

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DES COURS

GTI791 Projet spécial en génie des TI
LOG791 Projet spécial en génie logiciel
LOG792 Projet de fin d'études en génie logiciel

Reconstruction tridimensionnelle automatisée d'un environnement fermé

Mathieu Gascon-Lefebvre
GASM07049007

Junior Grégoire
GREJ25078401

Gabriel Kenderessy
KENG25049006

Département de génie logiciel et des TI

Professeur-superviseur
Luc Duong

MONTRÉAL, 13 août 2014
Été 2014

REMERCIEMENTS

Nous tenons à remercier:

Les membres du club Capra ayant participé à la conception du robot et nous ayant permis de l'utiliser pour notre projet.

Les commanditaires du club Capra.

Patrick Laurin pour nous avoir aidés à bien utiliser le «Pan Tilt Unit».

Le club Dronolab pour nous avoir permis d'effectuer des tests avec leur IMU.

Luc Duong pour avoir accepté de nous superviser et pour ses conseils.

Nos amis et familles pour leur support tout au long du projet.

Reconstruction tridimensionnelle automatisée d'un environnement fermé

Mathieu Gascon-Lefebvre
GASM07049007

Junior Grégoire
GREJ25078401

Gabriel Kenderessy
KENG25049006

RÉSUMÉ

La **reconstruction tridimensionnelle** des **environnements intérieurs** est un domaine encore assez récent et comportant beaucoup de lacunes. En effet, les solutions actuelles coûtent souvent trop cher, ne sont pas efficaces, sont trop lentes et ne fournissent pas nécessairement un résultat de qualité.

L'objectif de ce projet est donc de pallier à ces problèmes en proposant un système robotique **facilement utilisable, portable, efficace** et à **faible coût**.

Pour bien parvenir à cet objectif, le projet a été divisé en quatre parties: **l'architecture logicielle, le fonctionnement du robot, l'intelligence artificielle et la reconstruction 3D**.

Le système développé utilise le framework de robotique **ROS**, proposant une multitude d'outils, de librairies et de conventions permettant de résoudre simplement des problèmes courants dans ce domaine. Pour ce qui est de la reconstruction tridimensionnelle, «Point Cloud Library» (**PCL**) est utilisée.

La solution proposée utilise le robot du club **Capra**, mais ne dépend pas uniquement de celui-ci et peut être utilisée sur n'importe quel robot terrestre.

MOTS-CLÉS: Robotique, Reconstruction 3D, ROS, PCL, Intelligence artificielle, Cartographie, Navigation autonome, Faible coût, Club Capra

TABLE DES MATIÈRES

Remerciements	2
Résumé.....	3
Table des matières.....	4
Liste des tableaux.....	6
Liste des figures	7
Liste des abréviations, sigles et acronymes.....	9
Introduction	11
Mise en contexte.....	11
Objectifs	11
Méthodologie.....	13
Architecture logicielle	13
Besoins.....	13
Architecture ROS	13
Architecture matérielle	15
Ordinateur portable Lenovo ThinkPad W540	15
Moteurs Animatics SM23165DT.....	16
IMU Microstrain 3DM-GX1.....	16
Range Finder (LiDAR) SICK LMS100	17
BeagleBone Black.....	17
Caméra ASUS Xtion PRO LIVE	20
Lynxmotion Pan-Tilt Unit.....	20
Fonctionnement du robot.....	22
Transformations	22
Localisation et cartographie 2D.....	24
Navigation.....	30
Intelligence artificielle	32
Besoin pour une intelligence artificielle	32
Génération des objectifs à partir d'une carte	34
Comparaison des algorithmes du plus court chemin.....	39
Chemin optimal entre les objectifs de numérisation	41
Processus de numérisation.....	44
Reconstruction 3D	45
Comparaison entre les algorithmes de reconstruction disponibles	45
Description de l'algorithme de reconstruction 3D choisi.....	53
Résultats et discussion.....	58
Fonctionnement du robot.....	58
Localisation et cartographie 2D.....	58
Navigation.....	66
Architecture logicielle	68
Déplacement autonome dans un environnement connu	68
Génération de la carte statique de l'environnement.....	70

Modularité	71
Intelligence artificielle	72
Génération d'objectifs.....	73
Recherche du plus court chemin	83
Cas particuliers	87
Numérisation 3D	89
Conclusion.....	91
Considérations.....	91
Bibliographie	93
Architecture.....	93
Localisation et cartographie.....	93
Canonical Scan Matcher	93
Filtre de Kalman étendu	93
GMapping	93
Hector mapping	94
Localisation adaptative de Monte-Carlo	94
Navigation	94
Intelligence artificielle	94
Reconstruction 3D	95
Références	96
Document de vision.....	96
Document d'architecture	96
Plan de tests	96
Manuel utilisateur	96
Vidéos de démonstration.....	96
Reconstruction 3D	96
Génération de carte 2D	96
Navigation avec évitement d'obstacle	96
Intelligence artificielle en action	96
Code source.....	96

LISTE DES TABLEAUX

Tableau 1: Liste des abréviations, sigles et acronymes	9
Tableau 2: Spécifications du portable.....	15
Tableau 3: Spécifications du BeagleBone Black.....	17
Tableau 4: Légende du schéma électrique du BeagleBone Black	19
Tableau 5: Spécifications des servomoteurs	21
Tableau 6: Cadres référentiels utilisés dans le projet	22
Tableau 7: Comparaison des différents algorithmes de pathfinding retenus.....	40
Tableau 8: Les paramètres du test de référence de la génération d'objectifs.....	74
Tableau 9: Les paramètres du test pour une petite carte	75
Tableau 10: Les paramètres du test pour une grosse carte	76
Tableau 11: Les paramètres du test pour un petit filtre médian	77
Tableau 12: Les paramètres du test pour un gros filtre médian	78
Tableau 13: Les paramètres du test pour une petite région de partitionnement	79
Tableau 14: Les paramètres du test pour une grande région de partitionnement	80
Tableau 15: Les paramètres du test pour une petite distance par rapport aux murs	81
Tableau 16: Les paramètres du test pour une grande distance par rapport aux murs.....	82
Tableau 17: Métriques associées à l'algorithme Minimum Spanning Tree	83
Tableau 18: Métriques associées à l'algorithme de Prim	84
Tableau 19: Métriques associées à l'algorithme Double Greedy Shortest Path	85
Tableau 20: Métriques associées à l'algorithme Double Greedy Shortest Path par force brute.	86

LISTE DES FIGURES

Figure 1: BeagleBone Black assemblé.....	18
Figure 2: Schéma électrique du contrôle du PTU sur le BBB	19
Figure 3: Différents angles possibles pour le PTU	20
Figure 4: Représentation des cadres référentiels sur le robot physique	23
Figure 5: Représentation visuelle des cadres référentiels	23
Figure 6: Représentation des transformations statiques entre les différents cadres référentiels	23
Figure 7: Un trajet calculé par le planificateur global.....	30
Figure 8: Un obstacle contourné par le planificateur local	31
Figure 9: La carte de coûts locale	31
Figure 10: La carte de coûts globale.	31
Figure 11: La carte de données brutes	34
Figure 12: La carte recadrée	35
Figure 13: La matrice binaire	36
Figure 14: La transformée de distances euclidiennes	36
Figure 15: La transformée de distances filtrées	37
Figure 16: Partitionnement de la carte des distances	38
Figure 17: Carte avec objectifs générés.....	38
Figure 18: Plus court chemin trouvé avec DGSP à partir d'un certain point.....	41
Figure 19: Plus court chemin incluant les angles de numérisations.....	42
Figure 20: Démonstration de l'algorithme «Iterative Closest Point»	45
Figure 21: Démonstration du filtre MLS	50
Figure 22: Démonstration du filtre euclidien	51
Figure 23: Comparaison des trajets calculés	58
Figure 24:	60
Figure 25: Solution 2 : Moteurs + GMapping.....	60
Figure 26: Solution 3 : Hector SLAM	60
Figure 27: Trajet parcouru pour les tests	61
Figure 28: Solution 1	61
Figure 29: Solution 2	61
Figure 30: Solution 3	61
Figure 31: Solution 1	62
Figure 32: Solution 2	62
Figure 33: Solution 3	62
Figure 34: Résultat de la solution 1	63
Figure 35: Résultats de la solution 2.....	64
Figure 36: Résultats de la solution 3.....	64
Figure 37: Résultats de la solution 4.....	65
Figure 38: Envoi d'un objectif au robot en utilisant l'outil rviz	66
Figure 39: Évitement d'un obstacle dynamique: avant.....	67
Figure 40: Évitement d'un obstacle dynamique: après	67

Figure 41: Architecture du déplacement autonome dans un environnement connu	68
Figure 42: Architecture de génération de la carte statique de l'environnement	70
Figure 43: La carte du test de référence de la génération d'objectifs	73
Figure 44: La carte du test pour une petite carte	75
Figure 45: La carte du test pour une grosse carte	76
Figure 46: La carte du test pour un petit filtre médian	77
Figure 47: La carte du test pour un gros filtre médian	78
Figure 48: La carte du test pour une petite région de partitionnement	79
Figure 49: La carte du test pour une grande région de partitionnement	80
Figure 50: La carte du test pour une petite distance par rapport aux murs	81
Figure 51: La carte du test pour une grande distance par rapport aux murs	82
Figure 52: Carte représentant l'algorithme Minimum Spanning Tree	83
Figure 53: Carte représentant l'algorithme de Prim	84
Figure 54: Carte représentant l'algorithme Double Greedy Shortest Path	85
Figure 55: Carte représentant l'algorithme Double Greedy Shortest Path par force brute	86
Figure 56: Cas de l'îlot	87
Figure 57: Cas des numérisations multiples	88
Figure 58: Reconstruction finale	90

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

Tableau 1: Liste des abréviations, sigles et acronymes

AIMBot	(Automatic Indoor Mapping Robot) Le projet discuté dans ce document.
AMCL	(Adaptive Monte Carlo Localisation) Localisation adaptative de Monte-Carlo. Algorithme permettant d'estimer la position d'un robot dans une carte connue.
BBB	(BeagleBone Black) Micro-ordinateur utilisé pour le contrôle du <i>Pan-Tilt Unit</i> .
CSM	(Canonical Scan Matcher) Algorithme basé sur ICP permettant de trouver les correspondances entre deux numérisations 2D et ainsi de trouver un déplacement effectué.
DGSP	(Double Greedy Shortest Path) Algorithme de pathfinding utilisé dans le module d'intelligence artificielle du robot.
EKF	(Extendend Kalman Filter) Filtre de Kalman étendu permettant, en utilisant plusieurs données en entrée, d'approximer une position du robot.
GPIO	(General Purpose Input/Output) Ports multiusage sur les micro-ordinateurs et microcontrôleurs acceptant différents type d'entrées et de sorties.
IA	Intelligence artificielle.
ICP	(Iterative Closest Point) Algorithme permettant de fusionner deux nuages de points représentant le même objet.
IMU	(Inertial Measurement Unit) Centrale inertie. Instrument de navigation utilisant plusieurs capteurs et permettant de déterminer une orientation.
KNW	(K-Nearest Walls) Algorithme permettant de trouver les murs les plus proches d'un point ainsi que leurs angles par rapport à ce dernier.
LiDAR	(Light Detection And Ranging) Instrument de mesure optique permettant de connaître l'angle et la distance des obstacles.
LTS	(Long-term support) Au niveau logiciel, désigne un logiciel offrant un support officiel plus long que la durée habituelle.
MPGG	(Matrix Partitioning and Goal Generation) Algorithme permettant de

	générer une série non ordonnée d'objectifs à atteindre par le robot à partir d'une carte brute de l'environnement préalablement traitée et filtrée.
PCB	(Printed Circuit Board) Plaque reliant et assurant le bon fonctionnement des composantes électriques contenues dans le robot.
PCL	(Point Cloud Library) Librairie logicielle qui permet de manipuler des nuages de points en 3D.
PTU	(Pan-Tilt Unit) Périmérique composé de deux servomoteurs permettant d'effectuer une rotation à la verticale et à l'horizontale.
PWM	(Pulse-width modulation) Technique permettant de synthétiser des signaux électriques continus.
ROS	(Robot Operating System) Collection de frameworks logiciels aidant au développement de robots.
SLAM	(Simultaneous localization and mapping) Cartographie et localisation simultanées. Technique utilisée permettant de construire une carte d'un environnement tout en s'y localisant.
TOF	Type de caméra qui génère des images de profondeur en mesurant le temps de vol entre la caméra et les obstacles.

INTRODUCTION

Mise en contexte

La reconstruction 3D d'une pièce est habituellement une tâche complexe, dispendieuse et qui demande la mobilisation de beaucoup de ressources (humaines et matérielles).

Les entreprises œuvrant dans ce domaine proposent des solutions coûteuses demandant un environnement contrôlé et un expert présent sur place. De plus, ces solutions ne font pas une modélisation complète de l'environnement, mais bien une simulation d'environnement 3D par le biais de panoramas de photos.

Avec la collaboration du Club Capra et de leur véhicule autonome, le projet de reconstruction 3D automatisée, ou **AIMBot** (**A**utomatic **I**ndoor **Mapping **R**o**B**ot) tente de pallier aux problèmes existants dans ce domaine d'affaires. Grâce à une caméra permettant de capturer des images et la profondeur de celles-ci montée sur le robot fourni par le club, une reconstruction 3D de haute qualité peut être faite dans un délai raisonnable et à faible coût.**

Objectifs

Le projet AIMBot a pour but de remédier à certains problèmes présents dans le domaine de la reconstruction 3D intérieure.

L'objectif principal est d'avoir une reconstruction 3D haut de gamme. Pour atteindre un résultat acceptable selon des normes prédéfinies par l'équipe, une attention particulière a été apportée à la qualité du processus de numérisation 3D.

Par contre, une reconstruction de qualité supérieure entraîne des problèmes d'efficacité. Effectivement, si certaines numérisations n'atteignent pas le niveau de qualité désiré, il faut renumériser en utilisant une stratégie différente, engendrant ainsi des pertes de performance.

L'efficacité des reconstructions 3D est sans aucun doute un enjeu important, mais moins d'importance y est accordée dans l'optique où la qualité de ces dernières est grandement améliorée.

En utilisant une partie d'architecture logicielle et le véhicule autonome du Club Capra, une partie importante des coûts est déjà couverte. Le choix de la caméra et des autres périphériques nécessaires à la reconstruction 3D ont déterminé le coût du projet. Une limite de 200\$ a été fixée.

La partie haut niveau de l'architecture logicielle développée est indépendante du robot sur laquelle elle sera déployée. De cette manière, les capteurs utilisés peuvent facilement être changés à court ou à long terme pour faire place à des capteurs de meilleure qualité ou des capteurs ayant plus de fonctionnalités. De plus, la solution logicielle, basée sur une architecture ROS (Robot Operating System), est modulaire, donc, indépendante du reste de l'architecture dans laquelle elle sera intégrée. Ainsi, le système de numérisation 3D ne dépend pas de l'implémentation des autres périphériques du robot.

Toutefois, il a été réalisé que les techniques de localisation et de cartographie 2D utilisées par le robot dans le cadre de la compétition IGVC, se déroulant à l'extérieur, ne s'appliquent pas à une utilisation intérieure. Il a donc fallu déterminer les meilleures méthodes permettant de s'adapter à cet environnement.

MÉTHODOLOGIE

Architecture logicielle

Besoins

Le but premier de l'architecture logicielle développée était qu'elle soit très modulaire et que chaque composant puisse être remplacé facilement sans affecter le reste du système.

Cela permettrait donc de ne pas dépendre du robot de Capra présentement utilisé et de porter facilement la solution développée sur n'importe quel robot. La modularité permettrait aussi aux différents membres de l'équipe de développer leur partie sans nuire aux autres.

Un autre besoin était de ne pas avoir à réimplémenter des choses qui ont déjà été faites des centaines de fois dans le passé et de seulement s'attarder sur les fonctionnalités propres au projet.

Architecture ROS

Pour répondre à ces besoins, il a été déterminé d'utiliser le système ROS.

ROS est un ensemble d'outils, de librairies et de conventions pour la robotique développé par la communauté, ainsi que par un grand nombre d'entreprises oeuvrant dans ce domaine.

Plusieurs modules sont fournis permettant de fournir une base solide aux développeurs de robots, tels que des pilotes pour plusieurs périphériques, des outils de cartographie, de localisation et de navigation, des librairies de communication et une quantité énorme d'autres outils.

Pour effectuer la communication entre les différents noeuds, une architecture de type publication-abonnement est utilisée. Dans ce type d'architecture, tous les composants sont raccordés à un distributeur d'événements qui se charge d'envoyer les événements aux différents abonnés.

Le tableau suivant définit certains concepts de ROS dans le but de fournir une explication plus claire par la suite.

Noeud	Processus ayant une tâche précise et pouvant s'inscrire à certains canaux de communication et offrir des services.
Topic	Canal de communication utilisé pour le partage d'événements.
Message	Ensemble de valeurs identifiées par un nom et organisées ayant un lien entre elles. Par exemple, un message d'odométrie contiendrait des valeurs indiquant la position du robot, son orientation ainsi que sa vitesse.
Service	Les services offrent un mécanisme de requête/réponse permettant d'envoyer une requête directement à un noeud précis de manière plus efficace. On pourrait les comparer à des services web.
Maître	Gestionnaire en charge d'indiquer aux différents noeuds où se trouvent les topics et services désirés.

Ainsi, les différents noeuds peuvent envoyer des messages sur les canaux de communication sans se soucier de quels noeuds les recevront et de ce qu'ils feront avec. La seule restriction est d'utiliser le type de message spécifié pour utiliser le canal.

Toutefois, les topics ne sont pas toujours applicables à certains contextes et sont plus utilisés pour la réception d'événements que pour effectuer une action. Pour cette raison, les services ont été introduits. Par exemple, le noeud «PTU» pourrait exposer les services «Pan» et «Tilt» permettant d'effectuer une rotation horizontale ou verticale. Par la suite, il pourrait envoyer un message indiquant son statut sur un topic.

Un autre point fort de cette architecture est qu'elle peut être utilisée par plusieurs ordinateurs de manière distribuée. Avoir un poste de téléopération affichant un flux vidéo et permettant de contrôler le robot avec une manette de demanderait aucun effort supplémentaire.

Architecture matérielle

Le robot développé par le Club Capra comporte plusieurs capteurs et composantes critiques au bon fonctionnement du projet AIMBot. Certaines composantes matérielles ont été ajoutées au robot alors que d'autres n'ont simplement pas été utilisées dû à certaines différences majeures entre l'objectif du projet AIMBot et le but du club étudiant.

Cette section omet les capteurs et les composantes inutilisés par le projet. De plus, le fonctionnement interne du robot (PCB, schémas électriques, etc.), à l'exception du micro-ordinateur contrôlant la caméra, ne sera pas détaillé, faute de manque d'expertise dans le domaine du génie électrique.

Ordinateur portable Lenovo ThinkPad W540



Étant donné la grande quantité d'informations provenant des différents capteurs et la lourdeur des différents algorithmes utilisés, il était essentiel d'avoir un ordinateur assez puissant pour avoir un bon temps de réaction.

Tableau 2: Spécifications du portable

Processeur	Core i7 4700MQ / 2.4 GHz
Mémoire vive	16 GB
Carte graphique	NVIDIA Quadro K1100M
Disque dur	500 GB à 7200 trs/mn
Batterie	57++ (9 Cell) - Lithium Ion (Li-Ion) - 100 V DC

Avec de telles spécifications, un manque de puissance nuisant à l'exécution du système a pu être évité et l'équipe a pu se concentrer sur les meilleurs choix d'algorithmes plutôt que sur des optimisations inutiles. La batterie permet aussi d'effectuer des tests pendant quelques heures sans devoir la recharger.

Le système d'exploitation choisi sur le portable est Ubuntu 14.04 LTS, un choix essentiel pour permettre l'utilisation de ROS Indigo Igloo qui, comme le système d'exploitation, est une version LTS.

Moteurs Animatics SM23165DT



Le robot possède deux moteurs de ce modèle. Il s'agit de moteurs «intelligents» connectés par câble série auxquels il suffit d'indiquer l'accélération et la vitesse désirée. Ils sont donc beaucoup plus simples à utiliser que des moteurs traditionnels.

Ils sont aussi munis d'encodeurs très précis de 4000 comptes par révolution permettant de connaître le nombre de tours effectués.

Les roues montées dessus mesurent 30cm de diamètre.

IMU Microstrain 3DM-GX1



Un IMU est un capteur permettant de connaître plusieurs informations inertielles sur le robot, telles que son orientation, sa vitesse angulaire et son accélération linéaire.

Dans les bonnes conditions, l'angle est retourné avec une précision supérieure à 0.5°.

Il date toutefois de 2006 et commence à être désuet (le domaine de la robotique évolue très rapidement).

Range Finder (LiDAR) SICK LMS100



Ce capteur permet de détecter les différents obstacles dans un rayon de 270° avec une précision de 0.5°. Ainsi, à chaque numérisation effectuée, jusqu'à 540 points peuvent être trouvés. Sa portée est de 18 mètres. Il est normalement utilisé pour permettre au robot d'éviter les obstacles sur son chemin.

BeagleBone Black



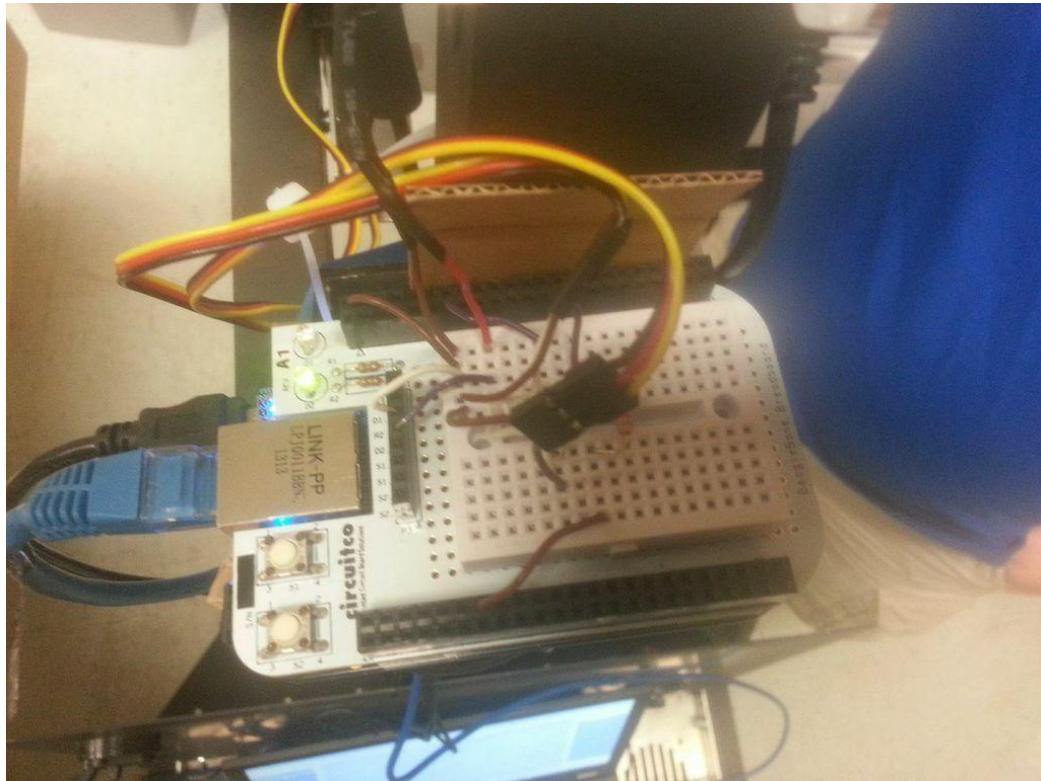
Le *BeagleBone Black*, ou BBB, est un micro-ordinateur permettant de contrôler le *Pan-Tilt Unit* grâce à ses sorties PWM. Le BBB se comporte exactement comme un PC normal à haut niveau: il peut reconnaître des périphériques USB et utilise le système d'exploitation Ubuntu 14.04 LTS. Par contre, il offre aussi des fonctionnalités à un plus bas niveau qui peuvent s'apparenter aux microcontrôleurs comme des ports GPIO, PWM et autres.

Tableau 3: Spécifications du BeagleBone Black

Processeur	AM335x 1GHz ARM® Cortex-A8
Mémoire vive	512MB DDR3 RAM
Stockage	Mémoire ROM de 4GB et fente pour carte SD
Graphiques	Accélérateur graphique 3D et sortie micro HDMI
Réseau	Prise Ethernet

Dans le contexte du projet AIMBot, le BBB est uniquement utilisé en conjonction avec ROS et la librairie Adafruit_BBIO. Le BBB est synchronisé pour démarrer au même moment que le portable (s'il est préalablement connecté par USB) et, lors de ses routines de démarrage, un noeud ROS initialise le PTU pour que l'IA soit en mesure de lui envoyer des angles lorsque le robot est en mode autonome. De plus, le noeud ROS se trouvant sur le BBB se connecte automatiquement au processus maître de ROS du portable par réseau offrant ainsi un endroit centralisé pour gérer les communications entre les différents noeuds ROS.

Figure 1: BeagleBone Black assemblé



Le BeagleBone Black est un micro-ordinateur modulaire permettant l'ajout de périphériques physiques communiquant avec les entrées et sorties disponibles sur la carte du BBB. Ces modules, appelés «capes», peuvent ajouter plusieurs nouvelles fonctionnalités comme des capteurs ou des écrans tactiles. La cape *Breadboard* offre au BBB une zone de prototypage sans soudure. Cette zone de prototypage est très conviviale pour les électriciens amateurs et permet une reconfiguration facile des circuits en cas de problème.

Pour communiquer avec le PTU, l'utilisation de la librairie Adafruit_BBIO est requise. Cette librairie agit comme une interface pour communiquer avec les entrées et les sorties du BBB. La seule fonctionnalité utilisée est la façade pour communiquer avec les PWM, mais ses fonctionnalités vont au-delà de ce simple besoin.

Figure 2: Schéma électrique du contrôle du PTU sur le BBB

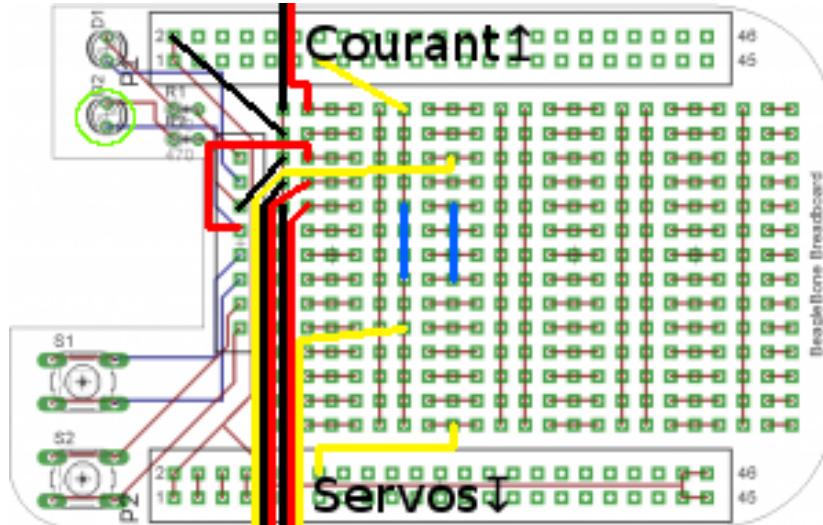


Tableau 4: Légende du schéma électrique du BeagleBone Black

GND (<i>Ground</i>)
Résistance (1kΩ)
Alimentation (5V)
Signal
LED

Puisque les servomoteurs sont alimentés par USB, le BBB doit avoir un de ses ports GND (Ground) connecté à l'alimentation des servomoteurs pour diminuer le bruit perçu par ces derniers lors de l'utilisation des PWM.

Un DEL est aussi connectée à l'alimentation des servomoteurs pour donner un retour visuel aux utilisateurs lorsque le PTU est en fonction.

Caméra ASUS Xtion PRO LIVE



Cette caméra, dans la même famille que la Kinect, permet de récupérer des images de profondeur ainsi que des images RGB permettant de générer des nuages de points.

Elle offre un champ de vision de 58° à l'horizontale, 45° à la verticale et 70° en diagonale. La résolution de l'image de profondeur est de 640x480 et celle de l'image RGB de 1280x1024. De plus, étant moins lourde que son homologue, elle peut être fixée sur un PTU.

Lynxmotion Pan-Tilt Unit



Le *Pan-Tilt Unit*, ou PTU, est une pièce mobile composée de 2 servomoteurs de type HS-422 soutenant la caméra ASUS Xtion PRO LIVE. Ils sont connectés via USB pour leur fournir un voltage de 5V. De plus, ils sont connectés à 2 PWM se trouvant sur le BBB pour obtenir un angle précis.

Figure 3: Différents angles possibles pour le PTU



Le PTU offre un balayage horizontal allant de -90 à 90 degrés et un balayage vertical allant de -30 à 90 degrés. Avec un balayage large, le PTU peut pointer la caméra à différents angles sans avoir à changer la position du robot.

Tableau 5: Spécifications des servomoteurs

Vitesse d'opération	0.21 sec/60°
Torque	3.3 kg·cm
Poids	45.5 g

Fonctionnement du robot

Transformations

Une difficulté en robotique est souvent de transformer certaines données d'un cadre référentiel à un autre afin de les traiter de manière plus intelligente.

Par exemple, le LiDAR est situé de manière statique sur le robot, qui lui se promène de manière dynamique dans son environnement. Pour générer une carte de cet environnement, il est donc important de transformer les obstacles détectés par le radar dans le bon cadre.

Pour effectuer ces transformations, un outil appelé «TF» est offert par ROS.

Il permet d'indiquer les différentes matrices de transformation relatives entre les différents cadres, puis de les multiplier afin de trouver la transformation désirée.

Tableau 6: Cadres référentiels utilisés dans le projet

odom	Position du robot indiquée par l'odométrie. Elle peut être imprécise.
map	Position du robot dans l'environnement. Elle est basée sur odom, mais améliorée grâce au capteur laser. Dans une situation parfaite, il n'y aurait pas de différence entre les deux cadres.
base_footprint	Position du centre du robot au niveau du sol.
base_link	Position du centre du robot.
laser	Position du LiDAR par rapport au centre du robot.
ptu	Position de la base PTU par rapport au centre du robot.
ptu_attachment	Position du dessus du PTU
camera_link	Position du centre de la caméra

Figure 4: Représentation des cadres référentiels sur le robot physique



Figure 5: Représentation visuelle des cadres référentiels

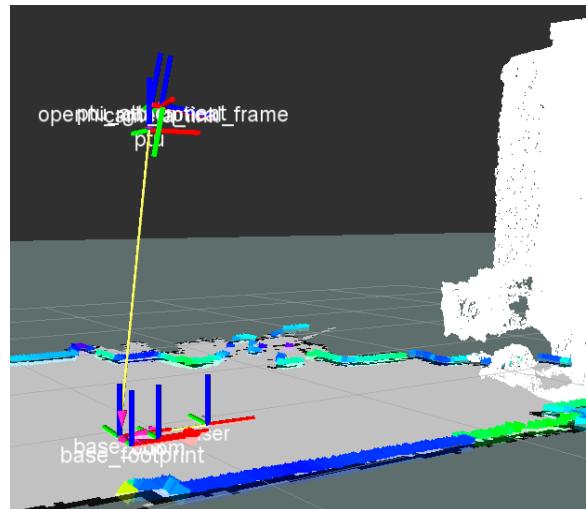
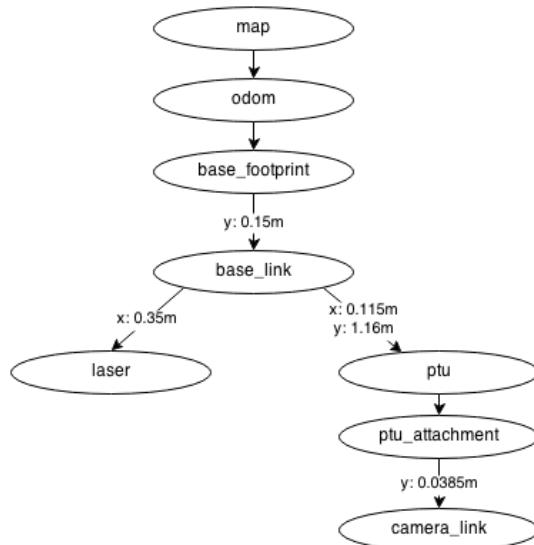


Figure 6: Représentation des transformations statiques entre les différents cadres référentiels



Ainsi, pour pouvoir utiliser le nuage de points produit par la caméra dans le cadre référentiel de la carte, il faudrait utiliser les multiplications de matrices suivantes:

$[camera_link] \bullet [ptu_attachment] \bullet [ptu] \bullet [base_link] \bullet [base_footprint] \bullet [odom] \bullet [map]$

Localisation et cartographie 2D

Un des défis majeurs du projet consistait à déterminer la position du robot de la manière la plus précise possible, l'environnement intérieur de l'ÉTS étant bien différent à celui où se déroule la compétition à laquelle participe Capra.

Afin de simplifier les choses, il a été convenu de d'abord générer une carte statique la plus précise possible, puis d'ensuite l'utiliser lors de l'étape de cartographie 3D.

Capteurs permettant de s'orienter

Pour s'orienter, le robot dispose normalement de plusieurs capteurs:

Moteurs Animatics SM23165DT

Ils retournent la position et l'angle actuel du robot par rapport à sa position initiale. À chaque fois qu'un déplacement calculé par les encodeurs de chaque moteur est reçu, un algorithme pour robot différentiel calcule la position que le robot devrait avoir.

Toutefois, à cause de certains problèmes mécaniques du robot actuel, les roues sont légèrement désaxées, ce qui cause une certaine imprécision dans le résultat. La distance parcourue est toutefois assez fiable.

IMU Microstrain 3DM-GX1

Ce capteur retourne plusieurs informations inertielles sur le robot, telles que son orientation, sa vitesse angulaire et son accélération linéaire.

Dans les bonnes conditions, l'angle est retourné avec une précision supérieure à 0.5°. Toutefois, il semblerait qu'il y ait beaucoup d'interférence dans les murs de l'ÉTS qui font fluctuer énormément l'angle retourné et le rendent inutilisable.

GPS Novatel Propak v3

Ce capteur retourne une position par rapport à la terre et offre normalement une précision située entre 15 et 30 cm.

Toutefois, il est conçu pour être utilisé à l'extérieur dans un environnement dégagé. Il n'est donc pas utilisable dans le cadre de ce projet.

Dans un environnement extérieur, il est normalement possible d'envoyer les données de ces trois capteurs à un filtre de Kalman étendu et d'avoir une bonne idée de la vraie position du robot. Toutefois, il n'a pas été possible d'utiliser la même technique à l'intérieur pour les raisons listées ci-haut.

Une solution, qui peut sembler un peu contre-intuitive, serait d'utiliser les données reçues par le LiDAR afin de calculer un estimé de la position. Un tel estimé peut être produit de plusieurs manières, comme par des algorithmes de cartographie et localisation simultanées (SLAM) ou par un algorithme tel que «Canonical Scan Matcher».

Algorithmes de localisation et de génération de carte

Plusieurs algorithmes peuvent être utilisés individuellement ou groupés afin de déterminer la position du robot et de générer une carte.

Filtre de Kalman étendu (EKF)

Type: Localisation

Cet algorithme est utilisé pour estimer la position en trois dimensions d'un robot, basée sur les positions venant de différentes sources, telles que l'odométrie des roues, l'orientation du IMU et, dans certains cas, de l'odométrie visuelle. Il peut aussi être utilisé en deux dimensions seulement, comme dans le cas du robot de Capra qui ne peut pas voler.

Canonical Scan Matcher (CSM)

Type: Localisation

Cet algorithme est une implémentation d'une variation très rapide de l'algorithme ICP, normalement utilisé pour les nuages de points.

Étant donné que les données renvoyées par le LiDAR sont en deux dimensions seulement, il est beaucoup plus rapide de trouver les correspondances entre deux scans qu'en 3D.

Il faut toutefois noter que cet algorithme peut mal fonctionner lors d'une grande variation d'angle entre deux numérisations.

GMapping

Type: Localisation et cartographie

GMapping est un algorithme de SLAM utilisant un filtre de particules Rao-Blackwellized très efficace. Il permet de produire une carte en deux dimensions très précise, tout en produisant un bon estimé de la position du robot. En entrée, il faut avoir une position approximative du robot, ainsi que des scans laser. Dans l'algorithme, chaque particule représente une trajectoire

possible du robot, contenant chacune sa propre carte. Ces particules survivent avec une probabilité proportionnelle à la vraisemblance des observations relativement à leur propre carte. Ainsi, après un certain temps, seulement les particules les plus réalistes sont gardées et les autres sont rejetées, fournissant une carte se rapprochant de la réalité.

[Hector mapping](#)

Type: Localisation et cartographie

Il s'agit d'un algorithme de SLAM développé par une équipe de l'Université Darmstadt en Allemagne. Il ne nécessite pas d'odométrie pour fonctionner et peut utiliser uniquement les scans du LiDAR. Toutefois, un décalage peut survenir à la longue et la fermeture de la boucle n'est pas garantie.

[Localisation adaptative de Monte-Carlo \(AMCL\)](#)

Type: Localisation

Cet algorithme utilise un filtre de particules afin de suivre la position d'un robot dans une carte connue. Chaque particule représente une position possible et, à chaque fois que le robot détecte quelque chose, elles sont rééchantillonnées en se basant sur l'estimation bayésienne récursive.

Il est possible de l'initialiser avec une position approximative afin d'améliorer ses résultats. Il faut aussi lui fournir des informations d'odométrie.

Combinaison d'algorithmes pour la génération de carte

Ces techniques peuvent être combinées avec différents paramètres dans le but de générer une carte en deux dimensions. Trois solutions ont été retenues:

Solution 1

CSM: Pour simuler un IMU selon l'odométrie visuelle

EKF: Avec les moteurs et le IMU simulé

GMapping en utilisant la position retournée par le filtre de Kalman

Avantage: Fonctionne bien dans les longs corridors où il n'y a pas de variation au niveau du scan

Solution 2

GMapping seulement en utilisant la position des moteurs en entrée

Avantages:

- Fonctionne bien dans les longs corridors où il n'y a pas de variation au niveau du scan.
- Plus simple à configurer.

Solution 3

Hector mapping seulement

Avantages:

- Ne nécessite pas d'informations d'odométrie pour bien fonctionner
- Encore plus simple à configurer

Combinaison d'algorithmes pour la localisation dans la carte

Quatre solutions sont aussi possibles pour la localisation dans la carte connue par la suite:

Solution 1

AMCL en utilisant la position des moteurs en entrée

Solution 2

CSM: Pour calculer le déplacement approximatif

AMCL en utilisant la position fournie

Avantage: Ne nécessite pas d'informations d'odométrie pour fonctionner.

Solution 3

CSM: Pour calculer le déplacement approximatif, avec la position des moteurs en entrée.

AMCL en utilisant la position fournie

Avantages:

- Fonctionne mieux que la solution 2 lorsqu'il n'y a pas beaucoup de détails.
- Plus simple à configurer que la solution 1

Solution 4

CSM: Pour simuler un IMU selon l'odométrie visuelle

EKF: Avec les moteurs et le IMU simulé

AMCL en utilisant la position fournie

Avantage: Fonctionne mieux dans les longs corridors où il n'y a pas de variation au niveau du scan

Navigation

Pour ce qui est de la navigation, il a été convenu d'utiliser le paquet «move_base» venant avec ROS. Son fonctionnement est divisé en deux parties principales:

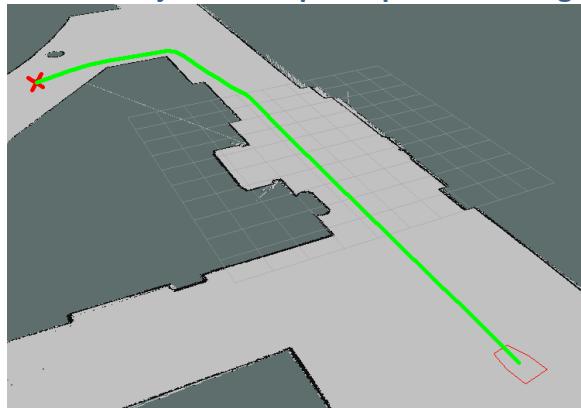
- La planification globale
- La planification locale

En l'utilisant, il suffit d'envoyer des objectifs de navigation (position en x et y, ainsi qu'un angle de destination) et le robot tentera de s'y rendre de manière optimale, tout en évitant les obstacles.

Planification globale

Ce module tente de trouver le meilleur trajet entre la position actuelle et l'objectif en utilisant la carte. L'algorithme de pathfinding utilisé est Dijkstra. Un trajet est ensuite envoyé au planificateur local.

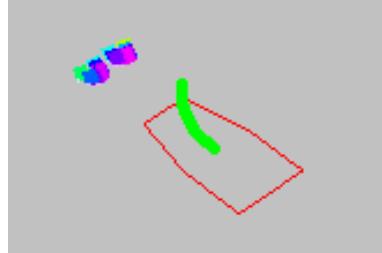
Figure 7: Un trajet calculé par le planificateur global



Planification locale

Ce module vérifie si la direction déterminée par le planificateur global est valide. En effet, dans un environnement dynamique, des obstacles pourraient s'être ajoutés ou se déplacer et il est important de ne pas entrer en collision avec ceux-ci. Le LiDAR est donc utilisé dans cette étape. Une vitesse (angulaire et linéaire) est ensuite envoyée aux moteurs afin d'effectuer le déplacement.

Figure 8: Un obstacle contourné par le planificateur local



Dans les deux cas, une carte de coût est utilisée afin de gonfler les obstacles et d'éviter des collisions indésirables. Ainsi, la trajectoire calculée sera plus sûre.

Figure 9: La carte de coûts locale

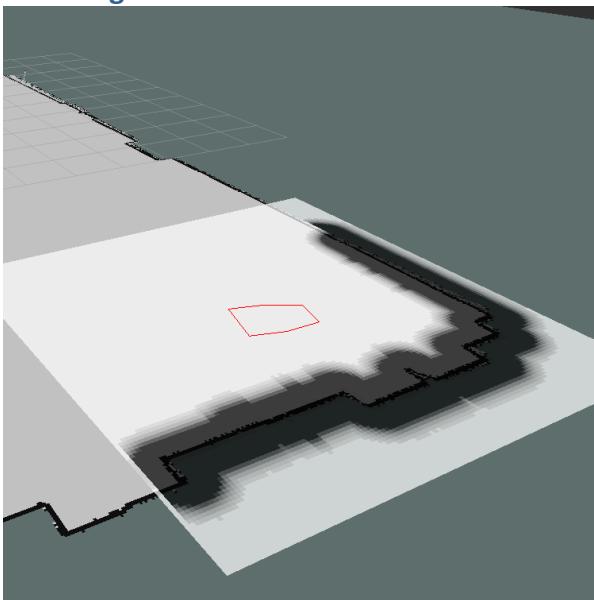
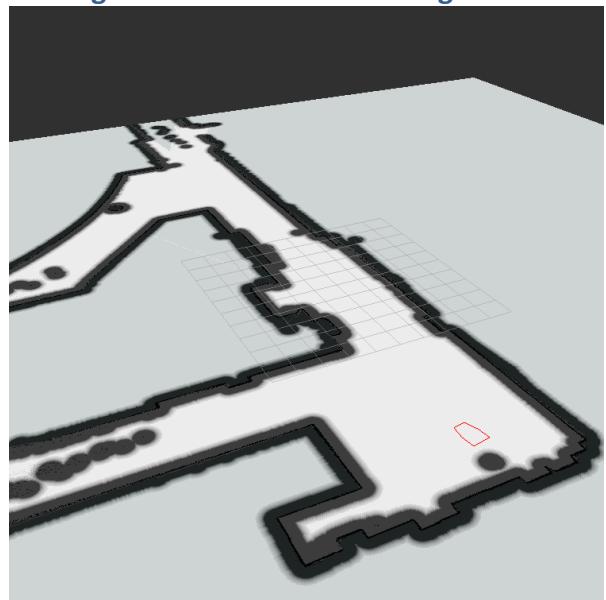


Figure 10: La carte de coûts globale.



Intelligence artificielle

Le module d'intelligence artificielle assure une numérisation complète de la carte fournie au robot dans un temps optimal. Plusieurs algorithmes ont été implémentés pour tenter de répondre à la problématique. Cette section décrit les stratégies implémentées pour optimiser les numérisations faites et les déplacements effectués par le robot. De plus, une comparaison des algorithmes et une description des problèmes reliés à l'intelligence artificielle sont incluses.

Besoin pour une intelligence artificielle

Pour être en mesure d'effectuer une numérisation complète de qualité, une intelligence artificielle fiable est nécessaire pour orienter le robot. Le processus de numérisation intérieure suit quelques règles simples:

- Déetecter tous les murs dans un environnement
- Placer un nombre optimal d'objectifs de manière à couvrir tous les murs détectés
- Se déplacer d'objectif en objectif de manière optimale
- Numériser les murs voulus en s'assurant que chaque numérisation a une série de points semblables pour aider la fusion des nuages de point

Malheureusement, une IA simple suivant les murs n'est pas une solution couvrant tous les cas d'utilisation. Une IA optimale devrait être en mesure de:

- Déetecter les «îlots» de murs qui ne sont pas autrement accessibles qu'en traversant un corridor
- Numériser plusieurs murs à partir d'une même position si ces derniers peuvent être numérisés
- Se limiter à un sous-ensemble de l'environnement
- Pouvoir être configurée sur mesure pour s'adapter aux différents environnements possibles

Dans le cadre du développement du projet, certains cas d'utilisation n'ont pas pu être implémentés dû à des limitations techniques ou à un manque de temps. Par exemple, l'IA développée ne peut pas:

- Naviguer entre deux murs étroits, dû aux limitations du système de navigation et de la caméra
- Offrir une suite logique de nuages de points, dû à l'architecture de certains environnements (par exemple, les discontinuités dans les murs)
- Naviguer dans un environnement inconnu sans carte générée préalablement

Avec les besoins et les limitations bien définis, une IA de qualité a pu être développée pour répondre à la problématique présentée.

Génération des objectifs à partir d'une carte

Avant de naviguer dans son milieu, le robot doit générer des objectifs à partir d'une carte représentant son environnement en 2D. Cette carte est représentée sous la forme d'une matrice 2D d'entiers de -1 à 100. La valeur -1 représente l'espace non cartographié et l'intervalle de 0 à 100 représente la probabilité qu'un obstacle se trouve à cet endroit. Chaque cellule représente une petite section de l'environnement. La résolution de cette section est prédéfinie et peut prendre n'importe quelle valeur (en m^2). Conséquemment, plus la résolution est petite, plus la carte est précise.

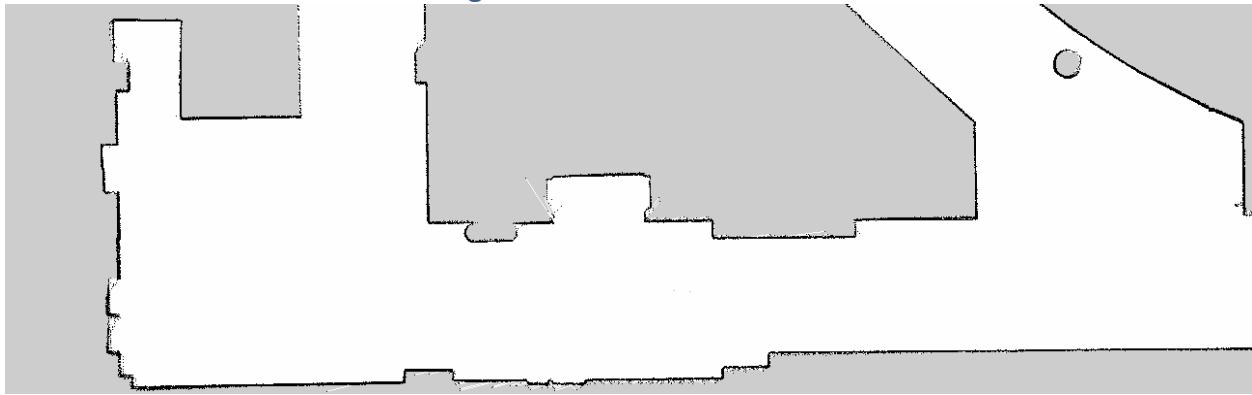
Figure 11: La carte de données brutes



Un premier algorithme a été développé prenant en entrée une matrice de valeurs sous forme d'image et produisant une séquence de points cartésiens appelée «objectifs». Cet algorithme, appelé *Matrix Partitioning and Goal Generation* ou MPGG, effectue plusieurs transformations sur la carte de données brutes pour tenter d'obtenir un nombre raisonnable d'objectifs de manière efficace.

Par souci de performance, la carte brute fournie au module d'intelligence artificielle est recadrée pour seulement prendre en considération une partie de cette dernière. Comme la plupart des variables affectant la performance et la qualité des numérisations, la dimension M x N voulue est paramétrée et complètement configurable par l'utilisateur.

Figure 12: La carte recadrée



La carte recadrée est lissée en utilisant un filtre médian. Un lissage est nécessaire, car la matrice contenant les données brutes peut contenir du bruit et fausser les résultats émis par le module d'intelligence artificielle.

MPGG transforme ensuite la matrice lissée en matrice binaire où 0 représente un obstacle et 1, un espace vide. De cette manière, l'algorithme de génération d'objectifs se simplifie énormément en ne prenant pas compte des espaces non cartographiés et des probabilités de présence d'obstacle.

Figure 13: La matrice binaire



Par la suite, une fois la matrice binaire générée, une transformée de distances y est appliquée. La transformée de distance calcule la distance euclidienne pour chaque élément *vide* par rapport à son élément *obstacle* le plus proche. Seulement les distances entre min_wall_distance et max_wall_distance sont retenues. Une carte de distances est ainsi obtenue et représente les positions acceptables sous forme de trajet où les algorithmes subséquents du module d'intelligence artificielle pourront placer des objectifs de numérisation.

Figure 14: La transformée de distances euclidiennes

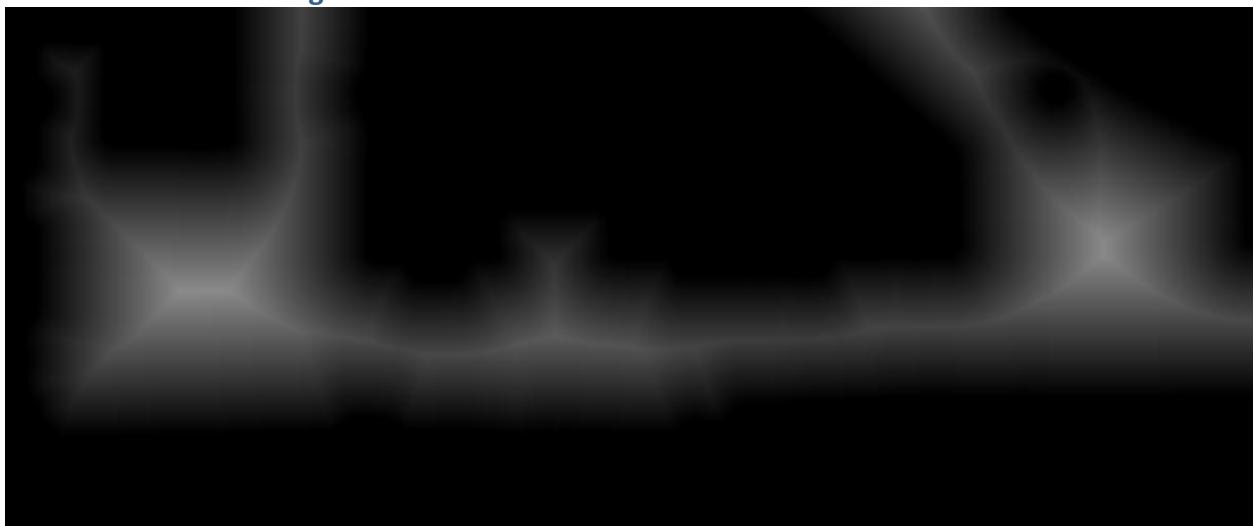
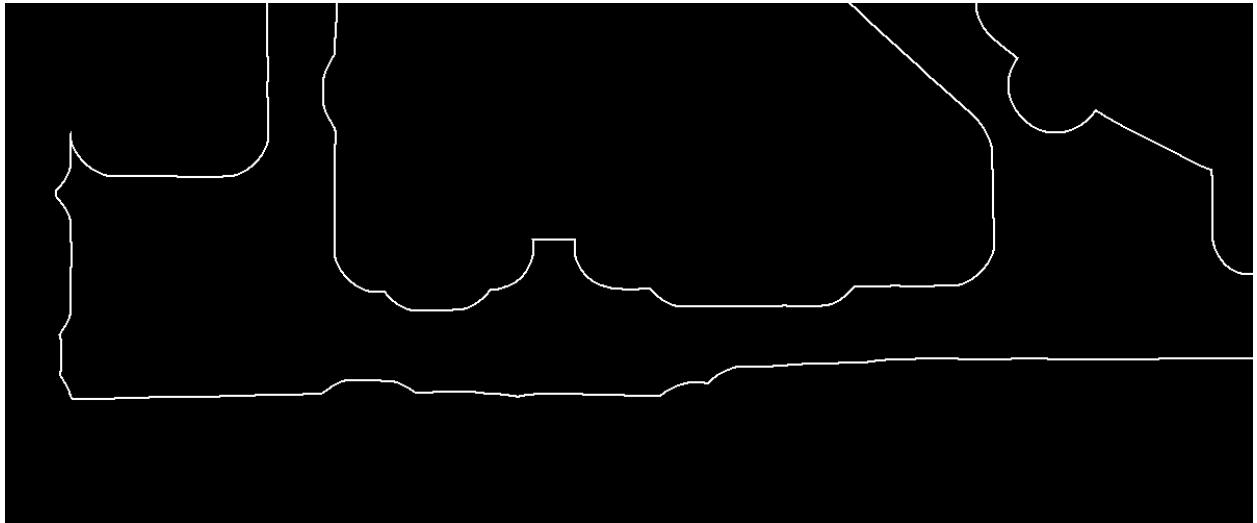


Figure 15: La transformée de distances filtrées



Avec la carte des distances générée, MPGG est en mesure de placer des points optimaux sur la carte pour effectuer la numérisation. Par souci de simplicité, les points sont placés à une distance fixe les uns des autres. Pour obtenir un meilleur recouvrement de la numérisation des murs, il faudrait positionner les objectifs en prenant également compte du champ de vision horizontal et vertical de la caméra (*FOV* ou *field of view*).

La carte de distance est donc séparée en blocs de taille fixe. Chaque bloc est traité séparément et, s'il contient au moins une *cellule vide*, il serait considéré comme étant un bon candidat pour contenir un objectif à condition que cet objectif potentiel se trouve à une certaine distance de l'objectif le plus proche. Le partitionnement de la carte des distances assure donc un placement d'objectifs de manière optimale selon des paramètres prédéfinis.

Figure 16: Partitionnement de la carte des distances

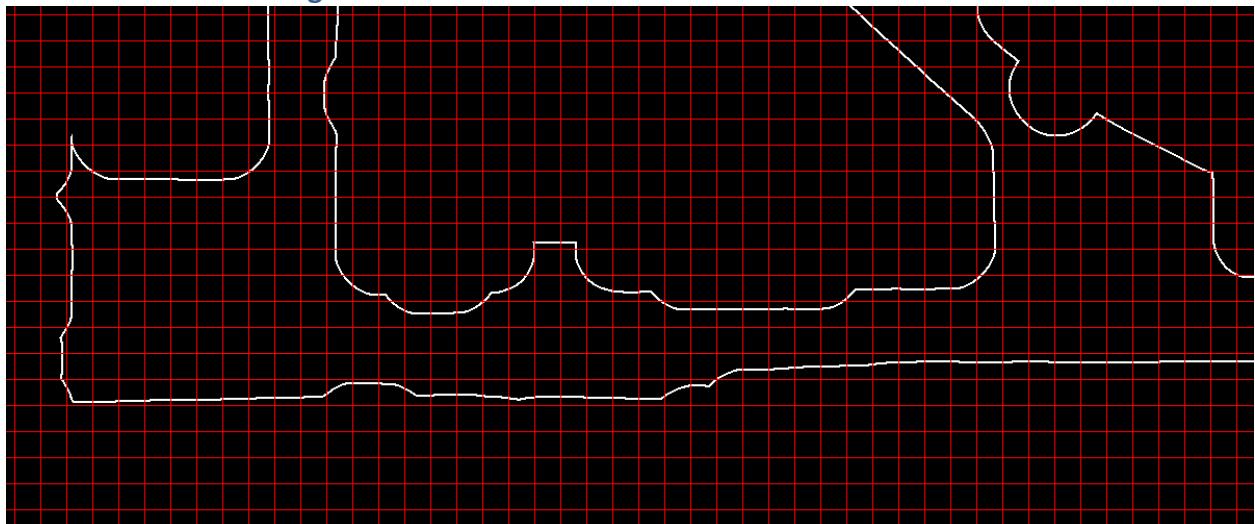
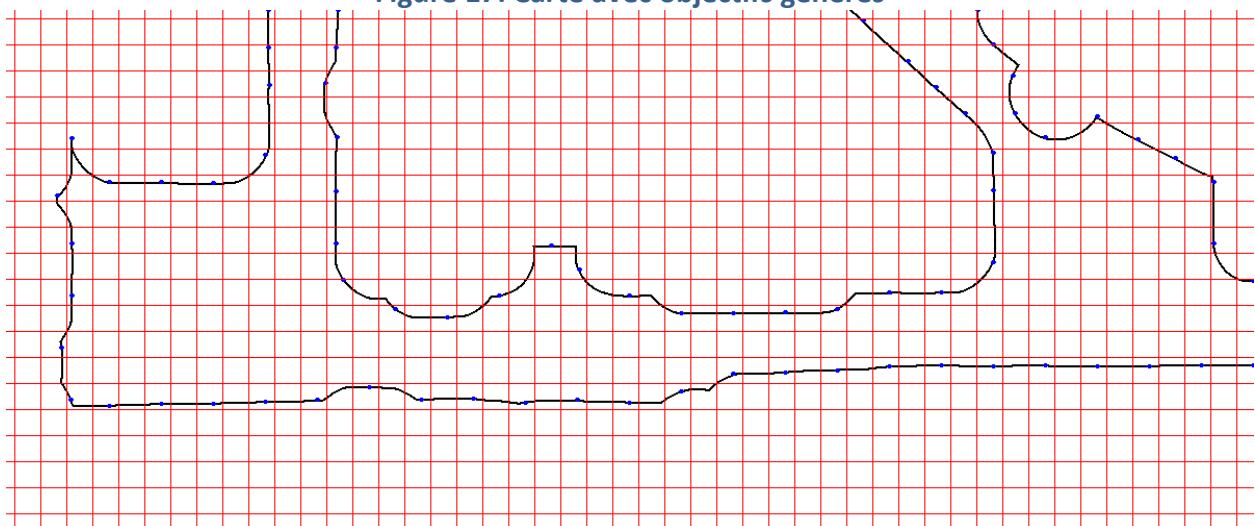


Figure 17: Carte avec objectifs générés



Comparaison des algorithmes du plus court chemin

Les objectifs obtenus avec l'algorithme de partitionnement de la matrice et de génération d'objectifs n'ont aucun ordre précis. C'est ici que les algorithmes de *pathfinding*, le cœur de l'intelligence artificielle, sont pris en compte. Une analyse détaillée a été produite pour trouver les meilleurs candidats pour produire le chemin le plus optimal.

Il faut d'abord se rappeler quelques propriétés des positions produites par l'algorithme de génération d'objectifs:

- Les positions ne sont pas ordonnées
- Le nombre de positions est restreint et dépend de la taille de la carte, de la distance voulue du mur, de la taille des blocs de partitionnement et de la distance minimale entre les objectifs
- Il est possible d'atteindre toutes les positions à partir de n'importe quelle position
- Il faut visiter toutes les positions pour obtenir un recouvrement complet des numérisations
- Pour obtenir le chemin le plus court entre toutes ces positions, il faut visiter chacune de ces dernières seulement une fois

Il s'agit ici d'un exemple classique d'un *problème de voyageur de commerce*. Malheureusement, il n'existe aucun algorithme donnant le résultat optimal en temps polynomial dans tous les cas car l'algorithme est classé comme étant *NP-complet*. En revanche, une solution acceptable peut être obtenue avec des algorithmes de base complémentés par des heuristiques.

Tableau 7: Comparaison des différents algorithmes de pathfinding retenus

Nom de l'algorithme	Description	Particularités
Minimum Spanning Tree	Chemin le plus court entre tous les noeuds d'un graphe, mais peut visiter deux fois le même noeud	Exécution lente et chemin non optimal dans la plupart des cas; Simplicité d'implémentation
Algorithme de Prim	Comme le MST, il trouve l'arbre le plus court, mais en se basant sur les valeurs des liens entre les sommets d'un graphe	Assure que la distance parcourue sera toujours au plus 2 fois la distance optimale
Plus Court Chemin Doublement Glouton	Trouve le plus court chemin entre 2 points et s'assure de visiter un point s'il doit absolument le visiter	Exécution plus lente, mais donne habituellement des résultats constants

Il a été décidé que pour respecter les objectifs du projet, il est crucial de choisir le chemin le plus court. Le temps de traitement pour trouver le plus court chemin se fait avant le début de la première numérisation et est négligeable. L'algorithme retenu est donc celui du Plus Court Chemin Doublement Glouton (*Double Greedy Shortest Path*, ou DGSP) par souci de constance et d'optimisation.

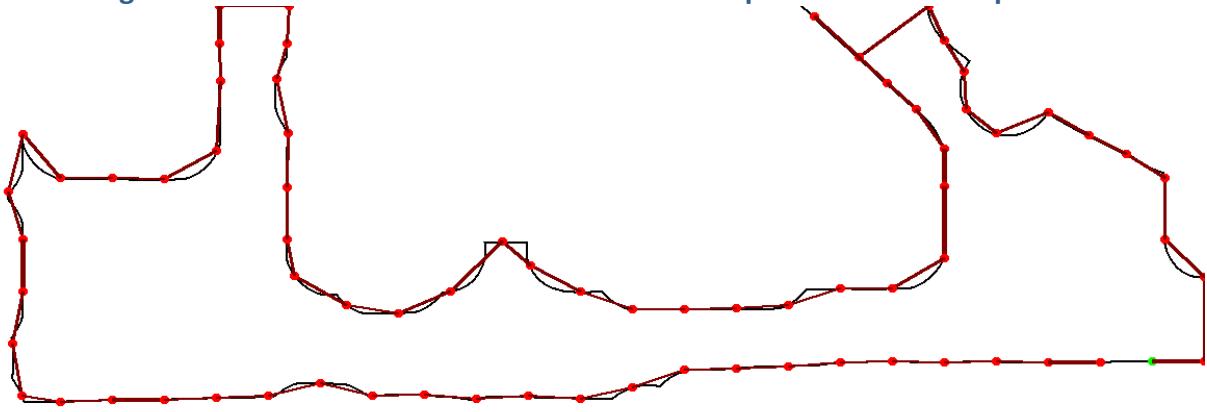
Chemin optimal entre les objectifs de numérisation

L'algorithme DGSP est exécuté pour chaque point trouvé par l'algorithme de génération d'objectifs. De cette manière, l'algorithme est en mesure de trouver le point de départ générant le plus court chemin entre tous les points.

L'algorithme DGSP s'assure de suivre les heuristiques suivantes:

- Un chemin allant de A à B est potentiellement optimal si ce chemin est le plus court par rapport à tous les autres chemins possibles partant de A
- Un chemin allant de A à B doit être emprunté si le chemin de B à A est le plus court par rapport à tous les autres chemins possibles partant de B

Figure 18: Plus court chemin trouvé avec DGSP à partir d'un certain point



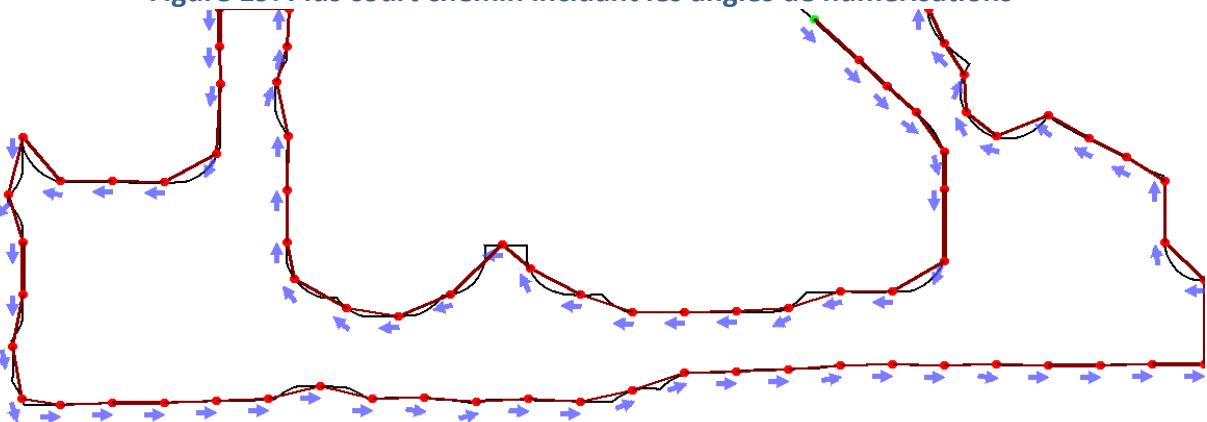
L'algorithme est dit *doublement glouton*, car il suit ces deux règles pour tenter d'atteindre une distance totale minimale à chaque objectif traité.

Angles de numérisation pour chaque objectif

Chaque objectif n'est pas nécessairement associé à une seule région de mur. Effectivement, un objectif peut être associé à plusieurs numérisations qui ne sont pas nécessairement continues. Pour pallier à ce problème, plusieurs angles sont associés à une seule position avec un troisième algorithme: *K-Nearest Walls* (ou K Plus Proches Murs).

Cet algorithme est basé sur l'algorithme KNN (K-Nearest Neighbors) pour trouver le point central des murs les plus proches d'un objectif donné. Pour chaque point P_i se trouvant sur un mur dans un rayon R de l'objectif O , celui ayant la distance minimale de O est retenu. La liste des points est parcourue de manière itérative en enlevant tout les points se trouvant dans le rayon R à chaque point P_i minimum trouvé jusqu'à ce qu'il n'y aie plus de points se trouvant sur les murs à traiter. Chaque point minimum trouvé est associé à un angle de manière à ce que cet angle représente l'angle du segment de droite $\overline{OP_i}$ perpendiculaire au mur mentionné.

Figure 19: Plus court chemin incluant les angles de numérisations



Par la suite, le premier angle trouvé est pivoté à 90 degrés en direction du prochain objectif pour positionner le robot de manière optimale pour qu'il complète son parcours. Pour les angles subséquents, si la différence avec l'angle du robot et l'angle voulu se trouve dans

l'intervalle des angles permis par le module de *Pan* du PTU, le PTU se chargera de faire pivoter la caméra sans que le robot modifie son orientation.

KNW permet donc d'associer un angle de départ à chaque objectif pour initialement bien positionner le robot puis parcourir la liste des angles à cet objectif pour positionner la caméra et numériser la région du mur voulue.

Processus de numérisation

Le processus de numérisation à chaque objectif est configurable pour augmenter le nombre de numérisations et, par conséquent, la qualité globale de la reconstruction. Pour chaque angle demandé à chaque objectif, le robot positionne sa caméra dans l'angle voulu et, par défaut, numérise 3 fois en effectuant un balayage vertical soit à -30 degrés, 0 degré et 30 degrés. De plus, le balayage horizontal peut être inclus, mais, le champ de vision de la caméra et la distance du mur par défaut fait en sorte que les numérisations additionnelles ne sont pas particulièrement utiles.

Le processus de numérisation est disponible sous forme de service ROS. Chaque numérisation est sauvegardé dans un dossier spécifique suivant le format suivant: pc_[date].pcd. Ce dossier contenant les fichiers de données de nuage de points est par la suite lu par le module de reconstruction 3D

Reconstruction 3D

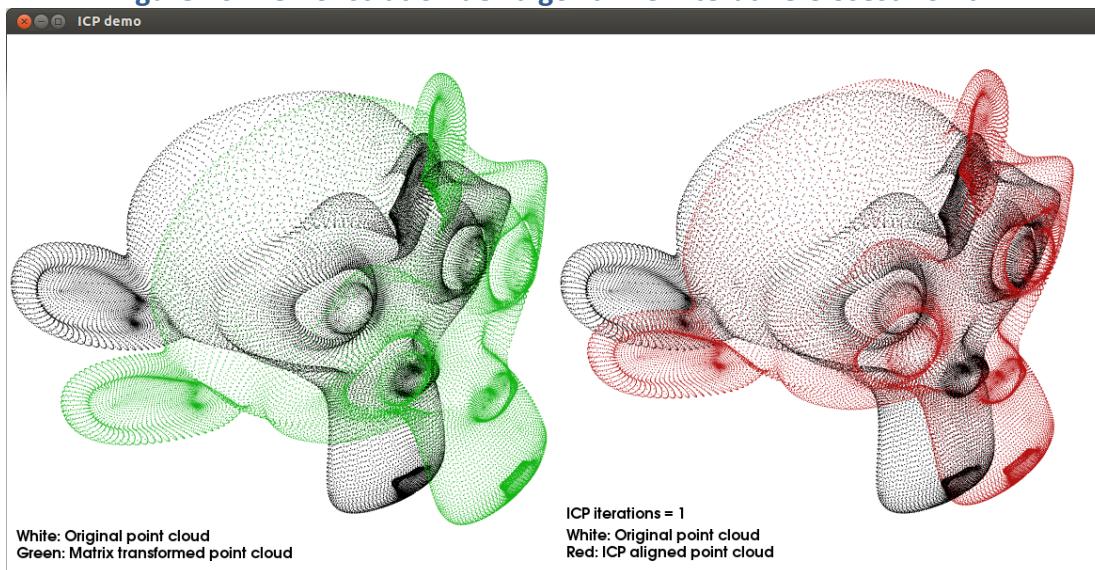
Comparaison entre les algorithmes de reconstruction disponibles

Les reconstructions 3d nécessitent l'utilisation de deux types d'algorithmes pour deux tâches principales. Il faut des algorithmes de lissage pour enlever le bruit des nuages de points et embellir esthétiquement le résultat final. Il faut aussi des méthodes pour juxtaposer de manière logique les nuages de points ensemble. Dans cette section, il sera question de présenter et de comparer ces différents algorithmes de même famille. À noter, ces algorithmes proviennent pour la plupart d'une librairie "open-source" qui gère et traite les nuages de points 3D. Cette librairie se nomme PointCloudLibrary ou pour être plus court PCL.

Algorithmes de juxtaposition des nuages de point

Pour juxtaposer deux nuages de points, il faut d'abord trouver les parties semblables dans les deux nuages. Ensuite il faut trouver une transformation 3D qui permet de coller à la bonne place ces deux nuages pour en former un seul uni et continu. Il existe plusieurs méthodes pour faire cela. Cependant, on peut classer ces techniques en deux grandes familles: les algorithmes itératifs des plus proches voisins et les algorithmes basés sur les descripteurs.

Figure 20: Démonstration de l'algorithme «Iterative Closest Point»



Les algorithmes itératifs des plus proches voisins

Ces algorithmes utilisent une structure de donnée qui se nomme arbre kd. Les arbres kd sont une forme d'arbre binaire qui classent les points d'un nuage en fonction de leur emplacement par rapport aux autres points dans leur nuage. Cette structure permet d'optimiser la recherche du plus proche voisin. Le principe de ces algorithmes, c'est d'utiliser la recherche du plus proche voisin pour estimer une transformation 3D entre les deux nuages de points. Ils appliquent ensuite la transformation à un nuage de points. Ensuite, ils réitèrent la recherche, ré-estiment une nouvelle transformation 3D, réappliquent la transformation sur un nuage de points jusqu'à tant que la correspondance entre les nuages de point soit satisfaisante. En général, la distance de correspondance maximale est définie par l'utilisateur de l'algorithme. PCL offre plusieurs types d'algorithmes itératifs du plus proche voisin. Les plus importants seront décrits ci-dessous.

ALGORITHME ITÉRATIF DU PLUS PROCHE VOISIN

Dans PCL, c'est l'algorithme itératif de plus proche voisin qui est de base. Les autres algorithmes de la même famille sont basés sur lui. Il y a trois paramètres importants à savoir pour utiliser cet algorithme: la distance maximale de correspondance, le nombre maximal d'itérations pour avoir une convergence et le epsilon. La distance maximale de correspondance permet de définir la distance maximale acceptable que deux nuages de point peuvent avoir après l'application des transformations 3D. Si l'algorithme n'est pas capable de rapprocher les deux nuages de points, la correspondance entre eux est considérée invalide et elle est rejetée. Le nombre d'itérations maximal définit le nombre de fois que l'algorithme peut itérer pour trouver la transformation 3D finale entre deux nuages de points. Si l'algorithme ne réussit pas à converger dans le nombre d'itérations fourni, la correspondance est rejetée. Bien entendu, ces deux paramètres sont intimement liés, si la distance maximale de correspondance est grande, l'algorithme va converger rapidement à un résultat, mais avec une juxtaposition de mauvaise qualité. Pour avoir, une juxtaposition de qualité, il faut diminuer cette distance, mais en

contrepartie, il faut augmenter le nombre d'itérations pour augmenter les chances de convergence. Cependant, plus le nombre d'itérations est grand, plus l'algorithme risque d'être lent. Le dernier paramètre est l'epsilon. Ce paramètre définit la tolérance de l'algorithme à l'imprécision de la transformation. En général, ce paramètre ne varie pas beaucoup.

ALGORITHME ITÉRATIF DU PLUS PROCHE VOISIN GÉNÉRALISÉ

Cet algorithme utilise l'algorithme précédent, mais en plus il applique une correction supplémentaire sur la transformation obtenue par l'algorithme itératif du plus proche voisin de base. Cet algorithme a été décrit pour la première fois par Aleksandr V. Segal, Dirk Haehnel et Sebastian Thrun qui sont trois étudiants de l'université Stanford. C'est une fonction de coût anisotropique qui est utilisée pour la correction. L'anisotropie signifie «qui dépend de la direction». Cette correction permet de raffiner l'alignement des deux nuages de point, mais avec un petit coût de performance. Il utilise les mêmes paramètres que l'algorithme de base. Il a un paramètre epsilon pour la transformation de rotation en plus. Ce paramètre fait un travail semblable à l'autre paramètre epsilon mais plus spécifiquement pour la rotation. Il ne varie pas beaucoup tout comme l'autre paramètre.

ALGORITHME ITÉRATIF DU PLUS PROCHE VOISIN NON LINÉAIRE

Il est basé sur l'algorithme de base. Cependant, il optimise la performance de la méthode avec un algorithme de Levenberg-Marquardt. Cette optimisation n'améliore pas les résultats de la transformation obtenue, mais améliore la vitesse d'exécution de l'algorithme de base. Cette méthode utilise les mêmes paramètres que la méthode de base.

ALGORITHME ITÉRATIF DU PLUS PROCHE VOISIN AVEC NORMALES

Voici le dernier algorithme basé sur la méthode itérative du plus proche voisin. Il utilise le même principe, mais il utilise un algorithme de point à plan pour calculer la transformation entre deux nuages de points. En général, il offre de meilleurs résultats que la méthode de base.

Les algorithmes basés sur les descripteurs

Ces algorithmes vont être abordés brièvement, car il n'ont pas été testés durant ce projet. Il a été choisi de s'attarder à l'approche avec le plus proche voisin, car ils sont plus faciles à utiliser et ils donnent des résultats satisfaisants. Cependant, avec un peu plus de temps, ces types de méthodes auraient été explorés. C'est la deuxième grande approche utilisée pour juxtaposer deux nuages de points. Elle ressemble beaucoup à l'approche précédente, mais n'utilise pas un arbre kb; elle utilise des descripteurs de nuages de points. Avec ces descripteurs, des éléments semblables peuvent être déduits entre deux nuages de points. Ensuite une matrice transformation peut être déduite de ces éléments pour juxtaposer les deux nuages de point. PCL offre plusieurs types de descripteurs.

Algorithme de lissage et de débruitage des nuages de point

Dans cette section, ce sont tous les algorithmes de lissage et de débruitage utilisés et testés durant le projet qui vont y être décrits. Ces algorithmes sont utilisés avant la phase de juxtaposition pour éliminer du bruit afin d'augmenter les chances de convergence des méthodes de juxtaposition. Ils sont aussi utilisés après la phase de juxtaposition pour débruiter et lisser le résultat final.

Filtre médian

Ce filtre a été utilisé pour débruiter les nuages de point. Il utilise un élément de la statistique qui se nomme la médiane. Une fenêtre d'une grandeur définie est configurée par le

développeur. Ensuite l'algorithme trouve la médiane de la distance entre les points contenus dans cette fenêtre et applique le médian trouvé à tous les points de la fenêtre. PCL offre ce filtre dans ses outils de filtrage. Il a été utilisé au début du projet.

Filtre bilatéral

Le filtre bilatéral est un filtre non linéaire qui dérive du filtre de bruit gaussien. Il peut être utilisé pour débruiter un nuage de points, mais il ne peut être utilisé sur le résultat final pour un beau rendu. PCL offre ce filtre, mais il ne peut être utilisé qu'avec des types de points particuliers et les nuages de points donnés par la caméra utilisée ne sont pas de ce type.

Filtre de grille de voxel

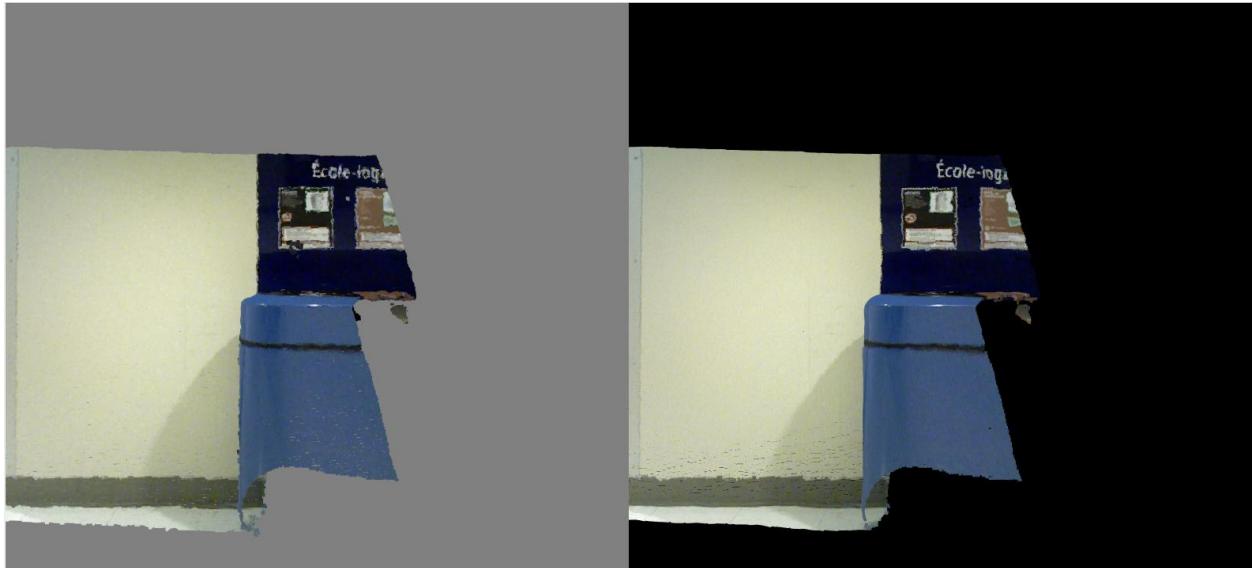
Ce filtre est nécessaire pour optimiser la performance de l'algorithme de juxtaposition. Ce filtre est utilisé pour faire du sous-échantillonage, c'est-à-dire diminuer le nombre de points redondants. Il est utilisé deux fois durant le processus de juxtaposition des points. D'abord, il est utilisé pour réduire la taille des nuages de points avant de les passer à l'algorithme itératif du plus proche voisin. Après, il est utilisé avec le nuage juxtaposé pour éliminer les points redondants. Ce filtre a trois paramètres. Ces paramètres sont en fait la paramétrisation de la grandeur d'un voxel en longueur, largeur et profondeur. Le nuage de points filtré va être divisé en voxel. Lorsqu'il y a plusieurs points dans un même voxel, les points supplémentaires sont éliminés pour en avoir un seul par voxel.

Filtre «Moving Least Square»

«Moving Least Square» est un filtre de ré-échantillonnage qui provient de PCL. Il utilise des fonctions polynomiales pour estimer les surfaces possiblement présentes dans le nuage de points fourni. Ensuite, il utilise ces surfaces estimées pour réaligner les points du nuage. Il fait un travail de débruitage et de lissage en même temps. Il est excellent avec les nuages de points qui contiennent des plans et des formes polygonales. Il est donc prédestiné à être utilisé avec des formes carrées comme des pièces ou des intérieurs de bâtiment. Ce filtre est utilisé dans

l'algorithme final de reconstruction pour filtrer les nuages de points bruts provenant de la caméra et pour le lissage final du rendu 3D.

Figure 21: Démonstration du filtre MLS



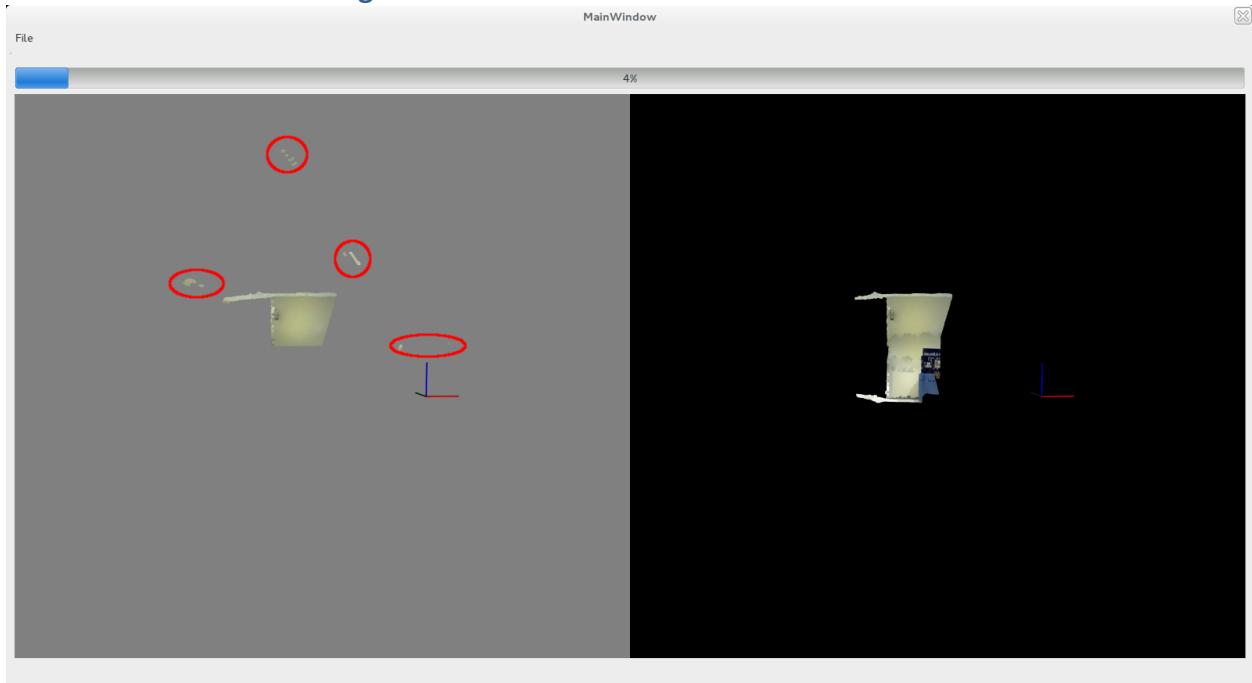
Filtre euclidien pour l'extraction de grappes de points

L'implémentation dans PCL se nomme «EuclideanClusterExtraction». Ce filtre est à vrai dire un filtre de grappage ou «Clustering». Il ne sert pas à l'origine à débruiter un nuage de points. Ce filtre trouve des agglomérations de points à l'intérieur d'un nuage et retourne le nombre de ces agglomérations de points et la position des points à l'intérieur de ces grappes. Dans le cadre de ce projet, il est utilisé pour enlever des artefacts produits par la caméra qui sont issus des réflexions.

Ces réflexions sont souvent des petits groupes de points qui sont grandement séparés du nuage de points principal. Cependant, ces petites agglomérations de points non désirées nuisent à la reconstruction 3D et diminuent l'esthétique du rendu final. Le filtre euclidien pour l'extraction de grappes de points a trois paramètres importants: la distance maximale entre les points d'une grappe, la grandeur minimale d'une grappe et la grandeur maximale d'une grappe.

Le premier paramètre sert à déterminer la distance maximale que peuvent avoir les points entre eux pour être considérés comme une agglomération valide. Le deuxième paramètre et le troisième paramètre servent à déterminer la grosseur valide d'une agglomération de points. Les deux derniers paramètres sont intéressants pour le filtrage des réflexions, car ces artefacts de réflexions forment toujours les plus petites grappes de points. Ainsi, en mettant le paramètre de grandeur minimale et maximale optimale, le «EuclideanClusterExtraction» enlève les réflexions qui sont des grappes de points invalides et garde les gros nuages de points qui proviennent vraiment de la pièce numérisée.

Figure 22: Démonstration du filtre euclidien



Filtres d'améliorations des couleurs

D'une numérisation à une autre, les conditions d'éclairage peuvent changer. Et en plus, la caméra s'autocalibre en fonction de ces conditions. Cela a pour effet de modifier les teintes de couleurs d'un nuage de points à un autre. Cela produit des effets de couleurs non désirés dans la reconstruction 3D finale. Voilà pourquoi plusieurs algorithmes ou méthodes ont été essayés pour corriger cela.

Comme c'est un problème de teinte et de luminosité, il a été pensé en premier de convertir les couleurs RGB en HSV et d'uniformiser la lumisité par le paramètre V du HSV. Cependant, la conversion de RGB et HSV n'est pas toujours parfaite et cela donnait des effets de couleurs non souhaités. Les couleurs filtrées pouvaient différer grandement des couleurs originales.

Une autre approche utilisée était de calculer la luminosité des couleurs RGB avec la formule suivante: $Y = 0.2126*R + 0.7152*G + 0.0722*B$. Ensuite, cette luminosité était normalisée à une valeur prédéterminée. Les couleurs filtrées gardaient leur intégrité en teinte, mais dans certains cas, la couleur devenait trop pâle. En fait, chaque couleur a une luminosité différente dans un même éclairage donc, en normalisant avec une même luminosité, les images ressortent en couleur pastel et cela n'est pas souhaité pour avoir une reconstruction réaliste.

La dernière approche testée est un mélange des deux précédentes. La méthode était divisée en trois étapes. La première étape était de classer les points du nuage selon leur teinte. La teinte était trouvée en convertissant les couleurs RGB en HSV, et le H représentait la teinte. Ensuite, la luminosité de chaque point était trouvée avec la formule vue ci-haut. Pour chaque teinte trouvée, une moyenne de la luminosité était calculée. Et finalement, chaque point de chaque teinte trouvé avait sa luminosité ajustée à la moyenne de brillance de sa teinte qui a été trouvée à l'étape précédente. L'intégrité des couleurs était conservée, mais la conversion en HSV ne fonctionne pas bien avec les blancs, les noirs et les teintes de gris. Comme les murs des pièces sont souvent blancs ou de couleur très pâle, cela affectait le rendu des couleurs grandement. La conversion des teintes de gris en HSV classait ces couleurs dans n'importe quelle teinte. Cette approche a dû être abandonnée.

Pour finir cette section sur les filtres de couleurs, il n'y a pas de bonne solution qui a été trouvée. Cependant, il y a des filtres de détection et de suppression des ombrages qui n'ont pas été explorés par manque de temps. En plus, ces algorithmes sont beaucoup plus complexes à implémenter.

Description de l'algorithme de reconstruction 3D choisi

Dans cette section, il y aura une description et une explication détaillée sur le fonctionnement de l'algorithme de reconstruction 3D choisi pour le projet. Il y aura une explication de comment les algorithmes présentés dans la section précédente seront intégrés dans cet algorithme. Il va y avoir trois sous-sections qui vont détailler trois grandes phases de la méthode de reconstruction. La première phase est celle qui s'occupe des nuages de points bruts provenant de la caméra TOF avant de les envoyer à l'algorithme de reconstruction. La deuxième phase est celle de la juxtaposition des nuages de points pour en faire une reconstruction 3D. Et la dernière phase, c'est le lissage de la reconstruction complète pour avoir le rendu final qui sera enregistré.

Phase 1 : débruitage et lissage des nuages de points bruts

Cette phase utilise deux algorithmes de filtrage vus dans la section précédente. Elle utilise le filtre MLS(Moving Least Square) et le filtre euclidien de recherche de grappe de point (EuclideanClusterExtraction). Cette étape est importante pour la reconstruction, car elle permet d'offrir un nuage de points de qualité à l'algorithme de juxtaposition. Cela optimise la qualité et la performance de la reconstruction finale.

D'abord, la numérisation provenant de la caméra directement contient plusieurs artéfacts de réflexion et de réfraction. Ces artéfacts sont présents dans le nuage de points brut sous forme d'agglomérations de points qui sont grandement éloignées du nuage de points principal. Pour enlever ces artéfacts, c'est le filtre «EuclideanClusterExtraction» qui est utilisé. Le nuage de points brut est donc passé par ce filtre en premier et le nuage résultant contient une grande agglomération de points unis et sans artéfacts de réflexion.

Ensuite, le nuage de points filtré n'est pas encore prêt à être passé à l'algorithme de reconstruction. Le nuage a encore besoin d'être lissé et débruité. Il est encore un peu flou et les points qui forment les murs ne forment pas un beau plan. Voilà pourquoi il est passé à travers

un filtre “Moving Least Square”. Ce filtre débruite et lisse en même le temps le nuage de points. Ce filtre replace les points des murs sur un même plan et améliore la netteté de l’image.

Et finalement, le nuage de point doublement filtré est passé à l’algorithme de juxtaposition. Cependant, il faut préciser quelques petits détails. Si c’est la première numérisation de la reconstruction qui est passée c’est-à-dire que c’est l’initialisation de l’algorithme, ce premier nuage de point est ajouté directement à la reconstruction 3D. Ce premier “scan” va servir de point d’ancrage pour la juxtaposition des numérisations suivantes.

Phase 2 : la juxtaposition des nuages de points

Voici le cœur de l'algorithme de reconstruction, la méthode de juxtaposition des nuages de points. C'est aussi la partie la plus complexe de l'algorithme. Il y a trois sous-étapes principales dans cette phase. Il y a une étape de sous-échantillonnage des nuages de points avec le filtre "VoxelGrid". Cette sous-étape sert pour optimiser la vitesse de traitement de l'algorithme suivant. Ensuite, les deux nuages de points sous-échantillonés sont passés à l'algorithme de juxtaposition qui se nomme "GeneralizedIterativeClosestPoint" pour trouver la transformation 3D nécessaire pour les coller ensemble. La transformation 3D est appliquée aux nuages de points et ils sont ajoutés à la reconstruction finale. Et finalement, un léger sous-échantillonnage est appliqué sur la reconstruction finale avec "VoxelGrid" afin d'éliminer les points redondants à la jonction de la superposition des deux nuages. Ces trois sous-étapes sont détaillées ci-bas.

L'algorithme de juxtaposition reçoit en paramètre trois nuages de points: la nouvelle numérisation provenant de la phase 1, la numérisation précédemment transformée et la reconstruction courante. Avec la numérisation courante et la numérisation précédente transformée , il y a création de deux nouveaux nuages de points sous-échantillonés avec le filtre "VoxelGrid". Le nuage de points provenant de la numérisation courante est maintenant appelé la source. C'est le nuage de points qui va être transformé pour être juxtaposé sur la reconstruction courante. Le nuage de points provenant de la transformation précédente est appelée la cible. C'est ce nuage de points que l'algorithme ICP va utiliser pour estimer la transformation 3D qui va être appliquée sur le nuage source. Pourquoi faut-il que les nuages de points soient sous-échantillonés? Parce que l'algorithme ICP utilise des arbres kd pour ses estimations et ces arbres deviennent très gros avec un gros nuage de points et l'estimation de la transformation devient très lent. En sous échantillonnant la taille des nuages passés à ICP, la taille des arbre kd rapetisse. Il n'est pas nécessaire de passer tous les points pour voir une bonne estimation. La rapidité de l'algorithme est grandement améliorée avec une qualité de reconstruction semblable.

Maintenant que les nuages sont sous-échantillonés et que la source et la cible ont été déterminées, ces entités vont être passées à la méthode "GeneralizedIterativeClosestPoint" ou

pour être plus court la méthode GICP pour estimer la transformation 3D. GICP a deux autres paramètres principaux en plus des deux nuages de points. Comme, il a été vu précédemment, les méthodes ICP ont un nombre d’itérations et une distance maximale de correspondances configurables. GICP a donc ces paramètres. Le nombre d’itérations maximales de GICP a été configuré à son initialisation. En général, le nombre d’itérations doit être élevé pour augmenter les chances de convergence de l’algorithme. Pour ce projet, ce paramètre a été paramétré à 1000. Il a été déterminé de manière empirique avec des tests visuels. Si ce paramètre est trop élevé, GICP perd de la performance et si ce paramètre est trop bas, le taux de convergence devient trop bas et GICP n’arrive pas à estimer une transformation. Par contre, la distance maximale de correspondance est déterminée dynamiquement par l’algorithme de reconstruction, car si cette distance est trop petite, il n’y a pas de convergence. Si cette distance est trop grande, GICP converge, mais la transformation trouvée n’est pas valide. Par conséquent, la juxtaposition n’est pas valide et les juxtapositions suivantes sont gâchées. La solution trouvée a été de fixer une distance de correspondance initiale petite. Si “GeneralizedIterativeClosestPoint” ne converge pas avec cette distance, elle est progressivement augmentée jusqu’à ce que GICP converge à une estimation ou jusqu’à une distance maximale. En général, les numérisations convergent avec une distance maximale de correspondance située entre 0.005 mètre et 0.015 mètre. Une fois que GICP trouve une matrice de transformation pour le nuage de points source, cette matrice est appliquée sur le nuage source complet, c'est-à-dire le nuage de points qui n'a pas été sous-échantillonné. Ensuite le nuage de points transformé est copié pour être utilisé pour le prochain “scan”. Il va être utilisé comme le nuage de cible pour la prochaine itération. Ensuite, le nuage de points source transformé est ajouté à la reconstruction courante.

Et finalement, le nuage de points de la reconstruction courante est légèrement sous-échantillonné avec un filtre “VoxelGrid”. Ce filtrage a pour but d’enlever les points redondants qui sont présents dans la jonction du nouveau nuage de points ajouté et du reste du nuage de points. Maintenant, l’algorithme attend une nouvelle numérisation de la phase 1. Si c’est le dernier nuage de points numérisé, l’algorithme passe à la phase 3.

Phase 3 : Lissage du rendu final

Une fois que toutes les numérisations 3D brutes ont été traitées par la phase 1 et qu'elles ont été ajoutées à la reconstruction courante, cette phase peut commencer. Dans cette phase, la reconstruction entière va être lissée pour tenter d'éliminer les jonctions de nuages qui paraissent visuellement. Les ombrages et les différences de couleurs d'une numérisation à l'autre paraissent dans le rendu final et cela diminue la qualité graphique de la reconstruction. À défaut d'avoir trouvé mieux, le rendu final est repassé dans le filtre MLS. Cependant, le filtre MLS est bien pour débruiter et améliorer la forme d'un nuage de points en provenance d'une numérisation brute provenant d'un caméra TOF, mais il n'est pas indiqué pour corriger les problèmes de couleurs. Au final, le filtre MLS améliore tout de même le rendu total, car il réaligne un peu les jonctions entre les nuages de points, mais il fait peu de choses contre les différences de couleurs.

RÉSULTATS ET DISCUSSION

Fonctionnement du robot

Localisation et cartographie 2D

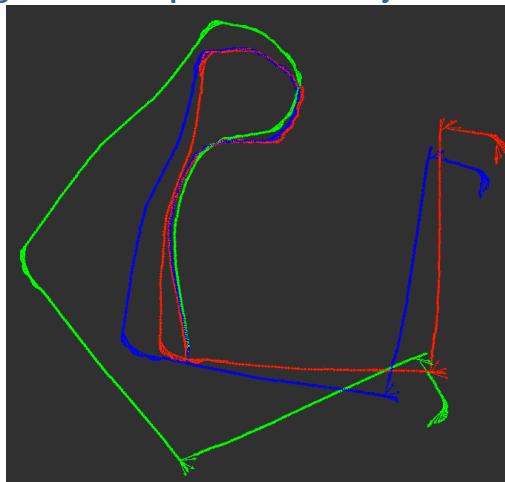
Localisation sans carte statique

Comme expliqué précédemment, l'angle renvoyé par les moteurs n'est pas facilement utilisable à cause du trop gros décalage qui se produit lorsque le robot tourne. Se fier uniquement sur cette valeur empêcherait de fournir quoi que ce soit d'acceptable. Toutefois, les tests ont démontré que le déplacement parcouru calculé se rapproche très bien de la réalité.

Pour ce qui est de l'algorithme CSM, il a été découvert lors des tests que la position retournée n'est pas du tout fiable. Lors d'un déplacement dans un long corridor sans beaucoup de détails, il est impossible de connaître le déplacement puisque les numérisations laser sont presque toutes identiques. Toutefois, en ce qui a trait aux rotations, CSM s'est révélé être très fiable. Un filtre de Kalman étendu a donc été utilisé en utilisant la position des moteurs et l'angle retourné par CSM en entrée.

L'image ci-dessous représente le trajet calculé par les moteurs (en vert), par le filtre de Kalman (en bleu), ainsi que le trajet réel (en rouge).

Figure 23: Comparaison des trajets calculés



On constate que les angles calculés par les moteurs sont effectivement décalés par rapport à la réalité.

Pour ce qui est de la position retournée par le filtre de Kalman, elle semble quand même plutôt acceptable malgré un léger décalage par rapport à la réalité. Elle pourrait donc être utilisée plus tard lors de la génération de la carte en deux dimensions et de la localisation dans cette carte.

Génération d'une carte 2D

Les trois solutions de cartographies proposées précédemment ont été testées au premier étage du pavillon A de l'ÉTS dans le but de les comparer. Les cartes suivantes représentent la salle des pas perdus en bas à droite, ainsi que les corridors adjacents.

Cet endroit a été choisi à cause de sa complexité, des longs corridors sur le chemin, ainsi que pour tester la capacité des différents algorithmes à fermer la boucle.

Figure 24:

Solution 1 : CSM + EKF + GMapping

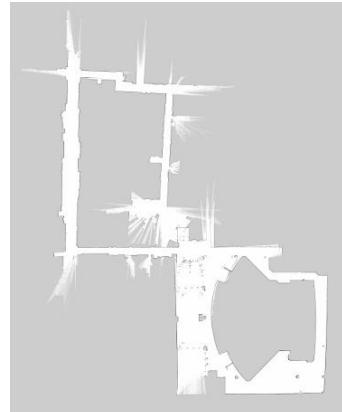


Figure 25: Solution 2 :

Moteurs + GMapping

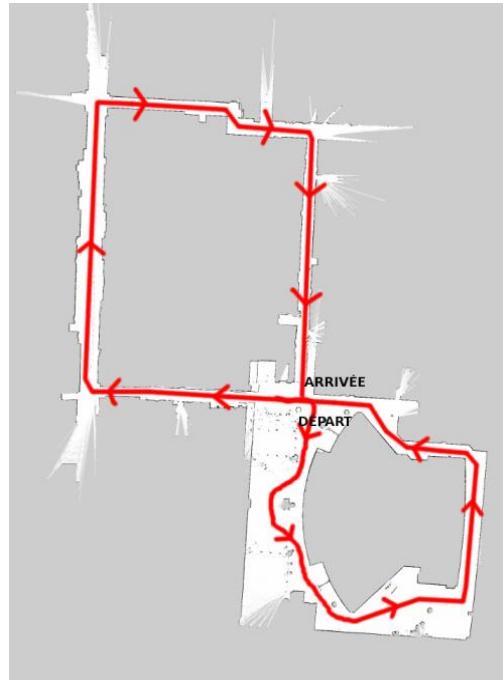


Figure 26: Solution 3 : Hector SLAM



Le trajet parcouru pour la génération des trois cartes est indiqué ci-dessous. Les points d'arrivée et de fin sont situés à l'intersection entre les 2 corridors et la salle des pas perdus.

Figure 27: Trajet parcouru pour les tests



Plusieurs caractéristiques peuvent être observées lors de la comparaison des différentes cartes.

En premier lieu, à l'intersection entre les points de départ et d'arrivée, on remarque que les trois solutions ne ferment pas la boucle de la même façon.

Figure 28: Solution 1



Figure 29: Solution 2



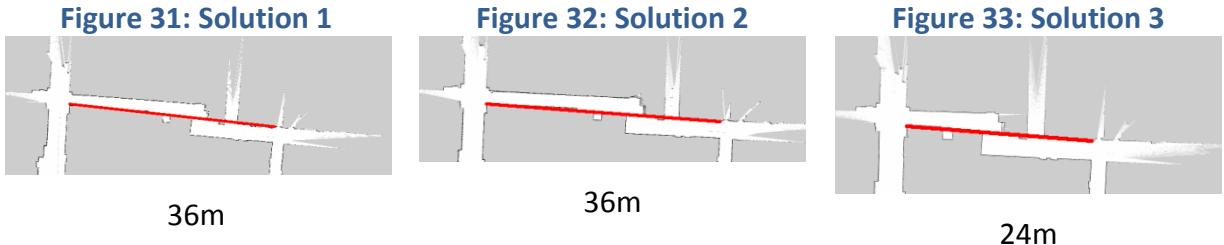
Figure 30: Solution 3



La solution 3 n'offre pas du tout un résultat acceptable à ce niveau.

De plus, bien que les deux premières se ressemblent, la solution 1 présente légèrement moins d'artéfacts.

En second lieu, le corridor situé en haut complètement de la carte présente peu d'obstacles et de caractéristiques. La mesure de ce corridor a été calculée à partir de la carte générée par chaque solution.



La solution 3 (Hector), qui n'utilise pas d'odométrie n'a pas réussi à bien performer dans cette section. L'algorithme croyait pendant un bref instant que le robot était arrêté étant donné qu'il n'y avait pas de différence majeure entre les différentes numérisations laser. Les deux autres solutions avec l'algorithme GMapping n'ont pas connu ce problème puisque des données d'odométrie venaient améliorer sa précision.

Il a donc été convenu d'utiliser la solution 1 pour générer la carte utilisée par le robot lors de la navigation autonome.

Localisation dans l'environnement à l'aide d'une carte statique

Il a été très difficile d'obtenir des résultats satisfaisants en ce qui a trait à la localisation dans la carte statique.

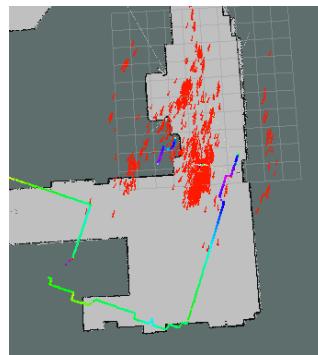
Il arrivait très souvent que le robot n'était plus capable de connaître sa position, rendant ainsi la navigation impossible.

Les différentes solutions possibles ont été comparées, en offrant une précision de plus en plus élevée. Dans les images qui suivent, les flèches rouges représentent les estimations de position utilisées par AMCL. Plus elles convergent, plus la position est précise. L'objectif est que les lignes de couleur générées par le LiDAR s'emboîtent parfaitement sur la carte située en arrière-plan, signifiant ainsi une bonne position.

Solution 1: AMCL avec moteurs

Cette solution était complètement inutilisable. Les scans étaient rarement ajustés avec la carte et une position valide était rarement obtenue.

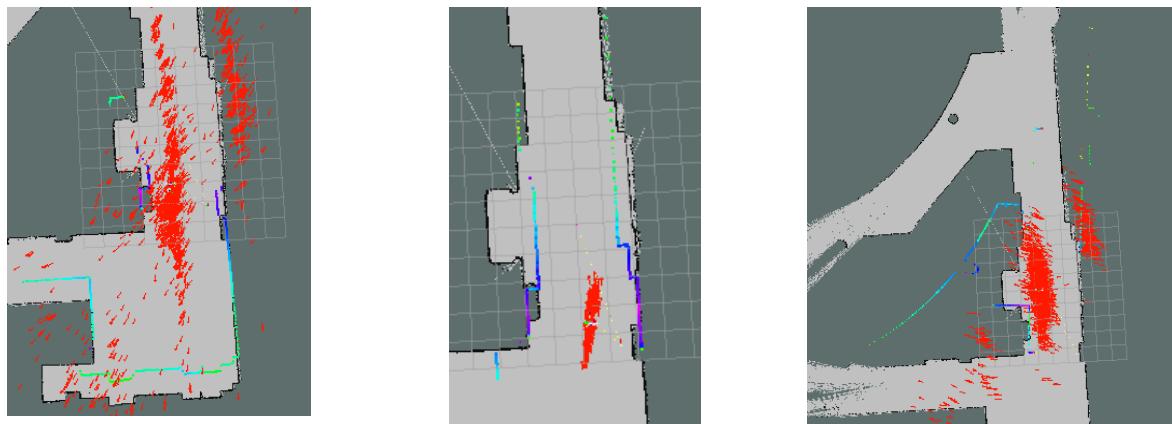
Figure 34: Résultat de la solution 1



Solution 2: CSM + AMCL

Cette solution fonctionnait pendant quelques instants et des résultats absurdes se mettaient à apparaître dès que le robot tournait ou que les scans étaient trop semblables (dans un corridor sans beaucoup de détails). On remarque, surtout dans la troisième image, qu'un décalage est survenu.

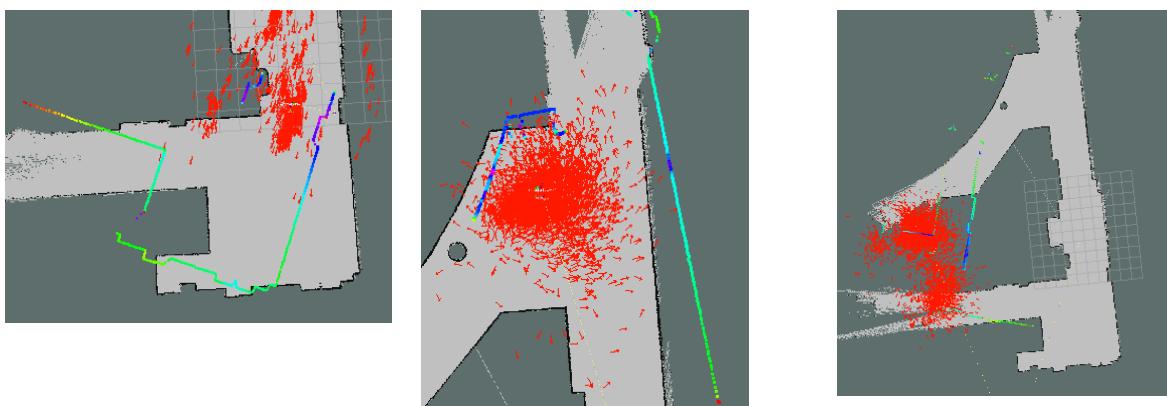
Figure 35: Résultats de la solution 2



Solution 3: CSM + AMCL avec moteurs

Cette solution fonctionnait relativement bien pendant un certain temps, puis, dès qu'une difficulté était rencontrée, commençait à retourner des positions aberrantes parfois situées en dehors de la carte. Des téléportations survenaient aussi, rendant le tout inutilisable.

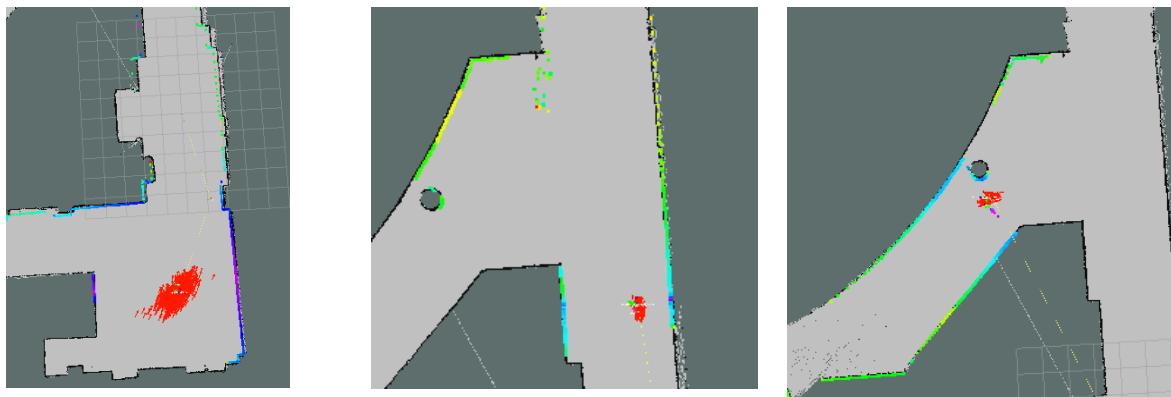
Figure 36: Résultats de la solution 3



Solution 4: CSM + EKF + AMCL

Cette technique était celle fournissant les meilleurs résultats. Les scans tendaient la plupart du temps vers un alignement avec la carte, même si des légères fluctuations apparaissaient. Il arrivait aussi qu'un plus gros décalage survienne, mais il se corrigeait automatiquement après quelques secondes.

Figure 37: Résultats de la solution 4



La solution 4 semble donc être le mieux qu'il soit possible de faire avec les capteurs disponibles.

La meilleure option serait d'avoir un IMU fonctionnel et d'utiliser un filtre de Kalman en utilisant ses valeurs d'orientation. La position deviendrait alors excellente puisqu'il serait possible de corriger les problèmes d'orientation des moteurs. Toutefois, ce n'était pas une option dans le cadre du projet.

Navigation

La navigation a été testée au premier étage du pavillon A de l'ÉTS, près du local A-1150 et de l'ascenseur adjacent.

La planification globale a d'abord été évaluée en s'assurant qu'aucun obstacle dynamique n'était présent. Des objectifs étaient envoyés en utilisant le logiciel de visualisation «rviz» fourni avec ROS. Par la suite, la planification locale a été testée en positionnant dynamiquement différents obstacles devant le robot et en regardant comment il réagissait.

La majorité des objectifs étaient bien atteints, sauf dans certains cas:

- Lors d'une rotation trop rapide, il arrivait que le robot perde sa position dans la carte, se croie ailleurs et cesse d'avancer en se pensant bloqué par un mur qui n'existe pas vraiment. La solution a été de diminuer la vitesse de rotation du robot de telle façon à ce qu'il soit très improbable que le robot perde sa position.
- Lorsque le robot passait trop près d'un mur, il arrivait qu'il se croie bloqué, performait quelques manœuvres désespérées pour se déprendre et cessait d'avancer. Ce problème peut toutefois être ignoré dans le cadre du projet, puisque l'intelligence artificielle ne fournira jamais d'objectifs passant près d'un mur.

Figure 38: Envoi d'un objectif au robot en utilisant l'outil rviz

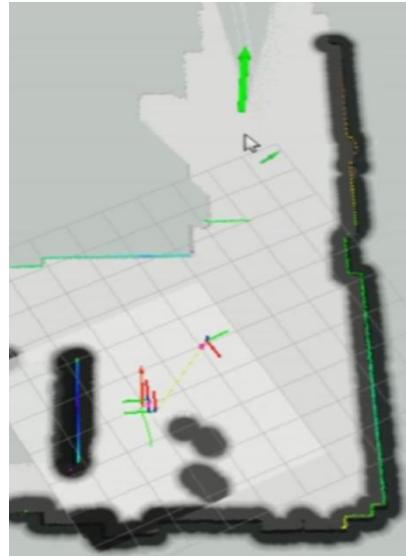


Figure 39: Évitement d'un obstacle dynamique: avant

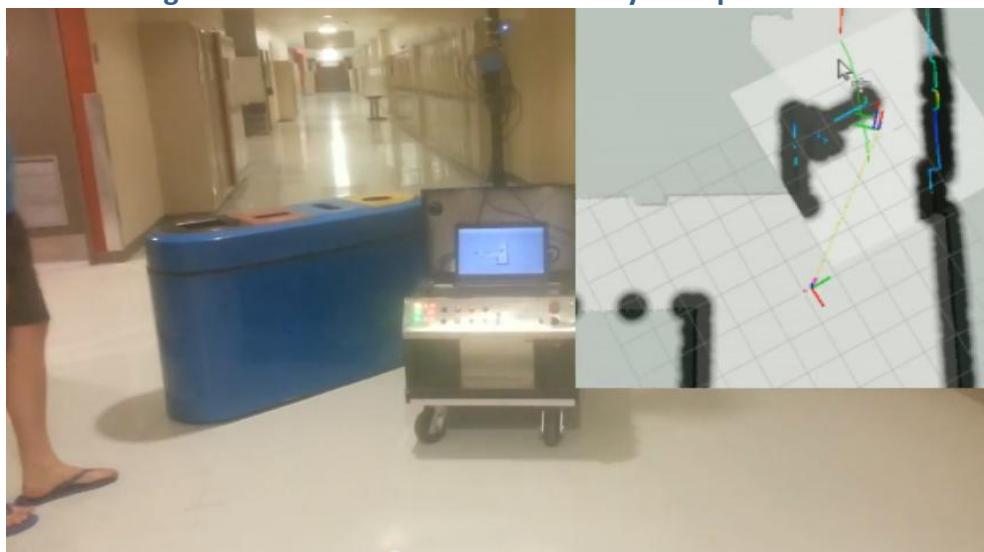


Figure 40: Évitement d'un obstacle dynamique: après



Une vidéo de ce test est disponible en ligne: <https://www.youtube.com/watch?v=8uoFwVkeRU>

Les résultats offerts par la navigation sont très acceptables dans le cadre du projet et pour offrir une bonne base à l'intelligence artificielle.

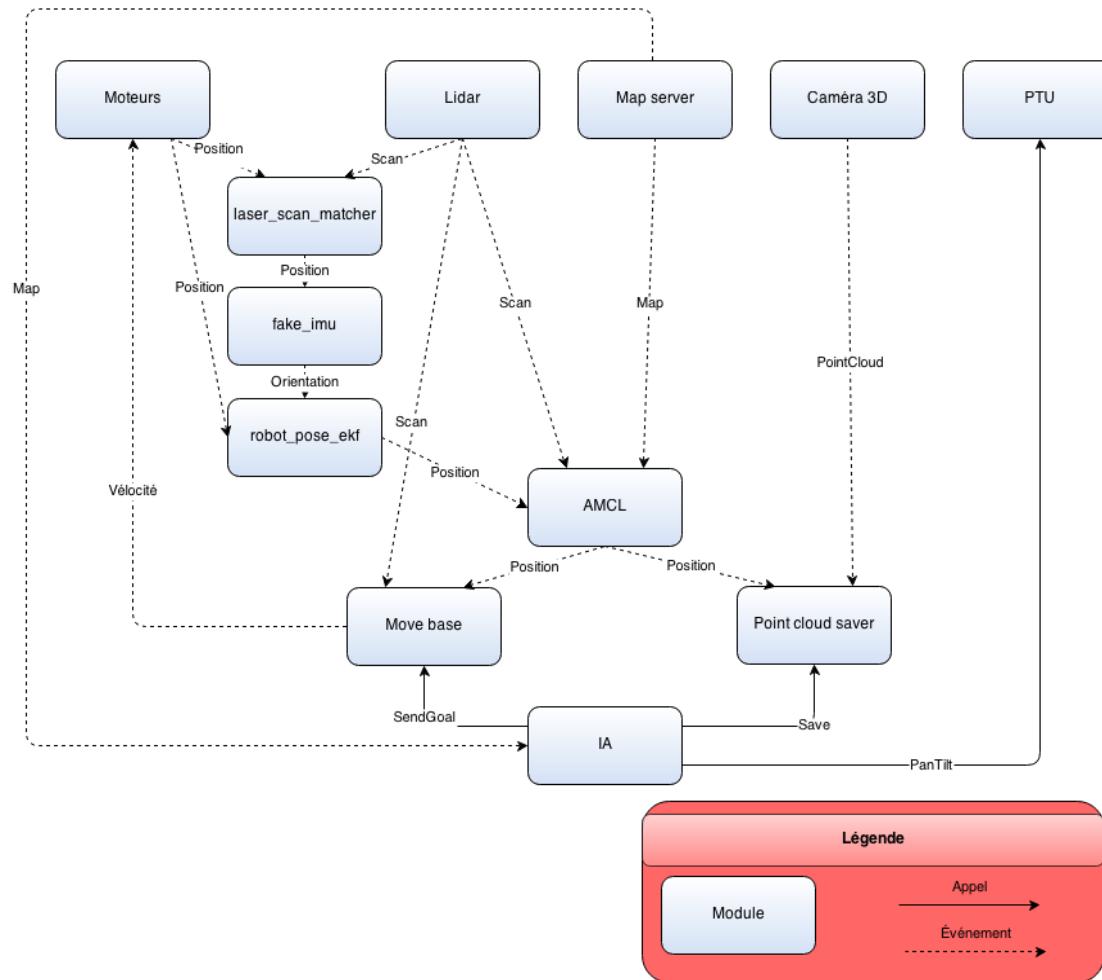
Un point à améliorer serait la détection d'obstacles non uniformes sur la hauteur. En effet, le LiDAR ne voit qu'à une certaine altitude, plutôt basse. S'il rencontrait une table, il ne verrait donc que ses pattes et risquerait d'entrer en collision avec le dessus et de s'endommager.

Architecture logicielle

L'architecture résultante peut maintenant être présentée maintenant que tous les choix au niveau des algorithmes à utiliser pour la localisation et la cartographie 2D ont été effectués. Les diagrammes ci-dessous présentent le flux de données entre les différents modules utilisés. Ils ont été légèrement simplifiés par souci de compréhension et de lisibilité. L'idée reste toutefois la même.

Déplacement autonome dans un environnement connu

Figure 41: Architecture du déplacement autonome dans un environnement connu



On constate donc que tout commence avec les moteurs et le LiDAR qui envoient leurs informations au reste du système.

Par la suite, le noeud «`laser_scan_matcher`»(CSM) utilise l'information reçue des deux capteurs afin de calculer une position approximative. Cette position est envoyée au noeud «`fake_imu`» qui l'utilise afin de simuler une boussole. L'orientation résultante et la position des moteurs sont ensuite envoyées au filtre de Kalman qui calcule une meilleure approximation de la position.

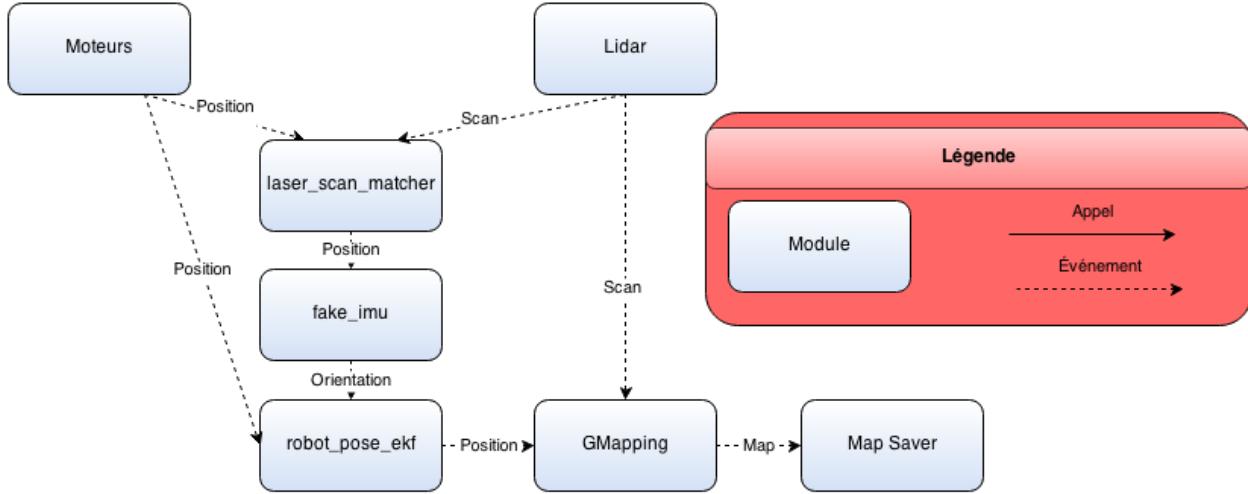
Pour améliorer cette position encore plus, l'algorithme AMCL utilise en plus la carte statique et les scans laser. Elle sera ensuite utilisée par «`move_base`» et par le sauvegardeur de nuage de points.

En parallèle, l'intelligence artificielle vérifie si «`move_base`» a atteint l'objectif qui lui a été envoyé. Si c'est le cas, une commande est envoyée au PTU afin de positionner correctement la caméra. L'IA demande ensuite au noeud «`pointcloud_saver`» d'enregistrer les informations que la caméra voit.

Une fois cela fait, un nouvel objectif est envoyé à «`move_base`», qui enverra des commandes de vitesse aux moteurs afin de déplacer le robot.

Génération de la carte statique de l'environnement

Figure 42: Architecture de génération de la carte statique de l'environnement



Le principe de base est le même que pour le diagramme précédent. Toutefois, au lieu d'utiliser AMCL pour s'orienter dans une carte connue, la position et les scans sont envoyés au noeud GMapping afin de générer une carte basée sur ces informations. La carte peut ensuite être sauvegardée dans un fichier grâce au noeud «Map Saver».

Modularité

Cette architecture est très modulaire et beaucoup de composants sont facilement remplaçables. Pour utiliser le système sur un robot différent, il suffirait de remplacer les modules du moteur et du LiDAR, puis d'effectuer quelques changements de configuration sur la taille du robot.

Le PTU serait aussi très facilement remplaçable par un plus performant, pourvu qu'il offre des services équivalents («Pan», «Tilt» et/ou «PanTilt»).

Pour ce qui est de la caméra 3D, OpenNI est utilisé afin d'offrir une couche d'abstraction entre le pilote logiciel et PCL. Elle serait donc aussi extrêmement simple à remplacer.

Une autre amélioration possible serait de mélanger les deux diagrammes ensemble et de remplacer AMCL et le serveur de cartes par GMapping. Le robot pourrait donc se déplacer de manière autonome dans un environnement inconnu. Il faudrait toutefois aussi adapter l'intelligence artificielle pour qu'elle soit fonctionnelle dans cette configuration.

Intelligence artificielle

Pour analyser les résultats obtenus dans le module d'intelligence artificielle, il a fallu trouver des métriques appropriées classifiant la performance des algorithmes. La qualité n'est pas évaluée dans ce module, car dans ce contexte, la qualité dépend directement de la superficie des surfaces numérisées. Effectivement, un des objectifs principaux de ce module est d'avoir un processus de numérisation complet. Sauf indication contraire, tous les résultats présentés supposent que la superficie numérisée totale est maximale.

Des métriques seront présentées pour les 2 premières parties du module d'intelligence artificielle: les temps totaux minimum, maximum et moyen d'exécution pour chaque algorithme, et les distances maximales, minimales et moyennes pour les algorithmes trouvant le plus court chemin. De plus, certains cas particuliers ont été détaillés.

Il est important de noter que les cartes utilisées dans les tests ont toutes une résolution de 0.025 mètre par pixel. Chaque test est exécuté 100 fois pour obtenir une métrique minimale, maximale et moyenne. Les tests ont été exécutés sur un Acer Aspire utilisant un processeur AMD A6-3400M à 1.4 GHz et 6GB de mémoire RAM.

Finalement, une image représentant le résultat de l'algorithme testé et un tableau associé expliquant les valeurs utilisées sont inclus. Les tableaux suivant le tableau des paramètres du test de référence n'incluront que les résultats et les paramètres changés par rapport au test de référence.

Génération d'objectifs

L'algorithme de génération d'objectifs s'assure de générer et positionner des objectifs à atteindre par le robot. Une bonne métrique pour cet algorithme est son temps d'exécution. Plusieurs tests ont été effectués pour analyser la variation de plusieurs paramètres sur le temps d'exécution de l'algorithme:

- Variation de la taille de la carte, $m \times n$, en mètres
- Variation de la taille du filtre médian, $p \times q$, en mètres
- Variation de la taille des régions de partitionnement, $u \times v$, en mètres
- Variation de la distance minimale et maximale des murs, **a et b**, en mètres

Il est important de noter que le paramètre représentant la distance minimale entre chaque objectif dépend directement de la taille des régions de partitionnement et suit la formule $\sqrt{u^2 + v^2}$.

Chaque paramètre a subit une variation, indépendamment des autres paramètres. Les tests ont été comparés à un test de référence illustré dans la figure 43 et son tableau associé.

Figure 43: La carte du test de référence de la génération d'objectifs

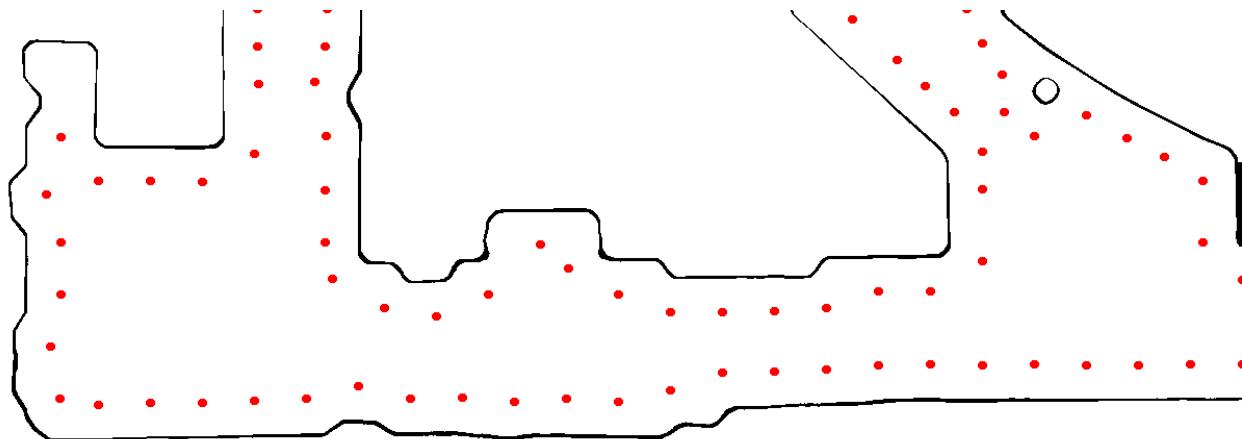


Tableau 8: Les paramètres du test de référence de la génération d'objectifs

Hauteur de la carte	12.5 mètres
Largeur de la carte	30 mètres
Taille du filtre médian	0.5 mètre
Hauteur des régions de partitionnement	0.625 mètre
Largeur des régions de partitionnement	0.625 mètre
Distance minimale entre chaque objectif	0.884 mètre
Distance minimale des murs	0.8 mètre
Distance maximale des murs	0.85 mètre
Temps d'exécution minimal	0.964 seconde
Temps d'exécution maximal	1.111 secondes
Temps d'exécution moyen	1.059 secondes

Variation de la taille de la carte

Le premier test consiste à faire varier la taille de la carte traitée. Les paramètres de hauteur et de largeur de la carte sont testés pour obtenir une carte plus grosse que le test de référence ainsi qu'une carte plus petite que le test de référence.

Figure 44: La carte du test pour une petite carte

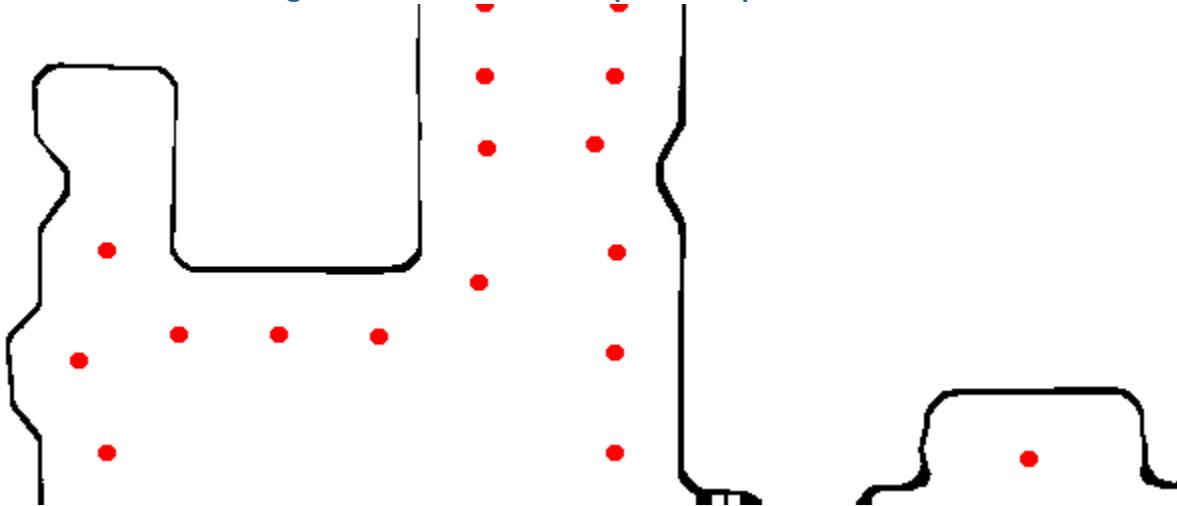


Tableau 9: Les paramètres du test pour une petite carte

Hauteur de la carte	6.25 mètres
Largeur de la carte	15 mètres
Temps d'exécution minimal	0.0735 seconde
Temps d'exécution maximal	0.0875 seconde
Temps d'exécution moyen	0.0765 seconde

Figure 45: La carte du test pour une grosse carte



Tableau 10: Les paramètres du test pour une grosse carte

Hauteur de la carte	18.75 mètres
Largeur de la carte	45 mètres
Temps d'exécution minimal	2.042 secondes
Temps d'exécution maximal	2.335 secondes
Temps d'exécution moyen	2.241 secondes

La grosseur de la carte affecte énormément le temps d'exécution de l'algorithme de génération d'objectif. Naturellement, si l'algorithme doit traiter une grande superficie, il prendra un temps considérable. Dans le cas d'une carte 1.5 fois plus grosse, l'algorithme prend 2 fois plus de temps. L'inverse est aussi vrai, mais ne semble pas être proportionnel. Pour une carte 2 fois plus petite, l'algorithme est presque 14 fois plus rapide.

Variation de la taille du filtre médian

Le deuxième test consiste à modifier la taille du filtre médian s'occupant du lissage de la carte brute. Comme le test de variation de la taille de la carte, 2 tests ont été exécutés pour un petit et un grand filtre médian.

Figure 46: La carte du test pour un petit filtre médian

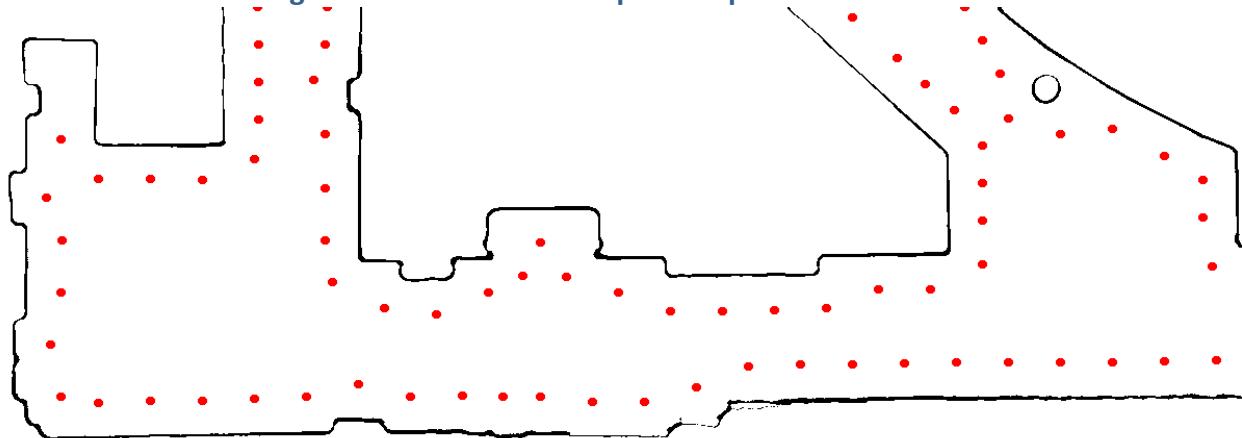


Tableau 11: Les paramètres du test pour un petit filtre médian

Taille du filtre médian	0.25 mètre
Temps d'exécution minimal	1.217 secondes
Temps d'exécution maximal	1.309 secondes
Temps d'exécution moyen	1.249 secondes

Figure 47: La carte du test pour un gros filtre médian



Tableau 12: Les paramètres du test pour un gros filtre médian

Taille du filtre médian	1 mètre
Temps d'exécution minimal	0.931 seconde
Temps d'exécution maximal	1.059 secondes
Temps d'exécution moyen	0.958 seconde

Plus la taille du filtre médian est grande, plus la carte sera lissée et vice-versa. Par contre, une carte lissée implique aussi une perte de précision de la position des murs. La performance est peu affectée par la variation de paramètre et, dans l'optique d'obtenir une meilleure reconstruction, il est préférable de choisir une valeur plus petite pour la taille du filtre médian. Une carte trop bruitée peut déclencher d'un problème de LiDAR ou d'une mauvaise cartographie.

Variation de la taille des régions de partitionnement

Ce troisième test affecte directement la quantité d'objectifs à placer sur une carte. Une grande quantité d'objectifs assure une numérisation complète, mais peut aussi entraîner des numérisations inutiles. Ce test fait varier la taille des blocs de partitionnement pour que ces derniers soient plus grands et plus petits que le test de référence.

Figure 48: La carte du test pour une petite région de partitionnement

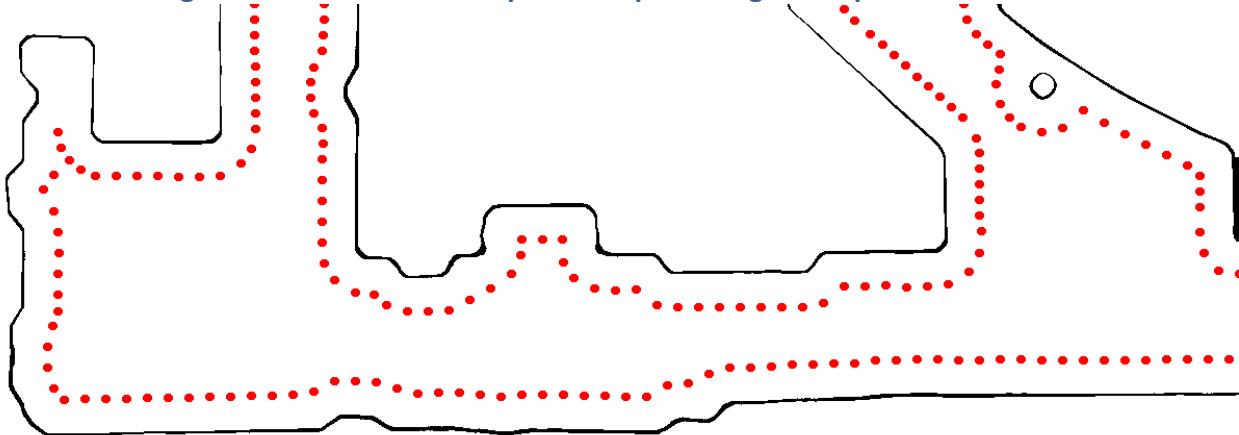


Tableau 13: Les paramètres du test pour une petite région de partitionnement

Hauteur de la région de partitionnement	0.25 mètre
Largeur de la région de partitionnement	0.25 mètre
Distance minimale entre chaque objectif	0.354 mètre
Temps d'exécution minimal	2.269 secondes
Temps d'exécution maximal	2.597 secondes
Temps d'exécution moyen	2.478 secondes

Figure 49: La carte du test pour une grande région de partitionnement

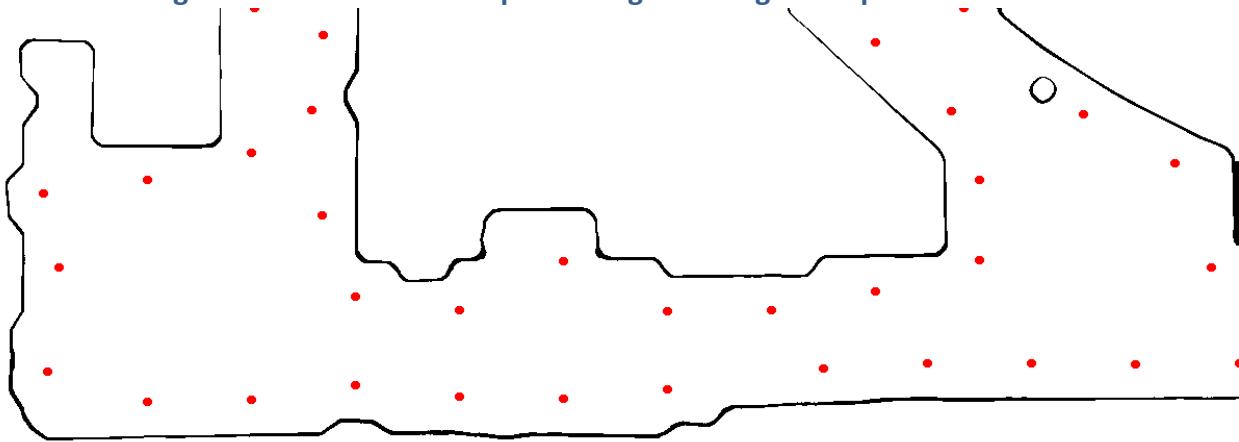


Tableau 14: Les paramètres du test pour une grande région de partitionnement

Hauteur de la région de partitionnement	1.25 mètres
Largeur de la région de partitionnement	1.25 mètres
Distance minimale entre chaque objectif	1.768 mètres
Temps d'exécution minimal	0.463 seconde
Temps d'exécution maximal	0.552 seconde
Temps d'exécution moyen	0.512 seconde

Dans le cas de ce paramètre, le temps d'exécution est proportionnel à la valeur du paramètre. Par contre, comme dans le cas du filtre médian, une faible quantité d'objectifs entraînera une numérisation de basse qualité. L'objectif du projet stipule que chaque numérisation doit avoir une partie commune avec au moins une autre pour pouvoir générer une reconstruction 3D. Il est donc préférable d'opter pour un plus grand nombre d'objectifs. Évidemment, un trop grand nombre d'objectifs n'est pas nécessairement utile donc, un juste milieu doit être trouvé.

Variation de la distance minimale et maximale des murs

Ce quatrième et dernier test fait varier la distance minimale et maximale par rapport aux murs. Comme le test de variation de la taille des blocs de partitionnement, ce test affecte le nombre d'objectifs générés. En faisant varier ce paramètre, les décisions prises dans le module d'intelligence artificielle affectent la capacité du robot à s'aventurer dans des espaces plus ou moins étroits. De plus, en étant plus loin des murs, un plus gros champ de vision peut être atteint nécessitant ainsi moins d'objectifs à visiter.

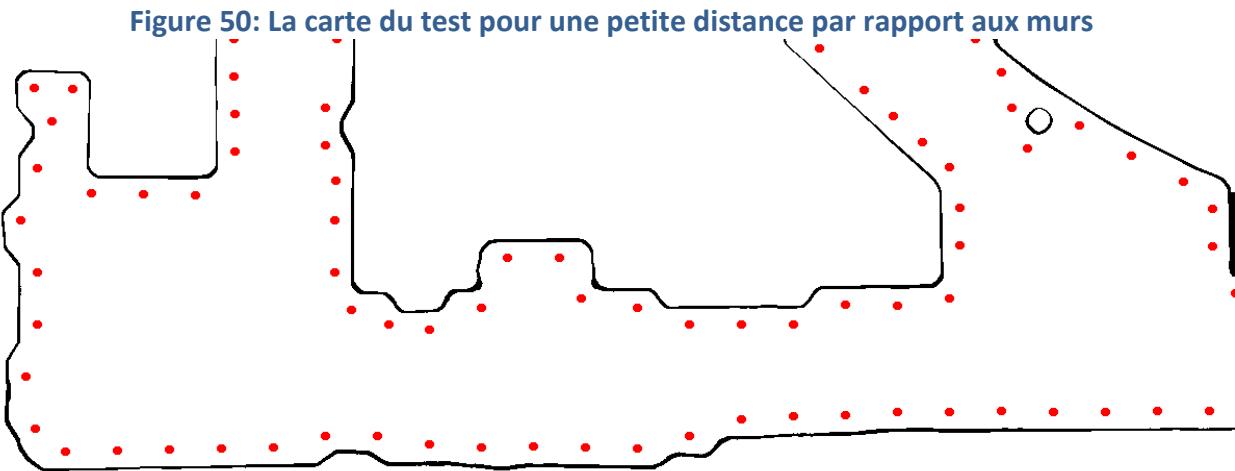


Tableau 15: Les paramètres du test pour une petite distance par rapport aux murs

Distance minimale des murs	0.4 mètre
Distance maximale des murs	0.45 mètre
Temps d'exécution minimal	1 seconde
Temps d'exécution maximal	1.122 secondes
Temps d'exécution moyen	1.056 secondes

Figure 51: La carte du test pour une grande distance par rapport aux murs

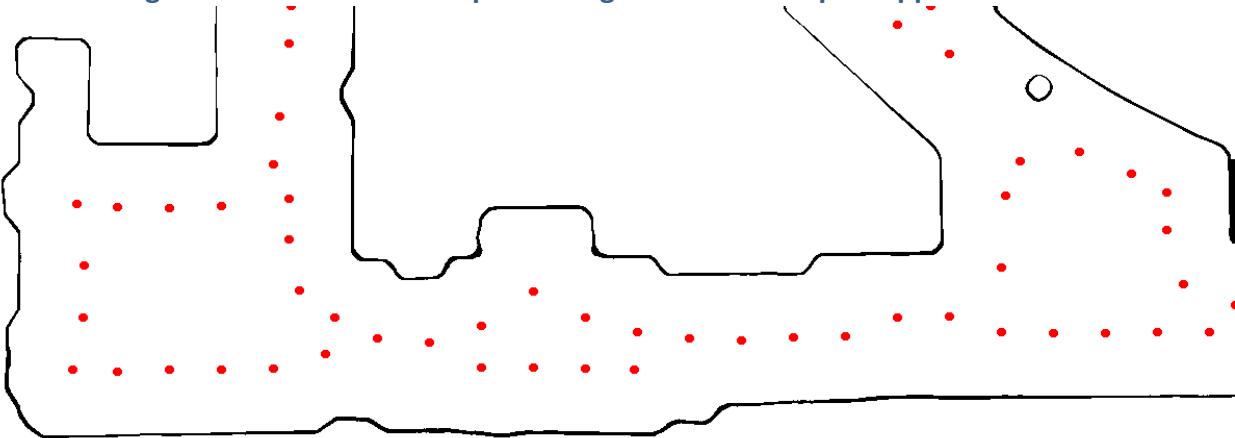


Tableau 16: Les paramètres du test pour une grande distance par rapport aux murs

Hauteur de la région de partitionnement	1.5 mètres
Largeur de la région de partitionnement	1.55 mètres
Temps d'exécution minimal	0.636 seconde
Temps d'exécution maximal	0.770 seconde
Temps d'exécution moyen	0.697 seconde

Comme le test de variation de la taille des blocs de partitionnement, la valeur de ce paramètre semble être proportionnelle au temps d'exécution de l'algorithme. Ce paramètre doit être ajusté en fonction de l'environnement à numériser. Si l'environnement est simple, la distance par rapport aux murs peut être plus grande. Par contre, un environnement complexe comme celui du test peut contenir des objets nécessitant une numérisation plus proche. Par exemple, dans la figure 51, la colonne en haut à droite ne semble pas être numérisée, car l'algorithme s'assure de garder la même distance par rapport aux murs pendant tout son trajet.

Recherche du plus court chemin

Trois algorithmes ont été testés lors du développement du module d'intelligence artificielle.

Comme mentionné dans leurs sections respectives, chacun a ses avantages et ses inconvénients.

Figure 52: Carte représentant l'algorithme Minimum Spanning Tree

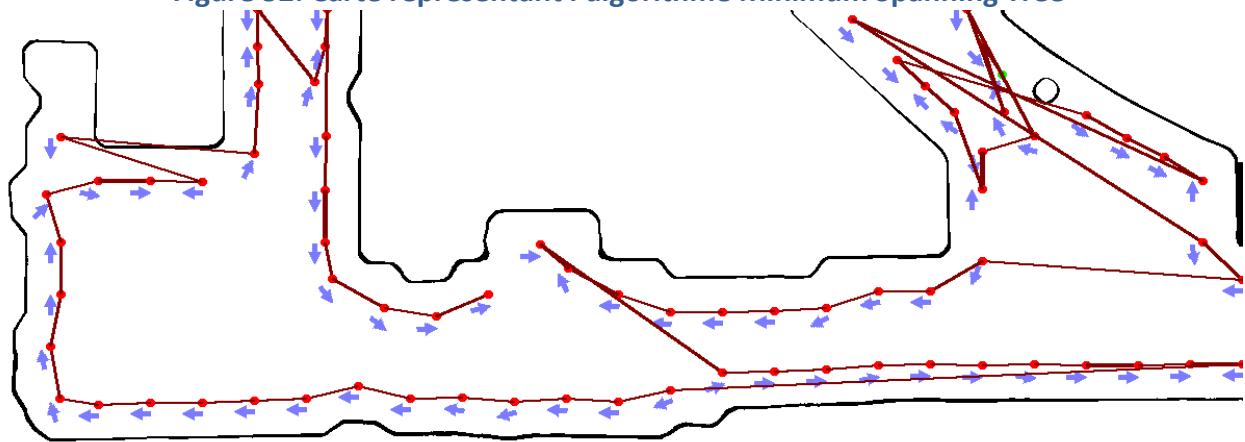


Tableau 17: Métriques associées à l'algorithme Minimum Spanning Tree

Distance parcourue maximale	215.951 mètres
Distance parcourue minimale	121.059 mètres
Distance parcourue moyenne	164.026 mètres
Temps d'exécution minimal	6.32 secondes
Temps d'exécution maximal	6.708 secondes
Temps d'exécution moyen	6.504 secondes

Figure 53: Carte représentant l'algorithme de Prim

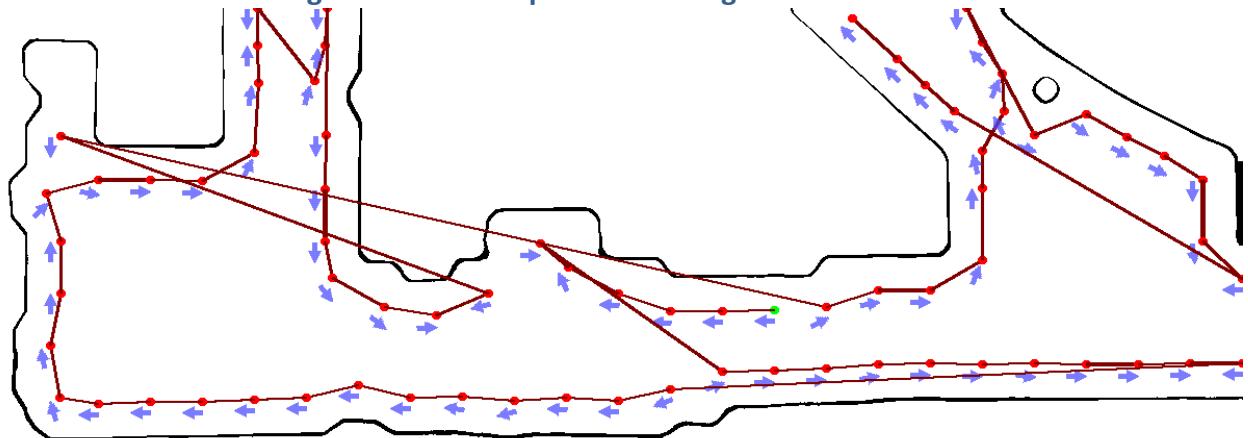


Tableau 18: Métriques associées à l'algorithme de Prim

Distance parcourue maximale	142.949 mètres
Distance parcourue minimale	130.321 mètres
Distance parcourue moyenne	137.497 mètres
Temps d'exécution minimal	5.066 secondes
Temps d'exécution maximal	5.281 secondes
Temps d'exécution moyen	5.18 secondes

Figure 54: Carte représentant l'algorithme Double Greedy Shortest Path

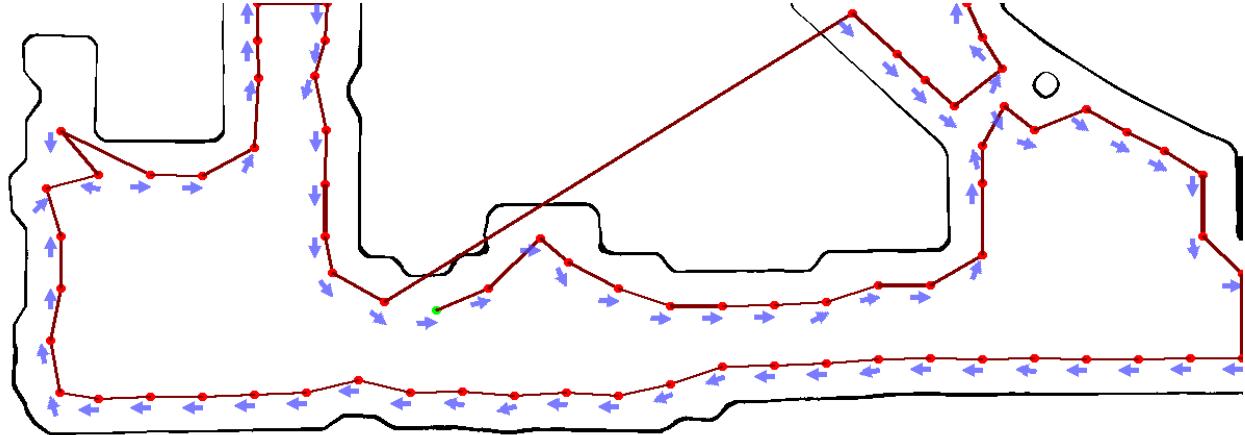


Tableau 19: Métriques associées à l'algorithme Double Greedy Shortest Path

Distance parcourue maximale	102.433 mètres
Distance parcourue minimale	89.256 mètres
Distance parcourue moyenne	95.815 mètres
Temps d'exécution minimal	5.367 secondes
Temps d'exécution maximal	5.643 secondes
Temps d'exécution moyen	5.489 secondes

Figure 55: Carte représentant l'algorithme Double Greedy Shortest Path par force brute

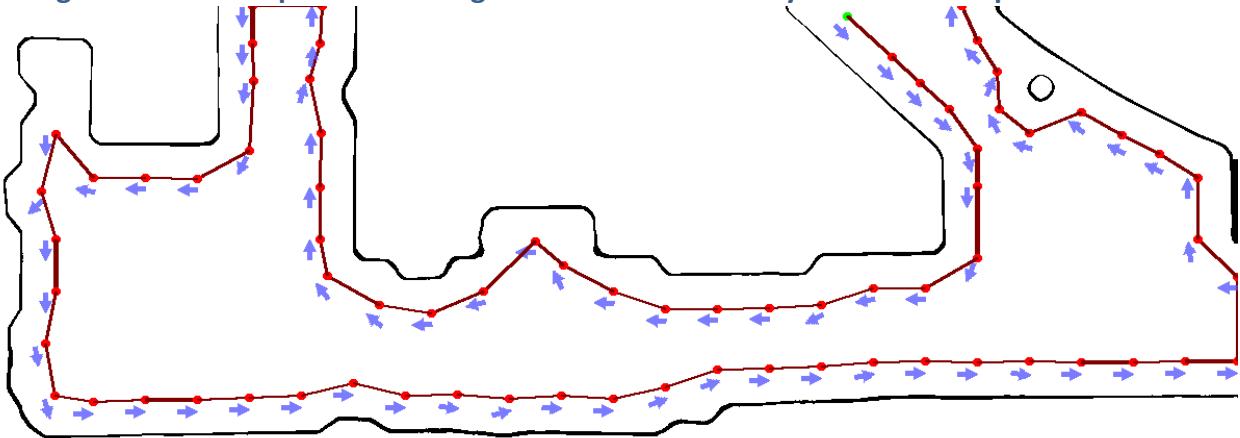


Tableau 20: Métriques associées à l'algorithme Double Greedy Shortest Path par force brute

Distance parcourue optimale	89.256 mètres
Temps d'exécution minimal	7.969 secondes
Temps d'exécution maximal	8.256 secondes
Temps d'exécution moyen	8.115 secondes

Chaque algorithme présente certains avantages sur différents niveaux comme la modifacibilité de l'algorithme et la performance de l'algorithme. Dans le contexte du projet, seule la performance des algorithmes est analysée.

L'algorithme DGSP est sans aucun doute celui offrant la meilleure performance. Il donne une distance se rapprochant de la distance optimale dans la plupart des cas pour un temps d'exécution moyen légèrement supérieur au temps d'exécution moyen de l'algorithme de Prim. De plus, pour une faible quantité d'objectifs, il est intéressant d'utiliser la force brute et DGSP pour trouver le chemin optimal à chaque exécution.

Cas particuliers

Certains environnements sont complexes et peuvent poser un problème au module d'intelligence artificielle. Deux cas particuliers ont été retenus et des tests empiriques ont été rédigés pour s'assurer que ces derniers sont bien traités.

Cas de l'îlot

Dans certains cas, l'environnement peut contenir des parties de murs qui ne sont pas connectés aux autres. L'algorithme de recherche du chemin le plus court doit s'assurer de visiter tous les objectifs générés, même si ceux-ci ne sont pas connectés à première vue.

Figure 56: Cas de l'îlot

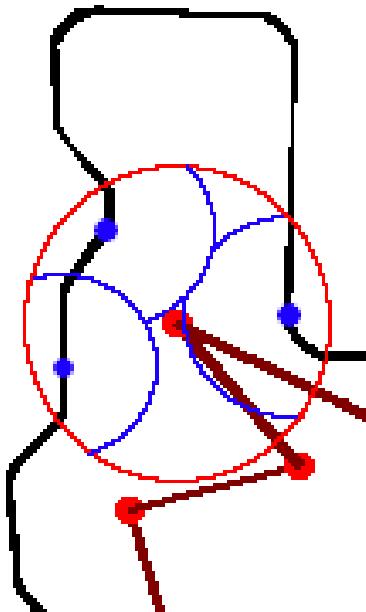


L'algorithme du plus court chemin s'assure de visiter tous les îlots. Ce ne serait pas possible avec un algorithme de base longeant un seul mur.

Cas des numérisations multiples

Certains objectifs se trouvent à une distance optimale de plusieurs murs n'étant pas nécessairement connectés. Pour se faire, plusieurs numérisations doivent être effectuées à partir d'une seule position.

Figure 57: Cas des numérisations multiples



La figure 57 illustre 3 numérisations différentes pour un seul objectif. Pour chaque point de numérisation trouvé sur le mur le plus proche, les positions se trouvant à proximité de ces dernières seront éliminées. De cette manière, plusieurs angles de numérisation peuvent être récupérés.

La figure 57 illustre également la surface éliminée (représentée par un rayon bleu) à chaque point de numérisation trouvé.

Numérisation 3D

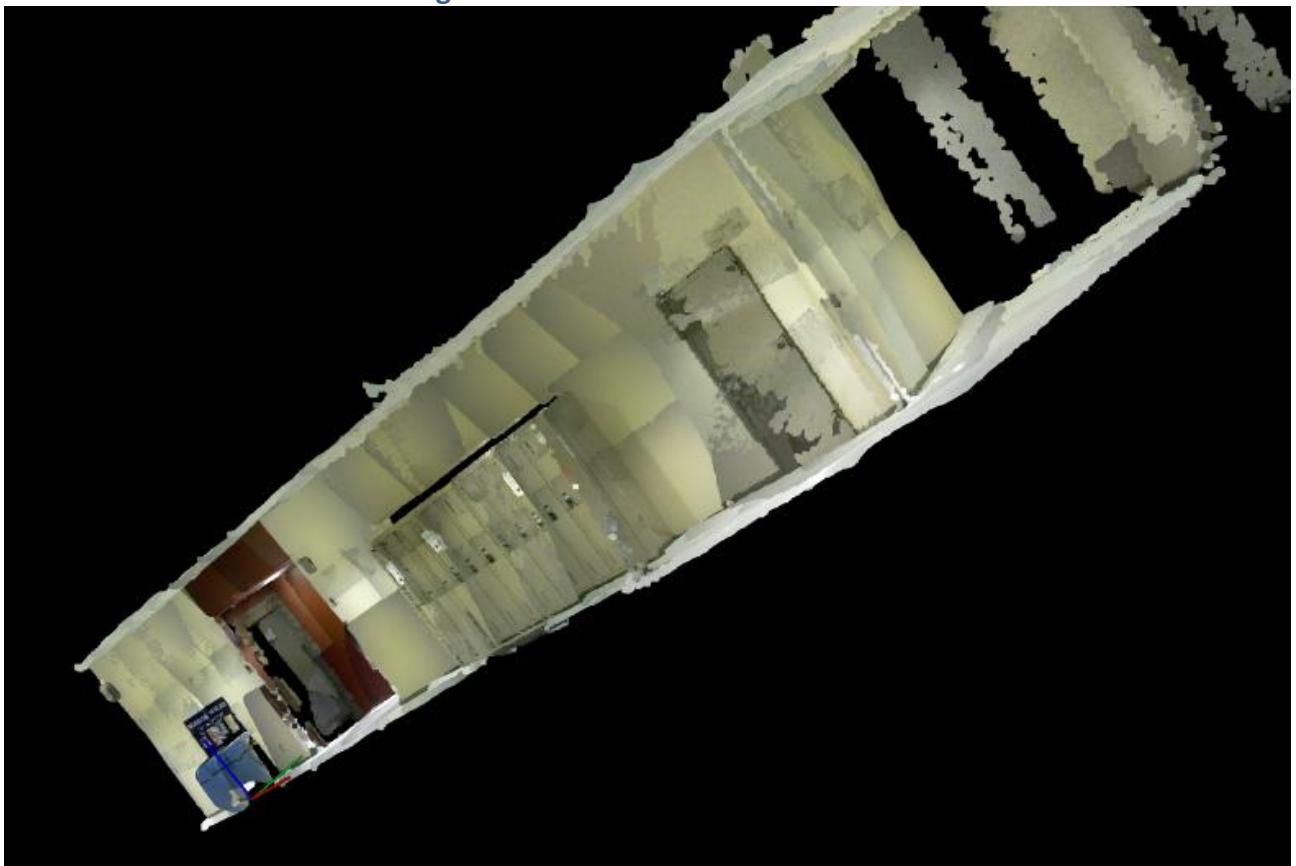
La reconstruction 3D représente un grand défi. Il y a beaucoup d'algorithmes de lissage, de débruitage et de reconstruction à tester. Durant ce projet, seulement quelques uns de ces algorithmes ont été explorés. Malgré tous les problèmes rencontrés durant ce projet, finalement une bonne reconstruction 3D a pu être faite. Pour s'assurer que la reconstruction est représentative de la réalité, il y a deux étapes principales. D'abord, il faut s'assurer que visuellement la reconstruction 3D soit cohérente. Ensuite, il faut établir des métriques sur les dimensions du nuage de points final. Dans cette section, ces deux étapes vont être détaillées.

La première étape de validation est une inspection visuelle de la reconstruction. Il faut regarder les plans des murs correspondants entre eux. Il faut aussi vérifier si les couleurs des éléments sont bien juxtaposées. Il y a des éléments colorés dans les nuages de points comme des affiches ou des poubelles qui peuvent être utilisés pour valider si une jonction entre deux ou plusieurs numérisations sont valides. Dans le cadre de ce projet, cette validation permettait de faire une première validation sur le rendu final.

La deuxième étape de validation est une inspection des métriques du nuage de points finaux. Avec PCL, il y a des métriques qui peuvent être révélées. Le volume total du rendu peut être trouvé. Cependant, les étapes ultérieures à la reconstruction ont créé une carte en deux dimensions de la pièce numérisée. Cette carte peut être comparée au rendu final pour une validation supplémentaire. Cela n'a pas été fait dans le cadre du projet, mais cela a été pensé. Pour comparer la carte avec le rendu 3D, il aurait fallu faire une tranche de la base de la reconstruction 3D et juxtaposer cette tranche à la map. Ensuite, en faisant la moyenne de distance minimale au carré entre les points (Mean Square Error) de la tranche et les points de la carte, il y aurait eu une métrique de validation intéressante. Cependant, comme le carte et le rendu 3d ne sont pas dans le même format et la même orientation, cette métrique n'est malheureusement pas présente dans le projet.

Malgré le manque de métrique, la validation visuelle des reconstructions est prometteuse. Les rendus 3D ont visuellement une bonne juxtaposition des numérisations, sans artefacts de réflexion et sans bruit apparent. Par contre, les différences d'éclairage et de couleur entre les numérisations ne sont pas encore nettoyées dans le rendu final. Au final, la géométrie de reconstruction est logique, mais la différence de teintes de couleur aux jonctions est apparente et diminue l'aspect esthétique du rendu final.

Figure 58: Reconstruction finale



CONCLUSION

En conclusion, le projet AIMBot a su mettre en place une bonne base pour effectuer des reconstructions 3D d'environnements intérieurs. Les résultats obtenus répondent à la plupart des objectifs énoncés en début de projet. Néanmoins, les objectifs établis en phase de début de projet étaient très ambitieux et il a fallu les rectifier pour obtenir des résultats plus réalistes.

Plusieurs considérations ont pu être rédigées par rapport à chaque partie du projet. De plus, certains projets futurs découlant de ces considérations ont pu être établis.

Considérations

Malgré le fait que les objectifs du projet ont été atteints, plusieurs facettes du projet AIMBot peuvent être améliorées.

Premièrement, pour ce qui est de l'architecture logicielle et du fonctionnement du robot, la plus grande amélioration serait de faire l'acquisition d'un IMU pouvant bien fonctionner dans un environnement intérieur. De cette manière, la position du robot serait beaucoup plus précise.

De plus, l'utilisation de la caméra pour détecter les obstacles en parallèle avec le LiDAR pourrait assister le robot dans les environnements complexes. Par exemple, dans un environnement contenant des tables, le robot, dans son état actuel, ne détecterait que leurs pattes et pourrait risquer une collision. Ceci est dû au fait que le LiDAR ne fait qu'un balayage horizontal. Une représentation tridimensionnelle est nécessaire pour pouvoir éviter les obstacles de manière optimale.

Deuxièmement, dans le contexte du module d'intelligence artificielle, une amélioration essentielle serait de pouvoir s'orienter et numériser un environnement inconnu. De cette

manière, le robot n'aurait pas à préalablement connaître son environnement. Le processus global serait plus rapide et pourrait s'adapter à n'importe quel environnement.

De plus, la performance des algorithmes développés pourrait être améliorée. Certains algorithmes pourraient être facilement parallélisés pour tirer avantage des multiples coeurs du CPU du portable utilisé.

Finalement, le processus de numérisation 3D pourrait aussi être amélioré. Les couleurs du rendu final ne reflètent pas la réalité, donc, il faudrait trouver une méthode pour les ajuster.

Comme le module d'intelligence artificielle, les algorithmes développés dans le cadre du contexte de la numérisation 3D pourraient avoir un temps d'exécution plus rapide.

De plus, une amélioration majeure serait de pouvoir reconstruire un environnement 3D avec des nuages de points non reliés.

BIBLIOGRAPHIE

Architecture

- ROS. «About ROS», [En ligne]. <http://www.ros.org/about-ros/> (Page consultée le 16 juillet 2014)
- R. Champagne. «Les styles architecturaux de la famille Component & Connector Component & Connector», LOG430 - Architecture logicielle, Montréal, École de technologie supérieure, 2012

Localisation et cartographie

Canonical Scan Matcher

- Censi. «CSM», [En ligne]. <http://censi.mit.edu/software/csm/> (Page consultée le 3 juillet 2014)
- Censi «An ICP variant using a point-to-line metric», *Proc. IEEE Int. Conf. Robot. Autom.*, pp.19 -25 2008

Filtre de Kalman étendu

- ROS. «robot_pose_ekf», [En ligne]. http://wiki.ros.org/robot_pose_ekf (Page consultée le 2 juin 2014)

GMapping

- ROS. «gmapping», [En ligne]. <http://wiki.ros.org/gmapping> (Page consultée le 6 juin 2014)
- OpenSlam. «GMapping», [En ligne]. <https://www.Openslam.org/gmapping.html> (Page consultée le 6 juin 2014)
- C. Stachniss, W. Burgard. «SLAM - Getting it Working in Real World Applications», [En ligne]. IROS'05: International Conference on Intelligent Robots and Systems; 2 au 6 août 2005; Edmonton, Canada. <http://www2.informatik.uni-freiburg.de/~stachnis/rbpf-tutorial/iros05tutorial-gridrbpf-handout.pdf> (Consulté le 11 août 2014)

- G. Grisetti , C. Stachniss and W. Burgard «Improved techniques for grid mapping with Rao-Blackwellized particle filters», *IEEE Trans. Robot.*, vol. 23, no. 1, pp.34 -46 2007

Hector mapping

- ROS. «hector_slam», [En ligne]. http://wiki.ros.org/hector_slam (Page consultée le 13 juillet 2014)
- S. Kohlbrecher, J. Meyer, O. von Stryk and U. Klingauf, «A Flexible and Scalable SLAM System with Full 3D Motion Estimation», In Proc. of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR'2011), 50-55, Kyoto, Japan, Nov. 1-5, 2011.

Localisation adaptative de Monte-Carlo

- ROS. «amcl», [En ligne]. <http://wiki.ros.org/amcl> (Page consultée le 26 juillet 2014)
- F. Dellaert, D. Fox, W. Burgard, and S. Thrun. «Monte Carlo localization for mobile robots», ICRA-99.

Navigation

- ROS. «navigation», [En ligne]. <http://wiki.ros.org/navigation> (Page consultée le 20 juillet 2014)
- ROS. «move_base», [En ligne]. http://wiki.ros.org/move_base (Page consultée le 20 juillet 2014)
- ROS. «base_local_planner», [En ligne]. http://wiki.ros.org/base_local_planner (Page consultée le 20 juillet 2014)

Intelligence artificielle

- P. Cardinal, «Filtrage Spatial Non-Linéaire», GTI410 - Applications des techniques numériques en graphisme et imagerie, Montréal, École de technologie supérieure, 2012
- R. Fisher, S. Perkins, A. Walker and E. Wolfart, «Distance Transform», [En Ligne]. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm> (Page consultée le 28 juin 2014)

- R. Lawlor, «Prim's MST Algorithm», [En Ligne].
http://www.comp.dit.ie/rlawlor/Alg_DS/MST/Prim.pdf (Page consultée le 30 juin 2014)
- R. Bin Muhammad, «The Traveling Salesperson Problem», [En ligne].
<http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/AproxAlgor/TSP/tsp.htm> (Page consultée le 10 juillet 2014)
- P. Norvig, «The Traveling Salesperson Problem», [En ligne].
<http://nbviewer.ipython.org/url/norvig.com/ipython/TSPv3.ipynb> (Page consultée le 15 juillet 2014)

Reconstruction 3D

- Wikipédia.org, «k-d tree», [En ligne]. http://en.wikipedia.org/wiki/K-d_tree (Page consultée le 1 août 2014)
- PCL, «The PCL Registration API», [En ligne].
http://pointclouds.org/documentation/tutorials/registration_api.php#registration-api (Page consultée le 1 juillet 2014)
- PCL, «How to use iterative closest point», [En ligne].
http://pointclouds.org/documentation/tutorials/iterative_closest_point.php#iterative-closest-point (Consulté le 10 juillet 2014)
- Segal, D. Haehnel, S. thrun, «Generalized-ICP», [En ligne].
<http://www.roboticsproceedings.org/rss05/p21.pdf> (Page consultée le 20 juillet 2014)
- PCL, «Smoothing and normal estimation based on polynomial reconstruction», [En ligne]. <http://pointclouds.org/documentation/tutorials/resampling.php#moving-least-squares> (Page consulté le 28 juillet 2014)
- Wikipédia.org, «Luminance (Relative)», [En ligne].
[http://en.wikipedia.org/wiki/Luminance_\(relative\)](http://en.wikipedia.org/wiki/Luminance_(relative)) (Page consulté le 27 juillet 2014)
- RapidTables, «RGB to HSV conversion», [En ligne].
<http://www.rapidtables.com/convert/color/rgb-to-hsv.htm> (Page consulté le 15 juillet 2014)

RÉFÉRENCES

Document de vision

Voir Document_Vision.pdf

Document d'architecture

Voir Document_Architecture.pdf

Plan de tests

Voir Artefact_Plan_Tests.pdf

Manuel utilisateur

Voir Manuel_Utilisateur.pdf

Vidéos de démonstration

Reconstruction 3D

<https://www.youtube.com/watch?v=LbGPCin2QWo>

Génération de carte 2D

<https://www.youtube.com/watch?v=XtmBRzJQYuA>

Navigation avec évitement d'obstacle

<https://www.youtube.com/watch?v=8uoZFwVkeRU>

Intelligence artificielle en action

<https://www.youtube.com/watch?v=3I8mYHHksak>

Code source

<https://github.com/clubcapra>

<https://github.com/h017>