Rapport de projet de fin d'étude Ingénieur

Véhicule autonome pour cartographie

Département Informatique Électronique et Automatique

Affiliation :
Polytech Lille
Université de Lille
Bonduelle

Encadrants :
Xavier Redon
Thomas Vantroys
Alexendre Boe
Xavier Chenot

Etudiant :
Lina Mejbar

Septembre 2019 - Février 2020

Remerciements

Je tiens à remercier l'entreprise **Bonduelle** qui a proposé ce sujet très intéressant et qui m'a donnée la chance de travailler dessus.

Je remercie les encadrants école **Monsieur Xavier Redon**, **Monsieur Thomas Vantroys**, et **Monsieur Alexandre Boé** et les encadrants entreprise **Monsieur Xavier Chenot**, **Monsieur Erwan Niquet** pour leurs conseils et leur aide.

Ils étaient présents pendant toute la période du projet (septembre - février), pour suivre mon avancement et répondre à mes questions.

Je remercie tous **les professeurs de Polytech Lille** pour les connaissances acquises à ce jour.

Je tiens également à transmettre ma sympathie aux **membres du Fabricarium** de l'école pour leurs conseils, leur accueil ainsi que tout le matériel mis à ma disposition.

Enfin que tous ceux qui ont contribué à mener à bien ce projet trouvent ici l'expression de ma parfaite considération.

Introduction

Dans le cadre de notre dernière année en cycle ingénieur en spécialité : **Informatique, Microélectronique et Automatique,** nous avons l'occasion de réaliser un projet permettant de mettre en œuvre les connaissances vues au cours de nos études et d'en découvrir de nouvelles.

L'objectif de ce rapport est de présenter le travail que j'ai effectué entre septembre et février dans le cadre de mon projet de fin d'études. J'ai choisi de réaliser le projet numéro 9 qui a pour but de concevoir un véhicule autonome pour cartographie.

Afin de réaliser ce projet, j'ai eu à ma disposition un robot en kit de la marque **YAHBOOM** et plus particulièrement le modèle 6WD qui m'a été fourni par l'entreprise Bonduelle.

Dans ce rapport je vais dans un premier temps faire une **présentation du projet** incluant le contexte, la présentation du cahier des charges et la présentation du **robot**. Dans un deuxième temps, je vais détailler le travail que j'ai effectué sur la **cartographie**. Enfin, dans une troisième et dernière partie, j'expliquerai le travail effectué sur la page **web** de ce projet.

Table des matières

1 Contexte

1.1 Présentation de l'entreprise

Bonduelle est une entreprise fondée en 1853 par Louis Bonduelle, sa mission est d'être le référent mondial qui assure le bien-vivre par l'alimentation végétale. Le groupe Bonduelle est une multinationale qui fournit aux consommateurs plus de 100 pays des produits préservés grâce à des procédés naturels de conservation.

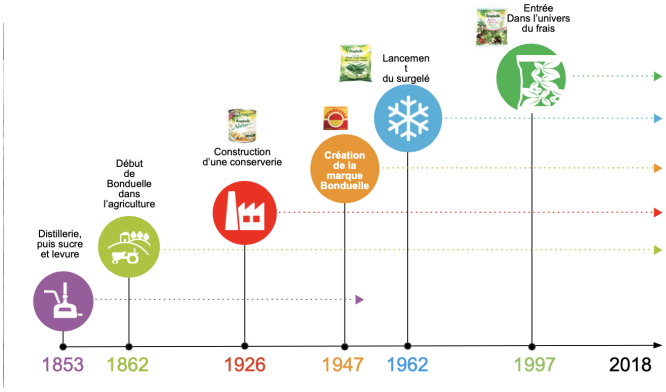


FIGURE 1 – Historique Bonduelle

Sur le schéma ci-dessus nous pouvons observer les grandes dates qui marquent l'histoire du groupe. C'est à partir de 1862 que le groupe s'est lancé dans l'alimentation végétale et c'est à partir de 1997 que le groupe est présent sur 3 modes de diffusions de ses produits, frais, surgelés et en conserve. Le groupe Bonduelle est une multinationale qui compte plus de 10 000 employés et qui réalise un chiffre d'affaires avoisinant les 2 milliards d'euros.

J'ai intégré la DSI au siège social du groupe Bonduelle qui se trouve à Villeneuve-d'Ascq.

La DSI est composée de différents pôles d'activité, dont « digital technology ». Ce pôle est constitué de 4 domaines :

- Data
- Devops
- Client Solutions
- Service Management

1.2 Problématiques

Le secteur de l'intelligence artificielle a été beaucoup marqué durant la dernière décennie par une évolution du nombre de recherches et de publications scientifiques. Son but est de concevoir des systèmes capables de reproduire le comportement humain et son raisonnement. S'il est vrai que ce domaine paraît très abstrait, il possède de nombreux avantages lorsqu'on sait comment l'utiliser. Le Retail, l'industrie ou encore la sécurité sont des secteurs qui deviennent de plus en plus performants lorsque l'intelligence artificielle est développée. L'objectif est de permettre une automatisation plus intelligente des tâches. Le machine Learning est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches statistiques pour donner aux ordinateurs la capacité "d'apprendre" à partir de données.

Ce projet est en collaboration avec l'entreprise **Bonduelle**. Aujourd'hui, leur problématique est la suivante : ils remarquent que dans les entrepôts de nombreux caristes peuvent se tromper lors du déplacement des palettes (lors de la récupération de la palette ou encore, lorsqu'ils déposent la palette dans le mauvais emplacement...). Ces erreurs, qui peuvent arriver même au plus performant peuvent être dues à la fatigue par exemple. De nos jours, l'automatisation des tâches dans les entrepôts est de plus en plus mise en avant. En effet cette dernière, permet d'éviter les erreurs des caristes, gagner du temps, et avoir une base de données des emplacements de toutes les palettes en temps réel. Le but n'est pas de remplacer les caristes, mais de libérer des énergies, et les placer sur d'autres tâches qui seront plus valorisantes. L'objectif est donc de concevoir un robot qui serait capable de cartographier en autonomie la pièce où il se trouve et par la suite de savoir se déplacer au bon endroit.

FIGURE 2 – Cariste dans un entrepôt – Robot dans un entrepôt

1.3 Définition du cahier des charges

Dans un contexte de développement de projets innovants, il m'a été demandé de concevoir un système autonome (véhicule mobile) capable de se déplacer dans une zone et de la cartographier. Pour ce faire, on a mis à ma disposition un Robot et un Lidar.

L'objectif du Lidar serait de permettre au robot d'en savoir plus sur son environnement et de lui permettre de se déplacer de façon autonome.

Le système devra ensuite pouvoir trouver le chemin adéquat pour se déplacer d'un point à un autre tout en évitant des obstacles fixes et/ou mobiles. Une fois que le robot se déplacera dans son environnement, il faudra le doter d'une intelligence pour qu'il différencie les obstacles mobiles des obstacles fixes. Il faudra donc utiliser du machine Learning et l'entraîner sur des réseaux de neurones.
Si on garde l'analogie du cariste et des palettes : le but final serait de placer le robot dans leur entrepôt, qu'il puisse cartographier l'environnement, et qu'il puisse déplacer une palettes de produit d'un point A à un point B, tout en évitant les obstacles.

Voici les grandes parties à réaliser lors de ce PFE, et mes choix techniques :

- **Création d'un site** - HTML, PHP ...
- **Contrôle du robot à distance** - WebSocket,
- **Cartographie** - ROS, OpenCV,
- **Déplacement intelligent du robot dans l'environnement** - Intelligence Artificielle

Mon rôle dans ce projet est une première itération qui consistera à faire des tests, étudier les solutions possibles, et enfin poser des bases fonctionnelles. Ce projet sera à terme complété par d'autres étudiants/ingénieurs.

2 Présentation des technologies et du robot

2.1 Raspberry Pi 4

En commençant le projet, j'avais à ma disposition un Raspberry Pi 3. N'ayant qu'un 1GB de RAM, j'étais assez limitée pour son utilisation. De nombreux outils sur lesquels je travaillais ne pouvaient pas fonctionner. En début novembre, j'ai eu l'occasion de passer à la Raspberry Pi 4GB. Elle est plus performante et fonctionne avec les outils nécessaires au projet, notamment ROS.

2.2 Lidar

Afin de pouvoir cartographier un environnement, il existe plusieurs technologies. Je vais travailler sur un LIDAR de la marque YDLIDAR . C'est l'acronyme de LIght Detection And Ranging, c'est-à-dire détection de la lumière et mesure de distance. Ce modèle fonctionne à 360 degrés et peut détecter un obstacle situé jusqu'à 10m de lui. Ci-dessous, une photo de ce dernier. Voici la Datasheet .

FIGURE 3 – Lidar utilisé pour le projet

2.2.1 Son fonctionnement

La détection et la télémétrie de la lumière (LiDAR) est une technologie similaire au radar, en utilisant le laser au lieu d'ondes radio.

Le principe LiDAR est assez facile à comprendre :
- Émission d'une impulsion laser sur une surface,
- Capture du laser réfléchi avec des capteurs,
- Mesure du temps parcouru par le laser

Calcul de la distance de la source :

$$Distance = \frac{vitesseLumiere * Temps}{2} \quad (1)$$

2.2.2 Code

L'entreprise YDLIDAR a déjà conçu une bibliothèque avec un premier exemple de code pour travailler sur ce lidar. Pour tout récupérer il faut cloner ce lien github : Github.

```
$ git clone  https://github.com/yangfuyuan/sdk
$ cd sdk
$ git checkout master
$ cd ..
$ mkdir build
$ cd build
$ cmake ../sdk
$ make
```

Pour lancer :

```
$ cd samples
$ ./ydlidar\_test
    $ Lidar[ ydlidar7 ] detected, whether to select current  radar(yes/no)? : yes
    $ Please enter the lidar serial baud rate : 2
    $ Please enter the lidar intensity : 1
```

Je me suis basée sur cet exemple (sdk/samples/main.cpp) que j'ai modifié pour que le programme ne demande plus les caractéristiques du Lidar. J'ai également modifié la fréquence de rotation du robot et rendu possible le stockage des données dans un fichier afin de les exploiter. Cela m'a permis au début du projet d'apprendre à utiliser le lidar avant de m'en servir avec ROS.

Ci-dessous la partie la plus intéressante du fichier main.cpp. C'est ici que l'on récupère les données du Lidar et que je stocke dans un fichier value.json. J'ai modifié la boucle principale pour que le lidar ne récupère qu'une fois les données par angle.

```
1  string const nomFichier("values.json");
2  ofstream monFlux(nomFichier.c_str());
3
4  if(laser.doProcessSimple(scan, hardError )){
5      for(int i =0; i < scan.ranges.size(); i++ ){
6                  float angle = (scan.config.min_angle + i*scan.config.ang_increment)*180/3.14;
7                  float dis = scan.ranges[i];
8                  ydlidar::console.message("\tAngle: %.2f et Distance %.2fm", angle,dis);
9                  monFlux << angle <<'/' << dis << endl;
10         }
11         monFlux.close();
12 } else {
13     ydlidar::console.warning("Failed to get Lidar Data");
14 }
```

2.3 Robot

On trouve de plus en plus de kits d'assemblage de robots programmables. C'est sur l'un d'entre eux, que je vais travailler dans le cadre de mon projet. Ce kit est commercialisé par la marque Yahboom. Ils vendent de nombreux modèles de robots : Char, Tank, des robots de 2 à 6 roues. Celui sur lequel je vais travailler, c'est ce kit .
Il contient : une caméra avant, 6 roues et 6 moteurs, une carte Arduino, la carte extenssive 6WD et une pile rechargeable.

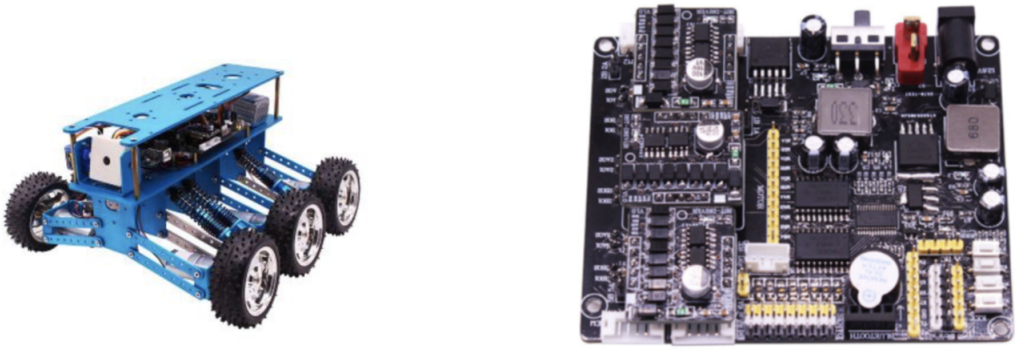


FIGURE 4 – Robot du projet

La carte d'extension 6WD est conçue avec trois prises d'entraînement de moteur et supporte jusqu'à trois modules d'entraînement de moteur. Chaque module peut conduire deux moteurs. Chaque moteur peut être contrôlé indépendamment donc les six moteurs peuvent produire six vitesses différentes en même temps. La carte d'extension 6WD prend en charge quatre contrôleurs populaires, y compris Raspberry Pi, Arduino..
Cf Datasheet de la carte

2.3.1 Déplacement du robot

Le robot possède 6 roues reliées à six moteurs à courant continu. Un MCC fonctionne à l'aide du principe de la PWM (Pulse Width Modulation), c'est de cette façon qu'il est possible d'agir sur la vitesse et le sens de rotation du moteur. Il faut utiliser les pins de sorties digitales de l'Arduino. On les commande avec une fonction de l'Arduino qui va prendre une valeur entre 0 et 255 :

```
analogWrite(pin,valeur);
```

Ce qui va définir la part de temps pendant laquelle le pin sera à l'état HAUT.

Pour permettre le déplacement du robot, il faut actionner les moteurs. Ci-dessous, une photo du robot :

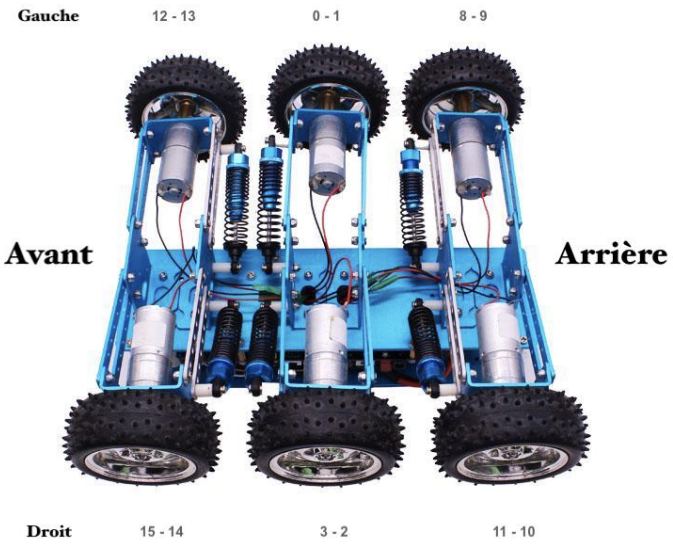


FIGURE 5 – Définition de l'orientation des roues

A côté de chaque de roue, il y a deux numéros. Prenons par exemple la roue avant droite, on peut lire " 15 - 14". Ces numéros représentent les pinMoteur, il y a deux valeurs : la première permet à la roue de tourner dans le sens anti-horaire (15) et la seconde dans le sens horaire (14).

2.3.2 Contrôler son déplacement : Encodeurs

Afin d'avoir plus d'informations sur le déplacement du robot, j'ai demandé des encodeurs, voici le modèle que l'on a mis à ma disposition.

Fonctionnement :

La plupart des encodeurs pour robots mobiles utilisent des capteurs optiques. L'idée est de placer un disque alternant des zones transparentes (ou tout simplement des trous) et opaques (le disque lui-même) entre un capteur de lumière et un émetteur de lumière. Ce dernier se place sur l'axe de rotation de la roue. La fréquence d'apparition des zones blanches et noires devant le capteur de lumière va indiquer la vitesse de rotation.

Le schéma suivant présente le principe de fonctionnement basique de l'encodeur :

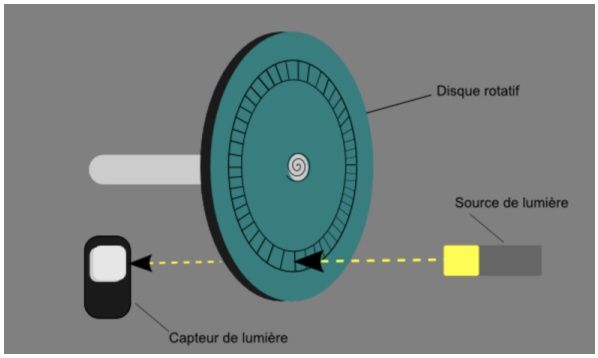


FIGURE 6 – Fonctionnement d'un encodeur

Modélisation et Impression 3D :

Initialement, le robot n'a pas été conçu pour supporter des encodeurs. En effet, il y a très peu de place entre l'axe du moteur et la roue. Pour gagner quelques centimètres, j'ai inversé les roues, mais ce n'est pas suffisant. J'ai donc modéliser les pièces moi-même pour permettre de positionner les encodeurs sur certaines roues (cf Figure 7 pour le rendu des pièces).

— **Modélisation 1** : Dans un premier temps, le but était de modéliser une pièce qui s'apparente à un "**Moyeu**" (Shaft hub), elle permettrait d'allonger l'axe du moteur. Ce n'était pas réalisable parce que le diamètre de l'axe moteur est de 4 mm. Il est impossible d'atteindre une telle précision avec une imprimante 3D. Même si c'était possible, la pièce n'aurait pas été suffisamment robuste pour supporter la rotation très rapide du moteur et d'entraîner avec elle la roue.
Conclusion : Ce n'est pas la bonne solution pour régler mon problème

— **Modélisation 2** : J'ai ensuite pensé à une autre façon d'installer l'encodeur sur le robot. Comme il n'y a pas de place pour le faire tenir, j'ai pensé à utiliser les suspensions du robot. En effet, il y a plusieurs de trous sur les suspensions bleues, je voulais en profiter pour accrocher le support de l'encodeur sur les roues avant du robot. D'un autre côté le disque fourni avec l'encodeur est trop petit, il ne rentre pas sur l'axe du moteur parce que son diamètre du disque central est trop petit. J'ai modélisé une roue à la bonne échelle pour qu'elle puisse tenir sur l'axe des coupleurs plutôt que sur l'axe du moteur.
Conclusion : La solution s'améliore mais n'est pas suffisamment fiable, en effet à force de rouler, le disque central s'élargit et n'est plus stable, ce qui engendre des erreurs de distance.

— **Modélisation 3** Enfin, je me suis fortement inspiré de la modélisation précédente pour arriver à ce modèle final. Au lieu de placer les encodeurs sur les roues avant, il est préférable de les placer sur les roues du milieu. C'est les roues qui patinent le moins, ce qui permettra d'avoir des résultats avec plus de précision. J'ai donc imaginé un support qui se fixe autour du moteur, et qui permet de placer l'encodeur au-dessus de la roue. Concernant le disque, ce qu'il fallait améliorer c'est sa fixation sur le coupleur, j'ai donc rajouté la possibilité de le fixer avec des vis au même endroit où l'on fixe le coupleur à l'axe moteur. Les trois pièces sont donc reliées grâce à deux vis. Enfin j'ai réduit le nombre de trous de 25 à 14 pour éviter que les performances de mon montage dépendent de la vitesse du robot. En effet, même si le robot prend beaucoup de vitesse, il faut qu'il détecte tous les changements d'état du disque.
Conclusion : Cette solution est parfaite parce que les résultats obtenus sont très fiables.

Ci-dessous les photos des différentes modélisations :

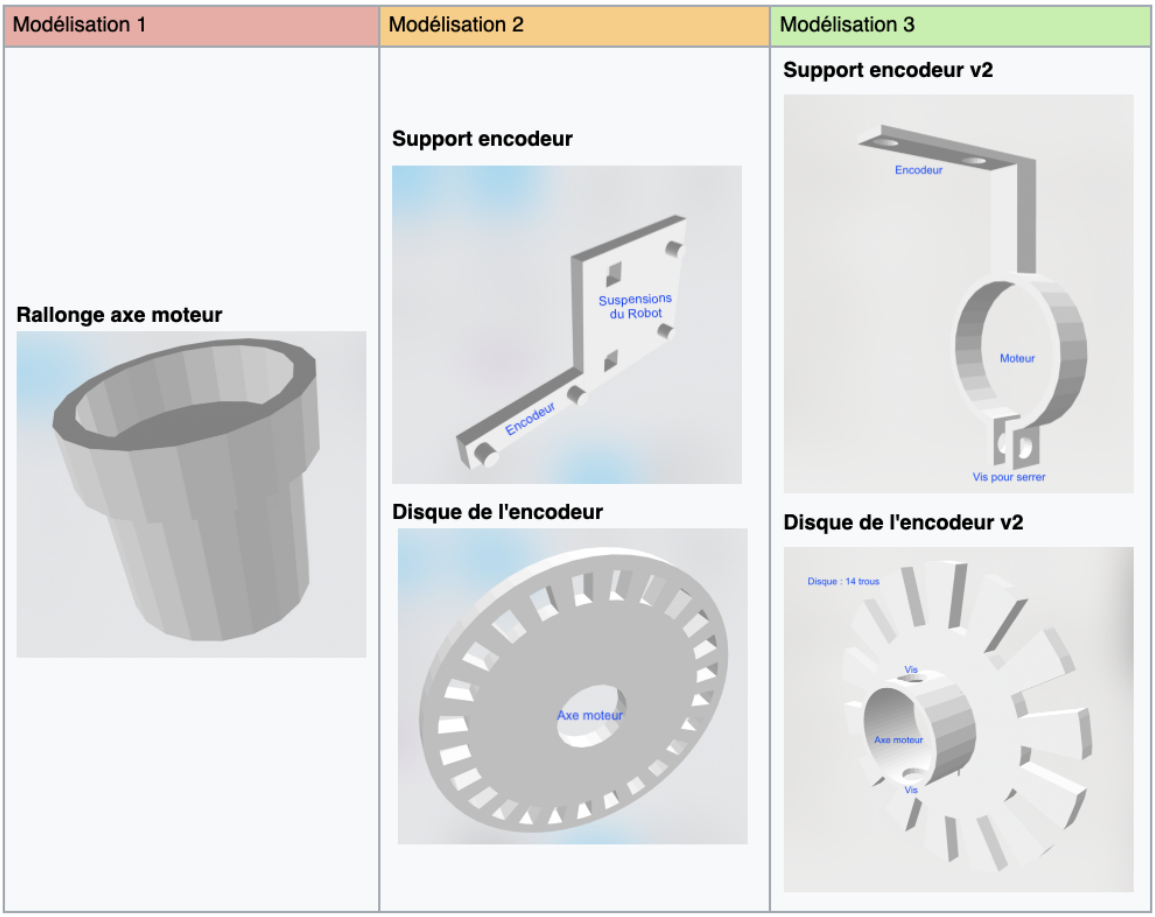


FIGURE 7 – Récapitulatif des modélisations

Voici le rendu final des deux pièces sur le robot :

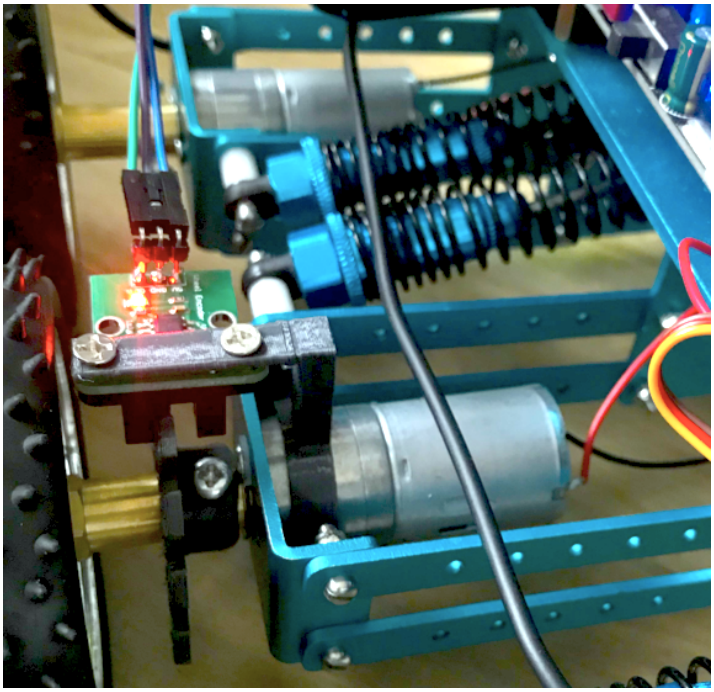


FIGURE 8 – Encodeur droit

Code :

```
#define ENCODER_PIN1 2
#define ENCODER_INT1 digitalPinToInterrupt(ENCODER_PIN1)
EncoderStepCounter encoder(ENCODER_PIN1, ENCODER_PIN1 );

// Dans la fonction setup() : Initialisation de l'encodeur
    encoder.begin();
    attachInterrupt(ENCODER_INT1, interrupt, RISING);

void interrupt() {
  encoder.tick();
}
// Dans la fonction loop() :
  signed char position = encoder.getPosition();
  if (position != 0) {
    count += position;
    encoder.reset();
  }
```

Voici un exemple de mon code pour un encodeur. Il utilise la librairie "EncoderStepCounter.h". Le principe est de provoquer des interruptions dès que le capteur détecte un changement d'état, et plus précisément sur un front montant. Pendant cette interruption, la fonction interrupt() est exécutée.

2.3.3 Contrôler sa rotation : Boussole

Voici la référence de l'accéléromètre/boussole que je vais utiliser pour ce projet : Adafruit Triple-axis Accelerometer+Magnetometer (Compass) Board - LSM303 [ADA1120]

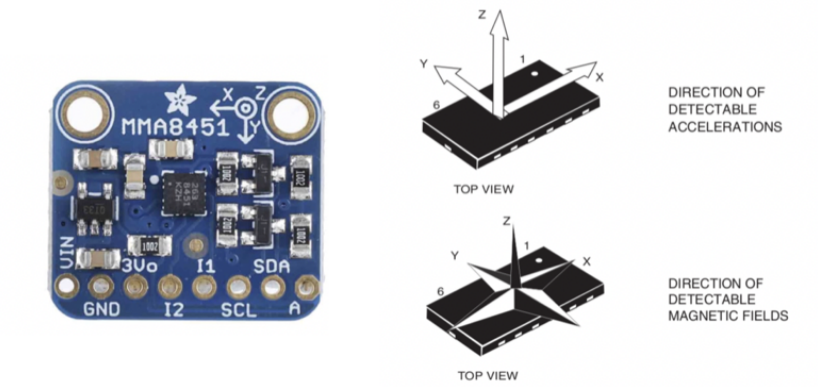


FIGURE 9 – Boussole du projet

J'ai utilisé la datasheet pour en savoir plus sur ce composant. Il permet d'avoir 3 variables de champ magnétique et 3 variables d'accélération. Il utilise le bus I2C pour envoyer les données en liaison série.

Câblage :
Comme j'utilise déjà les pins SDA et SDL pour la carte extensive qui contrôle les moteurs du robot, je me suis renseignée sur internet pour trouver une solution. Il est possible d'avoir à nouveau deux pins SDA et SDL en utilisant les pins A5 et A4. "Ceux qui disposent de l'Arduino Uno ou d'une carte compatible utiliseront les connecteurs A4 pour SDA (les données) et A5 pour SCL (l'horloge)".

Donc en résumé :

- VCC : 5V
- GND : GND
- SDA : A4
- SDL : A5

Code :
Une fois le câblage réalisé, j'ai récupéré la librairie du composant LSM303DHL sur le logiciel Arduino que j'ai inclus dans le code. Cette bibliothèque permet de récupérer les valeurs selon l'axe x, y et z. Il y a une relation mathématique qui me permet de convertir ces trois données en une donnée en degré :

$$Angle = \frac{180 * atan2(magy, magx)}{\pi} \tag{2}$$

En testant la boussole seule (loin du robot), j'avais des résultats assez cohérents. Mais en la plaçant sur le robot, je me suis rendue compte que les valeurs n'étaient pas utilisables. J'ai récupéré les valeurs que j'obtenais en fonction de l'orientation du robot (cf image à droite). J'ai ensuite testé la boussole en dehors du robot, (cf à gauche), j'obtiens des valeurs plus cohérentes mais ce n'est pas suffisant pour bien contrôler le robot (un quart de cercle représente 180 degrés et les trois autres 180 degrés). Le problème vient des champs magnétiques présents sur le robot qui perturbent la boussole.

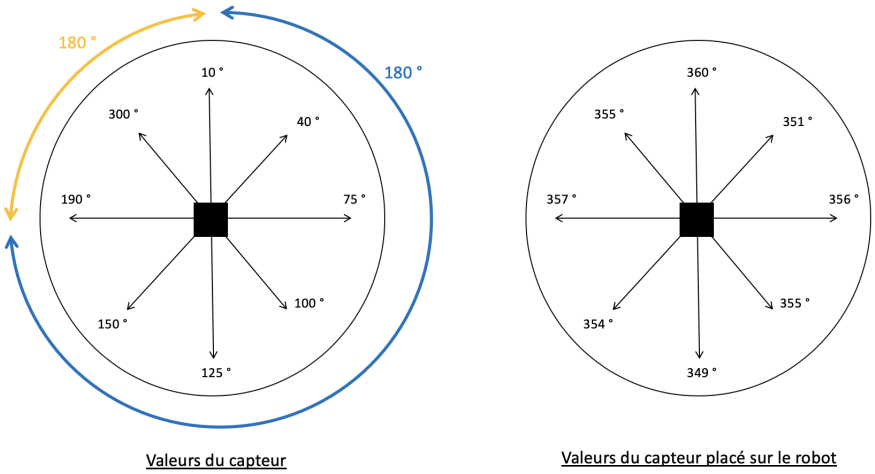


FIGURE 10 – Boussole défectueuse du projet

Il est vrai que plusieurs éléments sur mon robot (moteurs, carte Arduino et Raspberry Pi, batteries ...) peuvent impacter le champ magnétique autour de la boussole. Donc je cherchais une façon de le placer sur le robot pour avoir le moins d'interférence. Mais ce n'est que l'avant-dernière semaine du projet que l'on m'a informé que le capteur qu'on m'a donné pour le projet était réellement défectueux.

Voici le résumé de l'équipement que j'utilise pour ce projet. La boussole est le seul capteur que je n'utilise finalement pas dans ce projet parce qu'il ne fonctionne pas.

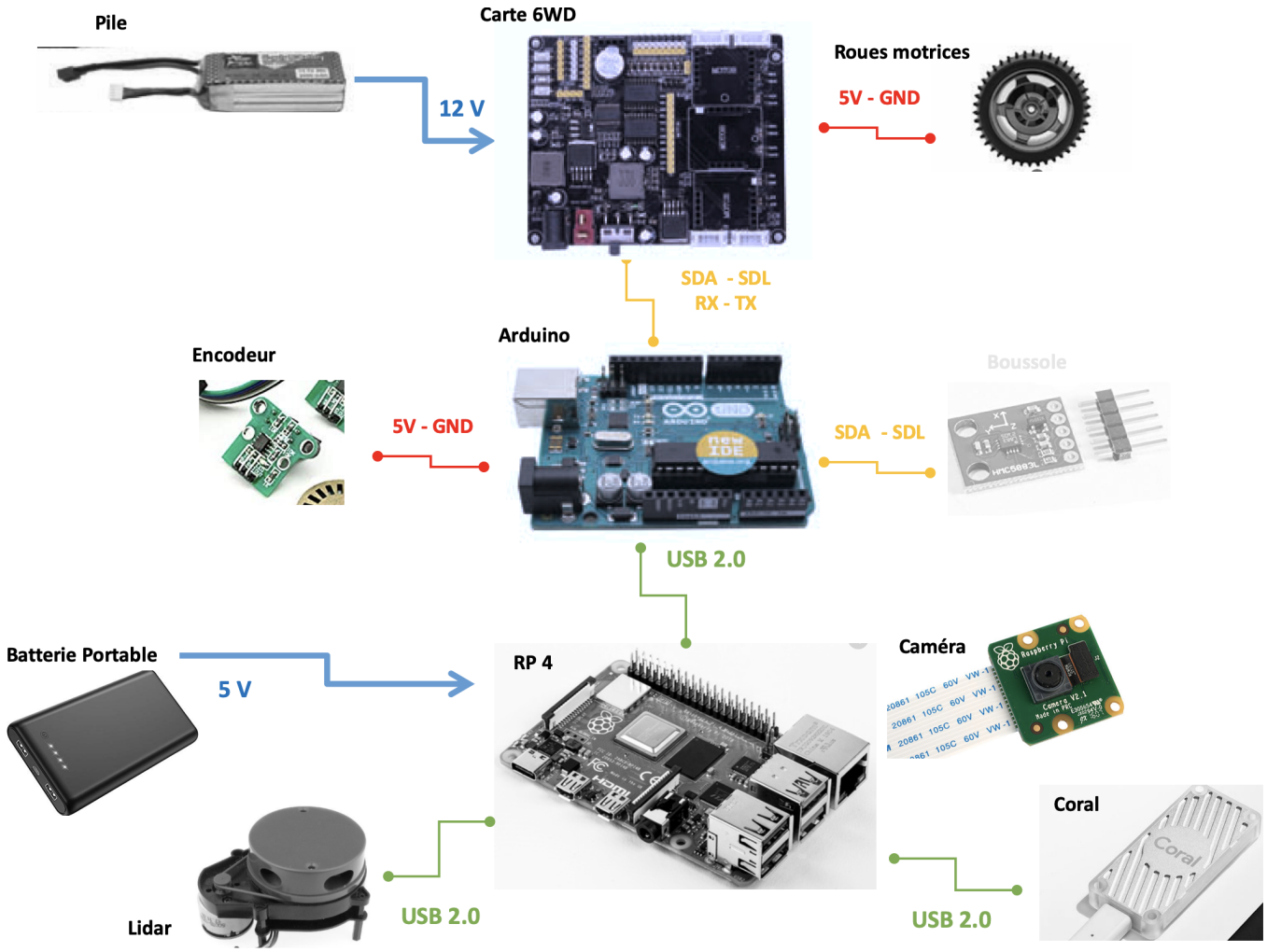


FIGURE 11 – Résumé de l'équipement

3 Cartographie

Au début du projet, on m'a laissé la liberté concernant les outils que j'utiliserai pour arriver à faire de la cartographie en continu. Je présenterai dans cette partie les deux axes majeurs de recherche.

3.1 OpenCV

3.1.1 Présentation

OpenCV (Open Source Computer Vision) est une bibliothèque Open Source qui propose plus de 2500 algorithmes pour le traitement d'images et le machine learning. Ces algorithmes peuvent être utilisés pour détecter et reconnaître des visages, identifier des objets, classifier des actions humaines dans des vidéos, suivre les mouvements de la caméra, suivre des objets, extraire les modèles 3D des objets, produire des images de points en 3D à partir de caméras stéréos, assembler des images pour produire une scène haute résolution, trouver des images similaires dans une base de données, suivre les mouvements des yeux... Elle a été créée pour proposer une infrastructure commune aux projets de vision par ordinateur et a été réalisée avec une considération importante pour l'efficacité de calcul et les applications temps réels.

3.1.2 Création de la cartographie

J'ai modifié le code du Lidar pour qu'il récupère les données et les enregistre dans un fichier JSON. J'ai ensuite créé un programme python qui récupère les données du fichier et qui crée une figure en plaçant les points en fonction de leur angle et de leur distance. Pour placer le point, il faut déterminer son abscisse et son ordonnée. Pour les obtenir, on doit d'abord se placer au centre de la figure et utiliser les formules de trigonométrie : SOH CAH TOA.

Le principe est de pouvoir récupérer les données du Lidar toutes les 30 secondes et la cartographie associée. Ensuite, l'idée est de récupérer deux cartographies successives et les comparer.

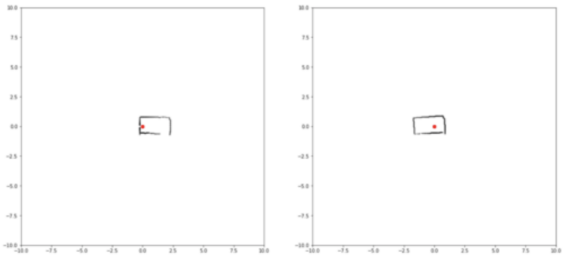


Figure 12 – Exemple de deux cartographies successives - Python

3.1.3 Mes recherches

Pour tracer les murs, j'ai pensé à :

— Reconnaître la forme de la pièce,
— Relier les points récupérés par le Lidar,
Mais le premier affichage de la cartographie ne donne rien. Python ne reconnaît pas les formes parce qu'il y a des endroits sans information.

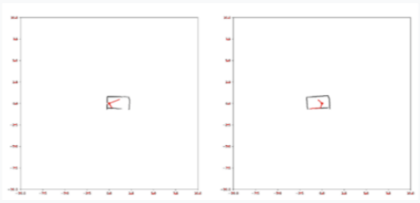
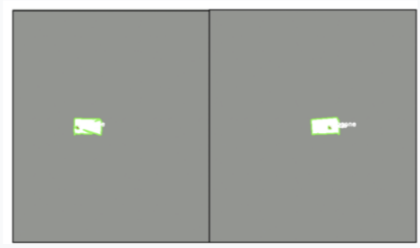
Résultats	Commentaires
	Comme on peut voir sur l'image, les informations ne sont pas exploitables. J'ai donc changé la façon d'afficher les données en m'inspirant de l'affichage de ROS. Le principe serait de mettre un fond gris, et de tracer des lignes blanches entre la position du robot et l'obstacle détecté ; voici le rendu du deuxième affichage ci-dessous.
	Comme on peut voir, les endroits sans information sont parfois complétés, et j'arrive maintenant à reconnaître les formes afin de tracer les contours de la pièce.
	Il reste à lisser les contours pour corriger les éventuelles erreurs du LIDAR. J'ai trouvé une thèse intéressante à ce sujet sur le site suivant, voici quelques résultats de leurs recherches sur la photo ci-dessous.

FIGURE 13 – Tracé des contours

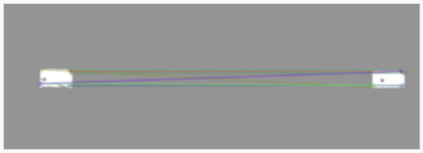
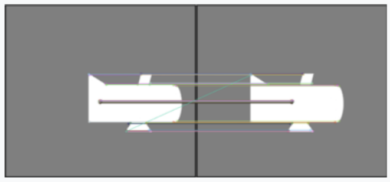
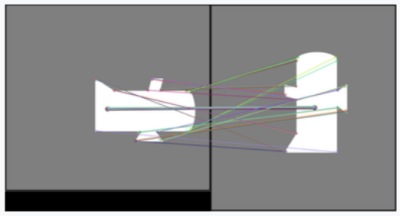
Résultats	Commentaires
	Comme on peut voir sur cette image, Python n'a détecté que 2 Keypoints et ils sont faux !
	J'ai essayé avec deux cartographies, on peut voir que le résultat s'améliore puisqu'il y a beaucoup plus d'informations. Cependant il reste quelques erreurs (1 seule : ligne violette).
	J'ai créé une forme avec beaucoup d'informations pour voir le résultat, on remarque à nouveau beaucoup de keypoints détectés et une fois de plus il n'y en a qu'un seul qui est faux.
	J'ai pivoté une des deux photos, le résultat est impressionnant ! Malgré la rotation, il y a toujours autant de keypoints et ils sont presque tous justes !

FIGURE 14 – Recherches des similitudes

3.1.4 Résultats

C'était très intéressant et instructif d'étudier la cartographie ainsi. Au début du projet ayant un RaspberryPi 3, je n'ai pas pu travailler sur ROS. C'est un système trop puissant, son installation nécessite beaucoup de puissance, sans parler de son lancement ! C'est pour cela que je me suis tournée vers OpenCV. En novembre, j'ai reçu le RaspberryPi 4 pour le projet. Il était préférable d'utiliser ROS à partir de ce moment, parce que le travail de la création de la cartographie en temps réel était déjà réalisée et que ma période de projet est trop courte pour tout reprendre.

3.2 ROS

3.2.1 Présentation

ROS (Robot Operating System) est un système open source qui permet à un utilisateur de contrôler un robot. Il n'a pas de langage de programmation défini et peut être programmé via plusieurs langages. Un système ROS comprend un certain nombre de nœuds indépendants qui communiquent avec les autres nœuds via une messagerie de type publication/abonnement. Par exemple, le driver d'un capteur peut être implémenté comme un nœud qui publie les valeurs du capteur dans un flux de messages. Ces messages pourraient être utilisés par n'importe quel nombre d'autres nœuds, comme des filtres, des collecteurs de données ou des systèmes haut niveau comme des systèmes de guidage ou de recherche de chemin.

3.2.2 Fonctionnement de ROS

ROS crée un réseau où tous les processus sont connectés. N'importe quel nœud du système peut accéder à ce réseau, interagir avec d'autres nœuds, voir les informations qu'ils envoient et transmettre des données au réseau : Voici une liste des principaux éléments de ROS et quelques commandes utiles :

— **Nodes (Nœuds) :** les nœuds sont des processus où le calcul est effectué. Pour avoir un processus qui peut interagir avec d'autres nœuds, il faut créer un nœud avec ce processus pour le connecter au réseau ROS. Habituellement, un système aura de nombreux nœuds pour contrôler différentes fonctions. Il est préférable d'avoir de nombreux nœuds qui ne fournissent qu'une seule fonctionnalité, plutôt que d'avoir un grand nœud qui fait tout dans le système. Les nœuds sont écrits avec une bibliothèque client ROS, par exemple, roscpp ou rospy.

```
rosnode info NODE // Affiche des informations sur un noeud
rosnode kill NODE // Arrete un noeud en cours d'execution
rosnode list // Liste les noeuds actifs
rosnode ping NODE // Teste la connexion avec le noeud
```

— **Master (Le maître)** : le maître fournit l'enregistrement des noms et le service de recherche au reste des nœuds. Il établit également des connexions entre les nœuds. Messages : les nœuds communiquent entre eux par le biais de messages. Un message contient des données qui fournissent des informations à d'autres nœuds. ROS propose de nombreux types de messages, et vous pouvez également développer votre propre type de message à l'aide de types de messages standard.

```
rosmsg list // Affiche tous les massages
rosmsg package // Affiche les messages d'un package
```

— **Topic :** Chaque message doit avoir un nom pour être acheminé par le réseau ROS. Lorsqu'un nœud envoie des données, le nœud publie un sujet. Les nœuds peuvent recevoir des sujets d'autres nœuds en s'abonnant simplement au sujet. Un nœud peut s'abonner à un sujet même s'il n'y a aucun autre nœud publiant sur ce sujet spécifique. Il est important que les noms des sujets soient uniques pour éviter les problèmes et la confusion entre les sujets portant le même nom.

```
rostopic echo /topic // Affiche les messages
rostopic find message\_type // Trouver un Topic par son nom
rostopic hz /topic // Affiche la frequence d'affichage
rostopic info /topic // Affiche des informations sur le topic, le type du message
    , le publishers et subscribers.
rostopic list // Affiche les topics en cours
rostopic pub /topic type args // Permet de publier une donnee dans un topic \\
```

— **Bag :** les bags sont un format pour enregistrer et lire les données des messages ROS. Les bags sont un mécanisme important pour stocker des données, telles que les données de capteurs, qui peuvent être difficiles à collecter mais qui sont nécessaires pour développer et tester des algorithmes.

```
rosbag // Permet d'enregistrer des donnees
```

3.2.3 Installation de ROS

A partir de début novembre, j'ai eu la possibilité de travailler avec une carte Raspberry Pi 4 (4GB RAM). J'ai choisi d'installer la version Melodic puisque c'est l'une des dernières versions mais surtout parce que cette version est supportée jusqu'en mai 2023. Son installation demande beaucoup de patience, mais une fois installée on est rapidement interpellé par le nombre d'outils déjà implémenté sur ROS.
Le tutoriel pour l'installation de ROS présent sur leur site, n'était pas à jour pendant que je travaillais sur ce projet. Si c'est toujours le cas, vous trouverez en annexe de ce rapport un tutoriel pour installer ROS sur la rapsberry Pi 4. (cf. annexe 1).

3.2.4 Mes recherches

Pour mon projet, j'ai besoin du package pour le Lidar proposé par son constructeur et celui d' Hector SLAM Hector . Il faut donc suivre les lignes 7 à 11 avec les deux liens github pour les ajouter au répertoire ROS.

RViz est l'outil de visualisation 3D de ROS. Le but principal est de montrer les messages ROS en 3D, ce qui nous permet de vérifier visuellement les données. Par exemple, il peut visualiser la distance entre le Lidar et un obstacle sous forme de nuage de points, la valeur d'image obtenue à partir d'un appareil photo, et bien d'autres sans avoir à développer séparément le logiciel.

```
1 //Lancement du noeud maitre
2 $ rocore
3 //Lancement du Lidar Seul avec RViz
4 $ roslaunch ydlidar lidar_view.launch
5 //Lancement du Lidar
6 $ roslaunch ydlidar lidar.launch
7 //Lancement d'Hector Slam avec les donnees du Lidar sur Rviz
8 $ roslaunch hector_slam_launch tutorial.launch
```

La deuxième ligne permet de lancer le noyau ROS, il permet de faire le lien entre les différents topics et noyaux. La commande 4 permet de lancer le noyau ylidar en exécutant le fichier lidar_view.launch. Un fichier de type launch est un fichier où l'on définit tous les paramètres et le fichier à exécuter. Ici le fichier exécute le fichier suivant : ydlidar_node.cpp. Il lance le Lidar, récupère les données qu'il publie sur le topic /scan. Une fenêtre s'ouvre et affiche le nuage de point. Il y a une deuxième façon d'afficher la cartographie en lançant les commandes 2, 6 et 8. Cette fois-ci la commande 6 n'ouvre pas rviz, mais publie directement les données sur le topic. La commande 8 permet de récupérer ces données et de construire une vraie cartographie. C'est hector_slam qui s'en charge. Ce dernier publie la cartographie sur le topic /map

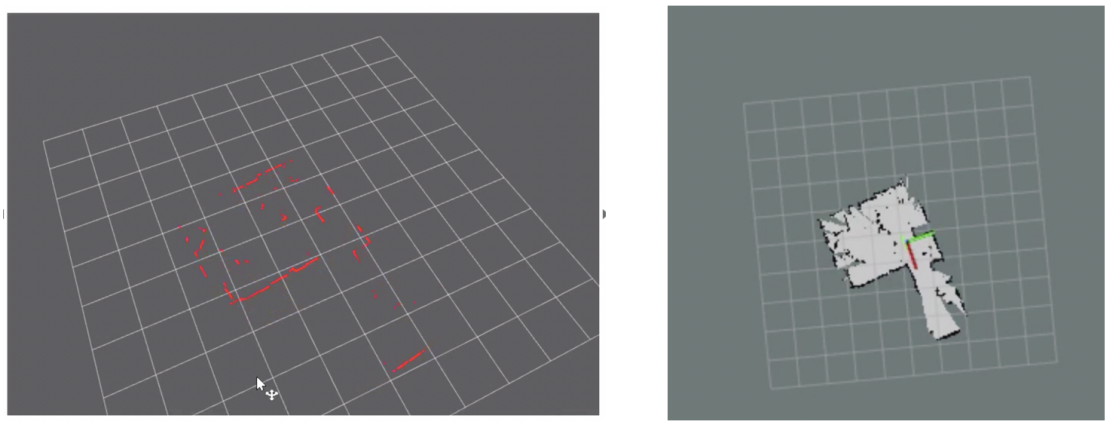


FIGURE 15 – Résultat ROS Lidar / ROS Lidar + HectorSLAM

On peut voir que cette version s'apparente beaucoup à ce que j'obtenais en utilisant Python et OpenCV. ROS positionne seulement les points en fonction de l'angle et de la distance. La seule différence non négligeable avec mon programme (Opencv) c'est que le programme tourne en temps réel, donc les points s'actualisent si l'environnement a changé.
Sur la deuxième photo (Figure 15), ROS trace les murs en noir et trace le chemin parcouru en vert. Les deux lignes verte et rouge permettent de situer le robot dans la pièce. Cette version s'actualise en Temps réel aussi.

L'inconvénient principal d'Hector SLAM, est qu'il faut faire attention avec les mouvements du robot. Il ne faut pas qu'ils soient trop brusques ou trop rapide. Voici le résultat :

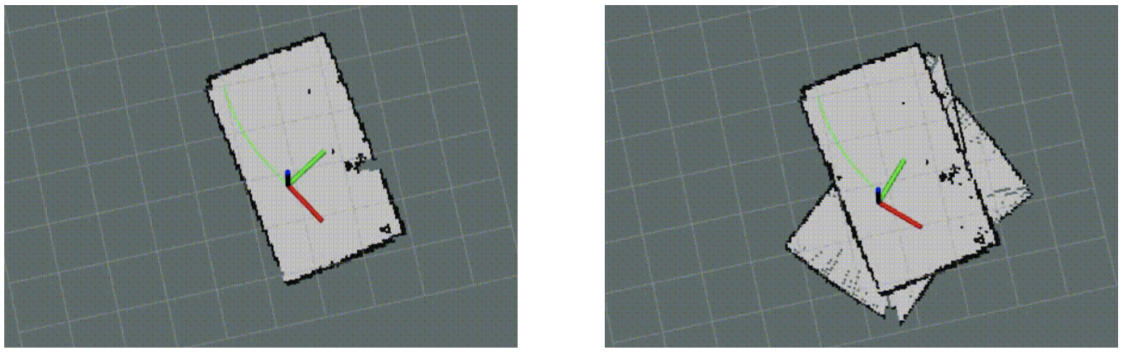


FIGURE 16 – Shift Problems

Ce problème est dû à des mouvements brusques du robot, lorsqu'il va trop vite : le phénomène de shift. Cela arrive souvent lorsque le robot tourne de façon trop rapide. Mais c'est également dû au fait que le robot n'est pas "solide". Lorsqu'il avance avec une vitesse importante et qu'il freine subitement. A cause des suspensions de mauvaise qualité, le lidar qui est placé sur le robot change de position et ne reste pas vraiment horizontal.

Il est possible de changer les paramètres de précisions dans le fichier de launch. Voici un exemple de fichier en annexe 2.

3.2.5 Contrôler le robot avec ROS

TurtleBot est un kit de robot avec un logiciel open source. TurtleBot a été créé au Willow Garage par Melonee Wise et Tully Foote en novembre 2010. Ce robot est capable de circuler dans votre maison, de voir en 3D la carte de la pièce. L'objectif de cette formation est de pouvoir rendre mon robot compatible avec ROS et donc de pouvoir le contrôler directement depuis la plateforme de ROS. Voici la page où l'on trouve le dossier nécessaire. Pour les personnes n'ayant pas le robot Turtlebot, il existe une simulation.

Dans un premier temps, il faut lancer :

```
1 $ roscore
2 $ rosrun turtlesim turtlesim_node
```

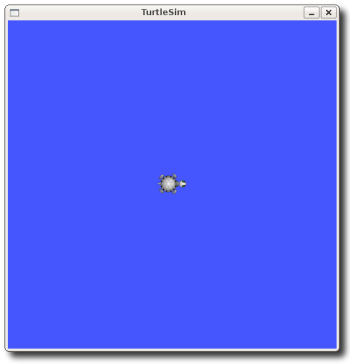


FIGURE 17 – TurtleSim

L'objectif à présent est de déplacer cette tortue dans son environnement. Il est tout à fait possible de la déplacer en utilisant les touches (up, down, right et left) avec la commande ci dessous :

```
1 $ rosrun turtlesim turtle_teleop_key
```

Mais cela ne fonctionnera pas avec un vrai robot par la suite. Pour que ROS puisse contrôler un robot il envoie les coordonnées dans un message de type Twist :

```
1 geometry_msgs/Vector3 linear
2 geometry_msgs/Vector3 angular
```

C'est un message qui se compose de deux vecteurs, un vecteur linéaire avec trois valeurs et un second vecteur pour trois valeurs angulaires. Il existe des limitations en ce qui concerne le robot : La navigation n'est possible que pour des robots "diferential drive" et des robots holonomiques. C'est pour cela que j'ai rajouté des encodeurs aux roues du milieu. Le robot reçoit un message de type twist avec des vitesses X, Y et θ qui lui permettent de se déplacer. Si le robot n'est pas en mesure de les comprendre, il est possible de créer un nœud ROS qui convertit un message vers un autre de type twist.
=> Il faut comprendre le fonctionnement des vecteurs sur le déplacement du robot.

Je vais donc opter pour le déplacement avec les deux vecteurs. J'ai fait des recherches concernant 'diferential drive' et je suis tombée sur ce site qui explique bien le fonctionnement mathématique avec le corrigé d'un exercice. J'ai remarqué que seules deux valeurs sur les 6 sont utiles. X permet d'avancer d'une distance et θz permet de tourner de l'angle indiqué.

Vecteur Linéaire	Vecteur angulaire
x	θ x
y	θ y
z	θ z

Pour déplacer la tortue avec ce type de message :

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist  1     '[ Xvalue, Yvalue, Zvalue  ]
    ' '[\theta Xvalue, \theta Yvalue, \theta Zvalue  ]'
```

Par exemple pour que le robot puisse se déplacer en réalisant la forme d'un carré, voici les commandes à réaliser quatre fois :

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist  1     '[2,0,0]' '[0,0,0]' // Le
    robot avance de 2
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist  1     '[0,0,0]' '[0,0,1.57]' //
    Le robot tourne sur lui meme de Pi/2 (=1.57)
```

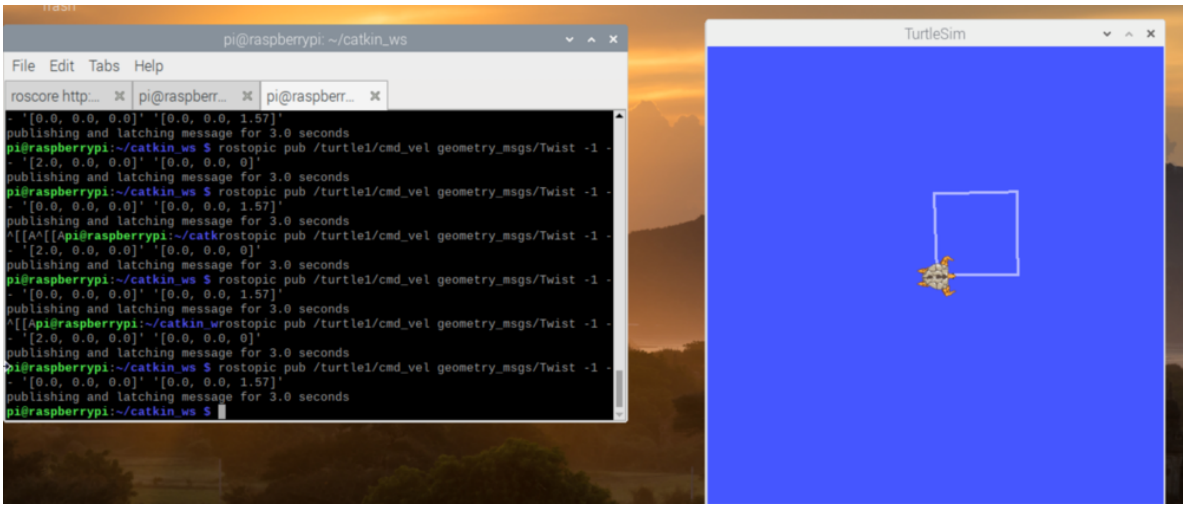


FIGURE 18 – Contrôle robot avec les messages "Twist"

J'aimerais pour la suite du projet, pouvoir contrôler le robot avec l'interface graphique de ROS. Pour le moment je contrôle les moteurs du robot en imposant une valeur entre 0 et 255 à chaque roue. (Pour rappel, le robot est équipé de 6 moteurs à courant continu). Avec ROS, la seule façon de contrôler un robot c'est en lui imposant les valeurs de vitesse x, y et θ. Mais je n'ai pas de boussole pour pouvoir contrôler le robot avec les messages Twist.

ROS Serial
Avant de me lancer dans les messages twist je me suis formée pour créer un noeud "subscriber" et "publisher".

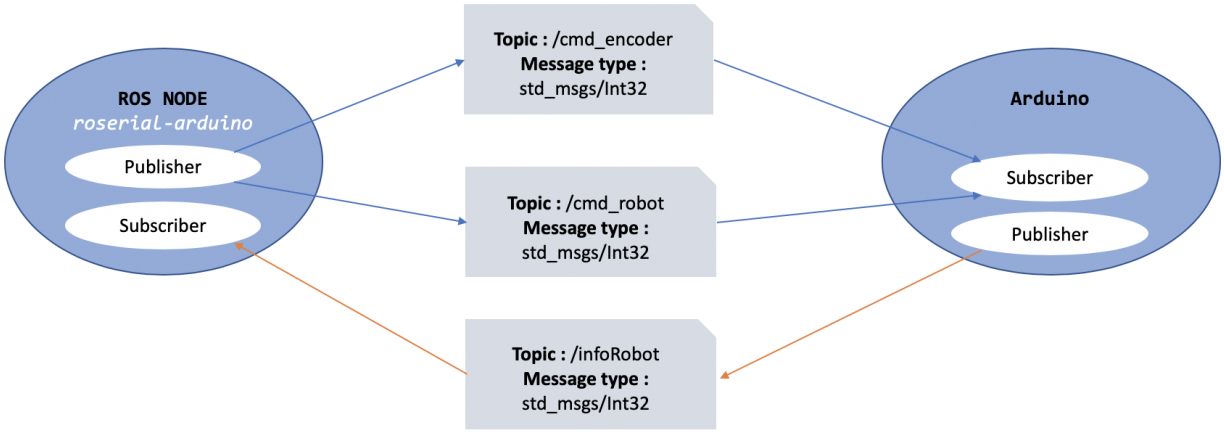


FIGURE 19 – Organisation

Voici la structure que j'ai choisi de développer. Cela me permet d'apprendre à avoir un code sur l'arduino qui publie des informations sur un topic mais aussi qui est abonné à deux topics.

Tout est fonctionnel, l'étape suivante est de réussir à convertir un message twist en commande simple de déplacement du robot. Cependant, comme mentionné, la boussole du projet est défectueuse. Sans cette dernière je ne peux pas écrire de message Twist. Arrivant à la fin du projet, j'ai donc préféré améliorer les fonctions existantes, les fonctionnalités du site et bien rédiger la documentation du projet. Cela permettra aux successeurs du projet de le reprendre avec beaucoup de précisions et d'aide pour continuer cette partie.

4 Web

4.1 Maquette du Site

Voici la maquette que j'imaginais au début du projet :

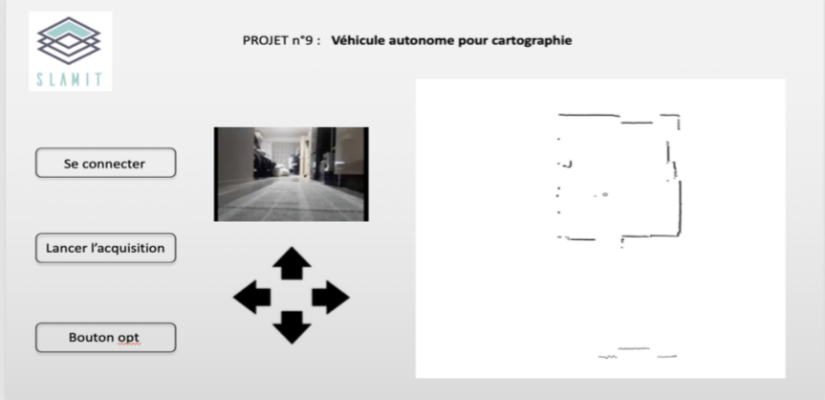


FIGURE 20 – Maquette du site

Il permet de répondre à toutes les attentes du cahier des charges :
- Connexion avec le robot
- Lancement du lidar pour effectuer une cartographie
- Bouton pour contrôler le robot
- Image de la caméra du robot
- Cartographie

4.2 Fonctionnement du site

4.2.1 Connexion avec le robot

Pour permettre la communication entre le robot (Arduino) et la Raspberry Pi, je vais utiliser des Websockets. C'est un protocole qui permet de développer un canal de communication sur un socket TCP. Il permet donc d'ouvrir une connexion permanente entre le navigateur et le serveur.

Code HTML

```
var localhost = '127.0.0.1:9000';
var ip_raspberry = '192.168.0.20:9000';
var websocket=new WebSocket('ws://'+ip_raspberry,'serial');
websocket.onopen=function(){ $('h1').css('color','green');};
websocket.onerror=function(){ $('h1').css('color','red'); };
```

Les deux dernières lignes permettent d'informer l'utilisateur de la page. Le titre est :
- Vert si la connexion entre le navigateur et la Raspberry Pi est fonctionnelle,
- Rouge dans le cas contraire.

4.2.2 Caméra et IA

Afin de rajouter de l'intelligence artificielle sur ce projet, Bonduelle a acheté un accélérateur USB : Coral

FIGURE 21 – Coral

Fonctionnement :

Avec Coral, Google a lancé une nouvelle plate-forme d'outils matériels et logiciels qui permet de développer des réseaux de neurones localement.

L'accélérateur USB Coral contient une puce Edge TPU (Tensor Processing Unit). L'accélérateur USB Coral a été conçu comme un périphérique facile à connecter pour mettre l'Edge TPU à la disposition d'une carte Raspberry Pi. Avec les réseaux de neurones qui fonctionnent sur l'appareil lui-même, cela permet d'ajouter de l'intelligence Artificielle à un projet. La plate-forme Coral contient également une collection de modèles pré-formés et pré-compilés qui sont optimisés pour le TPU Edge.

J'ai commencé cette partie l'avant dernière semaine du projet. Je n'avais pas le temps de créer mes propres modèles. J'ai donc récupéré le fichier de leur site qui est un exemple . C'est un programme python qui prend en argument une image à étudier et une bibliothèque (coconames).
On obtient après exécution de ce dernier, l'image d'entrée avec des rectangles entourant des objets et le pourcentage de certitude. J'ai modifié ce programme pour qu'il n'attende plus en argument une image, mais qu'il récupère le flux vidéo de la caméra et qu'il traite en temps réel les images.

Voici le rendu qui est assez impressionnant :

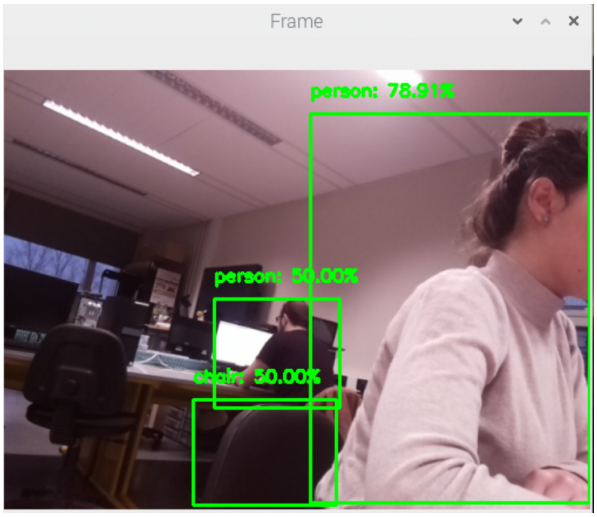


FIGURE 22 – Détection d'objet et de personnes

Enfin l'objectif final était que le programme retourne ce flux vidéo annoté par des rectangles sur une adresse IP pour que je puisse l'ajouter au site du projet.

```
# start the flask app
        app.run(ipadresse, port, debug=True,
                threaded=True, use_reloader=False)
```

Pour ce faire j'utilise le framework Flask en python.

4.2.3 Cartographie

DJS

ros2djs est le gestionnaire de visualisation 2D JavaScript standard pour ROS. Il est construit au sommet de roslibjs et utilise la puissance d'EaselJS. De nombreuses fonctionnalités ROS standard comme les cartes sont incluses dans le cadre de cette bibliothèque. ros2djs est développé dans le cadre de l'effort Robot Web Tools. Voici la documentation de cette bibliothèque.

Code

Il faut d'abord importer tous les fichiers JavaScript requis :
- easeljs.min.js
- eventemitter2.min.js
- roslib.min.js
- ros2d.min.js

Ensuite, il faut créer un objet nœud Ros pour communiquer avec un serveur rosbridge v2.0. Si l'on souhaite que le site puisse être accessible depuis un autre ordinateur, il faut modifier l'adresse localhost par l'adresse ip de la raspberry pi. Il faut créer une visionneuse 2D. Pour avoir la carte publiée par Hector ROS, il faut créer l'objet OccupancyGridClient. Dans le code HTML <body>, il faut rajouter une div avec un ID correspondant pour le spectateur.

```
function init() {
    // Connect to ROS.
    var ros = new ROSLIB.Ros({
      url : 'ws://localhost:9090'
    });
    // Create the main viewer.
    var viewer = new ROS2D.Viewer({
      divID : 'map',
      width : 600,
      height : 500
    });

    // Setup the map client.
    var gridClient = new ROS2D.OccupancyGridClient({
      ros : ros,
      rootObject : viewer.scene
    });
    // Scale the canvas to fit to the map
    gridClient.on('change', function(){
      viewer.scaleToDimensions(gridClient.currentGrid.width, gridClient.currentGrid.
          height);
    });
  }
```

En parallèle, il faut lancer cette commande sur la rpasberry pi :

```
roslaunch ydlidar lidar_view.launch
roslaunch ydlidar lidar.launch
roslaunch hector_slam_launch slamit.launch
//Cette commande permet de recuperer le topic map sur le site
roslaunch rosbridge_server rosbridge_websocket.launch
```

Voici la page finale du projet :

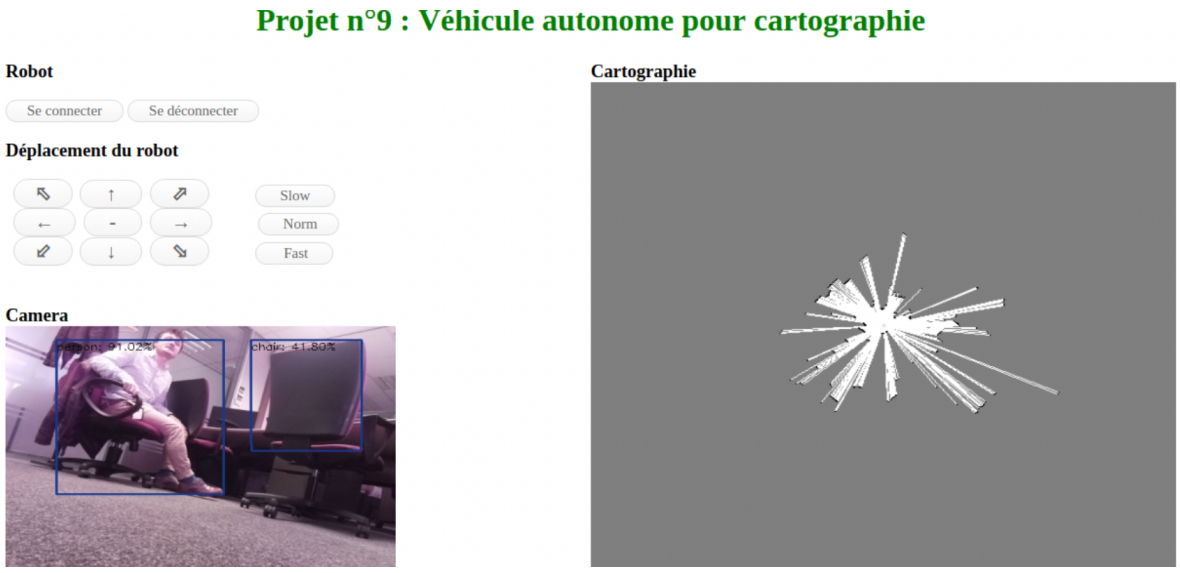


FIGURE 23 – Cartographie 1

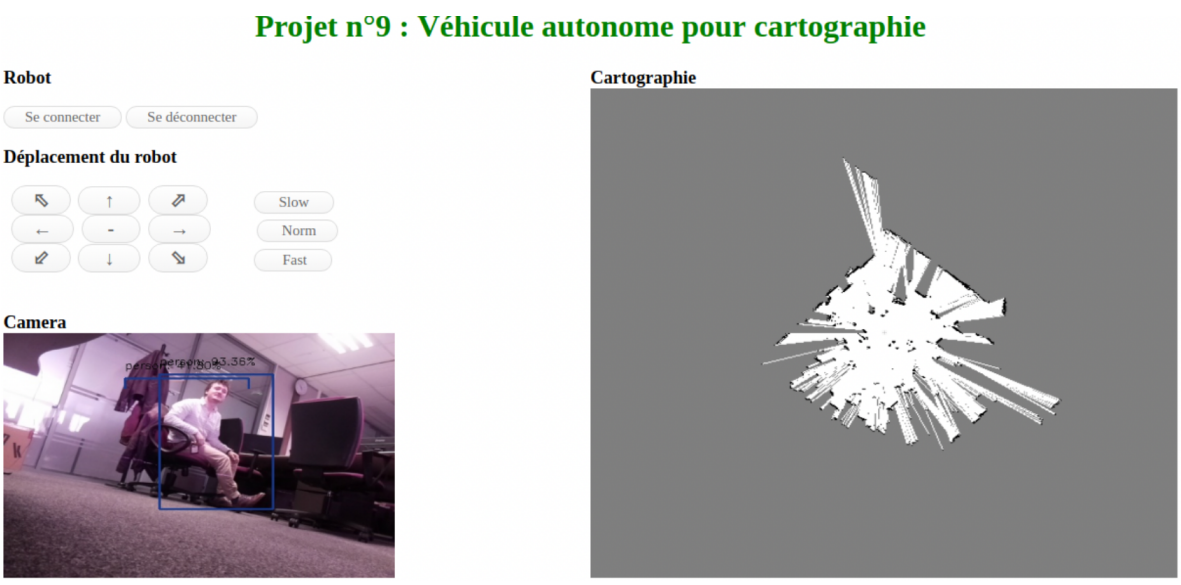


FIGURE 24 – Cartographie 2

5 Conclusion

Ce projet a été très enrichissant pour moi. C'est un de mes premiers projets fortement orienté en Robotique. J'ai beaucoup appris en très peu de temps, notamment le système ROS qui est vraiment un outil indispensable en robotique. Ce projet m'a aussi permis d'être confrontée à un réel projet d'ingénieur puisqu'il est réalisé en partenariat avec l'entreprise Bonduelle.

L'objectif de ce projet était de réaliser les premières recherches pour concevoir un véhicule autonome qui est capable de cartographier son environnement. Pour l'instant, je contrôle le robot à distance, je cartographie son environnement, il manque seulement la liaison entre ROS et le robot. Cela permettrait de contrôler le robot directement sur la cartographie affichée sur le site du projet. Malgré un cahier des charges conséquent et le fait que je sois seule sur ce projet, je suis assez satisfaite du travail que j'ai accompli tout au long de l'année et de l'avancée actuel du projet.

Durant ce projet, j'ai eu l'occasion d'approfondir mes compétences en programmation, en robotique mais également dans la recherche et dans l'auto-critique du travail accompli et la gestion de projet. Mais aussi en termes de traitement d'images, où j'avais de faibles connaissances.

Je remercie à nouveau l'ensemble de mes encadrants qui m'ont permis de réaliser ce projet. Cela reste une expérience humaine très enrichissante qui m'aidera certainement lors de mon stage ou encore en milieu professionnel une fois diplômée.

Table des figures

Annexes

1 Installation de ROS - 250 min

Etape 1 : Installation installateur ROS - Environ 5 min

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"
> /etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

sudo apt-get update

sudo apt-get install -y python-rosdep python-rosinstall-generator python-wstool
python-rosinstall build-essential  cmake
```

Etape 2 : Initialisation de l'outils ROS Dependencies - Environ 1 min

```
sudo rosdep init

rosdep update
```

Etape 3 : Création du répertoire des outils Environ 5 min

```
mkdir ~/ros_catkin_ws

cd ~/ros_catkin_ws

rosinstall_generator desktop --rosdistro melodic --deps --wet-only --tar >
melodic-desktop-wet.rosinstall

wstool init -j8 src melodic-desktop-wet.rosinstall

sudo apt-get install  libogre-1.9-dev
```

Etape 4 : Installation des dépendances - Environ 60 min

```
cd ~/ros_catkin_ws

rosdep install --from-paths src --ignore-src --rosdistro melodic -y
```

Etape 5 : Installation des 186 outils de ROS - Environ 120 min

```
cd ~/ros_catkin_ws

sudo ./src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release
--install-space /opt/ros/melodic -j2

source /opt/ros/melodic/setup.bash

echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
```

Etape 6 : Création du répertoire Catkin - Environ 5 min

```
mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/

sudo apt-get install python-catkin-tools

catkin_make

echo "source $HOME/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

Etape 7 : Installation des packages additionnels - Environ 30 min

```
cd ~/catkin_ws/src

sudo git clone  <link>

cd ..

sudo apt-get install python-rosdep

rosdep update

rosdep install --from-paths src --ignore-src -y

catkin_make
```

2 Fichier Launch ROS

```
<!  Frame names  >
<param name="map_frame" value="map" />
<param name="base_frame" value="$(arg base_frame)" />
<param name="odom_frame" value="$(arg odom_frame)" />
  <param name="pub_map_scanmatch_transform " value="true"/>
<!  Tf use  >
<param name="use_tf_scan_transformation" value="true"/>
<param name="use_tf_pose_start_estimate" value="true"/>
<param name="map_with_known_poses" value="true"/>

<param name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)"/>

<!  Map size / start point  >
<param name="map_resolution" value="0.025"/>
<param name="map_size" value="$(arg map_size)"/>
<param name="map_start_x" value="0.5"/>
<param name="map_start_y" value="0.5"/>
<param name="map_multi_res_levels" value="2" />

<!  Map update parameters  >
<param name="update_factor_free" value="0.4"/>
<param name="update_factor_occupied" value="0.9" />
<param name="map_update_distance_thresh" value="0.4"/>
<param name="map_update_angle_thresh" value="0.9" />
<param name="laser_z_min_value" value = " 1.0" />
<param name="laser_z_max_value" value = "1.0" />
<param name="laser_max_dist" value = "3" />


<!  Advertising config  >
<param name="advertise_map_service" value="true"/>

<param name="scan_subscriber_queue_size" value="$(arg scan_subscriber_queue_size)"/>
<param name="scan_topic" value="$(arg scan_topic)"/>

<!  Debug parameters  >
<!
  <param name="output_timing" value="false"/>
  <param name="pub_drawings" value="true"/>
  <param name="pub_debug_output" value="true"/>
 >
<param name="tf_map_scanmatch_transform_frame_name" value="$(arg tf_map_scanmatch_transform_frame_name)"
```