PFE - Rapport Intermédiaire

P9 : Véhicule autonome pour cartographie

Elève : MEJBAR Lina

Encadrants : Alexandre Boé / Xavier Redon /

Thomas Vantroys / Xavier Chenot

18 décembre 2019

I. Introduction et Contexte

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 décembre dans le cadre de mon projet de fin d'études.

A. 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 valorisantes.

Figure 1 : Cariste dans un entrepôt – Robot dans un entrepôt

B. Définition du cahier des charges

Dans un contexte de développement de projets innovants, il m'a été demandé de proposer 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 plaquettes : le but final serait de placer le robot dans leur entrepôt, qu'il puisse cartographier l'environnement, et qu'il puisse déplacer une plaquette 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, ROS, OpenCV,
- **Déplacement intelligent du robot dans l'environnement** - Machine Learning

II. Présentation des technologies

A. 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 le robot **Yahboom** Professional 6WD UNO R3 smart robot kit compatible with Arduino.
Ce kit contient : Une caméra avant, 6 roues et 6 moteurs, une carte Arduino et une carte 6WD, une pile rechargeable.

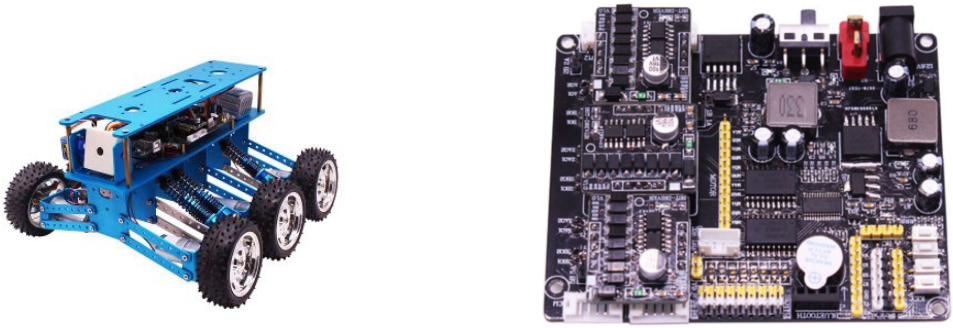
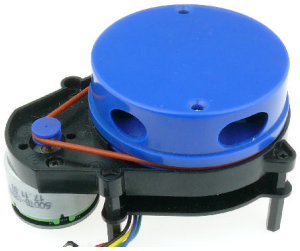


Figure 2: Robot avec sa carte extensible

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 Annexe 1 : Datasheet de la carte)

B. Lidar

Afin de pouvoir cartographier un environnement, il existe plusieurs technologies. Je vais travailler sur un LIDAR de la marque YLIDAR. 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° et peut détecter un obstacle situé jusqu'à 10m de lui. Ci-dessous, une photo de ce dernier.



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 => $Distance = \frac{Vitesse\ de\ la\ lumière * Temps\ écoulé}{2}$

C. 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, notamment ROS.
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.

Spécifications :

- Microprocesseur : Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit à 1.5GHz
- Mémoire : 4 Go
- Connectivité :
  - WiFi 2,4 GHz et 5,0 GHz – IEEE 802.11 b/9/9/n/ac – Bluetooth 5.0, BLE
  - Port Gigabit Ethernet
  - 2 ports USB 3.0
  - 2 ports USB 2.0.
  - GPIO : Connecteur GPIO standard 40 broches (entièrement rétrocompatible avec les cartes précédentes)
- Vidéo et son :
  - 2 ports micro HDMI (jusqu'à 4Kp60)
  - Port d'affichage MIPI DSI pour écran LCD
  - Port caméra MIPI CSI
  - Jack 4 pôles : vidéo composite + audio stéréo
- Multimédia :
  - H.265 (décodage 4Kp60), H.264 (décodage 1080p60, encodage 1080p30), openGL ES 3.0
- Carte SD :
  - Connecteur pour carte Micro SD (système d'exploitation et stockage de données)
- Alimentation :
  - 5V DC via connecteur USB-C (minimum 3A)
  - 5V DC via le connecteur GPIO (minimum 3A)
  - Alimentation par Ethernet (PoE) possible (avec carte PoE HAT en plus)
  - Environnement : Température de fonctionnement 0-50° C

III. Travail effectué

A. Mise en place

➔ 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);
```

qui va définir la part de temps pendant laquelle le pin sera à l'état HAUT durant chaque intervalle (255 correspondra à 100% du temps d'intervalle à l'état HAUT).

➔ Orientation du robot

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

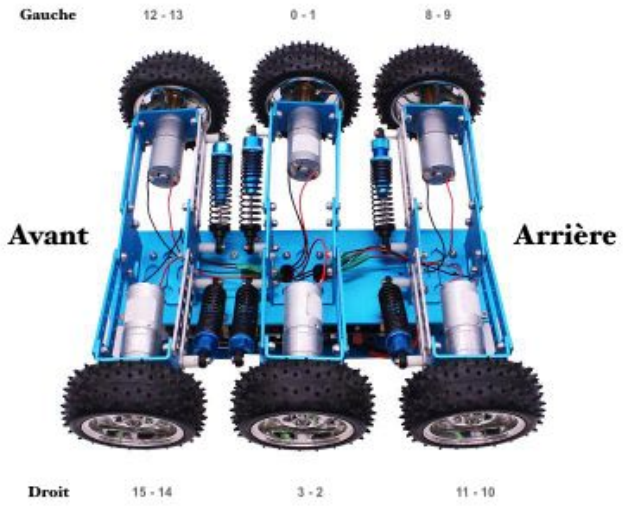


Figure 3 : Définition de l'orientation des roues

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).

➔ Lidar

L'entreprise YLIDAR, propose un logiciel avec le LIDAR X4 qui fonctionne uniquement sous Windows. Ce logiciel permet de visualiser les résultats du scan réalisé avec le LIDAR, avec une visualisation en 2D/3D des points. N'ayant pas de machine sous Windows, j'ai trouvé une autre façon de traiter les données.

L'objectif est de récupérer les données du LIDAR. J'ai modifié le fichier fourni par Ylidar afin d'afficher les valeurs de distances et d'angles. Ce fichier est codé en C++.

Traitement des points

L'affichage du traitement des points s'effectuera sur une interface web. Le but de cette partie est de placer un point en fonction de son angle et de sa distance à partir de l'origine O placé au centre de la page web.

J'ai codé une fonction qui s'en charge. En paramètre, les deux valeurs envoyées par le LIDAR : *distance* et *angle*. 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 page web, définir l'échelle (équivalence entre les pixels de la page et la distance réelle en mètre) et utiliser les formules de trigonométrie : SOH CAH TOA.

Affichage de la première cartographie

Conditions de la capture des données :

- Le Lidar était positionné en hauteur afin d'éviter d'avoir des distances faussées par les différents objets de la salle (Ordinateurs, tours ..)
- Position exacte Zabeth05 (représenté par le point O sur la cartographie)

Cette première version a été réalisée à Polytech dans la salle E306. On reconnaît bien la disposition de la salle mais si ce n'est pas le cas. Voici ci-dessous l'image légendée pour se mieux visualiser la pièce.

Figure 4 : Première cartographie légendée à droite

B. Développement du site web

Maquette du Site

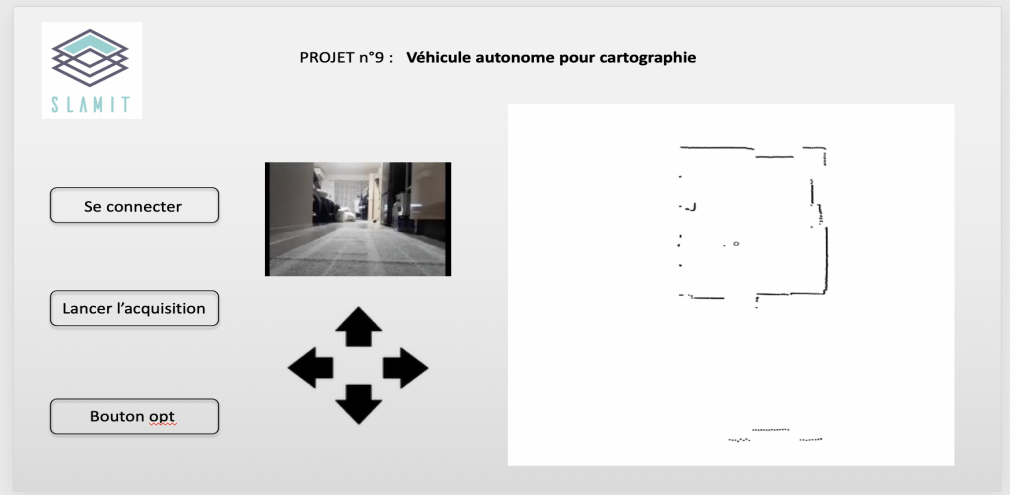


Figure 5 : Maquette du site

Pour le moment je n'ai pas encore travaillé le design du site. J'ai préféré travailler sur son fonctionnement. Voici le site pour l'instant.

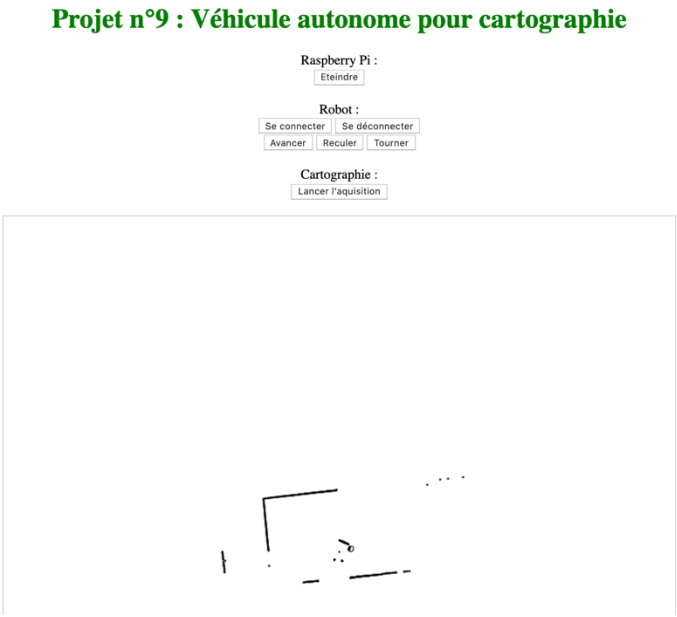


Figure 6 : Site du projet en construction

Fonctionnement

Voici le fonctionnement actuel du site. Une fois les données reçues du Lidar, les valeurs sont écrites dans un fichier JSON (précédemment TXT) qui est récupéré par le script JS qui place les points dans un canvas.

Ci- dessous une explication plus précise.

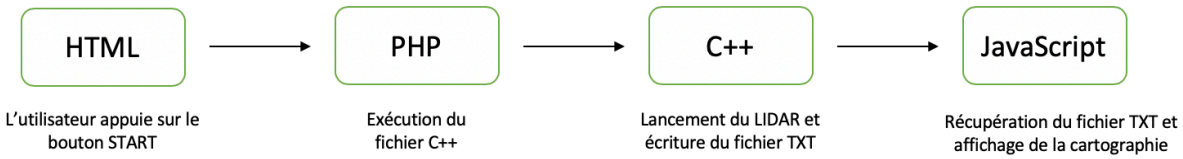


Figure 7 : Fonctionnement du back-end

⇒ Ce cheminement de l'information peut-être améliorer en utilisant GPI qui pourra être mis en place par la suite dans le projet.

Voici le code qui me permet de lancer l'acquisition à distance (depuis mon ordinateur connecté sur le même réseau que la Raspberry Pi) :

Fichier HTML

```
<body>
    <form action="lidar.php" method="post">
        <input type="submit" name="lidar" value="Start"/>
    </form>
</body>
```

Fichier PHP

```
<?php
define('LIDAR_ON','~/Lidar/ydlidar-master/build/samples/ydlidar_test');
 if(array_key_exists('lidar',$_REQUEST)){
   $lidar=$_REQUEST['lidar'];
   if($lidar=="Lancer l'aquisition") exec(LIDAR_ON);
   header('Location: http://192.168.0.20/serial.html');
   exit();
 }
?>
```

Changement de fichier

Voici le format du fichier JSON que j'ai choisi d'utiliser pour le projet :

```
{
  "0" : {
   "angle" : "-180",
   "distance" : "2.08"
  },
  "1" : {
  ...
}
```

Voici la partie du code qui permet d'avoir cette structure :

```
#include <jsoncpp/json/json.h>
file.open("lidar.json");
Json::StyledWriter styledWriter;
Json::Value lidarValues;
// boucle pour chaque valeur reçues
 lidarValues[to_string(i)]["angle"]=to_string(angle);
 lidarValues[to_string(i)]["distance"]=to_string(dis);
```

Contrôle du robot à distance

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.

C. Axes de recherches

1. Open CV

- **Présentation**

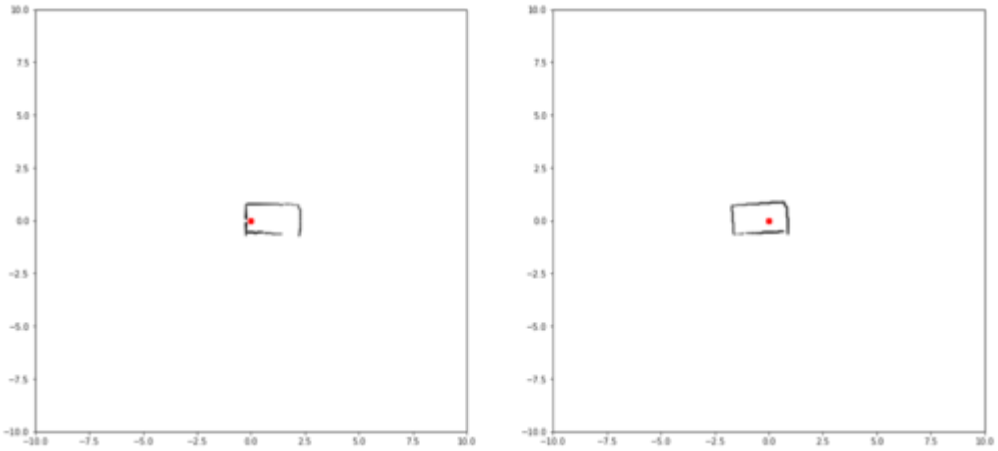
OpenCV (Open Source Computer Vision) est une bibliothèque Open Source qui propose plus de 2500 algorithmes pour le traitement d'images et le machine learning. Ces algorithmes peuvent être utilisés pour détecter et reconnaître des visages, identifier des objets, classifier des actions humaines dans des 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 réel.

- **Mes recherches**

Comme l'affichage des données est fonctionnel sur le site, j'ai cherché une façon d'exploiter mes résultats. J'ai choisi de creuser du côté du langage python parce qu'il possède une bibliothèque graphique libre pour le traitement d'images en temps réel. N'ayant pas de connaissance dans le domaine de Computer Vision, je me suis dans un premier temps fortement, inspiré des codes déjà rédigés sur le site suivant.

Affichage de la cartographie

J'ai codé un premier code, qui permet de récupérer un fichier JSON et de placer les points en fonction de l'angle et de la distance. J'ai gardé le même principe que l'affichage des données en Javascript.



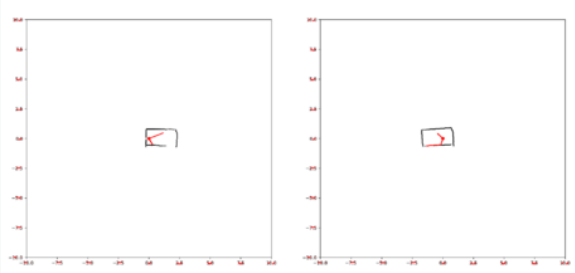
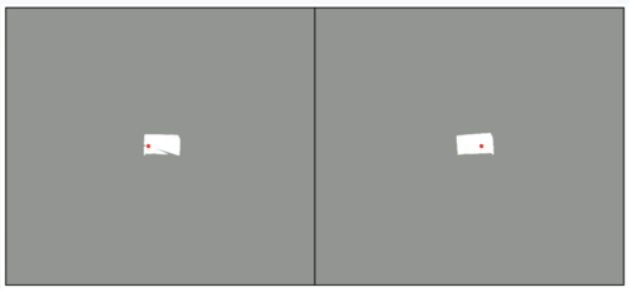
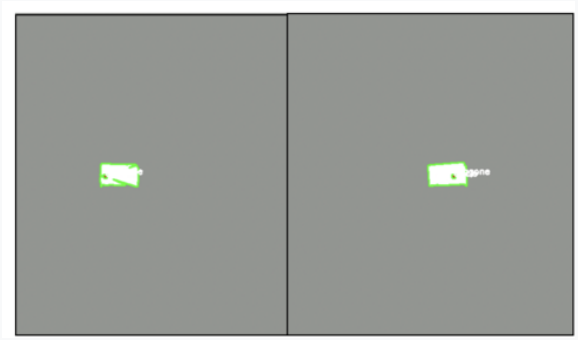
Sur cette photo, j'ai mis en parallèle une première cartographie, puis une seconde après le déplacement du robot dans la pièce. Le but est de pouvoir par la suite comparer deux versions afin de les superposer.

Tracé des murs

Pour tracer les murs, j'ai pensé à :

- Reconnaître la forme de la pièce,
- Relier les points récupérés par le LIDAR,

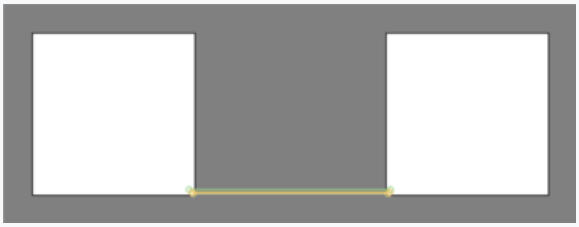
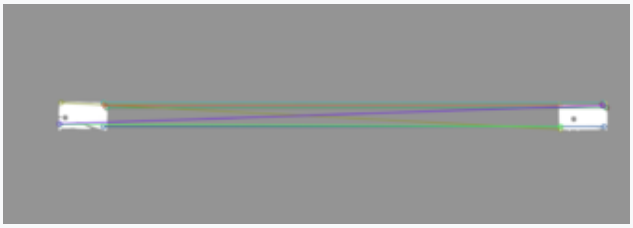
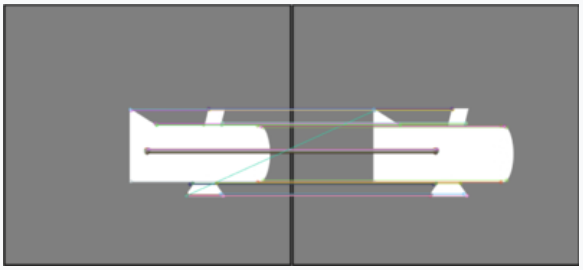
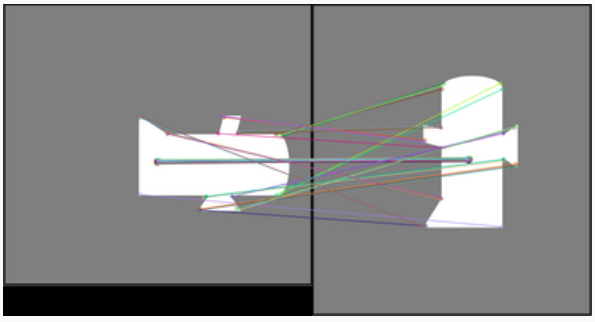
Mais le premier affichage de la cartographie ne donne rien. Python ne reconnaît pas les formes parce qu'il y a des endroits sans information.

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.

Recherche de similitudes

Afin de pouvoir superposer deux cartographies, je dois trouver des keypoints communs aux deux photos. Pour y arriver je vais utiliser un outil disponible en Python qui est SURF, SIRF, et OBO. Pour comprendre comment fonctionne SURF j'ai regardé plusieurs tutoriels sur YouTube.

Voici les résultats observés :

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 !

2. ROS

- 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.

- Mon travail

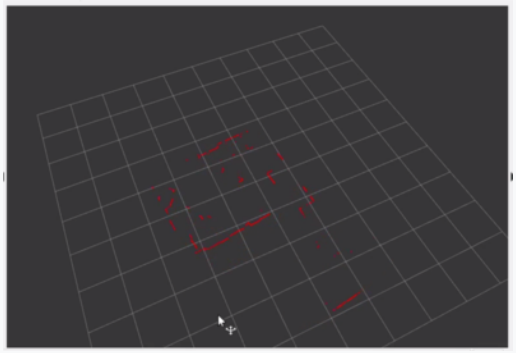
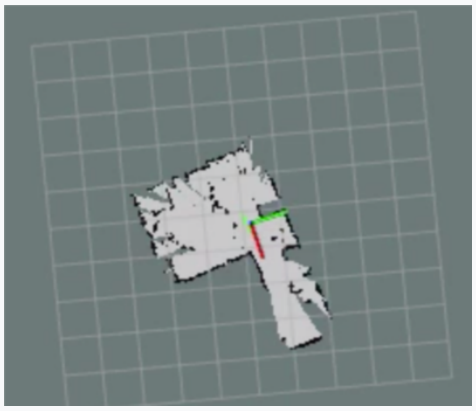
Au départ ayant une RaspberryPi 3, je n'ai pas pu travailler sur ROS, parce que c'est un système trop puissant. Son installation demande nécessite beaucoup de puissance, sans parler de son lancement !
A partir de début novembre, j'ai eu la possibilité de travailler avec une carte Raspberry Pi 4 (4GB 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. Pour mon projet, j'ai besoin du package pour le Lidar proposé par son constructeur.

```
$ rocore
$ roslaunch ydlidar lidar.launch
```

La première commande permet de lancer le noyau ROS, il permet de faire le lien entre les différents topics et noyaux. La seconde permet de lancer le noyau ylidar en exécutant le fichier lidar.launch. Un fichier de type launch est un fichier où l'on définit tous les paramètres et le fichier à exécuter. Ici le fichier lidar.launch exécute le fichier suivant : ydlidar_node.cpp. Il lance le Lidar, récupère les données qu'il publie sur le topic /scan.

J'utilise un autre package hector_slam qui à partir de données sur un environnement, permet de cartographier une pièce.

- Mes recherches

Sans Hector : ROS + YLIDAR		On peut voir que cette version s'apparente beaucoup à ce que j'obtiens sur le site du projet. ROS positionne seulement les points en fonction de l'angle et de la distance... La seule différence non négligeable avec mon programme c'est que le programme tourne en temps réel, donc les points s'actualisent si l'environnement a changé.
Avec Hector : ROS + YLIDAR + HECTOR		Dans cette version, 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.

Après ces recherches, j'ai choisi de me former sur ROS pour apprendre à bien l'utiliser et pouvoir profiter pleinement de tous les outils qu'il offre. J'ai donc suivi deux cours, un premier sur le site Udemy. Dans ce cours j'ai vraiment compris comment fonctionne ROS, j'ai aussi découvert qu'il y avait un outil de communication série :

```
$ rosrun rosserial_arduino make_libraries.py .
```

J'ai aussi découvert qu'il y avait de nombreux packages qui peuvent me servir notamment le ros_webtools.

IV. Travail à suivre

1 . ROS

Pour ce projet, j'arrive à contrôler le robot depuis un site web, et j'arrive à lancer une cartographie. En parallèle, j'ai avancé sur le traitement des données pour l'affichage de la cartographie. J'ai avancé sur deux versions, python avec OpenCv puis ROS. Dès que j'ai eu en ma possession la Rapsberry pi 4, nous avons finalement choisi d'utiliser ROS.
Une fois ROS installé sur la Raspberry Pi, j'ai réussi à contrôler le robot avec ROS en utilisant la liaison série, à afficher la cartographie sur l'interface graphique de ROS.
Pour la suite de mon projet, je dois chercher un moyen pour afficher cette interface graphique sur le site du projet. Je dois continuer à me documenter sur les différents outils disponibles, si je ne trouve pas de package, je vais devoir le coder moi-même.

2 . Bonus : Machine Learning

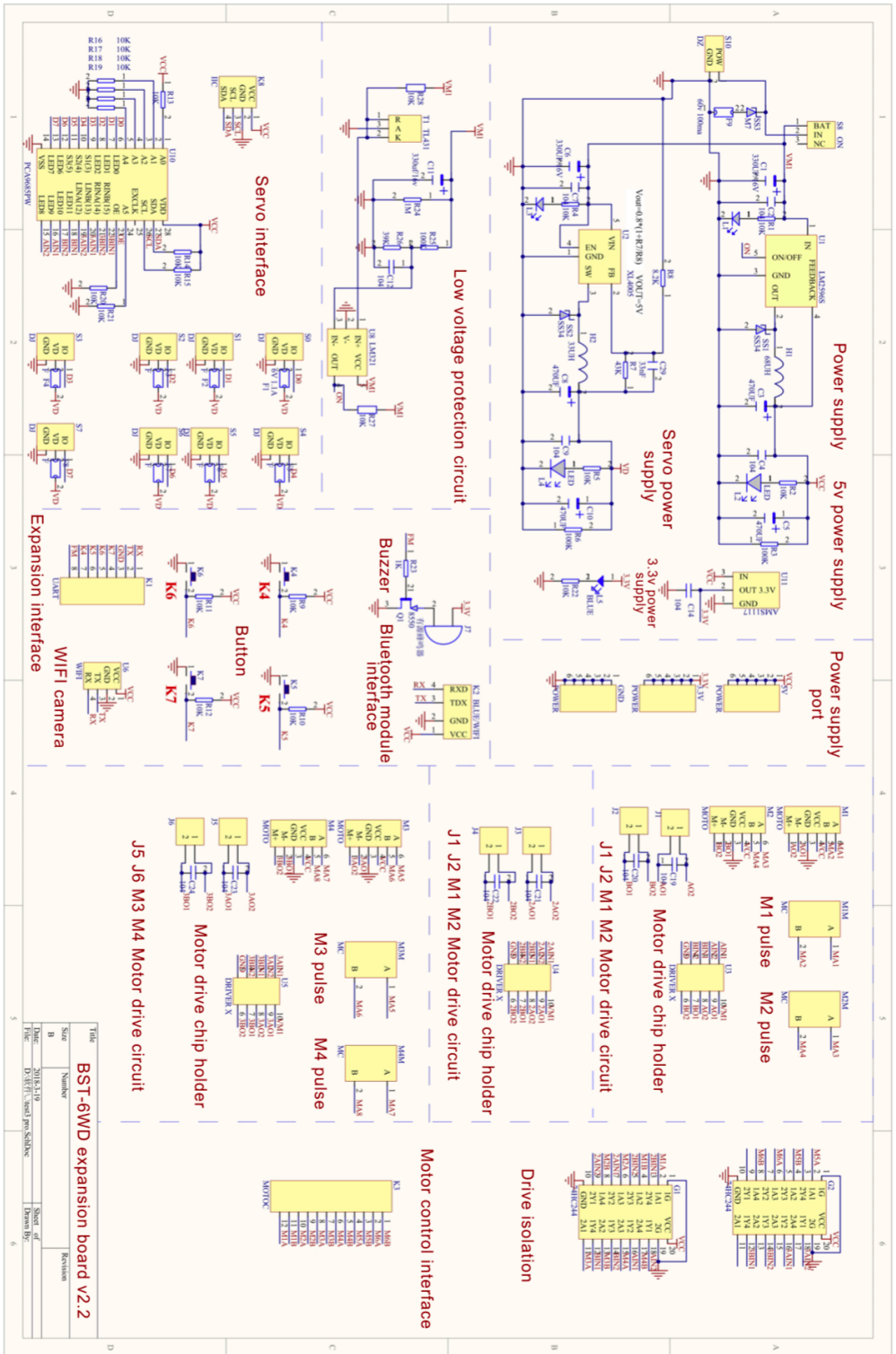
On m'a proposé d'ajouter en fonction de l'avancement de mon projet en mi-janvier/ février du machine Learning. Pour le permettre sur la RP4, ils ont acheté Coral c'est un USB Accelerator. Pour le moment, le sujet n'a pas encore été défini.

Conclusion

Cette première partie de projet, a été très enrichissante 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 est de réaliser un véhicule autonome qui est capable de cartographier son environnement. Pour l'instant, je contrôle le robot à distance, je cartographie son environnement, il me manque l'interaction entre la cartographie et l'utilisateur. En effet, l'objectif final est de pouvoir afficher la cartographie sur le site et de pouvoir cliquer à un endroit de cette cartographie et que le robot s'y déplace.

Annexe



Annexe 1 : Datasheet de la carte d'extension