

UNIVERSITE CLERMONT AUVERGNE Clermont-Ferrand

----

Ecole Universitaire de Physique et d'Ingénierie ----

## MEMOIRE DE STAGE

1<sup>ière</sup> année de Master

----

Spécialité : Mécatronique

Etudiant : NOCHI MAGOUO

**THEME: EVALUATION DES PERFORMANCES  
D'UNE BIBLIOTHEQUE OPEN-SOURCE DE SLAM  
LIDAR**

Date de soutenance : 02/09/2025

Entreprise : Institut Pascal

Encadrant : Pr Paul  
CHECCHIN

# Remerciements

*Ce rapport de stage marque une étape cruciale de mon parcours académique.  
Je tiens à exprimer ma profonde gratitude à toutes les personnes qui m'ont accompagné  
dans cette aventure.*

Je souhaite tout d'abord remercier chaleureusement mon tuteur de stage, **Monsieur Paul CHECCIN**, pour m'avoir accueilli au sein de l'**Institut Pascal** de Clermont-Ferrand. Sa confiance, ses précieux conseils et sa disponibilité ont été déterminants pour le bon déroulement de ce stage. Son expertise et son encadrement bienveillant ont grandement contribué à la réussite de ce projet.

Je tiens à exprimer toute ma reconnaissance à **Monsieur Laurent MALATERE** pour son implication ayant tenu lieu de véritable co-encadrement, ainsi qu'à **Monsieur Benoît THUILLOT** pour son accompagnement précieux tout au long de mon année universitaire. Leurs orientations avisées, leurs retours constructifs et nos échanges scientifiques ont considérablement enrichi mon expérience et mes compétences.

Je remercie également l'**équipe Per-Syst** de l'axe ISPR et l'ensemble du **personnel de l'Institut Pascal** pour leur accueil chaleureux et l'atmosphère de travail stimulante qui ont rendu cette expérience professionnelle particulièrement enrichissante.

Ma reconnaissance va aussi au **CIR ITPS**, dont la bourse a joué un rôle crucial pour améliorer mes conditions d'étude.

Enfin, je réserve un remerciement tout particulier à mes **chers parents**, **Monsieur TOUOYEM APPOLINAIRE** et **Madame MAGOUO CLOTILDE**, pour leur amour inconditionnel, leur soutien indéfectible et leurs encouragements constants tout au long de mes études. Leur confiance en moi a été ma plus grande force. J'exprime également ma profonde gratitude à toute **ma famille** pour son soutien infaillible.

# Résumé du sujet

Ce travail s'inscrit dans le contexte de la robotique mobile, où la localisation et la cartographie 3D fiables restent essentielles, en particulier lorsque le GNSS est indisponible. Nous nous intéressons à la librairie *LiDAR SLAM* de KITWARE implémentée sous ROS 2, dont les performances dépendent fortement d'un grand nombre de paramètres de configuration. L'objectif est double : (i) documenter l'intégration opérationnelle et *reproductible* de la librairie, (ii) analyser et évaluer l'influence des paramètres clés via des métriques de *confiance* (overlap, dispersion, temps de calcul) et de *trajectoire* (ATE/RPE), en scénarios *indoor* et *outdoor*. Pour cela, nous avons conçu une suite d'outils et un orchestrateur automatisant l'exécution des tests, la collecte et la synthèse des résultats. Les expériences montrent une trajectoire **quasi invariante** en *indoor* et une **sensibilité marquée** en *outdoor* aux paramètres d'extraction de points clés et d'optimisation lors de la localisation. Nous en tirons des **lignes directrices** de réglage conciliant précision et coût de calcul.

**Mots-clés** : Robotique mobile ; LiDAR SLAM ; ROS 2 ; Paramétrage ; Évaluation.

# Sommaire du rapport

<b>Remerciements</b>	i
<b>1 Introduction Générale</b>	1
<b>2 Présentation de l’Institut Pascal et de l’axe ISPR</b>	2
2.1 Présentation générale . . . . .	2
2.1.1 Historique . . . . .	2
2.1.2 Implantation géographique . . . . .	3
2.1.3 Activités . . . . .	4
2.1.4 Répartition interne et organigramme . . . . .	5
2.2 Présentation de l’axe ISPR . . . . .	6
2.2.1 L’équipe <i>PerSyst</i> (équipe d’accueil du stage) . . . . .	7
2.2.2 Environnement technique (plateformes et moyens) . . . . .	8
2.2.3 Interactions et positionnement scientifique . . . . .	8
<b>3 Fondements et approches du SLAM LiDAR</b>	9
3.1 Introduction . . . . .	9
3.2 Localisation . . . . .	9
3.2.1 Concepts fondamentaux . . . . .	9
3.2.2 Capteurs utilisés pour la localisation . . . . .	11
3.2.3 LiDAR (Light Detection And Ranging) . . . . .	11
3.3 Cartographie . . . . .	12
3.4 SLAM LiDAR . . . . .	13
3.4.1 Concepts clés du SLAM LiDAR . . . . .	13
3.4.2 Pipeline SLAM LiDAR . . . . .	14
3.4.3 Approches et évolutions majeures du SLAM LiDAR . . . . .	15
3.5 Évaluation : jeux de données et métriques . . . . .	15
3.5.1 Jeux de données usuels . . . . .	15
3.5.2 Métriques usuelles . . . . .	16
<b>4 Présentation de la librairie LiDAR SLAM de Kitware</b>	17
4.1 Présentation du travail à réaliser . . . . .	17

4.2	KITWARE . . . . .	18
4.3	Dépôt Git . . . . .	18
4.4	La librairie LidarSlam de Kitware . . . . .	19
4.4.1	Architecture globale et classes principales . . . . .	20
4.4.2	Pipeline de traitement . . . . .	20
4.5	Wrapper ROS 2 de la librairie . . . . .	21
4.5.1	Architecture générale . . . . .	21
4.5.2	Pipeline du wrapper . . . . .	22
4.5.3	Installation et prise en main de la librairie . . . . .	22
<b>5</b>	<b>Évaluation de la librairie LiDAR-SLAM de Kitware</b>	<b>24</b>
5.1	Introduction . . . . .	24
5.2	Jeux de données et plateforme expérimentale . . . . .	25
5.2.1	ROS 2 bags . . . . .	25
5.2.2	Plateforme logicielle et matérielle . . . . .	25
5.3	Méthodologie d'évaluation . . . . .	26
5.3.1	Facteurs et configurations testées . . . . .	26
5.4	Métriques et calcul . . . . .	27
5.4.1	Métriques internes de confiance ( <code>lidar_slam/Confidence</code> ) . . . . .	28
5.4.2	Outils d'évaluation et automatisation . . . . .	28
5.5	Résultats . . . . .	30
5.5.1	Configuration <i>outdoor</i> . . . . .	30
5.5.2	Configuration <i>indoor</i> . . . . .	36
<b>6</b>	<b>Conclusion Générale</b>	<b>40</b>
<b>Annexes</b>		
<b>A</b>	<b>Approches marquantes du SLAM LiDAR</b>	
<b>B</b>	<b>Fiche d'identité de Kitware, Inc.</b>	
<b>C</b>	<b>Manuel d'installation et de prise en main de LiDAR-SLAM</b>	
C.0.1	Pré-requis et dépendances (Ubuntu + ROS 2 ( dans notre cas Jazzy )) . . . . .	
C.0.2	Téléchargement du dépôt et compilation de la librairie . . . . .	
C.0.3	Compilation des paquets ROS 2 du SLAM . . . . .	
C.0.4	Paramétrage (YAML) et lancement . . . . .	
<b>D</b>	<b>Paramètres de configuration détaillés</b>	

# Table des figures

2.1	Implantations de l’Institut Pascal : campus des Cézeaux, sites hospitaliers et antennes IUT. [1] . . . . .	4
2.2	Plan de situation de l’Institut Pascal au campus des Cézeaux et emplacements des bâtiments associés. [1] . . . . .	4
2.3	Organigramme officiel de l’Institut Pascal : axes de recherche, plateformes/équipements et services de support. [2] . . . . .	6
2.4	Organigramme de l’axe ISPR (responsable d’axe, équipes et liens de pilotage). [3] . . . . .	7
2.5	Plateforme PAVIN (piste d’essais et bâtiments techniques) utilisée par l’axe ISPR, notamment par l’équipe PerSyst, pour la perception et l’évaluation de systèmes embarqués. [4] . . . . .	8
3.1	Illustration d’un repère cartésien ( $x, y, z$ ) associé à un robot. Source : [5] . . . . .	10
3.2	Illustration d’un repère global (rouge) et d’un repère local (vert) attaché à un objet. Source : [6]. . . . .	10
3.3	Schéma illustrant l’incrément d’odométrie entre deux poses successives du robot. Source : [7]. . . . .	11
3.4	Principe de mesure LiDAR par temps de vol : chaque écho correspond à un point du nuage 3D [8]. . . . .	12
3.5	Deux types de représentations cartographiques en SLAM : (a) modèles discrets (voxelisation, OctoMap) et (b) modèles continus (fonctions de distance, Voxblox). . . . .	13
3.6	Pipeline générique d’un SLAM LiDAR : front-end (undistortion, sélection de points, <i>scan matching</i> ) puis back-end (fusion cartographique, détection/fermeture de boucles). D’après Kudan [9]. . . . .	15
4.1	Logo de l’entreprise Kitware [10] . . . . .	18
4.2	Arborescence simplifiée du dépôt <i>LiDAR SLAM</i> de Kitware.[11] . . . . .	19
4.3	Chaîne de traitement de LidarSlam . . . . .	21
4.4	Architecture et pipeline côté ROS 2 : conversions des nuages, noeud SLAM, publications (keypoints, maps, pose, estimations de confiance, frame enregistrée) vers RViz. [12] . . . . .	23

5.1	Orchestrator d'automatisation des tests.	30
5.2	Vue RViz2 de la configuration <i>outdoor</i> (carte et trajectoire estimée).	31
5.3	Vue RViz2 de la configuration <i>indoor</i> (carte et trajectoire estimée).	37

# Liste des tableaux

2.1	Fiche d'identité synthétique de l'Institut Pascal.	3
4.2	Dossiers principaux du dépôt <i>LiDAR SLAM</i> de Kitware.[11]	19
5.1	Séquences ROS 2 utilisées. Les fréquences LiDAR sont calculées à partir du rapport <i>messages/durée</i> .	25
5.2	Plateforme logicielle utilisée pour l'évaluation.	25
5.3	Configuration matérielle de la station de test.	26
5.4	Métadonnées de la séquence <i>outdoor</i> et éléments de configuration.	31
5.5	Tests <b>significatifs</b> par impact relatif sur l' <i>overlap</i> ( $I_{\text{rel}} \geq 5\%$ ).	32
5.6	Tests <b>significatifs</b> par impact relatif sur <b>std_position_error</b> ( $I_{\text{rel}} \geq 5\%$ ).	33
5.7	Tests <b>significatifs</b> par impact relatif sur le <b>computation_time</b> ( $I_{\text{rel}} \geq 5\%$ ).	34
5.8	Meilleures configurations par métrique pour les tests les plus impactants (bag <i>outdoor</i> , recalculation <b>mean</b> ).	34
5.9	Classement par impact relatif (Outdoor) pour la ATE RMSE [m].	35
5.10	Classement par impact relatif (Outdoor) pour la RPE RMSE [m], <b>zéros exclus</b> .	36
5.11	Métadonnées de la séquence <i>indoor</i> ( <i>loop.db3</i> ) et éléments de configuration.	37
5.12	Paramètres les plus impactants pour l' <i>overlap</i> (classement par $I_{\text{rel}}$ ).	37
5.13	Paramètres les plus impactants pour <b>std_position_error</b> (classement par $I_{\text{rel}}$ ).	38
5.14	Paramètres les plus impactants pour le <b>computation_time</b> (classement par $I_{\text{rel}}$ ).	38
A.1	Méthodes LiDAR-SLAM marquantes (cols. comme l'image : Méthode, Année/Référence, Capteurs utilisés, Framework/Algorithme).	...
B.1	Fiche d'identité de Kitware, Inc.	...

# Chapitre 1

## Introduction Générale

**Contexte et enjeux.** La robotique mobile progresse rapidement et s'invite dans des environnements de plus en plus variés. Pour naviguer de façon sûre et autonome, un robot doit estimer sa pose et construire une représentation fiable de son environnement. Cette double exigence , localisation et cartographie, est au cœur des systèmes embarqués modernes, en intérieur comme en extérieur.

**Problématique.** Le SLAM (Simultaneous Localization and Mapping) répond à cette problématique en estimant simultanément trajectoire et carte à partir des capteurs. Parmi les solutions LiDAR, la librairie open-source *Lidar SLAM* de KITWARE est populaire pour sa précision et sa modularité. Son emploi repose toutefois sur une multitude de paramètres (échantillonnage, voisinages, modèles d'optimisation, filtrages) dont les réglages conditionnent directement robustesse, précision et latence. Comprendre l'influence de ces paramètres et établir des bonnes pratiques de configuration est donc essentiel pour garantir des performances reproductibles.

**Cadre du stage.** Ce stage est réalisé au sein du laboratoire Institut Pascal, au sein de l'équipe PerSyst. Il porte sur l'intégration opérationnelle de la librairie *Lidar SLAM* dans les plateformes robotiques de l'équipe.

**Objectifs du stage.** Ce travail vise à fournir une documentation claire pour l'implémentation de la librairie sous ROS 2, du *build* à l'exécution. Il a également pour but de proposer des directives de paramétrage afin d'obtenir des résultats optimaux selon la configuration visée (intérieur, extérieur, scènes plus ou moins structurées) et les critères recherchés.

# Chapitre 2

## Présentation de l’Institut Pascal et de l’axe ISPR

### 2.1 Présentation générale

L’**Institut Pascal** (UMR 6602) est une unité mixte de recherche et de formation interdisciplinaire placée sous la tutelle de l’**Université Clermont Auvergne** et du **CNRS**, avec le **CHU de Clermont-Ferrand** comme tutelle secondaire. Il est membre de *Clermont Auvergne INP* (ISIMA, Polytech Clermont, SIGMA Clermont) et est implanté au *campus universitaire des Cézeaux* à Aubière. Né de fusions successives de laboratoires (2012, 2017, 2021) dans les sciences de l’ingénierie et des systèmes, l’Institut regroupe aujourd’hui *422 personnes* et développe des recherches finalisées autour de trois domaines d’application : l’usine et ses écosystèmes, les transports du futur et l’hôpital du futur en s’appuyant sur cinq axes scientifiques (GePEB, ISPR, M3G, PHOTON, TGI). L’IP est par ailleurs engagé dans plusieurs dispositifs structurants et partenariats (FactoLab avec Michelin, LabEx IMobS, réseau CNRS EquipEx ROBOTEX, LabEx GaNeX et PRIMES, pôles de compétitivité CIMES, AXELERA, MINALOGIC, POLYMERIS, XYLOFUTUR, Institut Carnot MECD), qui contribuent à sa visibilité et à la mutualisation de ses moyens scientifiques.

Une fiche d’identité synthétique de l’Institut Pascal est proposée au Tableau [2.1](#).

#### 2.1.1 Historique

La structuration de l’**Institut Pascal** résulte d’un processus progressif engagé au tournant des années 2000. Un premier cadre est posé par le *CPER* (axe « Machines, Systèmes Performants & Intelligents »), puis par la fédération *TIMS* en **2004**, renforcée en **2006** avec l’arrivée de nouveaux laboratoires et projets. Entre **2007** et **2010**, plusieurs étapes structurantes à savoir : le projet *Innov@Pôle* (2007–2013), la création du *PRES Clermont Université* (2008), *Assises clermontoises de la recherche* (2009) et la réforme du *CNRS* (*ST2I* → *INS2I/INSIS*, 2010) conduisent à la *naissance du projet Institut Pascal* en **2011**. L’unité est ensuite constituée par fusions et réorganisations successives en **2012**, **2017** et **2021**, afin de regrouper, sur le site clermontois, les disciplines des sciences de l’ingénierie et des systèmes et de doter le laboratoire d’une masse critique et d’outils

Caractéristique	Description
Nom officiel	Institut Pascal (UMR 6602)
Statut & tutelles	Unité mixte de recherche CNRS / Université Clermont Auvergne ; tutelle secondaire CHU Clermont-Ferrand ; membre Clermont Auvergne INP
Adresse (siège)	Campus universitaire des Cézeaux, 4 avenue Blaise Pascal, TSA 60026 – CS 60026, 63178 Aubière Cedex, France
Téléphone	(+33) 4 73 40 72 50
Implantations	CHU Gabriel-Montpied (58, rue Montalembert, 63000 Clermont-Ferrand) ; CHU Estaing (1, rue Lucie et Raymond Aubrac, 63000 Clermont-Ferrand) ; IUT Montluçon (7, avenue Aristide Briand, 03100 Montluçon) ; IUT du Puy-en-Velay (8, rue Jean-Baptiste Fabre, 43000 Le Puy-en-Velay)
Direction	Directrice : Evelyne Gil ; Directeur adjoint : Christophe Vial ; Responsable administrative : Pascale Dugat
Effectif total	422 personnes
Axes de recherche	GePEB ; ISPR ; M3G ; PHOTON ; TGI
Genèse (repères)	Projet “Institut Pascal” en 2011 ; fusions/restructurations en 2012, 2017 et 2021
Site web	<a href="http://www.institutpascal.uca.fr/index.php/fr/">http://www.institutpascal.uca.fr/index.php/fr/</a>

**TABLE 2.1** – Fiche d’identité synthétique de l’Institut Pascal.

mutualisés au service de trois domaines d’application : l’usine et ses écosystèmes, les transports du futur et l’hôpital du futur.

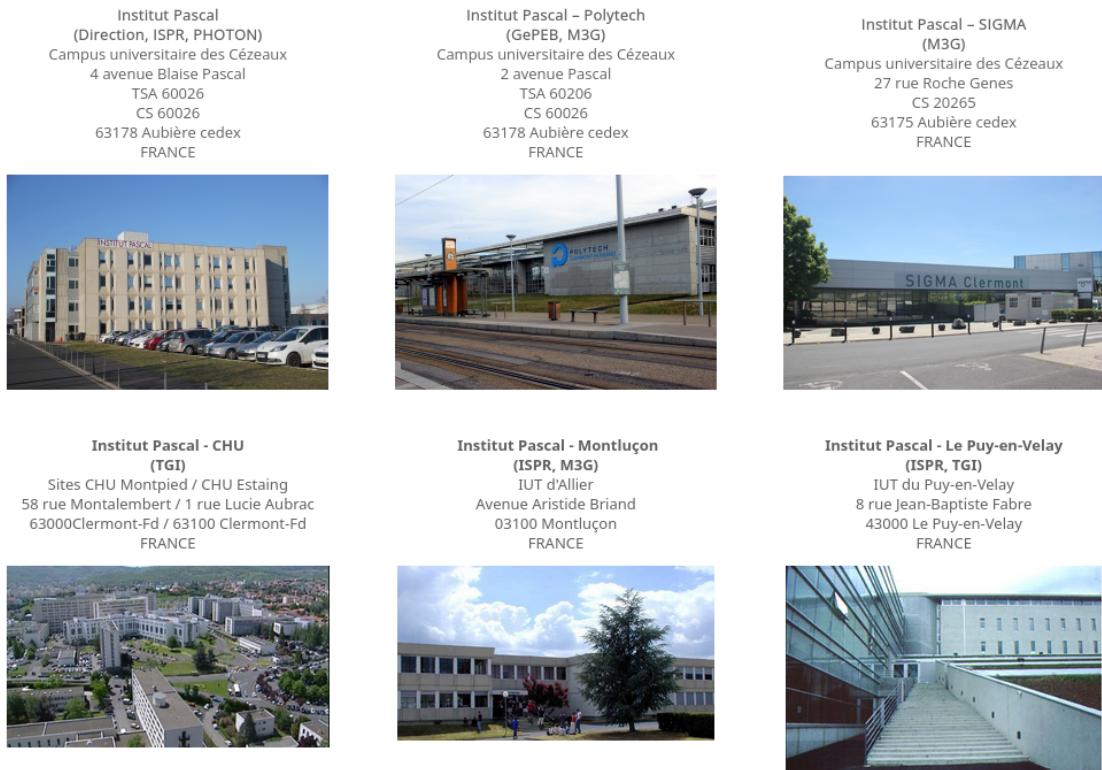
### 2.1.2 Implantation géographique

L’Institut Pascal est implanté au cœur du campus des Cézeaux, à Aubière. Le siège administratif et plusieurs équipes se situent au *4, avenue Blaise-Pascal (TSA 60026 / CS 60026, 63178 Aubière Cedex)*. D’autres implantations complètent cette présence sur le site clermontois et dans l’agglomération :

- **Bâtiment principal (Direction, ISPR, PHOTON)** : 4, avenue Blaise-Pascal, 63178 Aubière Cedex ;
- **Site Polytech (GePEB, M3G)** : 2, avenue Blaise-Pascal, 63178 Aubière Cedex ;
- **Site SIGMA (M3G)** : 27, rue Roche Genès (CS 20265), 63175 Aubière Cedex ;
- **Sites hospitaliers (TGI)** : *CHU Gabriel-Montpied*, 58, rue Montalembert, 63000 Clermont-Ferrand ; *CHU Estaing*, 1, rue Lucie et Raymond Aubrac, 63000 Clermont-Ferrand ;

— **Antennes IUT** : *Montluçon*, 7, avenue Aristide Briand, 03100 Montluçon ; *Le Puy-en-Velay*, 8, rue Jean-Baptiste Fabre, 43000 Le Puy-en-Velay.

Les Figures 2.1 et 2.2 illustrent ces différentes implantations ainsi que le plan d'accès.



**FIGURE 2.1** – Implantations de l’Institut Pascal : campus des Cézeaux, sites hospitaliers et antennes IUT. [1]



**FIGURE 2.2** – Plan de situation de l’Institut Pascal au campus des Cézeaux et emplacements des bâtiments associés. [1]

### 2.1.3 Activités

L’Institut Pascal se distingue par sa vocation *interdisciplinaire*. Ses domaines d’expertise comprennent :

- **Génie des procédés, énergétique et biosystèmes (GePEB)** : étude des procédés physico-chimiques, du traitement de l'énergie et de l'optimisation des systèmes biologiques ;
- **Image, systèmes de perception et robotique (ISPR)** : perception multi-capteurs, vision par ordinateur, robotique mobile, intelligence artificielle et traitement de l'image ;
- **Mécanique, génie mécanique, génie civil et génie industriel (M3G)** : mécanique des structures, matériaux, dynamique des fluides, vibrations, modélisation numérique des ouvrages et procédés industriels ;
- **Photonique, ondes et nanomatériaux (PHOTON)** : physique des lasers, optique non linéaire, nano-photonique, plasmonique et dispositifs micro-opto-électroniques ;
- **Thérapies guidées par l'image (TGI)** : développement de techniques d'imagerie médicale et de thérapies innovantes basées sur la navigation et la robotique médicale.

Ces axes scientifiques s'inscrivent dans trois domaines applicatifs stratégiques :

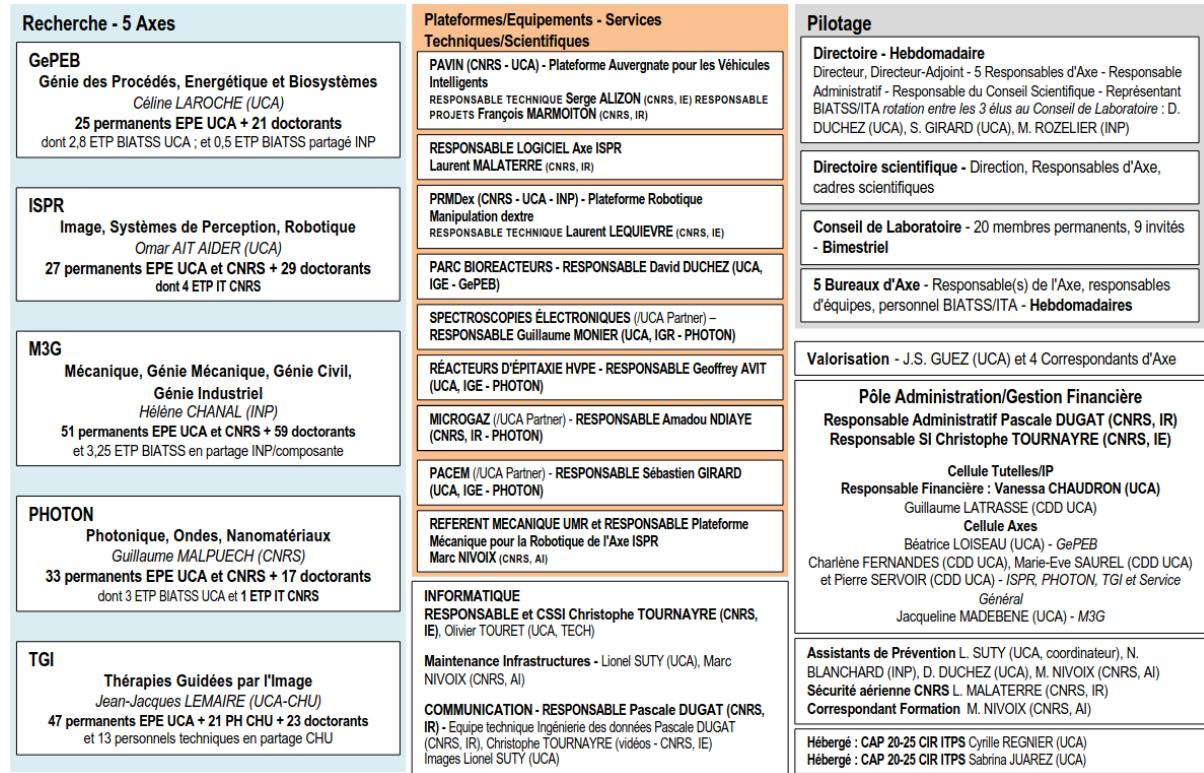
- **Usine du futur** : conception de procédés intelligents, robotisation, maintenance prédictive et optimisation énergétique, avec un intérêt particulier pour la transformation numérique des industries ;
- **Transports et mobilité** : véhicules autonomes, systèmes de navigation et perception environnementale, assistance à la conduite et logistique intelligente ;
- **Hôpital du futur** : imagerie médicale avancée, thérapies mini-invasives guidées par l'image, robotique chirurgicale et télésanté.

L'institut est porteur ou co-porteur de plusieurs projets structurants, notamment le laboratoire d'excellence **IMobS<sup>b</sup>** (mobilité robotisée et coopérative), l'équipement d'excellence **Robotex** pour la robotique, et le laboratoire commun **FactoLab** créé avec *Michelin* pour le développement de l'usine du futur. Il est également partenaire de réseaux de compétitivité régionaux comme *Céréales Vallée* et *ViaMéca*. Ces collaborations académiques et industrielles assurent une visibilité internationale à l'institut.

#### **2.1.4 Répartition interne et organigramme**

L'organisation interne de l'**Institut Pascal** s'articule autour de cinq axes de recherche (GePEB, ISPR, M3G, PHOTON, TGI), chacun piloté par un responsable d'axe. La gouvernance comprend une direction (directrice, direction adjointe), un pilotage scientifique (direction scientifique, responsables d'axes, conseils) et un pôle administratif/gestion financière. Des *services transversaux* assurent l'appui aux équipes : administration/finances, informatique et sécurité des systèmes d'information, communication, maintenance des

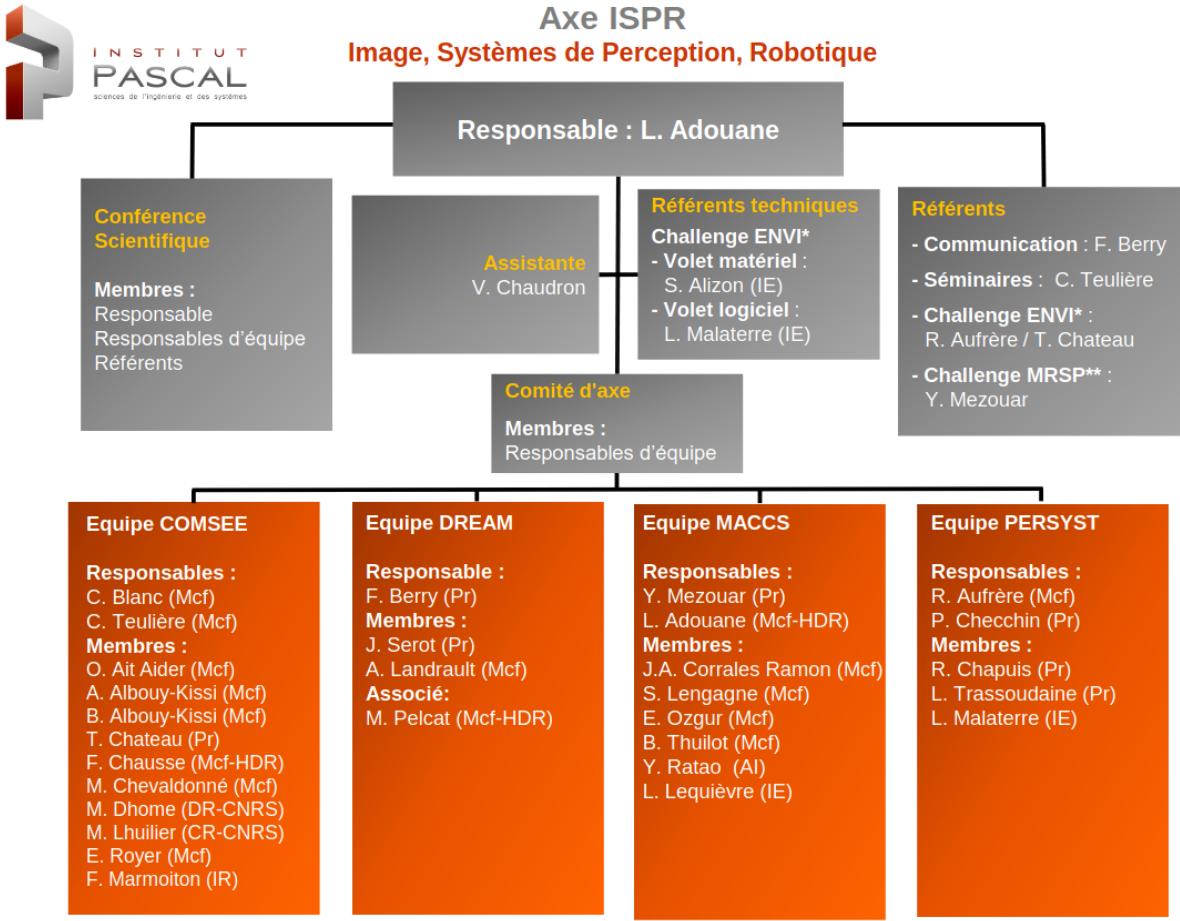
infrastructures, ainsi qu'un ensemble de *plateformes et équipements* techniques mutualisés (par ex. *PAVIN* pour les véhicules intelligents, plateforme robotique *PRIMEX*, *Parc Bioréacteurs*, dispositifs de *spectroscopies électroniques*, réacteurs HVPE, *MICROGAZ*, *PACEM*, etc.). Cette structuration favorise la mutualisation des moyens et l'interdisciplinarité entre axes et services. La Figure 2.3 présente l'organigramme officiel.



**FIGURE 2.3** – Organigramme officiel de l’Institut Pascal : axes de recherche, plateformes/équipements et services de support. [2]

## 2.2 Présentation de l'axe ISPR

Parmi les cinq axes scientifiques de l'**Institut Pascal**, l'axe *Image, Systèmes de Perception et Robotique* (**ISPR**) fédère les activités en vision par ordinateur, perception multi-capteurs et robotique. Les travaux couvrent la modélisation de la scène 3D, la fusion capteurs (caméras, lidars, radars, IMU), la localisation et cartographie (*SLAM*), la robotique mobile et la commande, avec des applications allant de la mobilité intelligente à l'imagerie guidée [13]. L'axe est structuré en équipes thématiques complémentaires, dont *PerSyst*, *ComSee*, *DREAM* et *MACCS*, favorisant des synergies fortes autour de la perception et de la robotique [13, 14]. L'organisation de l'axe est schématisée en Fig. 2.4.



\* Exploration et Navigation pour Véhicules Intelligents (ENVI) : véhicules routiers et urbains

\*\* Manipulation Robotisée et Système de Perception dans l'usine du futur (MRSP)

FIGURE 2.4 – Organigramme de l'axe ISPR (responsable d'axe, équipes et liens de pilotage). [3]

### 2.2.1 L'équipe *PerSyst* (équipe d'accueil du stage)

Au sein de l'ISPR, **PerSyst** (*Systèmes de perception*), sous la direction du Pr Paul CHECCHIN, est l'équipe dans laquelle s'est déroulé mon stage. Ses thématiques portent sur la *localisation* et la *fusion multi-capteurs* (SLAM mono/multimodal, vision–lidar–radar–IMU), la *collaboration multi-robots* et *multi-capteurs* (localisation décentralisée, cartographie partagée) et la *cartographie/modélisation de scènes* pour l'interprétation et la compréhension 3D [15]. Les activités couvrent :

- conception d'algorithmes de SLAM et d'odometrie visuo–inertielle/LiDAR ;
- stratégies de fusion adaptative (pondération, robustesse, métriques locales) ;
- perception pour robots mobiles, véhicules et plateformes instrumentées ;
- outillage d'évaluation et bancs expérimentaux sur capteurs réels et données publiques.

L'équipe publie régulièrement et met en commun jeux de données, outils logiciels et résultats expérimentaux, en interaction avec les autres équipes de l'axe et avec l'écosystème régional/national [15].

## 2.2.2 Environnement technique (plateformes et moyens)

Les recherches de l'ISPR et de PerSyst s'appuient sur des moyens *in situ* et sur des plateformes mutualisées de l'Institut Pascal. En particulier, la **plateforme PAVIN** (*Plateforme Auvergnate pour Véhicules Intelligents*) fournit un site expérimental pour le *développement, l'instrumentation et l'évaluation* des véhicules automatiques et des systèmes de perception embarqués(voir [Fig. 2.5](#)). Elle est utilisée, entre autres, par PerSyst pour des campagnes d'acquisition multi-capteurs, des tests de SLAM LiDAR et des évaluations en conditions réalistes [4].



**FIGURE 2.5** – Plateforme PAVIN (piste d'essais et bâtiments techniques) utilisée par l'axe ISPR, notamment par l'équipe PerSyst, pour la perception et l'évaluation de systèmes embarqués. [4]

## 2.2.3 Interactions et positionnement scientifique

L'axe ISPR interagit étroitement avec les autres axes de l'IP (PHOTON, GePEB, M3G, TGI) et avec les universités hospitalières, dans trois domaines d'application structurants de l'IP : *usine et écosystèmes, transports/mobilité* et *hôpital du futur*. Le positionnement de PerSyst, au croisement SLAM, fusion capteurs et robotique mobile, alimente ces domaines via des démonstrateurs (véhicules instrumentés, robots mobiles, systèmes de navigation et d'imagerie guidée), des collaborations universitaires et des partenariats industriels [13, 16].

# Chapitre 3

## Fondements et approches du SLAM LiDAR

### 3.1 Introduction

La *localisation* et la *cartographie* sont deux briques clés de la perception robotique mobile[17]. La première consiste à estimer en continu la position et l'orientation du robot (sa **pose**) dans un repère donné, tandis que la seconde construit une représentation exploitable de l'environnement [18]. Le *SLAM* (Simultaneous Localization And Mapping) vise à résoudre conjointement ces deux problèmes en limitant la dérive de l'odométrie par la détection de *bouclages* et une optimisation globale (*graph-SLAM*) [19, 20].

Dans ce chapitre, nous présentons : (i) les notions et méthodes de **localisation** (incluant la fusion multi-capteurs : EKF, graph-SLAM, VIO), (ii) les principales **représentations de carte** [21, 22], (iii) les **approches SLAM LiDAR** modernes (ICP/GICP/NDT, méthodes *LOAM-like*, temps continu, surfels, etc. [23–25]), et (iv) les **protocoles d'évaluation** (jeux de données et métriques [26, 27]). Cette structuration s'aligne sur les besoins de ce stage, qui vise l'évaluation rigoureuse d'une bibliothèque open-source de SLAM LiDAR.

### 3.2 Localisation

#### 3.2.1 Concepts fondamentaux

La localisation d'un robot repose sur l'estimation précise de son **état**, exprimé dans un ou plusieurs **référentiels**, et souvent dérivé via des mesures d'**odométrie**.

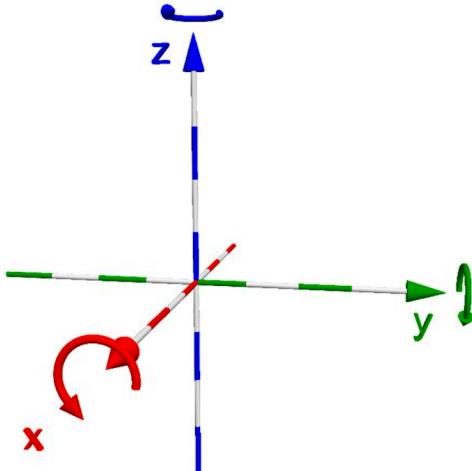
##### a) Pose et état

La *pose* d'un robot correspond à sa **position** et son **orientation** dans un repère donné.

- **Position** : coordonnées  $(x, y, z)$  exprimées dans un repère global ou local.
- **Orientation** : angles d'Euler  $(\phi, \theta, \psi)$  ou représentation quaternion  $(q_w, q_x, q_y, q_z)$ .
- **Vitesse et accélération** : parfois incluses dans le vecteur d'état à estimer.

Comme illustré à la [figure 3.1](#), la pose combine la position dans l'espace et l'orientation du robot dans un repère cartésien  $(x, y, z)$ , ce qui constitue la base des algorithmes de

localisation et de SLAM.



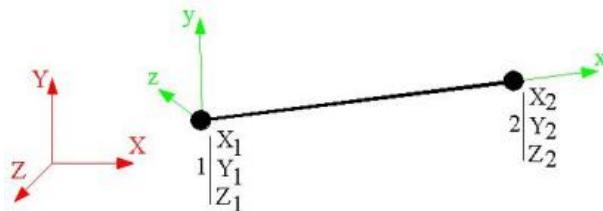
**FIGURE 3.1** – Illustration d'un repère cartésien ( $x, y, z$ ) associé à un robot. Source : [5]

### b) Référentiels

Deux référentiels sont essentiels en robotique mobile :

- **Repère global** : fixe, lié au monde ou à la carte de référence.
- **Repère local** : lié au robot ou à un capteur.

Les transformations entre ces repères sont décrites par des matrices homogènes  $T \in SE(3)$  (pour un mouvement 3D) ou  $SE(2)$  (pour un mouvement plan).



**FIGURE 3.2** – Illustration d'un repère global (rouge) et d'un repère local (vert) attaché à un objet. Source : [6].

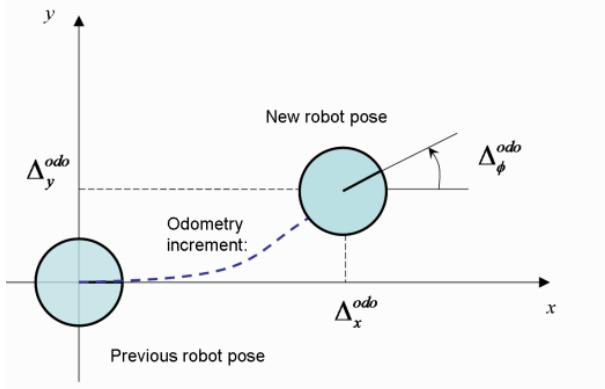
La figure 3.2 montre un exemple où un objet (ou robot) évolue dans un repère global, mais possède aussi son propre repère local, dont la position et l'orientation par rapport au global doivent être estimées.

### c) Odométrie

L'odométrie consiste à estimer l'évolution de la pose d'un robot à partir de mesures internes :

- **Odométrie à roues** : intégration des déplacements mesurés par les codeurs incrémentaux.

- **Odométrie visuelle (VO)** : estimation du mouvement à partir de séquences d'images.
- **Odométrie LiDAR** : recalage de nuages de points successifs.



**FIGURE 3.3** – Schéma illustrant l'incrément d'odométrie entre deux poses successives du robot.  
Source : [7].

Comme illustré à la figure 3.3, l'odométrie fournit un incrément  $(\Delta_x^{odo}, \Delta_y^{odo}, \Delta_\phi^{odo})$  permettant de passer de la pose précédente à la nouvelle pose estimée.

### 3.2.2 Capteurs utilisés pour la localisation

Il existe deux grandes catégories de capteurs utilisés pour la localisation d'un robot mobile. Les **capteurs extéroceptifs** (caméras, télémètres laser LiDAR, radars, GNSS) observent l'environnement et permettent de situer le robot *par rapport* à celui-ci. À l'inverse, les **capteurs proprioceptifs** (odomètres, capteurs inertIELS IMU) mesurent les mouvements *propres* au robot (déplacement, variations d'orientation) et ne donnent pas une localisation absolue [28].

Afin d'améliorer la fiabilité et la précision, il est courant de **combiner** plusieurs capteurs. Cette fusion d'informations, centrale dans les systèmes autonomes, compense les limites de chaque capteur et accroît la robustesse, y compris en cas de défaillance partielle [28].

### 3.2.3 LiDAR (Light Detection And Ranging)

Le LiDAR est un capteur actif essentiel pour la perception 3D en robotique, disponible en deux familles principales :

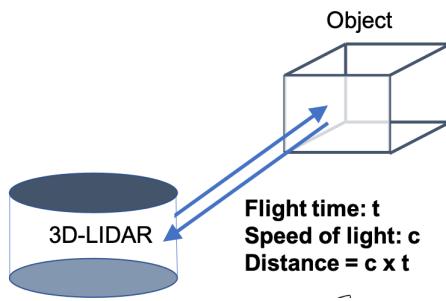
- **LiDAR 2D** : balayage plan (classique en SLAM 2D ou premières générations véhicules).
- **LiDAR 3D** : balayage multi-couches (indispensable en environnements non structurés).

## Principe de fonctionnement

La distance est mesurée par **temps de vol** (*Time of Flight, ToF*) d'une impulsion laser :

$$d = \frac{c \cdot t}{2}, \quad (3.1)$$

où  $d$  est la distance capteur–objet,  $c \approx 3 \times 10^8 \text{ m.s}^{-1}$  la vitesse de la lumière, et  $t$  le temps aller-retour du signal.



**FIGURE 3.4** – Principe de mesure LiDAR par temps de vol : chaque écho correspond à un point du nuage 3D [8].

Un cycle de mesure typique comprend : (i) l'émission d'impulsions (p. ex. 905 nm ou 1550 nm), (ii) la détection des échos (souvent plusieurs retours par impulsion), et (iii) la reconstruction 3D via la cinématique du capteur (miroirs rotatifs, MEMS, ou lignes à avalanche) [28].

## 3.3 Cartographie

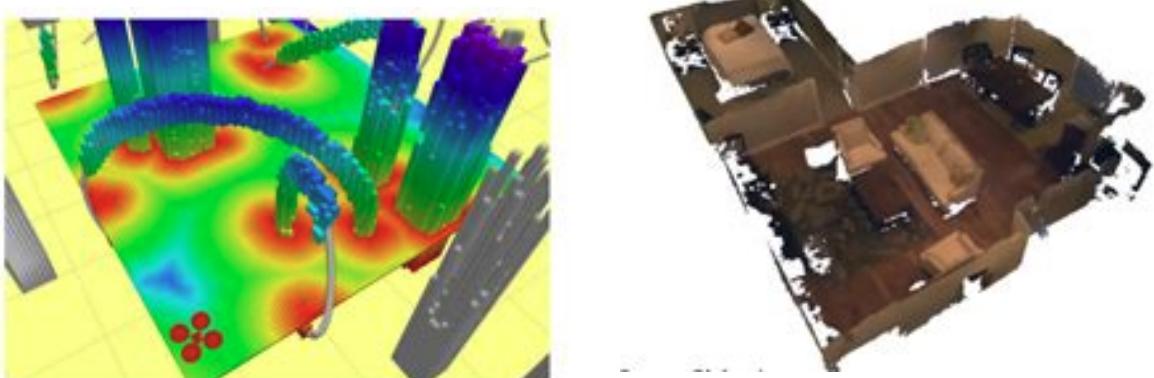
En robotique mobile, la **cartographie** (mapping) désigne la construction d'une représentation exploitable de l'environnement à partir de mesures capteurs (LiDAR, caméras, etc.). Cette carte sert à la **navigation** (planification et évitement), à la **localisation** (relocalisation sur carte) et, plus largement, à la **compréhension de scène**.

Dans le cadre du *SLAM*, la carte est estimée *simultanément* à la pose du robot, et doit rester :

- *cohérente géométriquement* (faible distorsion, gestion des bouclages) [19],
- *utile opérationnellement* (supporte les requêtes de planification, collision, relocalisation),
- *maintenable* (mise à jour incrémentale, fusion multi-passages, gestion mémoire) [18].

On distingue des cartes **locales** (fenêtre glissante ou sous-cartes, rapides à mettre à jour pour l'odométrie) et des cartes **globales** (optimisées après détection de bouclages). Les représentations peuvent être **discretées**, comme les cartes à voxels ou grilles d'occupation (Fig. 3.5a), ou **continues**, comme les champs de distance signés de type TSDF/ESDF

(Fig. 3.5b). Chaque approche présente des compromis en précision, coût mémoire et temps d'accès [21, 22], comme illustré globalement à la figure 3.5.



(a) Carte discrète : grille de voxels (*OctoMap*) [21].

(b) Carte continue : champ de distance signé (*TSDF/ESDF, Voxblox*) [22].

**FIGURE 3.5** – Deux types de représentations cartographiques en SLAM : (a) modèles discrets (voxélisation, OctoMap) et (b) modèles continus (fonctions de distance, Voxblox).

## 3.4 SLAM LiDAR

### 3.4.1 Concepts clés du SLAM LiDAR

Le **SLAM LiDAR** désigne un ensemble de techniques permettant, à partir des seuls nuages de points mesurés par un capteur LiDAR, d'estimer simultanément la trajectoire du robot et de construire une représentation fidèle de l'environnement. Il se distingue ainsi des SLAM visuels ou inertIELS en s'appuyant exclusivement sur des mesures de distance, ce qui le rend robuste aux variations d'éclairage et de texture tout en exigeant des traitements géométriques adaptés.

**1) Représentation des données.** Dans un système SLAM LiDAR, la scène est perçue sous forme d'un **nuage de points 3D**  $\mathcal{P} = \{\mathbf{p}_i \in \mathbb{R}^3\}$ , chaque point étant défini par sa position (et parfois une intensité de retour). Pour réduire la complexité et identifier les éléments discriminants, l'algorithme extrait généralement un sous-ensemble de **keypoints/features** : ce sont des points à forte courbure (arêtes) ou à faible courbure (plans) qui capturent l'ossature géométrique de la scène et facilitent le recalage [23, 29]. Enfin, l'utilisation d'une **grille de voxels** (voxel grid) permet de discréteriser l'espace pour filtrer ou regrouper les points, stocker des statistiques locales (comme dans les méthodes NDT) et accélérer les recherches de correspondances [30, 31]. Ces trois représentations — nuage dense, keypoints et voxels — sont des briques incontournables des pipelines SLAM LiDAR.

**2) Égomotion / Odométrie.** L'**égomotion** (ou **odométrie**) estime le mouvement relatif  $\mathbf{T}_{k \rightarrow k+1} \in SE(3)$  entre deux mesures successives, sans optimiser une carte globale. Deux grandes familles coexistent : (i) *méthodes directes* de type **ICP** (point-à-point / point-au-plan) et variantes **GICP** qui pondèrent par les covariances locales [32–34] ; (ii) *méthodes à features* (appariement arêtes/plans, voire surfels) [23, 24]. Selon le choix de la cible, on distingue **scan-to-scan** (recalage d'un scan courant sur le précédent) et **scan-to-map** (recalage sur une sous-carte locale agrégée) [35].

**3) Chaîne front-end / back-end.** Le **front-end** effectue la pré-intégration (deskewing), l'extraction de features et le recalage local (odométrie). Le **back-end** optimise un **graph de poses** global (bouclages, contraintes IMU/GNSS) par moindres carrés non linéaires (Gauss-Newton/LM), maintenant la *cohérence* de la trajectoire et de la carte [25, 35]. Des schémas **tightly-coupled** LiDAR-IMU (ou LiDAR-IMU-GNSS) améliorent la robustesse et réduisent la dérive [25, 36].

### 3.4.2 Pipeline SLAM LiDAR

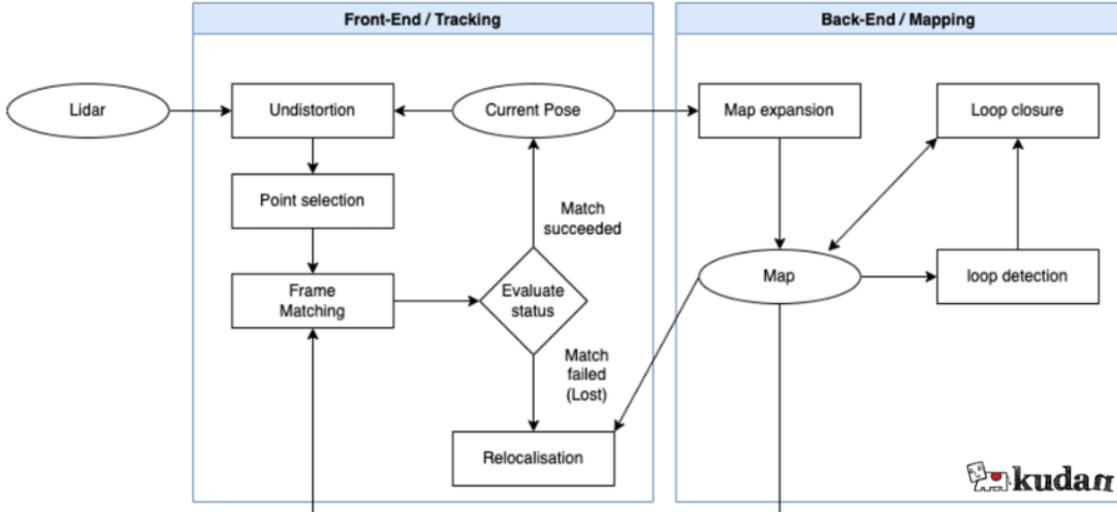
Le pipeline d'un SLAM LiDAR se décompose classiquement en deux étages : un *front-end* (estimation locale de la pose à partir du flux capteur) et un *back-end* (optimisation globale et gestion de la carte). Une vue d'ensemble est résumée à la Figure 3.6 [9].

#### Front-end (tracking).

1. **Undistortion / deskewing** : correction du flou induit par le mouvement pendant un tour de LiDAR à l'aide d'une prédiction de mouvement (IMU/odométrie).
2. **Sélection de points / features** : sous-échantillonnage et extraction (arêtes, plans, points stables), souvent avec *voxelisation* pour limiter le coût.
3. **Appariement (*scan matching*)** : estimation de la pose courante par recalage *scan-to-scan* ou *scan-to-map* (ICP/GICP, NDT ou appariement de features). Succès  $\Rightarrow$  mise à jour de la pose ; échec  $\Rightarrow$  tentative de **relocalisation**.

#### Back-end (mapping).

1. **Expansion de carte** : intégration des points de la trame courante dans la carte locale/globale.
2. **Détection et fermeture de boucles** : reconnaissance d'un lieu déjà visité, puis optimisation globale (graphe de poses) pour corriger la dérive accumulée et rendre la carte cohérente.



**FIGURE 3.6** – Pipeline générique d'un SLAM LiDAR : front-end (undistortion, sélection de points, *scan matching*) puis back-end (fusion cartographique, détection/fermeture de boucles). D'après Kudan [9].

### 3.4.3 Approches et évolutions majeures du SLAM LiDAR

Depuis les premiers recalages *scan-to-scan* (ICP dans les années 1990), le SLAM LiDAR a progressivement évolué : d'abord vers des modèles probabilistes plus robustes (GICP, NDT), puis vers des pipelines *features-based* de type LOAM, séparant odométrie rapide et *back-end* cartographique. Par la suite, des solutions variées ont émergé, exploitant soit des représentations implicites (IMLS), des approches denses à surfels (SuMa), ou encore des graphes de contraintes (Cartographer, HDL Graph SLAM). Plus récemment, l'intégration étroite de l'IMU (voire du GNSS) dans des factor-graphs temps-continu a permis d'améliorer la robustesse et de réduire la dérive (LIO-SAM, FAST-LIO2, CT-ICP, etc.).

Un panorama comparatif des méthodes emblématiques (année, capteurs exploités et principe algorithmique) est présenté dans le **Tableau A.1** de l'**Annexe A**.

## 3.5 Évaluation : jeux de données et métriques

L'évaluation d'un système SLAM LiDAR repose sur des jeux publics garantissant la reproductibilité et sur des métriques normalisées permettant la comparaison inter-méthodes.

### 3.5.1 Jeux de données usuels

Nous retenons quatre jeux de données représentatifs, couvrant des contextes et cinématiques variés.

- **KITTI Odometry** propose des séquences urbaines et suburbaines acquises en véhicule, avec LiDAR 3D et vérité terrain GPS/INS, et constitue la référence pour comparer dérives et fermetures de boucle [26].
- **MulRan** fournit des parcours répétés à travers les saisons ; il est particulièrement adapté à l'étude de la robustesse d'apparence et de la re-localisation globale [37].
- **Newer College** capture des trajectoires pédestres multi-niveaux (escaliers, espaces confinés) à haute cadence LiDAR, ce qui en fait un banc d'essai exigeant pour l'odométrie 3D [38].
- **NCLT** regroupe des acquisitions longue durée et multi-capteurs sur un campus, permettant d'évaluer la dérive à long terme et les stratégies de fusion [39].

### 3.5.2 Métriques usuelles

Les performances sont mesurées par des métriques de trajectoire, de détection de bouclages et de qualité de carte.

- **Erreur de trajectoire** : l'*ATE* (Absolute Trajectory Error) mesure l'écart global après alignement SE(3)/Sim(3), tandis que la *RPE* (Relative Pose Error) quantifie la dérive locale sur un intervalle  $\Delta$  [27].
- **Bouclages et re-localisation** : la qualité de détection se juge par la précision, le rappel et le score F1, avec validation géométrique des contraintes ajoutées au graphe.
- **Qualité de carte** : on considère des distances surface-à-surface (p. ex. Chamfer), le recouvrement, la compacité mémoire (octets / m<sup>2</sup>) et le temps de mise à jour ; pour les grilles d'occupation et TSDF/ESDF, on rapporte aussi l'erreur/probabilité par voxel et l'entropie.

# Chapitre 4

## Présentation de la librairie LiDAR SLAM de Kitware

### 4.1 Présentation du travail à réaliser

La bibliothèque **LiDAR SLAM** développée par **Kitware** est une solution open-source dédiée à la *localisation* et à la *cartographie* 3D en temps réel à partir de données LiDAR, avec la possibilité d'intégrer d'autres capteurs tels que l'IMU ou le GNSS pour améliorer la robustesse et la précision. Elle repose sur un pipeline modulaire inspiré des approches *LOAM-like*, intégrant des méthodes avancées de recalage, de gestion des *keyframes* et de détection de bouclages.

Dans le cadre de ce stage, l'objectif principal est d'**évaluer les performances** de cette bibliothèque afin d'identifier les paramètres et configurations optimaux pour son utilisation dans des applications robotiques développées au sein de l'**Institut Pascal**, et plus spécifiquement par l'équipe **PerSyst**. Cette évaluation doit permettre d'établir des recommandations concrètes en matière de paramétrage et d'intégration dans des plateformes robotiques réelles.

Le travail à réaliser s'articule autour de trois volets complémentaires :

- 1) **Installation et prise en main de la librairie.** La première étape consiste à déployer la bibliothèque *LiDAR SLAM* et son wrapper ROS 2 sur un environnement Ubuntu. Cela inclut l'installation des dépendances (Eigen, PCL, Ceres, etc.), la compilation via `colcon` ainsi que la validation du bon fonctionnement à travers des tests initiaux. Cette phase doit aboutir à la mise en place d'un environnement reproductible et documenté, utilisable par d'autres membres de l'équipe.
- 2) **Étude approfondie du fonctionnement de la librairie.** L'objectif est de comprendre en détail le pipeline interne : extraction de points-clés, estimation d'odométrie, optimisation de graphe de poses et détection de bouclages. Une attention particulière sera portée à l'architecture logicielle (modules C++), ainsi qu'au processus d'intégration avec ROS 2 (nœuds, topics, paramètres YAML). Cette analyse permettra d'identifier le rôle de chaque module et de clarifier la logique de configuration.
- 3) **Campagne de tests et analyse des résultats.** Enfin, différents jeux de données seront utilisés afin d'évaluer les performances de la bibliothèque. Les résultats seront

étudiés à l'aide de métriques de confiance (overlap, temps de calcul, erreurs de pose) ainsi que des métriques standards de SLAM (**ATE**, **RPE**). L'objectif est de comparer plusieurs configurations, d'analyser l'impact des paramètres de configuration ; et de dégager des **directives de réglage pratiques**. Ces recommandations permettront d'orienter les futures utilisations de la librairie dans des projets robotiques de l'Institut Pascal.

Ce chapitre présente la **librairie LiDAR SLAM de Kitware** et son environnement d'utilisation. Après une brève présentation de l'entreprise Kitware et de son dépôt Git associé, nous décrirons l'architecture logicielle de la librairie, ses principales classes et son pipeline de traitement. Enfin, nous présenterons le **wrapper ROS 2** développé pour faciliter son intégration dans des projets robotiques.

## 4.2 KITWARE

Fondée en 1998, **Kitware, Inc.** est une société américaine spécialisée dans le développement de solutions logicielles open-source pour la vision par ordinateur, l'imagerie médicale, la simulation et la visualisation scientifique. L'entreprise collabore avec des partenaires académiques, industriels et gouvernementaux, et est reconnue pour des projets phares tels que *VTK*, *ParaView* et *Slicer*. Sa division *Computer Vision* développe notamment la librairie **LiDAR SLAM** qui constitue l'objet de notre étude.



**FIGURE 4.1** – Logo de l'entreprise Kitware [10]

La **fiche d'identité** de Kitware est présentée en Annexe B.1.

## 4.3 Dépôt Git

La Figure 4.2 illustre l'arborescence (simplifiée) du dépôt *LiDAR SLAM* de Kitware.<sup>1</sup>

---

1. Le dépôt public est référencé en bibliographie sous [11].

```

slam/
├── slam_lib/
├── superbuild/
├── ros_wrapping/
├── ros2_wrapping/
└── paraview_wrapping/
    └── Plugin/
        └── doc/
            └── How_to_SLAM_with_LidarView.md
├── ci/
├── doc/
└── CMakeLists.txt
├── README.md
├── LICENSE
└── CHANGELOG

```

**FIGURE 4.2** – Arborescence simplifiée du dépôt *LiDAR SLAM* de Kitware.[\[11\]](#)

### Dossiers et rôles principaux :

Dossier	Rôle et contenu principal
slam_lib/	Cœur C++ : implémentation des classes, modules, fonctions, etc de la librairie.
ros_wrapping/	Intégration <b>ROS 1</b> : noeuds, messages/paramètres, fichiers launch.
ros2_wrapping/	Intégration <b>ROS 2</b> : noeuds composables, interfaces, paramètres, launch.
paraview_wrapping/	Plugin <b>LidarView/ParaView</b> pour la visualisation et l'exploration.
superbuild/	Scripts CMake « SuperBuild » pour récupérer les dépendances et construire le projet.
ci/	Configuration d'intégration continue.
doc/	Guides et ressources de documentation.
CMakeLists.txt	Configuration CMake racine du projet.
README.md	Présentation, installation et usage.
LICENSE	Licence du projet.
CHANGELOG	Historique des changements et versions.

**TABLE 4.2** – Dossiers principaux du dépôt *LiDAR SLAM* de Kitware.[\[11\]](#)

## 4.4 La librairie LidarSlam de Kitware

**LidarSlam** est une librairie C++ open-source, modulaire et centrée LiDAR, développée par Kitware pour l'odométrie 3D et la cartographie locale/globale avec possibilité de

fusionner des capteurs externes (IMU, GNSS, odométrie roues, caméras) et de faire de l'optimisation de graphe (bouclage) [11].

#### 4.4.1 Architecture globale et classes principales

L'architecture suit une chaîne front-end/back-end : extraction de points-clés, mise en correspondance *scan-to-map*, optimisation locale (Ceres), mise à jour d'une carte voxelisée, et, en option, optimisation de graphe pour corriger la dérive globale [11, 40, 41]. Les principaux blocs (avec quelques classes représentatives) sont :

##### Orchestrator — Slam

API principale qui enchaîne l'ingestion des trames, l'extraction des *keypoints*, l'undistortion (selon le modèle de mouvement), le matching *scan-to-map*, l'optimisation locale, la mise à jour des cartes, la fusion de capteurs et, si activé, le *pose-graph optimization*.

##### Extraction de points-clés — KeypointExtractor, SpinningSensorKeypointExtractor, DenseSpinningSensorKeypointExtractor

Produit des arêtes/plans (et *intensity-edges* si disponibles) robustes pour le recalage.

##### Mise en correspondance — KeypointsMatcher (+ KD-tree)

Recherche de voisins et calcul des résidus géométriques (edge→ligne, plane→plan, ...) pour l'optimisation.

##### Optimisation locale — LocalOptimizer + CeresCostFunctions

Assemble les résidus LiDAR et les contraintes capteurs externes, puis résout un problème non linéaire (Gauss–Newton/Levenberg–Marquardt via Ceres).

##### Capteurs externes — ExternalSensorManagers

Synchronisation, calibration extrinsèque et génération de contraintes (IMU, GNSS, odométrie, caméras, *landmarks*).

##### Carte locale / voxelisation — RollingGrid, VoxelGrid

Maintien de cartes de points-clés par type dans une grille locale glissante (taille de voxel, échantillonnage, vieillissement).

##### Optimisation de graphe (PGO) — PoseGraphOptimizer

Incorpore bouclages et contraintes globales (GNSS, poses externes, *landmarks*) pour corriger la trajectoire et ré-ancrer les cartes.

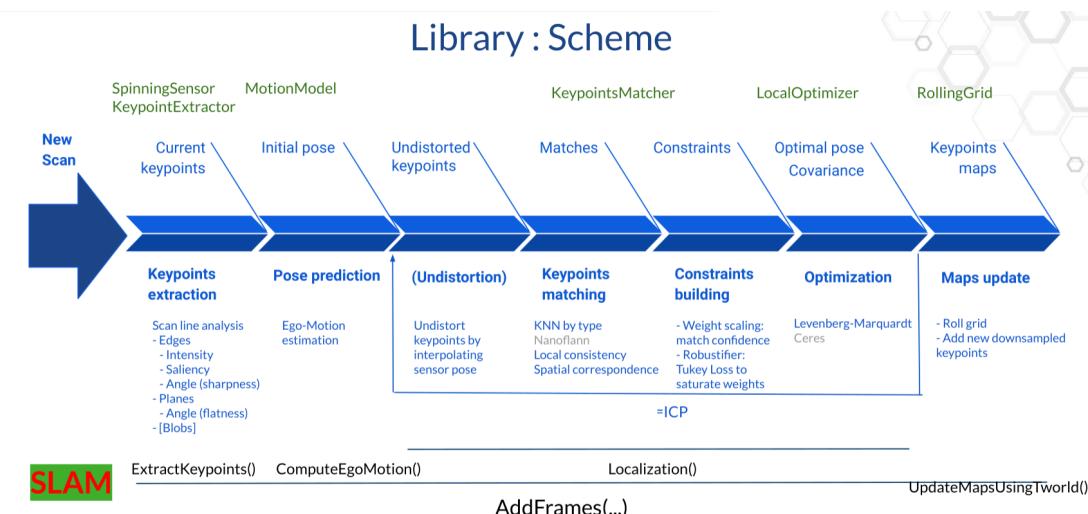
##### Types & utilitaires — LidarPoint, State, Enums, ConfidenceEstimators

Structures et estimateurs de confiance utilisés à travers la librairie.

#### 4.4.2 Pipeline de traitement

Le pipeline, résumé en Figure 4.3, suit les étapes suivantes :

- Acquisition** d'un nouveau scan LiDAR et pré-traitements (déttection des lignes de balayage).
- Prédiction de pose / undistortion** intra-scan via un modèle de mouvement (interpolation de pose/spline) quand activé.
- Extraction des keypoints** (arêtes, plans, *intensity-edges*) avec l'extracteur adapté au capteur (sparse/dense).
- Mise en correspondance scan-to-map** contre la carte locale par type de points-clés (KD-tree, contraintes géométriques).
- Construction des contraintes & optimisation locale** (Ceres) pour estimer la pose optimale et son incertitude, en intégrant éventuellement IMU/GNSS/odom/caméra/*landmarks*.
- Mise à jour des cartes voxelisées** (RollingGrid) et gestion mémoire (échantillonnage, vieillissement).
- (Optionnel) Bouclage/PGO** pour corriger la dérive globale si des fermetures de boucle ou des poses absolues sont disponibles.



**FIGURE 4.3 – Chaîne de traitement de LidarSlam** : extraction de keypoints, prédiction/undistortion, matching, construction des contraintes, optimisation locale et mise à jour des cartes voxelisées (PGO optionnel). [42]

## 4.5 Wrapper ROS 2 de la librairie

### 4.5.1 Architecture générale

Le *wrapper ROS 2* encapsule la librairie **LidarSlam** dans des nœuds et paquets ROS 2 afin de l'intégrer facilement à un système robotique. L'architecture se compose principalement des paquets suivants :

- `lidar_slam` : paquet principal. Fournit le nœud `lidar_slam_node` (exécute le pipeline SLAM : extraction des keypoints, egomotion/undistortion, matching *scan-to-map*, optimisation locale, mise à jour des cartes, PGO optionnel) et, au besoin, `aggregation_node` (agrégation/slices/détection d'obstacles).
- `lidar_conversions` : nœuds de conversion (Velodyne/Ouster/Livox/Hesai/Robosense ou générique) qui transforment les données brutes des drivers au format attendu par le SLAM (contenant les champs `x,y,z,time,intensity,laser_id,label`).
- `slam_visualization` : plugin RViz (*SlamControlPanel*) pour piloter le SLAM à chaud (RESET, SAVE/LOAD cartes et trajectoire, set\_pose, (dés)activer la mise à jour de carte, lancer un bouclage, etc.) et visualiser des indicateurs de confiance.
- `external_sensors_msg` : messages minimaux pour interopérer avec certains capteurs externes (ex. AprilTag, Livox) si les paquets correspondants ne sont pas installés.
- `tests` : nœud et scripts de test/CI (replay et comparaison aux logs de référence).

### Fichiers centraux

- `launch/slam_{ouster,velodyne,livox,hesai}.launch.py` : démarre l'ensemble du pipeline ( lancement des noeuds driver,du noeud de conversion, des noeuds publant les TF (statiques et dynamiques ), du noeud `lidar_slam_node`, de RViz, et (si demandé) du noeud `aggregation_node`.
- `params/slam_config_{indoor,outdoor}.yaml` : Contient l'ensemble des paramètres de configuration du SLAM (frames, ego-motion/undistortion, extraction, registration, voxel grid, PGO, détecteur d'échec, sorties).

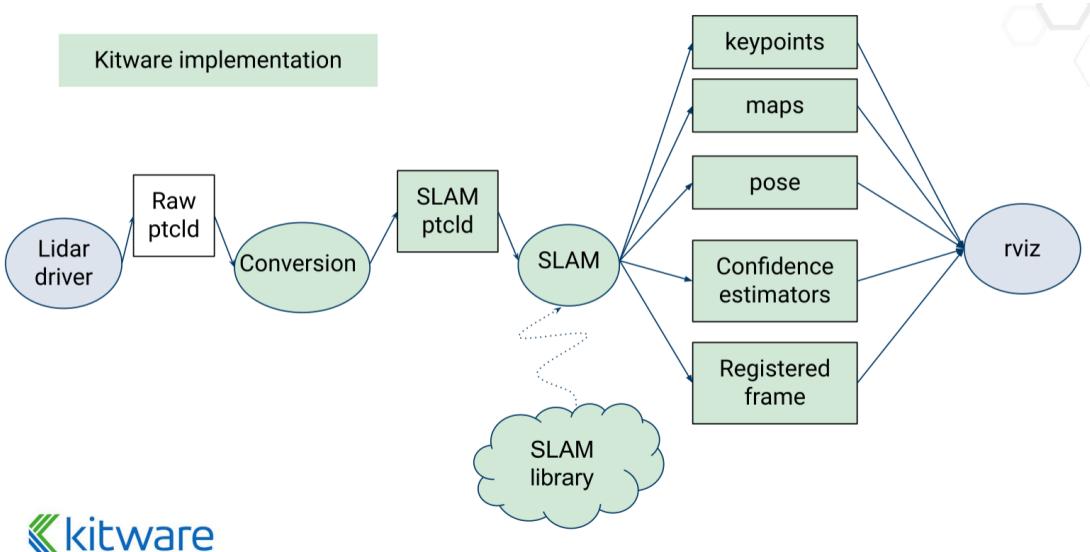
#### 4.5.2 Pipeline du wrapper

Le flux typique est le suivant : le *driver LiDAR* publie un nuage brut → un *nœud de conversion* le transforme au format SLAM → `lidar_slam_node` exécute la chaîne SLAM (front-end/back-end, fusion capteurs externes, PGO optionnel) → les *sorties* (keypoints, cartes, pose, frame enregistrée, métriques de confiance) sont publiées vers RViz et les consommateurs ROS 2. La Figure 4.4 illustre ce pipeline.

#### 4.5.3 Installation et prise en main de la librairie

L'installation et de prise en main de la librairie se déroule en cinq étapes principales :

1. Installer ROS 2 Jazzy et les dépendances (`Eigen3`, `Ceres`, `PCL`, `Nanoflann`, etc.).
2. Cloner le dépôt `slam` de Kitware.
3. Configurer et compiler la librairie C++ avec les dépendances locales.
4. Compiler les paquets ROS 2 (`ros2_wrapping`) au moyen de `colcon`.



**FIGURE 4.4 –** Architecture et pipeline côté ROS 2 : conversions des nuages, nœud **SLAM**, publications (keypoints, maps, pose, estimations de confiance, frame enregistrée) vers RViz. [12]

5. Paramétriser puis lancer le pipeline (fichiers YAML et *launch*).

Le guide d’installation complet (commandes et astuces de compilation) est détaillé en **Annexe C**.

# Chapitre 5

## Évaluation de la librairie LiDAR-SLAM de Kitware

### 5.1 Introduction

La bibliothèque LiDAR-SLAM de Kitware constitue un choix pertinent pour la cartographie et la navigation autonome car elle est *modulaire* (cœur C++ clair séparant front-end et back-end), *intégrable* (wrapper ROS 2, outils LidarView/ParaView) et *multi-capteurs* (LiDAR au centre, avec prise en charge optionnelle d'IMU, GNSS, odométrie et balises). Elle propose en outre des mécanismes d'optimisation de graphe (PGO) et des indicateurs internes de confiance, éléments clés pour une exploitation robuste en robotique mobile.

Dans le cadre de ce stage, l'objectif est d'**évaluer les performances** de cette bibliothèque afin d'identifier les paramètres et configurations optimaux pour son utilisation dans des applications robotiques développées au sein de l'**Institut Pascal** (équipe **PerSyst**). Cette évaluation vise une intégration fiable et efficace sur des robots mobiles opérant dans des environnements variés (intérieur / extérieur).

La démarche expérimentale s'appuie sur **deux rosbags** ROS 2 fournis par Kitware (un scénario *indoor* et un scénario *outdoor*). Pour chaque bag, nous faisons varier des *groupes de paramètres* de la librairie (p. ex. *ego-motion*, *undistortion*, etc.) et nous analysons l'impact sur la trajectoire et la qualité de la solution. Faute de *ground truth*, la trajectoire obtenue avec la configuration par défaut sert de *référence* pour les métriques d'écart.

Les performances sont mesurées selon trois volets complémentaires :

- **Métriques internes de confiance** exposées par la librairie : *overlap*, *std\_position\_error*, *computation\_time* ;
- **Analyse de la trajectoire estimée** : composantes ( $x$ ,  $y$ ,  $z$ ) et attitudes (roll, pitch, yaw) ;
- **Métriques evo** (erreurs trajectoires) : *ATE* (Absolute Trajectory Error) et *RPE* (Relative Pose Error).

## 5.2 Jeux de données et plateforme expérimentale

### 5.2.1 ROS 2 bags

Nous utilisons deux bags ROS 2 issus de la formation *LiDAR SLAM training* (03/10/2023, Kitware). Les deux séquences contiennent des données LiDAR Velodyne (`sensor_msgs/PointCloud2`) publiées sur le topic `/velodyne_points`. Chaque bag est décrit par sa durée, son nombre de messages et la fréquence effective du LiDAR (environ 10 Hz), ainsi que par sa nature (extérieur/intérieur).

Bag	Type	Capteurs	Durée	Hz	LiDAR
<code>bag1.db3</code>	Outdoor	LiDAR seul ( <code>PointCloud2</code> )	01 :16.462	9.926	Hz
<code>loop.db3</code>	Indoor	LiDAR seul ( <code>PointCloud2</code> )	02 :20.369	9.931	Hz

**TABLE 5.1** – Séquences ROS 2 utilisées. Les fréquences LiDAR sont calculées à partir du rapport *messages/durée*.

### 5.2.2 Plateforme logicielle et matérielle

Les expériences ont été réalisées sous **Ubuntu 24.04.2 LTS (Noble)** avec **ROS 2 Jazzy**. Le dépôt LiDAR-SLAM de Kitware a été cloné en sa version **v3.0-2-g95381e21**. Les tableaux ci-dessous ( 5.2 et 5.3 ) offrent une vue d'ensemble sur l'environnement logiciel et matériel utilisé.

#### Environnement logiciel.

Élément	Détail
OS	Ubuntu 24.04.2 LTS ( <i>Noble Numbat</i> )
Noyau	Linux 6.11.0-21-generic
ROS 2	Jazzy ( <code>ROS_VERSION=2, ROS_PYTHON_VERSION=3</code> )
Variables utiles	<code>ROS_AUTOMATIC_DISCOVERY_RANGE=SUBNET</code>
GCC	13.3.0
Pilote NVIDIA	550.120
Repo slam	<b>v3.0-2-g95381e21</b>

**TABLE 5.2** – Plateforme logicielle utilisée pour l'évaluation.

### Plateforme matérielle.

Composant	Détail
Machine	Dell Precision 3650 Tower (UEFI v1.5.2)
CPU	Intel Core i9-10900K (10 coeurs, HT)
RAM	64 GiB
GPU	NVIDIA GeForce RTX 3080
Stockage	NVMe 1 To (SK Hynix PC711) + HDD 2 To (Seagate)
Réseau	Intel I219-LM (1 Gbps)
Affichage	Dual 1920×1080@60 Hz

**TABLE 5.3** – Configuration matérielle de la station de test.

## 5.3 Méthodologie d'évaluation

### 5.3.1 Facteurs et configurations testées

Comme dit précédemment, le fonctionnement du Slam est paramétré grâce au fichier `slam_config_{indoor|outdoor}.yaml` fourni par Kitware. À l'intérieur de ce fichier, nous faisons varier **un paramètre à la fois**, avec quelques **combinaisons ciblées** lorsque c'est pertinent pour observer des interactions. On regroupe ainsi :

**Pour l'étape d'ego-motion et celle de localization :**

- Les paramètres liés au *point to edge matching* (`edge_nb_neighbors`, `edge_min_nb_neighbors`, `edge_max_model_error`).
- Les paramètres liés au *point to plane matching* (`planarity_threshold`, `plane_max_model_error`, `max_neighbors_distance`).
- Le nombre max d'itérations pour l'ICP et l'optimisation par Levenberg–Marquardt (`ICP_max_iter`, `LM_max_iter`).
- Les limites de saturation des résidus par *Tukey loss* (`init_saturation_distance`, `final_saturation_distance`).

**Pour l'étape d'extraction des points caractéristiques (Keypoints Extraction) :**

- Les limites de distances par rapport au capteur :  
`min_distance_to_sensor`, `max_distance_to_sensor`.
- Les limites d'angles azimuth : `min_azimuth`, `max_azimuth`.
- Les paramètres liés à l'extraction des points d'arête :  
`edge_depth_gap_threshold`, `edge_nb_gap_points`,  
`edge_intensity_gap_threshold`.

### 5.3.1.1 Paramètres évalués.

De façon globale, les familles de paramètres explorées sont les suivantes, en renvoyant pour le détail à l'Annexe D (*Paramètres de configuration détaillés*) :

- **Extraction de points caractéristiques (keypoints)** — cf. Annexe D, rubrique 12 — *Keypoint Extraction (ke.\*).*
- **Ego-motion (ICP local & fonctions coût)** — cf. Annexe D, rubrique 10 — *Ego\_motion\_registration.*
- **Localisation (scan→map sur carte locale)** — cf. Annexe D, rubrique 11 — *Localization.*
- **Paramètres SLAM généraux** (undistortion ; interpolation) — cf. Annexe D, rubrique 8 — *SLAM (général).*

### 5.3.1.2 Nomenclature des expériences.

Chaque test adopte un nom explicite qui encode les paramètres modifiés, et les configurations testées portent les noms des valeurs attribuées à ces paramètres et parfois un terme décrivant la configuration. Exemples :

- ER\_ICP\_LM\_Iter/rapide-3-10 :  
ICP\_max\_iter=3, LM\_max\_iter=10.
- ER\_point\_to\_edge\_match/plus-court-6-3-0.1 :  
edge\_nb\_neighbors=6, edge\_min\_nb\_neighbors=3,  
edge\_max\_model\_error=0.1.
- KE\_downsampling\_voxel\_grid/1.5 :  
voxel\_grid\_resolution=1.5 (avec max\_points figé).

## 5.4 Métriques et calcul

### ATE (Absolute Trajectory Error)

**Définition.** L'ATE mesure l'écart *absolu* de position entre la trajectoire estimée  $\{T_i\}_{i=1}^N$  et la trajectoire de référence  $\{\hat{T}_i\}_{i=1}^N$ . On note  $p_i = \text{trans}(T_i) \in \mathbb{R}^3$  et  $\hat{p}_i = \text{trans}(\hat{T}_i) \in \mathbb{R}^3$  les positions dans un repère commun.

**Erreur instantanée.**

$$e_i^{\text{ATE}} = \| p_i - \hat{p}_i \|_2.$$

**Agrégats usuels.**

$$\text{ATE}_{\text{RMSE}} = \sqrt{\frac{1}{N} \sum_{i=1}^N (e_i^{\text{ATE}})^2}, \quad \text{ATE}_{\text{mean}} = \frac{1}{N} \sum_{i=1}^N e_i^{\text{ATE}}, \quad \text{ATE}_{\text{median}} = \text{median}\{e_i^{\text{ATE}}\}.$$

## RPE (Relative Pose Error)

**Définition.** Le RPE évalue l'erreur sur le *mouvement relatif* entre deux instants séparés d'un décalage  $\Delta \geq 1$ . On considère les poses  $T_i, T_{i+\Delta} \in SE(3)$  et  $\hat{T}_i, \hat{T}_{i+\Delta} \in SE(3)$ .

**Erreur relative de pose.**

$$E_i = (\hat{T}_i^{-1} \hat{T}_{i+\Delta})^{-1} (T_i^{-1} T_{i+\Delta}) \in SE(3).$$

On décompose  $E_i = \begin{bmatrix} R_i & t_i \\ 0 & 1 \end{bmatrix}$  avec  $R_i \in SO(3)$  et  $t_i \in \mathbb{R}^3$ .

**Composantes transl./rot. instantanées.**

$$e_i^{\text{RPE,trans}} = \|t_i\|_2, \quad e_i^{\text{RPE,rot}} = \theta(R_i)$$

où  $\theta(R) = \cos^{-1}\left(\frac{\text{trace}(R)-1}{2}\right)$  est l'angle de rotation en radians.

**Agrégats (sur  $M = N - \Delta$  termes).**

$$\text{RPE}_{\text{RMSE}}^{\text{trans}} = \sqrt{\frac{1}{M} \sum_{i=1}^M (e_i^{\text{RPE,trans}})^2}, \quad \text{RPE}_{\text{RMSE}}^{\text{rot}} = \sqrt{\frac{1}{M} \sum_{i=1}^M (e_i^{\text{RPE,rot}})^2}.$$

### 5.4.1 Métriques internes de confiance (lidar\_slam/Confidence)

Le noeud `lidar_slam_node` publie à chaque pose un message `lidar_slam/Confidence` qui regroupe des indicateurs utiles au suivi de la qualité et à la détection d'échec.

**Overlap [0, 1].** Part de recouvrement entre les points courants et la carte de keypoints. Règles empiriques : bon  $> 0.6$ , mauvais  $< 0.4$ . Des *sauts* d'overlap signalent souvent une pose erronée ou un changement brutal de scène.

**std\_position\_error (scalaire).** Écart-type de l'erreur de position *fourni* par le SLAM à partir de la matrice de corrélation.

**computation\_time (s).** Temps de calcul de la réception des données à la sortie de la pose. Indicateur clé pour juger du respect des contraintes temps réel et du coût des réglages (rayons de voisinage, itérations ICP/LM, robustification, ...).

### 5.4.2 Outils d'évaluation et automatisation

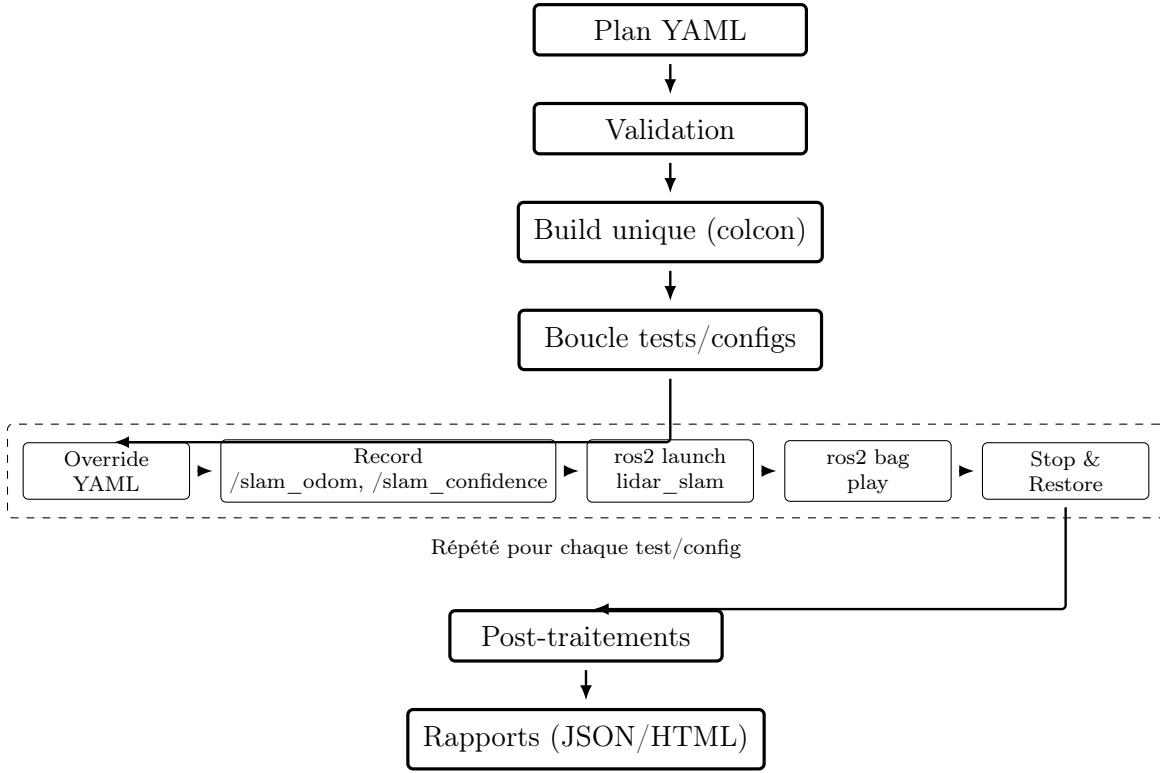
Dans le cadre de cette étude, j'ai développé une suite de scripts pour industrialiser l'évaluation du SLAM. Ils couvrent notamment : (i) l'extraction des *métriques de confiance*

en **CSV**, (ii) la génération de **graphiques synthétiques** (séries temporelles et panneaux comparatifs) pour chaque configuration testée, (iii) l'extraction des trajectoires (TUM/CSV), (iv) le calcul des **métriques evo** (ATE, RPE) selon notre protocole, et (v) la **mise en forme** de *tableaux de synthèse* et de rapports (JSON/HTML). Constatant que le processus d'exécution des tests et de récolte des résultats était très répétitif, j'ai d'abord *automatisé des étapes isolées* (scripts dédiés), puis j'ai écrit un **orchestrateur unique** (`slam_orchestrator.py`) qui pilote *l'ensemble de la chaîne* de bout en bout.

### Composants (rôle).

- `odom_csv_extractor, collect_confidence_metrics.py, collect_summaries.py` : extraction des métriques de confiance (`/slam_confidence`) en CSV, agrégation de ces derniers puis mise en forme et classification sous forme de tableaux respectivement.
- `extract_trajectories.py, extract_trajectory_plots.py` : export TUM de la trajectoire + figures  $x, y, z$ , roll, pitch, yaw, vitesses.
- `compute_evo_metrics_no_align.py, summarize_by_metric.py` : calcul et synthèse ATE/RPE + génération d'un tableau récapitulatif des résultats obtenus pour ces métriques.
- `run_plots_confiance.sh` : génération d'un panel de graphiques (*overlap, nb\_matches, std\_position\_error, computation\_time*).

**Orchestration globale.** Le fonctionnement de `slam_orchestrator.py` est résumé par le schéma ci-dessous : lecture et **validation** d'un plan `.yaml`, **build unique**, puis boucle *test/config* qui (i) *override* le YAML, (ii) lance l'*enregistrement* (`/slam_odom, /slam_confidence`), (iii) `ros2 launch`, (iv) `ros2 bag play`, (v) *arrêt propre & restauration*, avant d'exécuter les **post-traitements** et de produire des **rapports** JSON/HTML, comme illustré à la [figure 5.1](#).



**FIGURE 5.1** – Orchestrateur d’automatisation des tests.

### Entrées / sorties.

- **Entrées** : plan.yaml (expérience, bag, référence TUM, tests/configs), YAML défaut (indoor/outdoor), chemin du bag.
- **Sorties** : bags enregistrés (/slam\_odom, /slam\_confidence), CSV de confiance, trajectoires TUM/CSV, métriques *evo* (ATE/RPE), figures (trajectoires, métriques), tableaux de synthèse, report.json/html.

## 5.5 Résultats

### 5.5.1 Configuration *outdoor*

**Contexte du scénario.** Cette séquence correspond à un **scan LiDAR urbain** : un véhicule équipé d’un scanner parcourt une *rue résidentielle* avec trottoirs, maisons et végétation (arbres, haies). L’environnement présente des arêtes de bâtiments, des plans de chaussée et de façades, ainsi que des éléments végétaux moins structurés, comme illustré à la figure 5.2.

Élément	Détail
Bag ROS 2	<code>bag1.db3</code> ( <code>sqlite3</code> , 265.2 MiB)
Durée	76.462 s (2020-10-21 09 :49 :58 → 09 :51 :14)
Topic LiDAR	<code>velodyne_points</code> ( <code>sensor_msgs/PointCloud2</code> )
Nombre de messages	759
Fréquence LiDAR (estimée)	≈ 9.93 Hz (759/76.462)
Capteurs externes	Aucun topic IMU/GNSS/odom/tags dans ce bag
launch	<code>lidar_slam</code> ( <code>slam_velodyne.launch.py</code> )
YAML utilisé	<code>slam_config_outdoor.yaml</code>
Trajectoire de référence	<i>Trajectoire obtenue avec les paramètres par défaut</i>

TABLE 5.4 – Métadonnées de la séquence *outdoor* et éléments de configuration.

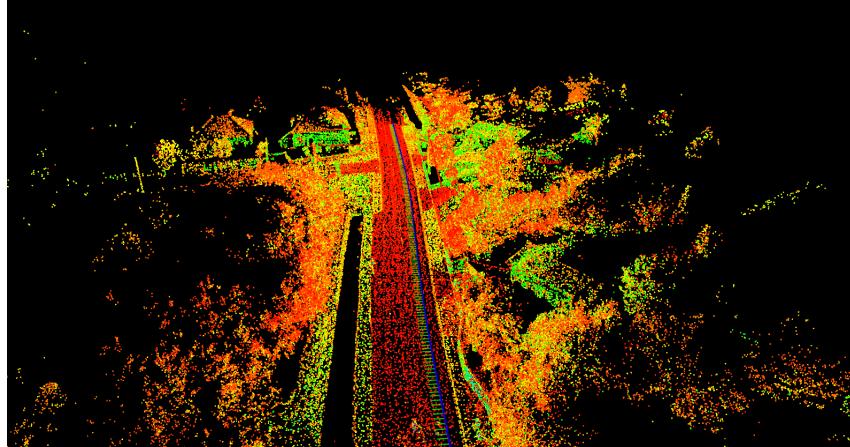


FIGURE 5.2 – Vue RViz2 de la configuration *outdoor* (carte et trajectoire estimée).

### 5.5.1.1 Métriques de confiance

**Principe d'évaluation.** Pour chaque **test** (groupe de paramètres) et pour chaque **métrique de confiance**  $M$  (`overlap`, `std_position_error`, `computation_time`), on calcule un **impact relatif** :

$$I_{\text{rel}}(M, \text{test}) = 100 \times \frac{M_{\text{best}} - M_{\text{worst}}}{2(M_{\text{best}} + M_{\text{worst}})} (\%),$$

où  $M_{\text{best}}$ ,  $M_{\text{worst}}$  représentent respectivement le meilleur et le pire résultat obtenu en respectant la *direction bénéfique* de la métrique :

- **Overlap** : plus grand = mieux ⇒  $M_{\text{best}} = \max$ ,  $M_{\text{worst}} = \min$ .
- **Std position error** : plus petit = mieux ⇒  $M_{\text{best}} = \min$ ,  $M_{\text{worst}} = \max$ .
- **Computation time** : plus petit = mieux ⇒  $M_{\text{best}} = \min$ .

Cet impact relatif étant étant invariant à l'échelle des résultats du fait de sa normalisation, on qualifie un effet de **significatif** lorsque  $I_{\text{rel}} \geq 5\%$ .

**Note de lecture.** Dans tous les tableaux de cette section, la colonne **N** indique le **nombre de configurations** évaluées à l'intérieur du *test* considéré.

### 5.5.1.2 Analyse par métriques de confiance

**Principe d'évaluation (recalc sur mean).** Pour chaque **test** (groupe de paramètres) et pour chaque **métrique de confiance**  $M$  (`overlap`, `std_position_error`, `computation_time`), on calcule l'**impact relatif** (écart relatif symétrisé) à partir des *valeurs moyennes* extraites des fichiers `summary/summary_by_metric.xlsx` (feuille dédiée à chaque métrique ; seules les colonnes dont le nom contient « mean » sont utilisées) :

$$I_{\text{rel}}(M, \text{test}) = 100 \times \frac{M_{\text{best}} - M_{\text{worst}}}{2(M_{\text{best}} + M_{\text{worst}})} (\%),$$

en respectant la direction bénéfique de chaque métrique (plus grand = mieux pour `overlap` ; plus petit = mieux pour `std_position_error` et `computation_time`). Nous qualifions un effet de **significatif** si  $I_{\text{rel}} \geq 5\%$ . *Note de lecture* : **N** indique le **nombre de configurations testées** dans le test considéré.

**Rappels d'interprétation (Confidence.msg).**

- **Overlap**  $\in [0, 1]$  : plus élevé est meilleur (recouvrement).
- **Std position error** : plus petit est meilleur (dispersion de position).
- **Computation time** : plus petit est meilleur (latence par trame).

**Overlap** : . Classement par  $I_{\text{rel}}$  décroissant.

Famille	Test	N	$I_{\text{rel}} (\%)$	Meilleure config	Pire config
KE	<code>downsampling.max_points</code>	6	32.3	5000	200
misc	<code>interpolation_model</code>	3	23.8	0.0	2.0
KE	<code>input_sampling_ratio</code>	5	17.6	1.0-par-defaut	0.2
KE	<code>neighbors_side_nb</code>	5	14.7	3	50
KE	<code>blob</code>	2	8.6	enabled	disabled-par-defaut
localisation	<code>blob_nb_neighbors</code>	6	8.1	10	disabled
KE	<code>azimuth_angle</code>	4	7.9	225-135	315-45

TABLE 5.5 – Tests significatifs par impact relatif sur l'**overlap** ( $I_{\text{rel}} \geq 5\%$ ).

**Interprétations** : Sur la séquence *outdoor*, l'`overlap` est principalement piloté par la **densité** et la **sélection** des points clés ( keypoints ). Augmenter `ke.downsampling.max_points` est la manipulation la plus efficace ( $I_{\text{rel}} = 32.3\%$ ), devant l'effet du modèle d'interpolation qui nécessite d'être linéaire ( option 0 ). Un `input_sampling_ratio élevé` et un `neighbors_side_nb modéré` favorisent aussi l'`overlap` ; des voisinages trop larges diluent les structures et nuisent aux appariements. L'**activation des blobs** et un nombre de voisins raisonnable pour la localisation

(blob\_nb\_neighbors=10) apportent un gain mesuré (8–9%), cohérent avec la présence d’objets arrondis et de végétation.

**Std\_position\_error.** Classement par  $I_{\text{rel}}$  décroissant.

Famille	Test	N	$I_{\text{rel}} (\%)$	Meilleure config	Pire config
KE	downsampling.max_points	6	44.8	5000	200
KE	input_sampling_ratio	5	39.4	1.0-par-defaut	0.2
SLAM	interpolation_model	3	33.8	0.0	2.0
localisation	blob_nb_neighbors	6	31.0	4	disabled
KE	blob	2	29.3	enabled	disabled-par-defaut
localisation	point_to_edge_and_point_to_plane_matching	5	27.5	permissif	tres-strict
localisation	init_and_final_saturation_distance	5	22.2	ultra-strict	tres-lache
KE	azimuth_angle	4	20.4	225–135	315–45

TABLE 5.6 – Tests significatifs par impact relatif sur std\_position\_error ( $I_{\text{rel}} \geq 5\%$ ).

**Interprétations :** Les résultats montrent des tendances nettes :

- **Densité de keypoints (KE).** Augmenter le nombre de points clés conservés réduit clairement la dispersion de pose. Les deux leviers les plus efficaces sont *downsampling\_max\_points* (meilleur à 5000, pire à 200) et *input\_sampling\_ratio* (meilleur à 1.0, pire à 0.2). Un échantillonnage trop agressif appauvrit les contraintes et augmente la variance.
- **Modèle d’interpolation (SLAM).** Le modèle *linéaire* (interpolation\_model=0) produit des poses plus stables que le *cubique* (=2) sur ce bag. Les modèles d’ordre élevé semblent amplifier le bruit.
- **Contraintes blob en localisation.** Activer *blob* et fixer *blob\_nb\_neighbors* autour de 4 améliore la stabilité. Désactiver *blob* ou utiliser trop peu de voisins dégrade la localisation, notamment dans les zones végétalisées où les arêtes/plans sont moins structurés.
- **Paramètres d’appariement (localisation).** Des réglages *plus permissifs* pour les correspondances point–arête et point–plan (davantage de voisins / tolérances moins strictes) réduisent la variance, alors qu’un paramétrage strict fragilise l’estimation.
- **Robustification (Tukey).** Des distances de saturation élevées ( *ultra-strictes*) coupent mieux les outliers et font baisser l’écart-type, tandis que des fenêtres *très lâches* laissent passer du bruit.
- **Fenêtrage angulaire (KE).** Un secteur frontal *large* (*azimuth\_angle* 225–135) stabilise mieux la pose qu’un cône frontal *étroit* (315–45). Garder un large champ utile devant le véhicule apporte plus de paires contraintes/cohérentes.

**Computation\_time.**  $I_{\text{rel}}$  est reporté avec un *signe négatif* pour matérialiser un *gain* de temps (latence en baisse).

Famille	Test	N	$I_{\text{rel}}$ (%)	Meilleure config	Pire config
KE	downsampling.max_points	6	-45.3	200	5000
KE	input_sampling_ratio	5	-21.8	0.4	1.0-par-defaut
misc	interpolation_model	3	-25.9	0.0	2.0
KE	blob	2	-20.0	disabled-par-defaut	enabled
localisation	blob_nb_neighbors	6	-18.3	disabled	15
localisation	ICP_and_LM_interations_2	7	-17.1	Low-ICP	High-LM-High-ICP
KE	min_beam_surface_angle	5	-11.1	10-par-defaut	20
KE	azimuth_angle	4	-8.3	315-45	0-360-par-defaut
KE	neighbors_side_nb	5	-5.4	3	50

TABLE 5.7 – Tests significatifs par impact relatif sur le computation\_time ( $I_{\text{rel}} \geq 5\%$ ).

**Interprétations.** Le temps de calcul est surtout dépendant de la *densité de keypoints* et, dans une moindre mesure, par le coût de l’optimisation. Dans nos tests, réduire fortement le nombre de points-clés (ke.downsampling\_max\_points= 200) est de loin l’action la plus efficace pour accélérer le pipeline. Diminuer l’échantillonnage en entrée (ke.input\_sampling\_ratio≈ 0.4) apporte un gain additionnel. Choisir un modèle d’interpolation simple est nettement plus léger que les modèles plus complexes . Désactiver l’extraction des *blobs* et la recherche de leurs voisins (ke.blob=disabled, localisation.blob\_nb\_neighbors=disabled) réduit aussi le coût. Abaisser les itérations ICP/LM (*Low-ICP*) allège encore l’optimisation. Enfin, conserver un angle minimum faisceau-surface modéré (min\_beam\_surface\_angle= 10°), restreindre l’azimut au secteur avant (azimuth= 315–45°) et limiter les voisins latéraux (neighbors\_side\_nb= 3) contribuent à diminuer le temps de calcul.

**Synthèse directionnelle.** Pour guider le réglage, le tableau ci-dessous regroupe, pour les tests les plus impactants, la **meilleure configuration** par métrique :

Famille	Test	Best overlap	Best std_pos	Best ctime
KE	downsampling.max_points	5000	5000	200
KE	input_sampling_ratio	1.0-par-defaut	1.0-par-defaut	0.4
misc	interpolation_model	0.0	0.0	0.0
localisation	blob_nb_neighbors	10	4	disabled
KE	blob	enabled	enabled	disabled-par-defaut
localisation	point_to_edge_and_point_to_plane_matching	permissif	permissif	permissif
localisation	init_and_final_saturation_distance	tres-lache	ultra-strict	tres-lache
KE	azimuth_angle	225-135	225-135	315-45
KE	neighbors_side_nb	3	50	3

TABLE 5.8 – Meilleures configurations par métrique pour les tests les plus impactants (bag outdoor, recalcul mean).

### 5.5.1.3 Analyse de la trajectoire : métriques evo

**Protocole.** En l’absence de vérité terrain, chaque configuration est évaluée *relativement* à la trajectoire *par défaut* au moyen des métriques evo usuelles : **ATE RMSE** (erreur absolue) et **RPE RMSE** (erreur relative entre poses, translation).

**Critère d'impact ( $I_{\text{rel}}$ ).** On mesure l'effet d'un paramètre sur  $M \in \{\text{ATE RMSE}, \text{RPE RMSE}\}$  via l'*amplitude relative symétrisée* :

$$I_{\text{rel}}(M, \text{param}) = 100 \times \frac{\max(M) - \min(M)}{2(\max(M) + \min(M))},$$

calculée *par test* sur les configurations du paramètre. Impact *significatif* si  $I_{\text{rel}} \geq 5\%$ .

**Classement par impact ATE RMSE.** le tableau ci-dessous présente les résultats obtenus pour l'ATE

Famille	Paramètre	N	$I_{\text{rel}} [\%]$	Meilleure config	ATE RMSE [m]	Pire config	ATE RMSE [m]
localisation	init_and_final_saturation_distance	5	41.9	strict	0.1802	ultra-lâche	2.0379
KE	azimuth_angle	4	40.4	225-135	0.2349	315-45	2.2234
KE	min_beam_surface_angle	5	30.4	5	0.06201	30	0.2544
KE	input_sampling_ratio	5	30.0	0.8	0.1425	0.4	0.5686
KE	downsampling.voxel_grid_resolution	5	29.8	0.2	0.0448	5	0.1686

**TABLE 5.9 – Classement par impact relatif (Outdoor) pour la ATE RMSE [m].**

**Interprétations et recommandation** L'ATE est surtout sensible au **fenêtrage spatial**, à la **robustification**, puis aux choix de **keypoints** :

- **Saturation des résidus** (init/final\_saturation\_distance) : Des réglages **stricts** stabilisent l'ICP/LM (0.180 m) tandis que des seuils **ultra-lâches** dégradent fortement (2.038 m).
- **Fenêtrage angulaire** (min/max\_azimuth) : Une fenêtre **large orientée avant** 225–135 réduit l'ATE (0.235 m) par rapport à 315–45 (2.223 m), grâce à des contraintes plus informatives.
- **Qualité des KP** (min\_beam\_surface\_angle) : Un seuil **5°** donne la meilleure ATE (0.062 m) ; **30°** filtre trop et dégrade (0.251 m).
- **Densité d'entrée** (input\_sampling\_ratio) : échantillonner trop réduit l'information géométrique.
- **Résolution de voxels** (ke.downsampling.voxel\_grid\_resolution) : Une grille **fine** 0.2 m (0.045 m) bat 0.5 m (0.169 m) ; conserver le détail local aide l'ICP.

Ainsi, pour **minimiser** l'ATE, il faut effectuer une saturation stricte à l'étape de localisation, maintenir fenêtre angulaire relativement grande, min\_beam\_surface\_angle=10°, input\_sampling\_ratio≈0.8 et réduire la résolution de la grille de voxels à 0.2 m.

**Classement par impact RPE RMSE.** le tableau ci-dessous présente les résultats obtenus pour la RPE

Famille	Paramètre	N	$I_{\text{rel}} [\%]$	Meilleure config	RPE RMSE [m]	Pire config	RPE RMSE [m]
localisation	max_neighbor_distance	6	39.3	7.5	0.0228	3.5	0.0267
localisation	init_and_final_saturation_distance	5	27.7	strict	0.0234	ultra-lâche	0.0817
KE	azimuth_angle	4	15.3	225–135	0.0251	270_90	0.0473
KE	input_sampling_ratio	5	12.9	0.8	0.0272	0.4	0.0462

TABLE 5.10 – Classement par impact relatif (Outdoor) pour la RPE RMSE [m], zéros exclus.

**Interprétations et recommandation** La dispersion relative du *RPE RMSE* est principalement pilotée par les paramètres de **localisation**. Deux réglages ressortent nettement :

- **Rayon de voisinage (max\_neighbor\_distance)** : c'est l'effet le plus marqué. Un rayon autour de *7.5 m* (plus élevée que la valeur par défaut de 5 m) réduit le RPE par rapport à des valeurs plus petites (*3.5 m*). L'algorithme trouve alors des correspondances plus stables tout en restant local.
- **Saturation de Tukey (init\_and\_final\_saturation\_distance)** : un réglage *strict* aide à contenir les résidus aberrants et baisse le RPE, alors qu'un réglage *ultra-lâche* dégrade les résultats.

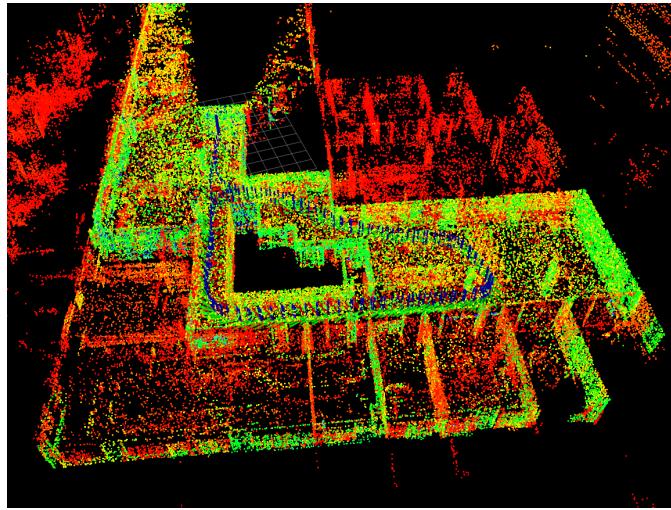
Les réglages côté **KE** jouent un rôle secondaire mais utile :

- **Fenêtre d'azimut (azimuth\_angle)** : privilégier une fenêtre frontale de type *225–135* améliore la cohérence des incrémentations à court terme.
- **Échantillonnage d'entrée (input\_sampling\_ratio)** : une valeur *0.8* offre un bon compromis entre stabilité et coût de calcul.

Ainsi, pour **minimiser** le RPE, il convient de tout d'abord porter **max\_neighbor\_distance** autour de *7–8 m* et serrer la saturation de Tukey (*strict*). Ensuite ajuster l'azimut vers une fenêtre frontale relativement large et je maintenir **input\_sampling\_ratio** proche de *0.8*. Il est bien sûr utile de surveiller l'impact sur le temps de calcul et l'overlap pour préserver l'équilibre global.

### 5.5.2 Configuration *indoor*

**Contexte du scénario.** Ici, il l'agit d'une séquence **en environnement intérieur** : le robot effectue une *boucle* dans un bâtiment de couloirs orthogonaux et pièces adjacentes. Géométrie dominée par des *plans verticaux* (murs/cloisons) et des *arêtes franches*. La trajectoire repasse près du départ, favorisant la détection de *boucles*, comme illustré à la figure 5.3.



**FIGURE 5.3** – Vue RViz2 de la configuration *indoor* (carte et trajectoire estimée).

Élément	Détail
Fichier bag	loop.db3 ( <i>sqlite3</i> , 1.7 GiB)
ROS Distro (bag)	humble
Durée	140.369 s (2023-09-12 19 :03 :56.57 → 19 :06 :16.94)
Topic LiDAR	/velodyne_points (sensor_msgs/msg/PointCloud2)
Nombre de messages	1394
Fréquence LiDAR (estimée)	≈ 9.93 Hz (1394/140.369)
Capteurs externes	Aucun topic IMU/GNSS/odom/tags dans ce bag
Wrapper / launch	lidar_slam (slam_velodyne.launch.py)
YAML utilisé	slam_config_outdoor.yaml
Trajectoire de référence	<i>Trajectoire obtenue avec les paramètres par défaut</i>

**TABLE 5.11** – Métadonnées de la séquence *indoor* (loop.db3) et éléments de configuration.

### 5.5.2.1 Analyse par métriques de confiance

**Principe d'évaluation.** Même définition d' $I_{\text{rel}}$  que précédemment.

**Overlap.** Classement par  $I_{\text{rel}}$  décroissant.

Famille	Test	N	$I_{\text{rel}}$ (%)	Meilleure config	Pire config
KE	edge	5	0.2	strict-1.0-12-25	tres-strict-2.0-15-30
LOC	max_neighbors_distance	5	0.2	12.0	8.0
ER	max_neighbors_distance	5	0.2	10.0	par-default-3.0
KE	azimuth	4	0.2	avant-etroit-315-45	tres-large-225-135
ER	point_to_plane_match	3	0.2	plus-voisins-7-0.04-0.1	par-default-5-0.04-0.1
LOC	point_to_blob_match	3	0.2	dense-14	par-default-10
KE	input_sampling_ratio	4	0.2	par-default-1.0	0.3
ego	motion	6	0.2	3	par-default-1
ER	Optim_Saturation	3	0.1	par-default-5.0-1.0	plus-large-7.0-1.5
KE	downsampling_max_points	4	0.1	2000	1500

**TABLE 5.12** – Paramètres les plus impactants pour l'overlap (classement par  $I_{\text{rel}}$ ).

**Std\_position\_error.** Classement par  $I_{\text{rel}}$  décroissant.

Famille	Test	N	$I_{\text{rel}} (\%)$	Meilleure config	Pire config
ego	motion	6	0.5	par-defaut-1	4
ER	max_neighbors_distance	5	0.1	10.0	2.0
SLAM	undistortion	4	0.1	0-aucun	par-defaut-2-rafine
SLAM	interpolation_model	3	0.1	1-quadratique	2-cubique
KE	input_sampling_ratio	4	0.1	par-defaut-1.0	0.5
KE	downsampling_voxel_grid	4	0.1	0.5	1.5
KE	downsampling_max_points	4	0.1	2000	1500
KE	edge	5	0.1	strict-1.0-12-25	par-defaut-0.5-10-20
KE	azimuth	4	0.1	avant-etroit-315-45	par-defaut-0-360
KE	dist_sens	3	0.1	eloigne-1.0-300.0	par-defaut-1.0-200.0

TABLE 5.13 – Paramètres les plus impactants pour std\_position\_error (classement par  $I_{\text{rel}}$ ).

**Computation\_time.** Classement par  $I_{\text{rel}}$  décroissant (valeurs négatives = gain).

Famille	Test	N	$I_{\text{rel}} (\%)$	Meilleure config	Pire config
ER	max_neighbors_distance	5	-1.1	8.0	10.0
ego	motion	6	-1.0	5	2
KE	input_sampling_ratio	4	-0.9	0.75	par-defaut-1.0
LOC	max_neighbors_distance	5	-0.9	par-defaut-3.0	8.0
KE	azimuth	4	-0.8	avant-large-270-90	tres-large-225-135
KE	dist_sens	3	-0.8	par-defaut-1.0-200.0	eloigne-1.0-300.0
LOC	ICP_LM_Itér	3	-0.7	rapide-2-10	lent-5-20
ER	point_to_plane_match	3	-0.7	par-defaut-5-0.04-0.1	plus-voisins-7-0.04-0.1
LOC	Optim_Saturation	3	-0.6	plus-large-3.0-0.8	par-defaut-2.0-0.5
KE	downsampling_voxel_grid	4	-0.6	2.0	1.5

TABLE 5.14 – Paramètres les plus impactants pour le computation\_time (classement par  $I_{\text{rel}}$ ).

**Interprétations :** Dans ce scénario en environnement fermé, l’overlap moyen est proche de 0,60, ce qui correspond à un niveau « bon » au sens du message Confidence. Surtout, la dispersion des résultats est très faible (cf. Tab. 5.12–5.14) : quelles que soient les variations de paramètres (extraction, voisinages, itérations, modèles d’interpolation/undistortion), les métriques bougent peu. Cette stabilité s’explique par la nature indoor : distances courtes, géométrie très structurée (murs/plans répétés), vitesse faible, et overlap élevé entre scans consécutifs. Dans ces conditions, l’ICP dispose de suffisamment de contraintes et opère déjà dans une zone « saturée » où augmenter la densité de points, le rayon de recherche ou le nombre d’itérations n’apporte qu’un gain marginal sur l’overlap et l’erreur de position, tout en coûtant du temps de calcul.

En pratique, il suffit donc de conserver la configuration indoor par défaut et de n’appliquer que des ajustements légers pour réduire le temps de calcul sans dégrader la qualité : rayon de voisinage modéré (p. ex. 8 m côté ego-motion, rayon plus court côté localisation), input\_sampling\_ratio légèrement réduit (p. ex.  $\sim 0.75$ –0,9), et une fenêtre d’azimut plutôt frontale. Les options plus « coûteuses » (undistortion raffinée, trop d’itérations ICP/LM) n’apportent pas d’amélioration visible ici.

### 5.5.2.2 Métriques *evo* : constat global

En intérieur, l'**ATE RMSE** et le **RPE RMSE** varient très peu d'une configuration à l'autre : la plupart des réglages aboutissent aux mêmes niveaux d'erreur. Cette *quasi-invariance* reflète un *plateau de solution* lié à la géométrie de la scène (couloirs orthogonaux, nombreuses arêtes franches, plans verticaux) et au fait que la trajectoire boucle rapidement. Autrement dit, les associations de points restent stables et les petites modifications de voisinage, de robustification ou d'échantillonnage ne déplacent pas sensiblement la solution, ce qui rejoint les métriques de confiance.

**Bilan (indoor).** Dans ce bâtiment aux couloirs orthogonaux et plans verticaux bien structurés, la géométrie constraint fortement le problème : l'*overlap* se stabilise autour de  $\approx 0.6$  et, malgré les variations de KE/ER/LOC/SLAM, **ATE** et **RPE** ne prennent que quelques valeurs discrètes, signe d'une trajectoire **quasi invariante** aux réglages. Autrement dit, sur ce bag indoor, les leviers internes affectent surtout le *temps de calcul* via la densité d'échantillonnage, bien plus que la précision finale.

**Recommandations (avec références).** Pour dépasser le plateau observé en *indoor*, il est judicieux d'explorer :

- **La fusion de capteurs externes** : combiner LiDAR, IMU (stabilisation de la trajectoire) et, selon le robot, l'odométrie roue et/ou des marqueurs fiduciaires (ancrages locaux) [25, 35, 43].
- **La fermeture de boucle avec optimisation de graphe de poses (PGO)** : contraindre la dérive à long terme lorsque la trajectoire revisite une zone connue [35, 44].

Ces pistes sont prévues et documentées dans le wrapper ROS 2 de LiDAR-SLAM (capteurs externes, commandes de loop closure/PGO) [45, 46].

# Chapitre 6

## Conclusion Générale

Ce travail a porté sur l'évaluation expérimentale d'une bibliothèque open-source de SLAM LiDAR (Kitware) sous ROS 2, en *indoor* et *outdoor*, au moyen de métriques de confiance (*overlap*, *std\_position\_error*, *computation\_time*) et de trajectoire (ATE, RPE). Pour garantir la reproductibilité, j'ai développé une chaîne d'automatisation complète (scripts dédiés et *slam\_orchestrator*) permettant de rejouer des campagnes paramétriques et de générer automatiquement tableaux et figures.

**Bilan scientifique.** En *indoor*, les résultats sont quasi invariants pour des variations modérées de paramètres, ce qui autorise des profils *temps réel* sans perte visible. En *outdoor* la qualité est favorisée par une densité de points élevée, un modèle d'interpolation linéaire et des appariements permissifs, tandis que la latence est réduite par des blobs désactivés et un échantillonnage plus faible ; le compromis précision/coût de calcul est donc structurant.

**Apports professionnels.** Ce stage m'a introduit concrètement au SLAM LiDAR, m'a familiarisé avec l'écosystème ROS 2 et la programmation C++ (librairie *lidar\_slam*) et Python (suite d'outils et *orchestrator*). J'ai ainsi acquis des pratiques de développement reproductible et d'évaluation expérimentale à l'échelle.

**Apports personnels.** La conception de l'*orchestrator* a renforcé mes capacités de résolution de problèmes et ma persévérance face aux cas limites. Le suivi rapproché avec mon encadrant a amélioré mon écoute et mon travail supervisé.

**Perspectives.** Les suites immédiates portent sur l'évaluation systématique de la fermeture de boucle en *indoor* et l'élargissement des campagnes *outdoor* (scènes et réglages), afin d'affiner les lignes directrices de paramétrage et de consolider les conclusions.

# **Annexes**

# Annexe A

## Approches marquantes du SLAM LiDAR

Cette annexe synthétise quelques méthodes représentatives du SLAM LiDAR. Le tableau ci-dessous rappelle, pour chaque approche, l'année (référence), les capteurs, et l'idée algorithmique principale.

**TABLE A.1** – Méthodes LiDAR-SLAM marquantes (cols. comme l'image : Méthode, Année/Référence, Capteurs utilisés, Framework/Algorithme).

Méthode	Année / Référence	Capteurs utilisés	Framework / Algorithme
LOAM	2014 — [23]	LiDAR 3D	Deux threads (odométrie rapide + mapping) avec features arêtes/plans et désentrelacement (deskew).
Cartographer	2016 — [35]	LiDAR 2D/3D (+ IMU)	Sous-cartes ( <i>submaps</i> ), détection de bouclages et optimisation de graphe en temps réel.
IMLS-SLAM	2018 — [47]	LiDAR 3D	<i>Scan-to-model</i> basé IMLS (surfaces implicites) pour un recalage robuste et précis.
SuMa	2018 — [24]	LiDAR 3D	SLAM dense à surfels, association projective et bouclages via rendu de vues.
LeGO-LOAM	2018 — [29]	LiDAR 3D (+ IMU opt.)	Allégé et optimisé sol : segmentation du sol + features arêtes/plans, adapté UGV.
HDL Graph SLAM	2020 — [48]	LiDAR 3D (+ IMU opt.)	SLAM graphe (NDT, contraintes plan-sol), correction interactive et bouclage.
LIO-SAM	2020 — [25]	LiDAR + IMU (+ GNSS)	Lidar-Inertial factor-graph ( <i>smoothing &amp; mapping</i> ), deskew IMU, bouclages.
CT-ICP	2021 — [49]	LiDAR 3D	Odométrie temps-continu élastique + SLAM avec détection de bouclage (image d'élévation) et PGO.
FAST-LIO2	2021/22 — [50]	LiDAR + IMU	LIO direct via IKF + carte <i>ikd-Tree</i> , sans extraction de features (très rapide).



## Annexe B

### Fiche d'identité de Kitware, Inc.

Cette annexe présente sous forme de tableau les renseignements clés sur l'entreprise à l'origine de la librairie évaluée.

**TABLE B.1** – Fiche d'identité de Kitware, Inc.

Élément	Informations
Nom	Kitware, Inc.
Date de fondation	1998
Siège social	Clifton Park, New York, USA
Domaines d'activité	Vision par ordinateur, imagerie médicale, simulation, visualisation scientifique, intelligence artificielle
Produits phares	VTK, ParaView, Slicer, TeleSculptor, LiDAR SLAM
Type	Société privée
Site web	<a href="https://www.kitware.com/">https://www.kitware.com/</a>

# Annexe C

## Manuel d'installation et de prise en main de LiDAR-SLAM

Cette annexe présente de façon détaillée les étapes d'installation et de prise en main de la librairie lidar slam étudiée. Elle regroupe les prérequis, l'installation, la compilation et un exemple de lancement.

### C.0.1 Pré-requis et dépendances (Ubuntu + ROS 2 ( dans notre cas Jazzy ) )

**Installer ROS 2 Jazzy (édition *desktop-full*).** Les paquets ROS 2 fournissent l'écosystème (messages, TF, outils de build) nécessaire au wrapper.

```
sudo apt-get update
sudo apt-get install ros-jazzy-desktop-full
echo 'source /opt/ros/jazzy/setup.bash' >> ~/.bashrc
source ~/.bashrc
```

**Dépendances indispensables.** Ces bibliothèques sont requises par le cœur C++ :

- **Eigen3** : algèbre linéaire.
- **Ceres-Solver** : optimisation non linéaire (pose/ICP).
- **PCL** : traitement de nuages de points.
- **nanoflann** : recherche de plus proches voisins.

Installation :

```
sudo apt-get install -y libeigen3-dev libceres-dev libpcl-dev libnanoflann-dev
sudo apt-get install -y ros-jazzy-pcl-ros
```

**Dépendances optionnelles (à activer selon vos besoins).**

- **g2o** : optimisation de graphe de poses (PGO).
- **GTSAM** : intégration/contraintes IMU.
- **OpenCV** : fonctions de vision si caméras utilisées.
- **gps\_umd** : messages/outils GNSS.
- **apriltag** : détection de tags fiduciaires.
- **velodyne** : pilote et utilitaires ROS pour LiDAR Velodyne.

Installation :

```
sudo apt-get install -y libg2o-dev libgtsam-dev libopencv-dev
sudo apt-get install -y ros-jazzy-gps-umd ros-jazzy-apriltag ros-jazzy-velodyne
```

**Bouclage robuste (optionnel) : TEASER++.** Si vous souhaitez tester un détecteur/estimateur de bouclage robuste, compilez TEASER++ depuis les sources :

```
git clone https://github.com/MIT-SPARK/TEASER-plusplus.git  
cd TEASER-plusplus && mkdir build && cd build  
cmake .. -DCMAKE_BUILD_TYPE=Release  
make -j  
sudo make install
```

## C.0.2 Téléchargement du dépôt et compilation de la librairie

**Créer un espace de travail.** On regroupe la librairie et ses wrappers dans un workspace.

```
mkdir -p ~/colcon_ws && cd ~/colcon_ws
```

**Cloner le dépôt en mode récursif.** Le dépôt `slam` contient la librairie C++ et les wrappers ; l'option `--recursive` récupère les sous-modules.

```
git clone --recursive https://gitlab.kitware.com/keu-computervision/slam.git src
```

**Configurer et compiler avec des dépendances locales.** Cette méthode permet d'utiliser les versions déjà installées (Eigen, Ceres, PCL, g2o, etc.).

```
cmake -E make_directory build && cd build  
cmake ../src -DCMAKE_BUILD_TYPE=Release \  
-DCeres_DIR=/opt/ceres \  
-Dg2o_DIR=/opt/g2o  
cmake --build . -j
```

**Astuce en cas d'erreur MPI/VTK (MPI::MPI\_C introuvable).** Si CMake se plaint d'une cible MPI::MPI\_C manquante (VTK compilé avec MPI), activez aussi le langage C dans le projet ( ceci s'effectue dans le fichier `/src/CMakeLists.txt` ) :

```
project(LidarSlam  
LANGUAGES C CXX  
VERSION 3.0)
```

## C.0.3 Compilation des paquets ROS 2 du SLAM

Les paquets ROS 2 vivant dans `ros2_wrapping` se compilent avec `colcon`.

```
cd ~/colcon_ws  
colcon build --base-paths src/ros2_wrapping  
source install/setup.bash
```

**Astuce cv\_bridge (changement d'en-tête).** Sur certaines distributions ROS 2, l'en-tête s'appelle `cv_bridge.hpp` au lieu de `cv_bridge.h`. Si l'erreur suivante apparaît :

```
fatal error: cv_bridge/cv_bridge.h: No such file or directory
```

modifiez dans `ros2_wrapping/lidar_slam/src/LidarSlamNode.cxx` :

```
// avant
#include <cv_bridge/cv_bridge.h>
// après
#include <cv_bridge/cv_bridge.hpp>
```

puis relancez colcon build.

## C.0.4 Paramétrage (YAML) et lancement

**Fichiers de configuration.** Les YAML fournis (p. ex. `slam_config_indoor.yaml` / `slam_config_outdoor.yaml`) règlent notamment :

- les **frames** et les entrées : `odometry_frame`, `tracking_frame`, `lidars_nb`, `input...` ;
- la **carte** (voxel grid, résolution, export PCD) ;
- l'**odometrie/ICP** (voisinage, itérations, `slam.ego_motion`) ;
- la **fusion capteurs** (GPS/IMU/tags/poses externes, poids, seuils) ;
- la **publication** (`odom`/`TF`, `registered_points`, `keypoints/*`, `maps/*`, `confidence`) .

**Exemple de lancement (Velodyne).**

```
source ~/colcon_ws/install/setup.bash
ros2 launch lidar_slam slam_velodyne.launch.py
# Options possibles :
# ros2 launch lidar_slam slam_velodyne.launch.py use_sim_time:=false gps:=true
```

# Annexe D

## Paramètres de configuration détaillés

Cette annexe présente sous forme de tableau la liste commentée des paramètres de la librairie (frames, sorties, cartes, capteurs externes, journalisation, SLAM général, enregistrements ICP/LM, localisation, extraction de KP, etc.) en les regroupant en groupes fonctionnels.

### 1 - Cadres et entrées

Paramètre (type)	Rôle
odometry_frame (string)	Cadre de référence utilisé pour les estimations d'odométrie. Il s'agit de la base du système de coordonnées relatif des déplacements.
tracking_frame (string)	Cadre attaché au capteur ou au robot, utilisé pour suivre sa position et orientation dans le cadre d'odométrie.
wheel_frame (string)	Cadre lié aux roues du robot, utilisé pour les mesures d'odométrie issues des capteurs de roue.
ins_frame (string)	Cadre de l'INS (Inertial Navigation System), utilisé pour la fusion avec les mesures inertiales.
gps_frame (string)	Cadre du GPS, utilisé pour intégrer les mesures de position absolue.
lidars_nb (int)	Nombre de capteurs LiDAR utilisés ; si >1, on peut configurer plusieurs topics et paramètres associés.
frames_mode (int)	Mode de gestion des frames multi-LiDAR (ordre de fusion, agrégation, etc.).
frames_waiting_time (double)	Durée maximale d'attente pour synchroniser plusieurs frames en multi-LiDAR.

### 2 - Sorties (/output)

Paramètre (type)	Rôle
pose.odom (bool)	Publier la pose sous forme d'odométrie ROS ( <code>nav_msgs/Odometry</code> ).
pose.tf (bool)	Publier la transformation TF dynamique ( <code>odometry_frame → tracking_frame</code> ).

Paramètre (type)	Rôle
pose.predicted_odom (bool)	Publier une odométrie prédictive (extrapolée) en plus de la pose estimée.
registered_points (bool)	Publier le nuage de points recalé/undistordu pour visualisation/inspection.
keypoints.edges (bool)	Publier les points-clés de type arête.
keypoints.intensity_edges (bool)	Publier les arêtes d'intensité (si disponibles).
keypoints.planes (bool)	Publier les points-clés de type plan.
keypoints.blobs (bool)	Publier les points-clés de type blob.
maps (bool)	Publier les cartes locales de points-clés.
submaps (bool)	Publier des sous-cartes (submaps) pour analyses locales.
confidence (bool)	Publier le message de confiance (covariances, overlap, flags, temps de calcul).

### 3 - Cartes (/maps)

Paramètre (type)	Rôle
initial_maps (string)	Préfixe/fichiers de cartes à charger au démarrage (PCD).
initial_pose (double[7])	Pose initiale (quaternion + translation) si connue.
export_pcd_format (int)	Format d'export des nuages ( <i>ASCII</i> , <i>Binary</i> , <i>BinaryCompressed</i> ).

### 4 - Capteurs externes (/external\_sensors)

Paramètre (type)	Rôle
max_measures (int)	Nombre maximal de mesures externes conservées en tampon.
time_threshold (double)	Seuil temporel de synchronisation des mesures externes.
use_header_time (bool)	Utiliser les timestamps des entêtes ROS pour la synchro.
weight (double)	Poids global des contraintes capteurs externes dans l'optimisation.
saturation_distance (double)	Distance de saturation (robustification) pour leurs résidus.

### 5 - Capteurs externes : Odomètre roue (wheel\_encoder)

Paramètre (type)	Rôle
relative (bool)	Traiter la mesure comme un déplacement relatif.
reference (double)	Référence d'échelle (mètre/tick, etc.).
direction (int)	Sens de comptage (+1, -1) si nécessaire.
weight (double)	Poids de la contrainte odométrique dans l'optimisation.

## 6 - Capteurs externes : Poses externes / Tags / GPS / Camera

Paramètre (type)	Rôle
external_poses.enable (bool)	Activer l'utilisation de poses externes.
external_poses.weight (double)	Poids des contraintes de poses externes.
external_poses.saturation_distance (double)	Saturation des résidus associés.
tags.anchor_file (string)	Fichier d'ancrage (positions de tags fiduciaires).
tags.publish_tf (bool)	Publier des TF statiques pour les tags.
imu.weight (double)	Poids global des contraintes IMU.
camera.weight (double)	Poids des résidus issus de la caméra (si utilisés).

## 7 - Journalisation (/logging)

Paramètre (type)	Rôle
storage_type (enum)	Type de stockage des nuages (RAM PCL, PCD ASCII/Binary/Compressed).
only_keyframes (bool)	Ne stocker que les keyframes (réduction volume).
timeout (double)	Délai d'attente pour certaines opérations d'I/O.

## 8 - SLAM (général) : 2D/threads/undistortion/interpolation

Paramètre (type)	Rôle
2d_mode (bool)	Contraindre le mouvement au plan (yaw seulement).
n_threads (int)	Nombre de threads utilisés par la librairie.
interpolation_model (enum)	Modèle d'interpolation (LINEAR, QUADRATIC, CUBIC) pour l'egomotion/deskew.

Paramètre (type)	Rôle
undistortion (enum)	NONE / ONCE / REFINED / EXTERNAL — stratégie de <i>deskew</i> .

## 10 - Ego\_motion\_registration (ICP local & coûts)

Paramètre (type)	Rôle
max_neighbors_distance (double)	Rayon de recherche des voisins pour l'appariement.
edge_nb_neighbors (int)	Nb. de voisins pour les contraintes point→edge.
edge_min_nb_neighbors (int)	Nb. minimal de voisins requis (filtre robustesse).
edge_max_model_error (double)	Seuil d'erreur modèle (point→edge).
plane_nb_neighbors (int)	Nb. de voisins pour les contraintes point→plane.
planarity_threshold (double)	Seuil de planarité pour valider un plan local.
plane_max_model_error (double)	Seuil d'erreur modèle (point→plane).
ICP_max_iter (int)	Itérations maximales ICP (enregistrement).
LM_max_iter (int)	Itérations maximales Levenberg–Marquardt (optimisation).
init_saturation_distance (double)	Distance de saturation initiale (perte de Tukey).
final_saturation_distance (double)	Distance de saturation finale (perte de Tukey).

## 11 - Localization (scan→map sur carte locale)

Paramètre (type)	Rôle
max_neighbors_distance (double)	Rayon de recherche des voisins dans la carte.
edge_nb_neighbors (int)	Nb. de voisins pour les contraintes point→edge.
edge_min_nb_neighbors (int)	Nb. minimal de voisins requis (robustesse).
edge_max_model_error (double)	Seuil d'erreur modèle (edge).
plane_nb_neighbors (int)	Nb. de voisins pour les contraintes point→plane.
planarity_threshold (double)	Seuil de planarité des voxels/voisinages.
plane_max_model_error (double)	Seuil d'erreur modèle (plane).
ICP_max_iter (int)	Itérations max. ICP (localisation).
LM_max_iter (int)	Itérations max. LM.
init_saturation_distance (double)	Saturation initiale (Tukey).
final_saturation_distance (double)	Saturation finale (Tukey).
blob_nb_neighbors (int)	Nb. de voisins pour les contraintes de type blob (si activées).

## 12 - Keypoint Extraction (ke.\*)

Paramètre (type)	Rôle
enable.blob (bool)	Activer l'extraction des blobs (en plus des edges/planes).
min_distance_to_sensor (double)	Distance minimale au capteur retenue pour les KP.
max_distance_to_sensor (double)	Distance maximale au capteur.
min_azimuth (double)	Azimut minimal (fenêtrage angulaire).
max_azimuth (double)	Azimut maximal (fenêtrage angulaire).
edge_depth_gap_threshold (double)	Seuil d'écart de profondeur pour détecter une arête.
edge_nb_gap_points (int)	Nb. de points consécutifs constituant le “gap”.
edge_intensity_gap_threshold (double)	Seuil d'écart d'intensité pour détecter une arête.
downsampling.max_points (int)	Nb. max. de KP conservés par frame (après échantillonnage).
downsampling.voxel_grid_resolution (double)	Résolution (m) de la voxelisation pour l'échantillonnage.
input_sampling_ratio (double)	Ratio d'échantillonnage des points d'entrée [0..1].

# Bibliographie

- [1] Institut Pascal. Plan d'accès — institut pascal, 2025. URL <https://www.institutpascal.uca.fr/index.php/fr/plan-d-acces>. Consulté le 13 août 2025.
- [2] Institut Pascal. Organigramme de l'institut pascal, 2025. URL [http://www.institutpascal.uca.fr/images/documents/organigramme\\_IP.pdf](http://www.institutpascal.uca.fr/images/documents/organigramme_IP.pdf). Consulté le 13 août 2025.
- [3] Institut Pascal. Organigramme de l'axe ispr, 2025. URL [http://www.institutpascal.uca.fr/images/articles/ISPR/Organigramme\\_ISPR.pdf](http://www.institutpascal.uca.fr/images/articles/ISPR/Organigramme_ISPR.pdf). Consulté le 13 août 2025.
- [4] Institut Pascal. Plateformes expérimentales — équipements (dont pavin), 2025. URL <https://www.institutpascal.uca.fr/index.php/fr/l-institut-pascal/equipements>. Consulté le 13 août 2025.
- [5] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics : Modelling, Planning and Control*. Springer, 2010. ISBN 978-1849964507. doi : 10.1007/978-1-84628-642-1.
- [6] Notion de repère local et global. <https://www.utc.fr/mecagom4/MECAWEB/EXEMPLE/EX04/MEEA1.htm>. Consulté le 14 août 2025.
- [7] Tutorial : Motion models. <https://docs.mrpt.org/reference/latest/tutorial-motion-models.html>. Consulté le 14 août 2025.
- [8] Kudan. 3d lidar slam : The basics. <https://www.kudan.io/blog/3d-lidar-slam-the-basics/>, 2020. Consulté le 14 août 2025.
- [9] Kudan, Inc. 3d lidar slam : The basics. <https://www.kudan.io/blog/3d-lidar-slam-the-basics/>, 06 2022. Consulté le 13 août 2025.
- [10] Kitware, Inc. Kitware - open source solutions, 2025. URL <https://www.kitware.com/>. Consulté le 15 août 2025.
- [11] Kitware. Kitware lidar slam — open-source repository, 2025. URL <https://gitlab.kitware.com/>. Consulté le 15 août 2025.
- [12] Julia Sanchez and Kitware Europe. Lidar slam training – ros2 wrapper overview. <https://www.kitware.eu/lidar-slam-training/>, 2023. Session du 03/10/2023.
- [13] Institut Pascal. Ispr — presentation, 2025. URL <https://www.institutpascal.uca.fr/index.php/en/presentation-ispr>. Consulté le 13 août 2025.
- [14] Institut Pascal. Comsee — équipe ispr, 2025. URL <https://www.institutpascal.uca.fr/index.php/fr/comsee>. Consulté le 13 août 2025.
- [15] Institut Pascal. Ispr — persyst, 2025. URL <https://www.institutpascal.uca.fr/index.php/fr/per sist>. Consulté le 13 août 2025.
- [16] Institut Pascal. Présentation de l'institut pascal, 2025. URL <https://www.institutpascal.uca.fr/index.php/fr/l-institut-pascal/presentation>. Consulté le 13 août 2025.

- [17] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.
- [18] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam) : Part ii. *IEEE Robotics & Automation Magazine*, 13(3) :108–117, 2006.
- [19] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping : part i. *IEEE Robotics & Automation Magazine*, 13(2) :99–110, 2006.
- [20] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping : Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6) :1309–1332, 2016.
- [21] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap : An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3) :189–206, 2013.
- [22] Helen Oleynikova, Alexander Millane, Zachary Taylor, Juan Nieto, Roland Siegwart, and Cesar Cadena. Voxblox : Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1366–1373, 2017.
- [23] Ji Zhang and Sanjiv Singh. Loam : Lidar odometry and mapping in real-time. In *Robotics : Science and Systems (RSS)*, 2014. URL <https://www.ri.cmu.edu/publications/loam-lidar-odometry-and-mapping-in-real-time/>.
- [24] Jens Behley and Cyrill Stachniss. Efficient surfel-based slam using 3d laser range data. In *Robotics : Science and Systems (RSS)*, 2018.
- [25] Tixiao Shan, Brendan Englot, Carlo Ratti, and Daniela Rus. Lio-sam : Tightly-coupled lidar inertial odometry via smoothing and mapping. In *IEEE/RSJ IROS*, 2020. doi : 10.1109/IROS45743.2020.9341174.
- [26] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics : The KITTI dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012. doi : 10.1109/CVPR.2012.6248074.
- [27] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IEEE/RSJ IROS*, 2012. doi : 10.1109/IROS.2012.6385773.
- [28] Nasr Eddine Saadallah. 3d mapping in unstructured and gps-denied environments. Master’s thesis, Institut Pascal / Université d’Évry Paris-Saclay, 2025. Mémoire utilisé comme template et source contextuelle.
- [29] Tixiao Shan and Brendan Englot. LeGO-LOAM : Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *IEEE/RSJ IROS*, pages 4758–4765, 2018. doi : 10.1109/IROS.2018.8594299.
- [30] Peter Biber and Wolfgang Straßer. The normal distributions transform : A new approach to laser scan matching. In *IEEE/RSJ IROS*, pages 2743–2748, 2003.

- [31] Martin Magnusson. *The Three-Dimensional Normal-Distributions Transform : An Efficient Representation for Registration, Surface Analysis, and Loop Detection*. PhD thesis, Örebro University, 2009.
- [32] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2) :239–256, 1992. doi : 10.1109/34.121791.
- [33] Yang Chen and Gérard Medioni. Object modeling by registration of multiple range images. *Image and Vision Computing*, 10(3) :145–155, 1992. doi : 10.1016/0262-8856(92)90066-C.
- [34] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics : Science and Systems (RSS)*, 2009. URL [https://cs.stanford.edu/people/avsegal/generalized\\_icp.html](https://cs.stanford.edu/people/avsegal/generalized_icp.html).
- [35] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *IEEE ICRA*, pages 1271–1278, 2016. doi : 10.1109/ICRA.2016.7487258.
- [36] Wei Xu, Yuyang Xu, Fu Li, Yuheng Wang, Chao Xu, and Wei He. Fast-lio2 : Fast direct lidar-inertial odometry. *IEEE Trans. Robotics*, 2022. doi : 10.1109/TRO.2022.3153837.
- [37] Giseop Kim, Ayoung Kim, and Others. Mulran : Multimodal range dataset for urban place recognition. In *IEEE ICRA*, 2020. doi : 10.1109/ICRA40945.2020.9196922.
- [38] Wenxian Yu and Maurice Fallon. The newer college dataset : Handheld lidar, inertial and vision with ground truth. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4353–4360, 2020. doi : 10.1109/IROS45743.2020.9340695.
- [39] Nicholas Carlevaris-Bianco, Anush Mohan, and Ryan M. Eustice. University of michigan north campus long-term vision and lidar dataset. In *The International Journal of Robotics Research (IJRR)*, volume 35, pages 1023–1035, 2016. doi : 10.1177/0278364915614638.
- [40] Sameer Agarwal, Keir Mierle, et al. Ceres solver. <http://ceres-solver.org>, 2025. Bibliothèque d’optimisation non linéaire à moindres carrés.
- [41] Radu Bogdan Rusu and Steve Cousins. 3d is here : Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation (ICRA) Workshops*, 2011.
- [42] Julia Sanchez. Lidar slam training. Kitware – slides internes/formation, 10 2023. Schéma du pipeline LidarSlam.
- [43] John Wang and Edwin Olson. Apriltag 2 : Efficient and robust fiducial detection. In *IEEE/RSJ IROS*, 2016.
- [44] Frank Dellaert and Luca Carlone. Factor graphs for robot perception. *Foundations and Trends in Robotics*, 6(1-2) :1–139, 2017.
- [45] Kitware. Lidar-slam – ros 2 wrapper readme. <https://gitlab.kitware.com/keu-computervision/slam/>. Consulté pour la documentation des capteurs externes et des commandes de loop closure/PGO.
- [46] Kitware. Kitware lidar-slam is available with ros and ros 2. <https://www.kitware.com/>, 2020.
- [47] Jean-Emmanuel Deschaud. Imls-slam : Scan-to-model matching based on 3d data. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2480–2485, 2018.

- [48] Keisuke Koide, Jun Miura, and Emanuele Menegatti. Portable 3d lidar-based system for long-term and wide-area people behavior measurement. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5305–5312, 2020. doi : 10.1109/IROS45743.2020.9341263. URL [https://github.com/koide3/hdl\\_graph\\_slam](https://github.com/koide3/hdl_graph_slam).
- [49] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and François Goulette. Ct-icp : Real-time elastic lidar odometry with loop closure. *arXiv*, 2021. URL <https://arxiv.org/abs/2109.12979>.
- [50] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. Fast-lio2 : Fast direct lidar-inertial odometry. *arXiv*, 2021. URL <https://arxiv.org/abs/2107.06829>.
- [51] Ignacio Vizzo, Tiziano Guadagnino, Benedikt Mersch, Louis Wiesmann, Jens Behley, and Cyrill Stachniss. Kiss-icp : In defense of point-to-point icp – simple, accurate, and robust registration if done the right way. *IEEE Robotics and Automation Letters*, 8(2) :1029–1036, 2023. doi : 10.1109/LRA.2023.3236571.