
Projet tutoré ingénierie et conception : Visualisation en 3D du véhicule évoluant à faible vitesse dans son environnement extérieur

JIANHAO RUAN DIMITRI FREON

Table des matières

1	Introduction	1
1.1	Objectifs	1
1.2	Travaux réalisés précédemment	1
2	Solution	2
2.1	Matériel	2
2.2	Pistes explorées	5
2.2.1	Jetson	5
2.2.2	Utilisation de ZEDfu	5
2.2.3	Utilisation de Unity et des interfaces ZED Unity	6
2.2.4	Utilisation des API python	6
2.3	Solution Finale	7
2.3.1	Les nodes de ROS	7
2.3.2	Traitements des données	8
2.3.3	Affichage	9
3	Tests et résultats	11
3.1	Scénarios de test	11
3.2	Géométrie de la Plate-forme	12
3.3	Résultats	13
4	Conclusion	14
4.1	Objectifs	14
4.2	Pistes d'améliorations	14

Introduction

1.1 Objectifs

Le pilotage d'un véhicule s'effectue à l'aide des informations visuelles qu'on perçoit autour du véhicule et dans la majorité des cas cela suffit à éviter les différents obstacles. Cependant dans des contextes particulier comme un terrain particulièrement irrégulier le manque d'informations concernant l'environnement sous le capot et au niveau des roues peut affecter la capacité à franchir des obstacles et compliquer grandement le pilotage. Dans cette optique nous avons réaliser ce projet en collaboration avec Arquus avec pour objectif d'analyser et de reconstruire en 3D l'environnement autour d'un véhicule pour permettre la visualisation en temps réel de ce qu'il se trouve sous le capot, au niveau des roues du véhicule.

Cette objectif se décompose en plusieurs parties, tout d'abord dans la continuité des travaux de l'an dernier la solution va cette fois ci s'articuler autour d'une caméra stéréo (et non d'un Lidar) qu'il a fallu choisir puis prendre en main pour répondre au mieux aux objectifs. Ensuite niveau logiciel il y a également cette idée de continuité en reprenant la méthodologie et les concepts des algorithmes de la solution de l'année précédente en essayant de les adapter et de les améliorer. Et enfin on a également procéder à une phase de test sur le terrain dans différents scénarios avec un véhicule d'Arquus.

1.2 Travaux réalisés précédemment

L'année précédente un groupe a déjà eu l'occasion de travailler sur ce sujet avec un Lidar et ils ont pu déjà soulever des problématiques, liées notamment au matériel, et des pistes algorithmiques pour répondre aux besoins.

Au niveau de l'algorithme l'idée générale était de capturer les données 3D sous forme de nuage de points, de les stocker sur une distance suffisamment grande pour que ça atteigne les roues puis d'afficher en fonction du déplacement et donc de la vitesse du véhicule. Cette idée nous a servi de base pour notre propre solution.

Un des principaux problèmes du Lidar c'est sa faiblesse face à des surfaces foncées ce qui provoque de nombreuses pertes de données dans beaucoup de situations notamment avec les ombres. Le choix de la caméra stéréo ici apporte donc de la stabilité au niveau de l'acquisition des données.

Un autre problème soulevé était celui de la vitesse, en effet dans l'optique de réaliser un défilement de l'image 3D à la bonne vitesse il est crucial de connaître la vitesse du véhicule de manière précise. Leur méthode de calcul de vitesse à partir d'algorithme de traitement d'image était assez peu concluant on a donc décidé d'utiliser une autre méthode pour calculer la vitesse.

Solution

2.1 Matériel

Comme indiqué auparavant on a choisi de prendre une caméra stéréo pour régler les différents problèmes liés à l'utilisation du Lidar. Il existe plusieurs marque de caméra stéréo mais les 2 plus grosses sont Intel RealSense et StereoLabs. Le choix s'est porté sur la caméra ZED2i de Stereolabs qui assure une très bonne qualité d'image, possède une portée pouvant allait de 0.3m à 20m, une centrale inertuelle intégrée et une résistance à l'eau et à la poussière (utile en cas de pluie ou sur des chemins).



FIGURE 1 – Caméra stéréo ZED2i

Focal Length	2.12mm (0.008")
Field of View	Max.110°(H) x 70°(V) x 120°(D)
Aperture	f/1.8
TV Distortion	5.07%
Depth Range	0.3 m to 20 m (1 to 65.6ft)
Depth Accuracy	< 1% up to 3m < 5% up to 15m

FIGURE 2 – Spécifications ZED2i

Pour fixer la caméra sur le capot de la voiture on utilise un support avec trois ventouses et une rallonge USB pour relier la caméra à notre PC portable qui contient le logiciel



FIGURE 3 – Trépied à ventouses



FIGURE 4 – Rallonge USB

La Zed est une caméra de profondeur binoculaire. La figure 5 montre la même scène capturée par les caméras gauche et droite de la Zed2i.

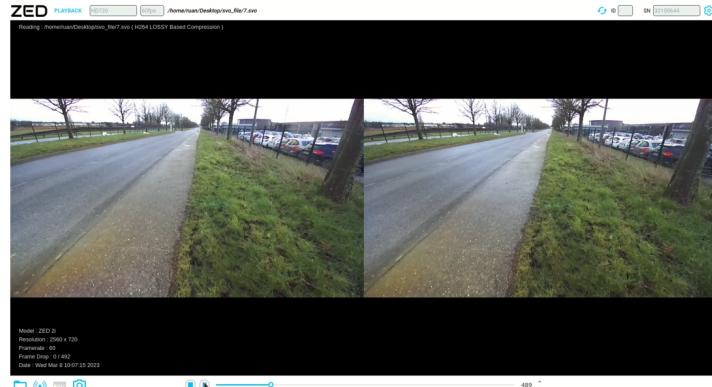


FIGURE 5 – La vue stéréo du caméra.

La caméra sélectionne les points correspondants dans les deux images sur la base de l'algorithme de correspondance des points caractéristiques. Elle calcule les informations de profondeur en utilisant la triangulation (re-projection) à partir du modèle géométrique des caméras rectifiées non déformées [2]. En supposant que les deux caméras sont coplanaires avec des axes optiques parallèles et la même longueur focale $f_l = f_r$, la profondeur Z de chaque point L est calculée par l'équation :

$$Z = \frac{fB}{xi^l - xi^r} \quad (2.1)$$

, où B est la distance de base et $xi^l - xi^r$ est la valeur de disparité.

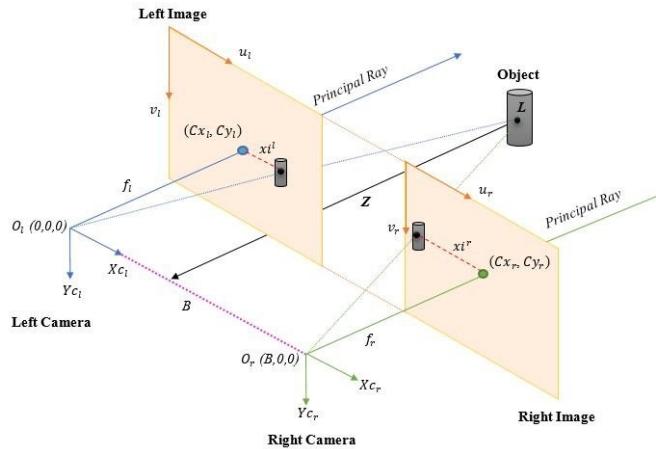


FIGURE 6 – La principe de fonctionnement de la ZED 2i. [2]

Le niveau de confiance de la correspondance des points est indiqué dans la figure 7 droite, qui indique la confiance de la valeur profondeur de la position correspondante.

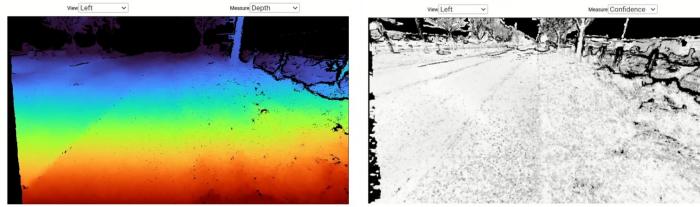


FIGURE 7 – L'image de profondeur et l'image de confiance.

La reconstruction 3D de la caméra ZED2i est sous forme de nuage des points et est réalisée par l'algorithme implémentée dans le calculateur embarquée de la caméra. La fréquence du flux de données fourni à l'utilisateur est de 14Hz. La figure montre l'exemple de reconstruction 3D de la scène.

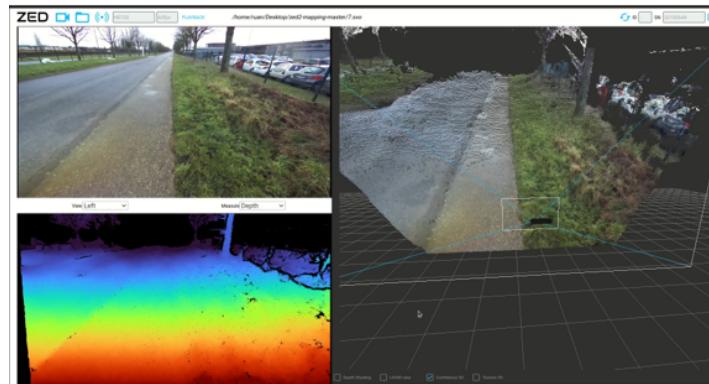


FIGURE 8 – La fonction de reconstruction 3D intégrée dans le caméra.

Les coordonnées des points du nuage de points collectés par la caméra ZED2i sont exprimées dans un système de coordonnées, dont l'origine est le centre optique de la caméra gauche. Les directions de l'axe x, y et z sont définie de manière légèrement différente pour les différentes interfaces informatiques, comme le montre la figure 9.

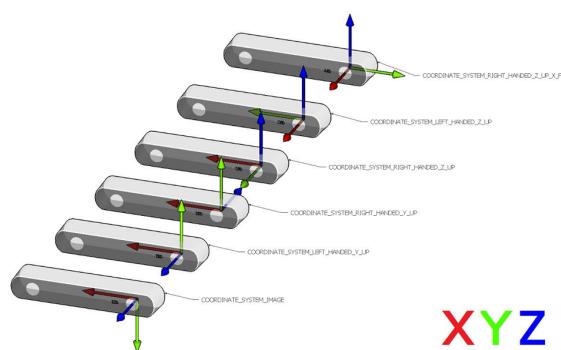


FIGURE 9 – Système de coordonnées de la caméra sous différentes interface.

2.2 Pistes explorées

2.2.1 Jetson

La caméra ZED2i fonctionne avec le logiciel de stereolabs ZED SDK et pour toute utilisation il est obligatoire de l'avoir installé. Une problématique du projet a été la nécessité de posséder une carte graphique pour pouvoir installer et faire fonctionner le ZED SDK. On avait à notre disposition un PC portable avec carte graphique mais on a également exploré la solution d'utiliser un kit de développement Jetson TX2 qui est spécifiquement conçu pour ce genre d'utilisation.



FIGURE 10 – Jetson TX2

Ce Jetson TX2 nous a finalement permis de développer et tester sans véhicule les différents algorithmes implémentés mais la difficulté à rendre son utilisation portable (dans une voiture) nous a amené à nous en servir uniquement pour le développement.

2.2.2 Utilisation de ZEDfu

ZEDfu est une méthode SLAM visuelle stéréo. Les développeurs de Stereolabs ont fourni des caméras avec divers outils et interfaces. La caméra stéréo ZEDfu peut être utilisée pour générer une carte 3D de l'environnement sous la forme d'un nuage de points[1].

Nous avons utilisé ZEDfu pour reconstruire la scène, mais le problème est que sa vitesse de reconstruction est lente et ne convient pas aux applications en temps réel.

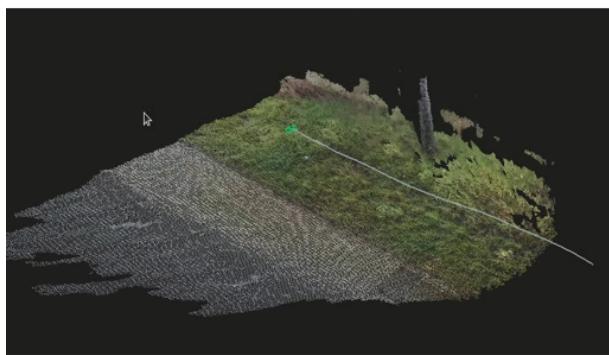


FIGURE 12 – Scénario reconstruite sous forme de Nuage des points avec trajet de mouvement, par ZEDfu.

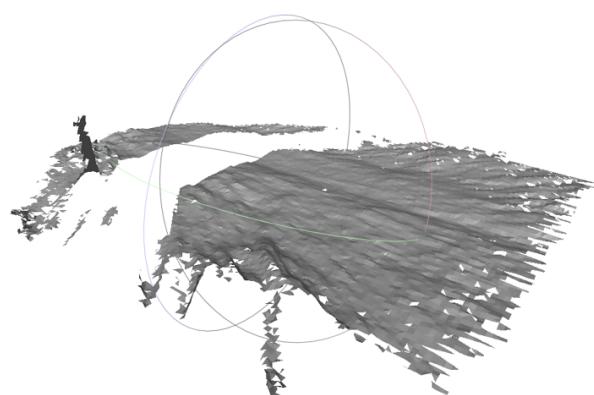


FIGURE 13 – Le Mesh enregistré par ZEDfu.

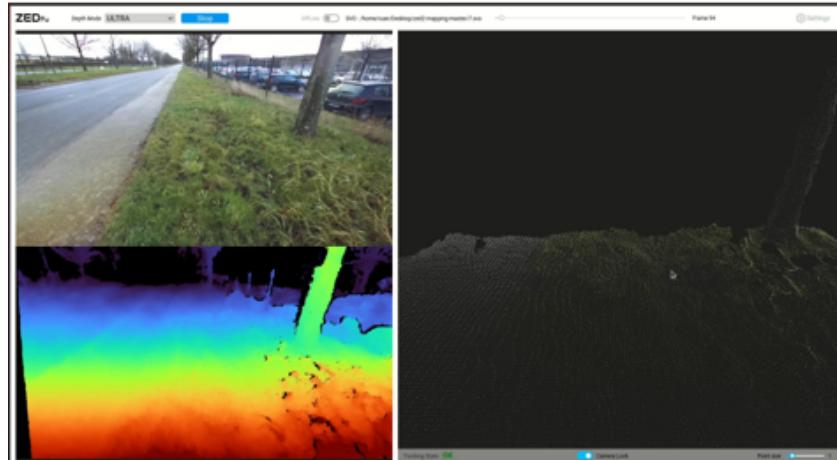


FIGURE 11 – Interface de pilotage ZEDfu.

2.2.3 Utilisation de Unity et des interfaces ZED Unity

Les travaux de l'année précédente avait été réalisés sous Unity un moteur 3D qui permettait d'afficher de manière assez simple un nuage de points et de modéliser les roues et la caméra. Dans une optique de continuité nous avions d'abord pensé à utiliser Unity également, d'autant plus qu'il existe des interfaces déjà existantes entre le ZED SDK et Unity. On a donc pu afficher de manière similaire un nuage de points capturé par la caméra dans l'interface Unity comme présenté ci dessous.



Beaucoup d'applications déjà existantes pour utiliser la caméra ZED avec Unity fonctionnent très bien (notamment beaucoup d'applications de VR) cependant pour notre projet l'objectif était de pouvoir récupérer de manière simple les nuages de points et d'effectuer un traitement (défilement notamment) avant de les afficher dans l'interface Unity. On s'est cependant heurté à la complexité d'utilisation des API dans Unity et donc en C# et la difficulté de compréhension des interfaces nécessaires à notre application nous a poussé à envisager une autre solution pour l'affichage.

2.2.4 Utilisation des API python

Une autre solution envisagé a été d'utiliser les API python de base qui permette d'accéder facilement aux différentes composantes de la camera et d'effectuer des opérations sur les données. On a notamment pu essayer le "spatial mapping" :

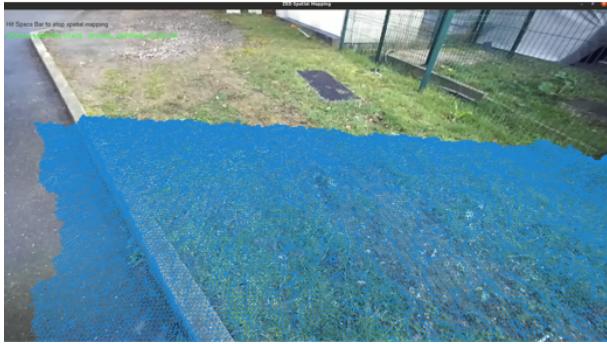


FIGURE 14 – Résolution haute.



FIGURE 15 – Résolution basse.

L'algorithme est suffisamment précis pour reconstruire de vastes zones et résiste aux rotations de la caméra à grande échelle. Cependant, il n'est pas aussi rapide qu'il pourrait l'être, avec 1 trame par seconde pour les reconstructions à haute résolution (figure 14) et 3 trames par seconde pour les reconstructions à basse résolution (figure 15).

Pour ce qui est de la partie nuage de point cependant en terme d'algorithme d'acquisition et de traitement des données cette méthode semblait être la bonne et a servi de base à l'algorithme final (également en python) mais cette fois ci le problème était de l'ordre de l'affichage. En effet pour l'affichage des données les scripts de base utilise la librairie OpenGL qui bien configuré permet bien d'afficher les données voulues (nuages de points, carte de profondeur, mesh) mais pour rajouter les roues, la caméra et changer facilement leurs positions cela n'était pas adapté.

2.3 Solution Finale

Pour la solution finale il fallait donc quelque chose qui nous permette à la fois de manipuler facilement les structures de données récupérées par la camera et également un affichage dans lequel on peut ajouter ou modifier un élément librement. On a donc décidé d'utiliser ROS qui permettait de faire tout ça en manipulant en Python avec les API ZED-ROS-Python les structures de données de manière efficace et en affichant avec RViz à la fois les nuages de points mais également le véhicule (roues + camera).



2.3.1 Les nodes de ROS

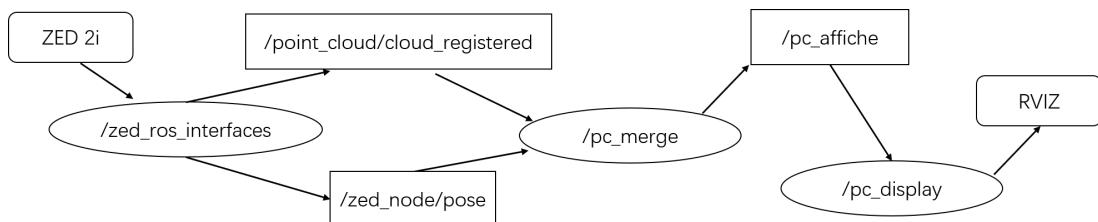


FIGURE 16 – Le rosgraph du système.

Le flux de traitement : tout d'abord, le noeud **/pc_merge** obtient le segment de nuage de points en temps réel ainsi que la pose à partir du sujet publié par le noeud **/zed_ros_interfaces**, puis le noeud **/pc_merge** procède à l'assemblage du segment de nuage de points et le publie ensuite en tant que topic **/pc_affiche**, le noeud **/pc_display** enregistre le topic **/pc_affiche** et affiche le résultat de l'assemblage dans rviz avec le modèle urdf de la caméra et des roues.

2.3.2 Traitement des données

Le nœud `/pc_merge` implémente l'algorithme de traitement des données. Les entrées du nœud sont les suivantes :

- `/point_cloud/cloud_registered` : le nuage des points capturé par ZED2i à l'instant t. Son format est de `sensor_msg.PointCloud2` (XYZ + RBG8), le type de message ROS.
- `/zed_node/pose` : la position et l'orientation de trame à l'instant t par rapport au position de l'instant 0. Il est estimé par la fonction **positional tracking** intégrée via ZED API.

Le **positional tracking** est la capacité du ZED2i à estimer sa position par rapport au monde qui l'entoure. Il est utilisé pour suivre le mouvement à six degrés de liberté (6DoF). La ZED utilise le suivi visuel de son environnement pour comprendre les mouvements du système qui le tient. Lorsque la caméra se déplace dans le monde réel, elle signale sa nouvelle position et son orientation.

La caméra ZED 2i est équipée d'un capteur inertiel capable de fournir des mesures fiables à haute fréquence des accélérations et des vitesses angulaires de la caméra en mouvement. Lorsqu'un capteur IMU (Inertial Measurement Unit) est disponible, les **informations inertielles** sont **fusionnées** avec les informations de **suivi visuel** pour fournir une estimation encore plus fiable des mouvements de la caméra.

L'API ZED renvoie des informations sur la pose pour chaque trame. Les données de pose suivantes sont renvoyées :

- Position : l'emplacement de la caméra dans l'espace est disponible sous la forme d'un vecteur [X, Y, Z].
- Orientation : l'orientation de la caméra dans l'espace est disponible sous la forme d'un quaternion [X, Y, Z, W].

La position et l'orientation de trame à l'instant t sont représentées dans la coordonnée de position de l'instant 0.

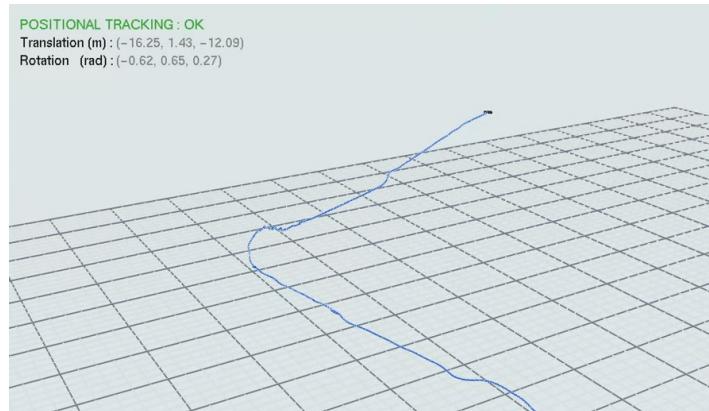


FIGURE 17 – Démonstration de **positional tracking**.

Avec les positions relatives reliées aux nuages des points, nous pouvons effectuer un premier assemblage du nuage de points avec des positions relatives. La figure 18 montre la processus de traitement du nœud `/pc_merge`.

Le nuage des points capturé à l'instant t est d'abord transformé par T , puis il est calibré par ICP [3] par rapport au nuage des points qui est maintenu et mis à jour. Le nuage des points capturé est ensuite transformé en coordonnée origine avec T_{inv} et est ajouté dans nuage des points maintenu. Le nuage des points maintenu est retransformée en coordonnée caméra et est fractionné. Le nœud publie le nuage des points pour afficher dans le topic `/pc_affiche`. Un échantillonnage aléatoire est effectué pour assurer que la mémoire n'est pas surchargée. Après l'échantillonnage, le nuage des points est retransformée dans les coordonnées de l'origine pour renouveler. Les opérations sur le nuage des points sont réalisées à l'aide de Open3D [4].

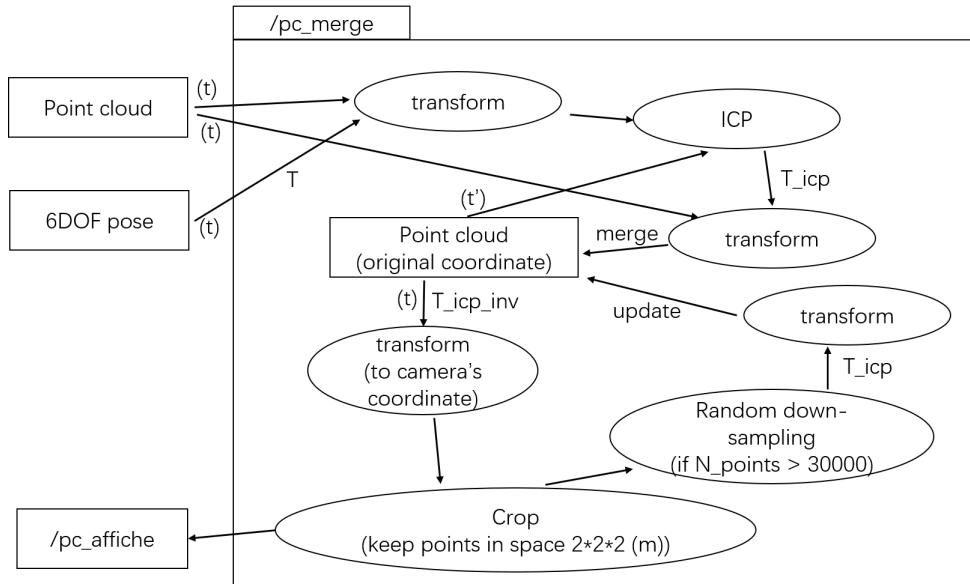


FIGURE 18 – Démonstration de processus `/pc_merge`.

2.3.3 Affichage

Pour l'affichage des données on utilise Rviz qui possède déjà une fonctionnalité d'affichage de nuage de points sous le format PointCloud2 et qui permet de visualiser soit classiquement les points avec leurs couleurs associés mais également un nuage de point colorés selon la profondeur selon un axe (utile selon l'axe Z pour mettre en évidence des obstacles).

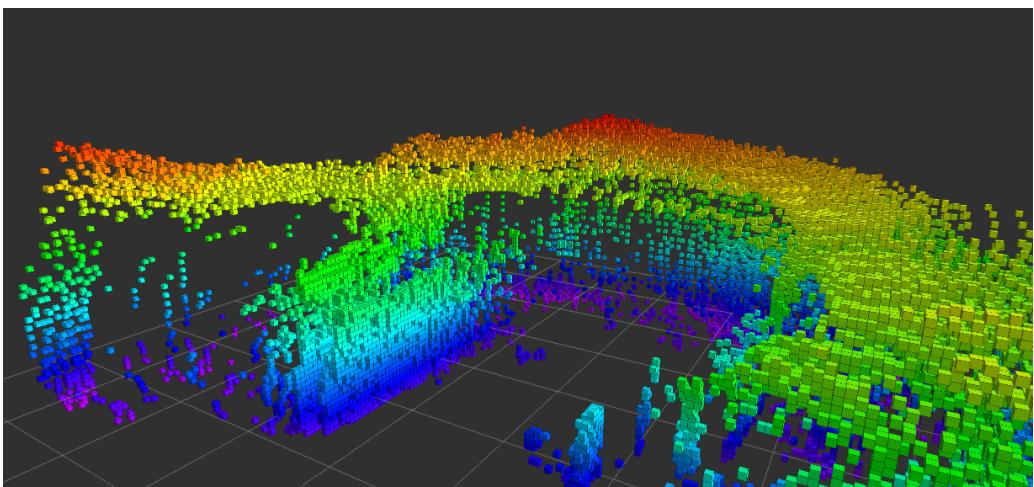


FIGURE 19 – Exemple de visualisation d'un nuage de point "coloré"

On rajoute ensuite la caméra et les roues sous la forme d'un modèle de robot qui est géré directement par ROS, il suffit alors de lancer le noeud " $robot_state_publisher$ " et de publier sur le bon topic. Le modèle de la caméra, les fichiers de description physique .urdf, existent déjà donc on peut récupérer une caméra fidèle à la réalité. On rajoute ensuite à cela des roues que l'on place selon différentes mesures (cf partie 3.2).

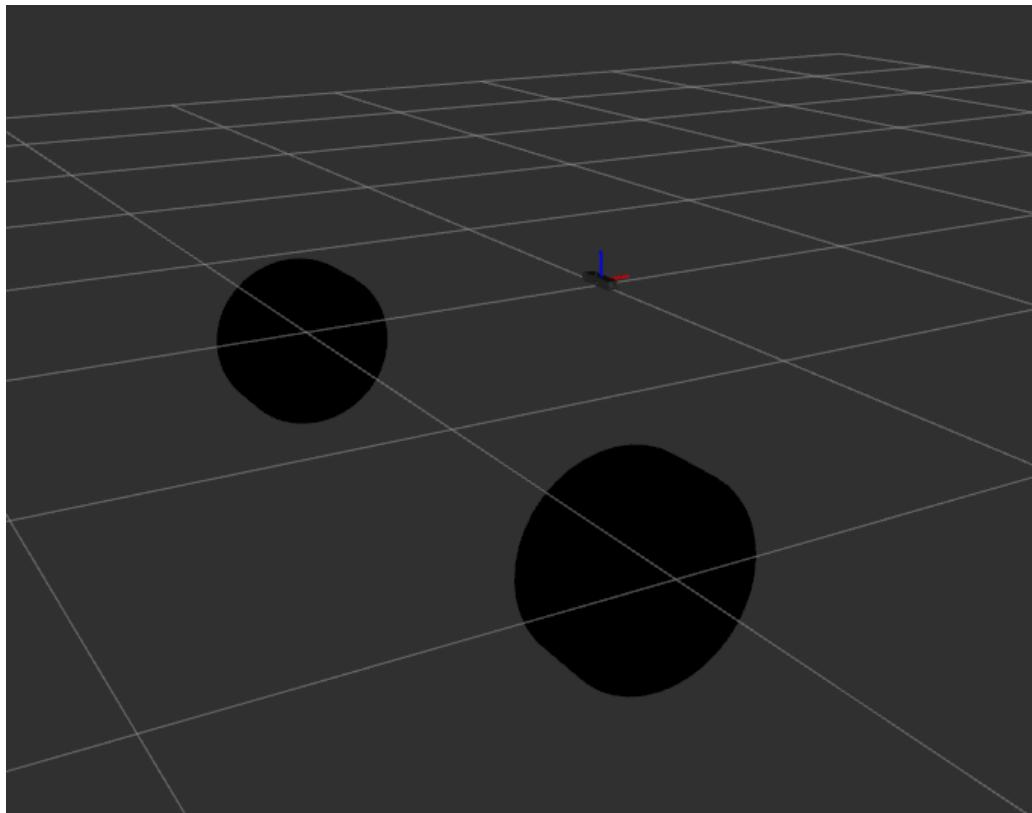


FIGURE 20 – Exemple de visualisation d'un nuage de point "coloré"

Tests et résultats

3.1 Scénarios de test

Pour réaliser des tests on s'est rendu chez Arquus pour utiliser un véhicule de chez eux sur lequel on a installé le dispositif. L'objectif était de tester différentes situations passage de trottoir, chemins avec des obstacles, fossés et également d'acquérir des enregistrements pour pouvoir régler de manière précise l'algorithme. Le dispositif était monté de la manière suivante :



FIGURE 21 – Dispositif de test vue de face



FIGURE 22 – Dispositif de test vue de coté

L'angle ($\approx 40^\circ$) et la position de la caméra ont été choisis en fonction de l'image obtenue, on a essayé d'avoir une image qui possède le plus d'éléments possible tout en assurant que les éléments en relief au sol soit bien détectés.

3.2 Géométrie de la Plate-forme

Afin de mesurer précisément la position de la caméra par rapport à la voiture, et en particulier aux roues, nous avons utilisé l'application IOS “Heges” pour reconstruire la scène en 3D. Cette application utilise le Lidar de l'Iphone pour mesurer la position 3D des points de la scène et utilise ensuite un algorithme de reconstruction 3D pour assembler les segments de la scène en une scène complète.

Les mesures sont exportées sous la forme d'un fichier .obj, que nous traitons à l'aide de Meshlab(figure 26). Nous mesurons d'abord les coordonnées des points clés, qui ont été représentés dans un système de coordonnées dont l'origine se situe au début du balayage. Ils sont (en mètre) :

- Centre camera : [0.032278 0.102365 -0.499337]
- Centre roue gauche : [-0.962336 -0.479768 -1.412287]
- Centre roue gauche contact terrain : [-1.018987 -0.779946 -1.587538]
- Centre roue droite : [0.550369 -0.088127 -1.871590]
- Centre roue droite contact terrain : [0.636688 -0.352260 -2.084967]



FIGURE 23 – Vue gauche.



FIGURE 24 – Vue centre.



FIGURE 25 – Vue droite.

Ensuite, à l'aide des outils de mesure de Meshlab, nous mesurons les distances clés nécessaires au processus de modélisation (en mètre) :

- Distance entre le centre du pneu gauche et le centre du pneu droit (M0) : 1.62689
- Distance entre le centre du pneu gauche et le sol (M1) : 0.373562
- Distance entre le centre du pneu droit et le sol (M2) : 0.365941
- Distance entre la caméra et le centre du pneu gauche (M3) : 1.471286
- Distance entre la caméra et le centre du pneu droit (M4) : 1.472871
- Distance entre la caméra et le sol (M5) : 1.289046

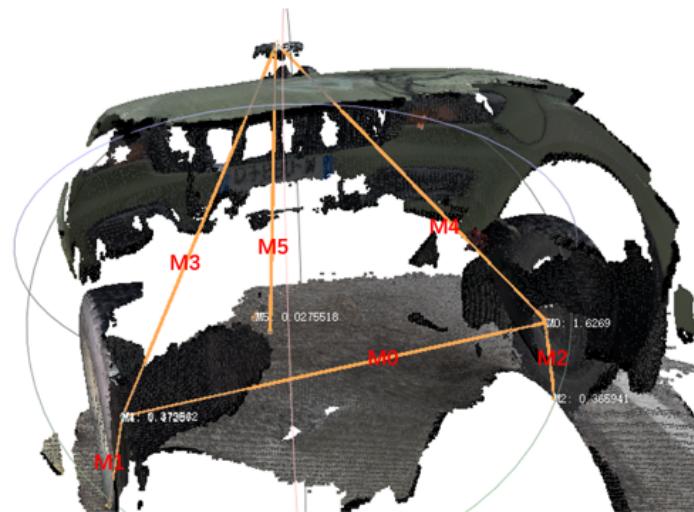


FIGURE 26 – Géométrie mesuré par Meshlab.

3.3 Résultats

On obtient les résultats suivants pour les nuages de points avec les couleurs de base :

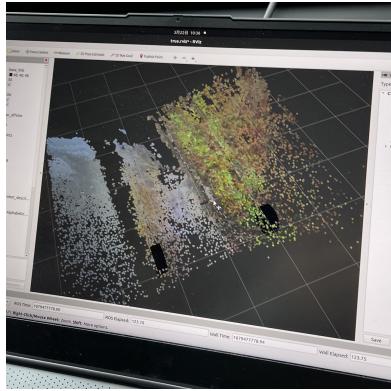


FIGURE 27 – Test nuage de point 1

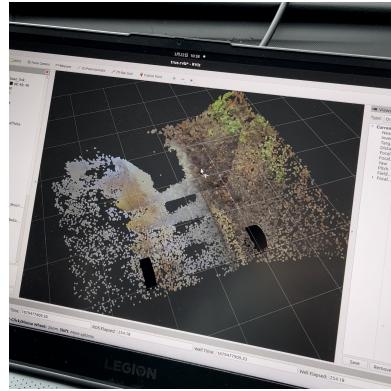


FIGURE 28 – Test nuage de point 2



FIGURE 29 – Test nuage de point 3

Et pour la visualisation colorée en fonction de la hauteur des points :

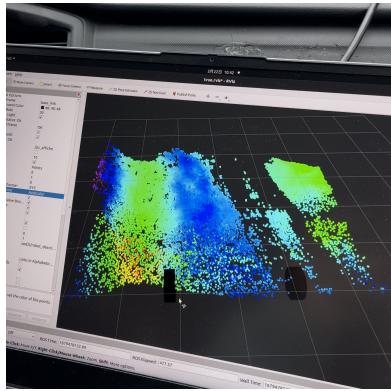


FIGURE 30 – Test nuage de point coloré 1

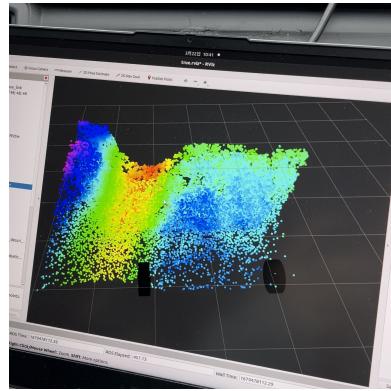


FIGURE 31 – Test nuage de point coloré 2

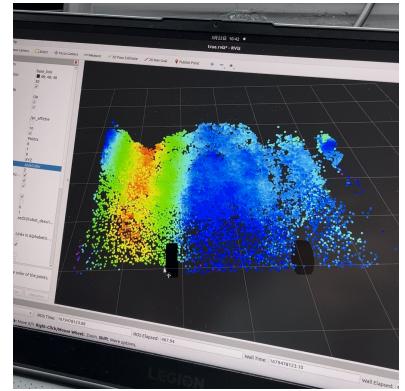


FIGURE 32 – Test nuage de point coloré 3

On observe donc que le rendu avec les couleurs réelles (figure 27,28,29) ou le rendu coloré (figure 30,31,32) peuvent permettre selon les situations de mieux visualiser le terrain. Dans les 1ères images les couleurs permettent de bien représenter le contraste entre la route et le trottoir avec de la terre et de l'herbe. On notera malgré tout une légère perte d'information proche du trottoir, peut être lié à la pluie et donc à l'eau qui se trouvait à cet endroit. Sur la version colorée ici cela met bien en évidence le fossé qui se trouve à gauche de la voiture et ainsi on se rend compte d'où se situe la roue par rapport à celui ci.

Conclusion

4.1 Objectifs

Comme on a pu voir au niveau des résultats, plusieurs objectifs ont été atteints. On a bien réussi à obtenir un affichage 3D de l'environnement sous le capot du véhicule, selon les conditions le rendu était plus ou moins correct mais globalement on arrive à discerner les principaux obstacles en relief et les zones plus ou moins profondes. Au niveau du défilement la vitesse récupérer par la caméra seulement semble assez bonne, elle reste évidemment une approximation cependant le défilement a été assez cohérent durant les différents tests réalisés.

4.2 Pistes d'améliorations

Cependant il reste encore beaucoup de choses à améliorer voire à ajouter à l'algorithme :

- La fonction API "spatial mapping" peut reconstruire la scène correctement mais sa vitesse n'est pas suffisant. La recommandation est d'utiliser C++ pour la programmation et d'intégrer l'accélération par GPU avec programmation parallèle.
- Un des problèmes qu'on a eu est la variation fréquente de la hauteur des points sans raison extérieur, certainement liée à l'estimation de la pose de la qui n'est pas parfaite. Une solution serait d'empêcher cela en vérifiant que la hauteur en moyenne reste similaire d'une boucle à l'autre.

- Concernant la vitesse on a donc estimé celle ci avec les données de la caméra directement cependant comme mentionné au début on a réfléchi à l'utilisation d'un boîtier type VBox qui permettrait de récupérer la vitesse à l'aide du logiciel CANalyzer. Par manque de temps on a seulement pu se connecter mais on a pas réussi à récupérer la vitesse. La récupération de la vitesse via ce boîtier pourrait améliorer la stabilité du défilement et permettre de meilleures performances.

- Pour l'affichage on a décidé d'utiliser Rviz dans une optique de simplicité et d'un outil adapté à ROS cependant le traitement des nuages de points se fait avec des objets de la librairie Open3D, ce qui nécessite une transformation des nuages de point ROS (PointCloud2) en nuage de points Open3D et une transformation inverse après calcul pour l'affichage. Ces opérations sont lourdes et les performances de l'algorithme n'étant pas si bonne que ça, cela pourrait être une bonne piste que d'éviter cette double transformation avec éventuellement un affichage via Open3D, ou une adaptation de l'algorithme de traitement des nuages de points avec les structures de données de ROS.

Bibliographie

- [1] Chinmay Kolhatkar and Kranti Wagle. Review of slam algorithms for indoor mobile robot with lidar and rgb-d camera technology. *Innovations in Electrical and Electronic Engineering : Proceedings of ICEEE 2020*, pages 397–409, 2021.
- [2] Luis Enrique Ortiz, Elizabeth V Cabrera, and Luiz M Gonçalves. Depth data error modeling of the zed 3d vision sensor from stereolabs. *ELCVIA : electronic letters on computer vision and image analysis*, 17(1) :0001–15, 2018.
- [3] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics : science and systems*, volume 2, page 435. Seattle, WA, 2009.
- [4] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d : A modern library for 3d data processing. *arXiv preprint arXiv :1801.09847*, 2018.