

Guidage kinesthésique pour l'assistance aux malvoyants Cybathlon VIS 2024

Rapport final

Juin 2023

Projet Industriel d'année 4 Spécialité Robotique

NGUYEN Victoria

FU Daniel

SHLYKOVA Olga

MOREL Théo

Encadrant : Lilian CARILLET

Client : Fabien VERITE

Pilote : Sébastien HINDERER

I-Introduction	3
Contexte	3
Cahier des charges : rappel des principaux éléments	4
II-Architecture mécanique	4
Conception : inspiration du pantographe	4
Modélisation	6
Le robot	6
Fixation de la caméra	8
Porte - PC	9
Prototypage : matériaux et machines utilisés	9
III-Architecture de contrôle des moteurs	11
IV-Architecture informatique : vision et navigation	12
Vision	12
Schéma sous ROS2 :	17
Navigation	17
V- Vis à vis du Cdc	20
VI- Documentation	21
Conclusion	21

I-Introduction

Le présent document a été réalisé durant l'année scolaire 2022-2023 de septembre à juin. Il s'agit d'un projet commandé par Mr F. Verité, chercheur à l'ISIR, encadré par Mr L. Carillet. Nous sommes 4 étudiants en seconde année d'école d'ingénieur spécialité en ROBOTIQUE.

Ce document s'ajoute à une large documentation déjà existante du projet, afin de présenter nos choix et réalisations finales vis à vis de ce projet.



Contexte

Ce projet se déroule dans le cadre du Cybathlon. Le Cybathlon-VIS est une compétition scientifique et technologique, au cours de laquelle des personnes en situation de handicap vont participer à des épreuves, le tout accompagné d'une équipe de scientifiques, démontrant ainsi l'apport des technologies pour les personnes handicapées. En 2024 se tiendra la prochaine édition de la compétition. Pour la première fois, une série d'épreuves est organisée pour des compétiteurs malvoyants/non voyants. Notre projet est de développer une technologie permettant de répondre aux objectifs de la compétition pour remplir les objectifs de l'épreuve.

Parties prenantes



Fabien Vérité
Client



Lilian Carillet
Encadrant



Sébastien
Hinderer
Pilote



Interlocuteurs privilégiés



Samuel Hadjes



Aline Baudry

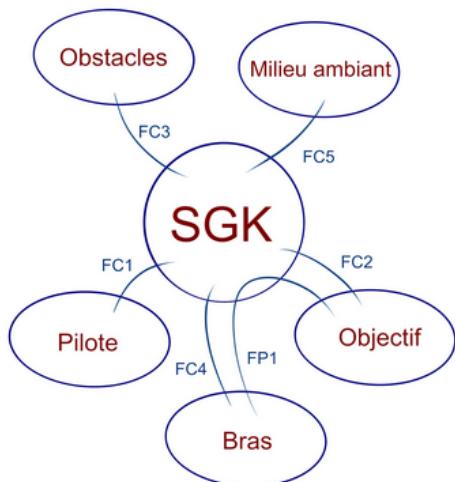


Ludovic Saint-Bauzel

Axel Lansiaux

Pour voir l'épreuve concernée par ce projet, voir Dossier de conception préliminaire, ainsi que les présentations slide.

Cahier des charges : rappel des principaux éléments



Fonctions :

FP1: Guider le bras du pilote pour l'amener vers un objectif

Contraintes :

FC1 : S'adapter au pilote

FC2 : Atteindre un objectif

FC3 : Éviter les obstacles

FC4 : S'adapter au bras

FC5 : S'adapter au milieu ambiant

Phase d'utilisation en fonctionnement

Tableau de caractérisation des fonctions et des contraintes en fonctionnement:

Fonctions	E.M.U*	Critères	Niveaux	Flexibilité
FP1: Guider le bras du pilote pour l'amener vers un objectif	<div style="border: 1px solid black; padding: 5px;"> Bras : Caractéristiques dimensionnelles du pilote (Sébastien Hinderer) </div> <div style="border: 1px solid black; padding: 5px; margin-top: 5px;"> Objectif : Réalisation de l'épreuve du Cybathlon (Annexe 1) </div>	<u>Guider</u> c'est : <ul style="list-style-type: none"> - Exercer une action sur le bras du pilote l'amenant dans une direction déterminée 	<ul style="list-style-type: none"> - Trouver un chemin à travers la piste de l'épreuve Cybathlon grâce à la technologie ROS - Guider le pilote suivant ce chemin - Exercer une force souple/douce 	2 0 0
FC1 : S'adapter au pilote	<div style="border: 1px solid black; padding: 5px;"> Pilote : Caractéristiques définies par le pilote attribué </div>	<u>S'adapter</u> c'est : <ul style="list-style-type: none"> - Sécuriser le pilote à chaque instant - Être conforme à la physionomie humaine - Être conforme aux 	<ul style="list-style-type: none"> - Tension et courant utilisés - Poids : <10kg - Taille et morphologie 	0 3 3

	client	dimensions du pilote attitré	du pilote - Espace de travail 50 x 50 cm	
FC2 : Atteindre un objectif	Objectif :	<u>Atteindre</u> c'est : - Permettre à la personne malvoyante de se déplacer librement comme tout autre personne	- Connaître le trajet optimal dans une pièce	1
FC3 : Éviter les obstacles	Obstacles : Conditions définies par le Cybathlon	<u>Eviter</u> c'est : - S'assurer de la sécurité de la personne vis à vis des obstacles - Enlever la collision avec des obstacles.	- Déetecter chaque obstacle (droite-gauche & devant-derrière) afin de ne pas mettre en danger la personne. - Déetecter chaque obstacle (haut-bas)	1 3
FC4 : S'adapter au bras	Bras	<u>S'adapter</u> c'est : - Transmettre l'information du système au bras du pilote - S'assurer d'une rigidité/flexibilité suffisante	- Transmettre l'information par moyen kinesthésique - Importance d'une structure rigide et flexible	2 1
FC5 : S'adapter au milieu ambiant	Milieu Ambiant : Conditions intérieures définies par le Cybathlon (Annexe 2)	<u>S'adapter</u> c'est : - avoir un système fonctionnel et éviter les dysfonctionnements fréquents	- Importance négligeable étant donné un environnement normal	3

*E.M.U - Élément du milieu d'utilisation

II-Architecture mécanique

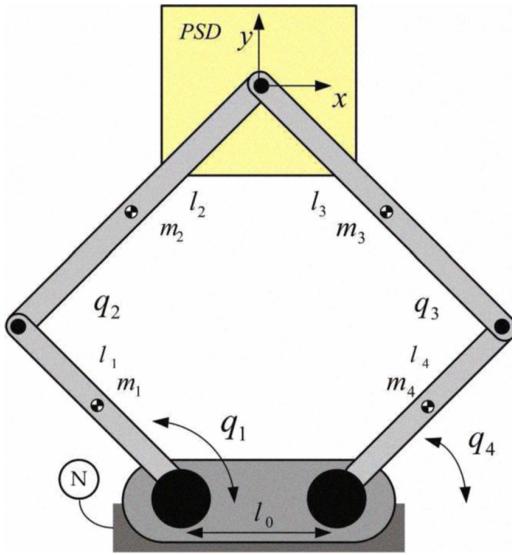
Le cahier des charges précise un guidage kinesthésique, c'est-à-dire via un déplacement d'une partie du corps. Nous avons décidé de nous concentrer sur le bras, plus précisément via un déplacement du coude, choix qui se justifie par la technique de guide d'aveugle qui est une référence. Par ailleurs, le client recherche aussi la notion de feedback positif, c'est-à-dire un moyen de dire au pilote ou aller, et non pas d'où ne pas aller.

Conception : inspiration du pantographe

Le but de la conception à réaliser n'est pas de réinventer une technique de guide de malvoyants, mais plutôt de simuler une technique déjà existante. Ainsi, pour déplacer le coude du pilote, nous allons créer une poignée pour sa main qui va simuler le comportement du bras de son guide.

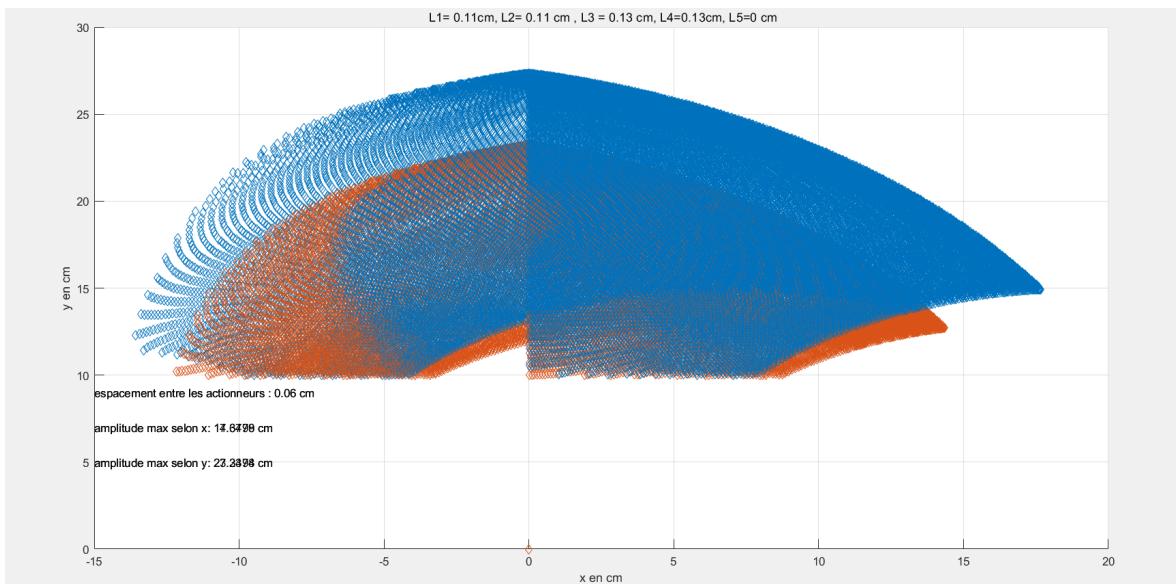
Le robot sera placé sur le harnais porté par le pilote et aura pour objectif de faire un retour clair en position, voire en force, par déplacement du bras de pilote par rapport à son corps.

La problématique que nous avions à adresser était de trouver le moyen d'actionner cette poignée. La solution choisie pour ceci était un mécanisme type pantographe, qui nous permet d'avoir un espace de travail en forme d'éventail - donc très pertinent pour communiquer une direction de mouvement - mais surtout qui nous permet de déplacer les parties les plus lourdes du mécanisme, les moteurs, vers le corps du pilote qui allait le porter.



Exemple du mécanisme mécanisme type pantographe

Pour choisir les dimensions des bras d'un tel mécanisme, nous avons réalisé des simulations Matlab qui nous permettaient de calculer l'espace de travail du robot en fonction des différentes configurations de ces longueurs.



Espace de travail du robot calcule avec une simulation Matlab

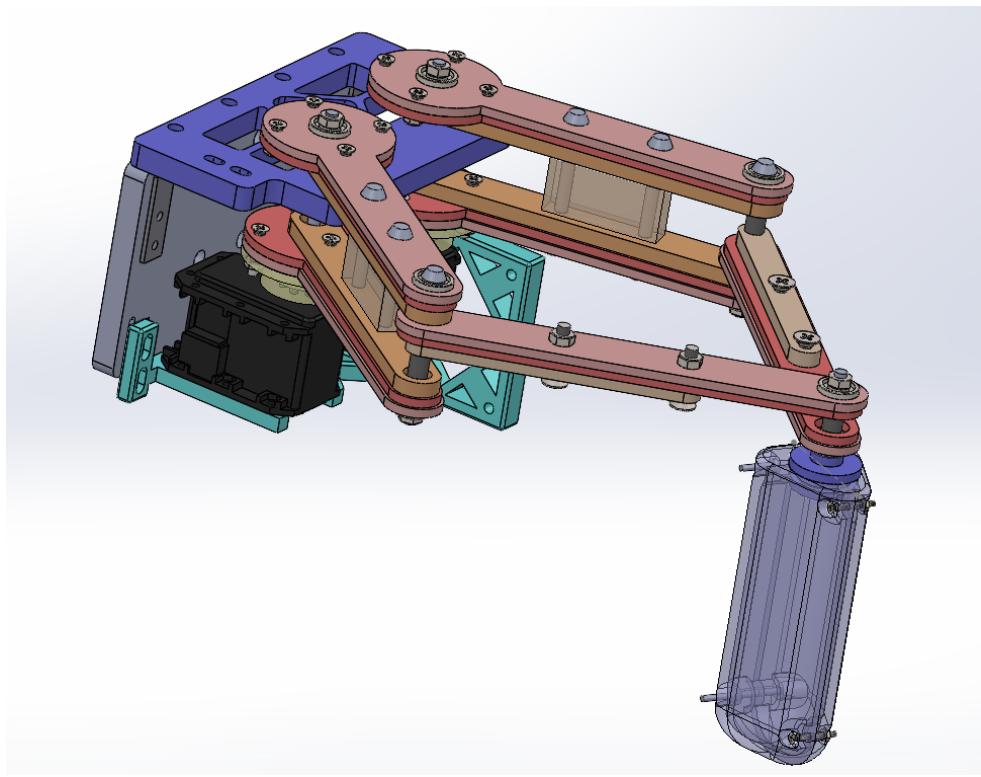
Le compromis le plus optimal entre l'amplitude des mouvements et le couple nécessaire fourni par le moteur sachant que le pilote aussi est susceptible d'exercer une force contraire au mouvement sur la poignée, en premier lieu semblait être pour les longueurs de 13cm pour les bras les plus proches du corps et 15cm pour ceux proches de la poignée. Contrairement à nos attentes, lors des tests effectués avec le pilote, nous nous sommes rendus compte que finalement un espace de travail aussi grand n'était pas nécessaire et nous avons réduit les longueurs à 11cm et 13cm, pour privilégier la rigidité du robot.

Modélisation

Le robot

En respectant les dimensions, nous avons procédé à la modélisation du mécanisme par stratoconception - conception "par couches" - telle que notre robot soit réalisable par découpage.

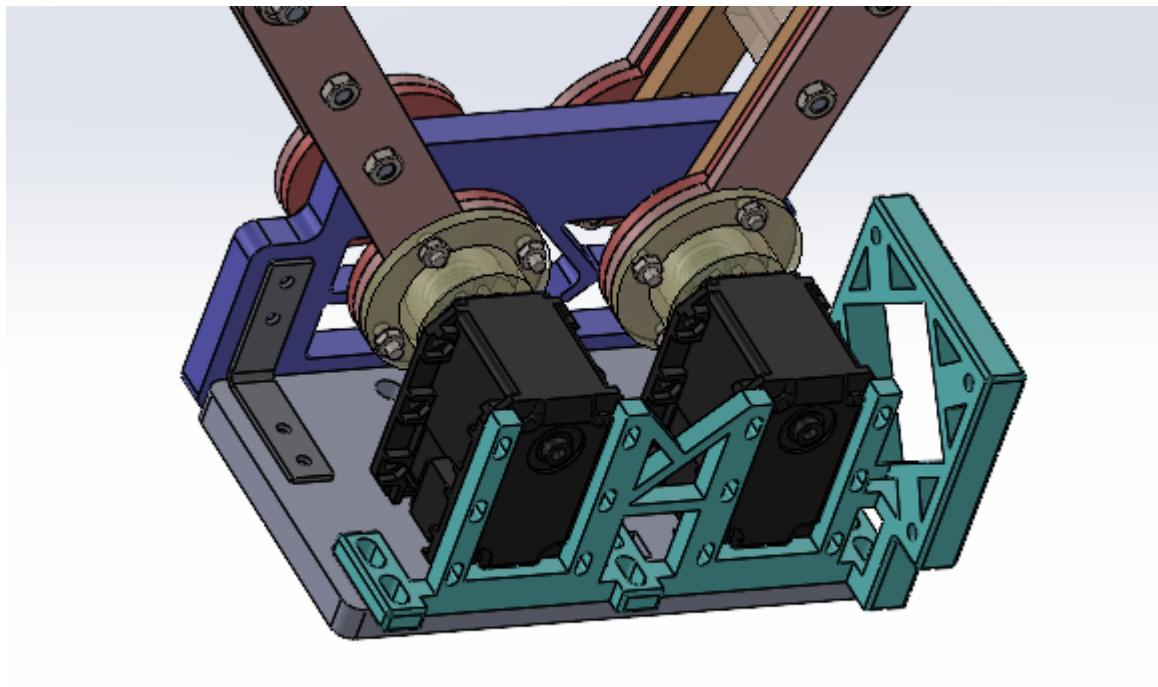
Suite aux tests du premier prototype, nous avons réalisé que la rigidité du mécanisme est très importante lors de la communication avec le pilote. Pour arriver à rendre le mécanisme le plus rigide possible, nous devons pouvoir contrôler les dimensions de tous les composants, et en particulier ceux qui faisaient partie des articulations.



Modèle du mécanisme final

Chacune des bras est réalisée en deux parties en “miroir” dont la distance entre les deux est contrôlée par une pièce qui sera réalisée, elle aussi, par découpage. Ceci nous permet d'avoir une hauteur du modèle moins susceptible au pliage du à son poids et au poids du bras de pilote qu'elle aura éventuellement à supporter.

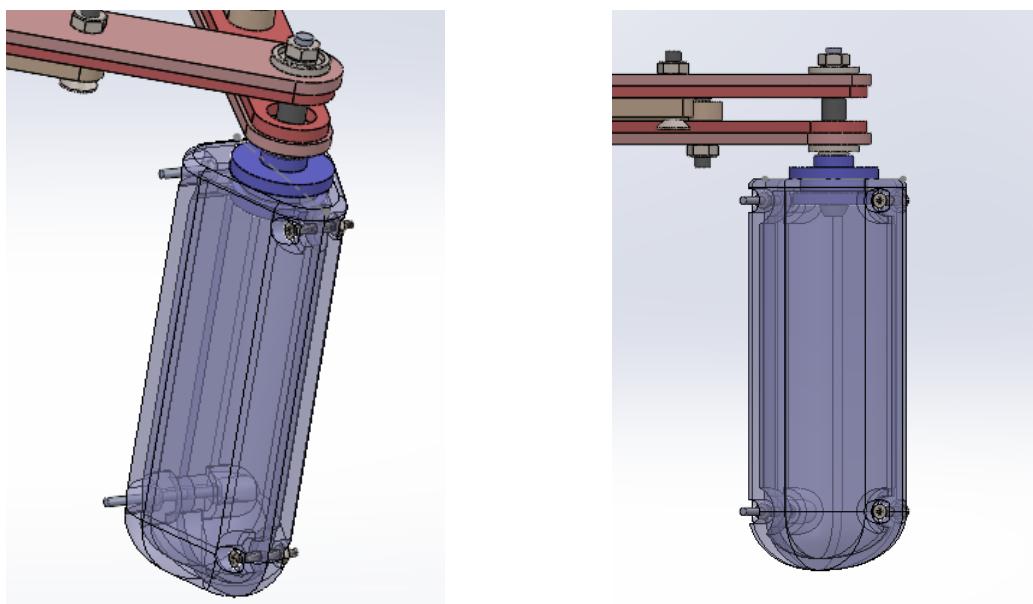
Le support a été conçu pour pouvoir se fixer facilement aux pièces existantes du harnais, mais aussi pour supporter le poids des bras, pour pas qu'il retombe sur les moteurs.



Fixation du robot au harnais

Le modèle prévoit aussi un emplacement pour la carte de commande des moteurs.

Il reste alors la poignée que nous avons modélisée en deux coques. A travers les tests nous avons appris que notre pilote ne s'ensert pas fort la poignée - il la touche presque sans s'y tenir. Alors une forme ergonomique n'est pas forcément utile dans notre cas et une forme simple sera assez efficace.



Modèle de la poignée

Fixation de la caméra

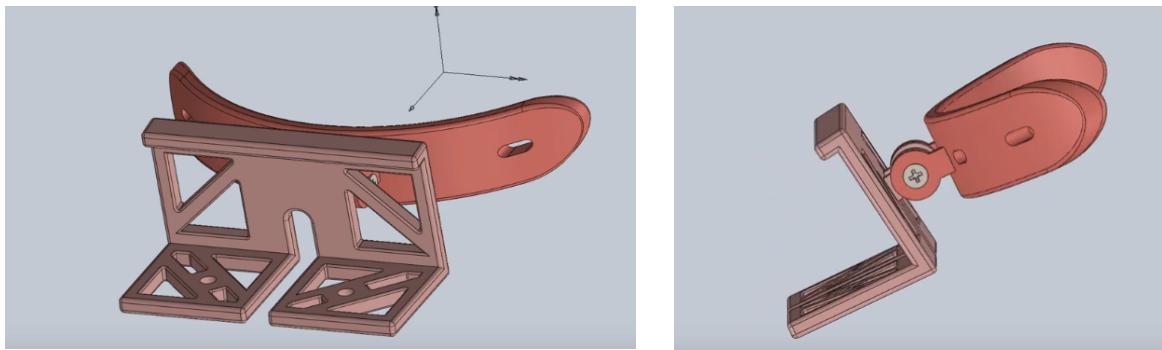
Le robot ne fonctionne pas tout seul, pour pouvoir naviguer, il lui faut des capteurs qui lui permettent de se retrouver dans l'espace. Le capteur avec lequel nous allons travailler est une caméra 3D, que nous avons décidé de fixer dans l'endroit le plus évident, c'est-à-dire la tête du pilote. Depuis son emplacement, elle pourra tout naturellement "remplacer les yeux" du pilote, tout en ayant la meilleure vue sur son environnement.

Pour simplifier la tâche, nous avons récupéré une visière depuis laquelle nous avons récupéré sa fixation intérieure. Il suffit donc maintenant de faire la liaison entre elle et la caméra.



La visière de sécurité

D'autre part, il faut rendre l'angle de vue de la caméra modifiable. Alors, nous avons mis la caméra sur un pivot qu'on pourra serrer et desserrer pour ajuster l'angle entre la caméra et le sol.



Modèle de la fixation de la caméra à la visière

Porte - PC

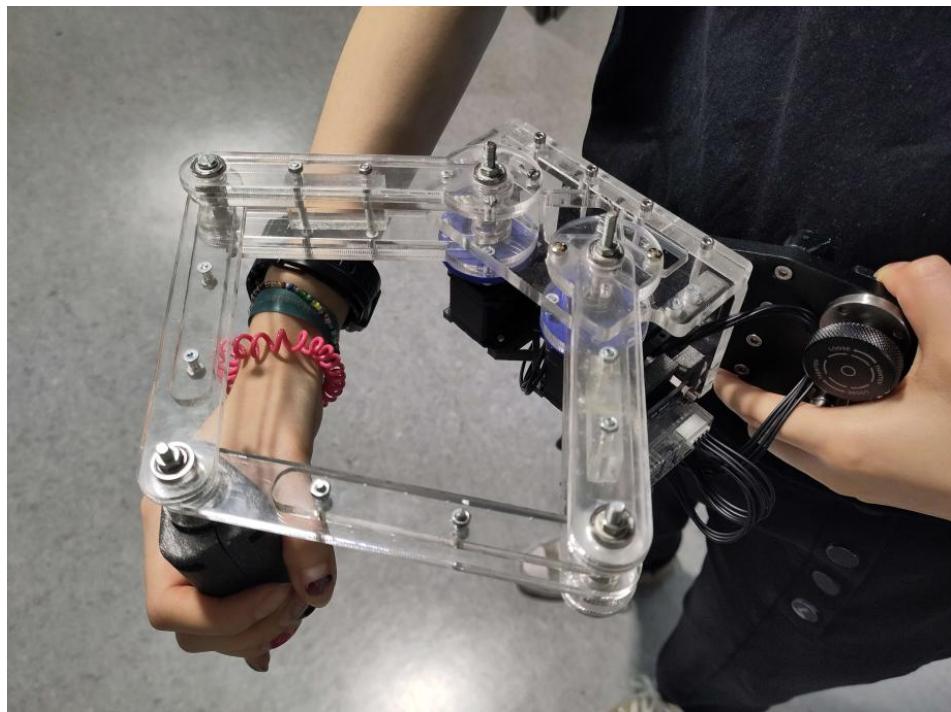
Prototypage : matériaux et machines utilisés

Nous avons marqué précédemment que nous avons modélisé nos pièces de manière à ce qu'elles soient réalisables par découpage. Ce découpage nous avons donc réalisé dans du plastique PMMA par découpe laser.



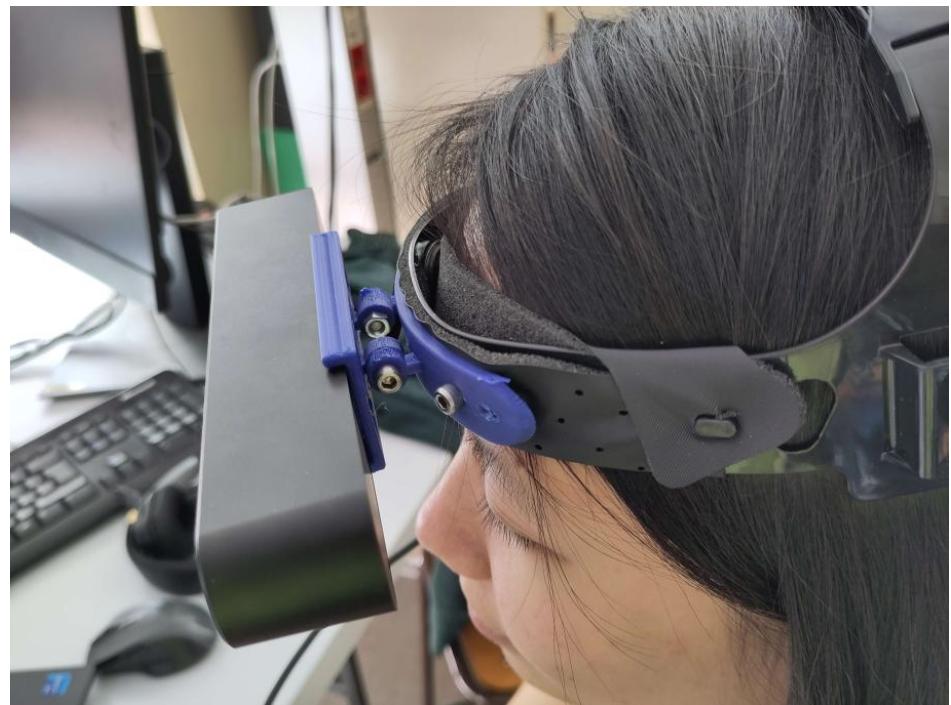
Pièces de la fixation au harnais découpées, avec une pièce d'origine (à droite en noir)

Les articulations ont été réalisées avec des roulements, dont la fixation a été contrôlée avec des entretoises coupées avec précision sur un tour à métaux.



Le prototype assemblé

La poignée, ainsi que la fixation pour la caméra ont été réalisées par impression 3D.



Assemblage de la fixation pour la caméra

III-Architecture de contrôle des moteurs

Nous utilisons des moteurs Dynamixel MX-28 qui sont robustes et souvent utilisés dans de la robotique.



Ces moteurs servent à amener l'effecteur du pantographe à une position que l'on souhaite. Ils peuvent effectuer une rotation de 360°, ce qui correspond à une plage de position moteur, c'est-a-dire que l'angle 0° correspond à une position moteur de 0 et 360° correspond à une position moteur de 4000.

a) Commande des moteurs

On peut commander les moteurs de 2 manières :

- Un mode avec une "goal" position en entrée
- Un mode avec une vitesse constante en entrée

Pour le 1er mode, il suffit de mettre une position moteur à atteindre en commande pour pouvoir les contrôler. Le problème de ce mode est que la vitesse de rotation est élevée et ne permet pas de récupérer la moindre information pour le pilote. Pour y remédier, nous avons effectué une interpolation entre la position initiale et la position finale pour avoir une multitude de points intermédiaires avec chacun leur propre "goal" position.

Pourquoi n'a-t-on pas gardé ce mode ?

Ce mode a bien effectué ce que l'on attendait, mais il s'est avéré que le mouvement n'était pas fluide et accompagné d'à-coups.

Pour le 2eme mode, on commande les moteurs grâce à une commande de vitesse constante en entrée, ce qui va effectuer une incrémentation de position moteur de manière infinie, jusqu'à ce que l'on donne une entrée nulle pour que les moteurs s'arrêtent. Avec ce mode, on a réussi à avoir un mouvement fluide, mais il faut maintenant que les moteurs s'arrêtent au bon moment.

Comment arrêter le moteur à la bonne position ?

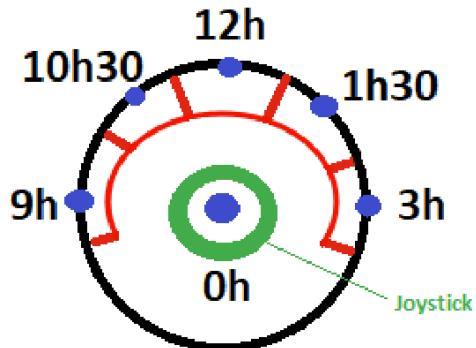
On récupère la position moteur actuelle et on la compare avec la position moteur que l'on souhaite atteindre avec une petite incertitude. On répète en boucle cette condition jusqu'à y arriver et arrêter les moteurs.

b) Contrôle manuel via la manette

Nous avons opté dans un premier temps pour un contrôle manuel du pantographe pour pouvoir y effectuer nos tests, notamment ceux des moteurs. C'est notamment grâce à la manette que l'on a pu choisir la méthode adéquate pour le contrôle des moteurs.

Comment contrôler le pantographe ?

Nous contrôlons le pantographe à l'aide du joystick gauche de la manette. Nous avons prédefini 6 positions possibles de l'effecteur (points bleus) en fonction de la position du joystick (le cercle noir représente la délimitation du joystick).



Chaque horaire est délimité par un cadran pour avoir une marge sur la position du joystick. Si le joystick ne se trouve dans aucun cadran, alors le pantographe reste au centre.

IV-Architecture informatique : vision et navigation

Vision

Nous utilisons une caméra ZED 2 de Stereolabs. [Site](#) Il s'agit d'une caméra stéréoscopique intelligente et équipée d'une centrale inertielles, permettant d'effectuer une cartographie en profondeur de son environnement d'une part, et de s'y localiser (position + orientation) d'autre part. Autrement dit, elle permet de faire tourner un SLAM.



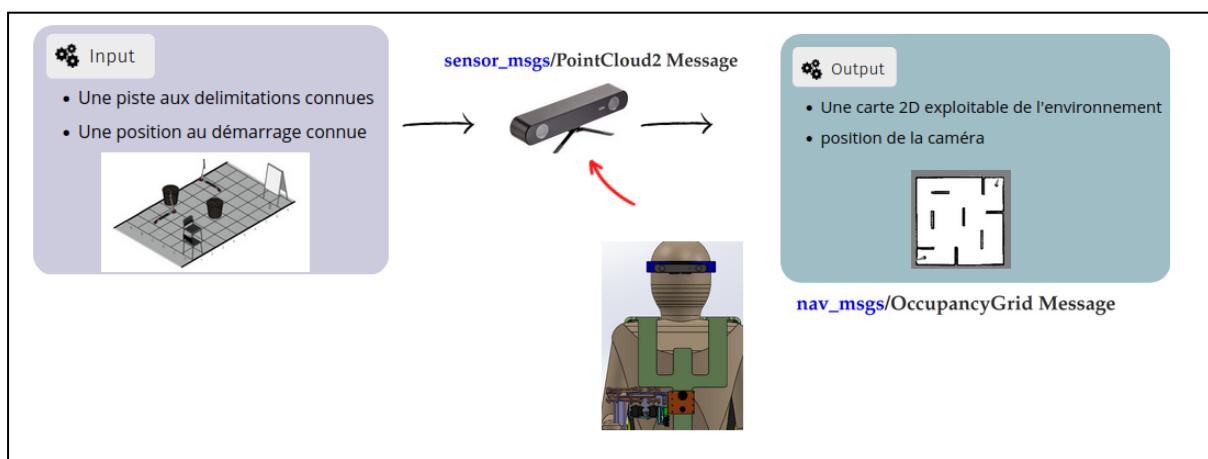
Nous utilisons cette caméra sous ROS2 Humble. Ainsi, il faut installer le wrapper de la caméra (voir le README), nous permettant de récupérer dans les topics les informations qui nous intéressent.

Le but de cette partie est d'aborder en détail l'automatisation du retour kinesthésique demandé dans le cadre de ce projet.

Qu'attendons nous de cette caméra ?

L'input constitue la piste aux dimensions connues. On suppose que la position d'origine est connue : le pilote démarre la course au milieu de la ligne de départ et est face à la ligne d'arrivée. La caméra, placée sur la tête du pilote, effectue une cartographie 3D et on attend en sortie une carte en 2D, une occupancy grid, qui représente l'environnement scanné projeté.

Ce qu'on attend de la caméra :



La caméra permet de cartographier son environnement , et le représente sous forme d'un nuage de points. Il s'agit d'une représentation discrète, et est soumise au bruit extérieur. On procède alors à une première étape de filtrage.

Pourquoi décide-t-on de passer en 2 dimensions ?

Car le guidage se fait avec 2 degrés de liberté, sur le plan de la piste. Il n'est pas nécessaire de savoir à quel point l'objet est grand pour pouvoir l'éviter. On espère seulement avoir sa position par rapport au pilote et quelle place il occupe pour pouvoir le contourner.

Par ailleurs, faire de la navigation en 3 dimensions est a priori plus lourd et donc plus lent que de la navigation en 2 dimensions.

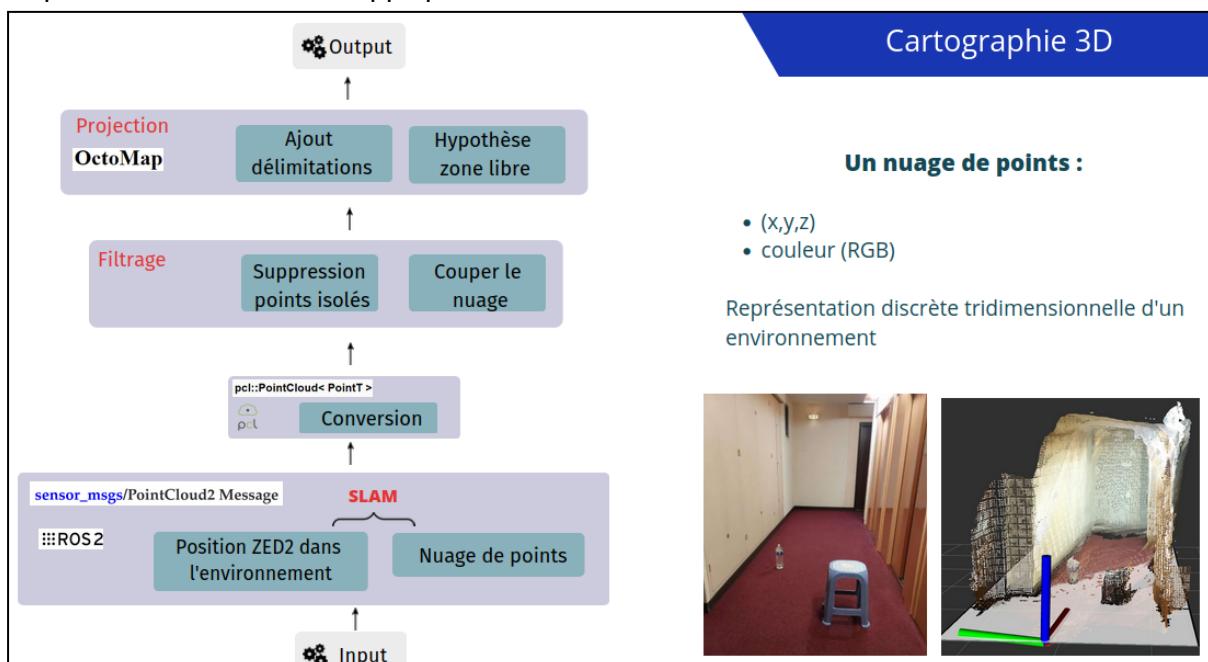
Notre algorithme sur la caméra

Dans un premier temps, la caméra placée sur la tête du pilote, et via son mouvement, effectue la cartographie 3D mentionnée précédemment, qui est représentée et transmise dans ROS, sous la forme d'un nuage de points.



Alors, l'objectif étant d'avoir en sortie une projection 2D exploitable pour de la navigation, il nous faut un nuage de points lisible. On a donc procédé à une première étape de filtrage 3D. Le filtrage s'opère via des fonctions issues de la bibliothèque [PCL](#).

Ainsi, le nuage de points venant de la ZED 2 et de ROS2, il nous est nécessaire de faire une étape de conversion afin d'appliquer des filtres.



Transformation en octree

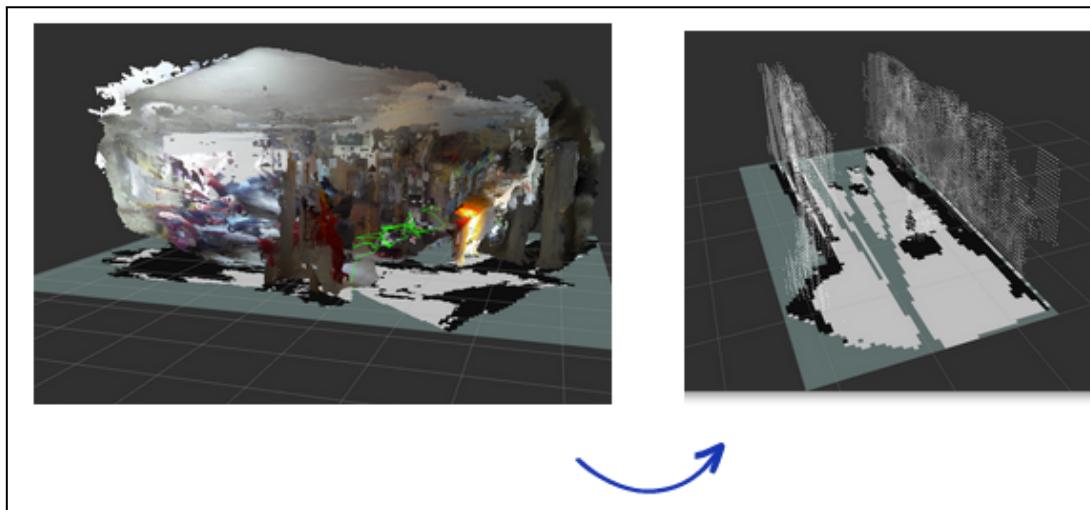
Octo Map est une bibliothèque permettant de faire du traitement de nuage de points et de rendu de carte 3D. Elle effectue notamment une représentation probabiliste, sous forme d'octree, représentation également discrète de l'environnement sous forme de voxel. Cette représentation nous est utile pour obtenir en sortie d'algorithme une projection 2D de l'octree, et d'avoir un message de type occupancy Grid à envoyer, via un topic ROS2, à la partie navigation explicité plus bas.

Filtrage

Nous avons, dans le code source d'octomap_server, rajouter des lignes de code permettant de premièrement couper le nuage selon les 3 dimensions : en effet la caméra voit beaucoup

plus loin et plus large que nécessaire par rapport aux dimensions connues de la piste. Nous avons également coupé le sol et le plafond, car ils nous gêneraient lors de l'étape de la projection 2D.

On rappelle que les dimensions de la piste sont de 3 X 5 mètres.



Par la suite, nous faisons un filtrage médian pour enlever les points ayant une densité trop faible par rapport au seuil fixé. Ils sont du bruit dus à l'éclairage ou autre.



Obtention de l'occupancy grid

Une fois le filtrage 3D effectué, octomap_server projette l'octree sur un plan, permettant d'obtenir une occupancy grid. Il s'agit d'une représentation 2D où chaque pixel a une valeur entre 0 et 100 correspondant à la probabilité qu'elle soit occupée ou non. Le résultat est observé ci-dessus.

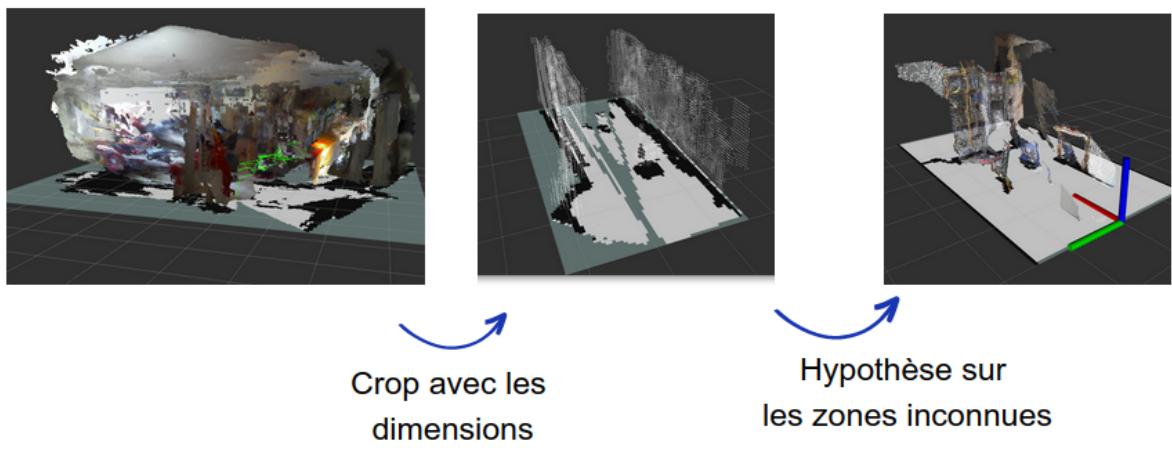
On constate que la carte n'est pas encore exploitable pour l'algorithme de navigation : il n'y a pas de délimitations. Il y a des zones inconnues, pouvant éventuellement poser problèmes pour la navigation.

Ajout des délimitations

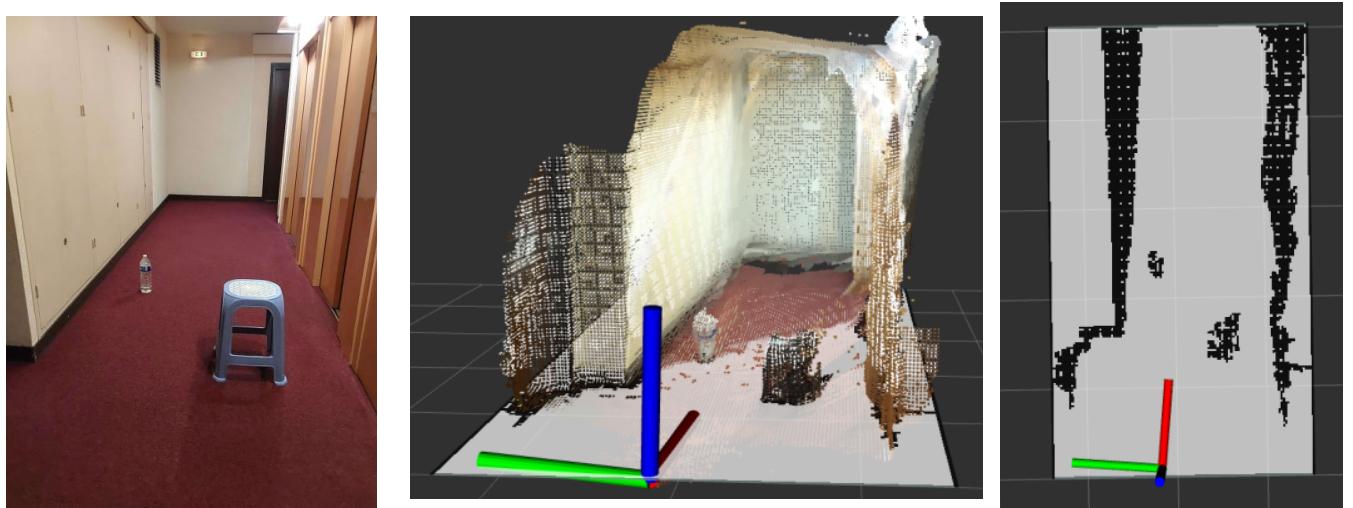
Toujours dans octomap_server, on ajoute des délimitations manuellement , qui représentent les obstacles imaginaires a ne surtout pas franchir. Pour cela, on place la valeur des pixels de la longueur à 100, correspondant à la valeur occupée.

Hypothèses zones inconnues

Enfin, nous décidons d'avoir une carte complète sans zones inconnues. Pour cela , nous faisons donc l'hypothèse qu'une cellule inconnue (non vu par la caméra) correspond à un espace libre. On estime que le déplacement du pilote permettra de rafraîchir l'état de chaque cellule.



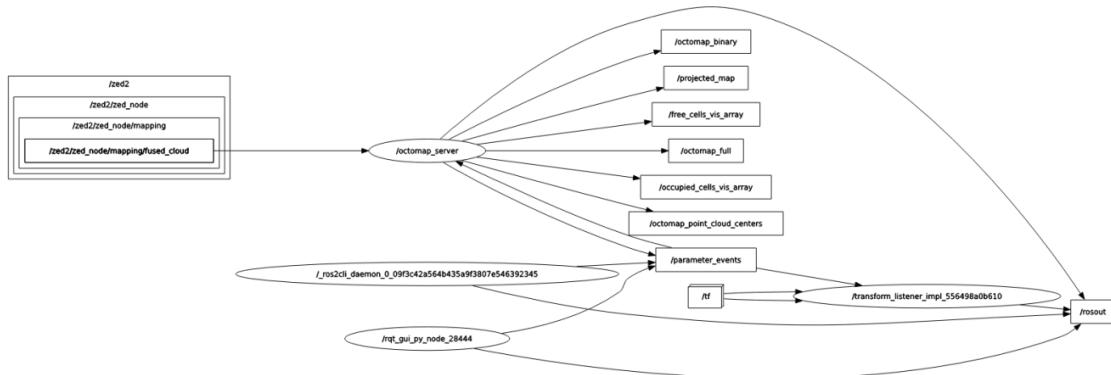
Le résultat sur des tests unitaires sont montrés ci-dessous avec deux obstacles : un tabouret et une bouteille d'eau transparente. On constate que les 2 obstacles sont bien reconnus :



Nos tests sont très satisfaisants pour l'utilisation comme on le voit sur l'image : bouteille et chaise sont bien détectées.

Une autre idée aurait été de faire un filtrage 2D pour contrer l'effet des obstacles parsemés , qui selon l'algorithme de navigation utilisé, aurait pu déstabiliser la suite.

Schéma sous ROS2 :



Ci dessus le schéma de la partie caméra sous ROS2.

Navigation

Maintenant que nous connaissons la position des différents obstacles sur la piste, grâce à l'occupancy grid, il nous reste à déterminer le chemin optimal à faire suivre au pilote pour atteindre la position finale sans toucher aucun obstacle.

Pour cela, nous avons pu tester 3 algorithmes de navigation différents et les comparer afin de choisir celui correspondant le mieux possible à nos demandes.

a) Champs de potentiel

La première idée est d'adapter un algorithme qui simule le comportement d'une particule dans un champ de potentiel. Dans cet algorithme, les obstacles présentent un potentiel négatif, qui repousse la particule, et la position ciblée présente un potentiel positif qui, au contraire, attire la particule.

Les paramètres, tels que les rayons d'influence des différents champs, leur force et le rayon de la particule étant modifiables, on adapte cet algorithme au travail dans un noeud ROS au sein de notre système.

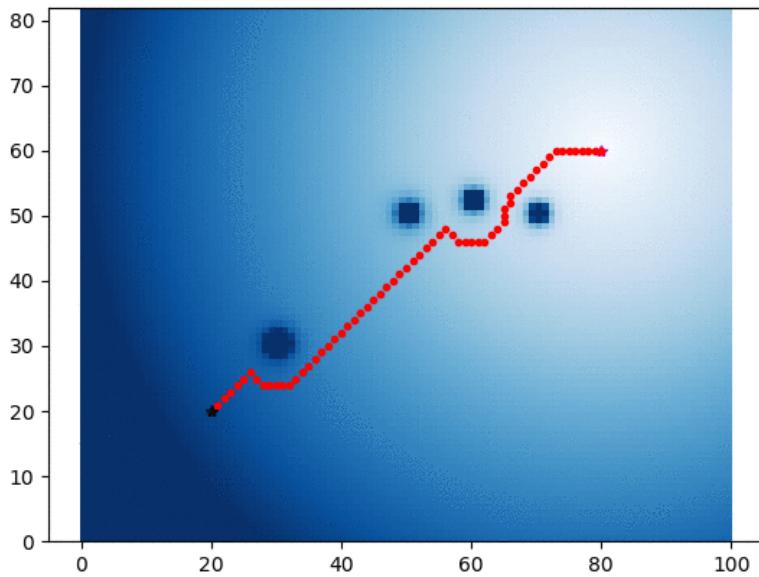


Illustration du fonctionnement de l'algorithme du champ de potentiel. Le potentiel négatif est présenté en couleurs foncées, tandis que le potentiel positif est présenté en couleurs claires. La particule trouve alors son chemin vers la position cible.

Ayant implémenté cet algorithme, nous nous sommes rendus compte que c'est un outil très puissant qui n'est pas du tout adapté à nos besoins. Premièrement, un champ négatif présente un obstacle. Or, nous ne savons pas détecter les obstacles séparément. Notre entrée en données est une projection en 2D d'un nuage des points en 3D. En donnant à chaque point un champ de potentiel, le calcul devient très long - d'autant plus long que le nombre de ces points et la distance entre la particule et la cible - ce que n'est pas adapté à une épreuve de durée maximale d'une minute.

b) Algorithme de Dijkstra

Le fonctionnement de l'algorithme de Dijkstra est de tracer un cercle, autour de la position actuelle (position de départ de l'algorithme), d'un certain rayon prédéfinie, initialisé à la taille du pilote (environ 0.5m). Chacun des points de ce cercle qui n'ont pas été visité avant sont considérés comme des voisins et tant que le point d'arrivée n'est pas un point voisin, le rayon et on recalcule les nouveaux points voisins. Une fois que nous avons trouvé le point d'arrivée, nous connaissons désormais le chemin optimal et donc la direction à suivre pour atteindre cette position.

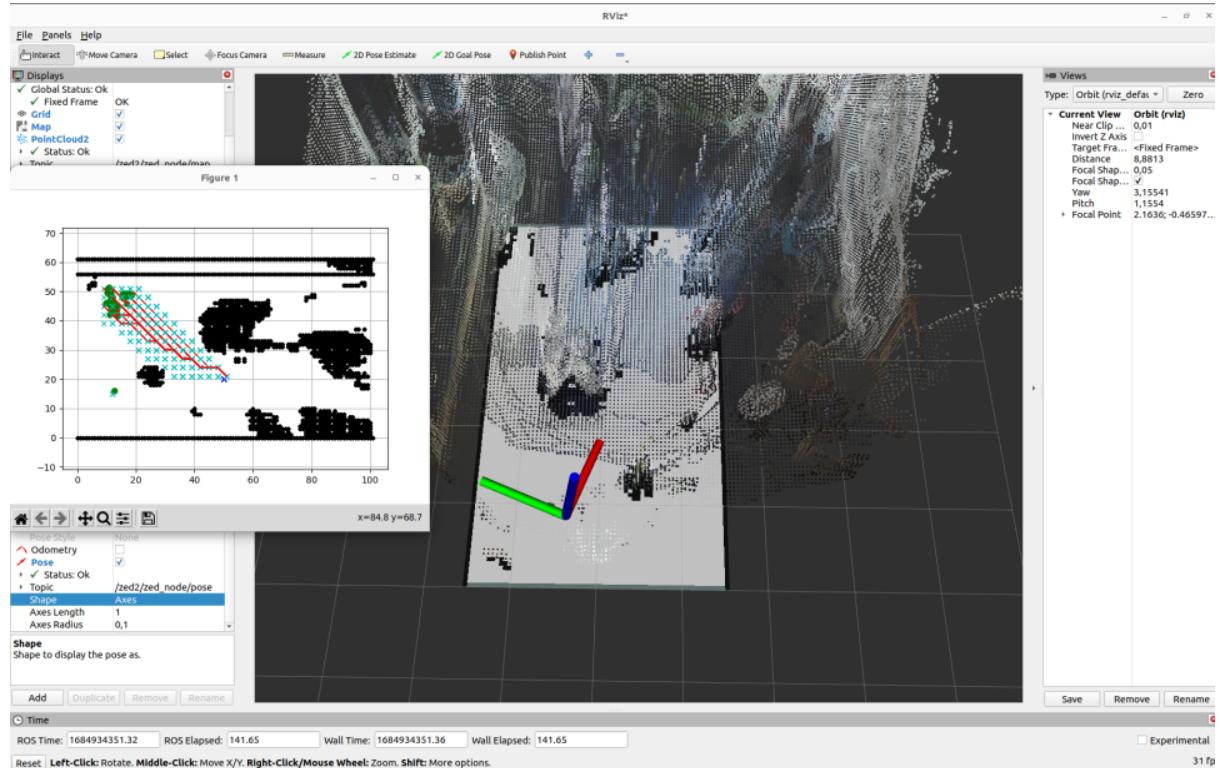
Après avoir réalisé plusieurs tests d'efficacité de l'algorithme, nous remarquons que le temps de calcul pour obtenir la prochaine direction à suivre est d'environ 1,6s pour une occupancy grid prédéfinie. Nous avons réalisé des tests avec la même occupancy grid sur les autres algorithmes afin de pouvoir les comparer entre eux.

c) Algorithme A*

Nous remarquons que le temps obtenu précédemment avec l'algorithme de Dijkstra peut être améliorable. En effet, le chemin le plus court pour aller d'un point A à un point B est la

ligne droite reliant ces deux points. Si, à la place de regarder tous les points à l'intérieur d'un cercle, nous regardons les points dans la direction de la droite la plus courte pour aller de A vers B, beaucoup moins de points seront regardés et donc le temps de calcul sera plus court. Cependant un temps de calcul plus long sera nécessaire si le chemin optimal n'est pas dans la direction de la droite, mais cela reste négligeable.

En réalisant le même test que pour l'algorithme Dijkstra, nous obtenons un temps de calcul de 0,8s environ correspondant à une amélioration de 50% du temps de calcul qui reste non négligeable lorsque l'on souhaite faire du temps réel.



Ci-dessus une photo de comparaison entre l'algorithme A* et Octomap.

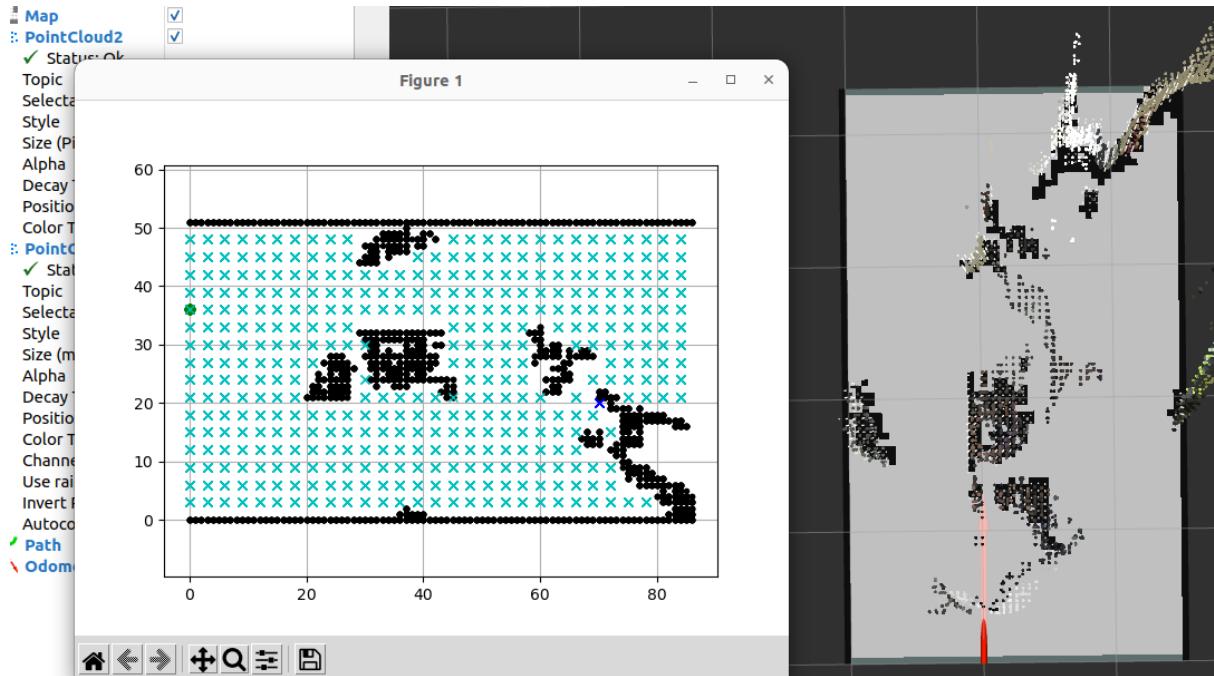
d) Algorithme D*

Nous avons aussi fait le test avec l'algorithme D* mais donnant des temps de calcul beaucoup trop important pour faire du temps réel, nous n'avons donc pas gardé cet algorithme.

Notre choix d'algorithme porte donc sur l'algorithme A* possédant les meilleures performances.

Nous pouvons maintenant regrouper les différentes briques afin d'effectuer les tests finaux et de corriger les éventuels problèmes que nous pouvons remarquer. Tout d'abord, nous avons pu apercevoir que lorsque le pilote sortait de la zone définie dans Octomap de 5m par 3m (zone de l'épreuve), l'algorithme de navigation ne trouve pas de chemin possible pour

revenir sur la piste ni pour atteindre l'objectif, ceci reste un problème mineur comme de toute façon le pilote doit impérativement rester dans les limites de la zone. De plus, la communication entre la navigation et le déplacement des moteurs comprend un léger retard et envoie dans la direction au pilote 1 à 2 secondes plus tard que prévu.



La figure ci-dessus montre le problème de l'algorithme qui ne renvoie aucune réponse lorsque le pilote se trouve hors des limites de la piste.

VI- Documentation

Les codes, fichiers CAO, mises en plans et autres sont disponibles sur [gitlabSU](#). Un readme est présenté et permet de prendre en main toutes nos ressources.

Conclusion

Ce rapport final liste toutes nos démarches et notre analyse rétrospective sur notre travail. Le cahier des charges est disponible pour lecture.