Master Systèmes Dynamiques et Signaux

Mémoire de Master

Approche ensembliste pour l'estimation d'une vérité terrain : application aux problèmes de localisation et cartographie en robotique mobile

Auteur :
M. Clément MACADRÉ

Encadrant :
Dr. Rémy GUYONNEAU

Président :
Pr. Laurent HARDOUIN
Jury :
Pr. Anne HEURTIER
Dr. Remy GUYONNEAU
Dr. Mirvana HILAL
Dr. Sébastien LAGRANGE
Dr. Marie-Lise PANNIER

Version du 7 juillet 2022

Table des matières

Liste des acronymes

ATE	*Absolute Trajectory Error / Erreur absolue de trajectoire*
Cerema	*Centre d'études et d'expertise sur les risques, l'environnement, la mobilité et l'aménagement*
CSP	*Constraint Satisfaction Problem / Problème de satisfaction des contraintes*
IAL	*Interval Analysis Localization / Localisation par analyse par intervalles*
IAPT	*Interval Analysis Pose Tracking / Suivi de posture par analyse par intervalles*
Imu	*Inertial measurement unit / Centrale inertielle*
Laris	*Laboratoire Angevin de Recherche en Ingénierie des Systèmes*
Lidar	*Laser imaging detection and ranging / Détection et estimation de la distance par laser*
PCL	*Point Cloud Library*
RANSAC	*RANdom SAmple Consensus*
RoMuLux	*RObot MesUrant des LUX*
ROS	*Robotic Operating System*
RPE	*Relative Pose Error / Erreur de posture relative*
SLAM	*Simultaneous Localization and Mapping / Cartographie et localisation simultanées*
voxel	*Volume + pixel*

Liste des notations

Problème de localisation

r	:	Robot
k	:	Temps discret
$\mathbf{x}_k = (x_{1_k}, x_{2_k})$	:	Position du robot r à l'instant k
θ_k	:	Orientation (direction) du robot r à l'instant k
$\mathbf{q}(k) = (x_1(k), x_2(k), \theta(k))$	:	Posture (i.e. position + orientation) du robot r à l'instant k
$\mathbf{u}_k$	:	Vecteur de contrôle appliqué au robot r entre les instant k et $k+1$
$\Delta_{\mathbf{x}_k}$	:	Distance parcourue entre $\mathbf{x}_k$ et $\mathbf{x}_{k+1}$
Δ_{θ_k}	:	Différence d'orientation entre θ_k et θ_{k+1}
$\max_{0<k<\infty}(\Delta_{\mathbf{x}_k})$	:	Distance maximale parcourue entre $\mathbf{x}_k$ et $\mathbf{x}_{k+1}$ pour $k \in \{0, \infty\}$
$\max_{0<k<\infty}(\Delta_{\theta_k})$	:	Différence d'orientation maximale entre θ_k et θ_{k+1} pour $k \in \{0, \infty\}$
$\varepsilon(k)$	:	Carte de l'environnement à l'instant k
$\mathbf{w}_{i,k}$	:	Position du $i^{\text{ème}}$ amers à l'instant k
$y_{i,k}$	:	$i^{\text{ème}}$ mesure de distance à l'instant k
$\gamma_{i,k}$	:	$i^{\text{ème}}$ mesure d'angle à l'instant k

Analyse par Intervalles

$\cap$	:	Intersection
$\cup$	:	Union
$\mathbb{X} \times \mathbb{Y}$	:	Produit Cartésien
$\mathbb{X} \setminus \mathbb{Y}$	:	$\{x \mid (x \in \mathbb{X}) \wedge (x \notin \mathbb{Y})\}$
$[\cdot]$	:	Enveloppe intervalle
$[x]$	:	Intervalle
$\overline{x}$	:	Borne supérieure de $[x]$
$\underline{x}$	:	Borne inférieure de $[x]$
$[\mathbf{x}]$	:	Vecteur d'intervalles

Chapitre 1

Présentation du sujet

1.1 Le paradoxe de la cartographie et de la localisation simultanées en robotique mobile

Dans le but de mettre au point des robots autonomes, il est nécessaire de déterminer la posture d'un robot dans son environnement [6]. Étant donné que les robots ne disposent d'aucune information sur leurs alentours, ils doivent être capables de construire une carte de l'environnement pendant leur déplacement et d'estimer correctement leur posture dans cette carte. La cartographie consiste à obtenir un modèle de l'environnement du robot, et la localisation consiste à estimer la posture du robot dans la carte obtenue. Pour construire la carte, nous devons connaître l'emplacement du robot et pour la localisation, nous avons besoin de la carte (paradoxe de l'œuf et de la poule).

Ainsi, en robotique mobile, nous chercherons donc à développer une solution validant les points suivants :

- Créer une carte pertinente d'un environnement complexe (assemblage de sous-espace, tenir compte des distorsions dues aux mouvements du capteur, etc.)
- Situer le robot dans cette carte en parallèle de sa création
- Lors de l'exploration, éviter de rajouter un espace déjà exploré à la carte globale
- Gérer les situations de "kidnapping" [13]
- Etc.

Savoir positionner un robot dans son environnement est une étape essentielle à la navigation autonome, pour résoudre cette problématique, il existe donc des algorithmes que nous allons maintenant présenter.

1.2 Les algorithmes de cartographie et de localisation simultanées

Un algorithme de SLAM (Simultaneous Localisation And Mapping ou localisation et cartographie simultanées en français), a pour but de construire une carte de l'environnement dans lequel évolue le robot et de pouvoir fournir ses coordonnées spatiales simultanément [3]. L'état du véhicule peut être défini différemment selon l'application : position et orientation 2D, 6D, vitesse, accélération, etc.

Ainsi, un algorithme de SLAM [1] doit être capable d'estimer à chaque instant k la posture du robot $\mathbf{q}(k)$ (position et orientation) ainsi que la carte de l'environnement $\varepsilon(k)$ en se fiant aux mesures $\mathbf{y}(k)$ issues d'un capteur (e.g. GPS, Imu [2], caméra, Lidar [3], sonar, etc.) et au vecteur de commande $\mathbf{u}(k)$. L'objectif d'un algorithme de SLAM sera donc d'estimer l'expression suivante :

$$P(\varepsilon(k+1),\mathbf{q}(k+1)\,|\,\mathbf{y}(1:k+1),\mathbf{u}(1:k)) \tag{1.1}$$

Pour résoudre le paradoxe de l'œuf et de la poule entre la cartographie et la posture du robot, nous ferons appel aux algorithmes SLAM. Ces algorithme règlent ce paradoxe en alternant les mises à jour des deux estimations 1.2 et 1.3.

$$P(\mathbf{q}(k)\,|\,\mathbf{y}(1:k),\mathbf{u}(1:k),\varepsilon(k)) \tag{1.2}$$

$$P(\varepsilon(k)\,|\,\mathbf{q}(k),\mathbf{y}(1:k),\mathbf{u}(1:k)) \tag{1.3}$$

D'après l'article [23] il existe de nombreuses approches pour réaliser une localisation et cartographie simultanées. Citons quelques exemples en se basant sur la banque d'algorithme open source openslam.org :

- **Gmapping** utilise un filtre particulaire Rao-Blackwellien pour créer une grille d'occupation à partir de données de distance laser [8]
- **CEKF-SLAM** est un algorithme basé sur un filtre de Kalman étendu compressé [10]
- **FALKO** repose sur la détection de points d'intérêt dans des mesures laser [18]
- **TORO** résout le problème de SLAM à l'aide d'un graphe en appliquant une descente de gradient stochastique pour minimiser les erreurs introduites par des contraintes [9]
- etc.

1.3 Pourquoi évaluer la précision d'un algorithme de SLAM

Le robot perçoit son environnement à l'aide de capteurs (e.g. caméra, Lidar, capteur à ultrasons, etc.). Ces mesures sont affectées par une incertitude propre aux équipements utilisés [23]. De plus, les algorithmes de SLAM travaillent avec ces données biaisées et ne peuvent donc offrir qu'un positionnement approximatif. En général, ces algorithmes prévoient une stratégie pour éviter que de petites erreurs cumulées produisent une dérive croissante de l'estimation de posture à travers le temps. Malgré tout, des erreurs subsistent et il est intéressant de pouvoir juger de la fiabilité des prédictions de localisation d'un algorithme de SLAM. Dans ce sujet de recherche, nous chercherons à comparer la trajectoire calculée par un algorithme à une vérité-terrain [21] (un relevé qui se veut exact auquel nous confrontons une estimation). L'obtention de cette vérité-terrain est la source des difficultés liées à l'évaluation de la précision d'un algorithme de SLAM. En effet, l'acquisition d'une vérité terrain est forcément affectée par une incertitude plus ou moins grande liée aux mesures. Il

1. *Simultaneous Localization and Mapping / Cartographie et localisation simultanées*
2. *Inertial measurement unit / Centrale inertielle*
3. *Laser imaging detection and ranging / Détection et estimation de la distance par laser*

existe donc un besoin de mettre au point une méthode permettant d'estimer et de borner les erreurs effectués par un algorithme de SLAM lors de l'estimation d'une trajectoire.

1.4 Présentation du sujet de recherche

Ce sujet de recherche est mené sur la base d'expérimentations réalisées à l'aide d'une simulation du robot RoMuLux (RObot MesUrant des LUX). Le projet RoMuLux [4] porté par le Cerema [5] a pour but de faciliter les mesures d'éclairage dans les établissements recevant du public soumis à des seuils normatifs. Le robot est équipé d'un Lidar "Velodyne Puck" [6] afin de fournir aux algorithmes de SLAM les mesures nécessaires à la localisation du robot. Trois algorithmes ont été implémentés dans ce système : Cartographer [14], hdl graph slam [20] et LeGO-LOAM [27]. Le robot est aussi équipé d'un luxmètre afin de mesurer l'éclairement à une certaine position. Sachant que le résultat final est une carte du bâtiment interpolée avec les mesures d'éclairage relevées, il est nécessaire de pouvoir évaluer la précision des positions fournies par les différents algorithmes de SLAM. Ce sujet de recherche vise donc à fournir une vérité terrain garantie, basée sur l'analyse par intervalles dans un contexte d'erreurs bornées, uniquement à partir de donnée Lidar. Cette vérité terrain sera ensuite utile pour évaluer la précision d'algorithmes de SLAM à l'aide de métriques originales.

Pour commencer ce rapport, nous présenterons dans le chapitre 2 une synthèse des approches traditionnelles d'évaluation d'algorithmes de SLAM [21], ainsi que certaines métriques de comparaison de performances communément utilisées [25]. Dans le chapitre 3, nous formulerons l'approche théorique nécessaire à l'obtention d'une vérité terrain dans le contexte de ce sujet de recherche. Nous détaillerons ensuite l'application des méthodes proposées dans le chapitre 4. Pour finir, nous commenterons les résultats de ce projet dans le chapitre 5.

4. Présentation du projet RoMuLux
5. *Centre d'études et d'expertise sur les risques, l'environnement, la mobilité et l'aménagement*
6. Velodyne Puck Data Sheet

Chapitre 2

Synthèse Bibliographique

Dans ce chapitre, nous présenterons l'approche et les métriques courantes utilisées pour évaluer les performances d'un algorithme de SLAM. Dans un second temps, nous présenterons l'analyse par intervalles ainsi que les CSP [1].

2.1 Évaluation d'un algorithme de SLAM

2.1.1 Approches conventionnelles

Les approches habituelles pour évaluer les performances d'un algorithme de SLAM se rangent en général en trois catégories [25]. Premièrement, l'approche compétitive où des systèmes robotiques s'affrontent dans le cadre d'un scénario défini. Par exemple le "DARPA Robotics Challenge" [2] où plusieurs équipes ont eu pour mission de développer des robots terrestres semi-autonomes pouvant accomplir des "tâches complexes dans des environnements dangereux, dégradés et construits par l'homme". Dans ce contexte, l'évaluation des performances se fait en fonction de la capacité qu'a le système à réaliser certains travaux. Deuxième approche, l'utilisation de collections d'ensembles de données accessibles au public qui sont fournies pour comparer les algorithmes sur des problèmes spécifiques. Ces dépôts de données fournissent en général une vérité terrain pour pouvoir évaluer la précision de l'algorithme utilisé. Troisième approche, le recours à des métriques et des méthodologies issues de publications scientifiques, par exemple, une évaluation via une simulation [1].

La procédure privilégiée dans notre cas de figure [2] (SLAM basé sur des relevés Lidar) est d'avoir recours à ces ensembles de données généralement appelés "benchmark" (français : étalon ou repère). Un benchmark est un point de référence servant à effectuer une mesure. Ainsi, détaillons la composition d'un benchmark connu issue de la publication "Towards a benchmark for RGB-D SLAM evaluation" [28]. Ce dépôt propose un grand ensemble de séquences d'images RGB-Depth, qui est une donnée issue d'une caméra comme un capteur Microsoft Kinect par exemple. Dans ce dépôt, les images sont accompagnées des trajectoires exactes de la caméra (i.e. la vérité terrain). La vérité terrain sert de référence pour l'évaluation des algorithmes de SLAM. Ainsi, un utilisateur cherchant

1. *Constraint Satisfaction Problem / Problème de satisfaction des contraintes*
2. DARPA Robotics Challenge

à tester les performances de son algorithme utilisera ce benchmark pour comparer les prédictions de localisation issue de son approche avec la vérité terrain. Notons que la vérité terrain n'est pas une donnée systématiquement fournie. Pour se convaincre qu'obtenir une vérité terrain est une démarche fastidieuse, nous trouverons dans l'annexe A.1 certaines méthodes utilisées pour la mesurer.

Nous retenons de cette section que comparer la prédiction d'un algorithme de SLAM avec une référence (la vérité terrain) peut se révéler difficile dès le moment où nous explorons un environnement complexe comme un bâtiment complet (plusieurs salles). Détaillons maintenant certaines métriques utilisées pour réaliser cette comparaison.

2.1.2 Métriques de comparaison des performances

Il existe deux approches distinctes dans l'évaluation des performances d'un algorithme de SLAM [25]. Premièrement, la comparaison globale (cf. l'équation 2.3), entre la carte générée et la vérité terrain. Celle-ci est effectuée en calculant la distance euclidienne et la différence d'angle entre la trajectoire estimée et celle de la vérité-terrain. Cette approche permet de fournir par exemple l'écart-type des erreurs, l'erreur maximale, la moyenne des erreurs, etc. Notons que la comparaison de l'erreur globale entre deux trajectoires peut ne pas être pertinente. En effet, si le robot commet une petite erreur angulaire (cf. la figure 2.1b), surtout au début du parcours, alors cette erreur sera conservée et aggravée tout au long de l'acquisition de données. Nous constatons le même problème avec une erreur de translation (cf. la figure 2.1a). Ainsi, malgré une cartographie globalement cohérente de la part de l'algorithme de SLAM nous constaterons de mauvaises performances en utilisant des métriques globales. Une autre approche [21] propose une comparaison entre les positions successives le long de la trajectoire du robot (cf. l'équation 2.5). Détaillons maintenant quelques métriques usuelles utilisées pour l'évaluation d'algorithmes de SLAM. Par la suite, nous ferons usage des coordonnées homogènes sous la forme d'une matrice M qui permet de caractériser les transformations d'un objet dans l'espace. Cette matrice comprend un vecteur de translation t et une matrice de rotation R.

$$M = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

$$trans(M) := t$$

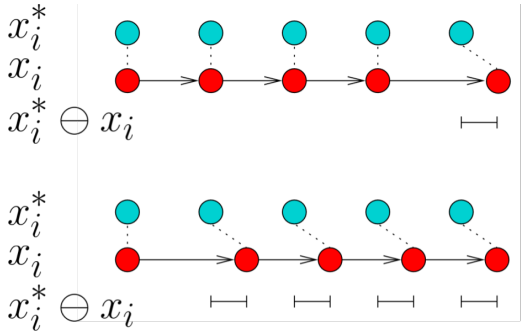
$$rot(M) := R$$

Concernant l'opération $tr(\cdot)$ de l'équation 2.1 : la trace d'une matrice carrée R est définie comme la somme de ses coefficients diagonaux :

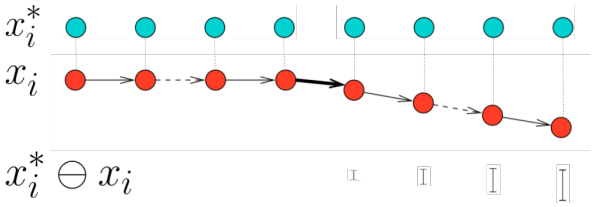
$$R = (a_{ij})_{1 \leq i,j \leq n}$$

$$tr(R) = \sum_{i=1}^{n} a_{ii}$$

Concernant l'opération $\angle(\cdot)$ de l'équation 2.1 : nous apprenons grâce à l'annexe "Appendix A : Extracting u and θ from R" de l'article [16] que cette opération permet d'extraire, à partir d'une



(a) Ce schéma présente un exemple où une comparaison d'erreur globale n'est pas pertinente. Les cercles bleus (clairs) représentent la vérité terrain tandis que les cercles rouges (foncés) représentent la position estimée par l'algorithme de SLAM. Dans la première situation, une erreur est commise à la fin du parcours résultant en une petite erreur de trajectoire globale. En revanche, si cette erreur est commise en début de parcours alors celle-ci se cumule et résulte en une plus grande erreur de trajectoire globale. (source image : [21])



(b) Autre exemple, le robot se déplace en ligne droite, mais commet une petite erreur d'orientation à mi-parcours puis continue sans commettre d'erreur supplémentaire. Avec une comparaison d'erreur globale, l'erreur de trajectoire sera croissante et ne reflétera pas vraiment les performances de l'algorithme. (source image : [4])

FIGURE 2.1 – Évaluation SLAM : Inconvénient d'une comparaison globale

matrice de rotation R, l'axe et le sens dans lequel a été effectuée la rotation ainsi que la valeur de l'angle de rotation.

$$\angle R := arccos(\frac{tr(R)-1}{2}) \tag{2.1}$$

Pour utiliser ces métriques, nous disposons d'une séquence de positions spatiales à partir de la trajectoire estimée P et de la trajectoire de vérité-terrain Q. Pour simplifier, nous supposons que les deux séquences sont synchronisées dans le temps et qu'elles ont toutes deux un même nombre d'échantillons n. En pratique, ces deux séquences ont des taux d'échantillonnage et des longueurs différents, des données potentiellement manquantes, etc. Cette méthode nécessite donc un pré-traitement des données.

Absolute Trajectory Error (ATE)

L'erreur de position globale [25] peut être évaluée en comparant les distances absolues entre la trajectoire estimée et celle de la vérité-terrain. Comme ces deux trajectoires sont mesurées dans deux systèmes de coordonnées cartésiennes différents, elles doivent d'abord être alignées. Par exemple, imaginons le cas de figure où les mesures dans le système de coordonnées d'une caméra (capture de la vérité terrain) doivent être reliées aux coordonnées d'un système attaché au robot (prédiction de position issue d'un algorithme de SLAM). Ceci peut être réalisé en utilisant la méthode de Horn [15] qui fournit l'isométrie euclidienne S. Notons qu'en géométrie, une isométrie est une transformation

qui conserve les longueurs. Définissons la matrice d'erreur E absolue de la trajectoire au moment i :

$$E_i := Q_i^{-1} S P_i \tag{2.2}$$

L'ATE [3] est définie comme l'erreur quadratique moyenne calculée à partir des matrices d'erreur E_i :

$$ATE_{rmse} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|trans(E_i)\|^2} \tag{2.3}$$

L'erreur absolue de trajectoire $\|trans(E_i)\|$ est la déviation de la trajectoire par rapport à la vérité terrain à chaque instant i.

Relative Pose Error (RPE)

L'erreur de posture relative [25] mesure la précision locale de la trajectoire entre deux prises de mesures successives. Par conséquent, l'erreur de posture relative correspond à la dérive de la trajectoire. Définissons la matrice d'erreur de posture relative au pas de temps i de la manière suivante :

$$F_i := (Q_i^{-1} Q_{i+1})^{-1} S (P_i^{-1} P_{i+1}) \tag{2.4}$$

La RPE [4] est généralement divisée en composantes de translation et de rotation. À partir d'une séquence de n poses de la caméra, nous obtenons $m = n-1$ matrices individuelles d'erreur relative de posture le long de la séquence. À l'instar de l'ATE, la composante de translation de la RPE est définie comme l'erreur quadratique moyenne pour chaque instant i :

$$RPE^i_{trans} = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \|trans(F_i)\|^2} \tag{2.5}$$

La composante de rotation de la RPE est définie comme l'erreur moyenne pour chaque instant i :

$$RPE^i_{rot} = \frac{1}{m}\sum_{i=1}^{m} \angle(rot(F_i)) \tag{2.6}$$

Pour l'évaluation des algorithmes de SLAM, il est pertinent de calculer la moyenne de la totalité des paires possibles dans les composantes de translation et de rotation [25].

En conclusion, nous retiendrons qu'il existe deux approches à l'évaluation des performances de localisation d'un algorithme de SLAM, une comparaison globale et une comparaison relative. Une approche globale sera plus critique d'une cartographie globalement cohérente. Pour ce projet de recherche, nous chercherons donc à mettre au point une métrique originale utilisant une approche relative.

3. *Absolute Trajectory Error / Erreur absolue de trajectoire*
4. *Relative Pose Error / Erreur de posture relative*

2.2 Localisation garantie via l'analyse par intervalles appliquée aux robots mobiles

Dans cette seconde partie, nous décrirons ce qu'est l'analyse par intervalles et nous détaillerons un CSP.

2.2.1 Présentation de l'analyse par intervalles

L'analyse par intervalles est un procédé mathématique prenant en compte les erreurs d'arrondis et de mesures des éléments intervenant dans un calcul. En arithmétique classique, une valeur est représentée comme un seul nombre tandis qu'en analyse par intervalles, chaque valeur est représentée par une plage de possibilités. En effet, une mesure aussi précise soit-elle sera forcément affectée par une incertitude qui se transmettra à travers les opérations jusqu'au résultat final. L'intérêt de l'analyse par intervalles est donc de pouvoir garantir un résultat fiable, dont nous sommes certains qu'il se trouve dans un intervalle final. En effet, au lieu de travailler avec un réel x, nous travaillons avec un intervalle $[a,b]=\{x\in\mathbb{R}|a\leq x\leq b\}$ comprenant x. En analyse par intervalles, l'application d'une opération mathématique $f(\cdot)$ à l'intervalle $[a,b]$ produit un intervalle $[c,d]$ comprenant les valeurs de $f(x)$ pour tout $x\in[a,b]$. Notons que des opérations classiques (e.g. $+,-,/,*$) ainsi que des fonctions (e.g. $\sqrt[n]{x},e^x,x^n,cos(x)$, etc.) peuvent être facilement appliquées à l'analyse par intervalles [24].

Nous trouverons dans l'annexe A.2, un exemple d'application de l'analyse par intervalles. Dans cet exemple, nous chercherons à calculer la position d'un obstacle à partir de la position (x_1,x_2) du robot, son orientation θ et d'une mesure y de distance issue d'un capteur.

Maintenant que nous avons appréhendé l'analyse par intervalles, attardons-nous sur un type de problème qu'il nous sera utile de résoudre dans le chapitre 4.

2.2.2 Problème de satisfaction de contraintes (CSP)

Comme son nom l'indique, ce type de problème consiste à réduire un domaine de possibilité original en ne gardant que les valeurs qui satisfont un certain nombre de contraintes. Formellement, un problème de satisfaction de contraintes est défini par trois ensembles $\langle V,D,C\rangle$, où V est un ensemble de n variables, D est un ensemble de n domaines de valeurs tels qu'un de ses éléments d_i soit le domaine d'une variable x_i de V, et C est un ensemble de contraintes.

$$\left\{\begin{array}{c} V=\{x,y,z\} \\ D=\{x\in[-\infty,2],y\in[-\infty,9],z\in[7,+\infty]\} \\ C=\{z=x+y\} \end{array}\right\} \tag{2.7}$$

La résolution d'un CSP consiste à réduire les domaines en supprimant les valeurs qui ne sont pas compatibles avec les contraintes. Le CSP 2.7 se résout efficacement à l'aide de l'analyse par intervalles [24] :

$$
\begin{aligned}
z = x + y &\Rightarrow [z] \in [z] \cap ([x] + [y]) \\
&\Rightarrow [z] \cap ([-\infty,2] + [-\infty,9]) \\
&\Rightarrow [7,+\infty] \cap ([-\infty,11]) = [7,11] \\
x = z - y &\Rightarrow [x] \in [z] \cap ([z] - [y]) \\
&\Rightarrow [x] \cap ([7,11] - [-\infty,9]) \\
&\Rightarrow [-\infty,2] \cap ([-2,\infty]) = [-2,2] \\
y = z - x &\Rightarrow [y] \in [y] \cap ([z] - [x]) \\
&\Rightarrow [y] \cap ([7,11] - [-2,2]) \\
&\Rightarrow [-\infty,9] \cap ([5,9]) = [5,9]
\end{aligned}
\tag{2.8}
$$

Avec pour solution les domaines contractés $[z]^* = [7,11]$, $[x]^* = [-2,2]$ et $[y]^* = [5,9]$.

Maintenant que nous avons abordé toutes les notions nécessaires à la compréhension de ce projet de recherche, détaillons dans le prochain chapitre l'approche théorique à l'obtention d'une vérité terrain grâce à une localisation garantie dans un contexte d'erreur bornée.

Chapitre 3

Approche Théorique

Dans ce chapitre, nous tâcherons de poser les bases théoriques nécessaires à la compréhension de l'implémentation de l'algorithme IAPT (Interval Analysis Pose Tracking). Cet algorithme est inspiré de l'algorithme IAL [1] [12] qui réalise une localisation globale du robot tandis que l'algorithme IAPT [2] se charge du suivi de posture du robot dans une carte d'amers. L'algorithme IAPT a pour but de fournir une série de boîtes (i.e. un domaine de posture possible) contenant à chaque instant et de manière garantie l'objet que nous cherchons à localiser. Cette liste de boîtes nous servira ensuite de vérité terrain pour évaluer les performances d'un algorithme de SLAM. Premièrement, nous définirons le type de robot que nous chercherons à localiser dans ce projet de recherche puis nous présenterons l'algorithme IAPT. Dans cette présentation, nous découvrirons que l'algorithme IAPT a besoin d'une carte d'amers pour localiser un robot. Ainsi, nous détaillerons dans une seconde partie le processus de création d'une telle carte. Dans un troisième temps, à l'aide de tous ces éléments, nous formaliserons le CSP 3.6 dont le résultat est une estimation garantie de la posture du robot.

3.1 Implémenter l'algorithme IAPT et obtenir une vérité terrain garantie

Notons que dans le cadre de ce projet de recherche, nous présenterons l'algorithme IAPT en prenant pour exemple son implémentation sur le robot RoMuLux [3]. Nous présenterons donc rapidement les propriétés importantes de ce robot pour ce projet.

3.1.1 Définition d'équations liées à la dynamique d'un robot type RoMuLux

Le robot RoMuLux est doté de 3 degrés de liberté (appui plan $\perp z$). D'après [19], ce système est caractérisé par les équations dynamiques à temps discret suivantes :

$$\mathbf{q}(k+1) = f(\mathbf{q}(k), \mathbf{u}(k)) \tag{3.1}$$

1. *Interval Analysis Localization / Localisation par analyse par intervalles*
2. *Interval Analysis Pose Tracking / Suivi de posture par analyse par intervalles*
3. *RObot MesUrant des LUX*

$$\mathbf{y}(k) = g_\varepsilon(\mathbf{q}(k)) \tag{3.2}$$

Concernant l'équation 3.1, la posture du robot $\mathbf{q}(k) = (x_1(k), x_2(k), \theta(k))$ est définie par sa position $\mathbf{x}(k) = (x_1(k), x_2(k))$ et son orientation $\theta(k)$ dans l'environnement, notée ε, au temps discret k. L'environnement $\varepsilon \in \mathbb{R}^2$ est un domaine bidimensionnel dans lequel le robot se déplace. La fonction f caractérise la dynamique du robot et $\mathbf{u}(k)$ correspond au vecteur de commande à l'instant k. Concernant l'équation 3.2, exprimons $\mathbf{y}(k) = (y_1(k), ..., y_n(k))$ comme le vecteur des mesures effectuées par un capteur (un Lidar dans le cas présent). Enfin, la fonction g_ε n'est pas connue, car elle dépend fortement de l'environnement, qui n'est pas connu parfaitement. Pour résumer, l'équation 3.1 nous apprend que la posture du robot au pas de temps $k+1$ dépend de sa posture actuelle et du déplacement qui sera effectué durant Δ_k. Enfin, l'équation 3.2 exprime simplement que le robot observe son environnement à l'aide de capteur (entaché d'incertitude) et que cette observation dépend de la posture du robot dans l'environnement.

Ainsi, d'après [12], nous exprimerons la dynamique du robot avec la fonction suivante :

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \\ \theta(k+1) \end{pmatrix} = \begin{pmatrix} x_1(k) \\ x_2(k) \\ \theta(k) \end{pmatrix} + \begin{pmatrix} sin(\theta(k))\Delta_{x,k} \\ cos(\theta(k))\Delta_{x,k} \\ \Delta_{\theta,k} \end{pmatrix} \tag{3.3}$$

Avec $\Delta_{x,k}$ la translation et $\Delta_{\theta,k}$ la rotation effectuée par le robot entre le temps k et $k+1$. Maintenant que nous avons défini l'espace dans lequel évolue le robot ainsi que sa dynamique de déplacement, nous pouvons présenter la façon dont l'algorithme IAPT se chargera de localiser ce robot.

3.1.2 L'algorithme IAPT

L'objectif de l'algorithme IAPT est de résoudre un problème de suivi de posture dans un contexte d'erreurs bornées. L'algorithme cherche à trouver la boîte (i.e. un intervalle 3D) $[\mathbf{q}(k)]$ contenant la posture $\mathbf{q}(k)$ du robot à chaque instant k de manière garantie :

$$\forall_k, [\mathbf{q}(k)] \text{ tq } \mathbf{q}(k) \in [\mathbf{q}(k)]$$

À chaque instant k, nous recevons une série de mesures y_i nous indiquant la distance séparant un certain nombre i d'amers du capteur ayant réalisé ces mesures. Cette information nous renseigne sur la position du robot. De même, nous avons accès à l'angle γ_i mesuré entre le repère local du capteur et les amers détectés dans l'environnement (cf. image 3.2). Cette information nous renseigne sur l'orientation du robot. Ainsi, nous pouvons estimer à chaque prise de mesure, la posture du robot en fonction de la position des amers détectés dans l'environnement. Pourtant, ces mesures sont affectées par des incertitudes propres au capteur qui les a réalisées. Il est donc nécessaire d'identifier les sources d'erreurs susceptibles d'influencer le calcul de $[\mathbf{q}(k)]$. Dans ce projet de recherche, nous utiliserons un Lidar "Velodyne Puck" doté d'une précision de mesure [4] de $\epsilon_y = \pm 3cm$. Définissons dès maintenant quelques variables qui seront utilisées pour déterminer le fonctionnement de l'algorithme IAPT. Définissons pour chaque mesures $y_i(k)$ un intervalle $[y_i(k)] = [y_i(k) - \epsilon_y, y_i(k) + \epsilon_y]$ comprenant la

4. Velodyne Puck Data Sheet

distance entre le robot et l'amer détecté, de manière garantie. Dans ce contexte, l'algorithme IAPT cherchera à fournir une liste $\mathcal{L}_k = \{[\mathbf{q}_k]\}$ avec $[\mathbf{q}(k)] = ([x_1(k)],[x_2(k)],[\theta(k)])$ contenant toutes les positions et orientations possible du robot tel que $\mathbf{q}(k) \in [\mathbf{q}(k)]$. Notons qu'à l'origine, nous estimons une posture initiale $[\mathbf{q}(0)]$ à l'aide de la connaissance de l'environnement et des données initiales issues du capteur Lidar selon la méthode présenté dans l'annexe A.7

Dans l'article [12] la localisation est réalisée à l'aide d'une carte sous forme d'une grille d'occupation. Notons que la discrétisation d'un environnement n'est pas une tâche triviale puisqu'elle nécessite l'intervention d'un opérateur compétent (cf. annexe A.1). De plus, cette démarche doit être répétée à chaque fois que le robot explore un nouvel environnement. Nous avons donc décidé, pour l'algorithme IAPT, de remplacer cette grille d'occupation par une carte d'amers.

3.2 Création de la carte d'amers

Dans ce projet de recherche, nous cherchons donc à résoudre un problème de localisation en ayant recours à une carte de l'environnement. Contrairement à l'article [12], nous ne disposons pas d'une discrétisation de l'environnement complet sous forme d'une grille d'occupation. En substitut, nous dispersons dans l'environnement un certain nombre d'amers, représenté par des plots sur l'image 3.1. Formulons le CSP à résoudre pour créer cette carte, qui consiste à résoudre un problème de contraction de distances deux à deux :

$$\begin{aligned} V &= \left\{ \{\mathbf{w}_i\}_{\forall i}, \{y_{i,j}\}_{\forall i,j} \right\} \text{avec} \, [\mathbf{w}_i] = [w_{1_i}, w_{2_i}] \\ D &= \left\{ \begin{array}{c} \{[\mathbf{w}_i]\}_{\forall i}, \{[y_{i,j}]\}_{\forall i,j} \\ \mathbf{w}_0 = ([0,0],[0,0]) \\ \mathbf{w}_1 = ([0,+\infty],[0,0]) \\ \mathbf{w}_2 = ([-\infty,+\infty],[0,+\infty]) \\ \mathbf{w}_{i\geq 3} = ([-\infty,+\infty],[-\infty,+\infty]) \end{array} \right\} \\ C &= \left\{ c_{y_{i,j}} : y_{i,j}^2 = (w_{1_i} - w_{1_j})^2 + (w_{2_i} - w_{2_j})^2 \right\} \end{aligned} \tag{3.4}$$

Avec y une mesure de distance et $\mathbf{w}$ une boîte contenant un amer de façon garantie. Nous remarquons dans D que les domaines de $\mathbf{w}_{1,2,3}$ ont été délimités dans le but de définir un repère au sein de la carte générée par ce CSP. En effet, nous remarquons que la boîte $\mathbf{w}_0$ sert d'origine à ce repère, et que $\mathbf{w}_1$ et $\mathbf{w}_2$ servent respectivement à définir l'orientation de l'axe des abscisses et des ordonnées. La solution de ce CSP sera une carte contenant les estimations de positions de chacun des plots (amers) contenus dans des boîtes dans un référentiel local. L'algorithme IAPT pourra donc utiliser cette information pour réévaluer la posture du robot dans le repère local de la carte, à chaque itération.

Observons la figure 3.1 pour repérer les éléments qui composent le CSP présenté précédemment. Nous relevons deux éléments :

1. En bleu, les contraintes affectées au CSP : le plot A sert d'origine, le plot C est situé sur l'axe des abscisses positives et le plot B possède une ordonnée positive.
2. En rouge, les distances plots à plots (i.e. $y_{i,j}$) qui composeront la matrice des distances.

Enfin, le contracteur associé à la contrainte $c_{y_{i,j}}$ est un *contracteur de distance* entre deux points que nous notons $C_{dst}([d],[\mathbf{a}],[\mathbf{b}])$ avec $\mathbf{a} = (a1,a2)$ et $\mathbf{b} = (b1,b2)$. Ce contracteur peut être construit

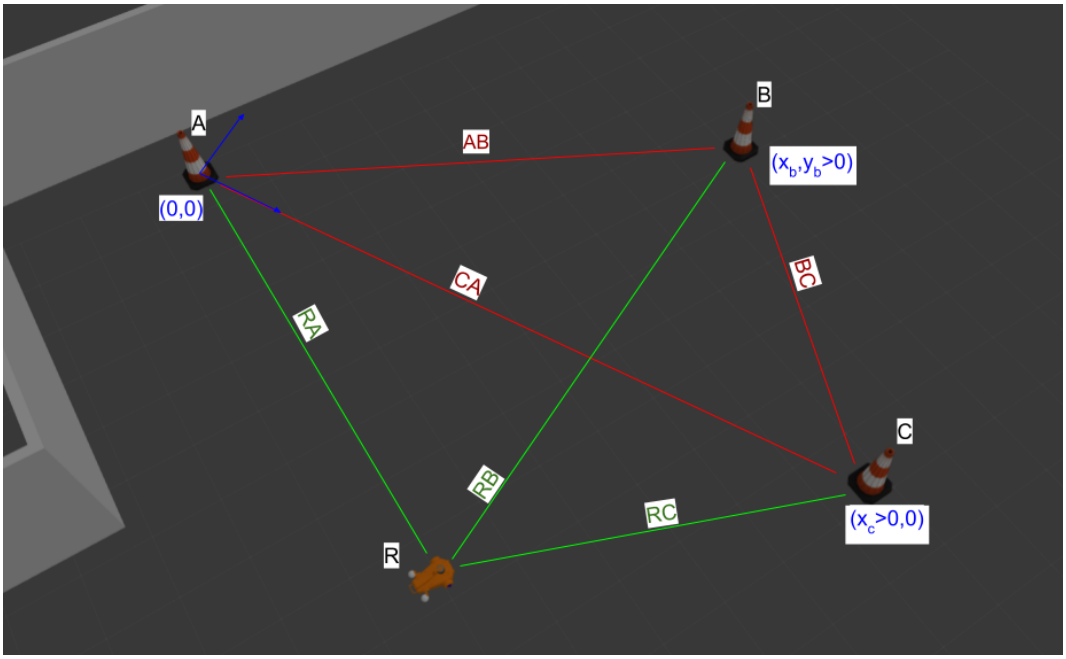


FIGURE 3.1 – Exemple de carte d'amers

comme une propagation avant-arrière sur l'équation suivante :

$$d^2 = (a_1 - b_1)^2 + (a_2 - b_2)^2 \tag{3.5}$$

Le contracteur $C_{dst}([d],[\mathbf{a}],[\mathbf{b}])$ est défini selon l'algorithme 2 que nous trouvons dans l'annexe A.4.

Maintenant que nous possédons une carte d'amers il ne nous reste plus qu'à définir le CSP résolu à chaque instant k par l'algorithme IAPT afin de fournir une série de boîtes $[\mathbf{q}(k)]$ contenant de manière garantie la posture du robot, dans le référentiel de la carte.

3.3 Formulation du CSP lié au problème de localisation

À l'aide de la thèse [11], nous pouvons expliciter le CSP à résoudre dans le contexte d'un problème de localisation :

$$\begin{aligned} V &= \{\mathbf{q}, \{y_i, \gamma_i, \mathbf{w}_i\}_{\forall i}\} \\ D &= \{[\mathbf{q}], \{[y_i], [\gamma_i], [\mathbf{w}_i]\}_{\forall i}\} \\ C &= \left\{ c_{\mathbf{w}} : \mathbf{w}_i = \begin{pmatrix} y_i \cos(\theta + \gamma_i) + x_1 \\ y_i \sin(\theta + \gamma_i) + x_2 \end{pmatrix} \right\} \end{aligned} \tag{3.6}$$

Ainsi dans le CSP 3.6, V correspond à un ensemble de variables : la posture du robot $\mathbf{q}$, la mesure de distance y_i, l'angle du capteur γ_i et la boîte contenant l'amer $\mathbf{w}_i$, avec i le nombre d'amers utilisés pour localiser le robot. Petite précision concernant la variable γ (cf. image 3.2), celle-ci correspond à l'orientation du rayon ayant servi à mesurer la distance séparant le robot d'un amer, par rapport à

l'orientation θ du robot. D est l'ensemble des domaines de valeurs des variables précédentes. Enfin, C est un ensemble de contraintes. Particulièrement, $c_{\mathbf{w}}$ correspond à la construction de la boîte $\mathbf{w}_i$ contenant un amer. L'algorithme lié au contracteur $c_{\mathbf{w}}$ est détaillé dans l'annexe A.5, pour un exemple d'utilisation, se référer à l'annexe A.2.

La résolution du CSP 3.6 nous fournira une boîte (i.e. un intervalle 3D) $[\mathbf{q}(k)] = ([x_1(k)],[x_2(k)],[\theta(k)])$ contenant toutes les positions et orientations possibles du robot à l'instant k.

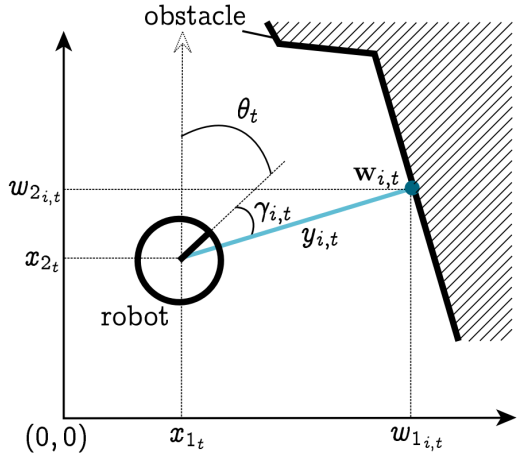


FIGURE 3.2 – Nous remarquons que γ correspond à l'angle de mesure d'un capteur par rapport à l'orientation θ du robot (source image : [11])

Intéressons nous maintenant à l'application technique des éléments théoriques que nous venons d'aborder.

Chapitre 4

Travaux réalisés

Dans ce chapitre, nous détaillerons les travaux réalisés pour obtenir une vérité terrain garantie et l'utiliser pour évaluer la précision d'un algorithme de SLAM. Premièrement, nous avons créé une simulation du robot et de son environnement. Cet environnement de travail a pour intérêt de nous donner accès à la vérité terrain qui est une information très utile pour vérifier la pertinence de nos résultats. Dans un second temps, nous présenterons les étapes nécessaires à la détection des amers (point de repère fixe et identifiable) utilisés pour la localisation du robot. Enfin, nous détaillerons l'implémentation de l'algorithme IAPT.

4.1 Simuler l'environnement, le robot et ses capteurs

Nous trouverons dans l'annexe A.3 la présentation du middleware ROS [1] qui est utilisé pour le développement de ce projet de recherche.

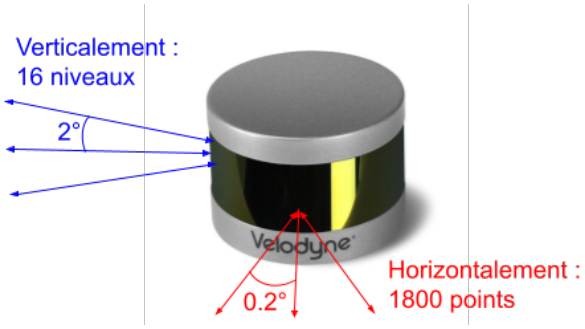
4.1.1 Gazebo

Gazebo est un simulateur 3D, cinématique et dynamique qui nous permet de simuler un robot et ses capteurs dans son environnement, avec la possibilité de modéliser les caractéristiques physiques du robot. Le but de cette simulation est de fournir un modèle virtuel fidèle à la réalité. Pour ce projet de recherche, nous simulons le robot RoMuLux qui serait utilisé pour d'éventuelles prises de mesures réelles. Ce robot est équipé d'un capteur Lidar dont nous simulons le fonctionnement. Ainsi, le logiciel Gazebo publiera le même type de message que publierait un véritable capteur lors d'une prise de mesure. Ce logiciel nous permet de créer une simulation indistinguable de la réalité du point de vue d'un processus "nodes" (cf. l'annexe A.3). En effet, l'algorithme IAPT, ainsi que tout autre algorithme de SLAM que nous souhaiterions tester, travailleront avec le même jeu de données Lidar. Il est donc important de construire une simulation rigoureuse pour s'approcher au maximum d'un modèle réaliste.

1. *Robotic Operating System*

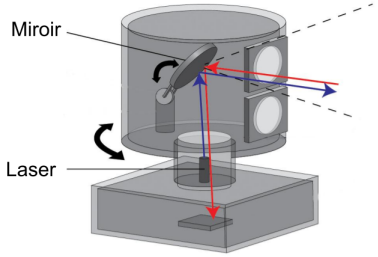
4.1.2 Lidar

Le capteur Lidar est un instrument de mesure utilisant des faisceaux laser. Lidar est un acronyme de l'anglais "Laser Imaging Detection And Ranging", soit en français "détection et estimation de la distance par laser". Pour effectuer ses mesures, le Lidar émet une impulsion laser qui ira potentiellement rebondir sur les surfaces alentour (e.g. murs, plafond, obstacles...) pour revenir au capteur. La distance est ensuite mesurée grâce au temps mis par la lumière pour faire l'aller-retour. Pour ce projet, nous utilisons un Lidar VLP-16 [2] qui a la particularité d'avoir une résolution en 3D. En observant l'image 4.1a nous remarquons que le capteur émet des faisceaux à 360° horizontalement, mais également sur 16 couches verticalement. L'inclinaison des faisceaux verticaux vont de -15° à +15°, chaque niveau séparé de 2°. Horizontalement, la résolution angulaire est de 0.2°, pour parcourir un tour complet (360°) nous relèverons donc 360/0.2 = 1800 points. Pour ce projet, le Lidar est cadencé à 10 révolutions par seconde, nous obtiendrons donc 10 nuages de points composés de 16∗1800 = 28800 points à destination des algorithmes de SLAM et d'IAPT. Pour émettre sur différents niveaux verticaux, le Lidar règle l'inclinaison d'un miroir comme nous pouvons le noter sur l'image 4.1b, pour les couches horizontales, le Lidar effectue des rotations sur lui-même grâce à un servomoteur.



(a) Répartition spatiale des impulsions [a]

a. Source image



(b) Orientation du laser [a]

a. Source image

FIGURE 4.1 – Lidar Velodyne VLP-16

4.1.3 Rosbag

Les Rosbags sont un élément important de l'environnement ROS, pour plus de détails sur le middleware ROS, se référer à l'annexe A.3. Ces fichiers contiennent un enregistrement de données horodatées. La création de ces archives se fait en s'abonnant à un ou plusieurs topics ROS, comme par exemple les données issues d'un capteur lidar, d'une centrale inertielle ou une vérité terrain issue d'un logiciel de simulation (e.g. Gazebo). Cet outil stocke les données des messages sérialisés dans un fichier au fur et à mesure de leur réception. Ces données peuvent ensuite être republiées ultérieurement à la même fréquence qu'elles ont été enregistrées. Cet outil permet donc à plusieurs algorithmes de localisation (SLAM, IAPT) de travailler séparément sur le même jeu de données.

2. Documentation Lidar VLP-16

4.2 Détecter et segmenter les amers contenus dans un nuage de points

Dans ce projet, nous utilisons un Lidar afin de détecter l'environnement. Ce type de capteur nous fournit comme données un nuage de points que nous devrons traiter afin d'en extraire l'emplacement des amers présent dans l'environnement. Pour ce faire, nous utilisons la librairie PCL (Point Cloud Library) qui nous donne accès à de nombreuses routines de filtrage, de segmentation, de détection de plan, etc. L'image 4.2 résume le pipeline utilisé [22] pour détecter et segmenter les amers contenus dans un nuage de points.

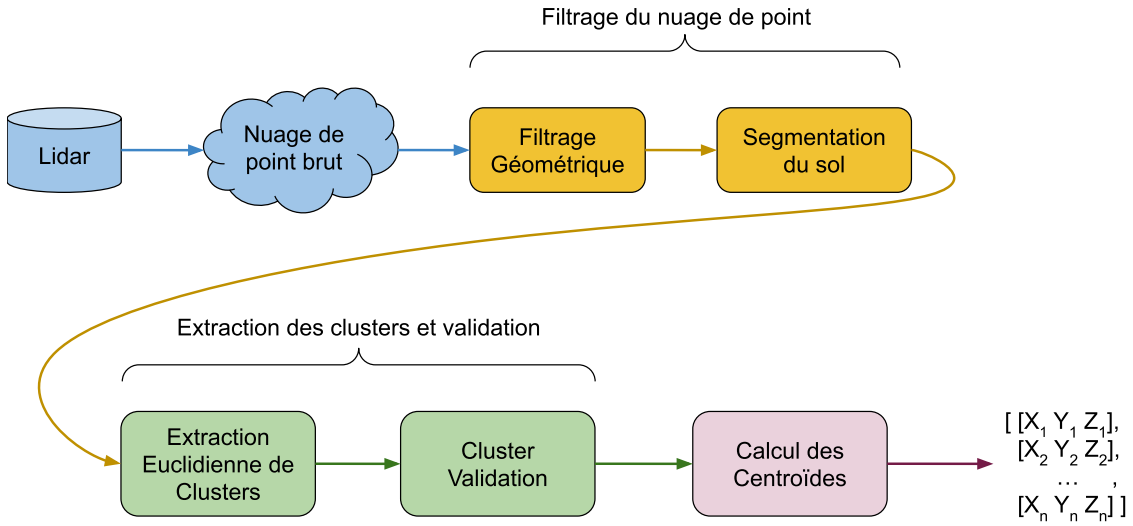


FIGURE 4.2 – Pipeline de détection de cônes dans un nuage de points

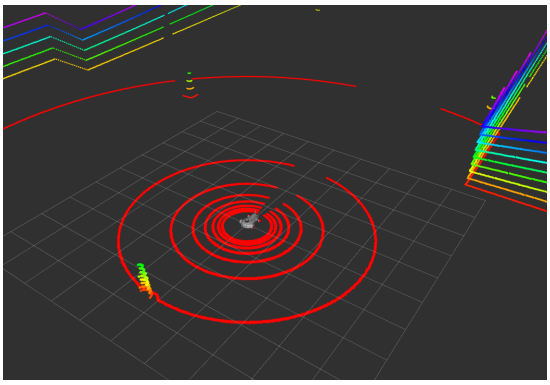
4.2.1 Filtrage d'un nuage de points

Le lidar VLP-16 [3] fournit 10 fois par seconde un nuage de points composé de près de 30000 points. Pourtant, seulement quelques points nous fournissent des informations utiles, c'est-à-dire l'emplacement des cônes qui nous servent d'amers dans ce projet. Traiter dans l'état cette grande quantité d'information impliquerait une charge de calcul élevée et un temps de traitement excessif. Nous chercherons donc à filtrer l'information en passant par deux étapes, le filtrage géométrique et la segmentation du sol.

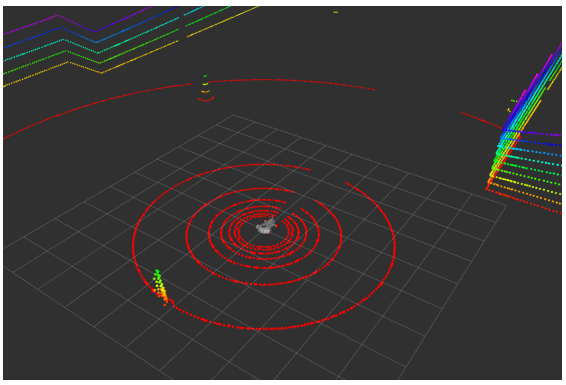
3. Documentation Lidar VLP-16

Filtrage géométrique

Nous allons tout d'abord faire passer le nuage de points par un filtre voxel[4]. Un voxel est la contraction de "volume" et "pixel". En effet, le voxel est à la 3D ce que le pixel est à la 2D. Le principe est donc de segmenter le nuage de points en un maillage 3D et de moyenner les éléments contenus dans chacune des cases (chaque voxel). Nous remarquons en comparant l'image 4.3a et l'image 4.3b, que dans le second cas le nuage de points paraît plus clairsemé tout en conservant les informations utiles. En effet, nous reconnaissons encore les murs, les cônes et le sol en ayant gardé seulement 25% des points qui composaient le nuage de points à l'origine.

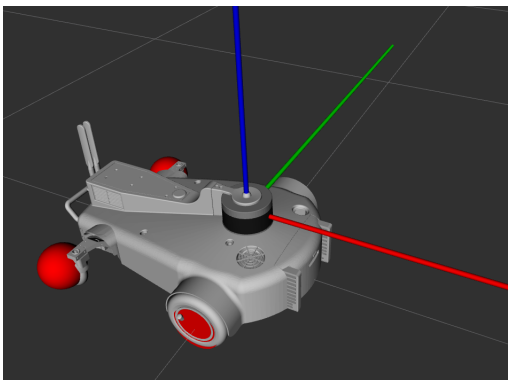


(a) Nuage de points brut

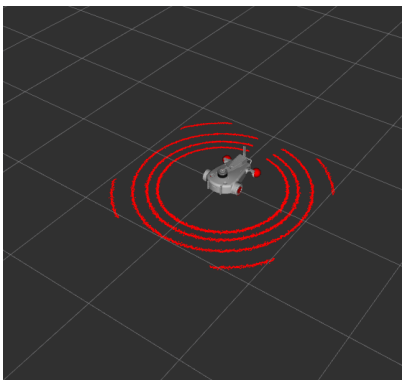


(b) Effet d'un filtre "voxel"

Nous pouvons davantage réduire la quantité d'information à traiter en appliquant un filtre cropbox au nuage de points. Ce filtre nous permet de délimiter une boîte à l'extérieur de laquelle toute information sera ignorée. En effet, nous avons accès à la position 3D de chaque point dans le référentiel du Lidar que nous pouvons observer sur l'image 4.4a. En partant du principe que le Lidar sera toujours orienté de la même manière (i.e. parallèle au sol) et que les cônes sont posés au sol, nous pouvons définir une boîte éliminant des parties du nuage de points n'ayant aucune chance de contenir un cône (i.e. tout point à une altitude supérieure à 1 m). Nous pouvons observer l'effet d'un cropbox filter grâce à l'image 4.4b où seulement les points contenus dans une boîte d'1 m^2 sont retenu.



(a) Repère associé au Lidar



(b) Effet d'un filtre "voxel"

4. *Volume + pixel*

Segmentation du sol

Après avoir sous-échantillonné le nuage de points et supprimé les points situés à une altitude supérieure à 1 m, nous chercherons à éliminer le sol. En effet, après avoir retiré les points associés au sol, il ne restera plus que quelques clusters isolés (e.g. les cônes, les murs et tout autres objets). Ce filtrage aura pour effet de réduire la taille du nuage de points et donc d'accélérer l'opération de recherche d'amers. Pour séparer les points appartenant au sol du reste du nuage, nous utiliserons la méthode RANSAC [5] [7] qui a pour intérêt d'être robuste aux changements d'inclinaisons du capteur Lidar. Cette méthode itérative consiste à tester une hypothèse dans un ensemble de données, et de ne garder que la meilleure solution trouvée au bout d'un certain nombre d'essais. Dans notre cas, l'hypothèse est qu'il existe dans le nuage de points un plan correspondant au sol. L'algorithme tentera d'ajuster un plan (2D) situé dans un espace (3D) en proposant une solution aléatoire à chaque itération. Cette solution potentielle contient à la fois des points pertinents (inliers), c'est-à-dire, des points qui appartiennent approximativement au plan, et des points aberrants (outliers), des points qui sont trop éloignés de ce modèle. À la fin d'un certain nombre d'itération, nous sélectionnerons la solution la plus probable. Cet algorithme est non-déterministe, il produit un résultat correct avec une certaine probabilité seulement, celle-ci augmentant à mesure que le nombre d'itérations est grand. Il n'y a donc aucune garantie d'obtenir la bonne solution. Il faudra donc ajuster empiriquement un certain nombre de paramètres de l'algorithme pour espérer obtenir un niveau de probabilité suffisamment élevé de trouver le plan correspondant au sol. Sur l'image 4.5 nous pouvons observer le nuage de points original (points blancs), superposé au nuage de points résultant de l'application des différents filtres détaillés plus tôt (points en couleurs). Le filtre voxel (sous-échantillonnage) a eu pour effet de réduire la densité du nuage de points de 75%. Nous observons que le filtre cropbox a supprimé les points situés à plus d'1 m d'altitude (relatif au lidar). Enfin la méthode RANSAC a permis d'identifier les points appartenant au sol et de les retirer. Il ne reste plus qu'à réaliser l'extraction et la validation des clusters correspondant aux cônes.

4.2.2 Extraction des clusters et validation

À l'aide des étapes précédentes, nous possédons un nuage de points composé de clusters isolés qu'il va falloir identifier et labelliser. Pour ce faire, nous utiliserons une méthode de segmentation utilisant pour critère discriminant la distance euclidienne. Enfin, nous utiliserons des hypothèses simples pour valider la présence ou non d'un cône dans un cluster segmenté.

Extraction Euclidienne des clusters

L'intérêt principal d'une méthode de clustering est de segmenter un nuage de points non organisé en parties plus petites (e.g. un cluster), de sorte que le temps de traitement global de ce nuage de points soit considérablement réduit. Pour réaliser cette segmentation nous partons du principe que le sol a été identifié et supprimé dans le nuage de points original. Le but de l'algorithme de clustering euclidien [6] est de repérer les différents clusters en analysant la distribution de densité des points dans l'espace, avec pour critère la distance euclidienne. Le fonctionnement de cet algorithme issu de la librairie

5. *RANdom SAmple Consensus*
6. Documentation Euclidean Clustering

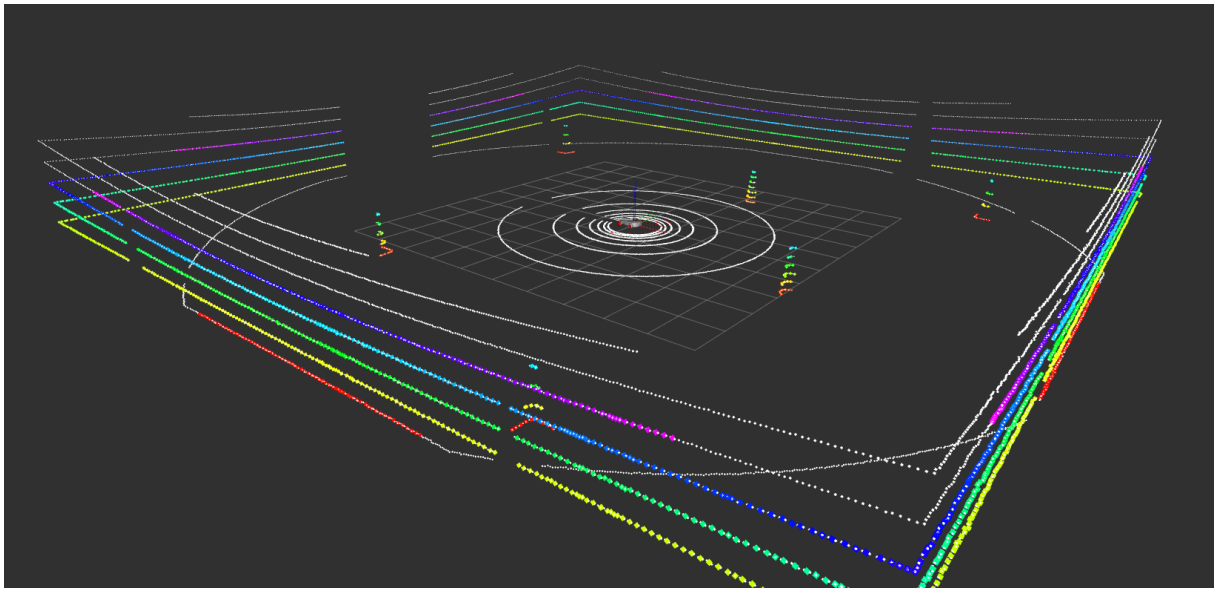


FIGURE 4.5 – nuage de points filtré (Filtre voxel, filtre cropbox, RANSAC)

PCL [7] est détaillé par l'algorithme 6 dans l'annexe A.10. La première étape de cet algorithme est de créer un arbre k-d [8] P à partir de la position spatiale de chaque point restant dans le nuage de points filtré précédemment. Un arbre k-d est une structure de données de partition de l'espace permettant de stocker des points, et de faire des recherches (e.g. plus proche voisin) plus rapidement qu'en parcourant linéairement un tableau stockant les coordonnées de points (e.g. nuage de points). L'intérêt de cette structure est que dans notre cas, cet arbre k-d en dimension 3 nous permettra de concentrer la recherche des plus proches voisins dans des zones spatiales spécifiques, ce qui améliore la vitesse de traitement. Les étapes suivantes de l'algorithme 6 consistent à assigner chaque points $\mathbf{p}_i$ à un cluster C_i à l'aide d'un test simple et d'une valeur de distance euclidienne 3D r. Cette valeur est définie empiriquement selon le rayon d'une sphère de recherche : pour chaque point, si un point voisin se trouve dans la sphère de recherche et qu'il n'a pas encore été traité, il est ajouté à une liste qui formera un cluster. La valeur de r est donc un seuil critique, car une trop petite valeur pourrait diviser un cône en plusieurs petits clusters. Inversement, une trop grande valeur pourrait amalgamer deux cônes en un seul cluster.

Une autre valeur ajustée empiriquement dans l'algorithme est le nombre minimum et maximum de points composant un cluster. Le but étant d'ignorer les trop petits objets (e.g. des éléments trop éloignés) ou de trop gros objets (e.g. un mur). À l'issue de ce traitement, le nuage de points original n'est réduit qu'à un certain nombre de clusters candidats contenant potentiellement un cône. Il est maintenant nécessaire de passer par une étape de validation pour identifier d'éventuels faux positifs.

Validation des Clusters

Comme nous travaillons dans un environnement contrôlé, nous ne nous attendons pas à différents types d'obstacles que des cônes dans l'environnement. Il est possible d'utiliser des approches so-

7. *Point Cloud Library*
8. Documentation k-dimensional tree

phistiquées pour analyser la forme d'un objet à l'aide de vecteurs normaux à sa surface. Cette approche est coûteuse en termes de temps de calcul, mais sera nécessaire pour identifier les cônes dans une expérimentation hors simulation. Dans notre cas, les tests simples et empiriques suivants sont suffisants pour identifier les clusters contenant des cônes :

1. La distance moyenne entre chaque points composant le cluster est inférieur à 20 cm.
2. Le cluster est composé de 20 à 150 points.
3. Le centre du cluster est situé à moins de 50 cm du sol. Tous les cônes touchent le sol, si le centre d'un cluster est trop haut, il est ignoré.
4. Le centre du cluster ne se trouve pas trop proche de l'angle mort dont le robot simulé est affecté. En effet, une partie de la surface d'un cône aurait pu disparaître dans cet angle mort, impactant le calcul du centroïde de ce cône.

Dans le cas où un cluster respecterait toutes ces conditions, alors les coordonnées du centroïde associé à ce cluster sont transmises à l'algorithme IAPT qui se chargera de localiser le robot.

Concernant le calcul du centroïde d'un nuage de points représentant un amer, il faut noter que le nuage de points que nous obtenons correspond à la face visible du cône relatif au capteur comme nous pouvons le voir sur la figure 4.6. Ainsi, calculer l'emplacement du centroïde en moyennant simplement les coordonnées des points composant ce nuage ne suffit pas, puisqu'il "manque" la moitié du cône. Nous trouverons dans l'annexe A.6 la résolution d'un problème de moindres carrés linéaires ayant pour effet d'améliorer le calcul des coordonnées de ce centroïde.

FIGURE 4.6 – Le nuage de points associé à un amer ne nous fournit d'information que sur la face exposée du cône

4.3 Implémentation de l'algorithme IAPT

Rappelons que l'objectif de l'algorithme IAPT est de résoudre un problème de suivi de posture dans un contexte d'erreur bornée. L'algorithme cherche à trouver la boîte $[\mathbf{q}(k)]$ à chaque instant k contenant la posture du robot de manière garantie. Les données d'entrée de l'algorithme IAPT (cf. algorithme 1) sont les boîtes $[\mathbf{w}_i]$ contenant un amer chacune, les mesures de distance $[y_i]$ séparant le capteur des i amers et les mesures d'angle $[\gamma_i]$ mesurées entre le repère local du capteur et les i amers détectés dans l'environnement (cf. image 3.2). Cet algorithme comporte deux étapes importantes : l'étape

Algorithm 1: Interval Analysis Pose Tracking

Data: $[\mathbf{w}_i(0)]$,$[\gamma_i(0)]$,$[y_i(0)]$

1 initialize $[\mathbf{q}(0)]$;
2 **for** $k \leftarrow 1$ *to* K **do**
3 update $[\mathbf{q}(k-1)]$ to $[\mathbf{q}(k)]$;
4 compute $[\gamma_i(k)]$ and $[y_i(k)]$;
5 contract $[\mathbf{q}(k)]$ and $[\mathbf{w}_i(k)]$ using Contractor $C_{\mathbf{w}}([\mathbf{w}_i(k)],[\gamma_i(k)],[y_i(k)],[\mathbf{q}(k)])$;

Result: $\mathcal{L}_K = \{[\mathbf{q}_k]\}_{\forall k}$

de **prédiction** (ligne 3) et l'étape de **contraction** (ligne 4) que nous détaillerons dans les sections suivantes. Notons que l'indice k itère à travers le nombre K de nuages de points enregistrés par le capteur Lidar. Enfin, la sortie de l'algorithme sera une liste $\mathcal{L}_k = \{[\mathbf{q}_k]\}$ avec $[\mathbf{q}(k)] = ([x_1(k)],[x_2(k)],[\theta(k)])$ contenant toutes les positions et orientations possibles du robot tel que $\mathbf{q}(k) \in [\mathbf{q}(k)]$.

4.3.1 Étape de prédiction

Dans cette étape, nous estimons une posture $[\mathbf{q}(k)]$ à chaque instant k qui sera ensuite contractée lors de l'étape suivante de contraction. Attardons-nous rapidement sur la phase d'initialisation de l'algorithme IAPT. Nous pouvons profiter de l'hypothèse que nous connaissons à l'avance les cônes utilisés pour créer le repère local de la carte d'amer. Grâce à cette information, nous pouvons facilement estimer une posture initiale $[\mathbf{q}(0)]$ du robot. Pour plus de détails sur cette estimation de posture initiale, se référer à l'annexe A.7. Après cette initialisation, la boîte $[\mathbf{q}(k)]$ peut être estimée de plusieurs façons différentes lors de l'étape de prédiction. Nous ne présenterons que deux méthodes implémentées lors de ce projet. La première approche consiste à agrandir la posture $[\mathbf{q}(k-1)]$ à l'aide de $\max_{0<k<\infty}(\Delta_{\mathbf{x}_k})$ et de $\max_{0<k<\infty}(\Delta_{\theta_k})$, respectivement la distance maximale parcourue et la différence d'orientation maximale entre $\mathbf{x}_k$ et $\mathbf{x}_{k+1}$ pour $k \in \{0,\infty\}$. Cette approche pessimiste garantie que le robot se trouve effectivement dans l'estimation de posture $[\mathbf{q}(k)]$. La deuxième approche consiste à estimer le déplacement effectué par le robot à l'aide de la commande de déplacement $\mathbf{u}_{k-1}$ ou en utilisant une centrale inertielle (Imu). Grâce au logiciel de simulation Gazebo présenté plus tôt, il est possible d'implémenter facilement une Imu ou de récupérer la commande de déplacement $\mathbf{u}_{k-1}$ et donc d'estimer le déplacement effectué entre les instants $k-1$ et k. Cette approche a pour avantage de fournir une mise à jour moins pessimiste de $[\mathbf{q}(k)]$. Enfin, une fois la posture $[\mathbf{q}(k)]$ estimée par quelconque méthode présentée précédemment, nous transférons cette prédiction à l'étape de contraction.

4.3.2 Étape de contraction

Rappelons que cette étape consiste à résoudre le CSP 3.6 présenté précédemment, nécessitant en entrée l'estimation de posture $[\mathbf{q}(k)]$ issue de l'étape de prédiction, l'estimation de position des amers $[\mathbf{w}_i]$ ainsi que les mesures de distance et d'angle entre le robot et les obstacles $[y_i]$ et $[\gamma_i]$. Comme présenté dans l'annexe A.7, le robot détecte des clusters dans le référentiel du capteur Lidar dont il est équipé. Le centre du Lidar étant l'origine de ce repère, les coordonnées spatiales des centroïdes de chaque clusters nécessiteront une transformation géométrique pour pouvoir être associés à la boîte $[\mathbf{w}_i]$

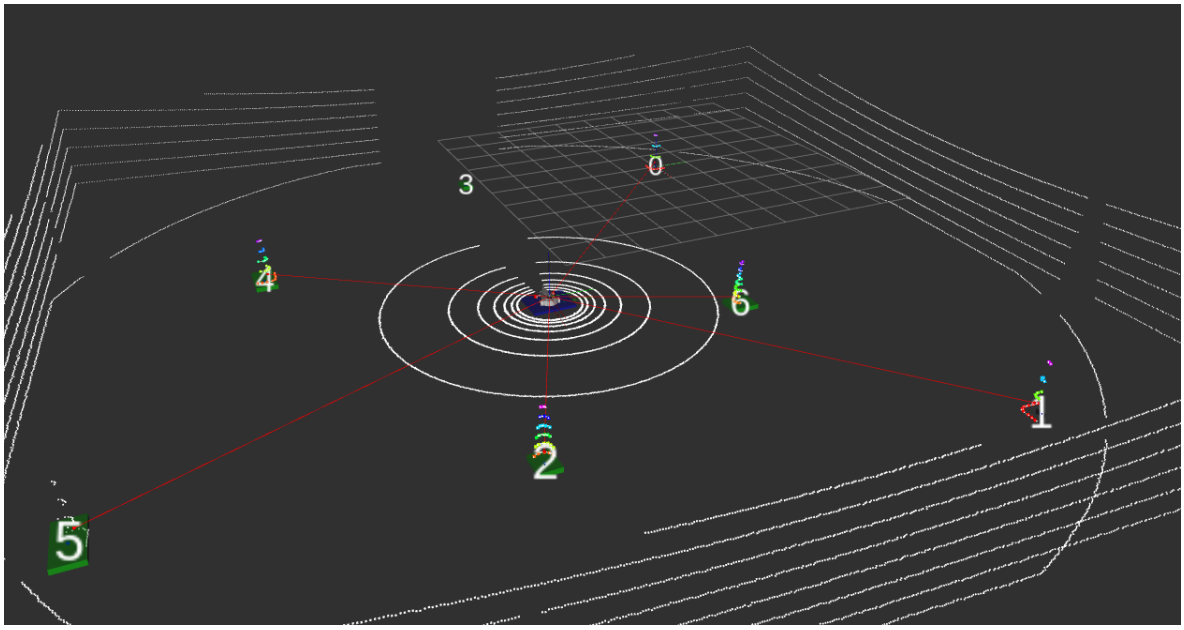
correspondante dans la carte d'amer. Cette transformation est la même que celle présentée dans l'annexe A.2. Une fois qu'un centroïde a été associé avec succès à une boîte $[\mathbf{w}_i]$ de la carte d'amer, nous pouvons procéder à l'étape de contraction en utilisant le contracteur $c_{\mathbf{w}}$ détaillé dans l'annexe A.5.

4.3.3 Visualisation

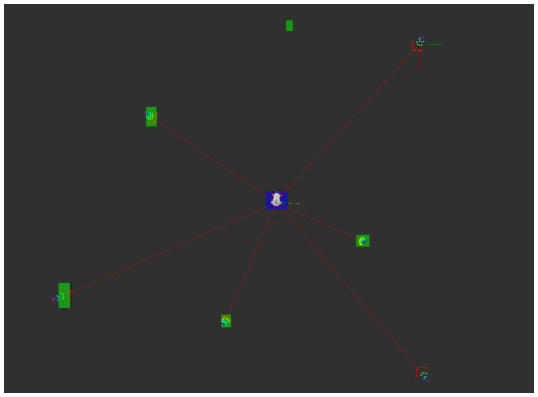
Nous utilisons le logiciel de visualisation 3D Rviz [9] pour illustrer graphiquement les principes présentés précédemment. Sur l'image 4.7a nous observons une vue d'ensemble du système mis en place pendant ce sujet de recherche. Premièrement, en blanc, nous voyons un nuage de points non filtré issu de la simulation d'un capteur Lidar. Nous y distinguons les murs, les cônes et le sol qui composent l'environnement simulé. En couleur arc-en-ciel, nous observons le résultat de la segmentation du nuage de points (le gradient de couleur correspond à la position z des points). Les murs et le sol ont été ignorés, il ne reste plus que des clusters qui correspondent uniquement à des cônes. Les boîtes vertes représentent les amers $[\mathbf{w}_i]$ issus de la carte d'amers dont la création a été détaillée précédemment. À l'aide des chiffres de l'image 4.7a, nous identifions les cônes qui ont servi à créer le repère local de la carte d'amer : avec l'origine au point 0, l'axe x passant par le point 1 et enfin l'axe y orienté vers le point 2 (cf. Création de la carte d'amers au chapitre 3). Enfin, au centre de l'image, nous distinguons le robot simulé contenu dans l'estimation de posture $[\mathbf{q}(k)]$ représenté par une boîte bleue. Les lignes rouges représentent la vérité terrain des mesures de distance et d'angle y_i et γ_i entre le robot et les obstacles. Nous rappelons que la vérité terrain n'est normalement pas une information disponible, nous représentons ces lignes rouges dans un but de compréhension. Les images 4.7b et 4.7c servent à montrer que le plot 3, dans l'angle mort du robot n'est pas encore détecté par le Lidar. Il n'existe donc pas de cluster à segmenter à cette position d'où l'absence de trait rouge vers ce point. Pourtant la boîte verte $[\mathbf{w}_i]$ correspondant à l'estimation de position d'un amer est déjà présente, cela est due au fait que nous possédons la carte d'amers à priori. Nous savons qu'un obstacle se trouve dans cette zone avant même de l'avoir formellement détecté.

Pour résumer, à chaque itération, l'algorithme IAPT fournit une zone dans laquelle se trouve le robot, de manière garantie. À travers le temps, cette série de boîtes $[\mathbf{q}(k)]$ formera une vérité terrain garantie. Ce résultat sera alors utilisé pour l'évaluation des prédictions de postures issues d'un algorithme de SLAM. Maintenant, que nous avons présenté l'essentiel des travaux réalisés, nous commenterons les résultats obtenus dans le prochain chapitre.

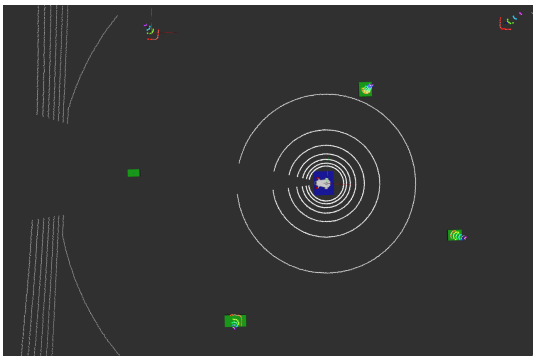
9. Documentation ROS Vizualisation



(a)



(b)



(c)

FIGURE 4.7

Chapitre 5

Résultats

Dans ce chapitre, nous utiliserons les résultats issus de l'algorithme IAPT à la prédiction de posture d'un algorithme de SLAM afin d'évaluer les performances de ce dernier. Pour commencer, nous présenterons les étapes nécessaires à la réalisation d'une phase de test. Ensuite, nous proposerons plusieurs critères d'évaluations lors de l'analyse des résultats obtenus. Enfin, nous discuterons des pistes d'améliorations permettant d'étendre la portée de ce projet de recherche. Notons les terminologies suivantes utilisées dans ce chapitre :

- La trajectoire du robot : liste de postures du robot dans le logiciel de simulation Gazebo
- La trajectoire estimée du robot : liste de postures du robot estimée par un algorithme de SLAM
- La vérité terrain garantie : liste des ensembles de postures du robot issue de l'algorithme IAPT

5.1 Présentation des scénarios de test

Cette phase d'expérimentation est réalisée à l'aide du logiciel de simulation Gazebo présenté dans le chapitre 4. Nous apprenons dans ce chapitre qu'il est possible de modéliser un environnement plus ou moins complexe, et d'y disséminer des cônes qui serviront d'amers pour l'algorithme de localisation IAPT (cf. images 5.1a et 5.1b). Nous avons ensuite enregistré dans un rosbag les nuages de points horodatés acquis à l'aide de la simulation d'un capteur Lidar monté sur un robot mobile. Nous avons aussi enregistré la trajectoire du robot, c'est-à-dire la posture du robot dans la simulation à chaque prise de mesures. Les nuages de points enregistrés dans les rosbags sont ensuite transmis aux algorithmes de SLAM et d'IAPT qui se chargeront de fournir respectivement la trajectoire estimée du robot et la vérité terrain garantie à chaque pas de temps. Enfin, nous enregistrons ces résultats afin de les analyser et de les visualiser.

En résumé, le déroulement d'une phase de test est le suivant :

1re phase : Création d'un environnement et dispersion dans celui-ci de cônes servant d'amers pour l'algorithme IAPT.

2de phase : Enregistrement dans des rosbags des nuages de points issus d'un capteur Lidar monté sur un robot terrestre.

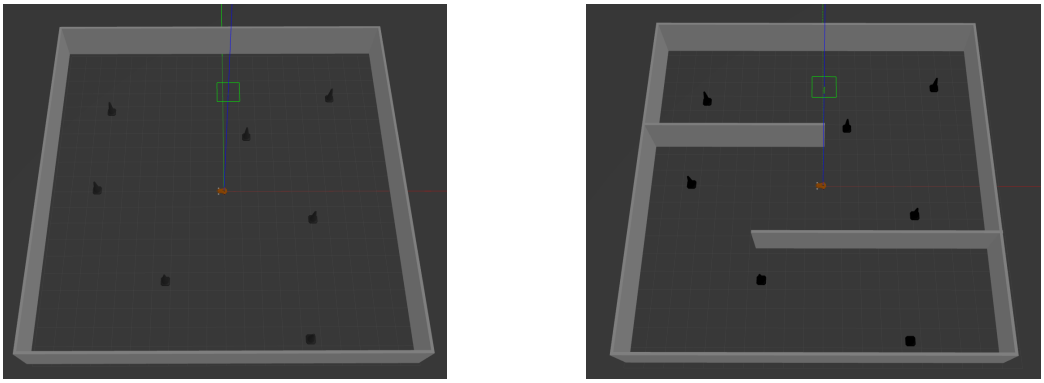
3e phase : Calcul de la vérité terrain garantie issue de l'algorithme IAPT en utilisant uniquement

les nuages de points enregistrés dans les rosbags.

4e phase : Calcul de la trajectoire estimée issue d'un algorithme de SLAM en utilisant uniquement les nuages de points enregistrés dans les rosbags.

5e phase : Extraction des postures issues des étapes précédentes en vue de leur analyse et de leur affichage dans le logiciel de visualisation Rviz.

À l'issue de ces étapes, nous possédons une série de postures estimée par un algorithme de SLAM ainsi qu'une série de boîtes contenant la posture du robot de manière garantie. Il ne nous reste plus qu'à évaluer la justesse des prédictions de l'algorithme de SLAM à l'aide de plusieurs critères d'évaluations.



(a) (b)

FIGURE 5.1 – Environnement de test : (a) simple, (b) complexe

5.2 Analyse des résultats

Pour commencer, il est important de comprendre la nécessité d'avoir recours à des approximations pour obtenir des résultats exploitables. Observons les deux images 5.2a et 5.2b. Pour les deux images, les points verts correspondent à la trajectoire du robot, les points rouges représentent la trajectoire estimée du robot et enfin, les boîtes bleues sont la vérité terrain garantie issue de l'algorithme IAPT. Notons que la hauteur des boîtes bleues renseigne sur l'incertitude associée à l'orientation du robot (i.e. plus la boîte est haute, plus l'incertitude est grande). De même, la taille de la boîte renseigne sur les incertitudes associées à la position du robot. Nous remarquons immédiatement que la taille des boîtes bleues de l'image 5.2a sont significativement plus grandes que celles de l'image 5.2b. Rappelons que ces boîtes sont issues de la résolution du CSP 3.6 présenté dans le chapitre 3. Les variables utilisées dans ce CSP sont les suivantes : $\mathbf{q}$,$\{y_i$,γ_i,$\mathbf{w}_i\}_{\forall i}$. Nous en déduisons que la taille des boîtes est directement liée aux incertitudes affectant chacune des variables susmentionnées. Ainsi, détaillons rapidement ces sources d'incertitudes :

Incertitude sur $\mathbf{q}$: Comme expliqué par le pseudo-code 1, le CSP 3.6 reçoit en entrée une estimation $[\mathbf{q}]$, plus ou moins pessimiste, de la posture du robot à l'instant k à l'aide de sa posture précédente à l'instant $k-1$. Pour réduire cette incertitude, nous chercherons à mettre au point une prédiction de posture plus précise.

Incertitude sur y_i et γ_i : Dans le chapitre 4 nous détaillons le pipeline de détection de cônes dans un nuage de points utilisé pour détecter des amers dans l'environnement. Nous expliquons aussi comment cette détection permet de mesurer une distance y_i et un angle γ_i.

Pour résumer, c'est la précision du calcul du centroïde associé à chaque cluster détecté qui conditionne la taille des intervalles $[y_i]$ et $[\gamma_i]$. Pour réduire ces incertitudes, nous chercherons à calculer l'emplacement de ce centroïde le plus précisément possible avec une approche similaire à celle présentée dans l'annexe A.6.

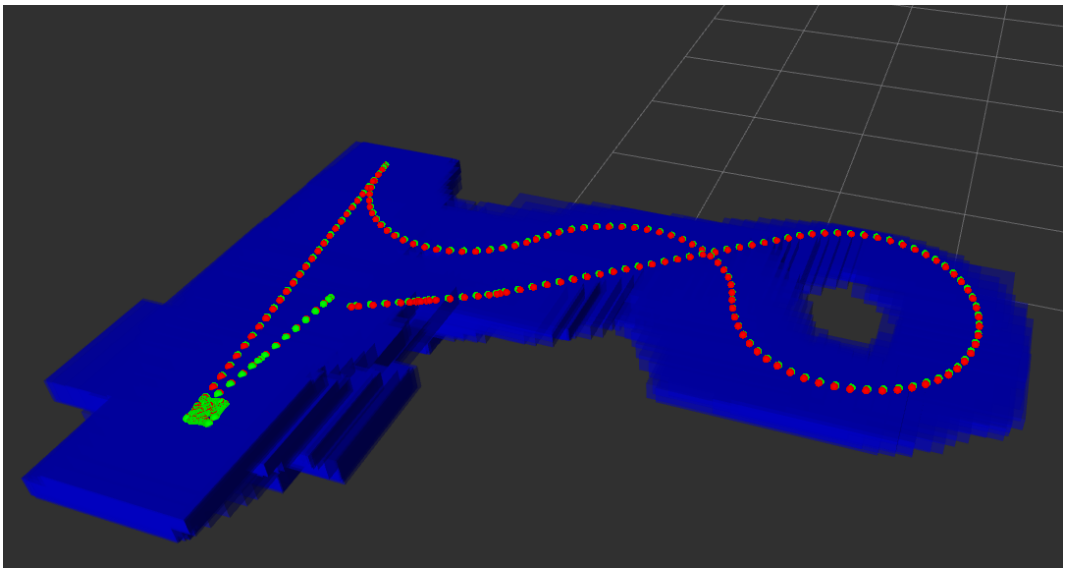
Incertitude sur $\mathbf{w}_i$: Le chapitre 4 nous apprend qu'une carte d'amers est obtenue à l'aide du CSP 3.4 et d'une matrice de distance contenant l'ensemble des mesures de distance plots à plots. La taille des boîtes $[\mathbf{w}_i]$ associé à chaque amer dépend uniquement de la précision de ces mesures. Pour réduire cette incertitude, nous chercherons à mesurer le plus précisément possible ces distances.

Nous discuterons dans la section suivante des améliorations permettant de passer du cas de figure 5.2a au cas 5.2b. Concentrons-nous maintenant sur l'analyse des résultats. Deux approches ont été retenues : un critère d'évaluation global et relatif. Le critère d'évaluation global est simple, il consiste à vérifier que la trajectoire estimée du robot trouve effectivement dans la vérité terrain garantie. Si ce n'est pas le cas, nous comptons le nombre d'erreurs et nous mesurons la distance entre l'estimation de posture et la boîte $[\mathbf{q}]$. Comme expliqué dans le chapitre 2, cette approche globale a pour défaut d'avoir une "mémoire". C'est-à-dire qu'une erreur à un certain pas de temps aura un impact sur les estimations futures, même si celles-ci sont cohérentes. Par conséquent, nous avons aussi implémenté un critère relatif, où nous vérifions que l'estimation de mouvement effectué entre deux pas de temps d'après un algorithme de SLAM se trouve effectivement dans l'enveloppe convexe des boîtes $[\mathbf{q}(k-1)]$ et $[\mathbf{q}(k)]$. D'après l'évaluation globale, dans le cas de figure 5.2b, nous constatons 6 erreurs de la part de l'algorithme de SLAM. Ces erreurs sont en réalité dues à une erreur qui s'est propagée sur les itérations suivantes. En revanche, le critère relatif ne relève aucune erreur, car celui-ci n'est pas assez restrictif pour des raisons que nous exposerons dans la section suivante. Pour finir, concernant les prédictions de posture issues d'un algorithme de SLAM qui se trouvent effectivement dans la boîte $[\mathbf{q}]$ calculée à l'aide de l'algorithme IAPT. Nous pouvons garantir, grâce à ce projet de recherche, que cette posture est cohérente. De plus, les incertitudes qui accompagnent cette prédiction sont clairement identifiées et ne sont autres que la boîte $[\mathbf{q}]$.

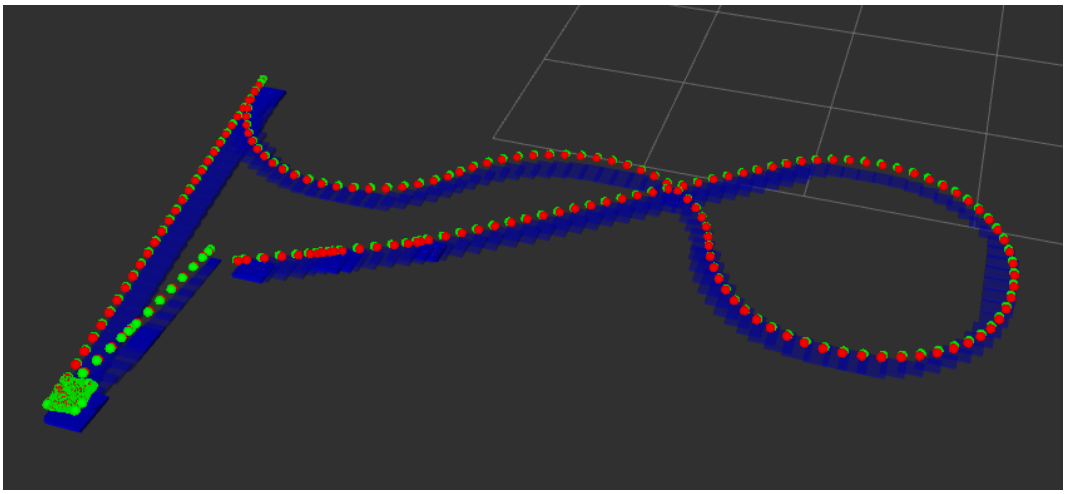
5.3 Pistes d'amélioration

Revenons à l'image 5.2a, ce résultat a été obtenu en effectuant une localisation pessimiste sur plusieurs points. Premièrement, l'étape de prédiction (cf. algorithme 1) de la nouvelle posture $[\mathbf{q}(k)]$ est estimée en ajoutant, à chaque itération, le plus grand déplacement que le robot peut effectuer entre deux pas de temps. Cela a pour intérêt de garantir que le robot reste dans la boîte $[\mathbf{q}(k)]$ à chaque itération. Il est possible d'améliorer cette prédiction en estimant le mouvement effectué par le robot à l'aide d'un capteur Imu. Cette approche appliquée dans le cas de figure 5.2b permet une estimation de $[\mathbf{q}(k)]$ moins pessimiste. Deuxièmement, nous avons constaté que le calcul approximatif des centroïdes associés à chaque clusters introduit d'importantes incertitudes. Comme nous le constatons sur l'image 5.2a, l'accumulation de toutes ces incertitudes rend le résultat trop pessimiste, ce qui empêche l'évaluation des performances d'un algorithme de SLAM. Pour obtenir l'image 5.2b nous sommes donc partis du principe que nous arrivions à calculer avec une haute précision le centroïde des amers détectés dans l'environnement. Ainsi, une première piste d'amélioration est de réduire au maximum les incertitudes qui affectent les variables d'entrée du CSP 3.6. Cela consiste à améliorer la mise à jour

de $[\mathbf{q}(k)]$ entre deux itérations de l'algorithme IAPT. Il faudra aussi améliorer le calcul du centroïde des clusters segmentés. Enfin, il faudra améliorer les mesures de distance cône à cône lors de la création d'une carte d'amer. Un dernier point d'amélioration est constaté grâce à l'image A.4, nous observons un passage de haute incertitude qui correspond au demi-tour du robot au fond d'un couloir. Pendant ce passage, aucun amer n'a pu être détecté pour contracter les postures $[\mathbf{q}(k)]$. Nous apprenons grâce à [17] qu'il est possible de procéder à une rétro-propagation de contraintes sur les postures précédentes une fois que nous réussissons à relocaliser le robot. Il sera donc bénéfique d'implémenter ce principe.



(a)



(b)

FIGURE 5.2 – (a) : Illustration d'une localisation trop pessimiste, et (b) une localisation dans les meilleures conditions possible

Conclusion

Grâce à ce projet de recherche, nous avons pu découvrir la raison d'être des algorithmes de SLAM : simultanément cartographier et localiser un robot dans un environnement inconnu. Nous avons constaté qu'il était difficile d'évaluer les performances de ces algorithmes, car cela nécessite de comparer les prédictions de postures issues de ces algorithmes avec une vérité terrain dont nous n'avons évidemment pas accès. Ce projet de recherche a donc pour objectif de fournir une alternative : une vérité terrain garantie, basée sur l'analyse par intervalles dans un contexte d'erreurs bornées, uniquement à partir de donnée Lidar. Nous avons donc proposé un algorithme de suivi de posture par analyse par intervalles (IAPT) utilisant des amers présent dans l'environnement pour localiser le robot à chaque mesure du capteur Lidar. Pour le développement de ce projet, nous avons utilisé le middleware ROS et plus particulièrement le logiciel de simulation Gazebo dans lequel nous avons simulé un environnement ainsi qu'un robot terrestre équipé d'un capteur Lidar. Ensuite, nous avons traité les nuages de points issus de ce capteur afin d'y détecter les amers utilisés par la localisation. Enfin, nous avons implémenté l'algorithme IAPT qui est chargé de calculer les ensembles de postures possibles du robot à chaque prise de mesures. Cette vérité terrain garantie ainsi obtenue a ensuite été utilisée pour évaluer les performances d'un algorithme de SLAM dans plusieurs scénarios de test. Lors de cette évaluation, nous avons proposé plusieurs critères complémentaires. Enfin, nous avons proposé un certain nombre de pistes d'améliorations après avoir identifié les limitations de ce projet.

Pour conclure ce rapport, je tiens à remercier M. Guyonneau, d'une part pour l'opportunité de travailler sur ce projet de recherche ainsi que de multiples autres projets qui m'ont passionné, et d'une autre part pour son encadrement, son support et ses conseils qui m'ont beaucoup apporté en enrichissant grandement ma formation dans cet établissement. Enfin, je remercie le Laris [1] pour m'avoir accueilli ainsi que mes collègues Arthur, Ahmed, Clémence, Emma et Yann pour leur compagnie chaleureuse.

1. *Laboratoire Angevin de Recherche en Ingénierie des Systèmes*

Annexe A

Annexes

A.1 Mesurer une vérité terrain

Nous pouvons mesurer une vérité terrain à l'aide d'une capture de mouvement à partir d'une caméra dont nous connaissons précisément la posture et les caractéristiques (i.e. les paramètres intrinsèques qui sont internes à la caméra, et les paramètres extrinsèques qui peuvent varier suivant la posture de la caméra dans l'espace de travail). Pour le dépôt issu de la publication [28], les positions de la caméra montée sur le robot ont été obtenues à partir d'un système de capture de mouvement de haute précision avec huit caméras de suivi à haute vitesse (100 Hz). Nous constatons donc que le recours à cet équipement est fastidieux pour l'exploration de tout un bâtiment. En effet, le robot se rendrait dans plusieurs salles, chacune nécessitant ce type d'installation. Autre exemple, une vérité-terrain convenable peut être créée en utilisant des scans alignés manuellement et en connaissant la structure de l'environnement dans lesquels ces scans ont été acquis. L'article [29] détaille cette méthode qui consiste à créer une carte de référence à l'aide de la CAO et des plans d'un bâtiment. Nous pouvons ensuite utiliser l'algorithme MCL pour mesurer le degré de corrélation entre les données réelles (i.e. la carte obtenue par CAO) et l'état prédit (i.e. la carte obtenue par SLAM). Cette approche nécessite donc des plans et l'expertise d'un opérateur pour être réalisable. Enfin, il est possible de créer un modèle du robot et de ces capteurs à l'aide d'un simulateur e.g. Gazebo. Si la simulation est consciencieusement réalisée [4], alors la carte générée par le modèle est proche de la réalité. Nous pouvons ainsi comparer la prédiction avec la vérité terrain à laquelle nous accédons facilement dans une simulation. L'inconvénient de cette approche est qu'il est nécessaire de créer un jumeau numérique rigoureux pour que la comparaison soit pertinente.

A.2 Exemple d'utilisation de l'analyse par intervalles pour calculer la position d'un obstacle

En guise d'exemple concret, utilisons l'analyse par intervalles pour calculer la position d'un obstacle. Référons-nous à la figue A.1a dans laquelle nous observons un robot muni d'un capteur lui permettant de mesurer la distance qui le sépare d'un obstacle. Puisque toutes mesures sont inévitablement entachées d'une certaine erreur, nous associons à la mesure y, la vérité terrain $\hat{y}$ qui

correspond à la véritable mesure. Nous appliquons la même terminologie à l'estimation de posture $\mathbf{q}=(x_1,x_2,\theta)$ du robot a qui nous associons la vérité terrain $\hat{\mathbf{q}}=(\hat{x_1},\hat{x_2},\hat{\theta})$. L'analyse par intervalles permet de travailler dans un contexte d'erreur bornées, c'est-à-dire qu'il nous est normalement possible d'estimer toutes les sources d'erreur susceptible d'influencer la mesure. Ici, nous avons quatre paramètres incertains, la position (x_1,x_2) du robot, son orientation θ et enfin, la mesure y de distance. Nous associons donc une incertitude δ (cf. image A.1b) à chacun de ces paramètres. Cela nous permet de créer des intervalles dont nous sommes absolument sûr qu'ils contiennent la vérité terrain. Nous construirons donc notre problème de façon à satisfaire les équations suivantes :

$$\begin{aligned}
&\hat{x_1}\in[x_1-\delta_{x_1},x_1+\delta_{x_1}]\Leftrightarrow[\underline{x_1},\overline{x_1}]\Leftrightarrow[x_1]\\
&\hat{x_2}\in[x_2-\delta_{x_2},x_2+\delta_{x_2}]\Leftrightarrow[\underline{x_2},\overline{x_2}]\Leftrightarrow[x_2]\\
&\hat{\theta}\in[\theta-\delta_{\theta},\theta+\delta_{\theta}]\Leftrightarrow[\underline{\theta},\overline{\theta}]\Leftrightarrow[\theta]\\
&\hat{y}\in[y-\delta_{y},y+\delta_{y}]\Leftrightarrow[\underline{y},\overline{y}]\Leftrightarrow[y]
\end{aligned} \tag{A.1}$$

En guise d'exemple, associons aux δ une valeur arbitraire que nous relevons dans une fiche technique ou en l'estimant empiriquement : $\delta_{x_1}=\delta_{x_2}=\delta_y=0.01m$ et $\delta_\theta=0.01°rad$. Nous disposons enfin de tous les éléments nécessaires pour estimer les coordonnées de la boîte $[\mathbf{w}]$ contenant à coup sûr l'obstacle détecté. À l'aide de relation trigonométrique simple, nous pouvons exprimer $[\mathbf{w}]$ de cette manière :

$$[\mathbf{w}]=\begin{pmatrix}[w_1]\\[w_2]\end{pmatrix}=\begin{pmatrix}[y]\times sin([\theta])+[x_1]\\[y]\times cos([\theta])+[x_2]\end{pmatrix} \tag{A.2}$$

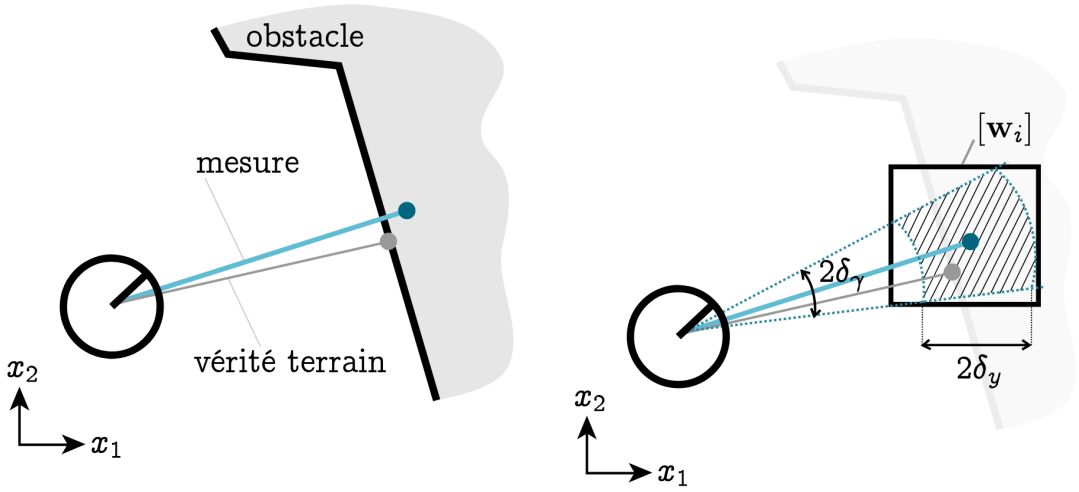
Admettons que nous estimons la position du robot aux coordonnées $(x_1,x_2)=(1,1)$, qu'il est orienté de $\theta=\frac{\pi}{4}$ et que la mesure de distance vaut $y=1m$, nous pourrons calculer les dimensions de la boîte $[\mathbf{w}]$:

$$\begin{aligned}
[\mathbf{w}]&=\begin{pmatrix}[w_1]\\[w_2]\end{pmatrix}=\begin{pmatrix}[\underline{w_1},\overline{w_1}]\\[\underline{w_2},\overline{w_2}]\end{pmatrix}=\begin{pmatrix}[\underline{y}\times sin(\underline{\theta})+\underline{x_1},\overline{y}\times sin(\overline{\theta})+\overline{x_1}]\\[\underline{y}\times cos(\underline{\theta})+\underline{x_2},\overline{y}\times cos(\overline{\theta})+\overline{x_2}]\end{pmatrix}\\
AN:[\mathbf{w}]&=\begin{pmatrix}[1.6830,1.7313]\\[1.6830,1.7313]\end{pmatrix}
\end{aligned}$$

Cette approche nous permet de garantir que la véritable mesure de l'obstacle se trouve à coup sûr dans la boîte [1.6830,1.7313],[1.6830,1.7313]. Notons que la taille de cet intervalle est directement liée aux incertitudes affectant la mesure de la posture du robot et de la mesure issue de son capteur.

A.3 ROS Middleware

ROS est une plateforme de logiciel libre à disposition des utilisateurs qui souhaiteraient développer des applications robotiques sous le système d'exploitation Linux. ROS est composé d'outils de compilation sur-mesure pour distribuer des logiciels facilement. En effet, il est possible de construire des "packages" contenant des scripts en C++ ou en python. Ces scripts permettent de lancer des applications, des algorithmes ou d'autres scripts supplémentaires pour interfacer ROS avec des robots. Dans une dynamique de partage open source, les utilisateurs et développeurs ROS



(a) Ce schéma représente un robot positionné dans un plan 2D et orienté d'un angle θ par rapport à l'axe x_1. Le robot effectue une prise de mesure y à l'aide d'un capteur monté sur l'avant du robot afin de mesurer la distance qui le sépare d'un obstacle. Cette mesure en bleu étant affecté par un certain nombre d'incertitude. Nous avons représenté en gris le véritable emplacement de l'obstacle : la vérité terrain. (source image : [11])

(b) Sous l'hypothèse d'erreurs bornées, nous connaissons les dimensions de la boîte $\mathbf{q}$ contenant à coup sûr la posture du robot, ainsi que l'incertitude δ_y qui affecte la mesure de distance issue du capteur du robot. Dans ces conditions, il est possible d'estimer les coordonnées d'une boîte contenant à coup sûr l'obstacle. (source image : [11])

FIGURE A.1 – Évaluation garantie de la position d'un obstacle sous l'hypothèse d'erreurs bornées

mettent parfois à disposition leurs codes et algorithmes. Cela nous permet d'avoir facilement accès à différents "plugins", ces morceaux de code servent à simuler le fonctionnement d'une large variété de capteurs, accélérant considérablement la simulation du projet.

Le grand intérêt de ROS est donc d'offrir une architecture de communication inter-processus relativement simple : sous ROS, les processus sont appelés des "nodes" et communiquent entre eux via des "topics". Ces communications sont orchestrées par un "master", qui se charge d'acheminer les messages publiés par des "publishers" (des nodes qui publient des données) aux différents "subscribers" (des nodes qui reçoivent des données). Ces messages sont standardisés ce qui rend la distribution des contributions de la communauté extrêmement simple. Ainsi, nous utiliserons le middleware et le simulateur 3D Gazebo pour accélérer le développement de ce projet de recherche.

A.4 Algorithme lié au contracteur de distance C_{dst}

Algorithm 2: Contracteur $C_{dst}([d],[\mathbf{a}],[\mathbf{b}])$

Data: $[d],[\mathbf{a}]=([a_1],[a_2]),[\mathbf{b}]=([b_1],[b_2])$

// Initialisations;
$[i_1]=[a_1]-[b_1]$;$[i_2]=([i_1])^2$;
$[i_3]=[a_2]-[b_2]$;$[i_4]=([i_3])^2$;
// Propagation arrière;
$[i_5]^*=[i_5]\cap[d]$;
$[i_2]^*=[i_2]\cap([i_5]^*-[i_4])$;
$[i_4]^*=[i_4]\cap([i_5]^*-[i_2]^*)$;
$[i_3]^*=[i_3]\cap(\sqrt{[i_4]^*}\cup-\sqrt{[i_4]^*})$;
$[a_2]^*=[a_2]\cap([i_3]^*+[b_2])$;
$[b_2]^*=[b_2]\cap([a_2]^*-[i_3]^*)$;
$[i_1]^*=[i_1]\cap(\sqrt{[i_2]^*}\cup-\sqrt{[i_2]^*})$;
$[a_1]^*=[a_1]\cap([i_1]^*+[b_1])$;
$[b_1]^*=[b_1]\cap([a_1]^*-[i_1]^*)$;
// Propagation avant;
$[i_1]^*=[i_1]^*\cap([a_1]^*-[b_1]^*)$;
$[i_2]^*=[i_2]^*\cap([i_1]^*)^2$;
$[i_3]^*=[i_3]^*\cap([a_2]^*-[b_2]^*)$;
$[i_4]^*=[i_4]^*\cap([i_3]^*)^2$;
$[d]^*=[d]\cap([i_2]^*+[i_4]^*)$;

Result: $[d]^*,[\mathbf{a}]^*=([a_1]^*,[a_2]^*),[\mathbf{b}]^*=([b_1]^*,[b_2]^*)$

A.5 Algorithmes liés au contracteur $C_{\mathbf{w}}$

Algorithm 3: Contracteur $C_{\mathbf{w}}([\mathbf{w}_i],[\gamma_i],[y_i],[\mathbf{q}])$

Data: $[\mathbf{w}_i]=([w_{1_i}],[w_{2_i}]),[\mathbf{q}]=([x_1],[x_2],[\theta]),[\gamma_i],[y_i]$

// Initialisations;
$[\mathbf{q}]^*=[\mathbf{q}]$;$[w_{1_i}]^*=[w_{1_i}]$;$[w_{2_i}]^*=[w_{2_i}]$;$[y_i]^*=[y_i]$;$[\gamma_i]^*=[\gamma_i]$;
// Boucle de contraction;
while Δ *n'est pas vérifiée* **do**
  $([w_{1_i}]^*,[\gamma_i]^*,[y_i]^*,[\mathbf{q}]^*)=C_{w_1}([w_{1_i}]^*,[\gamma_i]^*,[y_i]^*,[\mathbf{q}]^*)$;
  $([w_{2_i}]^*,[\gamma_i]^*,[y_i]^*,[\mathbf{q}]^*)=C_{w_2}([w_{1_i}]^*,[\gamma_i]^*,[y_i]^*,[\mathbf{q}]^*)$;

Result: $[\mathbf{q}]^*,[\mathbf{w}_i]^*=([w_{1_i}]^*,[w_{2_i}]^*),[y_i]^*,[\gamma_i]^*$

Algorithm 4: Contracteur $C_{w_1}([w_{1_i}],[\gamma_i],[y_i],[\mathbf{q}])$

Data: $[w_{1_i}],[\gamma_i],[y_i],[\mathbf{q}]=([x_1],[x_2],[\theta])$

// Initialisations;
$[i_1]=[\theta]+[\gamma_i]$;$[i_2]=sin[i_1]$;
$[i_3]=[y_i]\times[i_2]$;$[i_4]=[i_3]+[x_1]$;
// Propagation arrière;
$[i_4]^*=[i_4]\cap[w_{1_i}]$;
$[i_3]^*=[i_3]\cap([i_4]^*-[x_1])$;
$[x_1]^*=[x_1]\cap([i_4]^*-[i_3]^*)$;
$[y_i]^*=[y_i]\cap(\frac{[i_3]^*}{[i_2]})$;
$[i_2]^*=[i_2]\cap(\frac{[i_3]^*}{[y_i]^*})$;
$[i_1]^*=[i_1]\cap sin^{-1}([i_2]^*)$;
$[\theta]^*=[\theta]\cap([i_1]^*-[\gamma_i])$;
$[\gamma_i]^*=[\gamma_i]\cap([i_1]^*-[\theta])$;
// Propagation avant;
$[i_1]^*=[i_1]^*\cap([\theta]^*+[\gamma_i]^*)$;
$[i_2]^*=[i_2]^*\cap sin([i_1]^*)$;
$[i_3]^*=[i_3]^*\cap([y_i]^*\times[i_2]^*)$;
$[i_4]^*=[i_4]^*\cap([i_3]^*+[x_1]^*)$;
$[w_{1_i}]^*=[w_{1_i}]\cap[i_4]^*$;

Result: $[w_{1_i}]^*,[\gamma_i]^*,[y_i]^*,[\mathbf{q}]^*=([x_1]^*,[x_2],[\theta]^*)$

Algorithm 5: Contracteur $C_{w_2}([w_{2_i}],[\gamma_i],[y_i],[\mathbf{q}])$

Data: $[w_{2_i}],[\gamma_i],[y_i],[\mathbf{q}]=([x_1],[x_2],[\theta])$

// Initialisations;
$[i_1]=[\theta]+[\gamma_i]$;$[i_2]=cos[i_1]$;
$[i_3]=[y_i]\times[i_2]$;$[i_4]=[i_3]+[x_2]$;
// Propagation arrière;
$[i_4]^*=[i_4]\cap[w_{2_i}]$;
$[i_3]^*=[i_3]\cap([i_4]^*-[x_2])$;
$[x_2]^*=[x_2]\cap([i_4]^*-[i_3]^*)$;
$[y_i]^*=[y_i]\cap(\frac{[i_3]^*}{[i_2]})$;
$[i_2]^*=[i_2]\cap(\frac{[i_3]^*}{[y_i]^*})$;
$[i_1]^*=[i_1]\cap cos^{-1}([i_2]^*)$;
$[\theta]^*=[\theta]\cap([i_1]^*-[\gamma_i])$;
$[\gamma_i]^*=[\gamma_i]\cap([i_1]^*-[\theta])$;
// Propagation avant;
$[i_1]^*=[i_1]^*\cap([\theta]^*+[\gamma_i]^*)$;
$[i_2]^*=[i_2]^*\cap cos([i_1]^*)$;
$[i_3]^*=[i_3]^*\cap([y_i]^*\times[i_2]^*)$;
$[i_4]^*=[i_4]^*\cap([i_3]^*+[x_2]^*)$;
$[w_{2_i}]^*=[w_{2_i}]\cap[i_4]^*$;

Result: $[w_{2_i}]^*,[\gamma_i]^*,[y_i]^*,[\mathbf{q}]^*=([x_1],[x_2]^*,[\theta]^*)$

A.6 Calcul d'un centroïde en résolvant un problème de moindres carrés linéaires

L'annexe ci-dessous est basée sur la publication [5].
Nous allons présenter une méthode [5] ne nécessitant pas de connaissance a priori sur le cercle recherché. Nous cherchons le cercle (centre et rayon) qui correspond au mieux, au sens des moindres carrés, au nuage de points. Nous voulons donc minimiser la somme des carrés des distances résiduelles entre chaque point et le cercle. Le problème à résoudre peut s'écrire sous la forme :

$$\underset{\beta,r}{Min}\sum_{i=1}^{m}\{F_i(\beta,r)\}^2$$

avec β le centre du cercle $\beta=(x_c;y_c$, r le rayon et $F_i(\beta,r)$ la distance entre le point i, de coordonnées x_i,y_i, et le cercle, $F_i(\beta,r)=(x_c-x_i)^2+(y_c-y_i)^2-r^2$. Nous obtenons donc le problème suivant :

$$\underset{x_c,y_c,r}{Min}\sum_{i=1}^{m}\{x_c^2+y_c^2-2x_cx_i-2y_cy_i+x_i^2+y_i^2-r^2\}$$

A l'aide du changement de variable suivant : $z_1=2x_c, z_2=2y_c$ et $z_3=r^2-(x_c^2+y_c^2)$ le problème ci-dessus peut s'écrire de la forme :

$$\underset{Z}{Min}\{BZ-D\}^2$$

avec $B=\begin{pmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & 1 \\ x_m & y_m & 1 \end{pmatrix}$, $D=\begin{pmatrix} x_1^2+y_1^2 \\ \vdots \\ x_m^2+y_m^2 \end{pmatrix}$ et $Z=\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$

La solution de ce problème d'optimisation connu est de la forme

$$Z=(B^TB)^{-1}B^TD$$

avec $(B^TB)^{-1}B^T$ la pseudo-inverse de B.
Une fois Z calculé nous trouvons les valeurs de $(x_c, y_c, r$ optimales : $x_c^* = \frac{z_1}{2}$, $y_c^* = \frac{z_2}{2}$ et $r^*=\sqrt{z_3+\frac{z_1^2+z_2^2}{4}}$
Nous obtenons donc les coordonnées et le rayon du cercle qui correspond le mieux au sens des moindres carrés, au nuage de points.

A.7 Estimation initiale de $\mathbf{q}_0$

Au démarrage, le traitement du nuage de points détaillé dans le chapitre 4 nous permet d'identifier une série de clusters, et donc une liste de centroïdes représentant la position d'amers dans le référentiel du capteur Lidar. Rappelons que nous associons aux centroïdes une distance y et un angle γ, il est alors possible de définir la position d'un obstacle $\mathbf{w}_i$ détecté dans le repère du capteur.

$$\mathbf{w}_i=\begin{pmatrix} w_{1_i} \\ w_{2_i} \end{pmatrix}=\begin{pmatrix} y_i\,sin(\gamma_i) \\ y_i\,cos(\gamma_i) \end{pmatrix}$$

Pourtant, nous cherchons à localiser le robot dans le référentiel de la carte d'amers défini par le CSP 3.4. Il est donc nécessaire de renseigner la transformation géométrique (translation + rotation) utilisée pour passer du repère capteur au repère carte. Supposons que nous sachions quels cônes sont utilisés par le CSP 3.4 pour créer son repère local. Ainsi, nous accédons à la position dans le repère capteur des plots $\mathbf{w}_0$ et $\mathbf{w}_1$. À l'aide de $\mathbf{w}_0$ nous définissons la translation nécessaire (i.e. $(-w_{0_1},-w_{0_2})$) pour superposer les deux origines de nos repères. Il ne reste plus qu'à trouver l'angle θ permettant de superposer le cône $\mathbf{w}_1$ dans le repère capteur avec son équivalent dans le repère carte. Or, nous savons que $\mathbf{w}_1$ se trouve sur l'axe des abscisses. Par conséquent, nous définissons $\theta=atan(\frac{w_{1_2}-w_{0_2}}{w_{1_1}-w_{0_1}})$. Pour finir, nous définissons la posture $\mathbf{q}_0$ initiale du robot dans le repère carte :

$$\mathbf{q}_0=\begin{pmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & -\theta \end{pmatrix}\begin{pmatrix} -w_{1_0} \\ -w_{2_0} \\ 1 \end{pmatrix}$$

$\mathbf{q}_0$ ayant été initialisé, nous pouvons passer cette information à l'algorithme IAPT qui se chargera de localiser le robot pour les itérations suivantes.

A.8 Implémentation d'un algorithme de SLAM

Comme présenté dans l'introduction de ce rapport, il existe de nombreux algorithmes de SLAM, toutefois, nous avons décidé d'utiliser l'algorithme LeGO-LOAM [27]. Cet algorithme a pour avantage

de pouvoir estimer une série de postures du robot seulement à l'aide de nuage de points successifs. Nous le rappelons, ces nuages de points sont aussi ceux utilisés par l'algorithme IAPT pour estimer une série de boîtes $[\mathbf{q}(k)]$. Cet algorithme de SLAM permet de créer une carte et de localiser le robot dans celle-ci avec 6 degrés de liberté (i.e. $[t_x,t_y,t_z,\Theta_{roulis},\Theta_{tangage},\Theta_{lacet}]$) à l'aide de 5 étapes :

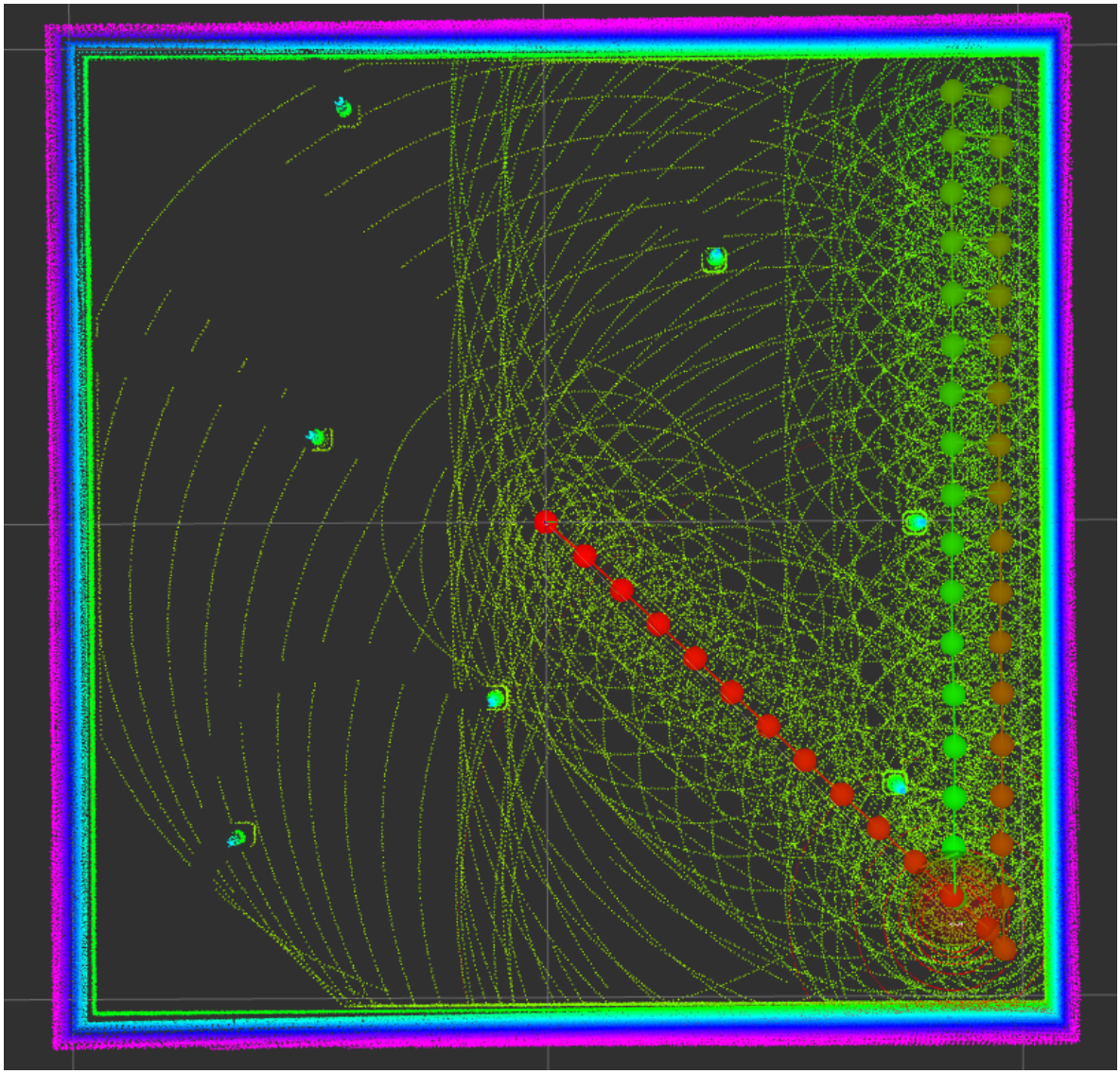
- **Le Filtrage**, comme présenté dans le chapitre 3, nous chercherons à réduire la densité du nuage de points pour accélérer le traitement des données, à l'aide de différents filtres (voxel, outlier removal, ground removal, etc...)
- **La Segmentation** a pour but d'associer les pixels appartenant aux mêmes objets, afin de segmenter ses objets et de les labelliser.
- **L'Extraction des composantes**, le but de cette étape est de distinguer les points appartenant à un plan de ceux qui délimitent ces plans. Cela permet d'obtenir des composantes caractéristiques qui seront utiles pour l'estimation de l'odométrie (c'est-à-dire l'estimation du mouvement effectué par le robot).
- **L'odométrie lidar**, cette double étape estime le déplacement du lidar entre deux scans. La première étape cherche à résoudre 3 des 6 degrés de liberté, $[t_z,\Theta_{roulis},\Theta_{tangage}]$ en faisant correspondre les composantes plan entre elles. Enfin ces valeurs sont utilisés comme contraintes pour estimer les 3 derniers degrés de liberté restant $[t_x,t_y,\Theta_{lacet}]$ en faisant correspondre les composantes bordures entre elles. Ainsi, nous obtenons la transformation avec 6 degrés de liberté effectuée par le robot entre deux scans consécutifs. Cette étape est exécutée 10 fois par seconde, le but étant d'avoir une estimation fréquente de la vélocité (vitesse et direction) avec une faible précision.
- **Cartographie lidar**, en contraste avec l'étape précédente, celle-ci n'est exécutée que deux fois par seconde, une fréquence moindre pour une précision accrue. Cette étape cherche à faire correspondre les nouveaux scans avec la cartographie réalisée jusqu'à présent, en ne gardant que les nouveautés dans l'assemblage de nuages de point qui forment la cartographie.

À ce stade, l'objectif de l'algorithme de SLAM est réalisé, nous possédons une carte de l'environnement dans lequel le robot évolue, ainsi que sa position dans celle-ci à travers le temps (cf. images A.2).

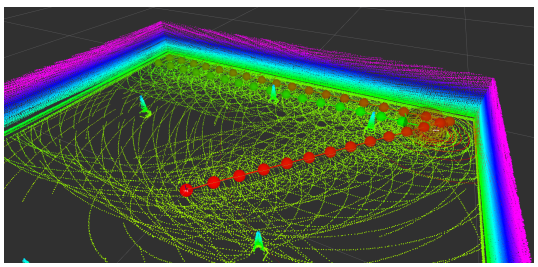
A.9 Visualisation des résultats issues des phases de tests

A.10 Algorithme de clustering euclidien

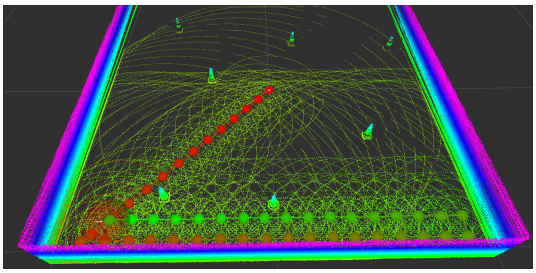
L'algorithme 6 est extrait de l'article [26].



(a)

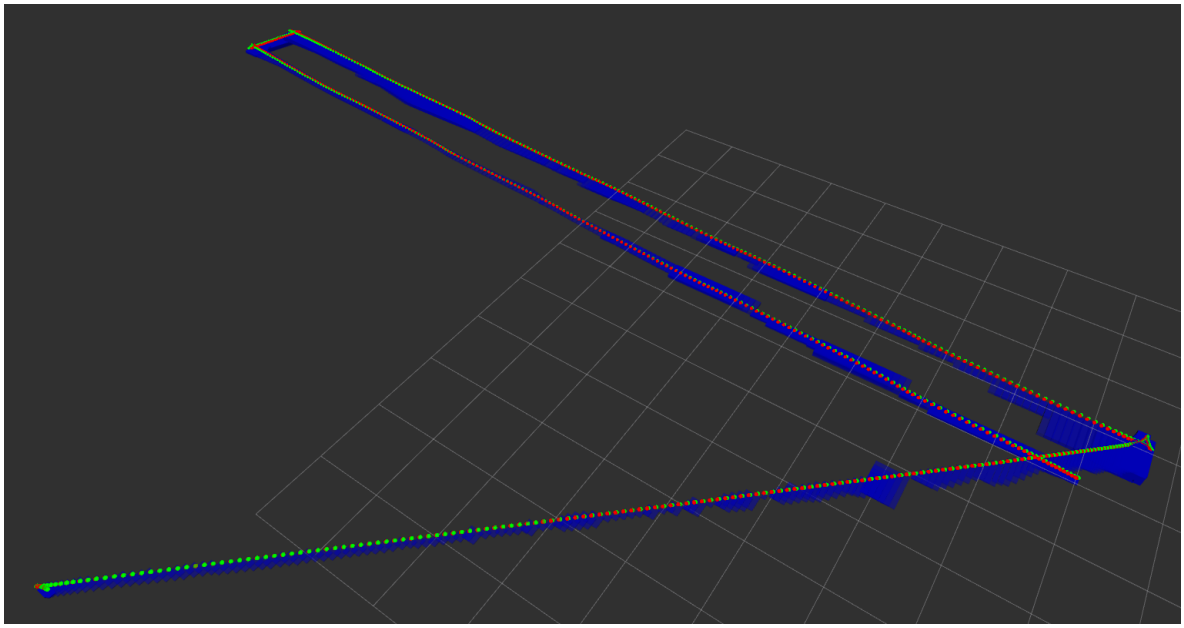


(b) Nous distinguons la trajectoire estimée par l'algorithme de SLAM à l'aide des balises de couleurs allant du rouge au vert.

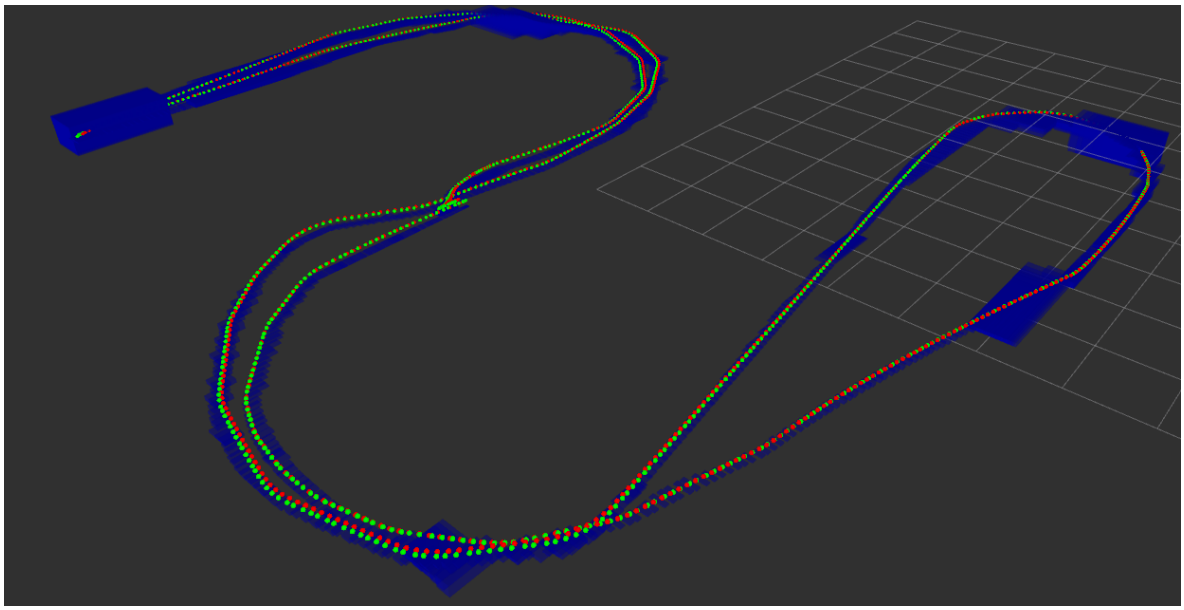


(c) L'algorithme a réalisé une cartographie de l'environnement, nous permettant de distinguer les cônes servant d'amers pour l'algorithme IAPT

FIGURE A.2



(a)



(b)

FIGURE A.3 – (a) : Test dans l'environnement 5.1a, et (b) test dans l'environnement 5.1b

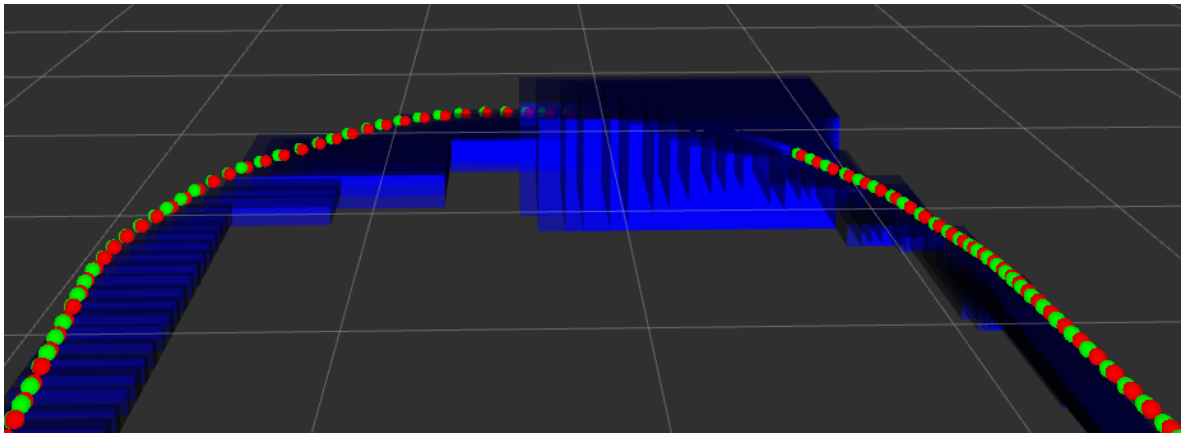


FIGURE A.4 – Test dans l'environnement 5.1b, nous constatons qu'au bout du couloir, les amers se sont retrouvé dans l'angle mort du robot, ce qui a eu pour conséquence d'empêcher la contraction des boîtes [**q**]

.

Algorithm 6: Euclidean Cluster Extraction

Data: P,C,Q,r

P = Kd-tree representation of the input point cloud dataset;
C = Empty list of clusters;
Q = Empty queue of the points that need to be checked;
r = Empirical value of the radius of the search sphere;
foreach $\mathbf{p}_i \in P$ **do**
    add $\mathbf{p}_i$ to the current queue Q;
    foreach $\mathbf{p}_i \in Q$ **do**
        Search for the set P_i^k of neighbouring points of $\mathbf{p}_i$ located in a sphere of radius r;
        if $\mathbf{p}_i \notin P_i^k$ **then**
            add $\mathbf{p}_i$ to Q;
    When the list of all points in Q has been processed;
    Add Q to the list of clusters C, and reset Q to an empty list;
The algorithm terminates when all points $\mathbf{p}_i \in C$;

Result: C

Bibliographie

[1] Stephen Balakirsky et al. « From Simulation to Real Robots with Predictable Results : Methods and Examples ». In : *Performance Evaluation and Benchmarking of Intelligent Systems* (août 2009). doi : 10.1007/978-1-4419-0492-8_6.

[2] Janusz Bedkowski et al. « Open source robotic 3D mapping framework with ROS - Robot Operating System, PCL - Point Cloud Library and Cloud Compare ». In : août 2015. doi : 10.13140/RG.2.1.4869.9605.

[3] Andreas Birk et Max Pfingsthorn. « Simultaneous Localization and Mapping (SLAM) ». In : jan. 2016. doi : 10.1002/047134608X.W8322.

[4] Wolfram Burgard et al. « A Comparison of SLAM Algorithms Based on a Graph of Relations ». In : déc. 2009, p. 2089-2095. doi : 10.1109/IROS.2009.5354691.

[5] Ian D. Coope. « Circle fitting by linear and nonlinear least squares ». In : *Journal of Optimization Theory and Applications* 76 (1993), p. 381-388.

[6] Hugh Durrant-Whyte. « Where am I ? A tutorial on mobile vehicle localization ». In : *Industrial Robot : An International Journal* 21 (avr. 1994), p. 11-16. doi : 10.1108/EUM0000000004145.

[7] Martin A. Fischler et Robert C. Bolles. « Random Sample Consensus : A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography ». In : *Commun. ACM* 24.6 (1981), p. 381-395. issn : 0001-0782. doi : 10.1145/358669.358692. url : https://doi.org/10.1145/358669.358692.

[8] Giorgio Grisetti, Cyrill Stachniss et Wolfram Burgard. « Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters ». In : *Robotics, IEEE Transactions on* 23 (mars 2007), p. 34-46. doi : 10.1109/TRO.2006.889486.

[9] Giorgio Grisetti, Cyrill Stachniss et Wolfram Burgard. « Non-linear Constraint Network Optimization for Efficient Map Learning ». In : *Intelligent Transportation Systems, IEEE Transactions on* 10 (oct. 2009). doi : 10.1109/TITS.2009.2026444.

[10] Jose Guivant et Eduardo Nebot. « Nebot, E.M. : Optimation of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation. IEEE Transactions Robotics & Automation Magazine 17(3), 242-257 ». In : *Robotics and Automation, IEEE Transactions on* 17 (juill. 2001), p. 242-257. doi : 10.1109/70.938382.

[11] Rémy Guyonneau. « Méthodes ensemblistes pour la localisation en robotique mobile ». Theses. Université d'Angers, nov. 2013. url : https://tel.archives-ouvertes.fr/tel-00961501.

[12] Rémy GUYONNEAU et al. « Guaranteed Interval Analysis Localization for Mobile Robots ». In : *Journal on Advanced Robotics* 28 (juill. 2014). DOI : 10.1080/01691864.2014.908742.

[13] Rémy GUYONNEAU et al. « The kidnapping Problem of Mobile Robots : A Set Membership Approach ». In : *7th National Conference on "Control Architectures of Robots"*. Nancy, France, mai 2012. URL : https://hal.archives-ouvertes.fr/hal-03114590.

[14] Wolfgang HESS et al. « Real-Time Loop Closure in 2D LIDAR SLAM ». In : *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, p. 1271-1278.

[15] Berthold HORN, Hugh HILDEN et Shahriar NEGAHDARIPOUR. « Closed-Form Solution of Absolute Orientation using Orthonormal Matrices ». In : *Journal of the Optical Society of America A* 5 (juill. 1988), p. 1127-1135. DOI : 10.1364/JOSAA.5.001127.

[16] Du HUYNH. « Metrics for 3D Rotations : Comparison and Analysis ». In : *Journal of Mathematical Imaging and Vision* 35 (oct. 2009), p. 155-164. DOI : 10.1007/s10851-009-0161-2.

[17] L. JAULIN. « A Nonlinear Set-membership Approach for the Localization and Map Building of an Underwater Robot using Interval Constraint Propagation ». In : *IEEE Transaction on Robotics* 25.1 (2009), p. 88-98.

[18] Fabjan KALLASI, Dario LODI RIZZINI et Stefano CASELLI. « Fast Keypoint Features From Laser Scanner for Robot Localization and Mapping ». In : *IEEE Robotics and Automation Letters* 1 (jan. 2016), p. 1-1. DOI : 10.1109/LRA.2016.2517210.

[19] Michel KIEFFER et al. « Robust autonomous robot localization using interval analysis ». In : *Reliable Computing Journal* 3.6 (2000), p. 337-361. URL : https://hal.archives-ouvertes.fr/hal-00844915.

[20] Kenji KOIDE, Jun MIURA et Emanuele MENEGATTI. « A portable three-dimensional LIDAR-based system for long-term and wide-area people behavior measurement ». In : *International Journal of Advanced Robotic Systems* 16 (fév. 2019). DOI : 10.1177/1729881419841532.

[21] Rainer KÜMMERLE et al. « On Measuring the Accuracy of SLAM Algorithms ». In : *Autonomous Robots* 27 (nov. 2009), p. 387-407. DOI : 10.1007/s10514-009-9155-6.

[22] A.R. LOPEZ. « LiDAR cone detection as part of a perception system in a Formula student car ». Mém. de mast. Barcelona School of Telecommunications Engineering, 2019.

[23] Mohsen MAHRAMI, Md. Nazrul ISLAM et Ramin KARIMI. « Simultaneous Localization and Mapping : Issues and Approaches ». In : *International Journal of Computer Science and Telecommunications* 4 (juill. 2013), p. 1.

[24] A. NEUMAIER. *Interval Methods for Systems of Equations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1991. DOI : 10.1017/CBO9780511526473.

[25] David PROKHOROV et al. « Measuring robustness of Visual SLAM ». In : *2019 16th International Conference on Machine Vision Applications (MVA)*. 2019, p. 1-6. DOI : 10.23919/MVA.2019.8758020.

[26] Radu RUSU. « Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments ». In : *KI - Künstliche Intelligenz* 24 (nov. 2010). DOI : 10.1007/s13218-010-0059-6.

[27] Tixiao SHAN et Brendan ENGLOT. « LeGO-LOAM : Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain ». In : oct. 2018, p. 4758-4765. DOI : 10.1109/IROS.2018.8594299.

[28] Jürgen STURM et al. « Towards a benchmark for RGB-D SLAM evaluation ». In : *RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics : Science and Systems Conf.(RSS)*. Los Angeles, United States, 2011. URL : https://hal.archives-ouvertes.fr/hal-01142608.

[29] Oliver WULF et al. « Benchmarking Urban Six-Degree-of-Freedom Simultaneous Localization and Mapping ». In : *J. Field Robotics* 25 (mars 2008), p. 148-163. DOI : 10.1002/rob.20234.

Résumé — This research topic aims at providing a guaranteed ground truth based on interval analysis in a bounded error context only from Lidar data. This guaranteed ground truth is then used to evaluate the accuracy of SLAM algorithms using original metrics.

Mots clés : Simultaneous Localization And Mapping, Ground truth, Interval Analysis, Bounded Errors, Constraints Satisfaction Problem, ROS, Gazebo.

Polytech Angers - Université d'Angers
62, avenue Notre Dame du Lac
49000 Angers