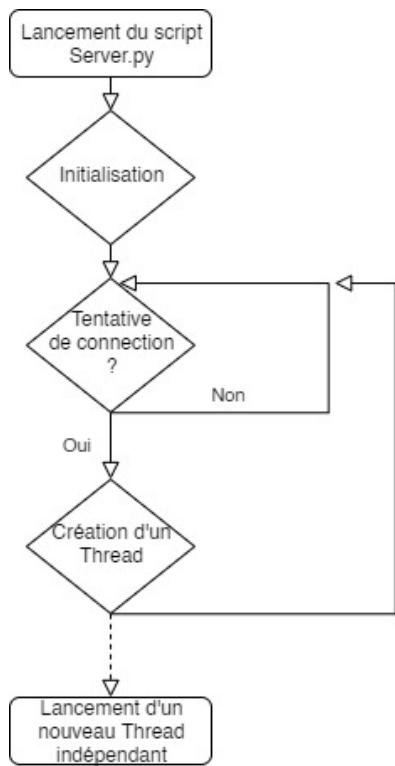
Le principe de connexion est le suivant :



1. Le client envoie au serveur l'ID qu'il souhaite obtenir
2. Le serveur lui demande alors une suite de caractère (mot de passe) contenu dans le fichier Guess.txt à un emplacement précis
3. Le client envoie le mot de passe alors demandé
4. Le serveur analyse le mot de passe. Puis ajoute le client en base de données et renvoie l'ID du client au client.

L'ID du client est composé de l'ID demandé suivi du caractère “ – ” et suivi d'un nombre.

Par exemple si un robot demande l'ID « heron » et qu'il est le 3<sup>ème</sup> à le faire, son ID sera : « heron-3 »



Une particularité du serveur est que de la programmation parallèle (Thread) a été intégrée à son programme.

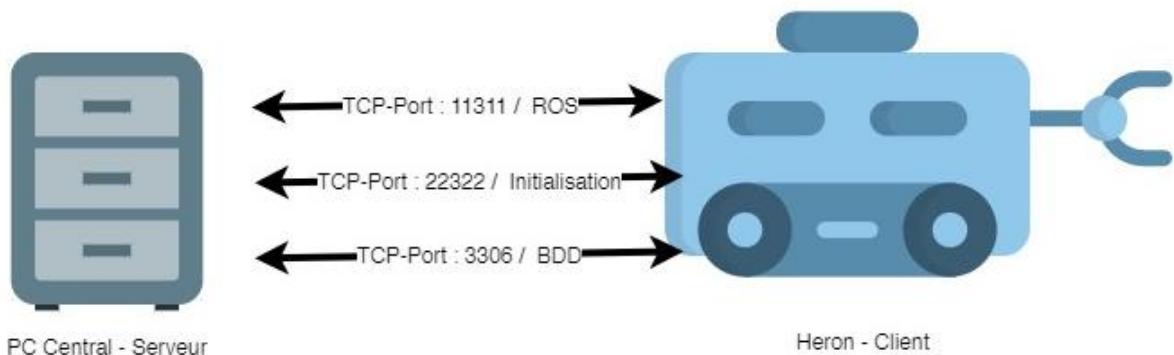
Le serveur écoute en boucle les tentatives de connexion TCP sur le port 22322.

A chaque tentative de connexion, le programme crée un thread dédié à cette connexion.

Cette programmation offre plusieurs avantages :

- ➔ Les tentatives de connexion ne sont pas bloquantes pour les tentatives suivantes. Un équipement qui arrête de répondre lors de la phase d'initialisation n'empêchera donc pas les autres équipements de se connecter.
- ➔ Les tentatives de connexion en parallèle sont également possibles. On peut imaginer avoir un jour un programme permettant de démarrer l'ensemble du démonstrateur.

Ce serveur permet de récupérer l'IP utilisé par l'équipement. L'idée pour améliorer la sécurité et la sûreté de l'environnement est en fait d'interdire (via des règles de pare-feu) toutes communications avec le PC Central à l'exception des communications TCP sur le port 22322. Une fois l'initialisation réussie, on autorisera cette IP à communiquer avec le PC Central sur les ports nécessaires (3306 pour le serveur MySQL et 11311 pour ROS).



Remarque : Le script du serveur utilise le fichier Guess.txt contenant l'ensemble des mots de passe. Assurez-vous d'avoir placer dans un même dossier le fichier Guess.txt et le dossier contenant le repo.

#### 4.2.1.2.2 Base de données

Une installation guidée de la base de données est décrite dans le ReadMe.

Dans un terminal linux, pour déployer le même système que celui décrit ultérieurement, vous devez exécuter les commandes suivantes :

```
mysql -u «l'ID souhaitez » -p
```

Ou, pour un accès « remote » (pensez à autoriser ce type de connexion)

```
mysql -u «l'ID souhaitez » -p -h « ip de l'host »
```

Après vous être identifié, dans cette instance MySQL, saisissez les requêtes suivantes :

```
CREATE DATABASE heronDatabase;
```

```
CREATE TABLE `AVAILABLE` (
    `availableLine` INT(11) NOT NULL AUTO_INCREMENT,
    `ID` TINYTEXT NOT NULL DEFAULT NULL,
    `Status` TINYTEXT NOT NULL DEFAULT NULL,
    `ComAvailable` TINYTEXT DEFAULT NULL,
    PRIMARY KEY (`availableLine`)
);
```

```
CREATE TABLE `COMMANDS` (
    `LineOrder` INT(10) NOT NULL AUTO_INCREMENT,
    `OrderID` TINYTEXT NOT NULL DEFAULT NULL,
    `Function` TINYTEXT NOT NULL DEFAULT NULL,
    `Target` TINYTEXT NOT NULL DEFAULT NULL,
    `Status` TINYTEXT NOT NULL DEFAULT NULL,
    `Source` TINYTEXT NOT NULL DEFAULT NULL,
    `ComOrder` TINYTEXT DEFAULT NULL,
    PRIMARY KEY (`LineOrder`)
);
```

```
CREATE TABLE `DICTIONNARY` (
    `DictionaryLine` INT(11) NOT NULL AUTO_INCREMENT,
    `ID` TINYTEXT NOT NULL DEFAULT NULL,
    `Function` TINYTEXT NOT NULL DEFAULT NULL,
    `ShortDescription` TINYTEXT NOT NULL DEFAULT NULL,
    `LongDescription` TINYTEXT DEFAULT NULL
);
```

La page suivante reprend les tables générées par ces requêtes

## AVAILABLE

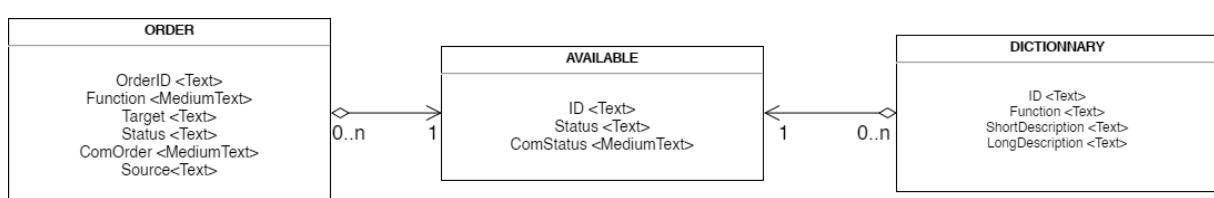
Champs	Type	Null	Key	Default	Extra
<b>availableLine</b>	int(11)	NO	PRI	NULL	auto_increment
<b>ID</b>	tinytext	NO		NULL	
<b>Status</b>	tinytext	NO		NULL	
<b>ComAvailable</b>	tinytext	YES		NULL	

## COMMANDS

Champs	Type	Null	Key	Default	Extra
<b>LineOrder</b>	int(10)	NO	PRI	NULL	auto_increment
<b>OrderID</b>	tinytext	NO		NULL	
<b>Function</b>	tinytext	NO		NULL	
<b>Target</b>	tinytext	NO		NULL	
<b>Status</b>	tinytext	NO		NULL	
<b>Source</b>	tinytext	NO		NULL	
<b>ComOrder</b>	text	YES		NULL	

## DICTIONNARY

Champs	Type	Null	Key	Default	Extra
<b>DictionaryLine</b>	int(11)	NO	PRI	NULL	auto_increment
<b>ID</b>	tinytext	NO		NULL	
<b>Function</b>	tinytext	NO		NULL	
<b>ShortDescription</b>	tinytext	NO		NULL	
<b>LongDescription</b>	tinytext	NO		NULL	



Description des tables :

**AVAILABLE** (*Cette table permet de savoir quel équipement est disponible à tout moment*) :

- **availableLine** - correspond au numéro de ligne de la commande
- **ID** – est l’ID de l’équipement
- **Status** – Status de la commande
- **ComAvailable** – Espace pour un éventuelle commentaire

**COMMANDS** (*Permet de lister toutes les commandes passées et en cours de préparation*) :

- **LineOrder** – similaire à availableLine
- **OrderID** – idem (pour ID)
- **Function** – nom de la fonction demandée
- **Target** – cible de la commande
- **Status** – statut de la commande
- **Source** - ID de l’équipement émetteur
- **ComOrder** - espace de commentaire

**DICTIONNARY** (*Liste toutes les fonctions disponibles pour chaque équipement*) :

- **DICTIONNARYLine** – même principe que pour les deux premières tables
- **ID** – ID de l’équipement considéré
- **Function** – Nom de la fonction proposé
- **ShortDescription** – Description de la syntaxe à adopter
- **LongDescription** – Description détaillée

Lors de la phase d'initialisation, l'équipement a rempli la table DICTIONNARY avec l'ensemble des fonctions qu'il propose de réaliser.

Une fois la phase d'initialisation terminée, l'ensemble des équipements vont suivre le principe décrit dans le schéma ci-contre. L'équipement va, en boucle :

1. Vérifier si une commande lui est destinée et est en attente
2. Si une commande lui est destinée et est en attente :
  - a. Passe son statut en indisponible
  - b. Accepte la commande
  - c. Réalise la commande
  - d. Passe le statut de la commande en « Done »
  - e. Passe son statut en disponible

On peut alors se poser quelques questions :

Peut-on passer une commande à un équipement indisponible ?

- Oui, mais il faudra attendre qu'il finisse de réaliser sa tâche en cours avant d'en réaliser une prochaine

Existe-t-il un ordre de priorité de réalisation des commandes ?

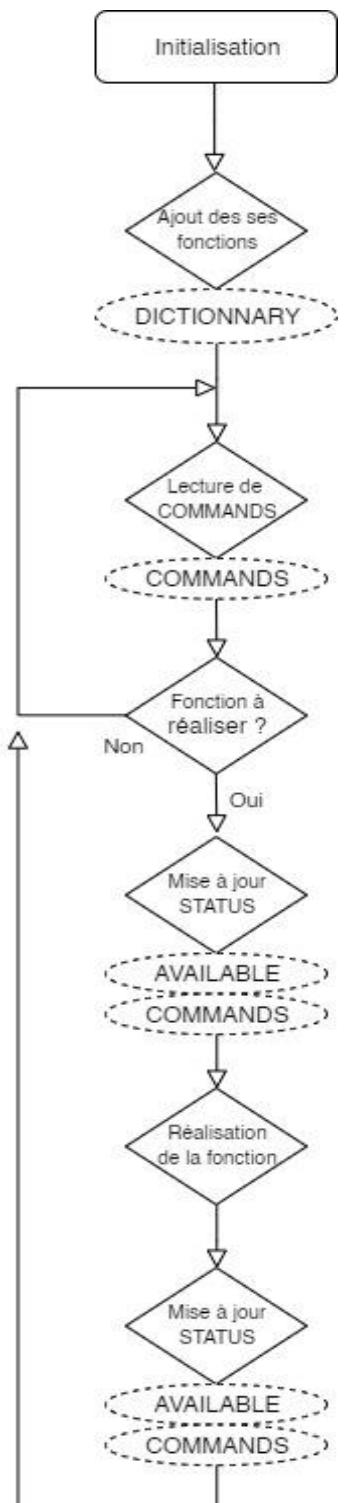
- Vous effectuez des requêtes SQL, aucune règle n'est prédéfinie, vous pouvez donc ajouter vos propres règles de priorité (par exemple basé sur les IDs de commandes (FIFO, FILO, utilisation de tag ...))

Si pour réaliser une commande, l'équipement A a besoin que l'équipement B réalise une tâche ?

- Il commencera par passer les « sous-commandes » aux équipements concernés. Une fois les « sous-commandes » réalisées, il termine sa tâche et déroule la fin du processus de manière classique. Ainsi les sous-processus ont été transparents pour l'équipement émetteur de la commande.

Un équipement peut-il se passer lui-même une commande ?

- Vous êtes certainement sur la mauvaise route, mais oui, vous pouvez.



#### 4.2.1.3 Test

Le tableau ci-dessous reprends certains tests de validation réalisés lors du projet :

Nom	Description	Date	Statut
P.O.C	Serveur capable de dérouler le processus de connexion.	3/10/2019	Ok
Déployable sur un autre réseau	Les variables spécifiques au réseau (principalement IP) sont facilement modifiables et la portabilité sur un autre réseau a été effectuée	10/12/2019	OK
Connexion Parallèle	Le serveur accepte des connexions en parallèle	5/11/2019	OK
Gestion des erreurs	En cas d'erreur (ayant pour origine le client), le serveur reste capable de gérer les connexions avec les autres client	5/11/2019	OK
Sécurité	Le serveur protège via un mot de passe robuste les fonctions vitales du système	6/10/2019	OK
Ré-exploitation des outils de développement	Les outils développés lors du développement du serveur sont réutilisables.	25/11/2019	OK
Connexion d'un équipement	Connexion d'un robot client. Adaptation du script et interaction.	10/12/2019	OK
Connexion de client multiple et interaction	Plusieurs robots sont connectés, et interagissent entre eux via la BDD (heron+ 2 niryo)	15/12/2019	Ok
Installation	En suivant le guide d'installation disponible sur le ReadMe du git, en partant d'une machine « nue », le système peut-être complètement réinstallé et est fonctionnel	15/10/2019	OK
Gestion des déconnexion	Le serveur détecte un équipement déconnecté, le retire de la base de données,		NOK

	et averti les utilisateurs ayant passé une commande auprès de cet équipement		
--	--	--	--

Des mesures de performances ont également été réalisée

Nom	Description	Résultat	Commentaire
Quantité de connexion acceptable	Combien de connexion peuvent être traité en 1 minutes	>>100	Les résultats sont variables mais toujours très largement suffisant pour connecter tous les équipements au serveur en moins d'une minute.
Débit du réseau Yncrea	Mesure du débit sur le réseau YncreaLab	>10 Mo /s	Le débit moyen obtenu, avec 2 000 mesures effectuées à différents moment (jour, heure, fréquentation des établissement...), était de 17 Mo/s.  Moins de 1% des mesures étaient inférieur à 10 Mo/s
Débit du serveur MySQL	Combien de temps est nécessaire au serveur MySQL pour traiter 100 requêtes ?	< 1 seconde	Il est peu probable qu'une du système puisse être imputé au serveur MySQL
Sécurité	Mot de passe trouvé par hasard (en connaissant le processus de connexion)	$1/10^{20}$	

#### 4.2.2 *Spécifications clients et tests*

##### 4.2.2.1 *Spécification*

Afin d'utiliser le serveur central, l'équipement doit commencer par effectuer un processus d'initialisation. Il doit, pour cela, posséder le fichier Guess.txt, ce fichier contient l'ensemble des mots de passe pouvant être demandé par le serveur lors de cette phase.

Un client général est disponible sur le git du projet, le processus a été décrit dans la partie précédente.

#### 4.2.2.2 Développement

##### 4.2.2.2.1 Initialisation

Le code ci-dessous correspond au squelette général du client proposé sur le git du projet :

```
6
7 import socket
8 import argparse
9
10 parser = argparse.ArgumentParser()
11 parser.add_argument("name")
12 args = parser.parse_args()
13 print(args.name)
14
15
16 def RecovPass():#Récupère la phrase de passe à l'intérieur du fichier et stock la chaîne de caractère dans une variable (retournée)
17     f= open("../Guess.txt","r")
18     data=f.read()
19     f.close()
20     return(data)
21
22 def Main():
23
24     Server = '10.224.0.52' #IP du Server
25     passwd=RecovPass() #Récupération de la Phrase de passe
26
27
28     port = 22322
29     s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)#Préparation de la socket
30     s.connect((Server,port))#Connection
31
32     message= (args.name)#nom demandé
33     s.send(message.encode('ascii'))#On envoi la demande
34     data = int(s.recv(1024).decode("ascii"))#On récupère le mot de passe demandé
35     s.send(passwd[data:data+24].encode('ascii'))#Et on l'envoi
36
37     data2 = s.recv(1024).decode('ascii')#On récupère le message confirmant l'autorisation de communication
38     #print(data2)
39     s.close()#Fermeture de la connection
40
41 if __name__ == '__main__':
42     Main()
```

Figure 119 Squelette général client BDD

On observe entre les lignes 22 et 33 l'initialisation de la connexion et l'envoi de l'ID demandé. (L'ID est ici passé en argument lors du lancement du script)

La ligne 34 correspond à la réception du mot de passe demandé et la ligne 35 à la réponse. Enfin, la ligne 37 correspond à la réception de l'ID.

##### 4.2.2.2.2 Utilisation

Afin de faciliter l'adaptation du squelette du client général à un robot spécifique, des outils tels que SELECTTools.py sont mis à disposition sur le git. Ici, par exemple la fonction SELECT permet de stocker dans une Liste l'ensemble d'une table (avec la possibilité de spécifier une condition)

La fonction FunctionOrderer(ID) permet quant à elle de récupérer uniquement les commandes destinées à un ID spécifique.

```

from InsertTools import CountID
import mysql.connector
import sys

def SELECT(table,condition=""):#Retourne sous la forme d'une liste les résultat une requete sql SELECT

    if(condition!=""):
        condition="where "+condition

    try:
        connection = mysql.connector.connect(host='10.224.0.52',database='heronDatabase',user='robot',
        sql_select_Query = "SELECT * FROM "+table+" "+condition+""
        print(sql_select_Query)
        #print(sql_select_Query)
        cursor = connection.cursor()
        cursor.execute(sql_select_Query)
        records = cursor.fetchall()
        #{print("la longueur : ",len(records))
        if (connection.is_connected()):
            connection.close()
            cursor.close()
        return records

    except:
        print("Erreur durant un SELECT ,",table)
        print(sys.exc_info())

def CommandsFor(ID):#Retourne LineOrder-OrderID-Function-Target-Status-Source-Com destiné à L'ID spéc:
    L=[]
    for elt in SELECT("COMMANDS"):
        if(elt[3]==ID):
            L.append(elt)
    return(L)

def FunctionOrdered(ID):
    return(SELECT("COMMANDS","TARGET='"+ID+"'"))

```

Figure 120 Exemple d'outils développés pour gérer la base de données

Pour montrer la BDD, on peut, par exemple utiliser :

```

Python
Import SQLTools
SQLTools.DisplayCOMMANDS()

```

Des scripts sont également mis à disposition, directement exécutable, ils font appel à ce genre de librairie. Par exemple ici pour un monitoring de la table associé au programme.

 <a href="#">disAv.py</a>	Simple script for display
 <a href="#">disCOM.py</a>	Simple script displaying COMMANDS
 <a href="#">disDIC.py</a>	simple script display DICTIONNARY

#### 4.2.2.2.3 Installation

## Installation

### Côté Serveur

#### Prérequis

L'ensemble des programmes a été rédigé en supposant que :

- L'IP du server est 10.224.0.53
- Son port 22322 est libre (connection TCP)
- Les scripts devront être adaptés si ces conditions ne sont pas vérifiées.

Prérequis	Source
MySQL	<a href="https://gist.github.com/arsho/ed4c2488714d9ad5629d4abc02a62eaa">https://gist.github.com/arsho/ed4c2488714d9ad5629d4abc02a62eaa</a>
MySQL-Connector-Python	<a href="https://github.com/mysql/mysql-connector-python">https://github.com/mysql/mysql-connector-python</a>

#### Création de la base de données heronDatabase

- Première solution

On ouvre une instance MySQL (on peut sélectionner un autre utilisateur que root)

```
mysql -u root -p
```

Puis dans cette instance

```
CREATE DATABASE heronDatabase;
```

- Seconde solution

Directement depuis le terminal

```
mysql -u utilisateur -password 'votreMotDePasse' -e 'CREATE DATABASE heronDatabase;'
```

Le processus d'installation du client, tout comme celui serveur est détaillé sur le ReadMe du git à l'adresse suivante : <https://github.com/yncreahdf-robotics/heron-interfaces/>

#### 4.2.2.2.4 Tutoriel créer un client

Vous trouverez ci-dessous un exemple commenté de « client » que nous avons développé pour un robot Niryo.

Prenez le temps de lire ce programme et de prêter attention aux commentaires qui vous indiqueront comment adapter ce programme à votre propre client et ainsi lui permettre de communiquer avec le PC Central.

N'oubliez pas d'ajouter le fichier « Guess.txt » dans le dossier de votre client python.

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python

import socket
import SQLTools
import SELECTTools
import InsertTools
import NiryoCommands

ListFunction=[['DICTIONNARY','DICTIONNARY',''],
['POSITION','POSITION(nPOS)',"Put the arm in a Position"],
['CLAMP','CLAMP(status)',"open with status= « OPEN », or CLOSE"]]

def RecovPass():#Récupération du mot de passe
    f= open("../Guess.txt","r")
    data=f.read()
    f.close()
    return(data)

def InitConnection(name):#Réalisation de la phase d'initialisation

    Server = '10.224.0.52' #IP du Server
    passwd=RecovPass() #Récupération de la Phrase de passe
    port = 22322

    s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect((Server,port))#Connection
    s.send(name.encode('ascii'))#On envoie la demande
    data = int(s.recv(1024).decode("ascii"))
    s.send(passwd[data:data+24].encode('ascii'))#Et on l'envoie

    data2 = s.recv(1024).decode('ascii')
    s.close()
    return data2

ID=InitConnection("Niryo")

print("-----")

while(True):#On boucle
```

```

commands=SELECTTools.CommandsFor (ID) #On récupère toutes les commandes
adressées
for elt in commands: #Pour chaque commande

    LineOrder, OrderID, Function, Target, Status, Source, ComOrder=elt[0],elt[1]
    ,elt[2],elt[3],elt[4],elt[5],elt[6] #On récupère les valeurs de
    chaques tables

    if(Status=='waiting'):#Si la commande est en attente

        if(Function=='DICTIONNARY'):#S'il s'agit de DICTIONNARY
            print("DICTIONNARY")
            #On accepte la commande
            InsertTools.ChangeCOMMANDS ("accepted",LineOrder)
            #On la réalise
            InsertTools.InsertDICTIONNARY(ListFunction, ID)
            #On passe son statut en terminé
            InsertTools.ChangeCOMMANDS ("done",LineOrder)

        if('POSITION' in Function):#Idem pour POSITION

            InsertTools.ChangeCOMMANDS ("accepted",LineOrder)
            #On va éliminer tout les caractères qui
            #ne sont arguments de la fonction
            nbPos=Function.replace('POSITION(,)')
            nbPos=int(nbPos[ : -1])
            NiryoCommands.Position(nbPos)
            InsertTools.ChangeCOMMANDS ("done",LineOrder)

        if('CLAMP' in Function):#Et CLAMP

            InsertTools.ChangeCOMMANDS ("accepted",LineOrder)
            #On cherche à savoir si l'argument est OPEN
            if('OPEN' in Function) :
                NiryoCommands.Clamp('OPEN')
            else :
                NiryoCommands.Clamp('CLOSE')
            PubForHeron.Circle(velocity_publisher)
            InsertTools.ChangeCOMMANDS ("done",LineOrder)

```

Nom	Description	Date	Statu
Connection au serveur	Le client général donne la possibilité de se connecter au serveur	26/09/2019	<u>ok</u>
Adaptation sur un robot Niryo	Le squelette général a été adapté et permet d'interfacer le robot avec le serveur	15/10/2019	<u>ok</u>
Client Multiple	Un système contenant le serveur + le client héron + 2 clients niryo est fonctionnel et permettent un interfaçage complet	14/12/2019	<u>ok</u>

## 5 Gestion de projet

Un projet de tel ampleur nécessite une certaine organisation, autant pour s'assurer de la bonne avancée du projet du point de vue technique, mais également au niveau humain.

### 5.1 Composition de l'équipe

L'ensemble des membres ayant travaillé sur ce projet sont tous issus du module de robotique et réalisent donc ce projet dans le cadre de leur année de M2.

Bien que tous les membres n'appartiennent qu'à un seul module, nous pouvons compter sur les profils et les personnalités différentes de chaque personne pour tirer profit de chacun et mener à bien ce projet. Voici donc la composition de l'équipe de robotique, année 2019-2020.



Robin GHYS  
**Chef de Projet**



Jean-Alexis HERMEL



Kilian FELLET



Léo DEDEINE



Thomas FREULON



Jean-Florian TASSART



Quentin CONSIGNY



Gilles HUBERT



Charles GOUDAERT



Sylvain LOUVET

### 5.2 Organisation interne et gestion de projet

Ce projet s'est déroulé sur une période de 6 mois. Un projet d'une telle ampleur nécessite donc un minimum d'organisation pour être certain de mener à bien celui-ci. Nous avons donc élu en premier temps, et cela l'unanimité, un chef de projet qui aurait la charge de maintenir cette vision globale du projet tout au long de celui-ci. Il avait également la responsabilité de répartir le travail, s'assurer de la bonne humeur au sein du groupe et de l'épanouissement de chaque individu sur son lieu de travail.

Mais l'objectif principal de celui-ci était tout de même de s'assurer du respect des objectifs fixés, ainsi que la livraison des livrables aux dates fournies.

Pour cela, nous avons utilisé différents outils pour maintenir cette cohésion au sein de l'équipe, pour se tenir au courant en temps réel de l'avancée du projet et bien d'autres encore.

### 5.2.1 Planning

Comme cité précédemment, notre projet s'est étalé sur une période de 6 mois au total, divisé en 3 sprints. A chaque fin de sprint, un grand pas en avant devait être franchi au niveau de l'avancée du projet qui se caractérisait donc par l'ajout de nouvelles fonctionnalités.

Le premier outil créé et mis à jour régulièrement par l'ensemble de l'équipe fut le planning. Celui-ci avait pour but de lister toutes les fonctionnalités attendues en fin de sprint avec une date butoir à respecter.

#### Etablissement des sprints : Point de vue technique

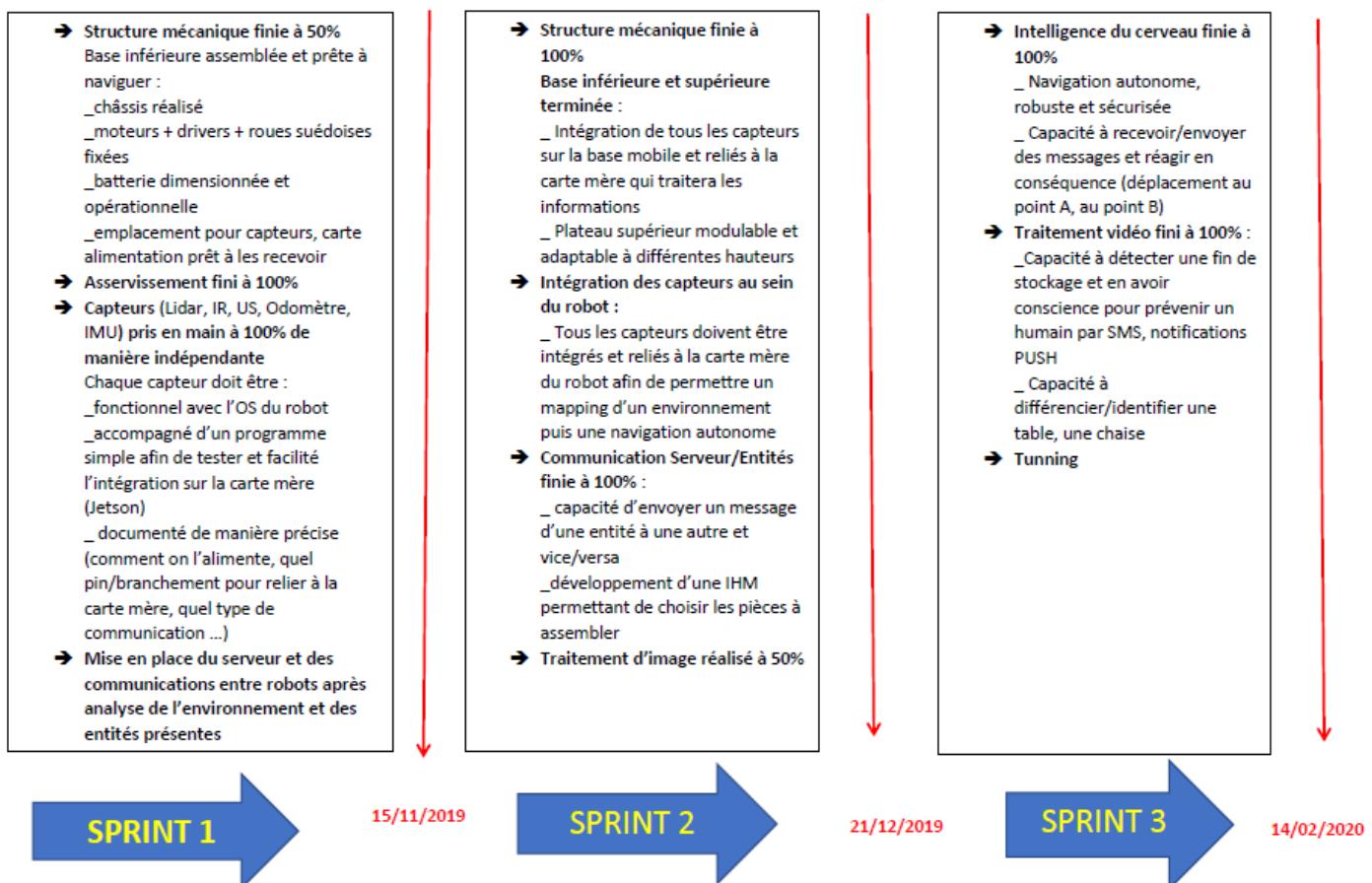


Figure 121 Exemple de planification attendu pour les 3 sprints

L'objectif de ce document était de permettre à l'ensemble du groupe d'être facilement mis au courant de l'ensemble des fonctionnalités attendues à chaque fin de sprint. De plus, le fait d'avoir réalisé ce type de document en début de projet nous a permis d'anticiper l'arrivée des sprints suivants afin de les démarrer avec une bonne idée des nouveautés attendues. Ce document a pour but d'exposer ces fonctionnalités de manière brève, sans rentrer dans les détails.

### 5.2.2 Gestion Prévisionnelle des Emplois et des Compétences (GPEC)

Afin de s'assurer que chaque membre du projet s'épanouisse dans son travail sur le projet, nous réalisions à chaque début de sprint un document intitulé GPEC (Gestion Prévisionnelle des Emplois et des Compétences).

Ce document fut indispensable pour réaliser la constitution des différents « pôles » dans lesquels chaque membre travaillerait pendant toute une durée de sprint. Une fois les différents pôles identifiés comme nécessaires au bon déroulement d'un sprint, il fallait à présent savoir quelles personnes constituaient ces pôles aux aspects différents (Ex : Pôle Mécanique, Electronique, Serveur ...). Pour cela, chaque membre estimait ses connaissances et son envie à travailler sur tel ou tel module par une note sur 5. Une fois que l'ensemble du groupe avait partagé ces notes, nous discutions ensemble pour répartir chaque membre dans les différents pôles en s'assurant de l'accord de chacun .

	Méca		Electronique		soft embarqué		soft applicatif	
	capacités	envies	capacités	envies	capacités	envies	capacités	envies
Gillou	2	1	3	3	3	3	3	3
Jeanflo	1	1	2	1	2	2	3	5
Kilian	3	3	2	3	3	3	2	2
Charles	4	4	3	3	3	4	2	2
Léo	3	4	2	3	2	1	2	2
JA	4	4	2	2	3	2	3	4
Thomas	2	4	2	4	1	4	2	4
Quentin	2	1	3	2	3	3	3	2
Sylvain	3	3	2	2	3	3	3	5
Robin	2	1	2	2	3	5	2	2

Figure 122 Exemple de document GPEC réalisé au début du 1er sprint

Cette discussion et répartition d'équipe était très importante à chaque début de sprint. Celle-ci nous a permis de maintenir une bonne ambiance et un bon équilibre au sein du groupe car chacun pouvait travailler dans un domaine qui l'intéressait. Chaque membre connaissait sa place et le rôle qu'il jouait pour faire avancer le projet.

### 5.2.3 Cycle en V et Planning de Production Documentaire (PPD)

Un choix important de notre part fut de suivre un modèle de cycle en V quant à la réalisation du projet.

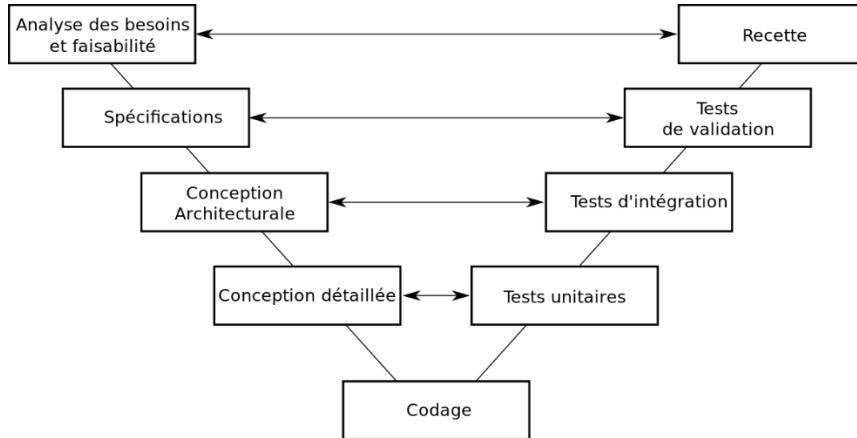


Figure 123 Schématisation du cycle en V

Ce choix d'organisation de projet nous a forcé à réfléchir de manière sérieuse à l'élaboration du projet afin d'éviter de se lancer directement dans la partie développement sans avoir pris le temps de réfléchir à la manière de le réaliser ni aux tests à mener pour vérifier la fiabilité des produits rendus.

Ce cycle en V nous a donc forcé à nous asseoir autour d'une table tous ensemble pour réfléchir aux documents que nous allions devoir réaliser avant de se mettre à développer quoique ce soit. Pour cela, nous avons décidé d'utiliser un Planning de Production Documentaire qui allait rythmer l'ensemble de notre sprint et vérifier que nous respections les deadlines imposées.

Quoi ?	Qui ?	Quand ?	Avancement
Cahier des charges	Robin	20/09/2019	100%
Etude de faisabilité + Rédaction plan de qualif	Robin	24/09/2019	100%
Spécification Globale + Rédaction Scénario Test	Robin	26/09/2019	100%
Spécification des modules et plan de valid (divisé en 5 parties)			
Module Mécanique	Jean-Alexis	24/09/2019	100%
Module Asservissement + Alimentation	Charles	24/09/2019	100%
Module Capteurs	Kilian	24/09/2019	100%
Module Intelligence Cerveau (Jetson TX2)	Sylvain	24/09/2019	100%
Module Serveur/Communication	Quentin	21/10/2019	100%
Spécification Interfaces			
Module Mécanique + Alimentation + Capteurs	J-A + Thomas	10/10/2019	80%
Module Alimentation + Asservissement	Charles + Thomas	10/10/2019	70%
Module Intelligence + Capteurs	Sylvain + Kilian	14/10/2019	90%
Module Serveur + Intelligence	Quentin + Sylvain	21/10/2019	80%
Spécification de conception + Rédaction tests unitaires			
Module Mécanique	Jean-Alexis	01-oct	80%
Module Asservissement + Alimentation	Charles	02-oct	80%
Module Capteurs	Kilian	03-oct	70%
Module Intelligence Cerveau (Jetson TX2)	Sylvain	04-oct	70%
Module Serveur/Communication	Quentin	05-oct	90%

Figure 124 Exemple de Planning de Production Documentaire

Comme on peut le constater, ce type de document avait ainsi pour but de responsabiliser les membres du projet à effectuer un travail de documentation. Cela permettait également de visualiser de manière très rapide si un pôle avait un peu plus de retard par rapport à un autre et donc d'identifier et anticiper rapidement les possibles retards. Pour éviter cela, nous n'hésitions pas à venir modifier les équipes (pour quelques jours uniquement) pour rattraper le retard créé et garder une bonne dynamique.

### 5.2.4 Graphe de performance

Le graphe de performances était un outil indispensable pour s'assurer du bon avancement ou non du projet de manière générale mais également de chaque pôle.

En début de projet et une fois les pôles formés, chacun d'entre eux donnèrent une estimation de la progression de l'activité de leur pôle sur l'intégralité des 3 sprints. Ces estimations permirent de réaliser une première courbe de progression pour chaque pôle (courbe en pointillé sur la Figure 5 ci-dessous).

Puis, de manière quotidienne, chaque pôle constatait sa réelle progression par rapport au projet et venait inscrire celle-ci dans le document afin de venir tracer une seconde courbe (la courbe pleine sur la Figure 5 ci-dessous). Ainsi, il était aisément de comparer les courbes estimées (et synonymes de succès) et les courbes réelles. Cela nous permettait de visualiser les pôles en avance sur leur estimations, mais surtout les pôles qui affichaient du retard pour venir leur apporter un soutien le plus rapidement possible.

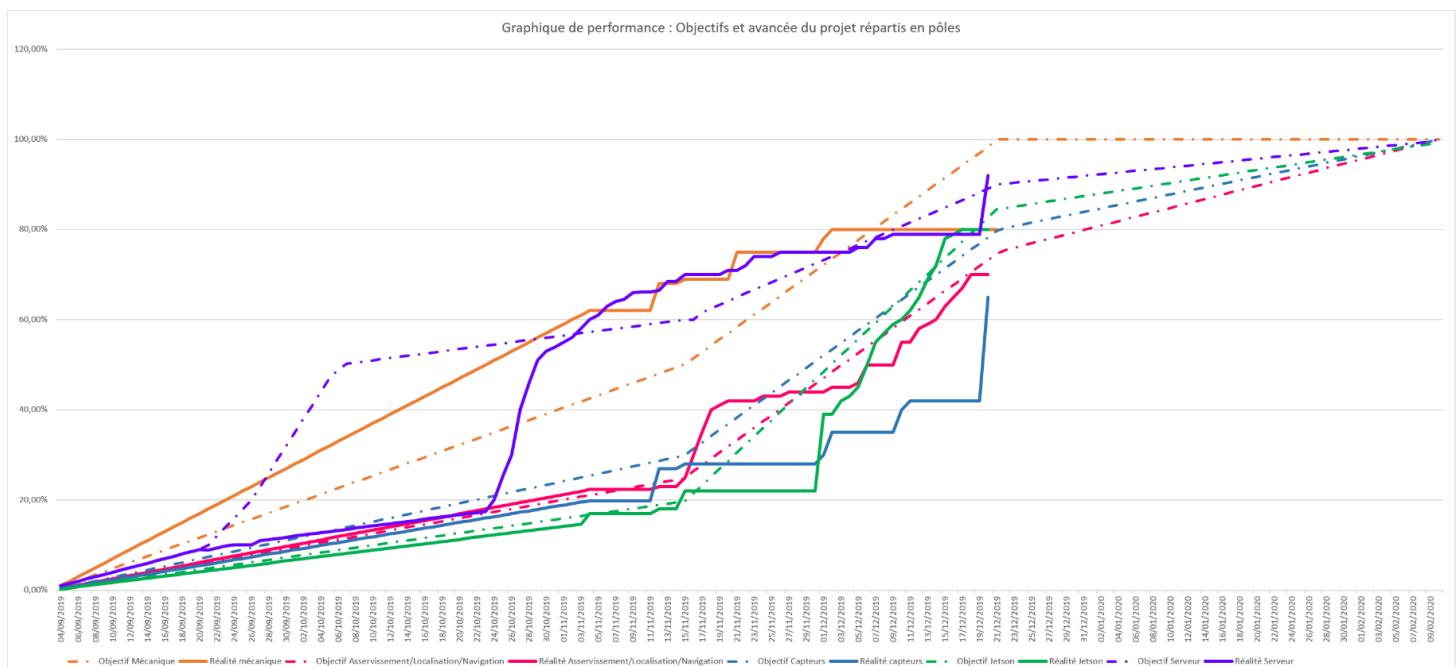


Figure 125 : Exemple de Graphe de performance réalisé en fin de 2nd sprint

### 5.2.5 Animation du projet (réunion d'équipe, réunion 5 min)

Afin d'organiser et assurer un suivi quotidien de l'avancée du projet, nous avions convenu de nous réunir chaque jour de projet pendant une durée de 15-20 min avant de démarrer la journée. Le but de cet échange étant que chacun expose à l'ensemble du groupe son avancée dans le projet afin de mettre en évidence les points succès et les points bloquants. Cet échange permet ainsi à l'ensemble du groupe d'être tenu au courant de l'avancée générale du projet et d'avoir une vision globale sur l'atteinte ou non des objectifs fixés. Afin que cette réunion ne soit pas trop chronophage tout en restant efficace, chaque membre avait pour mission de venir compléter un document Excel avant chaque réunion pour garder une trace des activités menées et de pouvoir résumer rapidement son activité devant l'ensemble du groupe.

Date	Qu'ai-je fait hier ?	Qu'est-ce qui a fonctionné ?	Qu'est-ce qui n'a pas fonctionné ?	Que vais-je faire aujourd'hui ?
23/09/2019	rédaction des tests, et fonctionnalités	tests terminés et specs terminées		commencer solidworks
24/09/2019	chassis et assemblage	decision des dimensions OK	peu productif	continuer chassis et test batterie
30/09/2019	création chassis avec charles, test moniteur batterie+charge	bonne structure de base, bon assemblage, batterie OK		continuer solidworks, reflechir à la structure et assemblage, voir pour récupérer infos batterie sur ROS (Gilles)
01/10/2019	un peu de solidworks, reflechir capteurs	gilles recup valeurs batterie,avancée correcte SW	trop de réunions pour bosser dans le temps imparti	voir pour découper/assembler chassis HEI ?
07/10/2019	assemblage réel sur le robot + 2e fournée de commande	chassis qui prend forme	pas reçu de commande mardi	m'éclater à tout monter + monter l'élec avec charles
08/10/2019	-	-	-	-
14/10/2019	assemblage des moteurs+roues	tout prend forme, pas de mauvaise surprise	maintien moteur en plastique faute de CNC	continuer l'assemblage, voir pour début glissière /1ers tests sur roues
15/10/2019	assemblage complet robot + tests roues	assemblage parfait		câblage + aide asservissement

Figure 126 : Exemple de document Excel à remplir avant la réunion quotidienne de 15 min

### 5.2.6 Outils de partage, communication et échange

Au cours du projet, différents outils ont été utilisés pour permettre les échanges d'informations, de documents ou encore de programmes informatiques.



**Slack** : Plateforme permettant de créer un groupe et d'échanger facilement entre tous les membres du projet.



**Trello** : Outil permettant de tenir une sorte de TODO List pour chaque pôle (pense-bête électronique)



**Github** : Plateforme permettant l'échange et le dépôt de nos programmes informatiques



**One Drive** : Plateforme permettant de stocker tous les documents WORD, PDF ...