PROJET SLASHR : IMPLÉMENTATION D'UN ALGORITHME DE SLAM-VI DANS UN ROBOT MUSHR ET COMPARAISON AVEC LA BASELINE LIDAR

Patrick BASTARD, Amélie BORDIN, Mathieu GODINEAU, Mathis LECLERC, Tom LE MENN[1]

([1]) Université de Rennes, ESIR

Représentant Industriel : [redacted]
Encadrant Universitaire : [redacted]

ABSTRACT

Le SLAM est un processus permettant aux robots de se déplacer en autonomie dans des environnements non cartographiés. Dans ce compte-rendu, nous proposerons plusieurs algorithmes de SLAM. Nous verrons leurs efficacités dans le cadre d'un projet de SLAM-VI sur un robot MuSHR. Bien que nous n'ayons pas pu comparer au LiDAR, les résultats nous montrent que les caméras du MuSHR sont bien adaptées pour réaliser un SLAM, notamment grâce à leur IMU intégrée. Il est nécessaire de bien définir les conditions optimales pour le SLAM.

Mots clés - MuSHR, Navigation intérieure, SLAM-VI

1. INTRODUCTION

1.1. Présentation de l'entreprise

Durant ce projet industriel, nous collaborons avec la Direction Générale de l'Armement (DGA). La DGA travaille avec le gouvernement afin d'équiper les forces armées et de préparer les systèmes de défense en France. Il existe plusieurs sites de la DGA en France ayant des expertises dans des domaines différents. Le centre de recherches nous ayant proposé ce sujet est celui de Bruz.

La branche basée à Bruz, avec laquelle nous travaillons, est celle de la Maîtrise de l'Information (DGA-MI). Elle possède une expertise dans les systèmes d'informations, de communication, de cybersécurité, de guerre électronique et de système de missiles.

Dans le cadre de ce projet, nous travaillons en collaboration avec un département spécialisé dans les systèmes de guidage et navigation. Cette branche travaille notamment sur les systèmes de localisation et de suivis de navigation dans des zones sans signal GNSS pour des avions, bateaux ou missiles. Pour faire cela, la méthode utilisée est le croisement de données entres des capteurs inertiels et des moyens externes ou de la correspondance cartographique. Cette fusion de données permet d'estimer un emplacement en corrigeant les erreurs de la navigation inertielle par des recalages sur des données existantes.

1.2. Présentation du sujet

1.2.1. Présentation du projet et du besoin

Le projet sur lequel nous œuvrons a été baptisé « le projet SlaSHR ». Il consiste à étudier les moyens de réaliser de la navigation absolue en environnement inconnu ou non coopératif en intérieur avec absence de signaux GNSS.

Ce projet est né suite au besoin de navigation pour des personnels humains en intérieur dans des lieux non cartographiés et avec l'utilisation de capteurs inertiels à moindre coûts fournissant des données bien moins précises. Nous devons donc utiliser des méthodes différentes en se basant uniquement sur du matériel remplissant les conditions citées précédemment.

Ce projet fait suite aux travaux de [redacted] pendant son stage à la DGA-MI. Celui-ci avait réalisé un premier état de l'art général, et testé ORB SLAM 3 [1] sur des bases données de la littérature. Nous avons pour objectif d'implémenter, entre autres, cet algorithme sur le MuSHR et d'en réaliser les tests sur des trajets indoor.

Figure 1: Robot MuSHR utilisé, et ses caméras T256 (en haut) et D435i (en bas).

1.2.2. Etude proposée par la DGA

Pour le projet SlaSHR, la solution que la DGA-MI souhaite étudier est l'implémentation d'algorithmes de

SLAM-VI (Localisation Et Cartographie en Simultanée avec capteur Visio-Inertiel) sur un projet open-source existant, le MuSHR [2]. Celui-ci comprend un robot tout terrain ainsi qu'un LIDAR, une caméra mono RGB-D inertielle (Intel D435i) et une caméra Tracking Fisheye (stéréo (Intel T256), le tout piloté par le mini ordinateur jetson nano de Nvidia.

L'attente autour de ce projet est la recherche sur les différents algorithmes déjà existants. Suite à une étude de l'art, il faut implémenter un ou une partie des algorithmes retenus. Il faut ensuite comparer les différents résultats obtenus avec ceux des autres algorithmes, la baseline Lidar ne sera finalement pas utilisée sur ce projet, en accord avec l'entreprise.

Nous avons également coopéré avec les différents acteurs du projet MuSHR sur ce projet afin d'étudier la faisabilité et la pertinence d'un travail en utilisant des outils open-source.

2. ÉTAT DE L'ART

Pour ce projet nous réalisons une étude sur plusieurs algorithmes et bases de données. Ceux-ci sont soit conseillés sur le forum MuSHR [3], soit par la DGA, ou encore trouvés par nos recherches pour cette étude. Nous répartissons les algorithmes de la littérature en trois catégories : non-utilisable, non-retenu, et retenu.

La deuxième partie de notre étude de l'art consiste à trouver des bases de données représentative de notre besoin, avec des exemples concrets afin de pouvoir tester notre projet dans les meilleures conditions possibles.

2.1. Algorithmes

Nous sélectionnons donc, dans un premier temps, parmi les algorithmes existants dans la littérature ceux qui se trouvaient être open source, bien documentés et disposant d'un GitHub complet. Cependant, cette première sélection regroupait des algorithmes qui ne convenaient pas à notre projet. Pour retenir un algorithme, nous avons déterminé qu'il devait également être compatible avec ROS [4] et devait disposer d'un mode Visio-Inertiel, et la méthode d'installation bien documentée.

	Kimera	ORB SLAM 3	XIVO
Sources	RGB Mono et Stéréo	RGBD Mono et Stéréo Fisheye	RGB Mono
Compatibilité ROS	☑	☑	☑
Atouts	- Rapide - Plus faible erreur absolue	- Multi Thread - Realtime - Diversité des méthodes	- Realtime - GitHub entretenu
Défauts	Pas toujours stable pour l'instant	Intégration ROS par un indépendant	Pas de documentation détaillée

Tableau 1: Algorithmes de SLAM conservés pour le projet

2.1.1. Algorithmes retenus pour ce projet

Parmi les algorithmes étudiés, nous ne retenons que trois algorithmes : KIMERA [5], XIVO [6] et ORB SLAM 3 [1]. Ceux-ci sont les plus populaires, à jour, et remplissent toutes nos conditions. La compatibilité avec ROS1 est un point non-négligeable, car l'environnement de simulation du MuSHR fonctionne sur RViz, un visualiseur 3D sous ROS1.

ORB SLAM 3 fournit également les fichiers de configuration correspondants à nos deux caméras directement sur leur GitHub, ce qui nous permet de l'utiliser sans avoir à calibrer les caméras : cela en fait donc une option de choix. KIMERA inclut pour sa part un outil de mapping en 3D intégré à la solution. XIVO est quant à lui prometteur pour une utilisation temps-réel.
Les caractéristiques de ces algorithmes sont détaillées dans le Tableau 1 ci-dessus.

2.1.2. Algorithmes non retenus pour ce projet

Plusieurs autres algorithmes ne sont pas conservés pour des raisons évidentes de non compatibilité avec le projet. En effet, la plupart de ces algorithmes sont incompatibles avec du SLAM ou ROS1 (le wrapper dédié pour l'implémentation du SLAM sur le MuSHR) comme OKVIS [7], BASALT [8] ou MSCKF [7]. D'autres sont dépassés par d'autres algorithmes plus récent. C'est le cas de ROVIO [9], ORBSLAM-VI [10], et VINS-Mono [11].

Les algorithmes du Tableau 2 ci-dessous n'ont pas été approfondis pendant le projet mais reste tout de même pertinents dans le cadre de projet implémentant du

SLAM. Nous avons préféré les algorithmes de la partie précédente pour des raisons de praticité.

	ROVIOLI [12]	VINS Fusion [13]
Source(s)	RGB Stéréo	RGB Stéréo/ Mono Fisheye
Compatibilité ROS	☑	☑
Atouts	- Continuité multi-sessions, map-merging, optimisation de batch	☒
Défauts	Beaucoup de traitements : performances élevées requises.	Stéréo un peu plus faible en performances

Tableau 2: Algorithmes de SLAM non-utilisés pour ce projet

Ces derniers sont mieux renseignés, déjà utilisés dans le cadre du MuSHR, plus faciles à implémenter selon nous ou tout simplement meilleur sur les retours qui nous ont été faits.

2.2. Bases de données

Afin d'évaluer les performances de nos algorithmes, nous avons besoin dans un premier temps de bases de données solides, elles pourront apporter des informations précises. Parmi les jeux de données publics utilisés pour évaluer les algorithmes de SLAM, nous avons retenus EuRoC MAV, KITTI et Visma. Ces bases seront utilisées afin de vérifier le bon fonctionnement et les performances de nos algorithmes sélectionnés. Employées pour de l'apprentissage à grande échelle pour de nombreux projets, ces bases de données sont complètes et représentatives des conditions à évaluer lors d'un SLAM.

Dans ce qui suit, nous allons présenter brièvement les jeux de données que nous avons jugés les plus intéressants. Une étude préliminaire réalisée par la DGA conforte également notre choix

2.2.1. EuRoC MAV

EuRoC MAV [14] est un ensemble de données collectées par un micro-véhicule aérien (MAV). L'ensemble des données disponible contient des images stéréo. Séparées en 12 situations différentes, classées par difficulté de localisation. On remarque grâce à la Figure 2 que la base de données contient des images pertinentes par rapport au contexte de navigation en intérieur. Cependant, il est important de préciser que ces acquisition ont été réalisées par un véhicule aérien et peuvent donc ne pas convenir sur tout les points à un véhicule terrestre.

2.2.2. KITTI

La base de données KITTI [15] est créée et utilisée par Toyota dans le cadre des recherches sur les véhicules autonomes. C'est une base de données contenant des scénarios de 6 heures de trafic routier dans plusieurs cadres différents comme visible sur la Figure 3. Ces données sont très intéressantes dans le cadre de navigation sur longues distance, mais pas représentative de la navigation en intérieur.

Figure 2: Exemple du dataset EuRoC MAV. Source : [14]

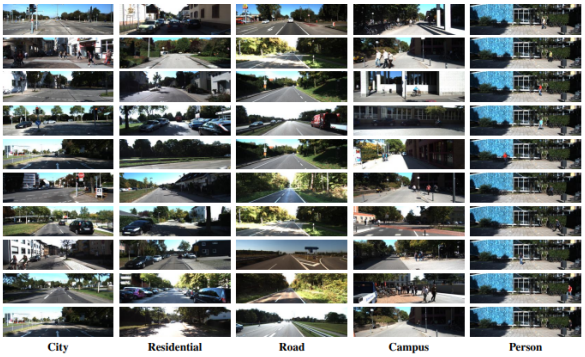

Figure 3: Exemples du dataset KITTI. Source : [15]

Figure 4: Exemples du dataset VISMA. Source : [16]

2.2.3. Visma

Visma (Mappage sémantique inertiel visuel) [16] contient des vidéos RGB et des mesures inertielles pour le développement de système de cartographie. Cette base de données à été développée par l'ingénieur qui est également à l'origine de l'algorithme XIVO, que nous avons retenu. Elle contient plusieurs images d'intérieur mais les cas présent dans cette base de données ne permettent pas d'avoir des cas de test concrets pour notre projet comme nous pouvons le voir sur la Figure 4 ci - dessus.

3. MÉTHODOLOGIE

3.1. Gestion du projet

3.1.1. Planification du projet

Nous organisons le projet autour des grands axes suivants: en premier lieu nous allons tester les implémentation sur des données publiques pour avoir des premiers résultats et pouvoir vérifier que l'installation et l'exécution des algorithmes sont correctes. Par la suite, nous prévoyons une phase d'acquisition de nos propres données avec l'entreprise et le matériel mis à disposition. Enfin nous allons tester et valider les implémentations sur les données acquises afin de confirmer et examiner la compatibilité des différentes approches dans un environnement intérieur.

3.1.2. Organisation du temps

Pour pouvoir visualiser et tester nos algorithmes en plus des bases de données trouvées, nous avons prévu plusieurs demi-journées en compagnie de notre représentant industriel de la DGA. En effet, ces demi-journées ont pour objectif de mettre à notre disposition le MuSHR et ses deux caméras Intel RealSense (une caméra RGB-D inertielle et une caméra Tracking Fisheye) pour pouvoir acquérir des données. Notre emploi du temps est organisé autour des demi-journées en compagnie de la DGA et du MuSHR et essentiellement pendant les semaines dédiées au projet industriel pour pouvoir nous consacrer pleinement à celui-ci.

3.1.3. Répartition des rôles

Afin de travailler efficacement sur ce projet, nous avons pensé à une répartition des rôles telle que : deux personnes se concentrent sur l'installation des algorithmes de SLAM, une personne s'occupe de tous les rendus, une autre personne se charge de la communication sur le forum GIT du projet MuSHR ainsi que la rédaction des tutoriels et enfin, une autre personne s'occupe de l'acquisition des données et des scénarios d'acquisition.

Bien sûr, les rôles attribués ne sont pas fixés, nous avons tous été assez flexibles sur le travail à effectuer. Si il y avait un problème pour l'installation des algorithmes, n'importe qui pouvait apporter son aide.

3.1.4. Communication

Nous organisons également des réunions régulières avec nos tuteurs entreprise et universitaire afin de discuter de nos avancements et de l'organisation des étapes suivantes. De nombreux échanges par mails entre ces réunions nous permettent de partager nos interrogations et problèmes rencontrés.

3.1.5. Étude des risques

Nous avons identifié plusieurs risques liés à ce projet, tels qu'un manque crucial de compétences des membres de l'équipe, une impossibilité d'avoir accès au MuSHR pendant le projet. D'autres risques moins impactants seraient que les résultats ne correspondent pas à nos attentes, ou qu'aucun algorithme étudié ne réponde au problème.

3.2. Méthode d'acquisition

Nous décidons d'acquérir nous-même de nouvelles données avec les caméras du MuSHR : en effet, les données des bases citées précédemment ne sont pas totalement représentatives du contexte (navigation indoor par un robot de petite taille qui ne se déplace qu'au sol). Il nous est alors nécessaire d'enregistrer des données plus cohérentes avec notre objectif.

Cependant, nous rencontrons des problèmes pour faire fonctionner la Jetson Nano du MuSHR malgré l'aide de notre tuteur de la DGA. Il est donc impossible d'utiliser le MuSHR comme une voiture radio commandée avec nos algorithmes exécutés en temps réel. L'acquisition automatique de données par le MuSHR en pilotage manuel est également impossible.

Une des solutions possibles pour l'acquisition de données sur le MuSHR est donc de détacher les deux caméras pour pouvoir les utiliser reliées à un ordinateur. Nous utilisons alors le logiciel Windows Intel RealSense Viewer pour faire l'acquisition de quelques jeux de données plus représentatifs de nos besoins que ceux déjà existants. Une autre solution est d'utiliser la librairie Intel sur Ubuntu avec ROS pour faire une acquisition et fusion des données du gyromètre et de l'accéléromètre récupérées par les caméras en plus des données visuelles capturées par les caméras RealSense.

Grâce à ces différentes techniques, nous créons notre base de données d'acquisition pouvant simuler les différentes configurations nécessaires. Nous récapitulons ces différentes acquisitions dans le Tableau 3 ci-dessous.

Acquisition	1	2	3	4
Lieu	Salle de TP position fixe	Couloir avec fenêtres Voir Erreur : source de la référence non trouvée	Escalier avec petites fenêtres Voir Figure 6	Couloir sans fenêtre
Fermeture de boucle	☒	☑	☑	☑

Tableau 3: Récapitulatifs des acquisitions

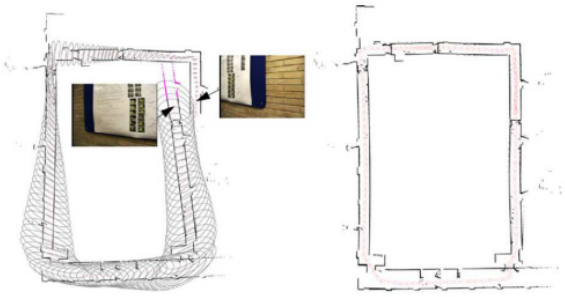

Figure 5: Exemple de correction de la carte et de la trajectoire suite à une fermeture de boucle. Source : [24]

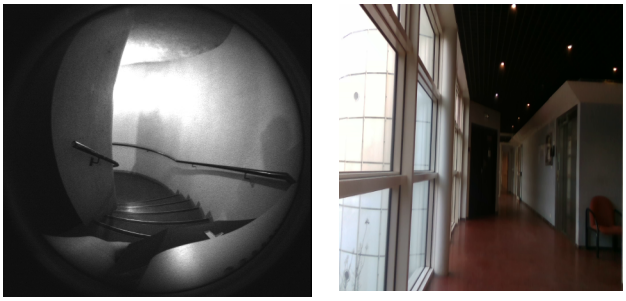

Figure 6: Un exemple d'acquisition dans les escaliers avec la caméra T256 à gauche et avec la caméra D435i dans le couloir à droite

Nous souhaitons préciser dans le Tableau 3 si les acquisitions sont effectuées en faisant une trajectoire avec fermeture de boucle ou non. En effet, dans le cas d'une trajectoire de fermeture de boucle, au moment du passage au départ une fois le trajet effectué, l'algorithme va optimiser la trajectoire. Il va automatiquement repérer qu'une boucle à été faite et va pouvoir corriger d'éventuelles erreurs de position en fermant cette boucle. On peut observer un exemple de cette optimisation sur la Figure 5 ci-dessus.

Les acquisitions sont effectuées avec nos deux différentes caméra, une Intel RealSense D435i (RGB-D, inertielle) ainsi qu'une Intel RealSense T256 (Stereo fisheye inertielle). Nous avons fournis quelques exemples de ces acquisitions sur la Figure 6 ci-dessus.

Cette nouvelle base de données personnalisée nous permet de tester nos algorithmes de SLAM-VI et d'évaluer leur efficacité dans des configurations différentes. Il faut faire tout particulièrement attention aux conditions de ces acquisitions. En effet, certains paramètres comme la luminosité de la pièce peuvent avoir un grand impact sur la qualité du mapping obtenu à l'aide de nos algorithme. C'est pour cela que nous précisons la présence des fenêtres dans les environnements d'acquisition car celles-ci apportent une très grande quantité de lumière qui pourrait jouer de manière défavorable sur les performances du SLAM.

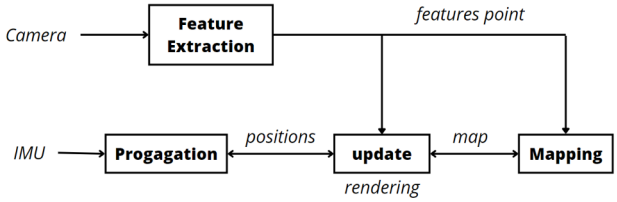

Figure 7: Schéma fonctionnel du SLAM-VI

3.3. Méthode d'implémentation

L'objectif du projet étant de tester les approches existantes et d'examiner si elles sont pertinentes dans notre contexte, nous n'avons pas réimplémenté les différentes approches mais nous nous sommes plutôt appuyés sur les implémentations open source des paragraphes 3.3.2 et 3.3.3.

3.3.1. Fonctionnement du SLAM-VI

Les algorithmes que nous allons tester implémentent tous un SLAM-VI, dont le fonctionnement général est visible sur la Figure 7. Le principe est simple et généralement identique suivant les implémentations, des points d'intérêts sont extraits des donnés caméra (image RGB), par exemple avec un algorithme SIFT. La position des points est déduite par triangulation et l'algorithme les utilise pour construire une carte du monde. La déviation générée est corrigée au fil du temps par les données IMU pour apporter une cohérence des données.

3.3.2. ORB SLAM 3

Dans un premier temps, nous utilisons le github de thien94 [17] qui a créé un wrapper ROS pour ORB SLAM 3. Nous suivons le guide d'installation proposé en faisant attention au fait que les dépendances ORB SLAM 3 doivent être installées avant. Nous récupérons ensuite les fichiers YAML concernant les caméras. Une modification des options de lancement est requise ainsi qu'une modification de certains fichiers pour la compatibilité par rapport à notre flux d'images.

Un point d'attention particulier est la séparation en gyromètre et accéléromètre des données IMU (Unité de Mesure Inertielle). Il faut les regrouper afin de faire fonctionner l'algorithme qui ne requiert qu'un seul topic de données IMU. Il faut donc passer par la librairie Intel et non l'utilitaire Intel RealSense Viewer.

Il est ensuite possible d'utiliser 'Evo' pour évaluer nos résultats de l'algorithme sur les données enregistrées.

3.3.3. *KIMERA*

Pour l'installation de Kimera, nous utilisons le github de MIT-SPARK [18] qui est le laboratoire qui a créé l'algorithme Kimera et aussi son wrapper ROS. Nous suivons le guide d'installation proposé en installant proprement ROS ainsi que toutes ces dépendances le tout dans un workspace dédié nommé catkin. Ensuite, nous pouvons commencer à installer le wrapper ROS de Kimera dans ce même workspace en suivant le guide. Attention, Il se peut qu'il y ait des problèmes de compatibilité entre gtsam et Kimera-VIO lors du build du workspace catkin (le workspace contenant toutes les dépendances de Kimera). Il est donc recommandé, pour pouvoir compiler Kimera et toutes ses dépendances, de suivre un poste github de la catégorie "Issues" [19]. Ce poste constitue un guide pour changer les versions des dépendances problématiques pour régler ce problème de version.

3.4. Méthode d'évaluation

Afin d'évaluer nos résultats, nous allons utiliser la librairie EVO [20], réalisée sous python, qui permet à partir de topics présents dans les fichiers .bag d'analyser notre SLAM, i.e. tracer la trajectoire à partir des calculs de pose et comparer deux trajectoires calculées avec les erreurs absolues (Absolute Pose Error) et relatives (Relative pose Error). L'APE calcule à chaque instant de la trajectoire l'écart entre la référence et la trajectoire estimée. C'est la mesure la plus utilisée pour le SLAM dans la littérature. La RPE mesure la dérive de l'estimation par rapport à la référence : cette mesure est pratique pour montrer les périodes les plus influentes sur la dérive pour une séquence.

Cet outil convient donc parfaitement à notre objectif, en fournissant des métriques interprétables [21].

4. RÉSULTATS

Maintenant que nous avons une implémentation fonctionnelle de l'algorithme d'ORB SLAM 3 et Kimera-VIO, ainsi que des outils pour évaluer ces algorithmes, l'objectif est donc de tester nos algorithmes de SLAM. Pour cela nous utilisons nos acquisitions et celles de EuRoC, puis nous comparons les résultats sur plusieurs aspects. Dans un premier temps, sur le SLAM visuel et le SLAM Visio-Inertiel. Ensuite sur les trajectoires avec et sans fermeture de boucle et les performances des caméras pour un même trajet. Enfin, nous comparons les algorithmes choisis entre eux.

À partir de tous ces paramètres, nous pourrons alors proposer les conditions idéales de SLAM pour une utilisation avec le MuSHR en indoor.

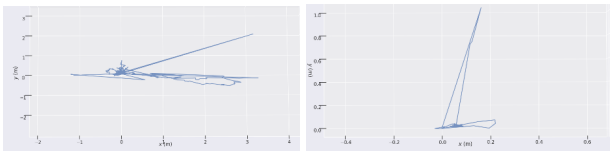

Figure 8: Trajectoires déterminées par ORB SLAM 3 sur les acquisitions avec fermeture de boucle. De gauche à droite : Acquisitions n°2, n°4. L'acquisition dans l'escalier n'a pas permis à ORB SLAM 3 de détecter des points d'intérêts.

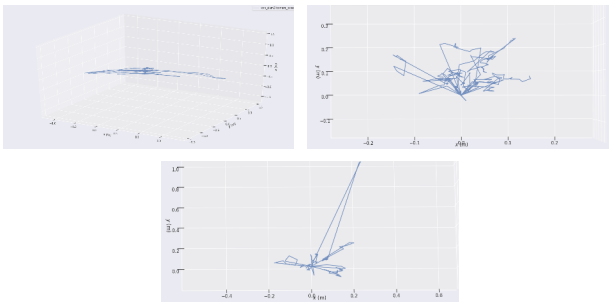

Figure 9: Trajectoires déterminées par ORB SLAM 3 sur les acquisitions sans fermeture de boucle. De gauche à droite : Acquisitions n°1, n°2, n°4.

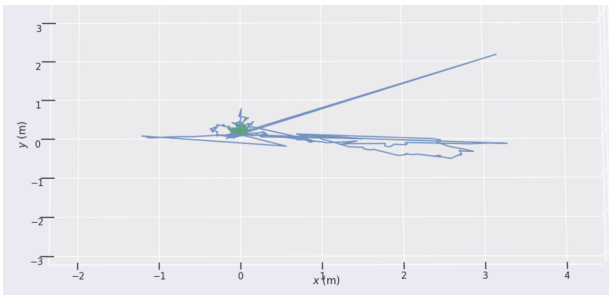

Figure 10: Comparaison entre les trajectoires calculées avec fermeture de boucle (en bleu) et sans fermeture de boucle (en vert) pour l'acquisition n°2.

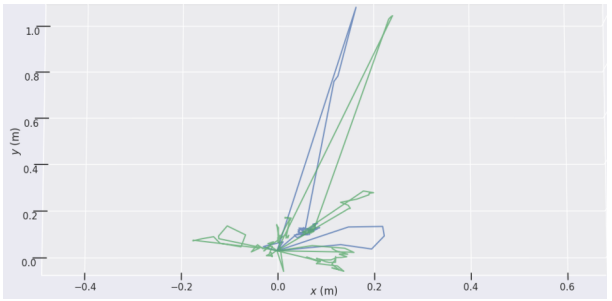

Figure 11: Comparaison entre les trajectoires calculées avec fermeture de boucle (en bleu) et sans fermeture de boucle (en vert) pour l'acquisition n°4

4.1. Évaluation qualitative des approches

Nous fournissons donc nos acquisitions sous format Ros bag aux deux algorithmes que nous avons implémenté : ORB SLAM 3 et KIMERA. Nous utilisons ensuite la

librairie python 'Evo' afin de traiter les résultats des SLAMs.

4.1.1. ORB SLAM 3

Dans un premier temps, nous testons ORB SLAM 3. Pour ce faire, nous modifions les fichiers de lancement et de configuration de l'algorithme afin de correspondre avec les paramètres et topics ROS de nos deux caméras. Cependant, le format d'image enregistré par la T265 (stéréo fisheye) est incompatible avec l'implémentation de ORB SLAM 3. Nous n'avons donc pu tester nos algorithmes uniquement avec les acquisitions faites par la D435i. Nous récupérons les données de trajectoire produites par ORB SLAM 3, pour les tracer avec EVO. Nous obtenons les résultats des figures 8, 9, 10 et 11 ci-dessus.

Tout d'abord, nous remarquons sur les Figure 8 et Figure 9 que l'acquisition n°3 dans les escaliers n'a donné aucun résultat de la part d'ORB SLAM 3. En effet, les environnements « escaliers » sont connus pour être particulièrement difficiles pour la réalisation de SLAM : la détection de points d'intérêts dans une pente est compliquée, car la caméra est orientée majoritairement sur le sol ou sur le plafond, donc des surfaces homogènes. ORB SLAM 3 détecte des points d'intérêts sur les hétérogénéités d'une surface (coins, meubles), notre cage d'escalier aux murs blancs unis et le fait que la luminosité oscille entre très basse et élevée au niveau des puits de lumière, explique l'échec du SLAM sur cette acquisition.

Ensuite, nous pouvons observer que les trajectoires varient beaucoup entre les acquisitions avec et sans fermeture de boucle notamment grâce à la Figure 11. Les trajectoires sont imparfaites, notamment lors du demi-tour de la boucle. Entre les acquisitions n°2 (couloir avec fenêtres) et n°4 (couloir sans fenêtre), la n°4 est celle qui reste le plus stable avec ou sans fermeture de boucle, bien qu'ayant une grosse anomalie vers le centre de la trajectoire.

La *Figure 10* nous permet de constater les différences de précision engendrées par la fermeture de boucle, et notamment sur l'acquisition n°2. Sans la fermeture de boucle, l'algorithme ne réussit pas à déterminer la trajectoire et concentre ses positions calculées autour du point de départ.

Enfin, nous exécutons l'algorithme sur la base de données EuRoC MAV (qui se rapproche le plus de nos conditions) pour tester l'importance de l'IMU dans les performances du SLAM.

Sur la Figure 13, nous pouvons observer que pour un même enregistrement, l'algorithme va calculer des trajectoire différentes en mono et en stéréo. Dans le calcul de la trajectoire en mono, l'erreur va être bien plus

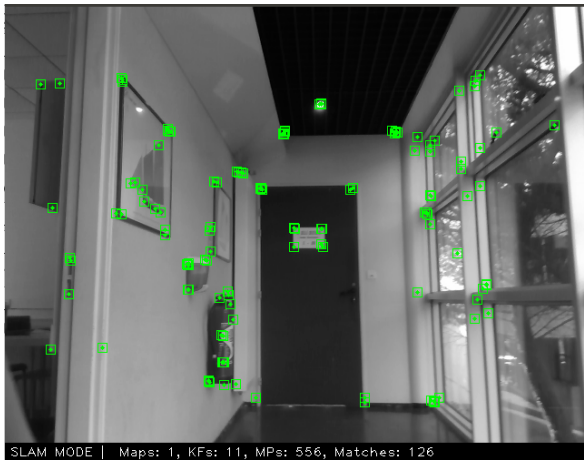

Figure 12: Exemple de points d'intérêts détectés par ORB SLAM 3 pour l'acquisition n°2.

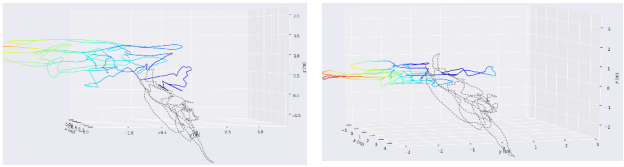

Figure 13: APE sur un enregistrement EuRoC en mono (à gauche) et stéréo (à droite) avec IMU (en couleur) et sans IMU (en gris pointillé) calculé avec ORB SLAM 3

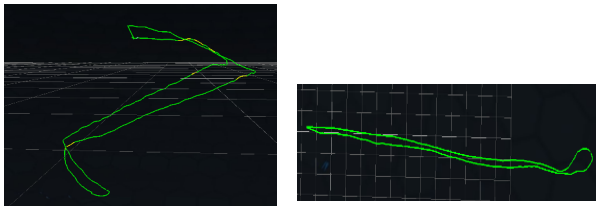

Figure 14: Trajectoires calculées par l'outil de tracking de la T256. A gauche, la trajectoire dans l'escalier. A droite, dans le couloir.

importante, ces résultats sont normaux vu que l'algorithme est beaucoup plus efficace et précis pour le calcul de la trajectoire avec une acquisition effectuée avec deux caméras en stéréo-vision. De plus, nous avons ici la comparaison directe entre l'erreur avec et sans l'IMU. Sur cette même figure, l'APE sans IMU est bien visible et est présente en stéréo et en mono. La dérive est d'autant plus importante sur l'axe vertical (*plusieurs mètres*).

4.1.2. Trajectoires INTEL de la T265

Bien que nous n'ayons pas pu traiter les images de la T256 avec ORB SLAM 3, nous pouvons observer sur le Figure 14 que nous avons accès aux trajectoires qu'elle calcule elle-même en interne. La trajectoire calculée est assez précise, et notamment dans les escaliers, là où ORB SLAM 3 n'a rien pu détecter.

4.1.3. KIMERA

Une fois l'installation de Kimera sur ROS finie, nous avons pu continuer le tutoriel pour pouvoir tester l'algorithme. La base de données pour le test utilisée est EuRoC MAV. Plus particulièrement un enregistrement stéréo d'un micro-véhicule aérien qui se déplace dans une pièce fermée avec fenêtres.

La Figure 15 nous permet de visualiser les résultats obtenus. En effet, grâce à l'application de visualisation Rviz, nous pouvons observer le calcul des points d'intérêts de Kimera-VIO sur l'image courante de l'enregistrement. Nous pouvons même obtenir la reconstruction 3D de notre enregistrement stéréo ainsi que le mapping de l'environnement en temps réel.

Cependant les résultats obtenus avec Kimera s'arrêtent ici. L'utilisation de machines virtuelles pour le fonctionnement de nos algorithmes était extrêmement limitant. Nous n'avons pas réussi à faire fonctionner Kimera-VIO sur nos propres acquisitions en raison de nombreux problèmes liés à ces machines virtuelles. Nous avons choisi de nous concentrer sur Orb Slam 3 plutôt que de passer encore plus de temps pour éventuellement avoir des résultats de Kimera tant les problèmes étaient nombreux.

4.1.4. Discussion

En résumé, nos résultats nous montrent que les deux facteurs les plus impactants sur le SLAM sont bien la présence d'un IMU et la nature de l'environnement. Le premier est que l'IMU offre une précision non-négligeable à la trajectoire calculée. Et le second, un environnement sans disparités sur les surfaces engendre une perte de précision (*cf erreur de trajectoire pour l'acquisition n°4)*

Pour finir, les trajectoires calculées par le tracking intégré à la T256 semble prometteuse, notamment dans les escaliers : cette caméra pourrait être la plus adaptée pour un SLAM-VI.

4.2. Partage avec le Git

Une partie importante du projet industriel sur lequel les encadrants de projet entreprise ont mis un point d'honneur, est la participation active au projet MuSHR sur GitHub.

Par exemple nous publions l'état de l'art, après l'avoir finis, sur le GitHub de notre projet [22]. Nous publions également un commentaire en Issue sur le projet MuSHR à la demande de nos encadrants, pour avoir l'avis de personnes travaillant déjà sur le projet. Nous avons des réponses [23] nous donnant des informations cruciales, notamment sur les performances de la carte Jeston nano présente sur le MuSHR qui limite grandement

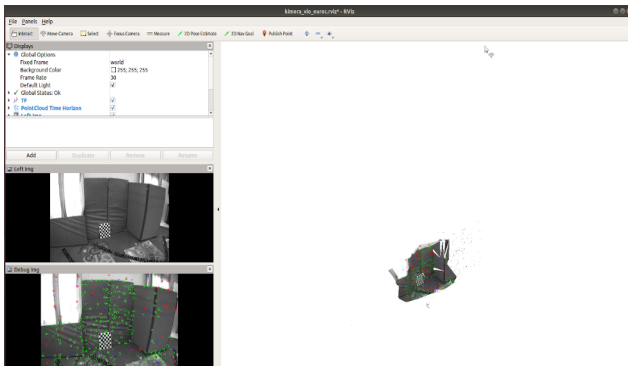

Figure 15: Visualisation de Kimera-VIO avec ROS dans Rviz sur un enregistrement stéréo d'EuRoC MAV dataset.

les possibilités. Ce retour nous permet également d'en savoir un peu plus sur certains algorithmes comme VINS-Fusion. Ils nous ont par ailleurs fourni une documentation utile pour ROS.

Une seconde partie de ce travail de documentation est de publier des rapports de ce que nous faisons sur notre dépot git SlaSHR [22]. Comme spécifié plus tôt, le gros de notre travail est l'installation des différentes solutions choisies. Nous publions des rapports d'installation complémentaires aux tutoriels existants pour les différents algorithmes permettant aux prochaines personnes voulant se plonger dans le sujet de bénéficier de notre expérience.

Dans un premier lieu nous commentons et précisons certaines étapes de l'installation de ORB SLAM 3, notamment avec des exemples concernant notre matériel (la caméra D435i et ses données IMU). Nous commentons également les difficultés rencontrées avec l'installation de Kimera en apportant des solutions que nous trouvons, comme des notices d'installation ou des dépendances à installer. Nous proposons enfin des perspectives d'amélioration pour continuer le projet.

5. CONCLUSION

Pour notre projet d'implémentation d'un algorithme SLAM sur un robot MuSHR, nous retenons trois algorithmes dont deux que nous réussissons à faire fonctionner et à tester dans le temps imparti du projet. Nos critères de sélection se basent sur une approche de navigation en intérieur sans présence de signaux GNSS. Le projet se place dans un cadre de recherche pour la DGA-MI et il est donc important d'analyser nos résultats et de les comparer entre eux afin de fournir le retour le plus précis et intéressant possible pour le futur développement de ce projet.

Les approches retenues sont les algorithmes ORBSLAM 3, KIMERA et XIVO. Durant cette période de recherche, nous avons obtenus des résultats concret et

pertinents pour le premier. Les résultats obtenus avec ces méthodes montrent que le type de caméra et les conditions extérieures jouent un grand rôle dans la précision de la trajectoire calculée. Il est alors nécessaire de déterminer les conditions optimales pour chaque caméra, afin de les utiliser de manière adaptative en fonction de l'environnement de navigation. Les environnements sans disparités, avec une variation de hauteur, ou avec un éclairage variant sont les plus difficiles à traiter. La présence d'un IMU est indispensable pour une localisation précise et sans dérive. Enfin, l'environnement de développement autour du MuSHR nécessite une attention particulière, car l'objectif final est de parvenir à du temps-réel, ce qui peut être compromis par des limitations techniques à l'heure actuelle

6. PERSPECTIVES

Lors de ce projet, nous avons rencontré de nombreuses difficultés liées à l'environnement utilisé par le MuSHR. Un point principal d'amélioration serait donc d'effectuer la transition de ROS1 vers ROS2, car celui-ci offre de nouvelles possibilités, et permet également d'utiliser des frameworks compatibles avec des versions plus récentes de python (>2.7). Cela est important car l'outil 'EVO' que nous avons utilisé pour évaluer les SLAM n'est plus maintenu que sur les versions de python les plus récentes.

Ensuite, une étude plus paramétrique des SLAM sur les algorithmes choisis permettrait d'avoir des résultats plus précis. Les paramètres évalués pourront être, par exemple, l'impact du nombre de points détectés, la qualité des données de l'IMU…

Étant donné que nous n'avons pas pu exploiter les données de la caméra de tracking (T256), une comparaison entre les deux types de caméras reste nécessaire pour la suite. Il pourrait même être envisagé de les utiliser en simultané pour les calculs.

Enfin, il serait intéressant de réaliser une base de données d'acquisitions indoor faites par le MuSHR en conditions réelles, mais surtout de mesurer une réalité terrain à laquelle se comparer, choses dont nous ne disposions pas lors de ce projet.

RÉFÉRENCES

[1] « ORB-SLAM3 ». SLAMLab - Universidad de Zaragoza, 13 février 2023. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/UZ-SLAMLab/ORB_SLAM3

[2] « MuSHR: The UW Open Racecar Project ». https://prl-mushr.github.io (consulté le 13 février 2023).

[3] « Discussions · prl-mushr/mushr », *GitHub*. https://github.com/prl-mushr/mushr (consulté le 13 février 2023).

[4] « ROS: Home ». https://www.ros.org/ (consulté le 13 février 2023).

[5] « Kimera ». MIT-SPARK, 9 février 2023. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/MIT-SPARK/Kimera

[6] « XIVO: X Inertial-aided Visual Odometry and Sparse Mapping ». UCLA Vision Lab, 10 février 2023. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/ucla-vision/xivo

[7] « OKVIS ». ETHZ ASL, 4 février 2023. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/ethz-asl/okvis

[8] « Vladyslav Usenko / basalt · GitLab », *GitLab*, 25 janvier 2023. https://gitlab.com/VladyslavUsenko/basalt (consulté le 13 février 2023).

[9] « rovio - ROS Wiki ». http://wiki.ros.org/rovio (consulté le 13 février 2023).

[10] W. Jing, « ORB-SLAM2 ». 1 février 2023. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/jingpang/LearnVIORB

[11] T. Qin, P. Li, et S. Shen, « VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator », *IEEE Trans. Robot.*, vol. 34, nº 4, p. 1004-1020, août 2018, doi: 10.1109/TRO.2018.2853729.

[12] « ROVIOLI Introduction », *GitHub*. https://github.com/ethz-asl/maplab/wiki/ROVIOLI-Introduction (consulté le 13 février 2023).

[13] « VINS-Fusion ». HKUST Aerial Robotics Group, 13 février 2023. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/HKUST-Aerial-Robotics/VINS-Fusion

[14] M. Burri *et al.*, « The EuRoC micro aerial vehicle datasets », *Int. J. Robot. Res.*, vol. 35, nº 10, p. 1157-1163, sept. 2016, doi: 10.1177/0278364915620033.

[15] A. Geiger, P. Lenz, C. Stiller, et R. Urtasun, « Vision meets robotics: the KITTI dataset », *Int. J. Robot. Res.*, vol. 32, p. 1231-1237, sept. 2013, doi: 10.1177/0278364913491297.

[16] X. Fei, « VISMA dataset tools ». 11 novembre 2022. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/feixh/VISMA

[17] zhaozhong chen, « ORB-SLAM3 ROS wrapper ». 8 février 2023. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/zhaozhongch/orbslam3_ros

[18] « MIT-SPARK/Kimera: Index repo for Kimera code ». https://github.com/MIT-SPARK/Kimera (consulté le 13 février 2023).

[19] « [TIP] A successful installation of Kimera-VIO-ROS Guide · Issue #180 · MIT-SPARK/Kimera-VIO-ROS », *GitHub*. https://github.com/MIT-SPARK/Kimera-VIO-ROS/issues/180 (consulté le 13 février 2023).

[20] M. Grupp, « evo ». 13 février 2023. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/MichaelGrupp/evo

[21] D. Prokhorov, D. Zhukov, O. Barinova, K. Anton, et A. Vorontsova, « Measuring robustness of Visual SLAM », in *2019 16th International Conference on Machine Vision Applications*

(MVA), mai 2019, p. 1-6. doi: 10.23919/MVA.2019.8758020.
[22] mgodineau, « mgodineau/SlaSHR ». 27 octobre 2022. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://github.com/mgodineau/SlaSHR
[23] « SLAM-VI on MuSHR, looking for advices · Discussion #94 · prl-mushr/mushr », *GitHub*. https://github.com/prl-mushr/mushr/discussions/94 (consulté le 13 février 2023).
[24] A. Angeli, « Détection visuelle de fermeture de boucle et applications à la localisation et cartographie simultanées. (Visual SLAM applications of loop-closure detection) », 2008. Consulté le: 13 février 2023. [En ligne]. Disponible sur: https://www.semanticscholar.org/paper/D%C3%A9tection-visuelle-de-fermeture-de-boucle-et-%C3%A0-la-Angeli/d9ffd9f90a14bb5b80fb7eecdca2ff62fde50bf4