Rapport de stage pour stage ingénieur de recherche

SLAM avec données peu denses en milieu karstique

Superviseur : Simon Rohou

mathis.le_scoezec@ensta.fr

Remerciement

Je tiens à remercier mes 2 encadrants de stage, Simon Rohou et Lionel Lapierre, qui ont toujours été là pour répondre à mes questions.
Je tiens également à remercier Gabriel Betton pour l'aide qu'il m'a apportée sur la partie simulation.

Sommaire

I - Avant-propos

Ce document est un rapport de stage concernant le SLAM avec des données peu denses en milieu karstique. Mon stage n'étant pas obligatoire, ce rapport a surtout pour but d'aider à la compréhension de mes travaux pour quiconque s'intéresserait à ces derniers.
Ce rapport se concentre donc uniquement sur le sujet de stage et n'est pas aussi formel qu'un rapport classique.
Je pars également du principe que le lecteur a une connaissance suffisante du lexique et des concepts employés.

Vous pouvez trouver tous les codes que j'ai réalisés dans le cadre de ce stage dans ce GitLab : *https://gitlab.ensta-bretagne.fr/lescoema/stagem1#*

Je précise sous chaque titre quels dossiers ou fichiers contiennent les codes en rapport avec cette section.

II - Sujet de stage et problématique

Mon stage fait partie d'un sujet de recherche plus grand qui est l'exploration des karsts par des robots sous-marins autonomes (Robots for Karst Exploration).

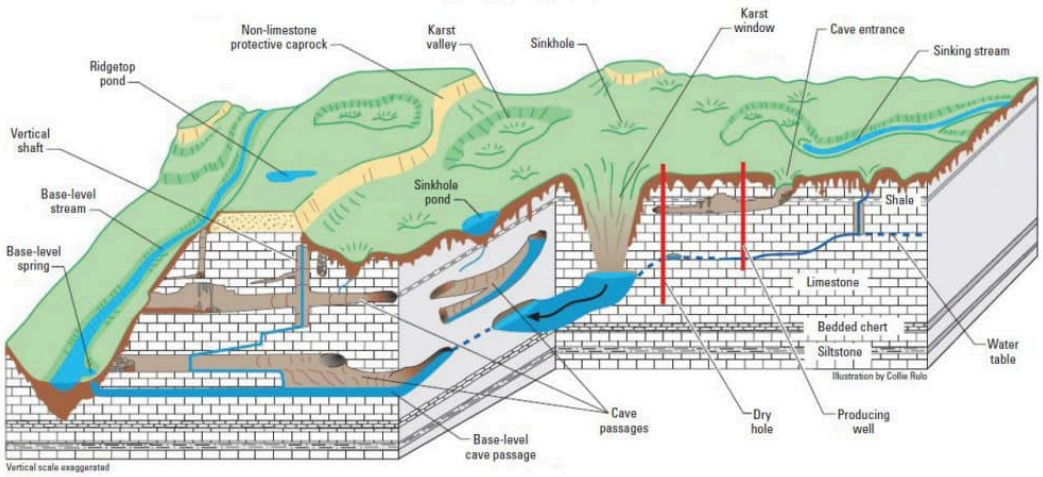


fig II.1 : Schéma d'un karst

Pour le moment (été 2025), une seule mission d'exploration a eu lieu dans les Fontaines de Nîmes. Le robot utilisé lors de l'expérimentation était équipé de capteurs basiques : un AHRS, un DVL et deux sonars à 360 degrés.

Comme nous n'avons pas de capteurs de positions absolues (dead reckoning), implémenter un algorithme SLAM devient indispensable si l'on veut que la position estimée du robot reste fiable. On a d'ailleurs pu constater des erreurs et un drift dans la position estimée par les capteurs lors de l'expérimentation à Nîmes. Ce drift vient principalement de l'erreur de mesure du yaw du robot par le magnétomètre. Une hypothèse est que cette erreur est causée par le magnétisme des moteurs.

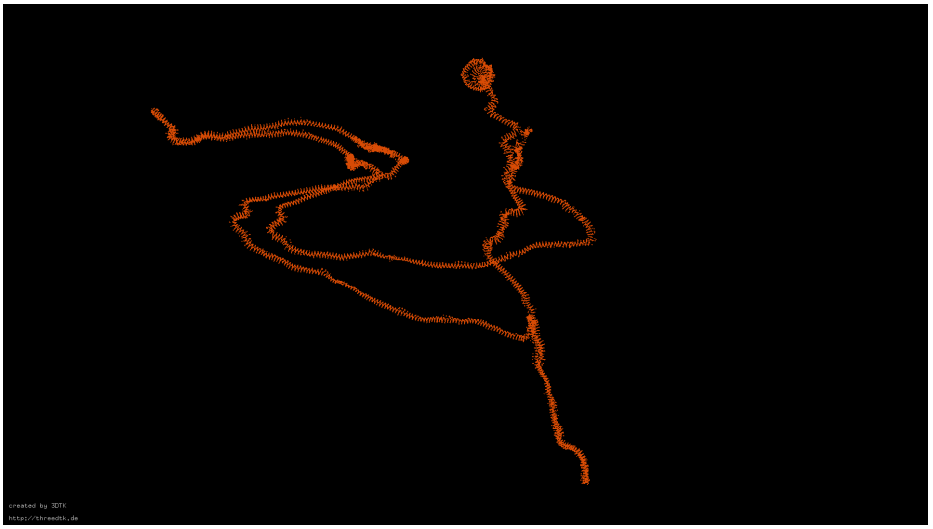
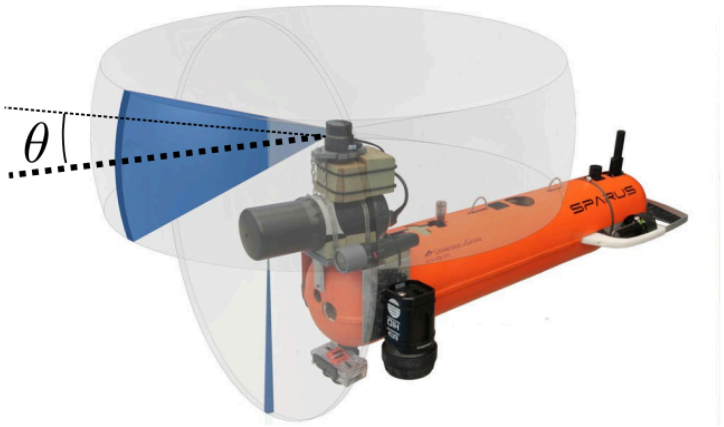


fig II.2 : Vue du dessus de la trajectoire du robot sans SLAM. On voit particulièrement bien le drift sur la branche du karst à gauche de l'image.

Le premier sonar tourne autour de l'axe vertical du robot. Le deuxième tourne autour de l'axe d'avancement du robot (le plus petit faisceau sur *fig II.3*).



Extracted from Mallios, A.; Ridao, P.; Ribas, D.; Carreras, M.; Camilli, R. Toward autonomous exploration in confined underwater environments. J. Field Robot. 2016, 33, 994–1012.

fig II.3 : Schéma des sonars

C'est ce dernier sonar qui nous intéresse le plus car il permet de faire une reconstruction 3D du karst qu'il est en train d'explorer. Dès que j'emploierai le terme "sonar", je me référerai à celui-ci.

Cependant ce sonar a une vitesse de rotation relativement lente, ce qui crée un nuage de points peu dense et partiel. Par conséquent, la reconstruction 3D du karst par le sonar est

incomplète, ce qui implique que même si le robot passe au même endroit entre l'aller et le retour, les scans aller et retour peuvent être différents.
Par exemple, sur la *fig II.4*, la distance à l'obstacle mesurée par le sonar diffère entre l'aller et le retour, alors que la trajectoire est identique, en particulier autour de 129,5 m.

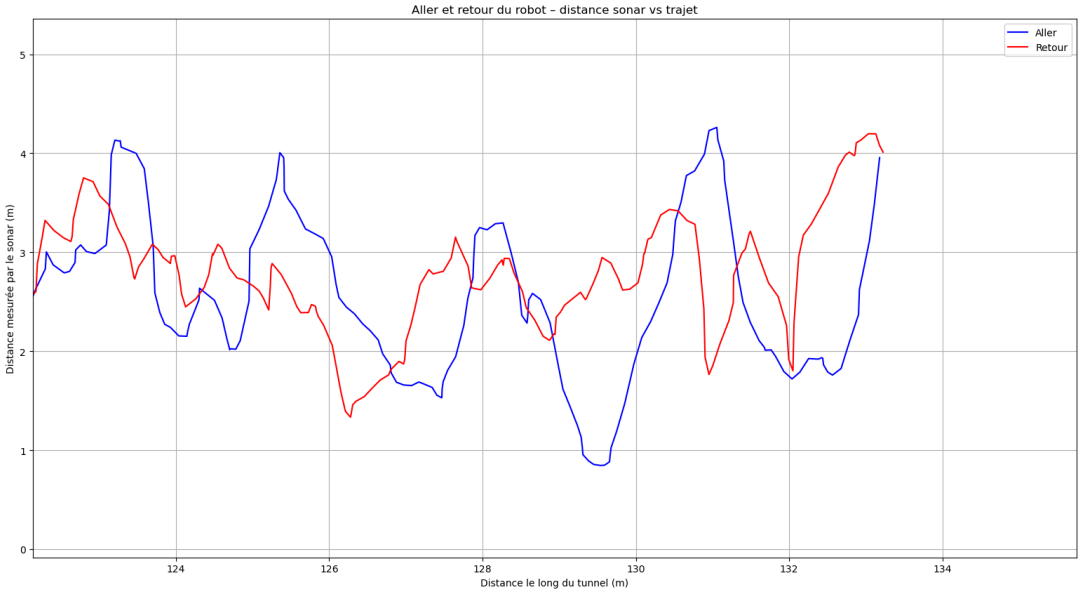


fig II.4 : distance d'obstacle lue par le sonar en fonction de l'avancement du robot

Les algorithmes de SLAM classiques sont faits pour fonctionner avec des LIDARs 3D qui captent des centaines de points 3D par seconde ou avec des robots évoluant sur un plan.
Nos conditions de tests étant très éloignées, ces algorithmes risquent de ne produire aucun résultat ou d'échouer facilement.

Durant mon stage, j'ai donc essayé d'explorer d'autres pistes et de tester des algorithmes de SLAM ou de détection de loop closure moins conventionnels.

Une simulation du robot et d'un environnement karstique m'a été fournie par Gabriel Betton. J'ai principalement travaillé avec les données de cette simulation car nous manquons de données réelles.
Par souci de simplification, toutes les simulations produisent de simples trajectoires aller-retour pour le robot.

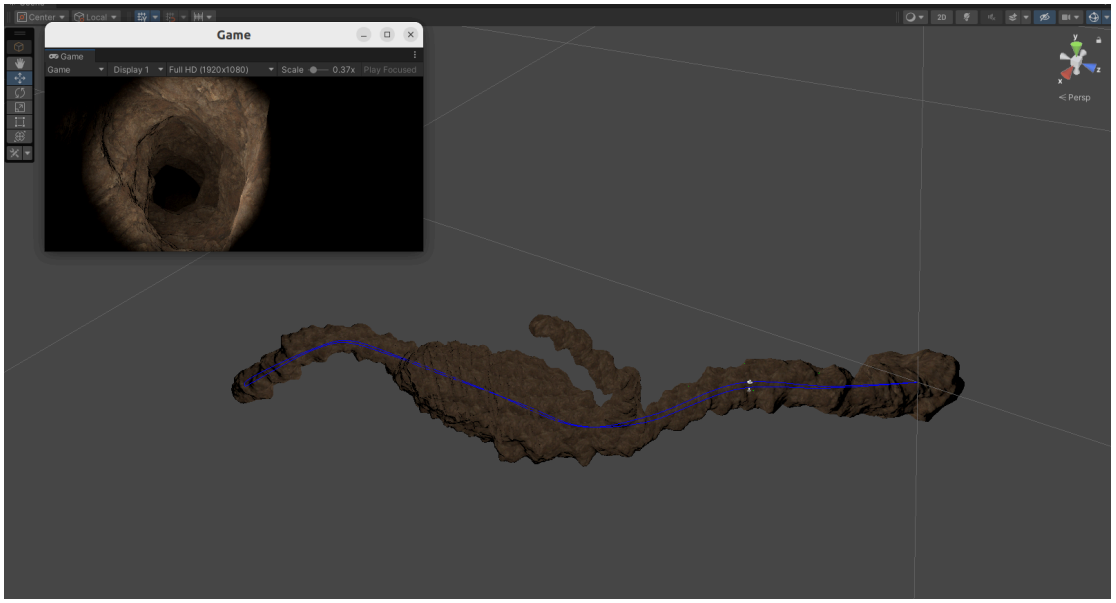


fig II.5 : Capture d'écran de la simulation

III - Simulation et reconstruction

Codes et assets dans le dossier Simu_Karst/

La simulation qui m'a été donnée a été réalisée avec Unity. J'ai changé peu de choses à la configuration globale de cette simulation à part la manière dont sont enregistrés les angles du robot.

Cependant j'ai rajouté différents environnements (grotte/karst) pour le besoin de mes tests. J'ai également créé un script qui permet de lancer plusieurs scènes à la suite avec des paramètres différents en un seul clic (*test_Manager.cs*).
Vous trouverez ces ajouts et modifications dans *Simu_Karst/karstunity 1(1)/karst/Assets*.

A noter que je ne me sers pas de la caméra présente sur le robot dans la simulation, je ne m'en suis donc pas occupé et elle est mal placée dans la plupart des scènes que j'ai créées.

La simulation écrit dans un fichier CSV plusieurs informations et données capteurs :

- l'attitude du robot (avec les angles en quaternion et Euler)
- la vitesse du robot
- les vitesses de rotation du robot
- les angles des deux sonars et leurs distances d'obstacles détectés respectifs
- la position du robot
- le temps auquel tout ça a été enregistré

Les principaux paramètres que nous pouvons modifier sont la vitesse de rotation des sonars et la fréquence d'acquisition des capteurs.

Si vous voulez plus d'informations, n'hésitez pas à demander à Gabriel Betton, le créateur de la simulation.

Mon premier objectif a donc été de reconstruire la trajectoire du robot et le nuage de points créé par le sonar. Les données de la simulation étant parfaites, une simple intégration sur les valeurs de vitesses en prenant en compte l'attitude du robot suffit.
J'ai donc ajouté la possibilité de rajouter du bruit et du drift lors de la reconstruction des données pour se rapprocher des données réelles.

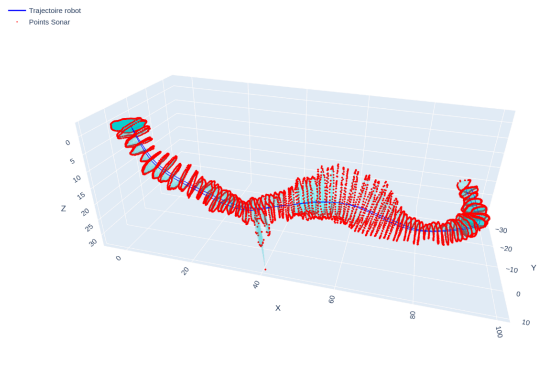


fig III.1 : Trajectoire et nuage de points reconstruits sans bruit ni drift

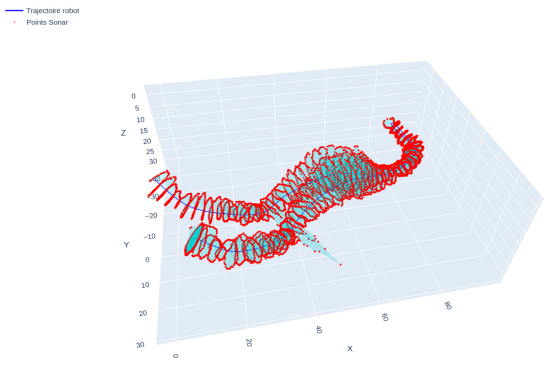


fig III.1 : Trajectoire et nuage de points reconstruits avec bruit et drift

IV - SLAM

A) Etat de l'art

Une autre tâche importante de mon début de stage était de faire l'état de l'art pour les algorithmes de SLAM qui se basent également sur des données peu denses ou qui sont faits pour des environnements similaires à un karst.

Comme déjà dit, la plupart des algorithmes de SLAM sont prévus pour fonctionner avec des LIDARs 3D, souvent à haute fréquence. Les algorithmes aboutis et open source qui sont spécialisés dans le traitement de capteurs de télémétrie à basse fréquence sont en fait inexistants. Beaucoup de ces programmes ne précisent pas explicitement la fréquence minimale d'acquisition de données mais recommandent certains LIDARs spécifiques, qui ont toujours une vitesse d'acquisition bien supérieure à notre sonar. Par exemple, il est recommandé d'utiliser des lidars 3D qui mettent à jour le nuage de points 10 fois par seconde minimum pour l'algorithme de SLAM Glim. Notre sonar nous permet plutôt d'être à 0.5 Hz, tout en faisant une acquisition de points bien plus faible qu'un lidar 3D.

Le second problème étant que la plupart des algorithmes conçus pour fonctionner avec un LIDAR utilisent l'odométrie visuelle et ne prennent pas nativement en charge les données odométriques d'autres capteurs.
C'est un défaut qui peut certainement se régler, mais j'ai jugé qu'il serait plus rapide et enrichissant de repartir de zéro plutôt que d'essayer de comprendre et modifier un code qui n'est pas le mien.

Enfin, la plupart des algorithmes de SLAM partent du postulat que le robot passera de nombreuses fois aux mêmes endroits, ce qui permet d'affiner les résultats à chaque passage. Mais dans notre cas, nous ne faisons qu'un seul aller-retour.

Cependant certains papiers ou algorithmes restent intéressants, voici une sélection :

Article scientifique	Lieu confiné ?	Données peu denses ?	3D ?
Graham, Matthew C., Jonathan P. How, and Donald E. Gustafson. "Robust Incremental SLAM with Consistency-Checking." IEEE, 2015. 117–124.	NON	NON	NON
Kamak Ebadi, Lukas Bernreiter, Harel Biggie, et al. "Present and Future of SLAM in Extreme Underground Environments" *Jet Propulsion Laboratory, California Institute of Technology*, arXiv preprint, 2022. arXiv:2208.01787.	OUI	NON	OUI

Danilo Tardioli, Lorenzo Cano and Alejandro R. Mosteo. "UAV Navigation in Tunnels with 2D tilted LiDARs", arXiv preprint, 2024. arXiv:2404.09688v1.	OUI	OUI	OUI
Prados Sesmero, Carlos, Sergio Villanueva Lorente, and Mario Di Castro. "Graph SLAM Built over Point Clouds Matching for Robot Localization in Tunnels." *Sensors*, vol. 21, no. 16, 2021, article 5340. https://doi.org/10.3390/s21165340.	OUI	NON	OUI
Hanzhi Zhou, Zichao Hu, Sihang Liu, Samira Khan. "Efficient 2D Graph SLAM for Sparse Sensing", arXiv preprint, 2023. arXiv:2312.02353v1.	NON	OUI	NON
K. R. Beevers and W. H. Huang, "SLAM with sparse sensing," *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, Orlando, FL, USA, 2006, pp. 2285-2290, doi: 10.1109/ROBOT.2006.1642043.	NON	OUI	NON

Voici également le lien d'une page Github regroupant tous les algorithmes de SLAM open-source : List-of-Open-Source-SLAM-projects

B) Evaluation des méthodes de recalage

Codes dans le dossier Bench_global_registration/

Même si les algorithmes classiques ne sont pas censés produire de bons résultats, les méthodes employées restent pertinentes.
Tout particulièrement le recalage de nuage de points pour détecter une loop closure et en déduire la transformation correspondante.

Les algorithmes classiques vont utiliser un scan (un tour de lidar) entier, le comparer à la carte déjà construite et estimer la similarité via un algorithme type ICP. Or dans notre cas, un scan complet représente très peu de points (une soixantaine), et la carte est incomplète (à cause de cette fréquence d'acquisition et de la vitesse de rotation du sonar).

Pour compenser ces informations pauvres, j'ai décidé d'assembler plusieurs scans en un seul nuage de points. On perd légèrement en précision (on se déplace uniquement en dead-reckoning le temps d'acquérir tous ces scans), mais on permet de donner plus de points à la méthode de recalage de nuage de points, ce qui devrait permettre de meilleurs résultats.
Je découpe donc mon nuage de points aller en n tranches de x points, et lors du retour, dès que j'ai une tranche de taille x, j'essaie de la faire matcher avec une des tranches de l'aller. Une fois le meilleur match trouvé, j'applique la transformation calculée par le recalage à la position de mon robot (lors de mes tests je prenais la position milieu de la tranche).

Par exemple, sur l'image ci-dessous, on peut voir le nuage de points créé à l'aller et celui créé au retour. Chaque couleur de l'aller correspond à une tranche. On applique ensuite la couleur sur la tranche retour qui semble le mieux matcher selon la méthode de recalage.

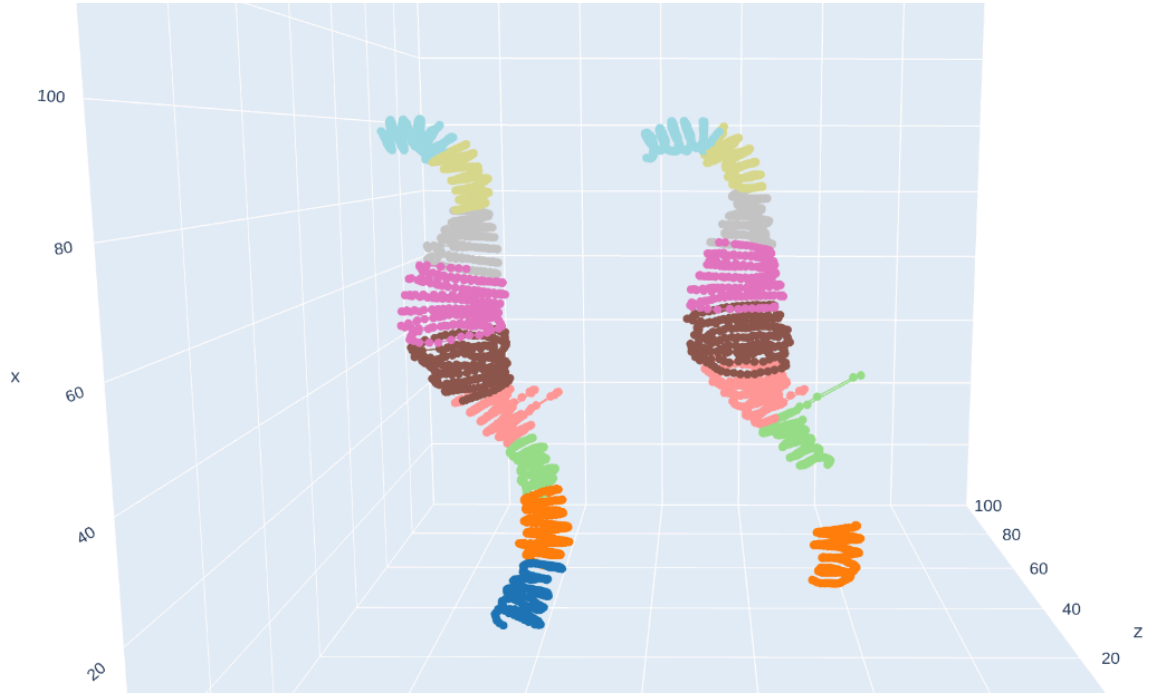


fig IV.1 : Découpage en tranche et identification des meilleurs matchs par ICP, aller à gauche, retour à droite

Mais il est important de noter que cette méthode dispose de nombreuses variables :

- On peut utiliser différentes méthodes de recalage
- Les résultats dépendent de la fréquence d'acquisition du sonar
- Mais aussi de la vitesse de rotation du sonar
- Des incertitudes (bruit) des mesures sonar
- Et la taille des tranches influera sur le résultat

J'ai donc décidé d'écrire un programme pour tester toutes les configurations possibles et essayer de trouver le meilleur compromis.
J'ai testé différent(e)s :

- méthodes de recalage (ICP, GICP et CPD)
- vitesses de rotation (16 à 650 deg/s)
- fréquences d'acquisition (7.5 à 100hz)
- bruits
- tailles de tranches (0.25 à 40m)

Comme on pouvait s'y attendre, plus la taille de tranches diminue,plus la vitesse de rotation diminue et plus le bruit augmente, moins les méthodes de recalage sont précises. Cependant la

fréquence d'acquisition n'a pas l'air d'avoir un grand impact, surement à cause de la vitesse de rotation du sonar relativement lente.

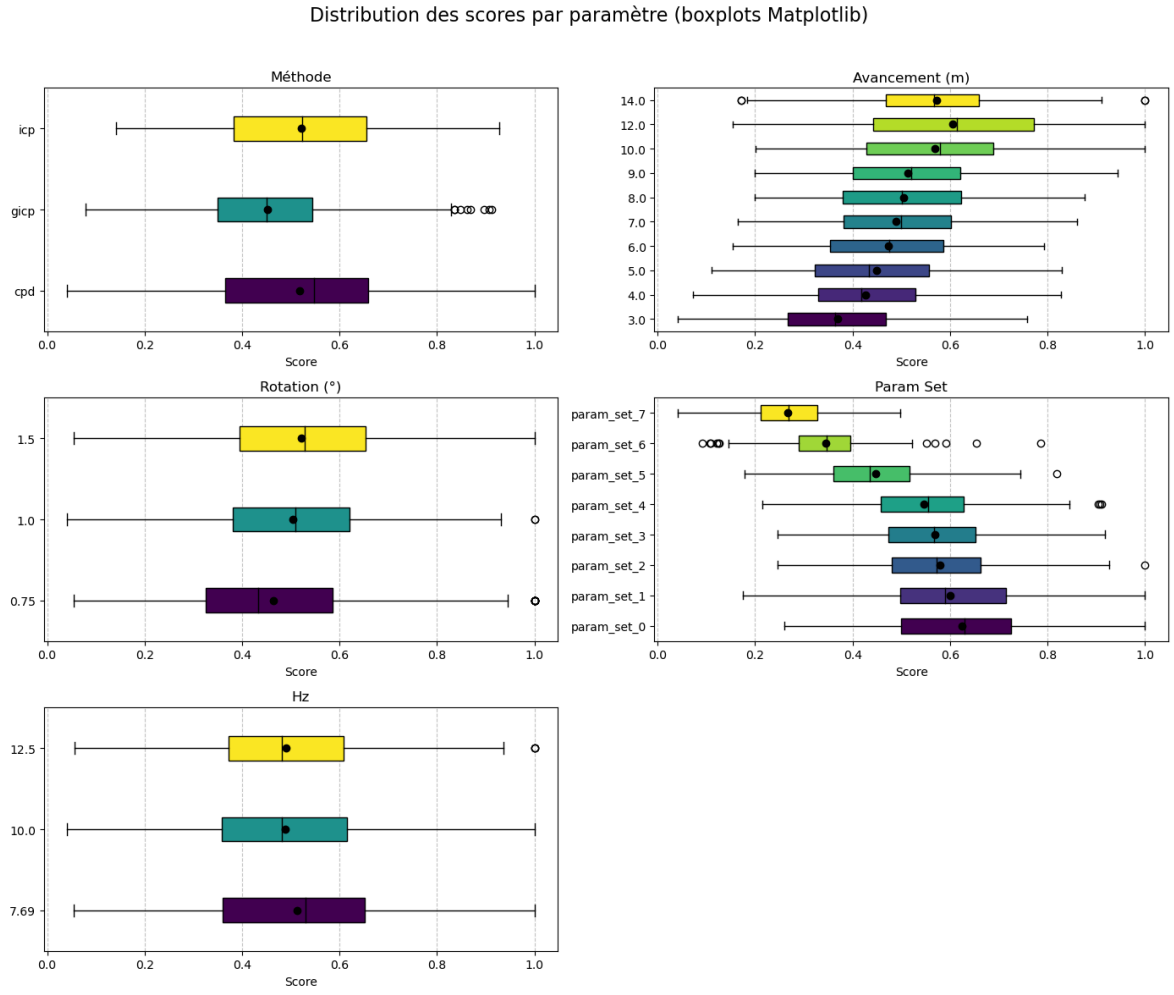


fig IV.2 : Exemple de visualisation pour la comparaison des méthodes de recalage

Mais on obtient tout de même des résultats exploitables pour des paramètres raisonnables (par exemple : 11 tranches sur 13 bien identifiées avec une fréquence de 7.5hz, une rotation de 50 deg/s, un léger bruit et des tranches de 10m).

Mes résultats montraient que le CPD était l'algorithme le plus précis, bien que ce soit le plus lent. Mais je me suis rendu compte à la fin de mon stage que je n'avais pas correctement paramétré l'ICP. Par conséquent, vu que l'ICP était juste légèrement moins précis que le CPD, on peut imaginer qu'il est en réalité supérieur.

Mes programmes peuvent être relancés mais avec la quantité de données et de tests et de la faible optimisation de mes codes, tout recalculer peut prendre plusieurs heures.

C) Mon SLAM

Codes dans le dossier GTSAM/

Une fois la loop closure réalisée grâce à l'ICP, il ne reste plus qu'à l'inclure dans un algorithme SLAM. La variante la plus répandue du SLAM est le Graph SLAM (estime la trajectoire d'un robot en optimisant un graphe où les nœuds sont ses poses et les arêtes ses observations).

J'ai utilisé la librairie GTSAM pour réaliser mon Graph SLAM, comme elle est populaire, simple à prendre en main et s'occupe de toute la partie mathématique et optimisation du SLAM.

Les résultats obtenus sont encourageants mais loin d'être parfaits. On voit bien que les trajectoires aller et de retour coïncident bien comme elles le devraient, mais le drift n'est que partiellement corrigé.
De plus, les résultats sont très sensibles à la taille des tranches.
J'évalue la performance du SLAM en calculant l'erreur RMSE entre la trajectoire parfaite du robot (ground truth) et la trajectoire corrigée après SLAM.

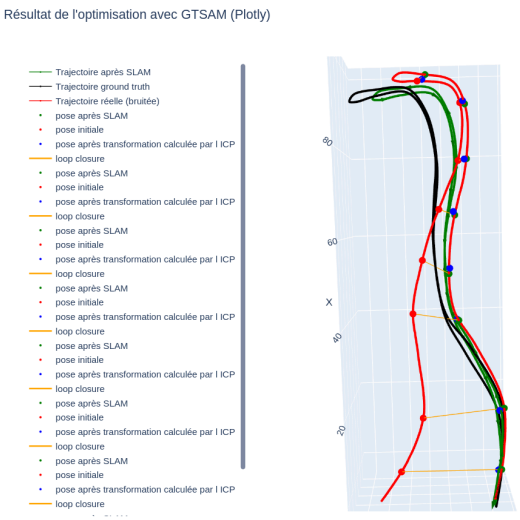


fig IV.3 : Résultat après SLAM, léger bruit et drift de 0.05 m/s, tranches de 14m, erreur RMSE finale : 3.35m

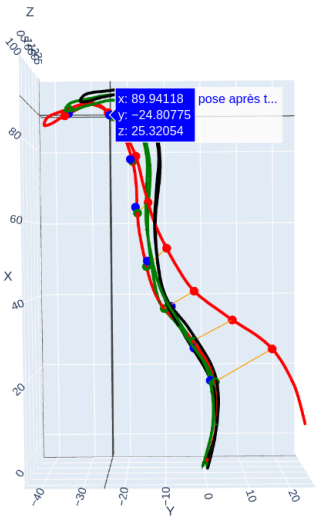


fig IV.4 : Pareil, mais avec un drift de -0.05 m/s et une erreur de 2.7m

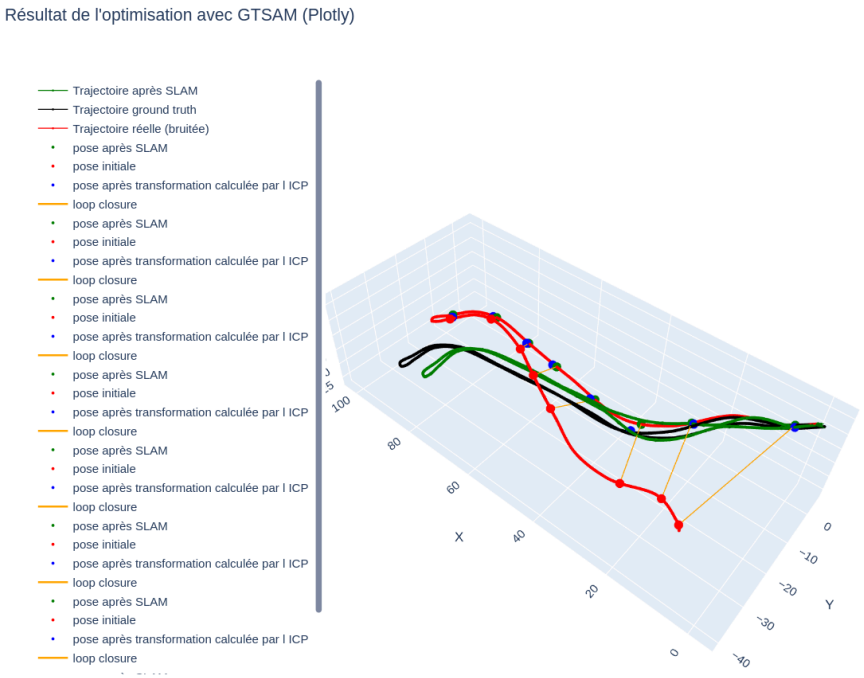


fig IV.5 : Résultat après SLAM, pas de bruits et drift de 0.08 m/s, tranches de 14m, erreur RMSE finale : 5.58m

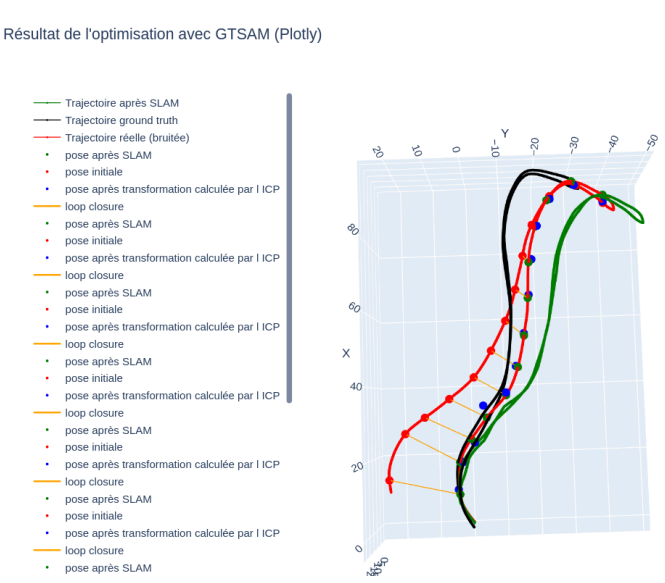


fig IV.6 : Résultat après SLAM, léger bruit et drift de -0.05 m/s, tranches de 9m, erreur RMSE finale : 11.87m

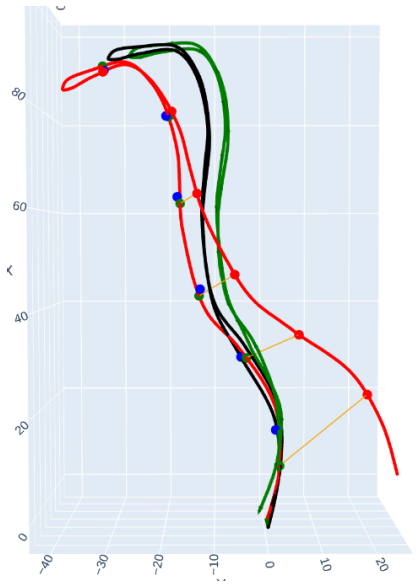


fig IV.7 : Pareil, mais avec tranches de 20m et une erreur de 3.84m

Avoir une seule loop closure avec référence fixe, un amer, (quand le robot revient à son point de départ) est sûrement un des problèmes principaux.
Même si notre ICP trouve les bonnes loop closures, la transformation calculée est peu précise dû à la pauvreté du nuage de point.
Les erreurs restent faibles (de l'ordre du mètre et d'une dizaine de degrés pour l'orientation), mais ces erreurs s'accumulent rapidement et ne sont jamais vraiment corrigées faute de références fixes.
Je pense que lorsque les tranches sont trop courtes, l'ICP a tendance à accumuler davantage d'erreurs, en particulier sur la composante angulaire de la transformation, ce qui affecte la précision globale du recalage.

Un autre stagiaire à Paris (Georgios Margaritis) a utilisé le logiciel du Dr. Andreas Nüchter pour faire un SLAM sur les données réelles et a obtenu des résultats similaires au mien (aller et retour coïncidant mais drift seulement partiellement corrigé).
Cependant, je pense que leur logiciel est plus simple à utiliser et plus stable que mes codes.

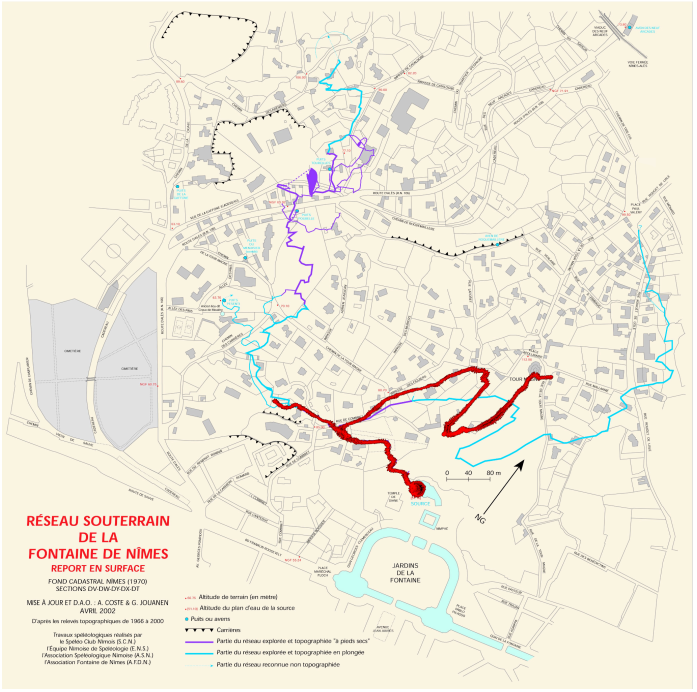


fig IV.8 : Superposition de la trajectoire corrigée avec SLAM réalisée par l'équipe de Paris (rouge) et de la rivière souterraine (ground truth, bleu clair)

Il sera sûrement difficile d'obtenir de meilleurs résultats avec un capteur qui donne si peu de données, sans amers ou sans pos- traitement.

V - Tests de différentes features

Au vu des limitations de la détection de loop closures par les méthodes de recalage, j'ai testé d'autres méthodes pour obtenir une loop closure.

A) Projection 2D

Codes tst_features/plot2D5proj.py et tst_features/plot2Dproj.py

J'ai tenté de visualiser différemment les données pour essayer de trouver d'autres features.

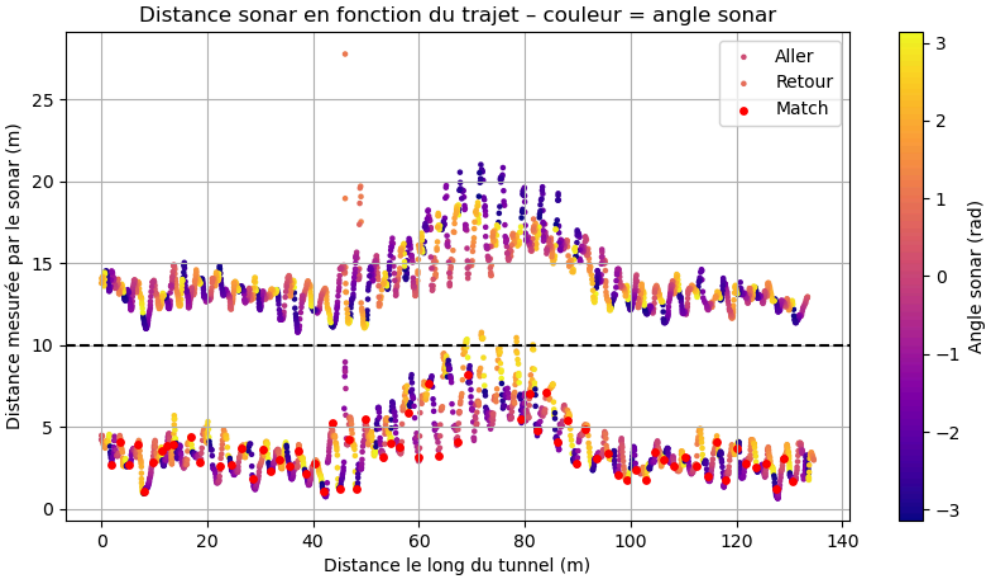


fig V.1 : Vue 2.5D du nuage de points (aller-retour), les points rouges sont les points que le sonar a tapés à l'aller et au retour

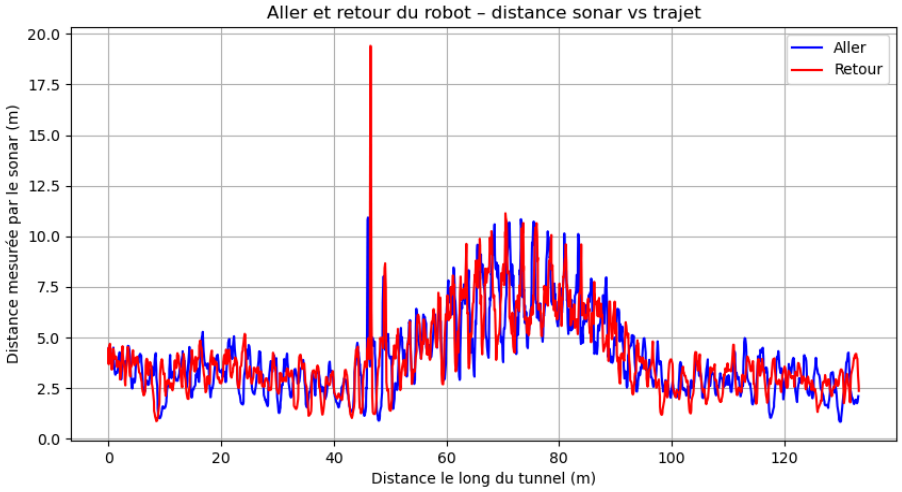


fig V.2 : Vue 2D du nuage de points (aller-retour)

Je n'ai rien trouvé de particulièrement probant.

B) FPFH

Code tst_features/tstFPFH.py

FPFH (*Fast Point Feature Histograms*) est une méthode classique pour repérer des features similaires entre 2 nuages de points. Malheureusement cet algorithme est conçu pour des nuages de points plus denses que le nôtre mais également moins monotones. Je n'ai donc obtenu aucun résultat probant.

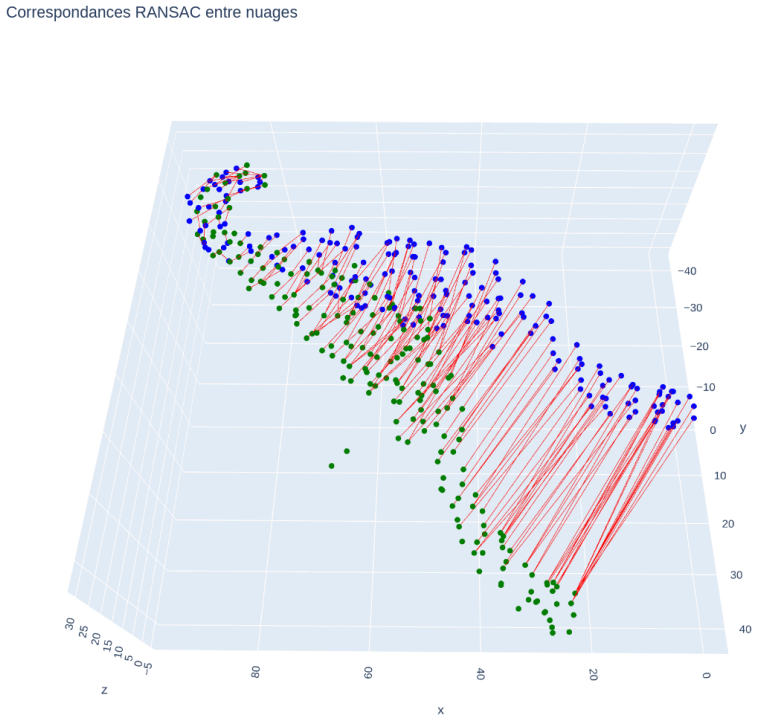


fig V.3 : Algorithme FPFH appliqué sur le nuage de points, les résultats ressemblent à de l'aléatoire

C) Projection 2D d'un scan sonar

Code tst_features/tst_1D_features.py

Ici l'idée est de prendre un scan complet du sonar (un tour complet). Je projette ce scan sur un plan 2D, j'obtiens donc un vecteur de x distances, avec chaque distance associée à un angle du sonar.
Je modifie ensuite ce vecteur pour que les distances correspondent à la distance par rapport au barycentre de mon scan et non plus par rapport à la position du robot. Le but est de minimiser l'importance de la position du robot et de ne caractériser que l'environnement (on suppose que l'attitude du robot entre l'aller et le retour est plus ou moins la même).

Les projections polaires de ces vecteurs donnent les images ci-dessous :

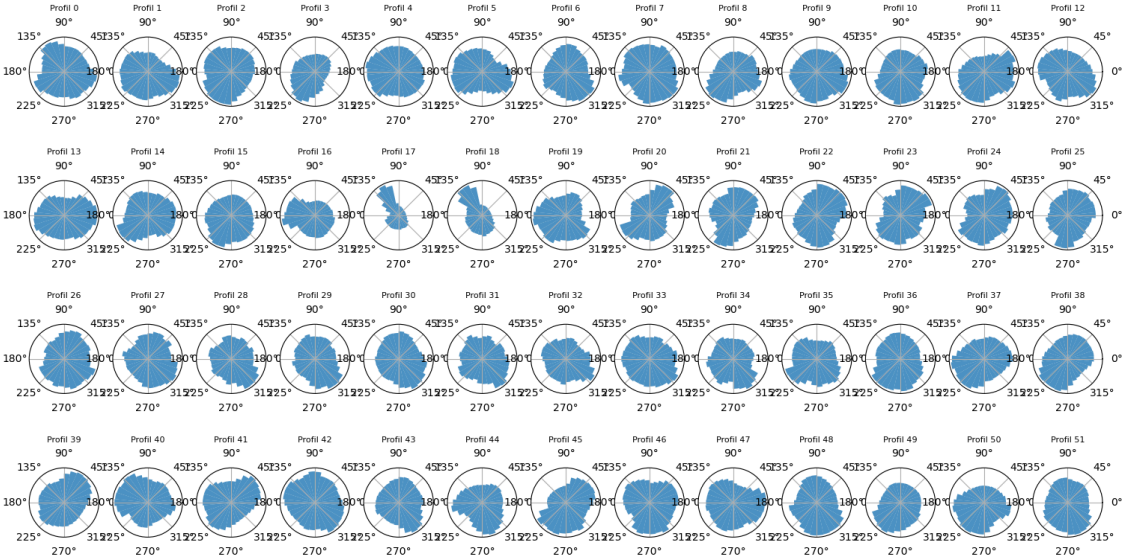


fig V.4 : Slices créées lors du chemin aller

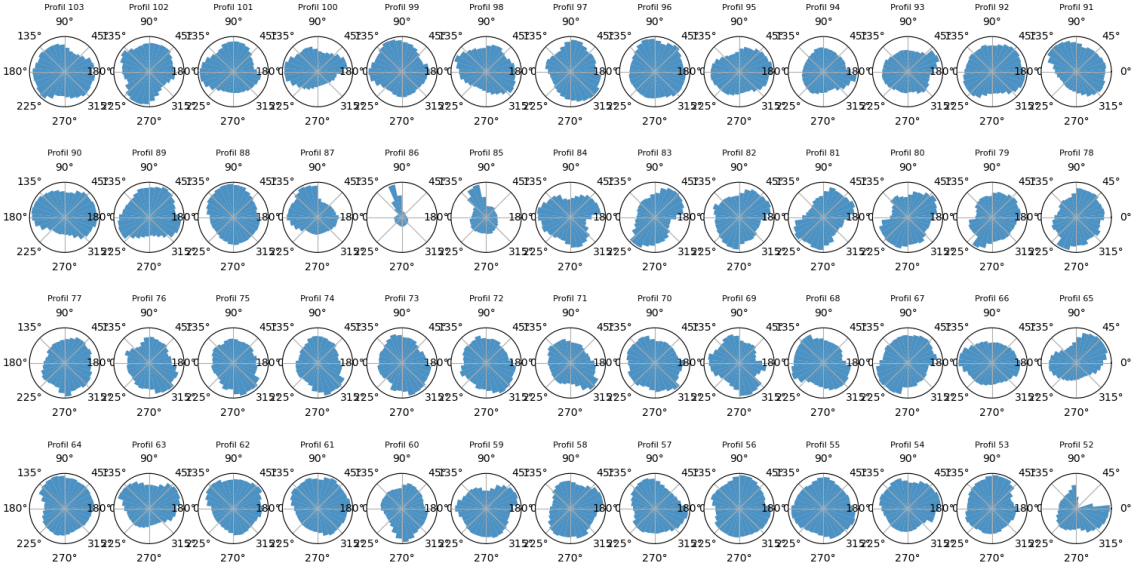


fig V.5 : Slices créées lors du chemin retour

J'obtiens donc 2 ensembles de projections 2D de scans (que je vais appeler slice dès maintenant). Un ensemble avec les slices aller et un ensemble avec les slices retour.
Si j'arrive avec précision à savoir quelle slice retour correspond à quelle slice aller, je serai capable de réaliser des loop closures avec des erreurs plutôt faibles au niveau de la position du robot (le robot avance à peu près d'un mètre le temps de faire un scan).

Pour déterminer la transformation de l'orientation, on pourrait ensuite utiliser différentes méthodes (ICP, simplifier les nuages de points correspondant à chaque slice en plan et essayer de les aligner au mieux...).

J'ai ensuite essayé différentes méthodes classiques pour comparer ces slices et voir quelle slice retour correspond à quelle slice aller.
Ces méthodes sont :

- Erreur RMSE entre les 2 vecteurs des slices
- Erreur MAE (Mean Absolute Error)
- Corrélation de Pearson
- Cosine similarity
- DTW (Dynamic Time Warping)

Je ne vais pas trop m'attarder sur ces méthodes, elles sont toutes très bien documentées.

Les résultats que j'ai obtenus sont acceptables mais loin d'être parfaits.
Pour évaluer la performance de la méthode, je prends les 5 slices retours qui correspondent le mieux à la slice aller selon l'algorithme.
Je divise ensuite les "scores" en plusieurs catégories :

- Perfect : la 1ère slice du top 5 est effectivement le bon match.
- Top 5 : au moins une des slices du top 5 est le bon match.
- Close : la 1ère slice du top5 est la bonne à ± 2 slices près.
- Top 5 close : au moins une des slices du top 5 est la bonne à ± 1 slice près.

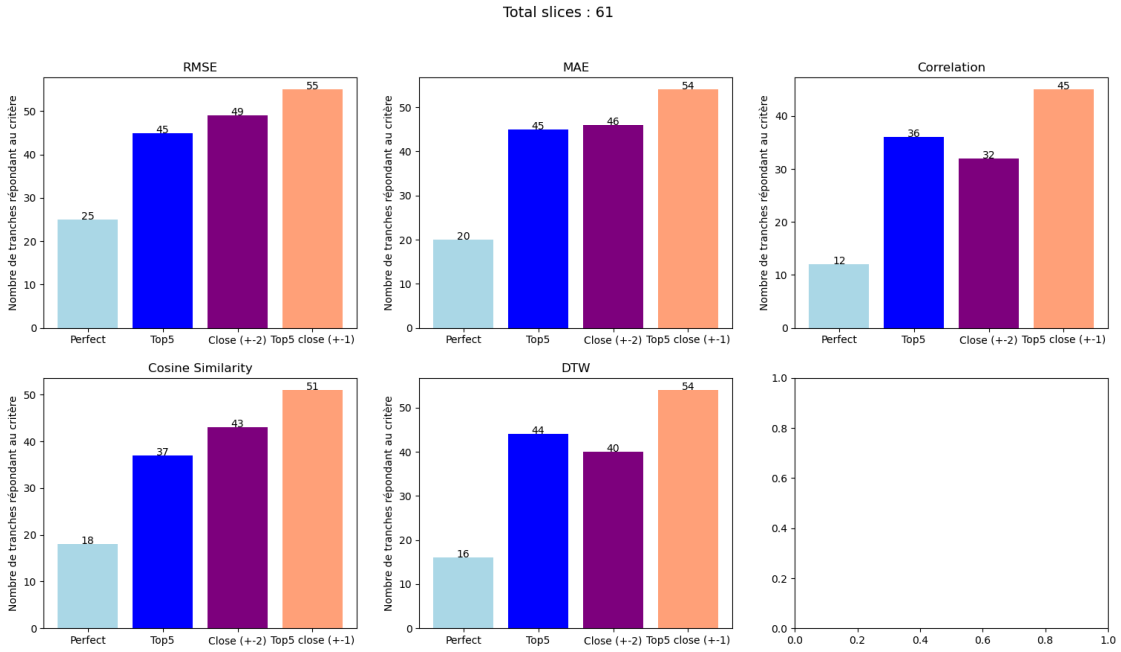


fig V.6 : Résultats obtenus sur des données sans bruit

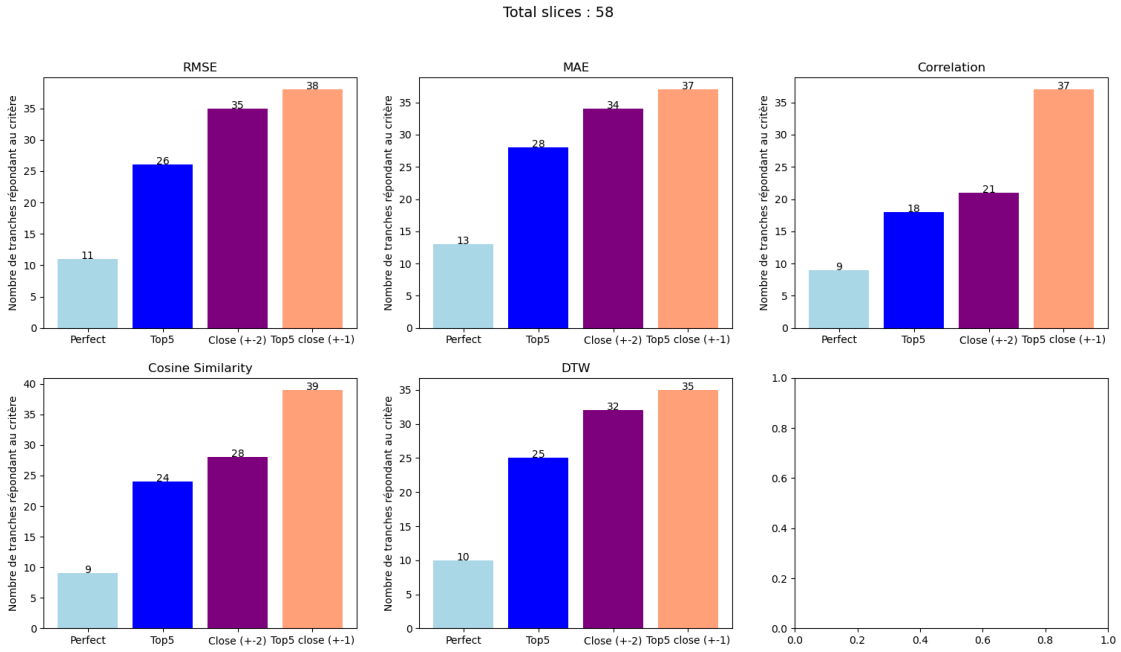


fig V.7 : Résultats obtenus avec données bruitées sur le même jeu de données

On peut voir que les différentes méthodes ont tout de même des résultats plus que corrects. Cependant, même si elles vont souvent être proches du bon résultat, elles trouvent relativement rarement le bon match.

Voyant que ces méthodes échouaient parfois sur des slices que j'arrivais à faire correspondre à l'œil nu, j'ai voulu me tourner vers l'IA pour tester si je pouvais obtenir de meilleurs résultats.

VI - AI

Codes dans le dossier slice_ai/

Pour tous mes codes liés à l'IA, j'ai utilisé la librairie PyTorch.
Comme je veux évaluer la similarité entre 2 slices, je me suis rapidement tourné vers un Siamese Neural Network. Ce type d'IA prend 2 entrées similaires (le plus souvent des images) et produit un score de similarité (qui est souvent la distance euclidienne entre les 2 encodings finaux).

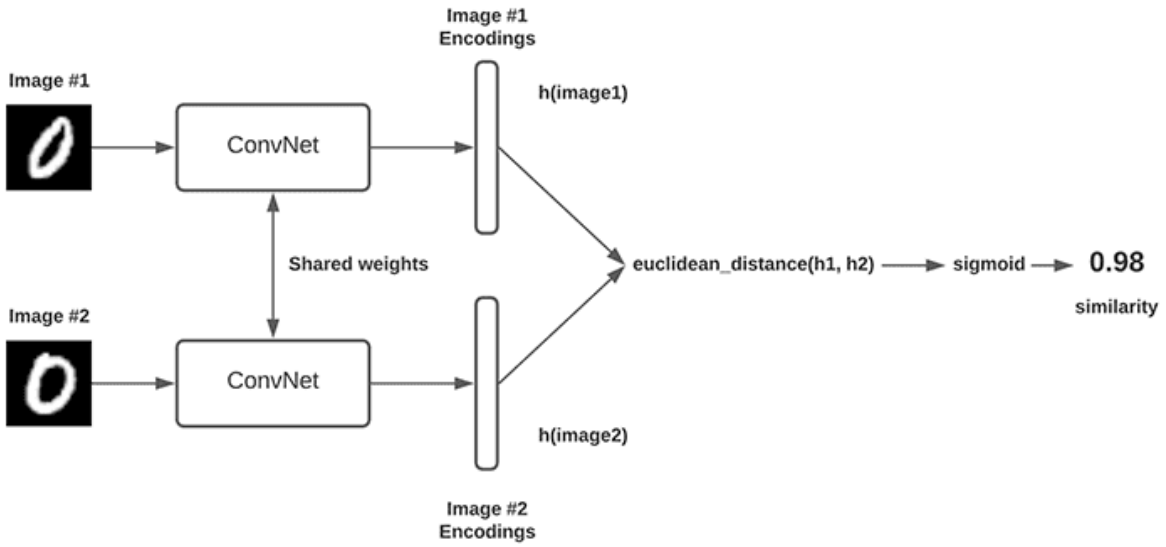


fig VI.1 : Schéma du fonctionnement d'un Siamese Neural Network

Pour donner suffisamment de données au réseau, j'ai créé une douzaine de simulations différentes. J'ai fait varier la vitesse de rotation du sonar pour chaque simulation, puis j'ai rajouté différents niveaux de bruits sur toutes les slices produites.

J'ai testé ce type de réseau avec 3 types d'entrées :

- Des paires de vecteurs des slices, avec un label positif si la paire représente 2 slices prises au même endroit et un label négatif si la paire est composée de 2 slices différentes -> échec, l'IA surapprend.
- Des paires d'images des slices, avec la même logique que pour les vecteurs -> mieux, l'IA arrive à reconnaître des similarités mais il y a encore beaucoup trop de faux positifs.
- Des triplets d'images (anchor, positif, négatif), j'ai donc une slice de référence, une slice similaire et une slice différente -> encore mieux qu'avec les paires mais j'ai toujours trop de faux positifs.

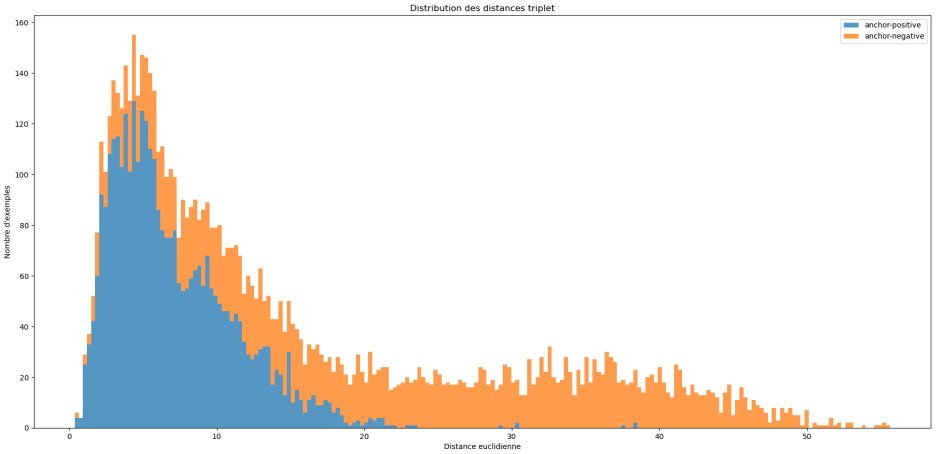


fig VI.2 : Histogramme empilé en fonction de la distance euclidienne calculée par le SNN, test du modèle entraîné sur les triplets sur une simulation de test. En bleu, les paires positives, en orange les négatives. On voit un début de séparation mais il reste beaucoup de faux positifs

J'ai testé différentes configurations de réseau, avec différents nombres de couches, de neurones... J'ai également passé différents types de slices au réseau, en essayant de lui donner des paires/triplets plus ou moins "durs" (avec des slices différentes mais prises à des endroits très proches).
Mais je suis rapidement arrivé à une limitation matérielle quand j'ai commencé à mettre des images en entrée de mon SNN. En effet, mon ordinateur personnel n'est pas équipé de GPU, ce qui fait que chaque epoch pouvait prendre plus de 30 minutes.
Et comme c'était la première fois que je faisais de l'IA, j'aurais bien aimé tester plus de paramètres et je suis sûrement passé à côté de configurations qui auraient donné de bien meilleurs résultats.
J'obtiens des résultats certes corrects, mais au final moins probants qu'avec les méthodes classiques vues dans la section précédente.

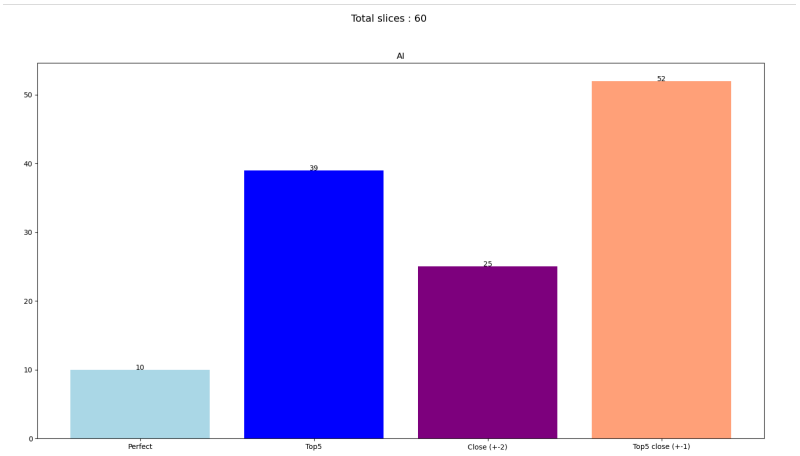


fig VI.3 : Résultat obtenu sur des données sans bruit

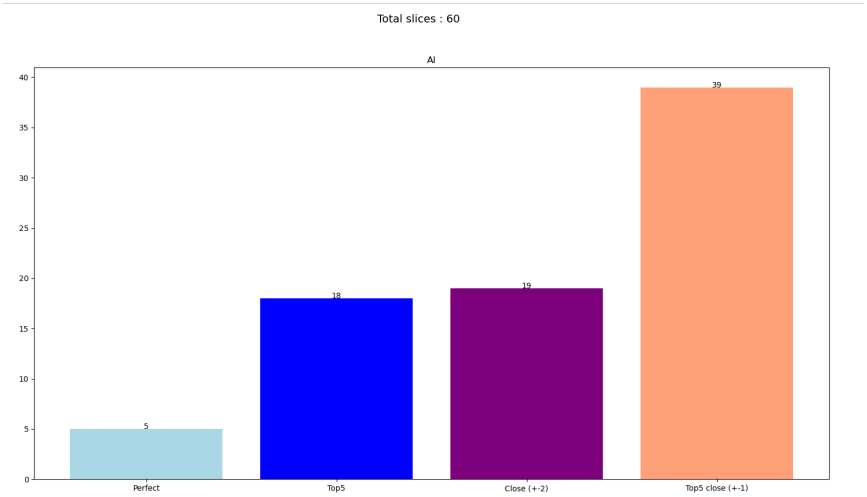


fig VI.4 : Résultat obtenu sur des données bruitées

Mais ce type de problème (similarité entre 2 images) étant classique et les résultats étant encourageants, je pense que quelqu'un avec plus d'expérience que moi devrait obtenir des modèles bien plus performants.

Cependant, il ne faut pas perdre de vue que cette solution reste très dépendante en énergie et en puissance de calcul. Il me paraissait intéressant d'aborder une approche nouvelle pour détecter des features mais celle-ci n'est certainement pas la plus habile. Peut-être que les solutions présentées dans la section précédente se révéleront suffisantes dans certains cas. Cependant, si aucune méthode traditionnelle ne convient parfaitement, je pense qu'il peut être intéressant de persévérer dans l'exploration de solutions IA.

VII - Conclusion

Bien que n'ayant pas réussi à trouver de solutions à la problématique, je pense que plusieurs pistes que j'ai explorées restent pertinentes.
Tout particulièrement le SLAM avec ICP, qui pourrait sûrement obtenir de bien meilleurs résultats assez facilement (meilleur matériel ou introduction d'amer avec du traitement d'image par exemple).
Les slices sont aussi une vision intéressante du problème, et peuvent peut-être inspirer de nouvelles idées à d'autres personnes travaillant sur le projet.

Il est important de garder en tête que tous mes travaux ont été réalisés sur des simulations et pas des jeux de données réelles, ce qui peut être un biais très important.
Enfin, j'ai testé de nombreuses méthodes et librairies, comme différentes approches de recalage de nuages de points, le FPFH ou encore PyTorch. Faute de temps, je n'ai pas pu toutes les explorer en détail, il est donc possible que certaines soient en réalité plus efficaces que ne l'indiquent mes résultats.

Ce stage a été également enrichissant d'un point de vue personnel. J'ai en effet pu découvrir et apprendre plusieurs algorithmes ou concepts (SLAM, Siamese Neural Network, méthode de recalage...) en plus d'avoir pu me familiariser avec le secteur de la recherche.