Vers une autonomie robotique d'exploration et de reconstruction simultanées d'environnements complexes

Anthony Brunel

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITE DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche : Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier

Vers une autonomie robotique d'exploration et de reconstruction simultanées d'environnements complexes

Présentée par Anthony BRUNEL

Le 18 Noveaumbre 2022

Sous la direction de Olivier STRAUSS et Cédric DEMONCEAUX

Devant le jury composé de

MARCHAND Nicolas, Directeur de Recherche CNRS, GIPSA-Lab, Grenoble INP	Président
VASSEUR Pascal, Professeur des Universités, Université de Picardie Jules Verne	Rapporteur
MARZAT Julien, Directeur de Recherche, HDR, ONERA	Rapporteur
ALLIBERT Guillaume, Maitre de Conférences, HDR, Université Côte d'Azur	Examinateur
FARAJ Noura, Maitre de Conférences, Université de Montpellier	Examinatrice
DEMONCEAUX Cédric, Professeur des Universités, Université de Bourgogne	Directeur de thèse
STRAUSS Olivier, Maître de Conférences, HDR, Université de Montpellier	Directeur de thèse
PÈRE Christian, Maître de Conférences, Gambi-M	Invité
LANDRIEU Jérémie, Docteur, Gambi-M	Invité

UNIVERSITÉ DE MONTPELLIER

Résumé – Dans ces travaux de thèse, nous abordons les problèmes de l'exploration et de la reconstruction 3D autonome d'environnements complexes. La méthode que nous proposons utilise un drone quadrirotor équipé d'un capteur de localisation et d'une caméra de profondeur. Dans l'objectif de résoudre ces problèmes, nous proposons SplatPlanner une méthode originale d'exploration autonome efficace. SplatPlanner réalise simultanément l'estimation de trajectoires d'exploration sans collisions et une reconstruction voxelique de l'environnement. Pour réaliser cette double tâche de manière efficace, nous présentons une méthode originale d'exploration s'appuyant sur le filtrage bilatéral. Cette méthode repose sur l'attribution d'un score d'intérêt aux voxels situés aux frontières des régions précédemment visitées afin de guider l'échantillonnage puis la sélection de trajectoires à exécuter. La faisabilité pratique de la mise en place de notre système a été éprouvée lors d'un vol réel, à bord d'un drone quadrirotor. Pour tester, évaluer et comparer équitablement les méthodes d'exploration autonome, nous proposons FLYBO, un environnement de simulation dédié aux méthodes d'exploration autonome utilisant un drone quadrirotor. Dans cette contribution, nous proposons des modèles de scènes 3D d'intérieur et d'extérieur réalistes, de complexités et de tailles variées. Nous fournissons des outils et des métriques pertinents pour l'évaluation comparative des performances d'exploration et de reconstruction de surfaces pour les méthodes d'exploration autonome. Nous proposons également une étude comparative de sept méthodes d'exploration issues l'état de l'art évaluées dans le cadre de FLYBO. Ces expériences illustrent en particulier le positionnement relatif de SplatPlanner comme étant le meilleur compromis entre efficacité d'exploration et justesse de reconstruction, à l'état de l'art.

Abstract – In this thesis, we address the problem of autonomous exploration and online 3D reconstruction of complexe environments using a quadrotor equipped with a localization sensor and a depth camera. To solve these problems, we present SplatPlanner, a novel autonomous exploration method that allows to simultaneously perform a collision-free exploration and a voxel-based reconstruction within a given environment. In order to perform this two task efficiently, we propose a new exploration algorithm based on a novel Permutohedral Frontier Filtering (PFF) which relies on a combination of highly efficient operations stemming from bilateral filtering. Specifically, our PFF aggregates spatial information about unobserved voxels which lie along the boundaries of unknown regions into density scores to guide the entire exploration. We propose a real flight experiment with a custom-built quadrotor which support the feasibility of our method in real conditions. In order to fairly and accurately assess the performance of autonomous exploration methods, we propose FLYBO, a simulation environment tailored for the task of autonomous exploration planning and online surface reconstruction using a quadrotor. FLYBO includes (i) 11 challenging realistic indoor- and outdoor scenes of increasing complexity and size, (ii) a comprehensive benchmark of 7 of the top-performing autonomous exploration algorithms, including methods without publicly available code, (iii) a unified experimental system factorizes the routines shared by autonomous planners in order to fairly and accurately assess their exploration performance in a controlled environment. In particular, SplatPlanner emerges as the best compromise regarding key performance metrics between exploration and surface reconstruction amongst the leading state-of-the-art systems.

Table des matières

Introduction

Introduction

FIGURE 0.1 – Campagne réalisée avec Gambi-M. Essai de vol télépiloté dans une infrastructure industrielle extrêmement complexe. Relevé topographique avec le dispositif Faro Focus X130.

Le relevé topographique et la reconstruction 3D d'environnements ou d'infrastructures sont essentiels pour une grande variété d'applications allant de la création de contenu numérique 3D jusqu'à l'aide à la navigation de robots ou de véhicules autonomes. Une campagne de relevés topographiques est généralement réalisée à l'aide de dispositifs aidant à capturer des informations 3D colorisées d'un environnement sous forme de nuages de points 3D et d'images couleur. Par exemple certains dispositifs sont équipés d'un LiDAR[i] et d'une caméra couleur haute définition, ce qui leur permet de restituer des représentations photoréalistes en 3D précises et complètes. Les caméras restituant en temps réel des données de profondeur (Intel RealSense, Microsoft Kinect, Structure Core...), pourraient apparaître comme une alternative plus compactes, plus légères et plus interessante car plus abordables (prix) que le LiDAR, mais les versions actuellement disponibles ne peuvent pas rivaliser avec ce dernier en termes de précision.

Gambi-M, l'entreprise finançant cette thèse CIFRE, utilise des données rapportées de relevés topographiques d'infrastructures, en majorité industrielles, afin de les enrichir et ainsi construire des maquettes numériques 3D de très haute précision. Ce type de maquette touche de nombreuses applications allant d'une planification de maintenance jusqu'à l'étude du démantèlement d'une infrastructure industrielle.

i. LiDAR (Laser imaging Detection and Ranging) : dispositif de télédétection par laser utilisant l'analyse du faisceau de lumière renvoyé vers son émetteur pour mesurer des distances. Ce type de dispositif permet par balayage du laser d'obtenir des nuages de points 3D.

Il existe plusieurs façons de réaliser de tels relevés. La méthode traditionnelle emploie des opérateurs humains qui déplacent séquentiellement un dispositif LiDAR-Caméra. Avec les dispositifs de haute précision disponible actuellement, une acquisition d'excellente qualité peut nécessiter plusieurs minutes, minutes pendant lesquelles le dispositif ne doit pas être déplacé. Sachant que le temps d'une campagne est corrélé au nombre de relevés effectués, le nombre de déplacements (acquisition) doit être minimal pour que cette campagne soit d'une durée minimale, et donc d'un coût minimal. Mais, en pratique, positionner correctement un tel dispositif pour réaliser une campagne dans un temps minimal, tout en gardant le même niveau de qualité, est difficile voire impossible dans les cas les plus complexes. La complexité et l'envergure des environnements dans lesquels sont effectuées ces campagnes rendent très difficiles, pour un opérateur humain, de juger de la qualité de ses relevées. L'opérateur n'est pas guidé de manière idéale (retour visuel souvent insuffisant) et peut, par exemple, difficilement s'assurer que toute l'infrastructure ait été relevée. À ceci s'ajoute le fait que des éléments du décor situés en hauteur sont souvent hors d'atteinte de l'opérateur. En milieu hostile (e.g. centrale nucléaire) un voire deux opérateurs sont nécessaires auxquels s'additionne l'achat ou la location du dispositif. Tout ceci rend une campagne de relevés de ce type couteuse pour parfois des résultats insatisfaisants car incomplets.

Une alternative intéressante aux relevés mobilisant un opérateur humain serait un système de cartographie et de reconstruction 3D autonome à l'aide de robots mobiles. Passer d'un contrôle humain haut niveau à une complète autonomie robotique réduirait considérablement les risques et coûts opérationnels de ces campagnes. Cela rendrait aussi possible la digitalisation de zones inaccessibles aux opérateurs humains en cas d'irradiation par exemple.

En utilisant un drone volant, cette alternative présenterait des avantages supplémentaires comme le fait d'accéder à des milieux difficilement atteignables pour un robot terrestre ou de capturer des éléments de l'infrastructure dont la mesure est inaccessible depuis le sol. Cependant, un drone volant possède des contraintes de charges embarquées rendant pratiquement impossible l'utilisation de dispositifs LiDAR-Caméra de hautes précisions, qui sont trop encombrants pour qu'un drone ainsi équipé puisse voler dans des infrastructures industrielles. À ceci s'ajoute les contraintes d'autonomie de batterie généralement limitée à 25-30 minutes.

L'automatisation des relevés topographiques peut être associée à l'exploration autonome qui fait l'objet de nombreux travaux en robotique depuis une trentaine d'années [1-3]. Les algorithmes d'exploration autonome visent à capturer au mieux l'ensemble d'un environnement inconnu délimité (au préalable), d'éventuellement maximiser la qualité de la reconstruction 3D et simultanément de réduire la durée de l'exploration.

Les recherches rapportées dans ce manuscrit se concentrent en priorité sur l'exploration autonome efficace pour la cartographie et la reconstruction 3D à l'aide d'un drone volant, celui-ci étant équipé de divers capteurs légers (caméra RGB-D [ii], unités de mesure inertielle (IMU) ...).

ii. les caméras RGB-D sont des dispositifs constitués d'une caméra couleur (RGB) et d'un capteur de profondeur (D).

Plan

Ce manuscrit est organisé en quatre chapitres. Le premier chapitre présente les algorithmes essentiels servant à développer un système robotisé autonome. Nous portons une attention particulière aux drones quadrirotors. Dans ce chapitre, nous y retrouvons des techniques de localisation et des techniques de cartographie simultanées, la cartographie d'occupation et de planification de trajectoire, des techniques d'asservissement pour le drone quadrirotor et d'exploration autonome. Le deuxième chapitre concerne les travaux menés sur la problématique d'exploration autonome. Dans ce chapitre, nous proposons une méthode de cartographie d'occupation ainsi qu'une méthode originale d'exploration autonome efficace. Dans le troisième chapitre, nous proposons un environnement de simulation (FLYBO) développé pour évaluer des méthodes d'exploration autonomes avec un drone quadrirotor équipé d'une caméra de profondeur. Notre contribution propose des environnements 3D réalistes, des métriques mesurant les performances des méthodes d'exploration, des routines algorithmiques partagés par les méthodes d'exploration autonomes ainsi que l'intégration de sept méthodes d'exploration provenant de la littérature. Le dernier chapitre analyse les résultats rapportés d'expériences menées tout au long de nos recherches. Dans ce chapitre, nous commençons par l'analyse de résultats obtenus à l'aide d'expériences faites en simulation (dans FLYBO). Nous terminons en présentant et commentant une expérience d'exploration autonome réalisée en condition de vol réel avec SplatPlanner [3]

Nous concluons ce manuscrit par un bilan de nos contributions et proposons des perspectives d'amélioration sur chacune de ces contributions.

Chapitre 1

De la localisation à l'exploration autonome

1.1 Introduction

Ces dernières années, les robots mobiles et plus précisément les drones quadrirotors suscitent une attention particulière de la part de la communauté scientifique. Maintenant, les drones sont utilisés dans de nombreuses applications parmi lesquelles la cartographie, l'inspection et la maintenance d'infrastructures, la reconstruction 3D, la recherche et le sauvetage de personnes perdues en milieu hostile, etc. L'un des nombreux défis scientifiques existant dans le domaine de la robotique de la vision par ordinateur et de l'informatique

graphique porte sur l'automatisation de telles applications. L'une des pistes de recherche pour automatiser ces applications (en particulier la cartographie et la reconstruction 3D) à l'aide d'un drone quadrirotor est l'usage d'un système de navigation autonome adjoint à un algorithme d'exploration autonome adapté.

L'exploration autonome est un domaine de recherche de longue date, qui a vu l'émergence des robots mobiles comme outils pertinents pour cette tâche. Cette évolution est due à l'amélioration des méthodes de localisation, de cartographie et d'asservissement robotique, rendant ces systèmes plus fiables et performants. La conception de systèmes de navigation et d'exploration autonomes peut être dissociée en cinq grands axes de recherche interagissant de manière plus ou moins complexe.

Ce chapitre est organisé en cinq sections. La première section porte sur le **SLAM** (Simultaneous Localisation And Mapping), méthode de localisation indispensable en l'absence de système global de positionnement. Savoir se localiser permet au robot équipé de capteurs tels qu'une caméra de profondeur de construire une cartographie d'occupation. Les cartographies d'occupation sont abordées dans la seconde section. Une carte d'occupation sert à informer le robot sur les éléments physiques peuplant son environnement. Le SLAM et la cartographie d'occupation permettent de planifier et de construire des trajectoires sans risque. Nous expliquons dans la troisième section comment sont construites ces trajectoires et présentons les méthodes qui permettent de le faire notamment celles appliquées aux drones quadrirotors. Ensuite, les ordres de trajectoires sont transmis et traités par une stratégie d'asservissement (de contrôle) adaptée au robot utilisé. La quatrième section présente brièvement le modèle dynamique d'un drone quadrirotor et les méthodes d'asservissement développées par la communauté scientifique. Enfin, nous présentons des travaux scientifiques portant sur l'exploration autonome d'environnements inconnus à l'aide de robots mobiles.

1.2 Localisation et cartographie simultanée

Le SLAM vise à résoudre le problème de localisation d'un robot ou d'un dispositif équipé de capteurs tels que des caméras monoculaires, des caméras stéréoscopiques, des caméras RGB-D, des unités de mesure inertielle (IMU), des LiDAR et d'autres. Le SLAM est un sujet extrêmement populaire dans la communauté scientifique depuis plus de 30 ans [4]. La popularité de ce problème est portée par l'émergence de diverses applications telles que la réalité augmentée avec comme exemple les lunettes *Microsoft Hololens*[i] ainsi que des sujets de robotique mobile tels que les véhicules autonomes ou bien l'exploration autonome. Pour résoudre le problème de localisation en robotique mobile, il est possible en premier lieu de penser à équiper le robot d'un système de positionnement global (GPS). Néanmoins, les applications ont rapidement montré que le GPS n'est pas toujours suffisant, notamment quand le signal est faible ou inexistant (e.g. en intérieur). Le SLAM est donc considéré comme l'un des verrous technologiques à une réelle autonomie en robotique.

i. www.microsoft.com/fr-fr/hololens

Le SLAM en robotique est formalisé généralement de manière probabiliste. L'objectif principal comprend simultanément (i) l'estimation de l'état du robot ou du capteur (localisation) et (ii) la construction ou l'amélioration d'une carte de l'environnement issue de la perception et des mesures de capteurs embarqués (cartographie). Typiquement, un modèle probabiliste prend en entrée une mesure bruitée Z et en déduit une estimation X modélisant l'état du robot. Dans les cas les plus simplistes, l'état du robot se réduit à sa pose (position et orientation), bien que d'autres mesures puissent être incluses comme la vitesse du robot, les biais des capteurs ou encore les paramètres d'étalonnages... Dans cette thèse, nous nous appuyons sur une localisation 3D avec position, orientation ainsi que d'autres grandeurs telles que la vitesse et l'accélération, ce sont les grandeurs généralement utilisées pour suivre et guider un robot. La cartographie, quant à elle, est construite à partir de repères d'intérêts acquis simultanément par la méthode de SLAM (par exemple, la position du robot, des points caractéristiques, des repères mesurés par les capteurs...). Ces repères décrivent l'environnement dans lequel le robot opère et servent à maintenir et à ajuster son positionnement au cours du temps. Il existe plusieurs articles de synthèse sur le SLAM sur une grande période abordant les diverses solutions proposées par la littérature telles que de 1986-2004 [5, 6], de 2004-2011 [7] et 2006-2016 [8]. Cadena et al. [8] proposent un découpage en trois périodes : la première ère du SLAM se situe entre 1986 et 2004, les auteurs la nomme l'âge classique du SLAM. Durant cette période les premières méthodes de SLAM émergent. Ces méthodes comprennent principalement des approches basées filtrage ou s'appuyant sur l'estimation du maximum de vraisemblance. Durant la deuxième période, il est question d'analyse algorithmique. Des études approfondies sont faites sur les propriétés fondamentales du SLAM, l'efficacité de convergence des algorithmes développés et leurs robustesses. Les premières bibliothèques open-source sont déployées pour la recherche. Enfin, d'après Cadena et al. nous entrons actuellement dans la troisième période du SLAM, une période pendant laquelle devraient se développer des méthodes plus robustes, plus performantes et plus intelligentes.

Le reste de cette section s'organise de la manière suivante : dans un premier temps, nous présentons un formalisme standard du problème de SLAM permettant au lecteur d'en comprendre les principes fondamentaux. Ensuite, nous détaillons plusieurs façons de résoudre le SLAM associées à certains types de capteurs notamment ceux utilisés dans l'exploration autonome en robotique mobile. Enfin, nous présentons les méthodes pertinentes pour notre cas d'usage, i.e, celles qui nous permettent une localisation suffisamment robuste pour proposer une solution au développement d'un drone volant autonome.

1.2.1 Formalisme du problème de SLAM

Le SLAM est fréquemment défini comme l'estimation d'un problème de maximum a posteriori (MAP), consistant à estimer χ^*, paramètres inconnus qui maximisent la grandeur $p(\chi|Z)$, la probabilité d'obtenir les paramètres χ compte tenu des mesures Z. En robotique, nous pouvons par exemple définir χ comme un ensemble de variables aléatoires, correspondant typiquement aux poses du (ou des) capteur(s), du robot ainsi que les positions de repères caractéristiques (e.g. points caractéristiques extraits d'une image). Les capteurs

embarqués sur le robot restituent un ensemble de mesures $Z : (i = 1...n)$ qui peuvent être exprimées par une fonction de χ telle que $z_i = f_i(\chi_i) + \epsilon_i$ avec $\chi_i \in \chi$, f_i une fonction connue, par exemple, le modèle d'observation et ϵ_i le bruit de mesure. L'estimation du maximum a posteriori s'écrit alors comme suit :

$$\chi^* = \arg\max_{\chi} \ p(\chi|Z). \tag{1.1}$$

En appliquant le théorème de Bayes et en supposant $p(Z)$ constant l'équation (1.1) devient

$$\chi^* = \arg\max_{\chi} \ p(Z|\chi)p(\chi), \tag{1.2}$$

avec $p(Z|\chi)$ qui correspond au maximum de vraisemblance et $p(\chi)$ une probabilité a priori de χ. Celle-ci pouvant être écartée si aucun apriori n'existe sûr χ. Dans ce cas, le problème devient une estimation de maximum de vraisemblance. Si nous supposons que les mesures z_i sont indépendantes alors le problème peut se reformuler de la manière suivante :

$$\chi^* = \arg\max_{\chi} \ p(\chi) \prod_{i=1}^{n} p(z_i|\chi_i). \tag{1.3}$$

De ce fait z_i dépend uniquement du sous-ensemble χ_i.

En acceptant l'hypothèse que le bruit de mesure ϵ_i suit une distribution normale centrée en zéro alors l'équation (1.3) peut être écrite de manière plus explicite :

$$p(z_i|\chi_i) \propto exp(-\frac{1}{2}\|f_i(\chi_i) - z_i\|^2_{\Omega_i}), \ p(\chi) \propto exp(-\frac{1}{2}\|f_0(\chi_i) - z_0\|^2_{\Omega_0}), \tag{1.4}$$

où Ω_i est une matrice d'information, i.e., l'inverse de la matrice de covariance. Puisque maximiser l'équation (1.3) revient à minimiser le logarithme négatif des probabilités, l'équation (1.3) revient à minimiser la somme des termes suivants :

$$\chi^* = \arg\min_{\chi} \ -log(p(\chi) \prod_{i=1}^{n} p(z_i|\chi_i)) = \arg\min_{\chi} \ \sum_{i=0}^{n} -\frac{1}{2}\|f_i(\chi_i) - z_i\|^2_{\Omega_i}. \tag{1.5}$$

On obtient ainsi un problème de moindres carrés (non linéaire pour le cas du SLAM). Sous réserve qu'une solution approchée soit connue, le problème se résoud habituellement avec des algorithmes d'optimisation tels que Gauss-Newton (GN) (utilisé par [9]) ou Levenberg-Maquardt (LM) (utilisé par ORB-SLAM [10]).

Ce n'est pas la seule manière de résoudre le problème de SLAM. Il est possible d'utiliser des méthodes basées filtrage [11]. Les méthodes basées filtrage visent à résoudre le problème de SLAM en estimant, et mettant itérativement à jour, une distribution de probabilité conjointe sur tous les paramètres pertinents (état du robot, des capteurs, des repères caractéristiques). Cette approche est connue sous le nom de filtre de Kalman [12]. Elle est populaire pour son efficacité en temps de calculs. Strasdat et al. [13] montrent que les mé-

thodes de SLAM basées filtrage ont une meilleure précision en cas de ressources de calculs limitées. De ce fait pour des systèmes embarqués cette approche peut être pertinente due à une complexité algorithmique favorable. Des méthodes l'utilisent aussi pour pré-traiter les données de certains capteurs telles les données de l'IMU [14] avant de les intégrer dans un processus d'optimisation basé GN ou LM. L'accroissement de la puissance de calcul et une meilleure compréhension théorique du problème de SLAM par le biais de graphes factoriels donnent toutefois un avantage substantiel aux méthodes d'optimisation GN ou LM.

1.2.2 Graphe factoriel non-linéaire

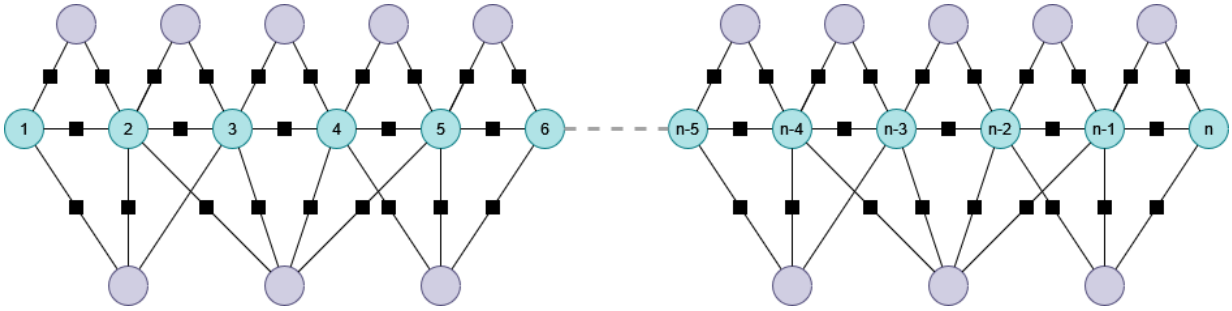


FIGURE 1.1 – Schéma d'un graphe factoriel illustrant le problème de SLAM en robotique mobile. Les cercles bleus sont les poses successives du robot, les cercles violets représentent des repères caractéristiques de l'environnement et les carrés noirs sont les facteurs, i.e., les fonctions qui relient des sous-ensembles de variables (e.g. pose du robot ou repère de l'environnement).

Les graphes factoriels non linéaires représentent une façon très intuitive de modéliser un grand nombre de problèmes en robotique dont le SLAM (quelle que soit la méthode pour le résoudre : GN, LM ou filtrage). Ils simplifient la conception et permettent une visualisation structurelle du problème. De nos jours, ils sont très utilisés dans de nombreuses bibliothèques de SLAM [15-17] et beaucoup repris dans les articles de la littérature portant sur le SLAM. Par ailleurs, les graphes factoriels illustrent une propriété de localité entre les paramètres du problème. Cette propriété montre que les matrices apparaissant dans une modélisation classique des problèmes de SLAM sont éparses. Chaque paramètre est uniquement lié à un sous-ensemble d'autres paramètres. Cette propriété est illustrée par la Figure 1.1. Les matrices éparses permettent aux bibliothèques de SLAM une résolution plus efficace du problème tel que l'inférence du MAP.

En guise d'exemple, on peut mettre en parallèle l'estimateur MAP pour le SLAM décrit précédemment et la conception du problème par graphes factoriels. Dans ce type de graphe les nœuds représentent les variables aléatoires χ (les paramètres du problème à estimer). Les arêtes représentent $p(z_i|\chi)p(X)$ ou plus particulièrement la fonction à minimiser définie dans l'équation 1.5).

1.2.3 SLAM Visuel

En vision par ordinateur comme en photogrammétrie, l'ajustement de faisceaux [18] est une approche qui consiste à affiner simultanément les coordonnées 3D décrivant la géométrie de la scène, les transformations relatives et parfois les caractéristiques optiques de la (ou des) caméra(s) utilisée(s) pour acquérir les images. Il existe une ressemblance évidente entre l'équation (1.5) et le problème d'ajustement de faisceaux. Le SLAM visuel (appelé aussi V-SLAM) dérive naturellement de cette approche. La différence principale entre l'ajustement de faisceaux et le V-SLAM vient du fait qu'en V-SLAM de nouvelles données sont intégrées continuellement et le problème est résolu de manière incrémentale. Tandis qu'en vision l'ajustement de faisceaux se fait sur un ensemble d'images définies au départ étant optimisé en une seule fois. De cette ressemblance, on en déduit que le SLAM visuel consiste à suivre des informations caractéristiques de l'environnement extraites dans une série d'images successives. Habituellement, l'objectif est alors d'estimer et d'affiner les coordonnées 3D de points caractéristiques tout en utilisant simultanément ces informations pour estimer les poses relatives entre les images. Les images sont généralement acquises par une caméra embarquée sur un robot. Les poses relatives et les coordonnées 3D décrivant la géométrie de la scène sont systématiquement intégrées dans la cartographie et réutilisées plus tard pour les estimations suivantes.

Le SLAM Visuel est intrinsèquement lié au traitement de l'image. Il suit les avancées de ce domaine. Les approches de V-SLAM peuvent être catégorisées, d'une part, en méthodes indirectes et méthodes directes et d'autre part, en méthodes éparses et méthodes denses.

— **Les méthodes indirectes** comme déjà décrit brièvement, utilisent une phase de prétraitement qui consiste à extraire des informations caractéristiques [19-21] sur l'image pour ensuite établir des correspondances. Cette étape permet de traiter ces informations caractéristiques comme des mesures bruitées afin d'estimer le déplacement du capteur [10, 22, 23]. Cette phase de prétraitement les rend de facto robustes aux problèmes photométriques (e.g. forts changements d'intensité).

— **Les méthodes directes** ne passent pas par une phase de prétraitement. Elles comparent directement les mesures du capteur (l'intensité mesurée en chaque pixel). Les méthodes directes minimisent une erreur photométrique tandis que les méthodes indirectes minimisent une erreur géométrique. La fonction d'erreur photométrique est fortement non-linéaire et seulement convexe localement. De ce fait, les méthodes directes requièrent une estimation initiale relativement proche de la solution optimale. En pratique, leur sensibilité lors de l'initialisation est un problème et donc limite leur utilisation. Elles sont aussi plus sensibles aux fort changements de lumière, aux forts déplacements et indirectement, à la fréquence d'acquisition des images dans le cas où la fréquence serait trop faible. En revanche, les méthodes directes peuvent exploiter l'image dans son ensemble, et délivrer une information plus dense dès lors qu'il existe une faible variation d'intensité entre les pixels. Également, ne pas dépendre de point caractéristique rend ces méthodes moins sensibles au flou sur l'image.

— **Les méthodes éparses** se caractérisent par une faible quantité d'informations extraites

de l'image. Habituellement, elles n'utilisent et ne reconstruisent qu'un sous-ensemble sélectionné de points indépendants (traditionnellement des coins de structure) [10, 22-24].

— **Les méthodes denses** tentent d'utiliser et de reconstruire tous les pixels du domaine de l'image 2D. Ce deuxième type d'approche requiert généralement une puissance de calcul importante. Elles sont en général optimisées à l'aide de cartes graphiques [25].

— **Les méthodes semi-denses** sont des approches intermédiaires qui s'abstiennent de reconstruire l'ensemble, mais visent toujours à utiliser et à reconstruire un sous-ensemble largement connecté [9].

En résumé, il est possible de catégoriser les méthodes de SLAM en quatre approches :

— **Les méthodes éparses+indirectes** sont la formulation la plus utilisée. Le problème de localisation et de reconstruction est résolu à l'aide de correspondances entre points caractéristiques tout en limitant le nombre de points extraits. Sa popularité est fortement liée à son efficacité et sa robustesse [10].

— **Les méthodes éparses+directes** s'appuient exclusivement sur une erreur photométrique sans associer d'information géométrique. L'erreur photométrique est définie par une somme des différences aux carrés sur un petit ensemble de pixels [26].

— **Les méthodes denses+directes ou "semi"-dense** s'appuient simultanément sur des a priori photométriques et géométriques [9].

— **Les méthodes denses+indirectes** utilisent généralement une phase de prétraitement par flot optique [27, 28]. Cette dernière stratégie est moins populaire dans la littérature, en particulier en robotique mobile.

La Figure 1.2 illustre 3 méthodes populaires couvrant 3 stratégies différentes de V-SLAM.

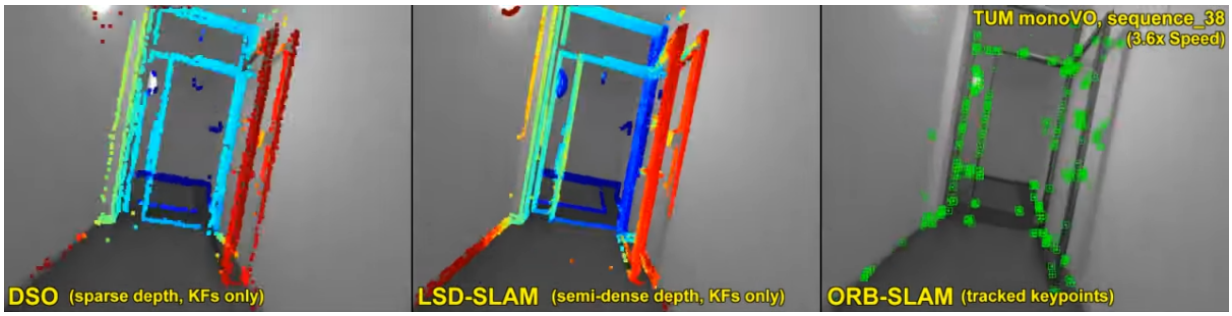


FIGURE 1.2 – Cette figure est extraite de `https://www.youtube.com/watch?v=C6-xwSOOdqQ` et [26]. C'est une représentation qualitative de l'utilisation de l'image par trois types de méthodes différentes. Les images de gauche à droite correspondent à DSO [26] méthode directe est éparse, à LSD-SLAM [9] méthode "semi"-dense direct et à ORB-SLAM [10] méthode éparse-indirecte

Le choix du nombre de caméras en V-SLAM visuel affecte divers aspects de la méthode et des résultats obtenus. Les méthodes de SLAM avec une caméra monoculaire ou SLAM

monoculaire nécessitent plusieurs images pour initialiser le suivi et la localisation. Cette méthode ne permet pas d'estimer l'échelle de l'environnement sans une mesure étalon. C'est donc une façon de se localiser à un facteur d'échelle près. Ce facteur peut, comme l'estimation de localisation, être sujet à une dérive. Or, en robotique mobile, la navigation nécessite de connaître l'échelle de la scène observée. C'est pourquoi les méthodes proposées dans la littérature associent aux caméras différents capteurs complémentaires tels que des IMU ou d'autres caméras. Cet ajout induit une étape supplémentaire d'étalonnage, positionner les capteurs les uns par rapport aux autres via une transformation isométrique 3D. Cette étape peut ajouter un biais qui dépend de la qualité de l'étalonnage, mais permet d'obtenir l'échelle de la scène sans l'utilisation d'a priori. Les capteurs RGB-D peuvent être aussi une alternative pour résoudre le problème de SLAM-Visuel [29, 30].

1.2.4 SLAM Visuel-Inertiel

Les méthodes récentes de SLAM pour la robotique associent très souvent des caméras et IMU [22, 23, 30]. On parle de VI-SLAM (Visual Inertial SLAM). Une IMU est un capteur qui acquiert des données à très hautes fréquences (plus de 100 mesures par seconde). L'IMU se compose habituellement d'un gyroscope mesurant la vélocité angulaire et d'un accéléromètre mesurant l'accélération linéaire. Malgré la nature bruitée des mesures produites par l'IMU, ce capteur permet un suivi de la position et de l'orientation par fusion et intégration des mesures. Par ailleurs, l'usage de l'accéléromètre permet d'éviter une dérive sur les rotations en roulis et tangage comme spécifié par [31]. La dérive concerne alors que les quatre degrés de liberté. Le couplage IMU-caméra permet un meilleur suivi de la trajectoire. Néanmoins, étant donné que ce type de capteur acquiert des données à une plus grande fréquence que les capteurs de vision, il est nécessaire de les pré-traiter. En général, les méthodes proposées dans la littérature font un compromis entre le temps de calcul et la précision. Typiquement, les méthodes basées filtrages permettent un traitement rapide de toutes les données, mais la précision est détériorée par le filtrage. D'autre part, les méthodes basées optimisation GN ou LM sont précises, mais coûteuses en temps de calcul et ne peuvent pas traiter toutes les données. Lupton et al. [32] proposent de combiner et d'intégrer les données IMU sur une période d'acquisition définie (généralement le temps d'acquisition entre deux images) afin de limiter l'estimation à une seule contrainte de déplacement. Ainsi, il est possible de se servir de cette contrainte comme information supplémentaire dans le processus d'optimisation effectué à chaque nouvelle acquisition d'image. Forser et al. approfondissent et fournissent plus de détails sur cette approche qui consiste à pré-intégrer les données IMU. Cette approche est reprise maintenant par de nombreuses méthodes de VI-SLAM récentes, telles que Qin et al. [31] Mur et al. [33] ou encore Usenko[23].

1.2.5 SLAM basé nuages de points

Une alternative aux méthodes de localisation basées vision monoculaire ou stéréo est l'utilisation de capteur de distance ou de profondeur. Ce sont des capteurs extéroceptifs

qui peuvent être passifs (e.g. caméra RGB-D Intel RealSense D455) ou actifs (e.g. LiDAR) et permettent d'obtenir un nuage de points 3D structuré. Dans ce cas, il est possible de réaliser la localisation en utilisant un algorithme de type ICP (Iterative closest points) [34]. Des variantes existent pour recaler des arêtes ou des plans (informations extraites du nuage de points) [35, 36]. Le principe de ce type d'algorithme consiste à minimiser une erreur de distance associant chaque nouveau point à son plus proche voisin dans la cartographie existante. Par itération successive, l'ICP retrouve le déplacement relatif entre deux nuages de points. En général, les méthodes basées ICP utilisent des capteurs LiDAR. Ce sont des systèmes d'acquisition actifs utilisant le temps de vol d'un ou plusieurs faisceaux de lumière émis afin d'établir une distance entre la source et la surface touchée. Des variantes associent LiDAR, caméra et IMU [37, 38] permettant ainsi de rendre les systèmes de localisation plus sophistiqués.

Le LiDAR est un dispositif possédant certaines limitations : c'est un dispositif plutôt cher, lourd et fragile par rapport aux capteurs visuels et inertiels. Les LiDAR sont beaucoup plus complexes à intégrer et à transporter sur un projet d'acquisition quand il s'agit d'environnements complexes avec des espaces restreints ou encombrés.

1.2.6 Synthèse de méthodes performantes de SLAM extrait de la littérature

	Estimation	Mono	Stereo	Fisheye	IMU	LiDAR	Précision	Robustesse
SVO [39]	BA	✓	✓	-	-	-	Très bonne	Très bonne
OKVIS [40]	BA	✓	✓	✓	✓	-	Bonne	Très bonne
ROVIO [41]	EKF	✓	✓	✓	✓	-	Bonne	Très bonne
ORBSLAM 3[30]	BA	✓	✓	✓	✓	-	Exc.	Exc.
VINS-Fusion [22]	BA	✓	✓	✓	✓	-	Très bonne	Exc.
DSO [26, 42-44]	BA	✓	✓	✓	✓	-	Bonne	Très bonne
BASALT [23]	BA	✓	✓	-	-	-	Très bonne	Très bonne
Kimera [45]	BA	-	✓	-	✓	-	Très bonne	Très bonne
T265 [46]	BA	✓	✓	-	-	✓	Exc.	Exc.
LOAM [47]	ICP			-	-	✓	Exc.	Très bonne
Lego-loam[48]	ICP	-	-	-	-	✓	Exc.	Très bonne
LINS [49]	ICP	-	-	-	✓	✓	Exc.	Exc.
Lio-sam[50]	ICP	-	-	-	✓	✓	Exc.	Exc.

Table 1.1 – Liste des méthodes de SLAM disponible et utilisable pour le développement d'un robot mobile autonome.

Dans cette sous-section, nous proposons une synthèse des méthodes développées par la communauté scientifique et le secteur industriel. Au travers de cette synthèse, nous souhaitons faciliter les recherches et le choix de la méthode de SLAM pour le développement

d'un robot mobile autonome. Nous présentons les méthodes sous la forme d'un tableau Table 1.1. Dans ce tableau, nous caractérisons chaque méthode à l'aide de différents critères. Le premier de ces critères est l'algorithme principal permettant d'estimer la localisation. Les trois acronymes utilisés sont :

— BA : pour *Bundle Adjustment*. Nous utilisons cet acronyme quand la méthode de localisation se fait à la manière de l'ajustement de faisceaux (V-SLAM) comme vu section 1.2.3.

— EKF : pour filtre de Kalman étendu. Certaines méthodes de VI-SLAM s'appuient uniquement sur le filtre de kalman étendu pour estimer la localisation.

— ICP : pour *Iterative Closest Point*. L'algorithme ICP est utilisé dans le cas de capteur LiDAR comme expliqué 1.2.5.

Le deuxième critère concerne les types de capteurs utilisés par chacune des méthodes. Nous retrouvons parmi ces capteurs, la caméra monoculaire (Mono), la caméra stéréoscopique (Stereo), la caméra fisheye[ii] Fisheye, l'unité de mesure inertiel IMU et le LiDAR. Enfin en troisième critère, nous proposons une appréciation qualitative de précision et de robustesse de chaque approche. Cette appréciation s'appuie sur les travaux de [30] pour la partie SLAM visuel et inertiel, ainsi que le recoupement des résultats présentés par les auteurs des différentes méthodes pour la partie LiDAR. Pour finir, mis à part le capteur T265 développé par l'entreprise Intel toutes sont open-sources disponible en ligne.

1.3 Carte d'occupation

La navigation autonome en robotique nécessite, pour un robot, d'avoir à tout instant une connaissance partielle ou totale des obstacles peuplant l'environnement dans lequel il évolue. Ceci permet au robot d'effectuer des déplacements sans risque de collision. Bien que la cartographie établie par les méthodes de SLAM puisse être envisagée, elle s'avère spécifique à la résolution du problème de localisation. Pour effectuer une navigation sans risque, la plupart des travaux cités dans la littérature utilisent habituellement une cartographie d'occupation.

La cartographie d'occupation est un processus algorithmique visant à insérer des informations spatiales exprimant la position (soit dans le repère du robot, soit dans le repère monde) des éléments physiques existants dans l'environnement (e.g. murs, objets, etc.). Plus précisément, la cartographie d'occupation modélise à l'aide d'une carte d'occupation les régions navigables (sans obstacle), les régions occupées (avec des obstacles) et les régions inconnues où aucune information n'a encore été acquise.

Habituellement, une carte d'occupation est une représentation de l'environnement s'appuyant sur la notion de cellules (partitionnement 2D ou 3D de l'espace). Chaque cellule contient des informations hétérogènes telles que son état (e.g. occupé, libre), sa probabilité

ii. Les caméras fisheye sont des capteurs de vision qui ont la particularité d'avoir une distance focale très courte. Ceci leur confère un champ de vision plus grand (+180° diagonale) mais aussi une très forte distorsion de l'image.

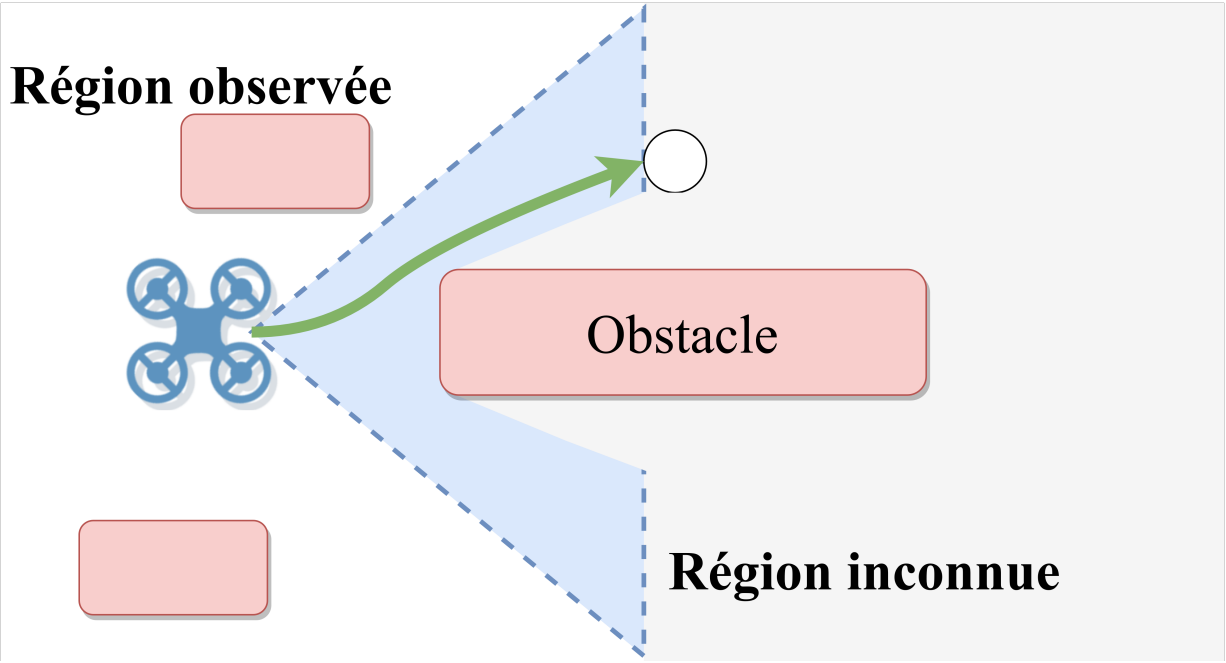


FIGURE 1.3 – Illustration schématique de la construction d'une carte d'occupation et d'un robot qui se déplace. La région observée (en blanc) correspond à un espace navigable. La zone grisée est une région inconnue. Les obstacles sont des éléments physiques de la scène, ils sont symbolisés en rouge. Le champ de vision du capteur est représenté en bleu.

d'occupation ou encore la distance à l'obstacle le plus proche. Ces informations concernent généralement la surface ou le volume que la cellule englobe. La taille des cellules ou leur résolution (exprimée en mètre) est déterminée automatiquement par l'algorithme de cartographie d'occupation ou spécifiée par l'utilisateur. Une valeur de résolution faible permet un découpage plus précis de l'environnement (des cellules plus petites) tandis qu'une valeur élevée occasionnera un découpage plus grossier mais est utile dans le cas de ressources de calculs limitées (avantages : espace mémoire utilisée plus faible, mise à jour des voxels plus rapides). La carte d'occupation est construite incrémentalement et mise à jour à chaque nouvelle mesure. Les mesures sont généralement issues d'un capteur de distance (e.g. LiDAR, caméra de profondeur). La mise à jour d'une carte d'occupation consiste à associer ou modifier les informations des cellules, intersectant la région observée par le capteur. La Figure 1.3 est une représentation schématique des informations importantes que doit contenir une carte d'occupation pour permettre l'estimation d'une trajectoire sans risque.

La construction de carte d'occupation repose sur les hypothèses de travail suivantes :

— Le robot est équipé d'un ou plusieurs capteurs de profondeur (ou de distance). Nous considérons dans cette section, ce qui semble une généralité dans tous les travaux que nous avons référencés, l'hypothèse suivante : les algorithmes de cartographie d'occupation que nous présentons considères que le robot est équipé d'une caméra de

profondeur (e.g. caméra stéreo, Microsoft Kinect, Intel RealSense D455...) ou d'un capteur LiDAR (e.g. Velodyne VLP16, Intel RealSense L515).

— Le robot est équipé d'un système de localisation (Section 1.2), étalonné vise à vis du capteur de profondeur/distance utilisé en particulier si la cartographie se fait dans le repère monde. Ceci implique que les paramètres extrinsèques entre le capteur de profondeur et le système de localisation soient également connus. Il est important de noter que les paramètres intrinsèques, de ces mêmes capteurs doivent aussi être connus.

— La dérive du système de localisation et le biais de mesure du capteur (distance ou profondeur) sont suffisamment faibles pour ne pas affecter la méthode de cartographie d'occupation. En général, les méthodes de cartographie d'occupation ne sont pas robustes à ce type d'erreur. En cas de dérive trop grande, la carte d'occupation ne représentera pas correctement l'environnement observé, ce qui peut provoquer des problèmes de navigation pour le robot.

Enfin, traditionnellement, une cartographie d'occupation doit présenter les caractéristiques suivantes :

1. **État d'occupation :** dans le but de planifier des trajectoires sans collision, il est nécessaire de prendre connaissance de l'état des cellules observées et de connaître les cellules appartenant aux régions non observées. Il est courant de définir trois états dans une carte d'occupation :
   — **"libre" :** la cellule est observée et est considérée sans obstacle, le robot peut naviguer dans toutes régions faites de cellules libres,
   — **"occupé" :** la cellule contient probablement un obstacle mesuré par le robot,
   — **"inconnu" :** la cellule n'a pas encore été observée, aucune information n'a été ajoutée en cette position dans la carte d'occupation,
   — **"frontière" :** la cellule est située à la frontière entre les régions navigables et inconnues. La section 1.6 introduit ce dernier concept.

   Le concept d'espace inconnu permet d'éviter de créer des trajectoires dans des espaces non observés contenant potentiellement des obstacles. Ce concept sert également à guider le robot vers ces régions afin de les explorer.
2. **Efficacité :** vu que la carte d'occupation est un composant clé dans la navigation en robotique, les méthodes de cartographie d'occupation doivent être efficaces et donc temps réel.
3. **Représentation probabiliste :** les cartes d'occupation construites à partir de capteurs de distance, sont entachées d'erreurs de mesures introduisant un bruit sur les valeurs estimées. Ces erreurs de mesures dépendent de la qualité du capteur utilisé et peuvent être pour certains capteurs de l'ordre du centimètre (<5cm) tandis que pour d'autres, ces erreurs peuvent atteindre une dizaine de centimètres. Des incertitudes supplémentaires peuvent s'y ajouter provenant d'erreurs d'estimation liées à

l'environnement et aux objets complexes qu'il contient comme par exemple, les réflexions d'objets (considérées comme des valeurs aberrantes) ou des obstacles mobiles pendant la construction d'une carte. Afin de construire un modèle fiable de l'environnement, la cartographie d'occupation agrège (ou fusionne) incrémentalement les nouvelles mesures aux anciennes tout au long de l'exploration. Ceci permet d'obtenir une estimation réelle et fiable de l'environnement. Ces mesures sont agrégées de manière probabiliste. Par ailleurs, une représentation probabiliste permet de fusionner les données provenant d'autres capteurs, voire d'autres robots [51].

1.3.1 Partitionnement 3D d'environnement

Il existe diverses manières de représenter un environnement discret par une structure de données informatiques. Cette sous-section présente les principales stratégies utilisées en robotique mobile. La première représentation couramment utilisée en robotique s'appuie sur les travaux de Moravec et al. [52], dans lesquels une grille régulière 2D est utilisée pour la cartographie d'une salle de laboratoire. La cartographie est faite à l'aide d'un robot mobile équipé de caméras et de sonars. Cette grille régulière discrétise l'espace en cellule de résolution identique. Elle a l'avantage de fournir un accès très rapide aux informations contenues dans les cellules. Cette stratégie a par la suite été étendue au domaine 3D. Il est maintenant commun de discrétiser l'espace 3D en cellules cubiques appelées voxels. Un voxel est un cube : c'est l'extension en 3D de la notion de pixel (pour une image). Le voxel représente un volume unitaire (de résolution variable) auquel il est possible d'associer diverses informations telles que l'état d'occupation ou la probabilité d'occupation, la distance à la surface la plus proche, etc. Les voxels sont organisés dans une structure de données permettant de reconstruire l'ensemble d'un environnement volumique, comme une grille régulière 3D. Cependant, construire une grille régulière (qui discrétise complétement l'environnement) sur trois dimensions, peut consommer beaucoup de ressources en termes de place en mémoire. De plus, cette structure est peu efficace si une taille approximative de l'environnement n'est pas connue au préalable (dans ce cas, la taille de la grille est réajustée en permanence, réduisant ainsi l'efficacité de ce procédé).

Des solutions s'appuyant sur les tables de hachage pallient ces problèmes [53-55]. La représentation d'une grille régulière par table de hachage permet d'alléger en mémoire la représentation informatique de l'environnement discrétisé. Une table de hachage utilise une fonction de hachage qui associe une clé (e.g. les coordonnées spatiales d'un voxel) et une valeur (information contenue dans un voxel ou bien un ensemble/bloc de voxels). L'utilisation d'une table de hachage permet si un voxel n'est pas observé (ou pas utile pour la tâche à résoudre) de ne pas allouer d'espace mémoire pour le représenter. Cette propriété réduit la quantité de mémoire utilisée et permet de construire une grille de résolution très fine (voxel plus petit), qui est une propriété très intéressante pour la reconstruction de surface [54, 55]. Cependant, le passage par une fonction de hachage affecte les performances d'insertion de nouvelles données. De plus, comme pour la grille dense, la résolution de ces voxels est fixe et ne peut être changée une fois le processus de cartographie initialisé.

Une autre structure de données communément utilisée en robotique mobile est l'Octree

[51, 56, 57]. Cette structure, de forme arborescente, utilise des voxels de résolution variable. L'espace tridimensionnel de départ est couvert par un voxel englobant qui est subdivisé récursivement jusqu'à la résolution voxelique souhaitée. Chaque nœud (octant) de l'Octree peut compter jusqu'à huit "enfants" (sous-noeud ou feuille). Comme un Octree est une structure de données hiérarchique, l'arbre peut être coupé à n'importe quel niveau pour obtenir une subdivision plus ou moins grossière, notamment si toute une branche possède le même état. Par exemple, si une branche de l'arbre est étiquetée comme libre, tout l'espace englobé par le nœud est donc libre. Cette propriété permet un gain en termes de mémoire utilisée. À l'inverse, la construction (e.g. insertion de nouvelle donnée) ou une requête (e.g. état d'un voxel) sur l'Octree nécessite dans le pire des cas de descendre entièrement l'arbre. Cette opération est de complexité logarithmique en fonction de la profondeur de l'Octree.

L'un des principaux usages du partitionnement d'espaces 3D en robotique mobile est la détermination de l'obstacle le plus proche, à partir d'une position donnée, afin d'établir des trajectoires permettant des déplacements sans risque. En général, cette distance est égale à l'envergure du robot à laquelle on ajoute une marge de sécurité. On peut aussi par sécurité légèrement dilater les régions occupées de la carte d'occupation. La façon dont est déterminé le voxel le plus proche ou la distance au voxel le plus proche d'une position donnée n'est pas la même lors de l'utilisation d'un Octree ou d'une grille régulière/éparse. Concernant l'Octree, déterminer le voxel occupé le plus proche d'une position donnée est réalisé en parcourant simplement l'arbre de la racine jusqu'à une feuille occupée en sélectionnant à chaque itération, le nœud dont le centre est le plus proche de la position donnée. Lorsqu'il s'agit d'une représentation par grille régulière ou éparse, les méthodes de cartographie d'occupation actuelles ajoutent la construction d'un champ de distance euclidien. La construction d'un champ de distance euclidien consiste à associer une valeur de distance (en mètre) entre le centre géométrique de chaque voxel de la grille et celui du voxel occupé le plus proche. Celle-ci peut être estimée soit à partir d'une valeur de distance signée tronquée (*Truncated Signed Distance Field* TSDF), estimée durant l'acquisition comme proposée par [53, 58]. Elle peut aussi être calculée directement à partir de la grille d'occupation en utilisant l'algorithme proposé par [59]. L'inconvénient de construire un champ de distance euclidien est qu'il nécessite d'être mis à jour régulièrement pour prendre en compte les changements dans la carte d'occupation lors de la navigation. Malgré cet inconvénient, cette méthode reste efficace et est souvent adoptée pour re-planifier efficacement des trajectoires [60, 61]. Ceci permet une navigation plus dynamique et plus réactive. La Figure 1.4 propose une illustration des données d'une carte d'occupation nécessaire pour un système d'exploration autonome.

Finalement, le choix de la structure –octree, grille régulière ou grille éparse- est important pour la navigation autonome. Pour choisir la structure adéquate, il faut considérer certains critères tel que :

— la vitesse d'insertion des données,

— l'efficacité pour déterminer l'obstacle le plus proche à partir d'une position donnée,

— la résolution minimale souhaitée (pour définir les voxels),

— la taille et la nature de l'espace de travail (très large, connu, inconnu, complexe).

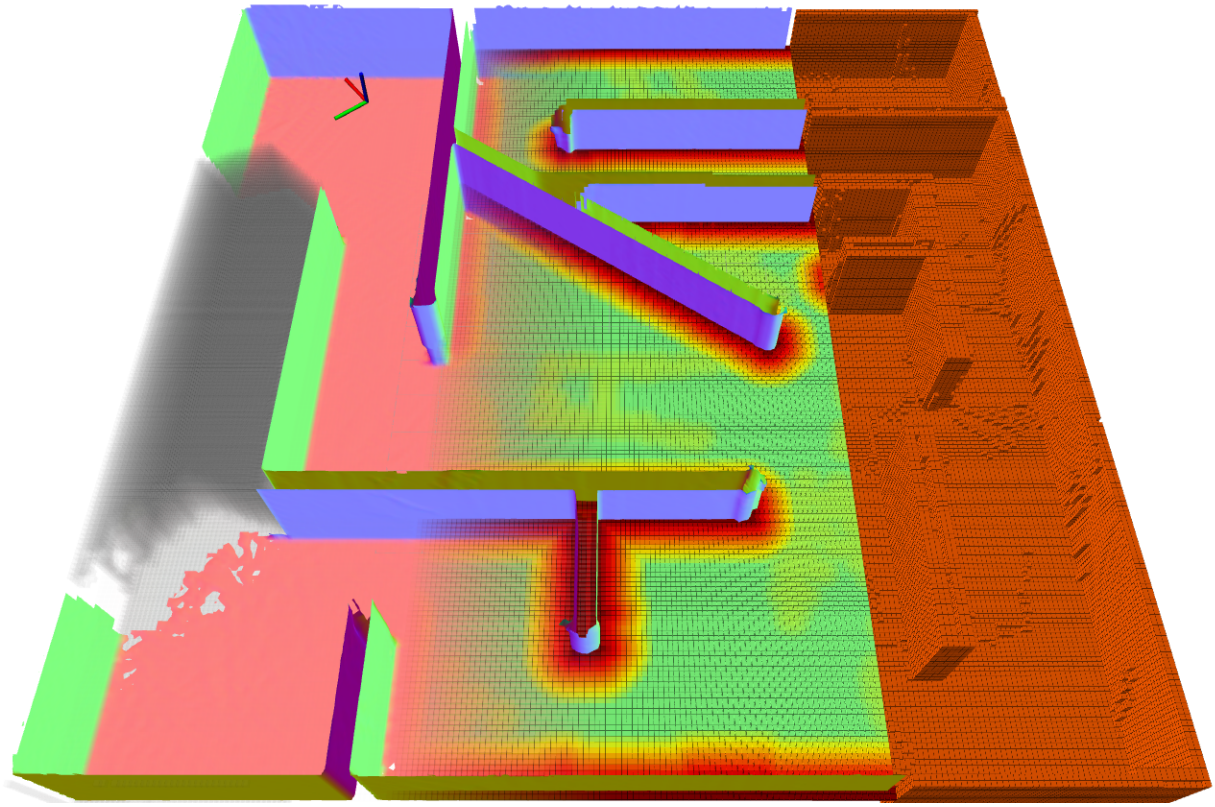


FIGURE 1.4 – Illustration des informations utilisées dans le système de carte d'occupation [62] sur l'environnement MAZE. Le trièdre correspond à la position du capteur. Les voxels gris correspondent à l'espace non exploré. Au milieu, une tranche des voxels colorés en fonction de la distance aux voxels occupés le plus proche (colorimétrie : rouge 0 (m) vert 2 (m) et plus). À droite en orange une partie des voxels considérés occupés. Enfin, le maillage, calculé à l'aide de la TSDF, est coloré en fonction de la normale des triangles (vert, rouge, violet). Pour faciliter la visualisation, nous affichons chaque information sur une sous-partie différente de la carte d'occupation. Les informations existent pour l'ensemble du volume couvert par la carte d'occupations.

Ce choix peut dépendre aussi de la tâche à résoudre et des ressources de calculs embarquées sur le robot.

1.3.2 Agrégation probabiliste

Un robot obtient des informations sur son environnement à partir de mesures qui proviennent de capteurs restituant habituellement des informations de profondeur ou de distance. Ces informations acquises puis agrégées dans des cellules d'occupation, sont sujettes à deux types d'erreurs :

1. Les erreurs de construction, de nature aléatoire ou systématique, peuvent provenir des capteurs et du système de localisation.
   - Les erreurs aléatoires sont corrigées à l'aide de modèle probabiliste que nous décrivons dans cette sous-section.
   - Les erreurs systématiques, qui peuvent résulter d'un mauvais étalonnage des capteurs et du système de localisation, ont généralement un impact minimal sur la qualité de la carte d'occupation construite, à condition qu'ils soient correctement étalonnés.
2. Les mesures aberrantes, erreurs de mesure potentiellement liées à l'environnement observé, par exemple, un élément réfléchissant va fausser la mesure de distance. Ce type d'erreur provoque l'agrégation de données erronées dans l'environnement. Il n'est pas traité par les modèles probabilistes utilisés dans les méthodes de cartographie d'occupation. Cependant, l'impact de ces erreurs est partiellement atténué par la nature incrémentale de mise à jour d'une carte d'occupation et le fait qu'un voxel peut être observé de différents points de vue. Les mesures aberrantes n'affectent pas en général la navigation, du moment que les éléments susceptibles de les engendrer sont peu représentés.

Pour tenir compte des erreurs aléatoires, chaque acquisition permet de mettre à jour une valeur de probabilité pour tout voxel situé entre le capteur et la surface mesurée. Ceci améliore la confiance dans l'état (e.g. état d'occupation, distance à la surface la plus proche) associé à chaque voxel observé.

Pour tenir compte de ces erreurs, et mettre à jour l'état d'occupation et la valeur de probabilité des voxels observés, il existe deux stratégies :

- **Lancer de rayon :** Tous les voxels entre le capteur et les mesures de surfaces sont traversés itérativement.
- **Reprojection :** Le centre géométrique de chaque voxel observé appartenant au champ de vision du capteur est reprojeté sur le plan image du capteur. Les mesures, si elles existent, sont ensuite associées aux voxels correspondants.

Le **lancer de rayon** permet une estimation plus précise de l'état d'un voxel traversé (plusieurs rayons peuvent passer par un seul et même voxel à chaque acquisition) tandis que la **reprojection** permet de fusionner les informations beaucoup plus rapidement (une seule mise à jour par voxel à chaque acquisition). La complexité du **lancer de rayon** est fonction de la résolution du capteur (nombre de mesures par acquisition), de la résolution de la carte d'occupation et de la distance de visibilité, alors que la **reprojection** est fonction du nombre de voxels dans le champ de vision du capteur qui dépend seulement de la résolution de la carte d'occupation. C'est pourquoi l'opération de reprojection est plus rapide à effectuer.

L'objectif principal adopté dans la littérature pour construire une carte d'occupation, qui représente un modèle fiable de l'environnement, est d'estimer de manière probabiliste l'espace dans lequel évolue le robot. En d'autres termes, soit $\mathcal{V} = \{v_i\}_{i=1...N}$ un ensemble de voxels représentant une carte d'occupation, soit t le temps courant comptabilisé en

période d'échantillonage, c'est la probabilité d'obtenir une certaine configuration d'occupation $p(\mathcal{V}|z_{1:t}, T_{1:t})$ pour chaque configuration possible de $\mathcal{V}$ compte tenu des mesures $z_{1:t}$ et des poses du robot (ou capteur) $T_{1:t}$. Il est d'usage de simplifier le problème en supposant l'indépendance entre les voxels de $\mathcal{V}$ [63]. De ce fait le problème devient :

$$p(\mathcal{V}|z_{1:t}, T_{1:t}) = \prod_{v_i \in \mathcal{V}} p(v_i|z_{1:t}, T_{1:t}), \tag{1.6}$$

où $p(v_i|z_{1:t}, T_{1:t})$ est la valeur de probabilité d'occupation d'un voxel v_i compte tenu des mesures $z_{1:t}$ et des poses du capteur associé $T_{1:t}$. L'état d'occupation de chaque voxel est déterminé à l'aide d'un seuil de probabilité arbitraire. Une explication complète et approfondie de la méthode est présentée dans [64, Chapitre 9]. Cette modélisation probabiliste est adaptée pour un Octree par la bibliothèque OctoMap [51], bibliothèque très populaire en robotique. Par ailleurs, un modèle probabiliste légèrement différent tenant compte de la géométrie peut être utilisé [65], permettant de reconstruire les surfaces de l'environnement. Ce modèle est adapté dans un système de cartographie d'occupation basé Octree [57] semblable à celui de [51].

Une alternative couramment utilisée en vision, et en informatique graphique [54, 55], s'appuie sur les travaux de Curless et al. [66]. Cette méthode est basée sur l'estimation d'un champ de distance signé tronqué (TSDF) dans une structure de données volumétriques (grille de voxel). Ceci permet une représentation implicite des surfaces et des régions potentiellement occupées. La valeur de TSDF d'un voxel est estimée incrémentalement de la manière suivante :

$$\begin{aligned} D_{k+1}(v_i) &= \frac{W_k(v_i)D_k(v_i) + w_{k+1}(v_i)d_{k+1}(v_i)}{W_k(v_i) + w_{k+1}(v_i)} \\ W_{k+1}(v_i) &= W_k(v_i) + w_{k+1}(v_i) \end{aligned} \tag{1.7}$$

où d_k et w_k sont respectivement une distance signée et la fonction de pondération du voxel v_i pour la $k_{ième}$ mesure. $D_k(v_i)$ et $W_k(v_i)$ sont l'accumulation des distances signées et des fonctions de pondération après intégration de la $k_{ième}$ mesure. $D_{k+1}(v_i)$ est négative quand le voxel se situe derrière une surface et positive sinon. Cette représentation implicite permet la reconstruction des surfaces observées. L'algorithme dit du "Marching Cubes" [67] permet en particulier de reconstruire les surfaces observées lors des acquisitions en suivant l'interface où le champ de distance construit par la TSDF s'annule. L'usage d'un seuil permet de décider si un voxel est libre, occupé ou inconnu à une valeur de risque fixée. Pour rappel, elle est également utilisée dans les bibliothèques Voxblox [53] et encore FIESTA [58], qui sont des systèmes complets de cartographie d'occupation très populaires pour la navigation autonome.

1.4 Planification de trajectoires

Grâce à sa carte d'occupation le robot dispose d'informations partielles ou totales sur les obstacles peuplant son environnement, les régions libres et les régions inconnues. Ces informations sont utilisées pour planifier et construire de futures trajectoires sans risques de collision. Il est courant de décomposer la construction de trajectoire en deux processus successifs :

— **Construction d'un chemin discret :** dans un premier temps la construction d'une trajectoire nécessite de trouver un chemin optimal et sans collision entre la position courante du robot et la position souhaitée aussi appelée position cible. Cette construction s'appuie très souvent sur la notion de graphes et minimise une fonction de coût telle que la distance totale à parcourir de la position de départ à la position d'arrivée. Le graphe peut être construit à partir des voxels de la carte d'occupation ou bien en s'appuyant sur des méthodes de planification par échantillonnage.

— **Trajectoire paramétrique :** dans un second temps le chemin discret est converti en une trajectoire paramétrique. Dans ce cas, la trajectoire paramétrique résultante est une fonction du temps ($t = 0$ le début de la trajectoire et $t = t_{max}$ la fin de la trajectoire). À chaque pas de temps t, le planificateur restitue une information de positionnement de vitesse et d'accélération sur tous les degrés de déplacement que le robot peut effectuer. La trajectoire paramétrique résultante sert de référentiel à suivre et doit être adaptée pour pouvoir être transformée en commande moteur par le système de contrôle du drone quadrirotor.

Il est important de noter qu'un drone quadrirotor a un espace de configuration composé de quatre degrés de liberté : trois degrés en translation (x, y, z) et un en rotation (le lacet ou en anglais le "yaw"). Puisque les collisions ne peuvent survenir que sur les axes de translation (x, y, z dans le repère monde) et compte tenu des propriétés avantageuses du drone quadrirotor, il est possible de se restreindre qu'aux trois composantes de translation pour établir un chemin sans collision. Le lacet est considéré dans la conversion du chemin en trajectoire paramétrique. Le lacet permet notamment d'orienter les capteurs du drone dans la direction souhaitée. D'autres grandeurs telles que la vitesse, l'accélération, ou bien l'a-coup sont aussi prises en compte lors de la conversion pour éviter de transmettre des ordres de déplacements infaisables au système de contrôle.

1.4.1 Recherche de chemin

Les méthodes de recherche de chemin dans une carte d'occupation peuvent être regroupées en deux catégories, à savoir : (i) les méthodes déterministes et (ii) les méthodes stochastiques. Étant donné une position de départ et une position cible et considérant qu'une solution existe, un algorithme de recherche de chemin déterministe produira toujours le même résultat et proposera toujours le même chemin. Tandis que les méthodes de recherche de chemin dite stochastique ont une notion plus faible de déterminisme, la qualité du résultat et du chemin produit dépend du nombre d'itérations et/ou du temps

alloué pour la recherche. Certaines méthodes stochastiques peuvent devenir déterministes en considérant un nombre d'itérations ou un temps de recherche suffisant voir infini alors que d'autres ne le sont pas.

Que ce soit les méthodes déterministes ou les méthodes stochastiques, la technique utilisée pour réaliser une recherche de chemin utilise la notion de graphe. Chaque arête du graphe constitue un chemin potentiel. Pour le cas des méthodes déterministes, le graphe se construit sur la carte d'occupation, chaque voxel *'libre'* est considéré comme un nœud et chaque nœud adjacent est relié par une arête. Parmi les méthodes de recherche de chemin déterministe, on retrouve l'algorithme de Djikstra [68] qui a notamment été utilisé pour de l'exploration autonome par drone quadrirotor [69]. L'algorithme de Djikstra a la particularité de produire un chemin optimal si une solution existe et le temps de recherche le permet. Des variantes de l'algorithme de Djikstra considèrent l'utilisation d'une heuristique minorante et monotone telle que : l'algorithme A^* [70] dont l'optimalité n'est garantie que sous certaines conditions, mais produit tout de même des résultats très compétitifs. En pratique, l'algorithme A^* est un bon compromis entre efficacité et optimalité de la solution. Nous pouvons également citer l'algorithme D^* [71, 72], la variante dynamique de A^*, dont le fonctionnement est le même que l'algorithme A^* mais avec la particularité de pouvoir re-planifier dynamiquement le chemin si nécessaire (e.g. provoqué par un nouvel obstacle inséré dans la carte d'occupation) sans pour autant relancer une recherche complète.

Étant donné que les méthodes déterministes considèrent la carte d'occupation comme un graphe dans son ensemble, leurs performances dépendent directement de la dimension de l'espace de configuration (e.g. 3D pour le drone quadrirotor) et de la résolution de la carte d'occupation (i.e. taille du graphe). En robotique, quand l'espace de configurations et/ou le graphe de possibilités est trop complexe, les stratégies de recherche de chemins passent par un échantillonnage *aléatoire* de l'espace de travail. Ces approches stochastiques permettent de respecter les contraintes d'efficacité mais peuvent produire des solutions sous-optimales et ne garantissent pas qu'une solution satisfaisante soit trouvée même si elle existe. Dans la littérature, l'échantillonnage suit diverses méthodologies guidées par deux éléments : (i) la construction probabiliste ou aléatoire des nouveaux états (échantillons) faisant office de nœud et (ii) la stratégie pour relier ces nœuds (avec des segments valides). Barraquand et al. [73] pionniers dans ce type d'approche proposent une procédure de type Monte-Carlo appliquée sur une carte de champ potentiel pour construire des trajectoires sans collision. Une carte de champ de potentiel est attractive vers l'objectif et répulsive autour des obstacles. Depuis ces travaux, de nombreux auteurs ont proposé des approches gravitant autour de ce concept. Une bibliothèque open-source a été développée pour l'analyse et la planification de chemin basée échantillonnage [74], couramment utilisée en robotique mobile. Les algorithmes de planification de chemin relevant de la technique d'échantillonnage les plus populaires à ce jour sont : les cartes de routes probabilistes (*probabilistic roadmap*, PRM) [75, 76] et les arbres aléatoires à exploration rapides (*Rapidly exploring Random Tree*, RRT) [77].

Les méthodes de types PRM comportent deux phases : (i) **La phase d'apprentissage** dans laquelle un graphe est construit itérativement par approximation des chemins faisables par le robot dans l'espace de travail. (ii) **La phase de requête** qui consiste à trouver le chemin

entre la position de départ et la position d'arrivée en utilisant des algorithmes de recherche de plus court chemin dans un graphe tel que vu précédemment dans ce chapitre.

Les méthodes de type RRT établissent une trajectoire possible entre deux positions en construisant une structure arborescente. Chaque feuille est étendue itérativement dans l'espace d'exploration. La solution feuille qui respecte au mieux les critères imposés (e.g. distance minimum à l'objectif) correspond au résultat. La trajectoire retenue est alors le chemin de la feuille à la racine de l'arbre.

Initialement, pour chacune des deux méthodes, c'est un échantillonnage uniforme aléatoire [75-77] qui était utilisé. Maintenant, il existe des stratégies d'échantillonnage qui améliorent les résultats dans certaines situations. Karaman et al. [78] proposent en particulier deux nouvelles variantes : PRM^* et RRT^* qui garantissent la convergence et l'optimalité de la solution si elle existe et si le temps de recherche est suffisant. Plus particulièrement, il existe des variantes du RRT^* qui accélèrent la vitesse de convergence, comme $RRT\#$ [79] ou bien d'autre s'inspirant de l'utilisation d'une heuristique minorante comme c'est le cas dans les algorithmes $A*$ et $D*$ tel que $Informed - RRT^*$ [80]. Elbanhawi et al. [81] examinent l'ensemble des méthodes d'échantillonnages en portant une attention particulière aux méthodes PRM et RRT.

Une représentation schématique des méthodes Djikstra, A^* et RRT est proposée dans la Figure 1.5. Les trois méthodes illustrées dans la Figure 1.5 s'avèrent être les plus représentées dans la littérature pour les applications de navigation de drones quadrirotors.

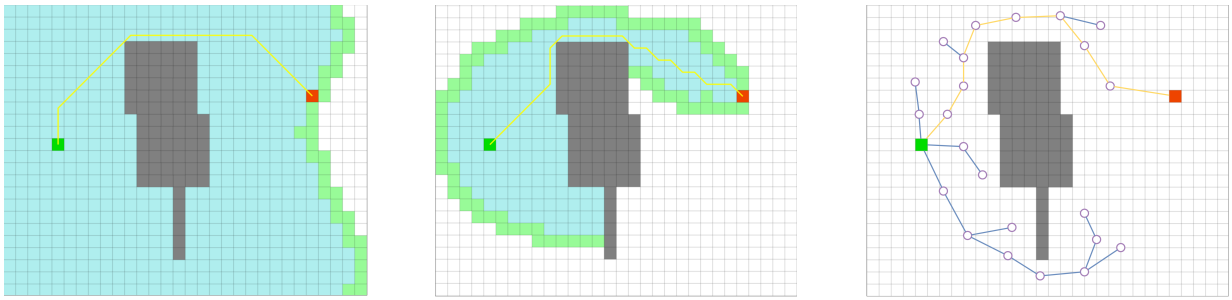


FIGURE 1.5 – Illustration de trois stratégies de recherche de chemins. L'algorithme de Djikstra, A^* et RRT. En cyan les voxels visités par l'algorithme, et en vert, les futurs voxels à visiter. En jaune le chemin à parcourir. Les nœuds de l'arbre sont représentés par les cercles violets, les arêtes (chemin valide) par les segments bleus. Pour l'algorithme de Djikstra et A^* nous pouvons clairement observer une différence en nombre d'itérations et de voxels traités.

1.4.2 Trajectoires dynamiques

Dans leur forme initiale, les algorithmes de recherche de chemin sont des méthodes discrètes qui restituent une séquence de positions (points de contrôle) formant un ensemble

de segments, le long desquels on souhaiterait que le robot se déplace. Sous cette forme, il est difficile de transmettre des ordres permettant un déplacement fluide d'un drone quadrirotor. Le chemin discret construit ne permet pas de créer une trajectoire sans à-coups (changement brusque d'accélération), continue au cours du temps (paramétrique) et dynamiquement réalisable par le système de contrôle du robot. Pour s'affranchir de cette insuffisance, les méthodes proposées dans la littérature s'appuient très souvent sur une seconde étape dont l'objectif est de construire une trajectoire paramétrique suivant au mieux le chemin préétabli. En robotique mobile, ce problème est traité à l'aide de courbes polynomiales ou courbes polynomiales paramétriques, e.g. des courbes de Bézier ou B-Spline. La trajectoire paramétrique construite devient dans ce cas de figure une fonction du temps.

L'intuition derrière ce processus est de considérer chaque segment de la trajectoire (en 3D) comme par trois fonctions polynomiales (une par coordonnée) indépendantes et paramétrées dans le temps :

$$t \in [0..t_f] \qquad s_i(t) \in [s_x(t), s_y(t), s_z(t)]^T \qquad s_i(t) = \sum_{n=0}^{\infty} \alpha_n t^n \tag{1.8}$$

où t_f est le temps total pour exécuter la trajectoire, n le degré et α les coefficients du polynôme. $s_i(t)$ est la position du robot au temps t pour l'axe i.

Cette représentation permet d'obtenir la position, mais, si le degré du polynôme est assez élevé, la vitesse $\dot{s}_i(t)$ et l'accélération $\ddot{s}_i(t)$ en particulier. Ce type de polynôme est construit en choisissant les contraintes de positionnement et de déplacement au début et à la fin du segment ainsi qu'une valeur de t_{max} pour que les contraintes de vitesse, d'accélération ou autre dérivée soient respectées au cours du déplacement. Séparer le problème de déplacement en recherche de chemin purement géométrique et puis calculer une trajectoire paramétrique est probablement la stratégie la plus populaire notamment pour l'exploration autonome par drone quadrirotor. Cependant, obtenir une trajectoire paramétrique optimale en termes de coût de contrôle pour le robot et de temps d'exécution de celle-ci conduit à un problème complexe d'optimisation non-linéaire. Les méthodes disponibles actuellement proposent des solutions presque optimales, réalisant un compromis entre temps de calculs, coût de contrôle (changement brusque d'accélération) et temps total d'exécution de la trajectoire. Par exemple, Richter et al. [82] comme Burri et al. [83] construisent une trajectoire paramétrique en définissant chaque segment du chemin calculé par l'algorithme de recherche discret (e.g. A^*, RRT) comme un ensemble de polynômes (quatre pour les quadrirotors trois pour le déplacement et un pour le lacet). Enfin, ils proposent une solution d'optimisation quadratique pour minimiser le coût de contrôle ainsi qu'une stratégie pour affiner le temps d'exécution de la trajectoire, tout en s'assurant que la trajectoire obtenue reste sans collisions. D'autres, comme Liu et al. [84], utilisent une variante de l'algorithme A^* (*Jump Point Search* [85]), se servent de polygones convexes pour représenter l'espace navigable autour de la trajectoire et enfin raffinent et modifient cette trajectoire avec la contrainte qu'elle doit rester dans l'espace navigable. Tordesillas et al. [86] proposent une

approche similaire en s'appuyant sur les courbes de Bézier cubiques.

Par ailleurs, il est possible d'adapter les méthodes de recherches discrètes pour qu'elles deviennent paramétriques et prennent directement en compte la dynamique du robot tout en optimisant la durée de la trajectoire et la distance parcourue [84, 87-89]. Cependant, ces méthodes seules sont souvent sous optimales ou ne respectent que difficilement les contraintes temps réel. Elles sont parfois utilisées pour établir une première estimation de la trajectoire paramétrique, la solution obtenue étant ensuite de nouveau optimisée pour obtenir une trajectoire paramétrique de meilleure qualité [61].

L'usage de trajectoires paramétriques offre la possibilité de re-planifier efficacement les déplacements du robot pour les rendre plus sûrs, parfois optimaux, et ainsi permettre une certaine réactivité du système. De nombreuses méthodes émergent autour de cette problématique (notamment pour le quadrirotor) [60, 61, 90-92]. La re-planification de trajectoires s'appuie très souvent sur un processus d'optimisation non linéaire local qui cherche à minimiser plusieurs critères avec le moins de modifications possible sur la structure de la trajectoire. Deux critères sont communément optimisés : l'a coup (réduisant l'effort du robot pour suivre la trajectoire) et la distance aux obstacles (rendant la trajectoire plus sûre). Habituellement, le processus est optimisé à l'aide de méthode quasi-Newton e.g. BFGS [93]. La re-planification permet aussi de préserver l'intégrité du robot en cas d'un changement de trajectoire imprévu (évitement d'obstacles dynamique, changement d'objectifs).

Exploiter le potentiel complet d'un drone quadrirotor et concevoir des trajectoires optimales est toujours un problème ouvert, il est très difficile de trouver un compromis entre faisabilité, risque et temps minimal. Une trajectoire paramétrique est dite optimale si le temps t_{max} est minimal et si elle reste réalisable par le drone quadrirotor. Dans l'objectif d'exploiter le potentiel complet du drone quadrirotor, Foehn et al. [94] proposent une méthode d'optimisation complexe qui calcule une trajectoire paramétrique très proche de l'optimalité et qui surpasse les performances de pilotes (humain) expert. Cependant, leur méthode d'optimisation n'est pas temps réel : le calcul et l'optimisation d'une trajectoire nécessite une durée allant de quelques minutes jusqu'à plus de 65 minutes dans le cas des scénarios que les auteurs proposent. De plus, une trajectoire optimale nécessite souvent que les moteurs soient utilisés à leur plein potentiel, ce qui rend la moindre perturbation extérieure dangereuse. La dernière tendance utilise l'apprentissage par renforcement pour concevoir des trajectoires qui respectent les contraintes de calcul temps-réel tout en s'approchant d'une trajectoire optimale [95, 96].

L'utilisation de trajectoires paramétriques est facilitée par la propriété de platitude différentielle [97] du drone quadrirotor. La trajectoire paramétrique est transmise au système de contrôle du drone quadrirotor. Nous abordons brièvement, dans la section suivante, la modélisation dynamique et le contrôle d'un quadrirotor.

1.5 Contrôle d'un drone multirotors

Les trajectoires paramétriques calculées précédemment servent de référentiel à suivre pour une architecture de contrôle qui convertit les informations de positionnement, vitesse,

accélération, etc, en commande pour les moteurs. Cette section fournit au lecteur les rudiments théoriques et pratiques du modèle dynamique d'un drone quadrirotor et présente les principales méthodes de contrôle issues de la littérature.

1.5.1 Quadrirotors

Le modèle dynamique d'un drone quadrirotor a reçu une attention particulière au cours de cette dernière décennie [97]. Des simulateurs ont été développés pour simuler de manière quasi-réaliste la physique et la dynamique de ce type de robot [98, 99]. C'est pourquoi ce drone est devenu un outil relativement fiable et touche aujourd'hui le grand public (e.g. DJI, Skydio, Parrot...). Le drone quadrirotor est le modèle de drone volant standard bien adopté par la communauté scientifique et industrielle. La Figure 1.6 schématise un drone quadrirotor, les repères généralement utilisés pour le contrôle et les forces qui lui sont appliquées. Ce type de drone fonctionne avec quatre moteurs (d'où son nom). Les quatre moteurs génèrent des forces et des moments par la rotation d'hélices. Ces forces permettent le vol stationnaire ou le déplacement du drone quadrirotor. Les vitesses des moteurs sont ajustées en vol à l'aide d'une loi de contrôle. La loi de contrôle se sert d'une position ou d'une trajectoire de référence ainsi que des capteurs embarqués (e.g. IMU, caméra...) pour ajuster la vitesse de rotation transmise aux moteurs.

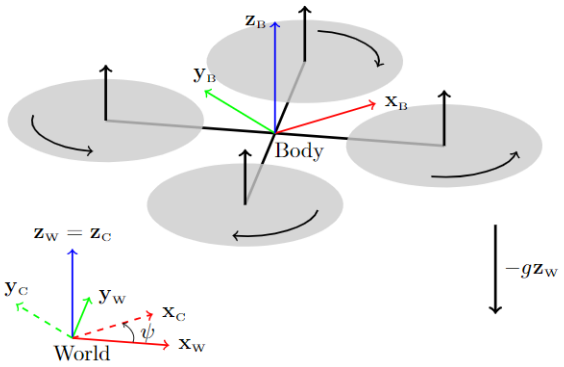


FIGURE 1.6 – Représentation schématique extrait de [100] modélisant un drone quadrirotor. La littérature utilise très souvent deux repères. Le repère **World** est décrit par la base orthonormée $\{X_w, Y_w, Y_w\}$. Son origine est variable (e.g. la position à laquelle le système de localisation est initialisé). Le repère **Body** est situé au niveau du centre de masse du quadrirotor, point pivot où les forces et les moments des moteurs s'appliquent. Le repère représenté par la base orthonormé $\{X_c, Y_c, Y_c\}$ est utilisé par les modèles de contrôle qui s'appuie sur la propriété de platitude différentielle. $-gZ_w$ est la force de gravité.

1.5.2 Architecture de contrôle

L'architecture de contrôle la plus couramment utilisée dans les drones quadrirotors s'appuie sur deux boucles de contrôle. L'architecture se compose d'un contrôleur en position et d'un contrôleur en attitude (tangage, roulis, lacet, poussée). La boucle de contrôle de position peut typiquement mettre en œuvre une loi de contrôle proportionnelle dérivée (PD) sur la position, la vitesse et l'accélération. La boucle PD exploite la dynamique du quadrirotor pour compenser la gravité et générer des accélérations lui permettant de suivre une trajectoire de référence. La sortie de cette boucle de contrôle est une accélération cible (force) qui encode la poussée de l'ensemble des moteurs et l'attitude désirée (tangage, roulis). On doit noter que le cap (ou lacet) qui correspond la rotation autour de l'axe vertical dans le repère monde (c.f. Z_W Figure 1.6) peut être choisi indépendamment. De même que les propriétés physiques des drones quadrirotors contraignent généralement le contrôle haut niveau (en position) sur 4 degrés de liberté seulement (et non 6 comme on pourrait le penser) à savoir position (i.e translation) et lacet. D'autres grandeurs peuvent être ajoutées pour chaque composante de positionnement telles que la vitesse et l'accélération. Les informations de tangage, de roulis et de lacet calculées sont utilisées comme entrées pour le contrôleur d'attitude. Celui-ci utilise habituellement une loi de contrôle PD pour restituer le couple aérodynamique à appliquer sur le quadrirotor. Le modèle dynamique du quadrirotor permet de convertir le couple aérodynamique et la poussée collective en vitesse angulaire pour les moteurs. Cette architecture est couramment nommée contrôleur en cascade car elle se compose de deux boucles de contrôle PD : une dite "haut niveau" (contrôle position) et une dite "bas niveau" (contrôle d'attitude). La Figure 1.7 schématise ce type d'architecture.

Lee et al. [101, 102] proposent un contrôleur en cascade non linéaire puis s'appuient sur des stratégies de contrôle géométriques. Ce type de contrôleur est utilisé dans des simulateurs réalistes pour drones quadrirotors tel que RotorS [98]. Une preuve établie par Mellinger et al. [97] suggère que les drones quadrirotors respectent la propriété de platitude différentielle. Cette propriété fournit une approche unifiée simple pour concevoir un contrôleur capable de suivre une trajectoire [103]. Ces travaux ont présenté un algorithme de contrôle qui calcule une poussée collective ciblée et des entrées de couple à partir des erreurs mesurées en position, en vitesse, en orientation et en gravitation. Avec cette méthode, des manœuvres agiles avec des vitesses de plusieurs mètres par seconde ont été réalisées. Enfin Faessler et al. [100] complètent ces travaux en ajoutant la force de trainée engendrée par les vibrations des hélices. Ils soutiennent que le drone quadrirotor respecte la propriété de platitude différentielle même en considérant la force de trainée. Considérer cette force permet un suivi plus précis de la trajectoire de référence.

La littérature propose d'autres méthodes qui s'appuient sur les modèles de commande prédictive MPC (*Model Predictive Control*) [104], qui est une alternative aux méthodes basées PD. Un MPC exploite le modèle dynamique du robot afin d'anticiper son futur comportement. Le MPC génère des commandes de contrôle sur un horizon temporel fini qui minimisent l'erreur de suivi dans ce même horizon en résolvant des problèmes d'optimisation sous contraintes. Les méthodes basées MPC ont la capacité de gérer des contraintes

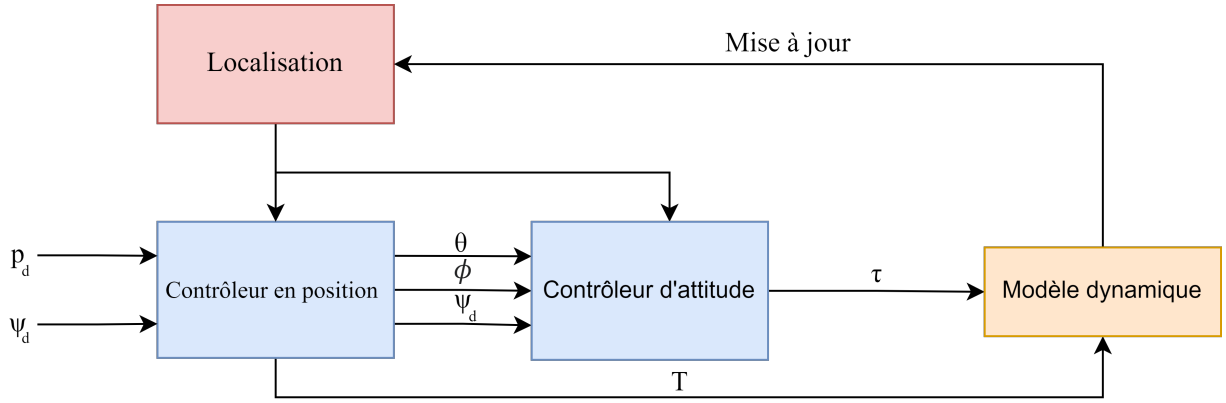


FIGURE 1.7 – Schéma d'une architecture de contrôle en cascade pour un drone multirotors. La position (x, y, z) désirée est dénoté par p_d et ψ_d correspond à l'angle lacet désiré. D'autres grandeurs telles que la vitesse et l'accélération pour chacune des composantes peuvent être ajoutées. θ et ϕ correspondent respectivement aux tangage et roulis cible calculée. Enfin τ correspond au torque aérodynamique calculée du quadrirotor et T à la poussée collective calculée par le contrôleur en position

d'entrées (ce qui évite des commandes irréalisables par le robot) et génèrent des commandes plus douces que les méthodes traditionnelles. Dans l'architecture de contrôle vue précédemment, les modèles prédictifs remplacent habituellement le contrôleur haut niveau (contrôleur en position) [105-108]. Il existe deux types de modèles prédictifs :

— **Un Modèle prédictif de contrôle linéaire (LMPC)** considère un modèle dynamique linéarisé.

— **Un Modèle prédictif de contrôle non-linéaire (NMPC)** peut utiliser un modèle dynamique non linéaire, des paramètres qui varient dans le temps, ainsi que des contraintes non linéaires. Ceci permet parfois de décrire un peu mieux le comportement d'un drone quadrirotor [108].

Les méthodes basées NMPC offrent de meilleurs résultats dans le suivi de trajectoire et le maintien de position en cas de perturbation extérieure que le LMPC [109]. Une étude récente portant sur les MPC pour les drones multirotors est présenté par Nguyen et al. [110] et obtient les mêmes conclusions. Enfin, Sun et al. [111] étudient et comparent le NMPC et les travaux de Faessler et al. [100] Differential-Flattness-Based Contrôler (DFBC - modèle de contrôle basé PD) concluant que le NMPC n'est pas supérieur au DFBC si les commandes/trajectoires demandées sont faisables. Par ailleurs, l'avantage du DFBC est qu'il est relativement moins exigeant en puissance de calcul que le NMPC. Le NMPC, quant à lui excelle quand il s'agit de pousser le drone quadrirotor aux limites de ces capacités de vol en termes de vitesse de déplacement. D'autres études et méthodes existent concernant le contrôle du drone quadrirotor. Notamment, certaines concernent les pannes de moteur [110, 112, 113], ou bien l'interaction avec des objets physiques [114] ou encore le transport

de charges [115].

Dans cette section, nous venons de présenter brièvement le modèle dynamique du drone quadrirotor. Nous avons vu l'architecture de contrôle couramment adoptée par les méthodes proposées dans la littérature, à savoir, l'architecture de contrôle en cascade. Pour finir, nous avons présenté un état de l'art portant sur deux stratégies de contrôle (PD et MPC) largement utilisées pour le contrôle d'un drone quadrirotor. Ces deux stratégies de contrôle sont suffisamment fiables pour résoudre des tâches tels que l'exploration autonome et le vol à haute vitesse.

1.6 Exploration autonome

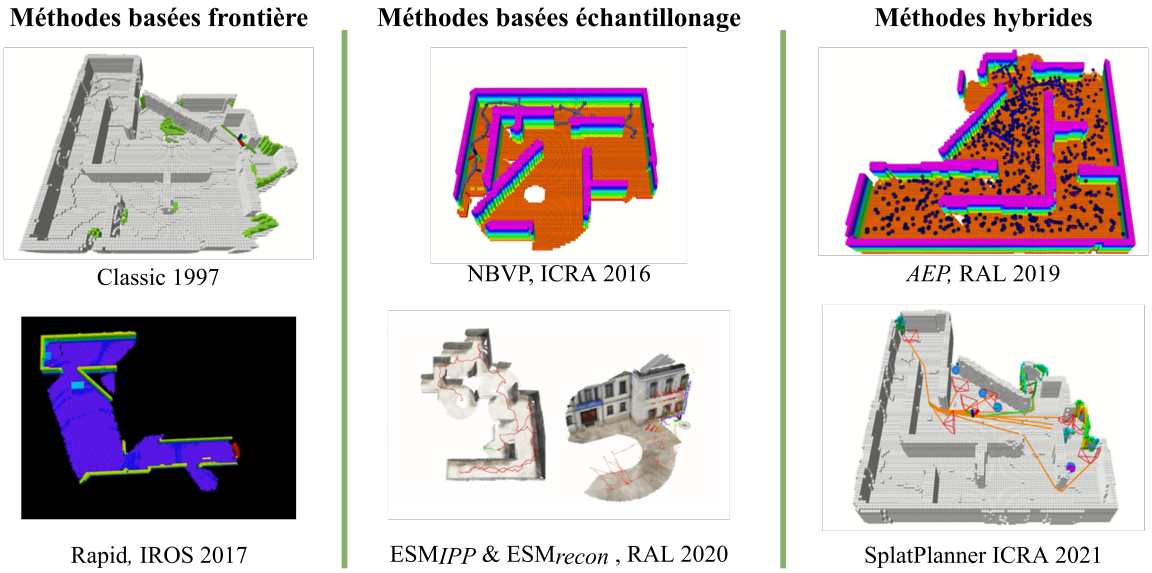


FIGURE 1.8 – Six méthodes état de l'art : (i) **Méthodes basées frontière :** Classic [2] Rapid [69] (ii) **Méthodes basées échantillonnage :** NBVP [116], ESM [117] (iii) **Méthodes hybrides :** AEP [118], SplatPlanner [3].

Les sections précédentes présentent les briques théoriques fondamentales pour la conception d'un robot autonome de type drone quadrirotor. Cette section aborde l'un des objectifs principaux de ces travaux de recherche, à savoir l'exploration autonome en environnement inconnu. Bien que l'exploration autonome ait été étudiée en robotique depuis plus de 30 ans [1], ce sujet voit récemment sa popularité augmenter dans plusieurs domaines de recherche tels que la vision [119] ou l'informatique graphique [120, 121]. L'exploration autonome présente des verrous similaires à ceux de la vision par ordinateur, dont l'essor laisse entrevoir des opportunités intéressantes pour la reconstruction 3D autonome d'environnements, avec des méthodes innovantes et performantes dans un domaine de recherche compétitif.

L'exploration autonome consiste à découvrir la totalité d'un espace délimité (i.e. l'environnement) sans connaissance a priori sur celui-ci, au moyen d'un robot terrestre ou aérien

(le plus souvent un drone quadrirotor) équipé de capteurs de distance ou de profondeur (e.g. LiDAR, caméra de profondeur). Les méthodes qui assurent cette tâche construisent incrémentalement une carte de l'environnement en estimant des déplacements permettant de compléter la carte. Les algorithmes d'exploration autonome sont habituellement catégorisés (i) en méthodes basées sur les frontières, (ii) en méthodes basées sur l'échantillonnage ou (iii) en méthodes hybrides. Les méthodes basées sur les frontières explorent exhaustivement chaque espace interstitiel entre une région visitée et une région inconnue. On appelle cet espace interstitiel des frontières. Il en résulte à terme une exploration totale de l'environnement. Les méthodes basées sur l'échantillonnage évaluent et choisissent les déplacements qui optimisent la découverte de régions non observées, en échantillonnant des positions réparties dans les régions visitées, permettant ainsi des déplacements plus efficaces. Les méthodes hybrides combinent les deux stratégies précédentes.

Bien que l'objectif principal soit de découvrir efficacement l'ensemble de l'environnement, des stratégies intermédiaires permettent, par exemple, d'améliorer la qualité de la cartographie, de se concentrer sur la qualité des surfaces, d'optimiser les trajectoires, ou bien de faire collaborer plusieurs robots. Finalement, toute méthode d'exploration cherche à répondre (de près ou de loin) à la question suivante : comment guider de manière efficace un robot afin d'obtenir une cartographie de l'environnement (i) rapidement, on parle alors d'efficacité d'exploration, et (ii) satisfaisante, c'est-à-dire une cartographie (et parfois la reconstruction surfacique de l'environnement) complète et correcte ? La Figure 1.8 présente six méthodes de l'état de l'art qui ont été implémentées sur un drone quadrirotor, pour chacune des trois catégories listées dans le paragraphe précédent.

1.6.1 Méthodes basées frontière

L'un des concepts fondamentaux utilisés en exploration autonome est celui de frontière. Dans la littérature, les frontières peuvent être interprétées et généralisées comme des zones (e.g. voxel ou information topologique) qui séparent les régions précédemment visitées des régions inconnues. En 1992, Maja et al. [1] présentent une méthode d'exploration qui utilise une représentation topologique 2D pour cartographier l'environnement. Maja et al. proposent d'utiliser les segments de murs, captés par un sonar monté sur un robot, comme repère topologique. Le robot explore l'environnement de manière autonome en prolongeant les segments de murs incomplets (les frontières sont situées sur les murs incomplets). Des stratégies similaires existent utilisant des capteurs plus récents de type LiDAR [122, 123]. Thrun et al. [124] combinent deux types de cartes d'occupation, une carte topologique et une grille régulière telle que vue en section 1.3. La représentation sous forme de grilles régulières de voxels était, à cette époque, un moyen simple et précis pour reconstruire l'environnement. Néanmoins, la puissance de calcul et de stockage de l'information des robots des années 1990 pouvaient contraindre à choisir des représentations plus compactes. Plus récemment, Santosh et al. [125] utilisent la vision par ordinateur pour établir une segmentation du sol et guider le robot vers les limites du plan qui définit le sol. L'inconvénient de cette approche est qu'ils font une hypothèse forte sur la topologie de l'environnement, nécessitant que celui-ci soit forcément composé de structures physiques (murs, sol) visibles

par le capteur. Cette hypothèse restreint clairement le champ d'utilisation de ce type d'approche. Balch et al. [126] proposent une méthode réactive s'appuyant sur une fonction répulsive incitant le processus d'exploration à sortir des régions visitées. Cependant, l'exploration purement réactive ne fonctionne pas bien dans des environnements trop grands et/ou complexes. En effet, les différentes forces issues de la fonction répulsive peuvent se compenser induisant des minima locaux et provoquant la fin de l'exploration. De manière similaire, E Silva Jr et al. [127] exploitent une carte de champ de potentiel basée sur des fonctions harmoniques guidant le robot vers les frontières. Cette méthode est robuste aux minima locaux mais est très coûteuse en calcul.

Ce sont les travaux précurseurs de Yamauchi et al. [2] (méthode nommée **Classic** dans la littérature) qui, en s'appuyant sur une carte d'occupation probabiliste, proposent une stratégie générique d'exploration autonome 2D indépendante de l'environnement. Dans leurs travaux, Yamauchi et al. introduisent la notion d'état frontière pour certains voxels d'une carte d'occupation. Par le biais de ce nouveau label, le robot peut se déplacer vers la frontière la plus proche trouvée à l'aide d'un algorithme de parcours en profondeur appliqué sur la grille (similaire à l'algorithme de recherche de chemin de Djikstra). Une fois la position atteinte, le robot effectue une rotation sur lui-même pour compléter la carte d'occupation et réitère le processus tant qu'il existe des frontières accessibles. L'état de frontière est ajouté aux trois états probabilistes qu'une carte d'occupation 2D possède (*libre*, *occupé*, *inconnu*). Dans **Classic**, cet état est caractérisé de la manière suivante : tout voxel *libre* adjacent à un voxel *inconnu* est étiqueté comme un voxel frontière. Les voxels frontières sont extraits par un processus analogue à la détection des bords en traitement d'images. Une fois extrait, les voxels frontières sont regroupés en régions de taille minimale, si possible comparable à celle du robot, pour permettre son passage. Holz et al. [128] évaluent plusieurs stratégies exploitant le paradigme de **Classic** dans des environnements intérieurs et argumentent autour de quelques améliorations. Leur première proposition est d'établir une stratégie plus réactive que **Classic** et vérifier si un voxel frontière cible est toujours une frontière (i.e. n'a pas encore été observé) au cours du déplacement du robot. Si tel n'est pas le cas, l'algorithme cherche la nouvelle frontière la plus proche. Leur seconde proposition consiste à segmenter l'environnement intérieur en salles indépendantes et forcer le robot à explorer entièrement chaque même salle avant de changer, évitant ainsi de devoir repasser plusieurs fois dans la même salle.

Certaines études cherchent seulement à améliorer la qualité de la cartographie suite à une campagne d'exploration autonome. Par exemple, Sim et al. [129, 130] utilisent l'information contenue dans chaque voxel (nombre d'observations acquises) pour estimer les régions mal reconstruites et finalement repasser par celles-ci une fois l'environnement entièrement cartographié par **Classic**. Repasser par des régions déjà visitées permet d'obtenir une meilleure estimation de l'état de chaque voxel. De plus, beaucoup d'informations de localisation (points caractéristiques) sont réutilisées, améliorant par conséquent la précision du système de localisation et donc la cartographie. Cela est vrai dans le cas du SLAM Visuel. Il est également possible d'appliquer un algorithme de type fermeture de boucle éliminant aussi de potentielles dérives du système de localisation.

L'exploration autonome basée sur les frontières a été étendue au problème de coordi-

nation multiagent [131]. L'usage de plusieurs robots peut, si l'environnement le permet, accélérer le processus d'exploration tout en maintenant une qualité de localisation et de cartographie appréciable. Ceci est dû au fait qu'une plus grande quantité de données est acquise simultanément. Cependant, une difficulté survient quant à la coordination et à l'échange d'informations entre les robots.

L'ensemble des méthodes présentées jusqu'ici considère un robot autonome terrestre. L'usage d'un drone quadrirotor avec une exploration basée frontière est proposé par Bachrach et al. [132] avec mise en évidence des difficultés du passage du 2D terrestre vers le 3D aérien. Une extension vers la 3D plus sophistiquée est proposée par Shen et al. [133, 134]. Les auteurs introduisent une carte d'occupation dense pour les obstacles et éparse pour les volumes vides. L'exploration est guidée par l'évolution d'une équation différentielle stochastique qui simule l'expansion d'un système de particules vers les régions inconnues. Les particules s'étendent aux frontières des régions visitées pour guider l'exploration. Par ailleurs, Cieslewski et al. [69] proposent une extension de **Classic** sous le nom de **Rapid**. Cette extension utilise un processus d'exploration à deux états : le premier état préserve la vitesse du quadrirotor quand il existe des frontières dans son champ de vision; le second état gère les autres cas, utilisant **Classic** pour se diriger vers la frontière la plus proche trouvée avec l'algorithme de Djikstra. Cette stratégie flexible réduit le temps d'exploration fait et montre des résultats en général supérieurs à ceux détenus avec **Classic** en particulier en termes d'efficacité volumique d'exploration.

Pour conclure, **Classic** et les méthodes qui dérivent de ce type d'approche permettent une exploration exhaustive de l'environnement (en termes de régions accessibles depuis la position de départ du robot), mais ont deux inconvénients majeurs :

— L'efficacité pour trouver le chemin vers la frontière la plus proche est variable. Cette efficacité dépend de plusieurs facteurs, notamment de la complexité du graphe (résolution de la carte d'occupation), mais également de la distance pour l'atteindre. Quand il reste peu d'espace inconnu dans l'environnement, le temps de calcul augmente et inévitablement l'efficacité d'exploration décroit considérablement.

— Les solutions basées frontière ont pour la majorité comme stratégie de rejoindre naïvement les régions inconnues, i.e. seulement guidées par la notion de frontières. Une fois la frontière atteinte, le robot effectue typiquement une rotation complète pour cartographier l'environnement immédiat. L'inconvénient est que ces solutions n'évaluent pas le potentiel qu'un déplacement vers une frontière peut apporter à l'exploration. Même si cet objectif naïf définit correctement le critère d'arrêt d'une méthode d'exploration (i.e. plus aucune frontière atteignable), guider l'exploration seulement avec des frontières engendre des décisions de déplacements sous-optimaux et donc une exploration inefficace.

Ces deux inconvénients portent l'attention vers des stratégies plus sophistiquées qui évaluent le gain potentiel de chaque déplacement. Ces stratégies s'appuient habituellement sur l'échantillonnage aléatoire de point de vue (position à laquelle on souhaiterait positionner le capteur) dans l'ensemble des régions visitées et atteignables. Chaque point de vue peut être ensuite évalué pour trouver le déplacement idéal et explorer l'environnement.

1.6.2 Méthodes basées sur l'échantillonnage

Les méthodes basées sur l'échantillonnage cherchent à optimiser le positionnement du robot et de son capteur dans l'environnement via une fonction de gain ou d'utilité. Cette fonction utilise les informations déjà obtenues et contenues dans une carte d'occupation afin d'éviter des déplacements inutiles qui auraient pour conséquence peu de gain d'information. Les méthodes basées sur l'échantillonnage cherchent à résoudre le problème de *Next Best View* (NBV). Le problème de NBV a été initialement énoncé par Connolly et al. [135] et consiste à déterminer le (ou les) prochain(s) meilleure(s) point(s) de vue (NBV) dans l'objectif d'explorer ou reconstruire un objet, une simple scène ou un environnement au complet. Pour résoudre le problème de NBV, Connolly et al. proposent un algorithme qui choisit le point de vue qui observe le plus de voxels *'non-observés'* dans une structure de type Octree englobant l'objet devant être reconstruit. Une fois le nouveau point de vue exécuté, le processus est répété et ceci jusqu'à ce que tous les voxels initialement étiquetés comme *'non-observés'* soient étiquetés *'vides'* ou *'occupés'*. Bien que ces travaux se placent dans un contexte très simplifié où le capteur orbite autour de l'objet et est orienté vers le centre de la scène, Connolly et al. ont établi **l'objectif fondamental** des méthodes basées échantillonnage en exploration autonome.

Depuis ces travaux initiateurs, planifier la (ou les) prochain(s) meilleure(s) point(s) de vue est une approche couramment étudiée en vision par ordinateur, en informatique graphique et en robotique [136, 137]. Le problème peut être formalisé de la manière suivante. Soit $S = \{s_1, ..., s_n\}$ une séquence de positions et orientations potentielles du capteur. Le problème de NBV est de trouver $\hat{s} \in S$ qui maximise une certaine fonction d'utilité $\mathcal{U}$:

$$s^* = \arg\max_n \ \mathcal{U}(s_n) \tag{1.9}$$

La plupart des méthodes de NBV utilisent des images (ou données) de distance (e.g. caméra de profondeur ou LiDAR) et considèrent (principalement dans le domaine de la vision et de l'informatique graphique) la reconstruction d'un objet 3D appelée aussi scène [137]. Parmi les premières stratégies proposées dans la littérature, plusieurs travaux considèrent une approche orientée capteur, basée sur le lancer de rayon pour estimer le potentiel gain d'information d'une région à observer [138-140]. Cependant, dans la plupart de ces travaux, les auteurs font l'hypothèse forte que le capteur peut être placé facilement à la position cible sans considérer les contraintes de déplacements. Le fait de s'affranchir des petites erreurs de localisation et de ne reconstruire qu'un objet 3D central à la fois ne permet pas d'avoir un système réaliste complétement autonome résolvant la tâche de NBV. La robotique mobile est un moyen d'automatiser et de généraliser la tâche réelle de NBV pour une reconstruction voxelique ou surfacique de plus grande envergure. Les méthodes NBV sont plus communément appelées méthodes basées sur l'échantillonnage en robotique mobile. Ces méthodes reprennent les principes de base utilisés pour résoudre la tâche de NBV en vision et informatique graphique (orienter correctement le capteur, le lancer de rayon et fonction de gain), mais ajoutent l'estimation de chemin sans collision menant vers les points de vue échantillonnés ainsi que l'exécution du chemin menant au meilleur point

de vue sélectionné.

Les travaux préliminaires de NBV en exploration autonome [141] utilisent un sonar monté sur un robot. Dans ces travaux une mesure de distinction est proposée dans le but d'établir un graphe topologique qui mène le robot vers les régions contenant la mesure de distinction la plus élevée. En d'autres termes, ces travaux cherchent à maximiser une mesure de distinction, communément appelée fonction d'utilité ou bien fonction de gain, qui permet au robot de découvrir efficacement de nouvelles régions de l'environnement.

À l'aide d'un plan de l'environnement, Gonzalez et al. [142] estiment les futurs déplacements du robot en résolvant une version étendue du problème de la galerie d'art [143]. Dans ces travaux, le problème de la galerie d'art est résolu via un échantillonnage aléatoire de positions cibles (points de vue) dans les régions visitées. La solution correspond au plus petit ensemble de points de vue parmi l'ensemble des points de vue échantillonnés couvrant toutes les régions de l'environnement. En continuité avec leurs premiers travaux, Gonzalez et al. [144] proposent une méthode d'exploration autonome en 2D avec un robot mobile terrestre. Leur méthode d'exploration autonome consiste à échantillonner aléatoirement des points de vue candidats dans une région suffisamment large et proche des régions inconnues. Le point de vue couvrant le plus de régions inconnues est choisi comme position cible du robot.

Surmann et al. [145] étendent le problème d'exploration autonome basée sur l'échantillonnage pour une cartographie 3D avec un robot terrestre équipé de LiDAR. Dans leur approche, le robot se localise à l'aide de l'algorithme ICP appliqué à des nuages de points acquis par le LiDAR. Pour trouver le meilleur point de vue, leur méthode positionne aléatoirement n candidats et utilisent un algorithme glouton qui consiste à approximer, en 2D, une solution du problème de la galerie d'art.

Bien que les méthodes précédentes introduisent le concept d'exploration autonome en échantillonnant de manière aléatoire des vues candidates, l'émergence de robots tels que les drones quadrirotors requiert un processus d'exploration autonome complètement défini en 3D (cartographie, évaluation des points de vue et déplacement). Bircher et al. [116] présentent l'une des stratégies (connu sous le nom de RH-NBVP ou NBVP) principales d'exploration autonome qui prend en compte ces difficultés. NBVP s'appuie sur l'algorithme de navigation RRT [77] présenté section 1.4. La méthode NBVP tire avantage de l'extension rapide de ce type d'arbre dans l'environnement à explorer afin d'estimer des chemins sans collision dans les régions visitées menant à des points de vue observants de potentielles régions inconnues. Dans ce type d'arbre, chaque nœud est un point de vue candidat qui, une fois le volume inconnu évalué, sert à estimer récursivement le gain total d'information d'un chemin potentiel. Le chemin se calcule récursivement en partant d'une feuille jusqu'à la racine (position du quadrirotor). La fonction générale de NBV est alors étendue à une séquence de points de vue, relative au temps pour parcourir tous ces points de vue. Cela revient à diviser la fonction d'utilité $\mathcal{U}$ (1.9) d'un nœud par le temps nécessaire au robot pour l'atteindre en tenant compte de toutes contraintes dynamiques. Dans cette formulation, les points de vue candidats proches sont privilégiés si le gain d'information est suffisant. Autrement, cette formulation fait un compromis entre une exploration locale et une exploration globale de l'environnement. Pour évaluer un chemin complet, la littérature

propose plusieurs types de fonctions qui peuvent privilégier soit les chemins courts, soit les chemins longs. En voici deux typiques pour ce cas d'usage :

— **Linéaire** [146] : chaque nœud d'un chemin (défini de la racine jusqu'à une feuille) est considéré de manière égale. Le score d'un nœud est la différence entre son gain et son coût (coût pondéré par un paramètre choisi arbitrairement). Certains nœuds peuvent avoir une valeur négative si le gain d'information est nul ou plus faible que son coût. Si on considère le $i_{ème}$ nœud d'un arbre donné, $\mathcal{U}(\boldsymbol{p}_i)$ la fonction qui calcule le volume inconnu observé à la position de ce nœud, $\mathcal{C}(\boldsymbol{p}_i)$ le cout pour atteindre cette position et $parent(i)$ le nœud parent du $i_{ème}$ nœud, alors le gain linéaire se calcule récursivement de la manière suivante :

$$g_{lin}(i) = g_{lin}(parent(i)) + \mathcal{U}(\boldsymbol{p}_i) - \alpha\mathcal{C}(\boldsymbol{p}_i),$$

avec α un paramètre sélectionné empiriquement qui pondère le cout. Une valeur de α élevé privilégiera les chemins courts tandis qu'une valeur faible privilégiera les chemins longs.

— **Exponentielle** [116] : ce type de fonction est strictement positif. Bien que les nœuds situés loin de la racine aient un coût plus élevé et donc un score plus faible que ceux situés à proximité du robot, cette fonction tant à privilégier les longs chemins. En reprenant les mêmes notations définies ci-dessus pour le gain linéaire, la fonction de gain exponentiel est :

$$g_{exp}(i) = g_{exp}(parent(i)) + \mathcal{U}(\boldsymbol{p}_i) * e^{-\alpha\mathcal{C}(\boldsymbol{p}_i)}.$$

D'autres variantes existent, par exemple Schmid et al. [117] proposent plutôt une normalisation globale évitant l'usage d'un paramètre sélectionné empiriquement (α) souvent difficile à choisir correctement.

L'approche de NBVP est répétée après chaque déplacement vers le nœud de l'arbre qui mène le robot vers les meilleurs points de vue. L'arbre est ensuite détruit puis reconstruit pour les futurs déplacements. Néanmoins, malgré l'efficacité et la facilité de convergence pour explorer les régions inconnues avoisinantes de ce type d'algorithme, ce paradigme d'exploration a tendance à rester bloqué dans une exploration locale. Cette situation dépend de la proximité avec les régions inconnues existantes. Si l'extension de l'arbre n'est pas suffisante pour atteindre ces régions, la méthode **NBVP** termine son exploration prématurément.

Plusieurs extensions découlent de ce paradigme. Ces extensions ont généralement pour but d'accélérer l'exploration et d'éviter une terminaison prématurée du processus d'exploration. Witting et al. [147] proposent une amélioration de la méthode qui consiste à maintenir l'historique des chemins et les points de vue utilisés. Cet historique est maintenu sous forme de nœuds dans un graphe couvrant la majorité de l'espace visité. Le graphe permet ensuite de guider l'exploration en cas de situation localement sans issue. L'idée de construire un graphe dans l'espace visité est reprise aussi par Dang et al. [148] avec une planification à deux états : locale (à la manière de NBVP) et globale (guidé par le

graphe qui couvre l'ensemble du volume libre). Respall et al. [149] et Batinovic et al. [150] considère en plus des améliorations pour accélérer l'évaluation des points de vue, accélérant ainsi le processus d'exploration de NBVP. D'une part, Respall et al. utilisent deux cartes d'occupation, une de résolution fine pour une cartographie précise et une de résolution plus grossière pour évaluer rapidement les futurs points de vue candidats. D'autre part, Batinovic et al. utilisent un algorithme efficace pour évaluer le volume inconnu qui se trouve dans le champ de vision du capteur. Cette méthode se base sur la méthode dite de *Recursive Shadowcasting* qui est notamment utilisée dans le domaine du jeu vidéo. Toujours pour éviter une terminaison prématurée de l'exploration, Schmid et al. [117] étend continuellement l'arbre en s'inspirant et modifiant l'algorithme RRT* [79]. Dans le but de préserver l'intégrité de la structure et que la racine reste toujours au niveau de la vue courante, leur algorithme modifié réarrange et/ou supprime continuellement les nœuds de l'arbre qui enfreignent certaines règles (e.g. une branche qui traverse un obstacle). Similairement, l'usage de PRM* [79] est considéré dans [151]. La structure est maintenue au cours de l'exploration. Elle est mise à jour après chaque itération. D'autres méthodes proporsent d'acclérer le processus en tenant compte des propriétés dynamiques du quadrirotor [152]. Grâce à ce type d'approche, le robot n'est pas obligé de s'arrêter à chaque nœud de l'arbre et préserve ainsi la fluidité de son déplacement.

L'avantage des méthodes basées sur l'échantillonnage réside dans l'usage de fonctions de gain arbitraires. Bien que l'exploration autonome s'appuie généralement sur l'évaluation d'un volume inconnu via un lancer de rayon [116, 117, 149], il existe des approches plus spécifiques. Par exemple, Bourgault et al. [153] optimisent conjointement la tâche d'exploration et de localisation via une formulation basée sur la théorie de l'information de Shannon. La théorie de l'information de Shannon est une façon naturelle de quantifier la précision de localisation, la qualité et l'exhaustivité de la carte d'occupation. L'objectif est alors de choisir des déplacements qui minimisent l'incertitude de localisation et maximise le gain d'information pour une carte d'occupation donnée. Bourgault et al. formulent cet objectif comme une fonction d'utilité construite par une simple combinaison linéaire sur les deux critères (incertitude de localisation et gain d'information). Cependant, trouver un chemin idéal d'exploration parmi un grand nombre de possibilités, tout en réduisant l'incertitude du système de localisation, est relativement coûteux. Il est nécessaire de réduire la quantité d'informations utilisées pour la prédiction afin de garantir un traitement temps réel, au détriment de l'efficacité de l'exploration. Pour ce faire, Sim et al. [154] proposent d'utiliser une façon plus rigide de calculer des déplacements via une trajectoire paramétrique formant une spirale dans l'environnement que le robot doit suivre. La spirale est modifiée en changeant les paramètres de la trajectoire paramétrique, ce qui permet de faire revenir le robot à son point de départ (revisiter pour améliorer la cartographie et la localisation) ou le faire explorer des régions jusqu'alors inconnues. Les paramètres de la spirale peuvent être choisis de façon à maximiser la quantité de régions inconnues visitée et minimiser l'incertitude en utilisant le déterminant de la matrice de covariance du système de localisation. Plus récemment, Papachristos et al. [155] choisissent une approche semblable à celle de **NBVP** et proposent d'optimiser une fonction d'utilité qui minimise à la fois l'incertitude de localisation et l'entropie de la carte d'occupation.

Dans la littérature, on peut retrouver d'autres fonctions de gain telles que celle proposée par Dang et al. [156] qui combine l'exploration et la détection d'objets ou bien celle de Schmid et al. [117] qui proposent de guider l'exploration en fonction de la qualité de reconstruction surfacique couplée au gain d'information en termes de volume inconnu. Hepp et al. [119] proposent d'apprendre via un réseau neuronal convolutif qui prend en entrée une représentation compacte de la scène et une fonction d'utilité qui prédit les meilleurs points de vue. Dans le domaine de l'apprentissage Schmid et al. [157] proposent une architecture neuronale sophistiquée de type auto-encodeur variationnel convolutif [158, 159]. Cette architecture utilise en entrée une carte d'occupation 2D et un premier échantillonnage aléatoire de points de vue. L'architecture, une fois entraînée, prédit une meilleure distribution des points de vue échantillonnés dans l'environnement ainsi que le gain et le meilleur le point de vue.

Pour conclure, les méthodes basées sur l'échantillonnage simplifient l'espace d'exploration car elles positionnent de manière aléatoire des vues candidates. Une méthode basée sur l'échantillonnage conférent la possibilité de résoudre différentes tâches grâce au choix d'une fonction de gain arbitraire. Cependant, le problème des méthodes telles que NBVP est la terminaison prématurée du processus d'exploration. Ce problème provoque une exploration et une reconstruction sous-optimale. Les méthodes qui résolvent ce problème nécessitent de maintenir une structure globale (un arbre ou un graphe) qui doit être continuellement mise à jour pour préserver l'intégrité des déplacements du robot dans son environnement, ce qui rend la tâche complexe et lourde. De plus, dans ces méthodes, il est courant de limiter les distances entre les nœuds (de l'ordre de 1.5 (m) à 3 (m)) ce qui rend les déplacements du robot inefficaces. Ces inconvénients orientent certains travaux vers une approche hybride combinant les avantages des stratégies basées sur les frontières et celles des stratégies basées sur l'échantillonnage.

1.6.3 Méthodes hybrides

Les approches hybrides utilisent un raisonnement bi-critère via une composition plus ou moins sophistiquée des deux approches présentées précédemment, à savoir (i) les méthodes basées sur les frontières et (ii) les méthodes basées sur l'échantillonnage. Pour rappel, les méthodes basées sur les frontières maximisent généralement le volume final exploré en parcourant exhaustivement toutes les frontières et en choisissant séquentiellement la plus proche. L'inconvénient est que cette stratégie écarte des chemins d'exploration plus pertinents en termes de quantité d'informations collectées par exemple. À l'inverse, les méthodes basées sur l'échantillonnage estiment et évaluent des objectifs avec une fonction de gain pour choisir l'objectif qui maximise cette fonction. Par conséquent, ces méthodes choisissent des déplacements plus pertinents, mais prêtent parfois moins d'attention au volume final exploré et à la distance totale parcourue. Les méthodes hybrides quant à elles font usage : (i) d'une fonction de gain pour évaluer des points de vue et estimer les déplacements pertinents et (ii) des frontières pour guider les futurs échantillonnages de points de vue. Il en résulte généralement une exploration globale et locale plus efficace. Nous listons ci-dessous les concepts utilisés dans les méthodes basées sur les frontières

et les méthodes basées sur l'échantillonnage cohabitant dans les méthodes d'exploration hybrides :

— Le concept de frontière délimitant les régions libres des régions inconnues.
— Une façon de grouper des amas de frontières en un objectif unique pour simplifier l'échantillonnage de vues candidates.
— Une stratégie d'échantillonnage de nouveaux points de vue et nouvelles trajectoires.
— Une fonction de gain plus ou moins sophistiquée permettant d'évaluer une vue candidate et guider le choix de la NBV.

Stachniss et al. [160] fait une comparaison des méthodes d'exploration basées sur les frontières, des méthodes basées sur l'échantillonnage et des méthodes hybrides dans le cas d'utilisation d'un robot mobile terrestre équipé de sonars. De leur comparaison, il ressort que les méthodes basées sur les frontières sont plus efficaces que celles basées sur l'échantillonnage si le robot est capable d'intégrer continuellement les mesures issues du sonar durant son déplacement. De plus, les stratégies basées sur les frontières optimisent la distance parcourue, contrairement aux méthodes basées sur l'échantillonnage. Cette comparaison montre que les méthodes hybrides présentent de meilleures performances car elles font un bon compromis entre exhaustivité de la recherche grâce aux frontières et optimisation des informations collectées par des points de vue échantillonnés. Nos travaux [62] comparent ces trois approches d'exploration dans le cas spécifique d'un drone quadrirotor. Nous avons pu aboutir à des conclusions similaires dans la mesure où le robot ne se déplace quasiment qu'à l'horizontal, comme dans les travaux de Stachniss et al. [160]. Toutefois, nous avons observé dans les scènes 3D ayant une hauteur maximale élevée que les performances des méthodes basées frontière (e.g. **Classic**) se détériorent et elles deviennent beaucoup moins efficaces. À l'inverse, les deux autres stratégies ne sont pas réellement affectées par cette modification des conditions d'utilisation.

Stachniss et al. [161] proposent par la suite une solution hybride originale basée sur un filtre à particules, qui permet d'estimer les probabilités d'occupation des voxels cartographiés dans une grille régulière. Leur solution utilise une fonction de gain calculant la quantité d'informations collectées parmi trois actions de déplacement possible pour un robot terrestre. La première action est identique à **Classic** et dirige le robot vers la frontière la plus proche. La seconde action permet de réduire l'incertitude de localisation et d'améliorer la cartographie déjà existante en guidant le robot vers des régions déjà explorées. Enfin, la dernière action dirige le robot pour effectuer une fermeture de boucle et ainsi éliminer toute dérive du système de localisation. Parmi les trois actions, celle qui obtient le meilleur score d'utilité est choisie. Une exploration basée sur ces trois actions permet d'aboutir à une cartographie de qualité. Cependant, dans une optique d'exploration rapide, la qualité de la cartographie est souvent au détriment du temps d'exploration. De plus, la première action, celle qui est purement exploratoire, réutilise l'approche naïve de **Classic** (la frontière la plus proche est choisi), or nous avons vue : (i) que cette approche se dégrade dans le cas de scène 3D haute et (ii) que cette approche peut engendrer des déplacements moins optimaux qu'une approche basée sur l'échantillonnage car un seul point de vue est considéré.

Dans le but d'obtenir une exploration plus efficace les méthodes plus récentes permettent de considérer plus de points de vue. Beaucoup d'approches s'orientent vers des structures arborescentes pour la navigation et l'exploration. Typiquement, plusieurs méthodes [162-164] s'inspirent de l'algorithme SRT [165] (Sensor-based Random Tree) qui est une variante de l'algorithme RRT spécialement conçue pour une exploration tenant compte de la perception des capteurs. Initialement, les SRT sont construits de manière incrémentale par le robot en utilisant un planificateur local aléatoire. Celui-ci génère à chaque pas la configuration suivante, cette génération étant guidée par les frontières se situant dans la région navigable perçue autour de la position courante du robot. Freda et al. [162, 163] utilisent les SRT en s'appuyant sur la notion de frontière anssi qu'une fonction de gain combinant le gain d'information et l'incertitude de localisation. Les SRT ont été utilisés également pour réaliser une exploration basée drone quadrirotor [164] dans une configuration d'exploration à élévation fixe (2D).

Les méthodes récentes d'exploration autonome hybrides utilisent souvent l'algorithme RRT. Par exemple, Selim et al. [118] améliorent les performances de la méthode **NBVP** en proposant AEP, une extension hybride. Dans un premier temps, les auteurs d'AEP accélèrent l'évaluation des nœuds candidats échantillonnés via une approche de lancer de rayon éparse, accélérant par conséquent le processus d'exploration. De plus, pour partiellement résoudre le problème de terminaison prématurée de NBVP, la méthode AEP retient les positions de nœuds précédemment étendus par NBVP et les réévaluent seulement s'ils sont dans une région modifiée par l'exploration. Les nœuds retenus sont considérés comme des frontières quand le gain d'information potentiel calculé est élevé. L'exploration est guidée vers ces nœuds quand la stratégie NBVP ne trouve plus de solution.

Dai et al. [166] ainsi que Brunel et al. [3] se servent de l'algorithme RRT pour estimer des chemins et des points de vue qui mènent le robot à proximité de frontières candidates sélectionnées. L'arbre RRT s'étend jusqu'à trouver un nœud proche de chacun de ces candidats. D'une part, Dai et al. choisit les frontières candidates uniformément par octant dans un Octree [57] utilisé comme carte d'occupation. Ils utilisent une fonction d'utilité qui minimise l'entropie de la carte d'occupation en choisissant les points de vue qui observent une forte entropie. D'autre part, Brunel et al. [3] s'appuient sur une méthode de filtrage bilatérale efficace qui attribue un score d'intérêt à chaque voxel frontière en fonction de la densité de volume inconnu à proximité. Les scores d'intérêts attribués servent à sélectionner des frontières candidates pertinentes. Cette stratégie permet d'orienter naturellement l'exploration vers les régions avec une forte densité de frontières et de régions inconnues. La fonction de gain choisie dans leur approche repose sur le volume inconnu observable par un point de vue cible relatif au temps nécessaire pour atteindre sa position. Cette fonction permet ainsi de faire un compromis entre une exploration locale et globale.

D'autres stratégies hybrides n'utilisent pas l'algorithme RRT mais approximent une solution du problème du voyageur du commerce pour trouver un chemin qui parcoure tous les amas de frontières extraits [167, 168]. Habituellement, ces méthodes sont composées de deux niveaux d'exploration : l'un va localement visiter les régions inconnues derrière le groupe de frontières le plus proche, tandis que l'autre approxime une solution au problème du voyageur du commerce pour estimer un parcours global efficace de tous les autres

groupes de frontières.

Des solutions hybrides ont été appliquées par le passé au problème de la coordination multi-robot [169, 170]. Franchi et al. [171, 172] ont aussi considéré l'utilisation des SRT pour cette problématique. Récemment, Dong et al. [121] ont présenté un système autonome d'exploration qui permet la coordination de plusieurs robots mobiles terrestres pour la reconstruction de scènes d'intérieurs. L'exploration y est guidée de façon à maximiser le volume découvert et la qualité de reconstruction tout en minimisant l'effort d'exploration des robots. Hardouin et al. [173] proposent une approche d'exploration utilisant une carte de routes probabilistes (PRM) pour estimer les points de vue et extraire des trajectoires d'exploration qui tiennent comptes de la qualité du modèle reconstruit, avec une fonction de gain guidée par la TSDF. Hardouin et al. [174] étendent leurs travaux au cas plusieurs drones quadrirotors. Faire collaborer plusieurs drones pour réaliser la cartographie d'un environnement peut être sujet à des contraintes de communication limitée (dans certain cas, aucune information ne peut être échangée), c'est la problématique traitée dans les travaux de Gao et al. [175]. Dans ces travaux, Gao et al. utilisent une stratégie d'exploration hybride avec un système de points de rendez-vous. Les points de rendez-vous servent de point de rencontre pour les robots. Lors de chaque rencontre, les robots échangent entre eux les régions cartographiées et sont redistribués dans les régions non encore cartographiées.

Finalement, les méthodes hybrides tirent avantage de l'exhaustivité d'exploration des méthodes basées frontières ainsi que de l'usage de fonction de gain arbitraire, permettant l'échantillonnage et l'évaluation de nouveaux points de vue. La combinaison de ces deux stratégies permet de réduire l'impact négatif des caractéristiques de chacune d'elles.

1.7 Conclusion

Au travers de ce chapitre, nous avons présenté les briques essentielles qui servent à développer un système robotisé autonome unifié avec une attention particulière sur les drones quadrirotors. Parmi ces briques essentielles, nous retrouvons :

— le SLAM, catégorie de méthodes s'appuyant sur le concept de localisation et de cartographie simultanée pour calculer continuellement des informations de localisation. Les informations calculées sont nécessaires à la navigation et à la cartographie d'occupation de l'environnement.

— la cartographie d'occupation modélisant à l'aide d'une carte d'occupation les régions navigables (sans obstacle)

— la planification de trajectoire sans collision dans une carte d'occupation

— les méthodes de contrôle du drone quadrirotor

Pour finir, nous avons proposé un état de l'art visant à résoudre le problème d'exploration autonome pour un robot équipé de capteurs. Cette dernière brique, décompose les méthodes proposées dans la littérature en trois catégories : (i) méthodes basées frontière, (ii) méthodes basées échantillonnage et (iii) méthodes hybrides, permettant la mise en contexte des contributions proposées.

Dans le chapitre suivant, nous présentons nos travaux [3] qui décrivent (i) une méthode de cartographie d'occupation utilisant une grille régulière et (ii) notre méthode d'exploration autonome hybride.

Chapitre 2

Exploration autonome efficace

2.1 Introduction

Maintenant que nous avons détaillé les composants et algorithmes essentiels nécessaires à la conception d'un système d'exploration et de cartographie autonome robotisé, nous présentons une stratégie d'exploration autonome adaptée aux drones quadrirotors équipés d'une caméra de profondeur. Notre méthode réalise simultanément l'estimation de trajectoires d'exploration sans collisions et une reconstruction de l'environnement sous forme d'une grille 3D de voxels. Pour réaliser cette double tâche de manière efficace, nous présentons une méthode originale d'exploration s'appuyant sur le filtrage bilatéral. Cette méthode repose sur l'attribution d'un score d'intérêt aux voxels situés aux frontières des régions précédemment visitées afin de guider l'échantillonnage de nouveaux objectifs puis la sélection de trajectoires à exécuter.

Nous commençons ce chapitre par une formalisation du problème d'exploration autonome à laquelle notre méthode s'intéresse. Puis, nous détaillons les concepts principaux qu'utilise notre système d'exploration et comparons avec l'état de l'art de la littérature scientifique. Enfin, nous présentons les briques importantes de notre approche, à savoir une méthode de cartographie d'occupation et une méthode d'exploration autonome hybride.

2.2 Formalisation du problème

Soit $\mathcal{V} \subset \mathbb{R}^3$ un ensemble de voxels appartenant à la carte d'occupation couvrant l'espace de travail à explorer. La tâche d'exploration autonome consiste à référencer l'état d'occupation $\{$*'libre', 'occupé'*$\}$ de tous les voxels $\mathbf{v} \in \mathcal{V}$ initialement étiquetés comme *'inconnu'*. L'objectif est de planifier des déplacements sans collision, effectués par un robot explorateur : dans notre cas d'étude, il s'agit d'un drone quadrirotor instrumenté d'une caméra de profondeur et d'un système de localisation étalonnés, rigidement attachés au drone. Les performances des méthodes sont évaluées dans des environnements simulés maitrisés. Le drone est limité en vitesse linéaire par la grandeur v_{max}, accélération linéaire a_{max} et vitesse angulaire en lacet ω_{max}. La caméra de profondeur quant à elle possède un champ de vue (horizontal et vertical) choisi arbitrairement et une distance de captation limitée à d_{max}.

Le système de cartographie et d'exploration que nous proposons exploite la notion de *'frontière'* [2]. Une frontière $\boldsymbol{f}$ appartient à l'ensemble des voxels $\in \mathcal{F} \subset \mathcal{V}$ étiquetés *'inconnu'* et voisin de voxels étiquetés *'libre'*. Notre méthode calcule et exécute itérativement des chemins sans collision pour le drone, jusqu'à ce que toutes les frontières atteignables aient été visitées ou qu'un temps limite ait été atteint.

2.3 État de l'art

Dans ces travaux, nous présentons une méthode d'exploration hybride reposant sur les deux stratégies principales d'exploration autonome :

1. la première utilise un raisonnement basé frontière [2],
2. la seconde est basée sur l'échantillonnage [116] des positions via des arbres aléatoires à exploration rapide (RRT) [77, 78, 176].

Cette stratégie hybride permet d'effectuer efficacement une exploration globale (couvrir l'ensemble des régions inconnues de la carte d'occupation via la notion de frontière) et une exploration locale (choix des trajectoires d'exploration et orientation du capteur sophistiquées) de l'environnement. Dans cette section, nous présentons les éléments innovants de notre méthode en comparaison de ce qui est proposé par les méthodes populaires de la littérature scientifique.

Les deux principales méthodes d'exploration basées frontière sont :

— **Classic** [2] qui propose de parcourir chaque frontière existante de manière gloutonne grâce à un algorithme de parcours en profondeur d'une carte d'occupation avec pour objectif de trouver la frontière atteignable la plus proche.

— **Rapid** [69] qui propose une exploration à deux états : le premier priorise les frontières dans le champ de vision du drone afin de maintenir son énergie cinétique, et le cas où aucune frontière n'est visible, le second état lance une itération de la méthode "Classic".

Notre méthode se différencie de ces deux approches en proposant une façon originale et intelligente de sélectionner les prochaines frontières. Dans nos travaux [3], nous proposons

d'extraire un ensemble de frontières filtrées $\widetilde{\mathcal{F}} \subset \mathcal{F}$ en utilisant des scores d'intérêt $s(\boldsymbol{v})$ par voxel $\boldsymbol{v} \in \mathcal{F}$. Ces scores dépendent de la densité locale de voxels inconnus. Pour approximer efficacement $s(\boldsymbol{v})$, nous nous appuyons sur un processus analogue à celui d'un filtre bilatéral, qui est un filtre de lissage non-linéaire utilisé en traitement d'images.

Le filtrage bilatéral peut être mis en œuvre de plusieurs manières. Il existe notamment plusieurs structures qui accélèrent les opérations de ce type de filtre [177]. Particulièrement, le treillis permutoèdrique est une structure supportant parfaitement les trois opérations exécutées dans un filtre bilatéral "standard" à savoir *Splat*, *Blur* et *Slice*. Cette structure sépare l'espace en simplex(e) (plutôt qu'en cubes ou hypercubes dans le cas d'un partitionnement sous forme de grille) ce qui réduit la complexité algorithmique des opérations et accélère les performances de calcul [177]. Le treillis permutoèdrique est notamment utilisé dans le filtrage Gaussien des images couleurs [177]. Sa popularité s'est étendue à plusieurs domaines, à commencer par celui de l'apprentissage profond. Par exemple, Jampani et al. [178] modifient les couches de convolution en remplaçant leur fonctionnement standard par un treillis permutoèdrique combiné avec les opérations de *Splat*, *Blur* et *Slice*. Certains s'inspirent des travaux de Jampani et al. pour le traitement et la segmentation de nuage de points 3D [179-181]. Le treillis permutoèdrique a servi pour résoudre un problème plus conventionnel qui est l'alignement de nuage de points via une formulation probabiliste et une résolution efficace [182].

Le résultat de ce filtrage appliqué sûr $\widetilde{\mathcal{F}}$, est un ensemble d'objectifs ou voxels frontières qui sont des candidats prometteurs que le drone doit visiter. Ces objectifs permettent de guider plus efficacement la planification de nouvelles trajectoires d'exploration en suivant un processus décisionnel qui privilégie les frontières aux scores élevés.

Au sein de notre méthode d'exploration autonome, chaque nouvelle trajectoire d'exploration est construite de manière équivalente à celles des méthodes basées sur l'échantillonnage. Les méthodes basées sur l'échantillonnage planifient leur exploration en utilisant l'algorithme *RRT*. Pour rappel, ces méthodes étendent aléatoirement une structure arborescente autour de la position courante du drone [116]. Chaque branche et chaque nœud de l'arborescence sont respectivement un chemin et une destination potentiels. Ces trajectoires sont évaluées pour estimer la trajectoire optimale au sens d'un critère qui est le rapport entre le volume inconnu potentiellement observable à une destination donnée et le temps d'exécution de la trajectoire. Cette trajectoire est ensuite exécutée par le drone.

Déployer aléatoirement des trajectoires uniquement à la manière *RRT* induit plusieurs problèmes :

1. Généralement l'algorithme *RRT* est paramétré pour s'étendre "aléatoirement" pour une quantité définie d'itérations. Si aucune région inconnue n'existe à proximité, l'exploration se termine. On parle de terminaison prématurée s'il existe encore des régions inconnues atteignables.
2. La longueur des trajectoires est définie par la taille des branches de l'arbre. Pour chaque étape de déplacement (portion entre chaque nœud) le long d'une branche, le drone accélère puis reste à vitesse constante puis décélère jusqu'à s'arrêter. Il en résulte une perte significative d'efficacité.

3. Ces méthodes sont affectées par le biais de Voronoï, avec une tendance à sélectionner des échantillons dans les régions de Voronoï, c'est-à-dire les régions libres les plus grandes. Ce biais affecte les performances des méthodes basées échantillonnage car les déplacements et calculs de gains ont plus de chance d'être faits dans de larges régions déjà visitées, occasionnant une perte de temps.

Ces problèmes apparaissent notamment dans [116, 117, 149]. À l'origine, l'algorithme *RRT* est développé pour effectuer une recherche de chemin. C'est pourquoi, dans nos travaux, nous utilisons ce type d'algorithme uniquement pour estimer les chemins allant de la position courante du drone jusqu'aux frontières candidates sélectionnées. L'algorithme *RRT* permet d'échantillonner des positions d'arrivées proches de ces frontières situées dans un volume inconnu. Ainsi, par rapport aux méthodes basées purement sur l'échantillonnage, nous évitons :

1. une terminaison prématurée, car toutes les frontières seront visitées,
2. des déplacements inutiles, le chemin est optimisé pour être le plus direct possible dans la mesure où il est trouvé par l'algorithme,
3. des calculs de gains inutiles dans des espaces déjà visités, seule la position d'arrivée mesure le gain de la trajectoire.

La meilleure trajectoire est choisie parmi un échantillon de trajectoires. Ce choix s'appuie sur une formule de gain standard impliquant le volume inconnu potentiellement observable à la position finale de la trajectoire et le temps d'exécution de cette trajectoire.

Notre méthode complète peut s'apparenter à celle proposée dans [166], cependant, dans la méthode que les auteurs proposent, la sélection des frontières est aléatoire. De plus, leur formule de gain utilise l'entropie de la carte d'occupation plutôt que celle favorisée par les méthodes basées sur l'échantillonnage qui utilise le volume inconnu [116, 117]. Bien qu'utiliser l'entropie permette d'améliorer les régions faiblement observées, les résultats, présentés dans [166] montrent une perte d'efficacité d'exploration dans des environnements larges. Cette perte d'efficacité est causée par la fonction de gain choisie basé sur l'entropie qui induit une préférence pour les petits déplacements. Ces petits déplacements n'améliorent que légèrement l'état de la carte d'occupation au détriment d'une exploration de l'ensemble des espaces inconnus.

Par ailleurs, la méthode d'exploration que nous proposons, utilise son propre algorithme de cartographie d'occupation basé sur une grille régulière. Ce choix est motivé par plusieurs défauts et manques constatés dans les autres méthodes de cartographie d'occupation qui utilisent des structures telles que l'Octree [51, 57] ou bien la grille éparse [53]. Un Octree, est une structure efficace en espace mémoire qui n'est pas réellement affecté par la taille de l'environnement. Cependant, dans un Octree, il est difficile d'en extraire précisément une information sur les frontières. De plus, l'insertion de nouvelles informations est plus lente que les autres structures. Les grilles éparses n'apportent aucun avantage puisque l'ensemble de l'environnement sera, *in fine*, visité. La table de hachage utilisée rend les méthodes basées sur une grille éparse de nature plus lente qu'une grille régulière, surtout lorsqu'il s'agit d'y insérer de nouvelles informations. De plus, il est important de noter

que les méthodes de cartographie d'occupation récentes [51, 53, 57, 58] n'extraient pas les frontières.

En résumé, les travaux présentés de ce chapitre apportent les contributions suivantes :

— Une méthode d'exploration autonome hybride originale qui cartographie et planifie des trajectoires d'exploration sans risque.

— Une nouvelle façon de quantifier et d'attribuer un score d'intérêt aux voxels frontières. Pour approximer ce score, nous avons adapté le treillis permutoèdrique et utilisons les opérations très efficaces du filtrage bilatéral.

— Le système d'exploration complet que nous proposons dépasse l'état de l'art en termes d'efficacité de l'exploration, comme le montrent les expériences quantitatives et qualitatives en environnements simulés contrôlés (c.f. 4).

— Nous proposons par ailleurs une expérience en environnement réel et des détails supplémentaires sur notre méthode et les expériences simulées dans la vidéo suivante `https://www.youtube.com/watch?v=DCcfA2HB1GI&t=82s`.

— Le code de notre méthode est disponible sur GitHub : `https://github.com/anthonybrunel/splatplanner`.

2.4 Cartographie d'occupation

2.4.1 Prérequis et données d'entrées

La méthode de cartographie d'occupation que nous proposons construit une carte d'occupation via une grille régulière tridimensionnelle composée d'un ensemble $\mathcal{V} \subset \mathbb{Z}^3$ de voxels $\boldsymbol{v}$. Cette grille partitionne un volume borné qui correspond à l'espace de travail, dans lequel le drone explorateur opère. La grille régulière est souvent représentée, comme un tableau d'une seule dimension contiguë en mémoire. Nous allons présenter dans cette section la façon dont nous insérons les mesures des capteurs de profondeur et d'odométrie dans la carte d'occupation, les étapes qui permettent sa mise à jour et les informations qu'elle restitue.

Notations, repères de références, changement de repère et référencement des voxels

Les repères de référence utilisés pour mettre à jour la carte d'occupation et effectuer des requêtes sur celle-ci sont listés ci-dessous :

— Le repère monde W est un repère orthonormé appartenant à $\mathbb{R}^3$. En simulation, l'origine O^W du repère W est définie à l'origine de l'environnement simulé. En expérience réelle, si aucun paramètre n'est spécifié, O^W est localisée là où le système de localisation est initialisé. Par rapport à la position initiale du drone : X^W pointe vers l'avant, Y^W pointe vers la gauche et Z^W pointe vers le haut. C'est le repère dans

lequel certaines opérations sont faites telles que la localisation, le calcul et le maintien de trajectoire, ou encore la visualisation de la carte d'occupation.

— Le repère relatif au capteur (caméra de profondeur) que nous dénoterons C est un repère orthonormé appartenant à $\mathbb{R}^3$. L'origine correspond au centre optique de la caméra. Z^C est l'axe optique du système X^C est aligné sur les lignes de l'image et Y^C sur les colonnes. Le nuage de points calculé à partir des mesures du capteur est référencé dans ce repère, avant d'être transformé dans le repère monde avec les informations de localisation.

Nous dénoterons T_c^W la transformation qui lie le repère monde W et le repère du capteur C. Cette transformation est donnée par le système de localisation. Nous ajoutons aussi que quel que soit $\boldsymbol{p}_i^W$ compris entre les bornes inférieures et supérieures de la grille régulière il existe un voxel $\boldsymbol{v} \in \mathcal{V}$ identifié par son centre géométrique $\boldsymbol{p}_v^W$.

Résolution

Un voxel v est associé à une position $\boldsymbol{p}_i^W$ via quelques opérations simples qui utilisent notamment la résolution r de la grille régulière. En exploration autonome [116-118, 166], les valeurs de résolution r utilisées, dans la littérature, sont en général soit : 0.1 (m) soit 0.2 (m). Le choix entre ces deux valeurs dépend de la taille de l'environnement à explorer. Une valeur élevée permet de garder des performances acceptables lors de la mise à jour de la carte d'occupation tandis qu'une valeur de résolution faible permet d'obtenir une carte d'occupation plus précise.

État d'occupation et TSDF

Comme évoqué précédemment, chaque voxel contient plusieurs informations. Ces informations sont mises à jour dès lors que le voxel est observé. Elles dépendent de la position du voxel dans l'environnement et du volume qu'il englobe. Nous listons les informations utilisées dans notre cartographie d'occupation ci-dessous :

— État d'occupation : un voxel est étiqueté *'occupé'*, *'libre'* ou *'inconnu'*.

— *'Frontière'* : un voxel *'inconnu'* peut être étiqueté comme *'frontière'* s'il est adjacent à un voxel *'libre'*.

— TSDF $D(\boldsymbol{v})$ [66] : $D(v)$ est une distance qui détermine si $\boldsymbol{v}$ est *'occupée'* ou *'libre'*. À l'initialisation, si aucun *'a priori'* n'existe, tous les voxels sont *'inconnus'*, aucune valeur de distance n'est attribuée. Dès lors qu'un voxel est observé par le capteur, une valeur $D(\boldsymbol{v})$ lui est attribuée. $D(\boldsymbol{v})$ est compris dans l'intervalle $[-1, 1]$, c'est une distance normalisée par rapport à la valeur de distance de troncature D_{trunc}. D_{trunc} est définie arbitrairement. Dans nos expériences nous prenons 0.3 (m) pour une carte d'occupation de résolution $r = 0.1$(m) et 0.6(m) pour $r = 0.2$(m). L'état

d'occupation d'un voxel est fonction de $D(\boldsymbol{v})$, si $D(\boldsymbol{v}) \in [-\frac{1}{3}, \frac{1}{3}]$ le voxel est *'occupé'* sinon si $D(\boldsymbol{v}) > \frac{1}{3}$ le voxel est *'libre'*.

Les informations décrites ci-dessus sont importantes, voire indispensables pour la navigation et l'exploration autonome d'environnements.

Données d'entrée (mesures capteur)

Notre méthode de cartographie d'occupation considère que le robot est équipé d'un capteur de profondeur (e.g. Intel Realsense D455, Kinect, ou bien caméra de profondeur simulée). Ce type de capteur restitue à chaque acquisition une image dans laquelle l'intensité de chaque pixel correspond à la distance entre le capteur et la surface observée. Pour pouvoir être exploités, les paramètres intrinsèques du capteur de profondeur doivent être connus. Celui-ci doit être étalonné et synchronisé temporellement avec le système de localisation. Le système de localisation donne la position, l'orientation, la vitesse linéaire et la vitesse angulaire dans le repère monde (W). Les paramètres intrinsèques permettent de connaître précisément le champ de vision du capteur de profondeur et, combiné au système de localisation, de reconstruire correctement les régions observées. À chaque acquisition, la cartographie d'occupation reçoit une image de profondeur I et la localisation T_c^W du capteur dans le repère monde.

2.4.2 Mise à jour de la carte d'occupation

Deux processus actualisent les informations contenues dans la carte d'occupation :

1. le premier s'effectue lors de la réception de nouvelles mesures. Il actualise l'état d'occupation et la TSDF de chaque voxel observé.
2. le second s'effectue à intervalle de temps régulier et actualise le champ de distance euclidien.

1. Nouvelles mesures

Lors de la réception de nouvelles mesures, plusieurs opérations s'effectuent. Nous les listons et les décrivons ci-dessous :

— **Projection π:** étant donné $\boldsymbol{p}_i^C$ un point 3D dans le repère de la caméra, π associe $\boldsymbol{p}_i^C$ à un pixel z_i en le projetant sur le plan image de la caméra. Nous définissons l'opération π comme suit :

$$z_i = \pi(\boldsymbol{p}_i^C).$$

Nous utilisons l'opération π pour déterminer si un voxel est occulté du point de vue de la caméra via une opération de différence sur la valeur de profondeur donnée par le pixel et la distance à laquelle est $\boldsymbol{p}_i^C$ de la caméra.

— **Rétroprojection π^{-1} :** étant donné z_i un pixel (comprenant ses coordonnées image et l'information de profondeur), π^{-1} calcule les coordonnées du point 3D $\boldsymbol{p}_i^C$ associé à z_i. Nous définissons l'opération π^{-1} comme suit :

$$\boldsymbol{p}_i^C = \pi^{-1}(z_i).$$

Cette opération est l'inverse de l'opération de projection, nous l'utilisons pour calculer un nuage de points avec l'image de profondeur acquise par le capteur.

— **Lancer de rayons ζ:** étant donné $\mathcal{V}$ une grille régulière, $\boldsymbol{p}_c^W$ la position de la caméra et $\boldsymbol{p}_i^W$ un point 3D calculé grâce à l'image de profondeur, l'opération de lancer de rayon ζ calcule la liste Q des voxels traversés par le rayon et compris dans l'intervalle $[-D_{trunc}, D_{trunc}]$ autour, $\boldsymbol{p}_i^W$. Nous définissons l'opération ζ comme suit :

$$Q = \zeta(\mathcal{V}, v, \boldsymbol{p}_i^W).$$

Davantage de détails sont donnés dans [183].

— **Mise à jour de la TSDF et des états d'occupation Γ :** étant donné une carte d'occupation $\mathcal{V}$, un voxel $v \in \mathcal{V}$ et un point 3D $\boldsymbol{p}_i^W$ acquis par le capteur, Γ restitue une nouvelle configuration de la carte d'occupation en attribuant une nouvelle valeur de TSDF et un nouvel état à v si nécessaire. Nous définissons l'opération Γ comme suit :

$$\mathcal{V}' = \Gamma(\mathcal{V}, v, \boldsymbol{p}_i^W).$$

Nous utilisons Γ pour mettre à jour la carte d'occupation en actualisant les valeurs de TSDF et l'état d'occupation (*'libre'* ou *'occupé'*) d'un voxel observé par le capteur.

— **Mise à jour des voxels *'frontières'* $\digamma$:** soit une carte d'occupation $\mathcal{V}$, un voxel $v \in \mathcal{V}$, $\digamma$ restitue une nouvelle configuration de la carte d'occupation en actualisant l'état de *'frontière'* de v en fonction de son voisinage local dans $\mathcal{V}$. Nous définissons l'opération $\digamma$ comme suit :

$$\mathcal{V}' = \digamma(\mathcal{V}, v).$$

L'état de *'frontière'* est attribué s'ils respectent la propriété d'une *'frontière'* définie précédemment.

Toutes ces opérations sont seulement appliquées dans le champ de vision courant du capteur. Le processus qui actualise la carte d'occupation est décrit dans l'algorithme 1. Cette actualisation commence par la mise à jour des informations de TSDF des voxels observés par le capteur situé proche d'une surface (murs, obstacles, objets...). Ensuite, les voxels situés dans le champ de vision sont projetés sur le plan image du capteur pour actualiser les voxels libres non occultés. Pour finir, tous les voxels compris dans le voisinage d'un voxel actualisé par les étapes précédentes est étiqueté ou destitué de l'état *'frontière'* s'il respecte les propriétés définies précédemment d'une *'frontière'*.

Combiner l'opération de lancer de rayon et l'opération de projection pour actualiser la carte d'occupation permet de garder : (i) la précision qu'offre le lancer des rayons pour les

Algorithme 1 : Mise à jour de la carte d'occupation

Entrées : I image de profondeur, T_c^W

$Q \leftarrow \emptyset$;
pour chaque *pixel* $z_i \in I$ **faire**
  si $d(z_i) > 0$ *et* $d(z_i) < d_{max}$ **alors**
    $p_i^W \leftarrow T_c^W * \pi^{-1}(z_i)$ ▷ p_i^W position du pixel i;
    $Q \leftarrow \zeta(V, \boldsymbol{p}_c^W, p_i^W)$ ▷ Q liste des voxels collectés par lancer de rayon;
    pour chaque $v \in Q$ **faire**
      $\mathcal{V} \leftarrow \zeta(\mathcal{V}, v, \boldsymbol{p}_i^W)$ ▷ Mise à jour des états des voxels collectés par lancer de rayons;
pour chaque *voxel* $v \in$ *au champ de vision de la caméra* **faire**
  si $v \notin Q$ **alors**
    $z_i \leftarrow \pi(p_v^c)$;
    $p_i^W \leftarrow T_c^W \pi^{-1}(z_i)$;
    $\mathcal{V} \leftarrow \Gamma(\mathcal{V}, v, p_i^W)$ ▷ Mise à jour des état d'occupation par projection;
pour chaque *voxel* v *dans le voisinage des voxels mis à jour* **faire**
  $\mathcal{V} \leftarrow \digamma(\mathcal{V}, v)$ ▷ Mise à jour des frontières;

voxels proches des surfaces et (ii) des performances de calcul acceptables. Pour mieux comprendre le gain d'efficacité, nous détaillons la complexité des deux opérations ci-dessous :

— **Lancer de rayon :** la complexité algorithmique de l'opération de lancer de rayon appliquée sur tous les voxels est $O(N_{ray} * d_{max}/r)$ où N_{ray} est le nombre de rayons lancés et dépend de la résolution de l'image (i.e. 307200 pour une image de $640 * 480$). La complexité du lancer de rayon croît en fonction de la résolution de l'image, de la résolution de la carte d'occupation (donnée en mètres) et également de la distance maximale de vision du capteur.

— **Projection :** la complexité algorithmique de l'opération de projection appliquée sur tous les voxels dans le champ de vision est $O(N_v)$ où N_v est le nombre de voxels couverts par le champ de vision du capteur.

En pratique et notamment dans le cas de nos expériences N_v est inférieur à $N_{ray} * d_{max}/r$ et l'opération de projection est moins coûteuse que celle du lancer de rayon. La Figure 2.1 illustre l'actualisation des voxels par projection (en vert) et par lancer de rayons (en rouge). Les Figures 2.2(a) et 2.2(b) montrent les performances des deux opérations dans un environnement simulé maîtrisé avec une configuration standard.

2. Champ de distance euclidien

En parallèle de l'insertion de nouvelles mesures et de la mise à jour des informations rela-

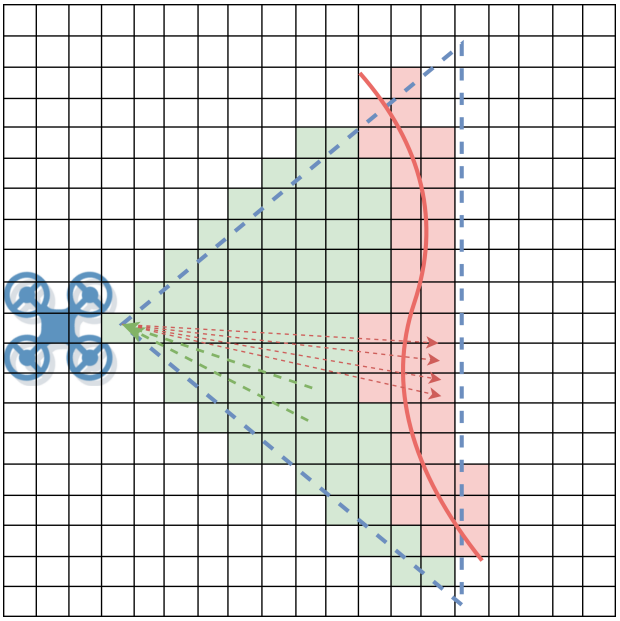


FIGURE 2.1 – Cette Figure illustre les voxels actualisés par reprojection (vert) et par lancer de rayon (rouge). Seuls les voxels colorés dans le champ de vision sont actualisés. Réduire le nombre de voxels traités par lancer de rayon réduit le temps de mise à jour de la carte d'occupation.

tives à la carte d'occupation, nous actualisons un champ de distance euclidienne (EDF) à intervalle de temps régulier. Pour ce faire, nous utilisons l'algorithme de Felzenszwalb et al. [59] (évoqué dans le chapitre 1). Pour rappel, cet algorithme attribue une valeur à chaque voxel $\boldsymbol{v} \in \mathcal{V}$. Cette valeur correspond à la distance en mètres entre le centre géométrique de $\boldsymbol{v}$ et celui du voxel *'inconnu'* ou *'occupé'* le plus proche de $\boldsymbol{v}$. Cet algorithme est de complexité linéaire, ce qui le rend efficace pour calculer un champ de distance euclidien. La mise à jour des distances est effectuée à une fréquence de 0.5 Hz (i.e. toutes les 500 ms). L'intervalle de temps choisi entre chaque mise à jour est suffisant dès lors que la vitesse de déplacement du drone reste raisonnable. Finalement, le champ de distance est utilisé pour planifier des trajectoires et limiter les déplacements du drone dans l'espace libre de la carte d'occupation. Ceci permet d'éviter au drone de s'approcher trop près des régions inconnues ou occupées.

2.4.3 Statistiques

Le temps de mise à jour de la carte d'occupation lors de la réception de nouvelles mesures est important. Ce temps ne doit pas excéder la fréquence d'acquisition du capteur si possible. Si c'est le cas, certaines images acquises par le capteur peuvent ne pas être considérées. La Figure 2.2 présente, sous la forme de quatre histogrammes le temps d'insertion prit par le lancer de rayon, la projection, la mise à jour des frontières et le temps total des trois processus cumulés. Ces mesures ont été acquises avec les configurations de simulation

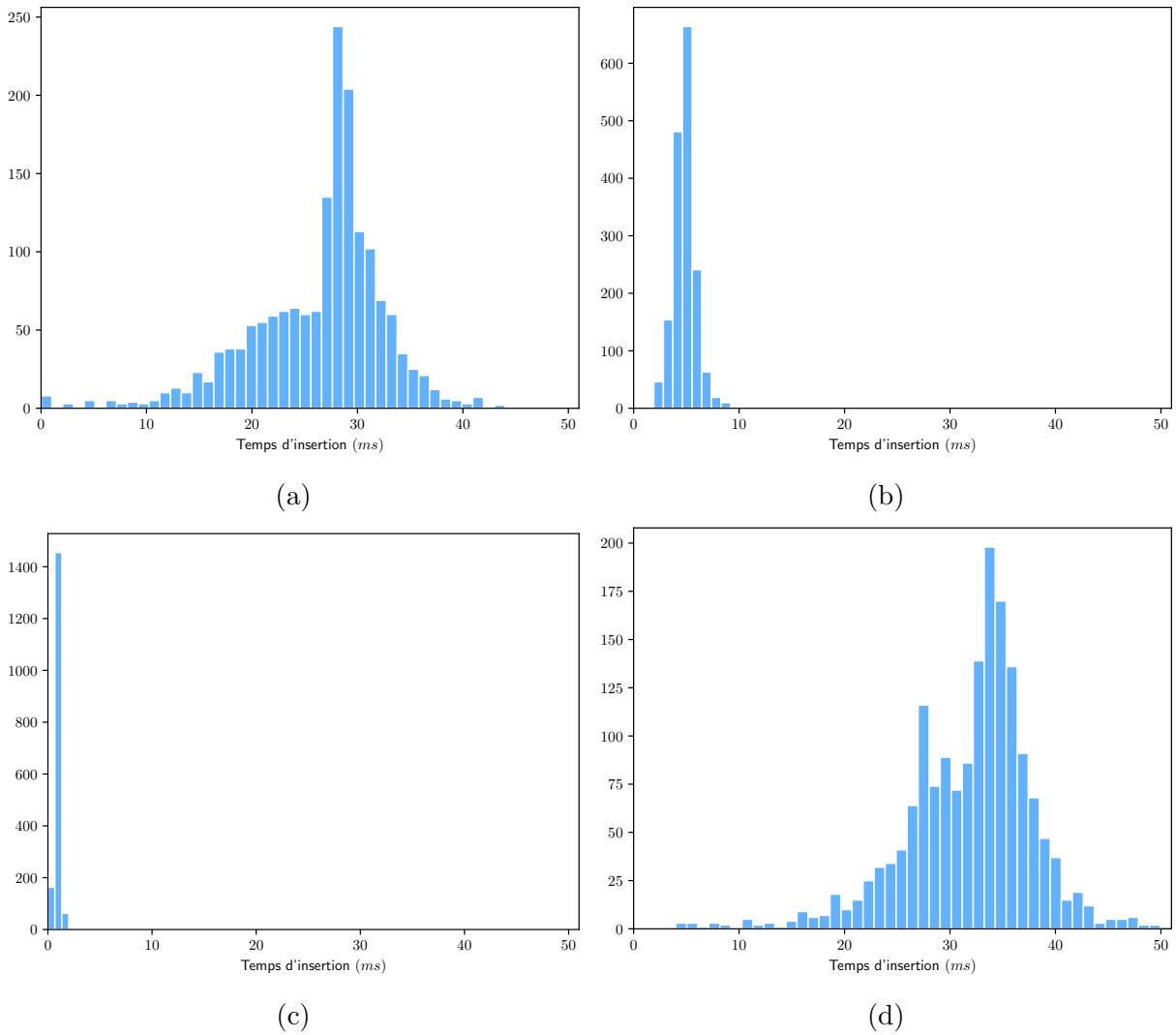
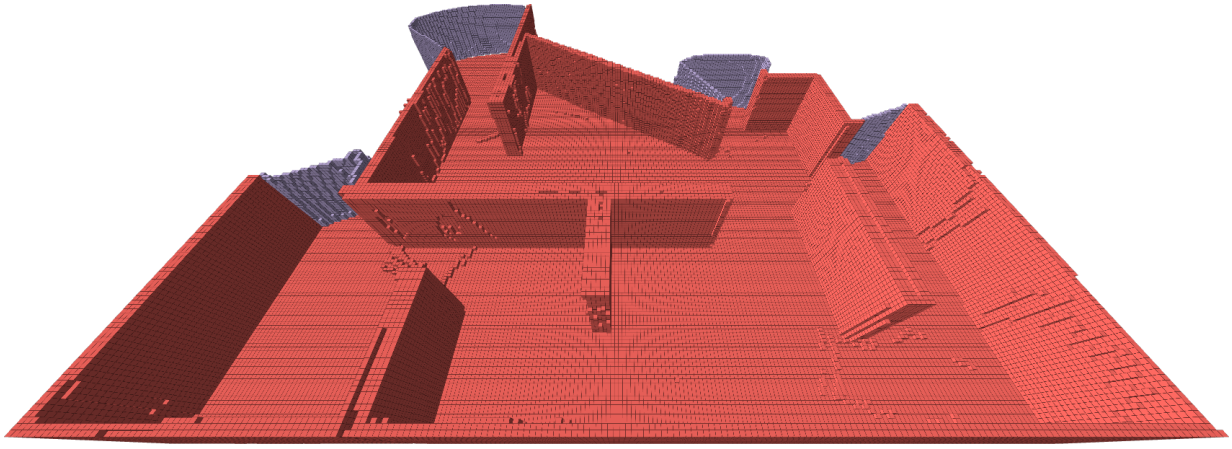


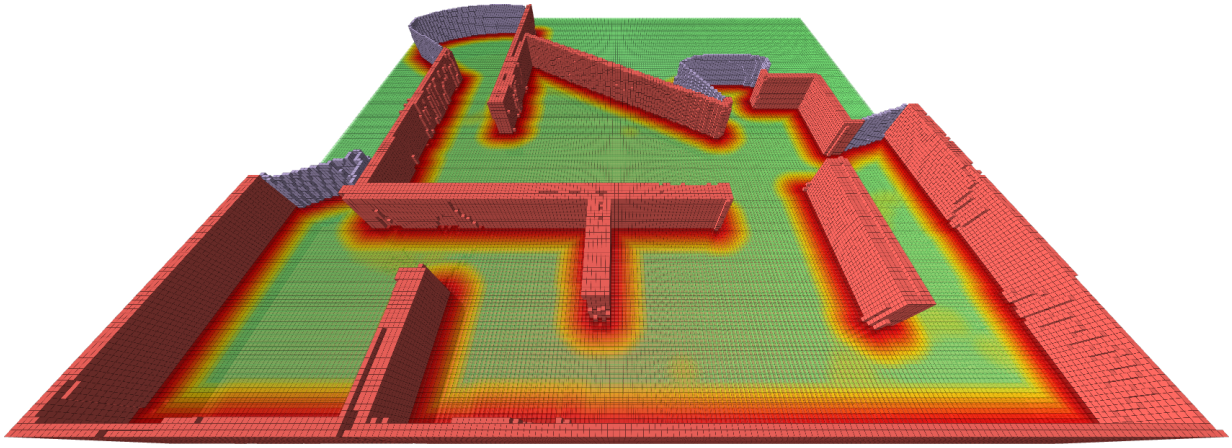
FIGURE 2.2 – Les histogrammes (a), (b) et (c) correspondent respectivement aux temps pris par l'opération lancer de rayons, aux temps pris par l'opération de projection et aux temps de mise à jour des frontières. L'histogramme (d) est le cumul de temps des trois opérations. Acquisition correspondant à la scène MAZE.

suivantes : une image de profondeur simulée de résolution 640x480, une résolution de la carte d'occupation 0.1 (m) et sur un ordinateur portable équipé d'un processeur Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz sous Ubuntu 20.04. Ces résultats montrent que notre cartographie d'occupation supporte assez facilement une fréquence d'acquisition de 20 Hz (fréquence raisonnable et suffisante pour notre cas d'usage) pour les configurations données dans la description de la Figure 2.2. Ces configurations sont très proches de celles qu'il est possible d'embarquer sur un drone quadrirotor. Le cumul des trois opérations prend en général moins de 40 ms (c.f. Figure 2.2(d)). La Figure 2.3 correspond à un affichage de l'état des voxels à un moment d'une expérience d'exploration autonome simulée. Sur cette figure les voxels contenus dans la carte d'occupation sont colorés en fonction de leur état ou de leur valeur d'EDF respectif à un temps. Pour améliorer les performances de la cartographie d'occupation ou sa qualité, il est possible de modifier des paramètres tels que la résolution de la carte d'occupation, la taille des images acquises, la distance maximale d'acquisition et la distance de troncation de la TSDF.

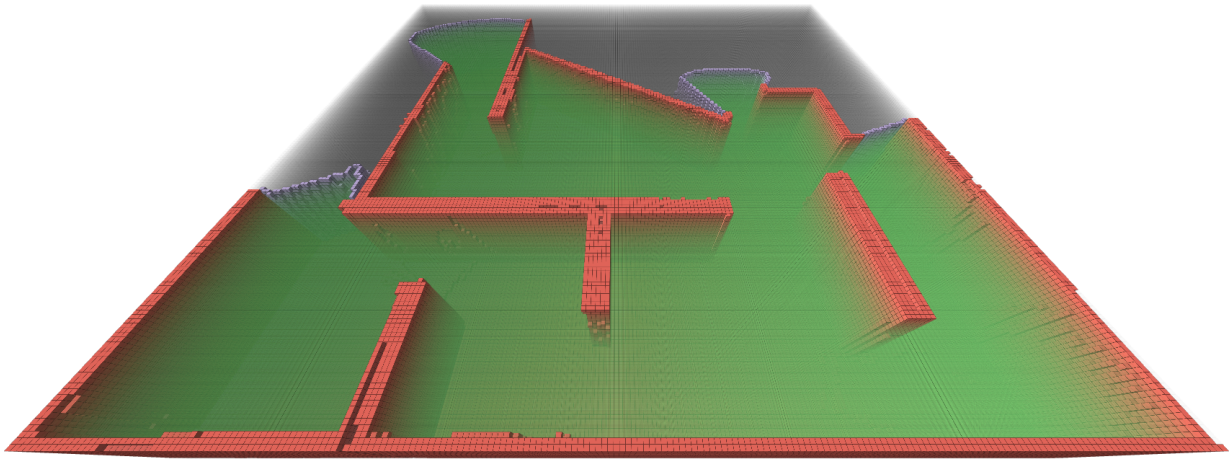
Les données calculées par l'algorithme de cartographie d'occupation que nous venons de présenter sont utilisées par la méthode d'exploration autonome que nous présentons dans la section suivante.



(a)



(b)



(c)

FIGURE 2.3 – Visualisation des différentes données restituées par notre méthode de cartographie d'occupation. (a) visualisation des voxels occupés (rouge), des frontières (violet). (b) visualisation d'une tranche du champ de distance euclidienne colorée en fonction de la distance à l'obstacle le plus proche (sur l'image : rouge 0m, vert >1m) (c) visualisation du volume libre en vert et volume inconnu en gris.

2.5 Méthode d'exploration proposée

2.5.1 Vue d'ensemble du processus d'exploration

La tâche d'exploration autonome avec un drone quadrirotor peut se compose d'une étape d'initialisation dans laquelle le drone décolle à 1 m de hauteur et effectue une rotation de 360 degrés autour de l'axe vertical (lacet). Cette étape permet d'initialiser la carte d'occupation (les régions *'occupées'*, les régions *'libres'* et les *'frontières'*) à proximité du drone.

L'approche d'exploration autonome (SplatPlanner [3]) proposée exploite les informations calculées par la méthode de cartographie d'occupation que nous avons définie précédemment. Nous calculons itérativement des trajectoires d'exploration sans collision à suivre par le drone, jusqu'à ce que toutes les frontières atteignables soient visitées ou qu'un temps limite soit atteint. Pour calculer ces trajectoires d'exploration, nous utilisons un système de score d'intérêt pour extraire un ensemble d'objectifs G de *'frontières'* candidates. À l'aide de l'algorithme RRT^*, à la manière des méthodes basées sur l'échantillonnage, nous construisons un chemin de la position courante du drone vers une position finale proche d'une des *'frontières'* candidates et ceci pour chaque objectif. Chaque position finale est évaluée pour mesurer le volume inconnu potentiellement observable par rapport au temps requis par le drone pour l'atteindre. Le chemin ayant le ratio (aussi nommé le gain) le plus élevé est choisi.

En résumé, l'évaluation d'une trajectoire d'exploration se décompose en quatre étapes détaillées dans cette section :

1. Attribution d'un score d'intérêt par voxel *'frontière'*.
2. Extraction des voxels *'frontières'*, candidats prometteurs.
3. Échantillonnage de trajectoires vers les candidats prometteurs.
4. Estimation du ratio des trajectoires échantillonnées et exécution de la meilleure d'entre elles.

2.5.2 Attribution d'un score d'intérêt aux frontières

Chaque nouvelle requête de trajectoire d'exploration commence par le calcul d'un score d'intérêt attribué au voxel $\boldsymbol{f} \in \mathcal{F} \subset \mathcal{V}$ étiqueté comme *'frontière'*. Le score d'intérêt est une approximation de la "densité" de voxels *'inconnus'* dans le voisinage local de $\boldsymbol{f}$. Ce score est le premier critère utilisé pour guider l'exploration. L'intuition est la suivante : étant donné qu'une *'frontière'* est un voxel *'inconnu'* voisin d'un voxel *'libre'*, une *'frontière'* avec un score élevé signifie qu'elle est à proximité d'une grande quantité de voxel *'inconnus'* et est donc plus probable de mener vers des régions inconnues "volumineuse".

Notre méthode d'attribution de score s'inspire du filtrage gaussien à haute dimension (dimension >= 3). Ce filtrage est un moyen d'effectuer le lissage de données contenues dans un espace euclidien arbitraire. En tant que tel, le filtrage gaussien à haute dimension est très utilisé en traitement d'image. Ce filtre peut être mis en œuvre en intégrant d'abord les valeurs d'entrée (e.g. couleur et position pour une image) dans une structure adaptée (grille,

treillis permutoédrique, kd-tree) avec une pondération par valeurs d'entrée. Cette première opération d'intégration est couramment appelée "**Splat**". Dans ce même espace, un filtre séparable gaussien est appliqué sur chaque valeur accumulée. Cette seconde opération de filtrage est couramment appelée "**Blur**". Enfin, les nouvelles valeurs calculées dans l'espace à haute dimension sont échantillonnées aux positions originales de départ avec la même pondération que lors de l'opération **Splat**. Cette dernière opération est couramment appelée "**Slice**". Les trois opérations successives constituent l'algorithme dit du "filtre bilatéral".

Dans l'objectif de calculer efficacement le score d'intérêt de chaque voxel *'frontière'*, nous utilisons un treillis permutoèdrique. C'est une structure adaptée pour le filtrage bilatéral permettant une application efficace des opérations de **Splat**, **Blur** et **Slice**. Nous commençons cette section par quelques définitions mathématiques du concept de treillis et plus particulièrement d'un treillis permutoèdrique. Ensuite, nous donnons des détails sur les propriétés structurelles et les opérations de manipulations utiles pour notre cas d'usage. Pour finir, nous proposons un nouveau processus d'attribution de score d'intérêt permettant d'extraire par la suite les voxels *'frontières'* intéressants via les trois opérations (**Splat**, **Blur**, **Slice**) appliquées habituellement sur une image. Chaque définition et théorème repris ci-dessous sont extraits et prouvés dans la note technique écrite par Baek et al. [184]. Nous nous servons de ces définitions pour introduire et expliquer le fonctionnement du treillis permutoèdrique et des opérations utilisées dans un filtre bilatéral.

Définition portant sur le treillis permutoédrique

Le terme treillis en mathématique peut être défini comme suit :

Définition 2.5.1. Un treillis L de $\mathbb{R}^d$ est un sous-groupe additif discret de $\mathbb{R}^d$, tel que le sous-espace vectoriel engendré par L soit égal à $\mathbb{R}^d$.

En pratique, un treillis peut être vu comme un ensemble infini de points dans $\mathbb{R}^d$ avec les propriétés suivantes : l'addition ou la soustraction de deux points de L produit un autre point de L; chaque point est séparé par une distance minimum arbitrairement choisie (borne inférieure) et enfin chaque point de L est à une distance maximale d'un autre point de L (borne supérieure). Un treillis est donc symétrique par translation et sa structure locale est invariante, ce qui facilite la conception d'algorithmes opérant sur cette structure, puisque chaque point du treillis peut être traité de la même manière. En guise d'exemple, la grille régulière utilisée dans notre méthode de cartographie d'occupation est un treillis entier de dimension 3, homéomorphe à $\mathbb{Z}^3$. Chaque voxel identifié par son centre géométrique est un sommet du treillis entier. Plus généralement, on peut noter que le treillis entier $\mathbb{Z}^d$ est un treillis dans $\mathbb{R}^d$ pour lequel les sommets sont des d-uplet d'entiers.

Une façon typique d'étudier la structure locale d'un ensemble de points discrets est d'examiner ses cellules de Voronoï.

Définition 2.5.2 ([184])**.** Soit $\boldsymbol{x}$ un point dans le treillis $L \subset \mathbb{R}^d$, la cellule de Voronoï de $\boldsymbol{x}$ est l'ensemble des sommets de L les plus proches de $\boldsymbol{x}$ parmi tous les autres sommets du treillis.

L'espacement uniforme naturel des sommets d'un treillis induit que les cellules de Voronoï sont des enveloppes convexes uniformes, liées les unes aux autres par translation. Les cellules de Voronoï d'un treillis maillent l'espace sous-jacent $\mathbb{R}^d$ vu que tout point dans $\mathbb{R}^d$ peut être associé au sommet du treillis le plus proche.

Définition 2.5.3 ([184])**.** La cellule de Delaunay d'un treillis est l'enveloppe convexe de tous les points du treillis dont les cellules de Voronoï partagent un sommet commun.

Les cellules de Delaunay se construisent facilement en passant par le graphe dual des cellules de Voronoï et induisent une notion naturelle de proximité pour tout point arbitrairement choisi dans l'espace avec les sommets du treillis. La Figure 2.4 montre les cellules de Voronoï et Delaunay pour le cas du treillis entier et du treillis permutoédrique en dimension 2. Notez que l'ensemble $\mathbb{R}^d$ utilisé dans les définitions ci-dessus peut être remplacé par n'importe quel ensemble isométrique de $\mathbb{R}^d$. Par exemple, H_d est un hyperplan de dimension d inclus dans $\mathbb{R}^{d+1}$ et est donc isométrique à $\mathbb{R}^d$. N'importe quel sous-ensemble discret additif de H_d est un treillis et ses cellules de Voronoï et Delaunay sont incluses dans H_d.

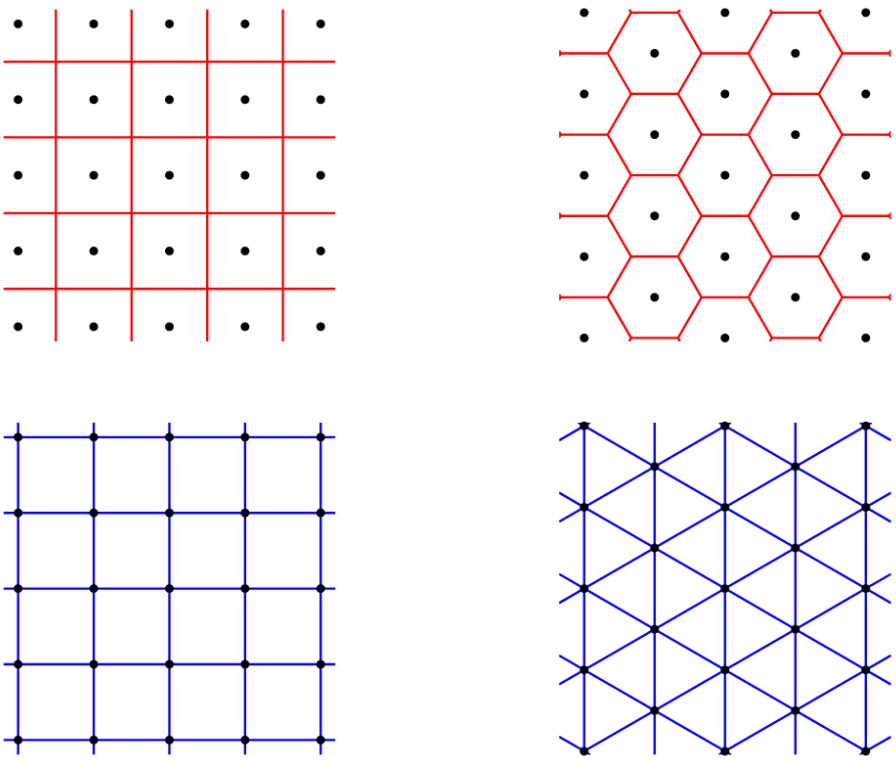


FIGURE 2.4 – Illustration extraite de [184]. Ligne du haut : les cellules de Voronoï de $\mathbb{Z}^2$ (à gauche) et A_d^* (à droite) avec les sommets du treillis. Ligne du bas : les cellules de Delaunay graphe dual des cellules de Voronoï au-dessus.

Définition 2.5.4 ([184]). Le treillis permutoèdrique racine A_d est défini par :

$$A_d := \{\ \boldsymbol{x} = (x_0, ..., x_d) \in \mathbb{Z}^{d+1} \mid \boldsymbol{x} \cdot \overrightarrow{\mathbf{1}} = 0\ \}.$$

Définition 2.5.5 ([184]). Le treillis permutoèdrique noté A_d^* est le treillis dual de A_d dans H_d:

$$A_d^* := \{\boldsymbol{x} \in H^d | \boldsymbol{y} \in A_d,\ \boldsymbol{x} \cdot \boldsymbol{y} \in \mathbb{Z}\}.$$

Traditionnellement, le treillis permutoèdrique A_d^* est le graphe dual du treillis permutoèdrique racine A_d. Il est d'usage de définir A_d^* 2.5.5 et A_d 2.5.4 comme des sous-ensembles de $H_d \subset \mathbb{R}^{d+1}$ de façon à faciliter leur manipulation. Où H_d est un hyperplan constitué des points dont la somme des coordonnées est égale à zéro :

$$H_d := \{\ \boldsymbol{x} \mid \boldsymbol{x} \cdots \overrightarrow{\mathbf{1}} = 0\ \},$$

avec $\boldsymbol{x}$ un point de l'hyperplan H_d.

Les deux définitions qui suivent assurent que tous les sommets de A_d^* sont définis dans $\mathbb{Z}^d$ en multipliant pour chaque dimension par $d+1$. Cette opération facilite la compréhension des propriétés du treillis permutoèdrique.

Définition 2.5.6 ([184]).

$$A_d^* := \{T(\boldsymbol{x}) | \boldsymbol{x} \in (d+1)\mathbb{Z}^{d+1}\},$$

avec T la fonction projection de $\mathbb{R}^{d+1}$ dans H_d.

Définition 2.5.7 ([184]).

$$A_d^* := \bigcup_{k=0}^{d} \{\boldsymbol{x} \in H_d \mid \boldsymbol{x} \text{ est un sommet de reste k }\},$$

$\boldsymbol{x} \in H_d$ est un sommet de reste k pour un $k \in \{0, ..., d\}$ si est seulement si toutes les coordonnées sont congruentes pour k modulo (d+1).

Information structurelle et manipulation du treillis permutoedrique

La façon la plus simple de visualiser et de construire A_d^* consiste à projeter une grille régulière définie par $(d+1)\mathbb{Z}^{d+1}$ le long du vecteur $\overrightarrow{\mathbf{1}}$ sur l'hyperplan H_d. Les cellules de Voronoï de A_d^* maillent H_d en permutoèdres. Les permutoèdres ont un polytope pour chaque dimension non nulle (e.g. pour les trois premières dimensions : un segment, un hexagone Figure 2.4 et un octaèdre tronqué tous uniformes). Un permutoèdre d'ordre d est un polytope de dimension $(d-1)$ inscrit dans un espace de dimension d. Le terme permutoèdre réfère au fait que les coordonnées de ses sommets peuvent être obtenues en permutant les coordonnées de n'importe quel sommet unique de la structure. Le graphe dual de A_d^* composé des cellules de Delaunay maille H_d en simplexe uniforme de dimension d.

Théorème 2.5.1 ([184]). *Les cellules de Delaunay de A_d^* sont des simplexes uniformes de dimension d et sont reliées au simplexe canonique par permutations et translations. Il est d'usage de définir le simplexe canonique de dimension d avec les sommets $\boldsymbol{s}_0, ..., \boldsymbol{s}_d$ dont les coordonnées du k-ième sommet sont calculées comme suit :*

$$\boldsymbol{s}_k = [\underbrace{k, ..., k}_{d+1-k}, \underbrace{k-(d+1), ..., k-(d+1)}_{k}].$$

En inspectant plus en détail les coordonnées des sommets du simplexe canonique, il est possible de voir que les deux inégalités suivantes sont respectées : $s_0 \geqslant ... \geqslant s_d$ et $s_0 - s_d \leqslant d+1$. Un point appartient au simplexe canonique si et seulement si ces deux inégalités sont respectées.

Cette propriété peut être étendue à chaque simplexe identifié par le sommet $\boldsymbol{s}_0$ en considérant toutes les permutations ρ possibles des coordonnées du simplexe canonique. Chaque permutation induit un nouvel ordonnancement des sommets $x_{\rho(0)} \geqslant ... \geqslant x_{\rho(d)}$, une nouvelle inégalité $x_{\rho(0)} - x_{\rho(d)} \leqslant d+1$ et donc un nouveau simplexe associé au permutoèdre d'origine.

Comme énoncé dans le Théorème 2.5.1, un treillis est invariant par translation, ainsi n'importe quel point $\boldsymbol{x} \in H_d$ appartient à un unique permutoèdre identifié par le sommet de reste 0 le plus proche. L'ordonnancement des coordonnées résultant de la différence $\boldsymbol{x} - \boldsymbol{l}_0$ permet de trouver les autres sommets de reste $k \in \{1, .., d\}$ et donc le simplexe englobant. En pratique cela revient à calculer d'abord le sommet $\boldsymbol{l}_0$ en arrondissant les coordonnées de $\boldsymbol{x}$ au multiple $d+1$ le plus proche. Ensuite chaque autre sommet $\boldsymbol{l}_k$ de reste k est donné par la formule, $\rho^{-1}(\boldsymbol{s}_k) + \boldsymbol{l}_0$ où $\boldsymbol{s}_k$ est un sommet de reste k du simplexe canonique et ρ la permutation qui permet de retrouver les coordonnées de $\boldsymbol{x} - \boldsymbol{l}_0$ triées par ordre décroissant. Trouver le simplexe englobant est utile pour les opérations **Splat** et **Slice** utilisées dans le cas du filtre bilatéral et également dans notre méthode d'attribution de score.

L'opération **Splat** accumule les valeurs du signal d'entrée $\boldsymbol{x} \in H_d$ aux sommets du simplexe englobant $\boldsymbol{x}$ avec une pondération appropriée par sommet. Il est d'usage dans le cas de l'utilisation de simplexe d'utiliser une pondération barycentrique pour interpoler le signal d'entrée (donné par $\boldsymbol{x}$) sur les sommets du simplexe englobant. Les coordonnées barycentriques $\boldsymbol{b}$ d'un point $\boldsymbol{x} \in H_d$ peuvent être calculées en associant $\boldsymbol{x}$ à un nouveau point $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{l}_0$ appartenant au simplexe canonique :

$$b_k = \begin{cases} \frac{\rho(\boldsymbol{y}_{d-k}) - \rho(\boldsymbol{y})_{d+1-k}}{d+1}, & k \neq 0 \\ 1 - \frac{\rho(\boldsymbol{y})_0 - \rho(\boldsymbol{y})_d}{d+1}, & k = 0, \end{cases} \tag{2.1}$$

avec $\rho(\boldsymbol{y})_k$ la permutation qui permet de trier les coordonnées de $\boldsymbol{y}$ par ordre décroissant. La valeur b_k est associée au sommet de reste k du simplexe englobant $\boldsymbol{x}$. La Figure 2.5 illustre les pondérations accumulées pour un sommet avec une valeur d'intensité de signal égale à 1.

L'opération de **Blur** implique de calculer tous les voisins connexes d'un sommet $\boldsymbol{l}$ donné

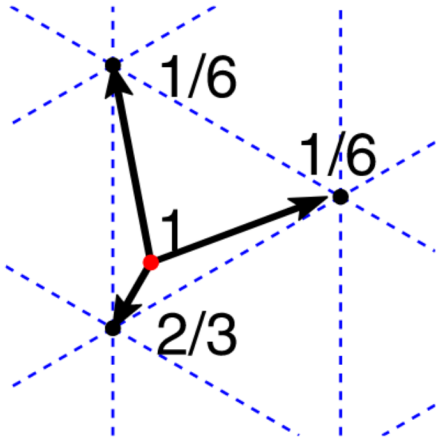


FIGURE 2.5 – Image extraite de [184]. Le signal d'entrée est représenté par le sommet rouge et possède une valeur de signal de 1. L'opération **Splat** accumule les valeurs de signal pondérées par les coordonnées barycentriques (1/6, 1/6 et 2/3) sur chaque sommet respectif du simplexe (triangle) englobant.

pour diffuser les valeurs accumulées sur chaque sommet du treillis. Les coordonnées des sommets voisins de $\boldsymbol{l}$ sont simplement séparées par un vecteur de la forme :

$$\pm[\underbrace{-1,..,-1}_{k},d,\underbrace{-1,...,-1}_{d-k}],$$

avec $k \in \{0,...,d\}$. En pratique, les sommets d'un treillis permutoèdrique sont organisés dans une table de hachage. La clé de chaque sommet est déterminée à partir de leurs coordonnées et la valeur contient l'information accumulée par sommet.

Soit

$$\begin{aligned}(E\boldsymbol{x})_d &= -\alpha_d x_{d-1} \\ (E\boldsymbol{x})_i &= -\alpha_i x_{i-1} + \frac{x_i}{\alpha_{i+1}} + (E\boldsymbol{x})_{i+1} \\ (E\boldsymbol{x})_0 &= \frac{x_i}{\alpha_1} + (E\boldsymbol{x})_1 \\ \alpha_i &= \sqrt{\frac{i}{i+1}},\end{aligned} \tag{2.2}$$

la fonction de récurrence proposée par Adams et al. [177] qui transforme un point quel-

conque $\boldsymbol{x} \in \mathbb{R}^d$ dans l'hyperplan $H_d \subset \mathbb{R}^{d+1}$ dans lequel A_d^* est défini, avec

$$E = \begin{bmatrix} 1 & 1 & \dots & 1 \\ -1 & 1 & \dots & 1 \\ 0 & -2 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -d \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{6}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\frac{1}{\sqrt{d(d+1)}} \end{bmatrix}.$$

Cette fonction est utile pour traiter tous les points $x \in \mathbb{R}^d$ du signal d'entrée. Dans notre cas d'usage, cette fonction transforme le centre géométrique de chaque voxel *'frontières'*/*'inconnus'* défini dans le repère monde W dans l'hyperplan H_d. Pour ensuite appliquer les opérations de **Splat** et **Slice** au voxel *'inconnus'*/*'frontières'*.

Le treillis permutoèdrique possède des avantages notables pour les opérations **Splat**, **Blur** et **Slice** telles que l'exécution d'opérations utiles et rapides visant à trouver le simplexe qui englobe une position quelconque, la découverte des sommets dans le voisinage connexe de n'importe quel sommet du treillis, ou bien l'utilisation de coordonnées barycentriques.

Processus et paramètres

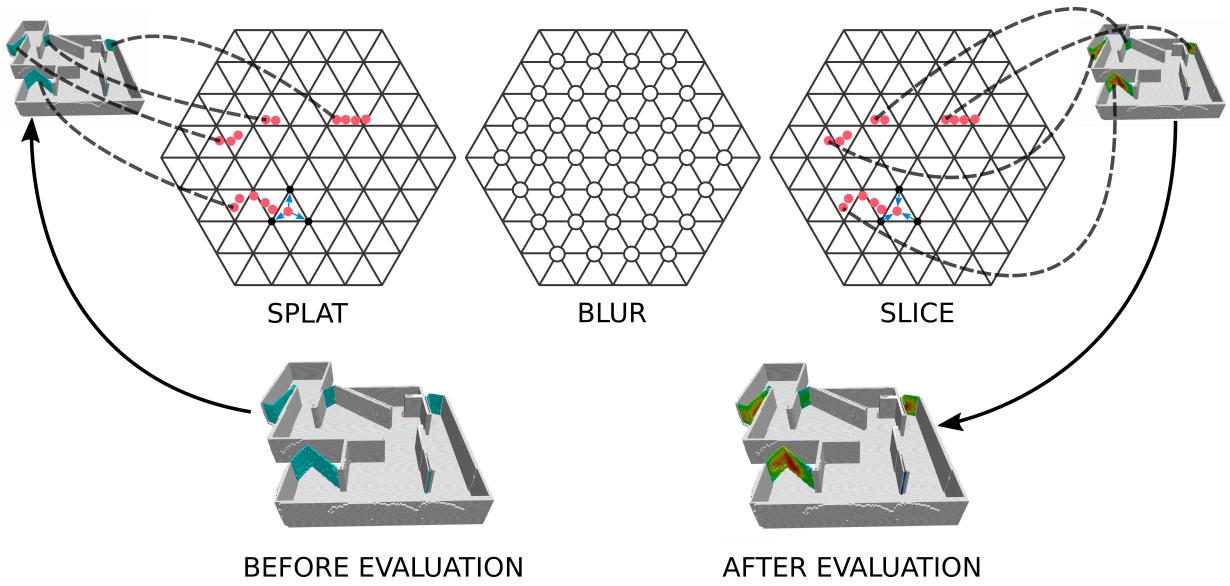


FIGURE 2.6 – Illustration du processus d'attribution de score d'intérêt pour les voxels *'frontières'*. Pour commencer, les voxels *'inconnus'* sont accumulés sur les sommets des simplexes englobants du treillis permutoèdrique (**Splat**). Chaque sommet diffuse la valeur accumulée vers les sommets voisins (**Blur**). Les valeurs accumulées sont restituées aux voxels *'frontières'* (**Slice**) produisant leur score d'intérêt.

Nous proposons un processus qui utilise le treillis permutoèdrique associé aux opérations **Splat**, **Blur** et **Slice** pour calculer et attribuer le score d'intérêt de chaque voxel *'frontière'* $f \in \mathcal{F} \subset \mathcal{V}$. Nous utilisons le centre géométrique des voxels *'frontières'*/*'inconnus'* comme (signal d')entrée du processus. Nous définissons $\boldsymbol{p}_v \in \mathbb{R}^3$ comme étant le centre géométrique du voxel $v \in \mathcal{V}_{inconnu} \subset \mathcal{V}$ et rappelons que $\mathcal{F} \subset \mathcal{V}_{inconnu}$. Nous décrivons maintenant les trois opérations **Splat**, **Blur** et **Slice** appliquées successivement pour attribuer le score d'intérêt à tout voxel *'frontière'*:

— **Splat**: la première opération du processus consiste à **Splat** l'ensemble de voxel $\mathcal{V}_{inconnu}$ sur les sommets du treillis permutoèdrique A_3^* défini dans l'hyperplan $H_3 \subset \mathbb{R}^4$. La première étape de cette opération transforme les centres géométriques des voxels *'inconnus'* $\boldsymbol{p}_v$ dans $H_3 \subset \mathbb{R}^4$ via l'équation de récurrence 2.2. Une fois $\boldsymbol{p}_v$ intégré dans l'hyperplan, nous identifions le simplexe englobant et calculons ses coordonnées barycentriques $\boldsymbol{b}$. Enfin nous accumulons chaque composante $\boldsymbol{b}_k$ des coordonnées barycentriques calculées au sommet k du simplex correspondant. La Figure 2.5 illustre ce processus.

— **Blur**: la seconde opération diffuse la valeur accumulée sur chaque sommet du treillis via un filtre (séparable) gaussien standard. Le filtre est appliqué à tous les sommets du treillis permutoédrique dont la valeur est strictement positive, produisant une combinaison pondérée avec les sommets voisins. Cette étape lisse les valeurs accumulées sur les sommets du treillis.

— **Slice**: la dernière opération est un calcul de somme pondérée impliquant le score d'intérêt pour chaque échantillon de $\mathcal{F}$. Le score d'un voxel f est la somme des valeurs accumulées sur les sommets du simplexe englobant f où chaque valeur est pondérée par les coordonnées barycentriques de f.

Il est possible de contrôler indirectement le volume englobé par les simplexes de A_3^* via Λ une matrice d'échelle diagonale 3×3. Pour réaliser ce contrôle, le centre géométrique de chaque voxel est multiplié par Λ avant l'opération **Splat**. Dans nos expériences, ce paramètre influence peu l'ensemble de notre méthode s'il est paramétré raisonnablement (e.g. $\Lambda = I_3$ est un compromis raisonnable). Avec ce paramètre, un voxel *'inconnu'* influence le score d'intérêt d'un voxel *'frontière'* si la distance (en mètres) qui les sépare est inférieure à trois fois la valeur choisie pour la diagonale (dans nos expériences 3 m). Concernant Λ des valeurs choisies trop faibles ou trop élevées sur la diagonale peuvent dégrader l'échantillonnage d'objectif d'exploration. D'une part, des valeurs très faibles reviendraient à agréger très peu de voxels par simplexe ce qui produirait des valeurs de densités homogènes entre les voxels *'frontières'*. Des valeurs de densités homogènes ne permettent pas d'extraire des voxels candidats correctement. D'autre part, des valeurs trop élevées agrégeraient des voxels trop éloignés. La notion de "proximité" entre les voxels agrégés dans le même simplexe serait alors perdue. Les valeurs de densités produites n'auraient pas réellement d'utilité et les objectifs extraits ne mèneraient pas forcément vers des voxels *'frontières'* à proximité de grandes régions *'inconnues'*.

Le processus complet est schématisé par la Figure 2.6. La Figure 2.7 montre un groupement de voxels *'frontières'* où chaque voxel est coloré en fonction de son score d'intérêt. Le

résultat montre que les frontières situées au centre du groupe ont une valeur plus élevée. Une grande valeur signifie une grande quantité de voxels *'inconnus'* à proximité et par conséquent un volume inconnu élevé.

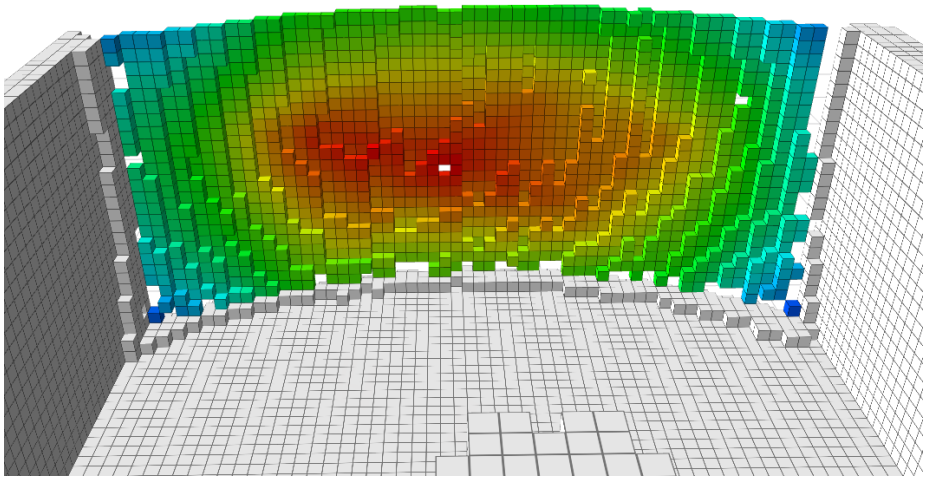


FIGURE 2.7 – Illustration d'un ensemble de frontières colorées de bleu foncé (score d'intérêt faible) à rouge (score d'intérêt élevé).

2.5.3 Extraction de frontières candidates prometteuses

Le résultat issu de l'étape précédente est un ensemble de scores par voxel *'frontières'*. Cette valeur est utilisée pour extraire une liste d'objectifs G. Cette liste d'objectifs est constituée de voxels *'frontières'* candidats à prochaine destination du drone. La phase d'*extraction de candidats* vise à privilégier les *'frontières'* présentant une valeur de score élevée tout en gardant une répartition uniforme des candidats dans l'espace.

L'opération d'extraction de voxels *'frontières'* candidats prend en entrée les positions des *'frontières'* et leur score d'intérêt dans l'ensemble de *'frontières'*, triés par ordre décroissant. Les candidats sont choisis en itérant sur l'ensemble ordonné des *'frontières'* et en ne retenant que les *'frontières'* suffisamment éloignées (jusqu'à un seuil de rayon r_{thresh}) des positions précédemment retenues. L'algorithme détaillé est fourni dans Algorithme 2.

L'ensemble des objectifs est limité en retenant les N_{scores} *'frontières'*, entités les mieux notées parmi les candidats extraits, augmenté de $N_{uniform}$ candidats extraits de manière aléatoire dans l'ensemble des objectifs restants. Cette limitation évite que le calcul d'une trajectoire d'exploration consomme trop de temps et se traduise par une perte d'efficacité pour l'exploration. Notre critère de sélection aléatoire assure une distribution égale des objectifs et profite à l'exploration locale et globale. D'autre part, le critère qui favorise les *'frontières'* les mieux notées garantit que les zones les plus prometteuses ne sont pas laissées de côté, quelle que soit leur distribution spatiale. D'après nos expériences, cela

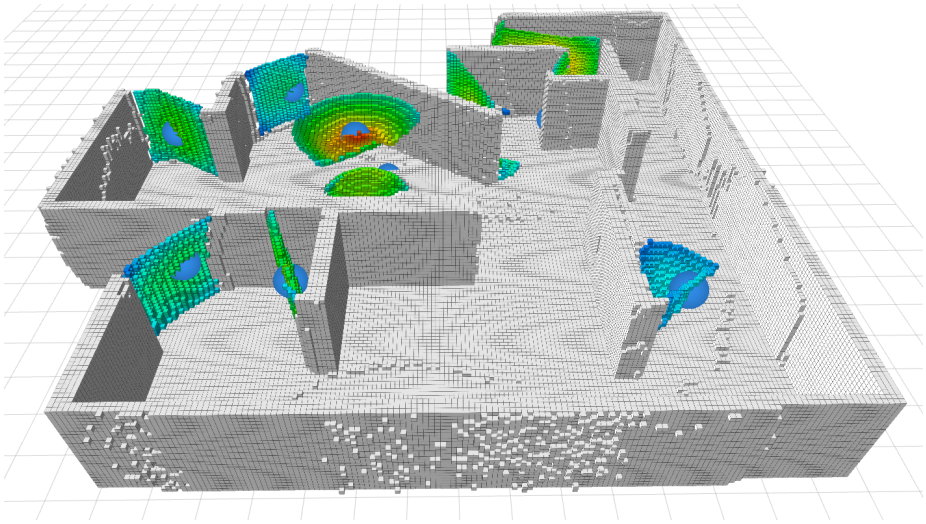


FIGURE 2.8 – Carte d'occupation : les voxels *'occupés'* sont en gris, les voxels *'frontières'* sont colorés en fonction du score d'intérêt et les sphères (bleues) représentent les objectifs extraits par l'Algorithme.2

donne un bien meilleur compromis entre exploration locale et globale que ce que donne un échantillonnage aléatoire pur [166], ou une exploration locale pure [118]. Dans nos expériences, le nombre d'objectifs est limité à $N_{scores} = 10$ et $N_{uniform} = 10$. La Figure 2.8 illustre avec les sphères bleues les objectifs extraits lors d'une expérience d'exploration autonome.

Algorithme 2 : Extraction des frontières candidates

Entrées : $\mathcal{P}$: liste de paires (position et score) $(\boldsymbol{p_i}, s_i)$ triées par ordre de score décroissant, r_{thresh} : seuil du rayon
Sorties : G : ensemble d'objectifs

```
pour chaque position p_i ∈ P faire
    discarded ← Faux;
    pour chaque p_j ∈ G faire
        si ||p_i − p_j|| < r_thresh alors
            discarded ← Vrai;
            aller à 9;
        fin
    fin
    si non discarded alors
        G ← (p_i, s_i)
    fin
fin
```

2.5.4 Échantillonnage de trajectoires d'exploration

Les frontières sélectionnées lors de l'étape précédente sont situées dans des zones menant vers des régions à explorer par le drone. Cette étape de l'algorithme consiste donc à construire des chemins sans collision, partant de la position courante du drone vers chaque voxel *'frontière'* de G. Il est important de noter qu'un voxel *'frontière'* n'est pas forcément atteignable puisque un voxel *'frontière'* est aussi étiqueté comme *'inconnu'*. Les voxels inconnus sont assimilés à des voxels occupés lors de la navigation et le robot doit naviguer à une distance supérieure à son envergure (dans nos expériences réelles et simulées : 0.6 m ou 1.0 m) de ces voxels. Par conséquent, la trajectoire doit s'arrêter avant l'objectif choisi, pour éviter toute collision due à des obstacles encore non observés.

Pour rappel, habituellement, un algorithme de plus court chemin a pour objectif de trouver un chemin dit optimal, en termes de distance parcourue, entre la position de départ et la position d'arrivée. Dans notre cas, l'objectif est un peu différent. Il consiste à trouver un chemin valide éventuellement optimal validant si possible les deux conditions énumérées ci-dessous :

1. La distance entre un voxel *'frontière'* de G et la position finale de la trajectoire ne doit pas dépasser la moitié de la distance maximale de vision du capteur. Cette condition assure que le drone est suffisamment proche du voxel en question pour correctement observer la zone inconnue.
2. Aucun obstacle ne doit se trouver entre la position finale et le voxel *'frontière'*. Cette condition assure que le voxel est visible.

Ces conditions permettent de guider l'algorithme RRT^* que nous utilisons et éventuellement arrêter la recherche si elles sont remplies. Nous avons évoqué le fonctionnement de l'algorithme RRT^* dans le chapitre précédent. Dans cette sous-section, nous fournissons plus de détails concernant son utilisation et son fonctionnement.

Pour mieux comprendre le fonctionnement de l'algorithme RRT^*, nous présentons dans cette partie la procédure de l'algorithme RRT et décrivons ensuite l'apport de l'algorithme RRT^* par rapport à l'algorithme RRT.

Procédure de l'algorithme RRT: Cette procédure construit un arbre dans l'espace d'exploration et permet de trouver un chemin vers une position d'arrivée à proximité d'un objectif donné.

1. La recherche est initialisée à la position courante $\mathbf{p}^W_{start}$ du drone. $\mathbf{p}^W_{start}$ est la racine de l'arbre.
2. Une position $\mathbf{p}^W_{rand}$ est aléatoirement échantillonnée dans l'espace de travail (la carte d'occupation).
3. On extrait le nœud $\mathbf{p}^W_{near}$ le plus proche de $\mathbf{p}^W_{rand}$, celui qui minimise la distance $L2$ entre $\mathbf{p}^W_{near}$ et $\mathbf{p}^W_{rand}$.
4. On calcule $\mathbf{p}^W_{new}$ une position située sur le segment défini par $\mathbf{p}^W_{rand}$ et $\mathbf{p}^W_{near}$. $\mathbf{p}^W_{new}$ minimise la distance $L2$ avec $\mathbf{p}^W_{rand}$ tout en maintenant une distance maximale avec $\mathbf{p}^W_{near}$.

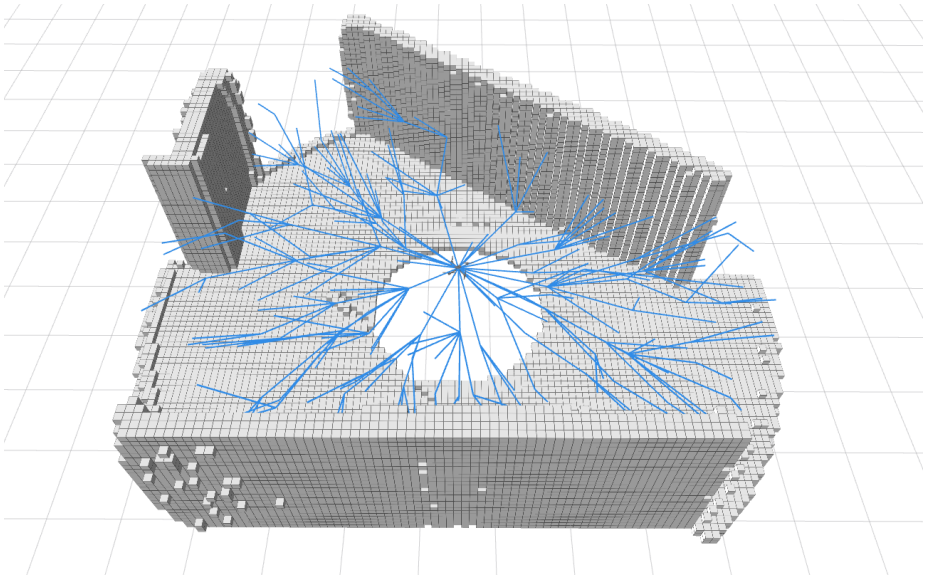


FIGURE 2.9 – Cette Figure illustre un arbre construit dans l'espace libre par l'algorithme RRT^*. Chaque segment (bleu) correspond à une arête et peut servir de chemin pour mener vers un objectif donné. La racine de l'arbre se trouve au centre, là où le drone quadrirotor est positionné.

5. On vérifie les collisions le long du segment défini par $\mathbf{p}_{new}^W$ et $\mathbf{p}_{near}^W$. La vérification des collisions assure que le segment n'est pas trop proche d'un obstacle. Si $\mathbf{p}_{new}^W$ est validé, on ajoute $\mathbf{p}_{new}^W$ à l'arbre en le liant à $\mathbf{p}_{near}^W$.
6. La recherche se termine et retourne un chemin quand une période de temps spécifiée et dépassée (dans nos expériences 5 ms par requête de chemin) ou bien que les deux conditions énumérées plus tôt sont validées. Si la limite de temps est atteinte, le chemin extrait de l'arbre correspond à celui dont la feuille (position 3D) minimise la distance quadritique avec l'objectif cible. La période limite de temps permet de garder une exploration efficace. Notons que si toutes les trajectoires extraites pour l'ensemble des objectifs ne permettent pas d'observer de nouveau voxels inconnus, la période limite de temps est étendue automatiquement.

L'algorithme RRT^* est une modification de l'algorithme RRT qui assure la convergence vers une solution optimale si le temps le permet. En pratique, l'algorithme RRT^* restitue des chemins plus courts et plus directs que l'algorithme RRT. L'idée sous-jacente à cette modification est de retenir la distance parcourue depuis la racine sur chaque nœud. Après l'ajout d'un nouveau nœud, son voisinage est examiné et le parent du nouveau nœud est modifié si cela permet de réduire son coût. La Figure 2.9 illustre l'arbre déployé dans les zones libres de la carte d'occupation pour une requête de chemin vers un objectif donné.

Veuillez noter qu'en utilisant une limite de temps de recherche, les solutions restituées par l'algorithme RRT^* sont parfois sous optimales. De plus les deux conditions énumérées pour valider une position d'arrivée ne sont pas toujours réalisables. Néanmoins, les solutions

restituées restent utiles. Les solutions renvoyées par l'algorithme RRT^* sont de deux types :

— Solution exacte : la solution est un chemin menant vers une position d'arrivée qui respecte les deux conditions énumérées ci-dessus.

— Solution approximative : la solution est un chemin composé d'une séquence de nœuds menant à la position (une feuille de l'arbre) qui minimise la distance $L2$ avec la position de l'objectif. Dans ce cas, la solution ne respecte pas au moins une des deux conditions énumérées ci-dessus.

Nous considérons toutes les solutions de la même manière pour trouver la meilleure trajectoire d'exploration, même si elles sont issues d'une solution approximative. Le chemin restitué par l'algorithme RRT^* est simplifié par l'utilisation d'une routine de simplification implémentée dans la bibliothèque OMPL [74]. Cette routine réduit la longueur du chemin.

Pour résumer, la procédure complète qui permet de trouver un chemin allant de la position initiale jusqu'à l'un des objectifs de G se compose d'une requête de recherche de chemin avec l'algorithme RRT^* et d'une routine de simplification du chemin restitué. Cette procédure est effectuée pour chaque objectif appartenant à G.

2.5.5 Estimation du volume inconnu

À l'issue de l'algorithme RRT^*, le système dispose d'un ensemble de trajectoires sans collision pour le drone quadrirotor. Chaque trajectoire est composée de segments linéaires qui peuvent être définis par un ensemble de points de contrôle $P_i^W = [\boldsymbol{p}_0^W, ..., \boldsymbol{p}_f^W]$. Dans cette sous-section, nous décrivons comment choisir la trajectoire qui priorise l'exploration en estimant le gain d'une trajectoire menant à une position finale $\boldsymbol{p}_f^W \in P_i^W$.

Dans un premier temps (i.), l'objectif est d'estimer le volume inconnu inclus dans le champ de vision du capteur à chaque position $\boldsymbol{p}_f^W$, compte tenu de deux contraintes :

1. Le drone quadrirotor ne peut effectuer que des rotations autour de l'axe vertical (axe de lacet).
2. La caméra montée sur le drone est orientée de manière à observer horizontalement vers l'avant du drone.

Dans un second temps (ii.), nous optimisons l'angle de lacet du drone de sorte à maximiser le volume inconnu observé à une position donnée. Ensuite (iii.), nous estimons le temps de trajet du drone quadrirotor compte tenu des contraintes de vitesse et d'accélération, pour chaque trajectoire, ceci dans le but de calculer le gain associé à chacune d'entre elles. Enfin (iv.), la trajectoire sélectionnée est traitée pour être exécutée par le drone.

i. Estimation du volume inconnu

Le volume inconnu est estimé à l'aide d'un balayage horizontal sur 360 degrés en considérant un champ de vision vertical fixe (spécifique à la caméra). Ce balayage s'appuie sur la technique du lancer de rayon où les rayons sont lancés à intervalles de pas réguliers. Le

pas est paramétré par deux valeurs d'angle, vertical (α) et horizontal (β). Chaque rayon est délimité par la distance maximale d_{max} de vision de la caméra. Le volume couvert par un rayon est estimé par un modèle de pyramide tronquée. Le sommet tronqué de la pyramide commence dès que le rayon traverse un voxel étiqueté *'inconnu'*. Nous appelons d_a la distance entre la caméra et le voxel inconnu intersecté. Une fois d_a initialisée, nous appelons d_b la distance de la base de la pyramide jusqu'au capteur. d_b est calculée dès lors que le rayon traverse un voxel *'libre'*, *'occupé'* ou que le rayon atteint d_{max} (dans ce dernier cas $d_b = d_{max}$).

Le volume d'une pyramide tronquée compte tenu des paramètres définis ci-dessus, d_a et d_b calculés par lancer de rayon et, α et β des constantes arbitraires égales a 1° dans nos expériences, peut être défini comme la sommes des aires des plans le long du rayon. L'aire d'un plan centré sur le rayon situé à la distance r est $(2 * \tan(\frac{\alpha}{2}) * r)(2\tan(\frac{\beta}{2}) * r)$ où $2*\tan(\frac{\alpha}{2})$ et $2*\tan(\frac{\beta}{2})$ correspondent à la longueur des côtés du plan. L'intégrale suivante, dont la résolution est triviale, en découle :

$$\begin{aligned} V(d_a, d_b, \alpha, \beta) &= \int_{d_a}^{d_b} (2 * \tan(\frac{\alpha}{2}) * r)(2\tan(\frac{\beta}{2}) * r)dr, \\ &= 4 * \tan(\frac{\alpha}{2})\tan(\frac{\beta}{2}) \int_{d_a}^{d_b} r^2 dr, \\ &= \frac{4}{3} * \tan(\frac{\alpha}{2})\tan(\frac{\beta}{2})(d_b^3 - d_a^3). \end{aligned} \tag{2.3}$$

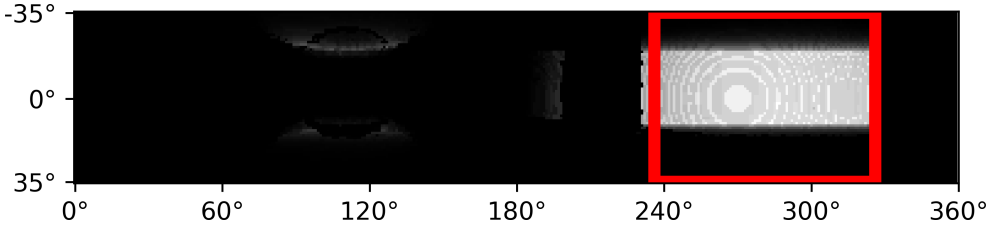
Le résultat du balayage 360° peut être illustré par une image en niveaux de gris dont l'intensité des pixels est fonction du volume inconnu traversé par un rayon. Ce rayon est défini par la droite passant par le centre optique du capteur et le pixel du plan image. La Figure 2.10(a) est un exemple de ce type d'image. L'intensité d'un pixel en niveau de gris correspond au volume traversé par le rayon correspondant normalisé entre 0 et 255 pour visualisation. La Figure 2.10(b) schématise la position du capteur par une pyramide renversée (colorée en vert), position à partir de laquelle est construite la Figure 2.10(a).

ii. Optimisation du lacet

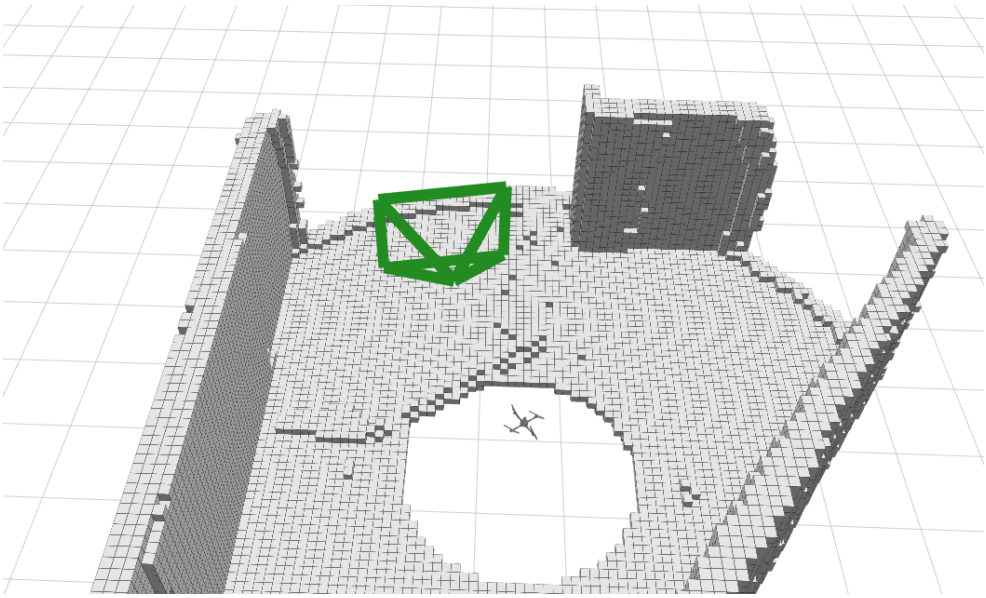
L'image I_k construite via un balayage 360° permet de calculer, la valeur d'angle de lacet ψ_k^W pour une position p_k^W donnée, maximisant la région inconnue que peut croiser le champ de vision du capteur. L'optimisation du lacet cherche donc ψ_k tel que :

$$\psi_k = \arg\max_{\psi} \ \mathcal{U}(\mu_k^W), \tag{2.4}$$

où $\mu_k^W = [p_k^W, \psi_k]$ et $\mathcal{U}(\mu_k^W)$ correspond au volume inconnu potentiellement observable à la pose μ_k^W. On résout l'équation (2.4) via une fenêtre glissante que l'on déplace sur l'ensemble de l'image I_k de façon à trouver l'angle qui maximise la somme des intensités des pixels. La



(a)



(b)

FIGURE 2.10 – Image résultant du balayage 360° Figure (a) capturé à la position schématisée par la caméra (en vert) Figure (b)

largeur de cette fenêtre correspond au champ de vision horizontal du capteur et sa hauteur correspond à au champ de vision vertical du capteur, c'est à dire la hauteur de I_k.

iii. Gain d'une trajectoire

L'optimisation du lacet aide à calculer le gain d'une trajectoire, qui correspond au rapport :

$$g(\mu_{f_i}, P_i) = \frac{\mathcal{U}(\mu_{f_i})}{T(P_i)}, \tag{2.5}$$

où μ_{f_i} est la pose finale de la trajectoire P_i et $T(P_i)$ le temps que le quadrirotor met pour exécuter la trajectoire complète. Le temps est estimé en tenant compte d'une vitesse maximale et fait l'hypothèse d'une accélération constante égale à l'accélération maximale du drone quadrirotor. La meilleure trajectoire P_m^W parmi un échantillon de trajectoires $\mathcal{P}$

peut être exprimée comme :

$$P_m^W = \underset{P_i^W \in \mathcal{P}}{\arg\max}\ g(\mu_{f_i}^W, P_i^W). \tag{2.6}$$

Finalement, l'équation (2.6) maximise explicitement l'exploration volumique en m^3/s parmi l'ensemble des trajectoires $\mathcal{P}$. La Figure 2.11 illustre une itération de notre méthode d'ex-

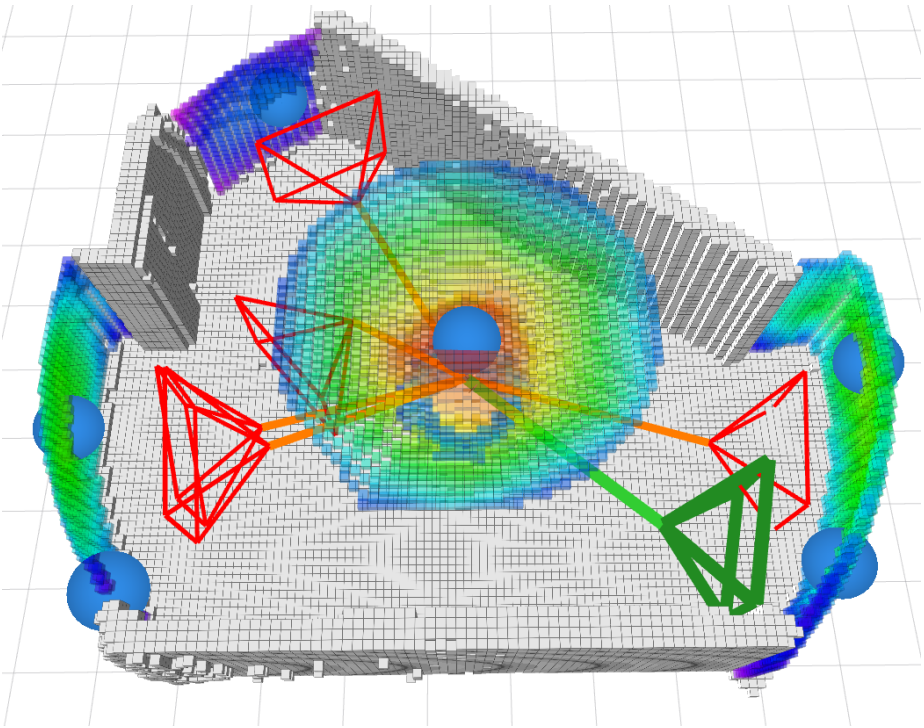


FIGURE 2.11 – Chaque trajectoire est évaluée selon la fonction de gain $g(.,.)$. Les schémas de couleur vert et rouge sont orientés dans la direction qui maximise le volume inconnu pouvant croiser le champ de vision du capteur. La trajectoire qui maximise $g(.,.)$ parmi toutes les autres trajectoires (orange) est colorée en vert.

ploration où l'ensemble des trajectoires échantillonnées est en orange, la meilleure trajectoire sélectionnée P_m^W est en vert et l'orientation du capteur pour les positions finales de chaque trajectoire est schématisée par une pyramide renversée.

Comme spécifié précédemment, si $\mathcal{U}(\mu_{f_i})$ égale 0 pour tous les chemins de $\mathcal{P}$, nous répétons l'étape d'échantillonnage de trajectoires d'exploration via l'algorithme RRT^* avec une période de temps plus élevée. Cela permet éventuellement de trouver des solutions exactes ou de nouvelles positions finales.

iv. Exécution de la trajectoire

Avant son exécution, la trajectoire P_m^W subit un ensemble de traitements préalables pour qu'elle soit réalisable par le drone quadrirotor. Le premier traitement consiste à optimiser le lacet pour chaque point de contrôle afin de maximiser le volume découvert tout au long de la trajectoire. Le second traitement transforme P_m^W en polynôme avec la méthode proposée par Richter et al. [82]. Ainsi, il est possible d'exprimer chaque déplacement (pose, vitesse et accélération linéaire et angulaire) le long de la trajectoire comme une fonction de t le temps compris entre 0 (le début de l'exécution de la trajectoire) et t_{max} (la durée d'exécutions de la trajectoire). De plus, leur méthode garantit le respect des contraintes de vitesse et d'accélération maximales, quelle que soit la valeur $t \in [0, t_{max}]$. Une fois traitée la trajectoire P_m^W est envoyée au système de contrôle et exécutée sur toute la durée t_{max}.

2.6 Conclusion

Dans ce chapitre, nous avons proposé une méthode d'exploration autonome sous le nom de SplatPlanner [3] calculant simultanément l'estimation de trajectoires d'exploration sans collision et une reconstruction voxélique de l'environnement via une grille régulière 3D. Pour réaliser cette tâche, nous avons présenté une méthode originale d'exploration utilisant le filtrage bilatéral et les opérations associées de **Splat**,**Blur** et **Slice** en s'appuyant sur la structure efficace qu'est le treillis permutoèdrique. Notre méthode attribue un score d'intérêt aux voxels situés aux frontières des régions précédemment visitées afin de définir de nouveaux objectifs grâce à un algorithme utilisant ces scores. Ces nouveaux objectifs guident l'échantillonnage de nouvelles trajectoires d'exploration. Pour optimiser l'exploration, nous avons proposé de sélectionner l'échantillon qui maximise le ratio entre le volume inconnu potentiellement observable et temps requis par le drone pour exécuter la trajectoire.

Nous verrons grâce aux multiples expériences, présentées dans le Chapitre 4, sur un jeu de données spécifiquement étudié dans un environnement de simulation contrôlé (FLYBO, Chapitre 3) que la méthode proposée offre de meilleures performances que les méthodes de l'état de l'art considérées quant à son efficacité et sa rapidité d'exploration.

Chapitre 3

Évaluation des systèmes d'exploration et de reconstruction autonomes : proposition d'un environnement de simulation (FLYBO)

3.1 Introduction

Les systèmes d'exploration et reconstruction autonomes actuels faisant usage d'un robot instrumenté d'une caméra de profondeur évaluent généralement leurs performances dans un environnement de simulation contrôlé [116-118, 121, 166]. Un environnement de simulation permet de s'assurer du bon fonctionnement du système, aide à minimiser les risques et les coûts opérationnels qu'une campagne d'exploration et de reconstruction 3D en circonstances réelles peut engendrer. En dépit des divers travaux scientifiques portant sur l'exploration autonome et de l'importance d'un environnement de simulation contrôlé, les systèmes d'exploration autonome sont très souvent évalués sur des jeux de données diverses et variées, parfois associées à des conditions expérimentales très différentes. Globalement,

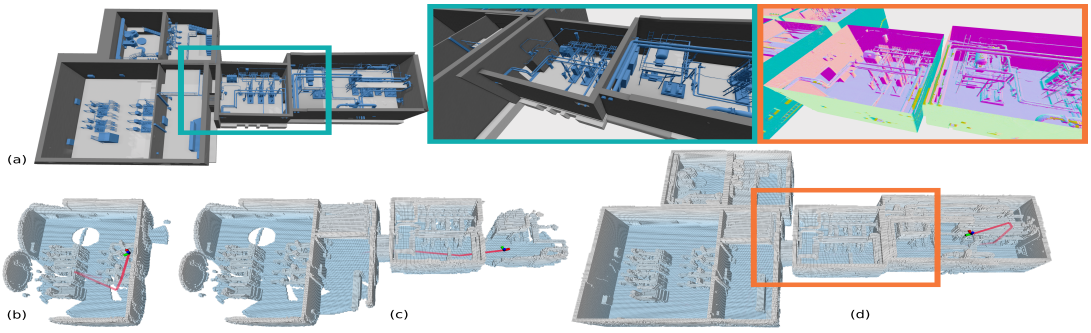


FIGURE 3.1 – Un drone volant équipé de capteurs d'odométrie et de profondeur explore de manière autonome un environnement synthétique complexe (a). La carte d'occupation est graduellement enrichie au cours des différentes étapes en planifiant des trajectoires d'exploration (b–d). Simultanément, les surfaces perçues sont également reconstruites en ligne (vues rapprochées). **FLYBO** fournit des scénarios synthétiques réalistes, des métriques et des routines communes à ce type de problème pour évaluer de tels systèmes.

il manque un environnement unifié permettant d'évaluer les systèmes robotisés d'exploration autonome. Ceci nous a motivé pour développer un environnement de simulation. Dans ce chapitre, nous présentons FLYBO [62], un environnement de simulation unifié, conçu pour tester et évaluer les méthodes d'exploration autonome. Plus particulièrement, dans cet environnement de simulation, nous considérons l'usage d'un drone quadrirotor équipé d'une caméra de profondeur et d'un système d'odométrie. Ce type d'équipement offre la capacité à ce type de robots volants d'explorer efficacement des environnements inconnus [3, 116, 118] tout en reconstruisant les surfaces [117] perçues lors de l'exploration. L'environnement de simulation proposé vise à évaluer précisément, via plusieurs métriques, les méthodes d'exploration autonome (e.g SplatPlanner [3] présenté dans le Chapitre 2). En l'occurrence, FLYBO intègre sept méthodes d'exploration et reconstruction autonome dont SpatPlanner [3].

Nous commençons ce chapitre par rappeler le fonctionnement des méthodes d'exploration autonome. Nous expliquons ce qui a motivé le développement d'un environnement de simulation unifié pour évaluer ces méthodes. Ensuite, nous présentons le jeu de données proposé dans FLYBO. Puis, nous présentons les composants de FLYBO, partagés par les méthodes d'exploration autonome et nécessaires à leur fonctionnement, suivi des différentes métriques d'évaluation et des méthodes d'exploration intégrées à FLYBO. Nous finissons sur une discussion présentant quelques pistes possibles d'extension du système de simulation et une conclusion générale de ce chapitre.

3.2 État de l'art

L'exploration autonome est étudiée depuis plus de 30 ans [1, 2], ce sujet a vu récemment sa popularité augmenter dans plusieurs domaines de recherche telle que la vision par ordinateur [119], l'infographie [120, 121, 185] et la robotique [69, 116]. Comme introduit dans les chapitres précédents, les méthodes d'exploration sont couramment catégorisées en trois stratégies : les méthodes basées sur les frontières [2, 69], celles basées sur l'échantillonnage [116, 117] et celles hybrides [3, 118].

En dépit de toutes ces recherches scientifiques, nous avons observé un manque de rigueur dans l'évaluation des méthodes d'exploration autonome induit par le manque d'un environnement de simulation unifié adapté. Développer un environnement de simulation est un processus long et couteux, nous pouvons en retrouver quelques-uns dans la littérature dédiés aux drones quadrirotors : AirSim [99] et Flightmare [186]. AirSim et Flightmare sont des simulateurs polyvalents développés en particulier pour entrainer des méthodes d'apprentissage profond et d'apprentissage par renforcement [95]. Leur polyvalence réside dans l'usage de moteurs de jeux très complexes et très réalistes (Unity [i] et Unreal Engine [ii]). Cependant, ce type de moteur de jeux induit un cout qui complique l'interaction avec des processus extérieurs au moteur de jeu et engendre souvent une surcharge de calcul inutile. Il n'est pas rare de trouver dans la littérature des travaux s'appuyant sur des outils et des simulateurs moins complexes dédiés aux drones quadrirotor tels que RotorS utilisant Gazebo [98]. Par exemple, Tezenas Du Montcel et al. [187] s'est servi de RotorS et Gazebo pour évaluer des algorithmes d'évitement d'obstacle. En ce sens, pour s'affranchir de cette complexité supplémentaire, nous nous servons aussi de RotorS et Gazebo dans FLYBO.

La tâche d'exploration autonome est étroitement liée à celle de la reconstruction 3D en ligne permettant d'évaluer la qualité de la géométrie produite par les systèmes d'exploration autonome lors d'une campagne d'exploration et d'inspection. Cette seconde tâche consiste à calculer un maillage de la scène explorée à partir des surfaces observées par le robot. Généralement, les travaux portant sur l'exploration et la reconstruction autonome avec un robot équipé d'une caméra de profondeur utilisent une reconstruction basée sur la TSDF de Curless et al. [66]. Le calcul de la TSDF et l'utilisation de l'algorithme dit du "Marching Cube" [67] permet de produire un maillage en temps réel avec une caméra de profondeur [117, 121, 188, 189]. Nous précisons que la tâche de reconstruction que nous évaluons dans ces travaux se concentre strictement sur la reconstruction en ligne (au cours de l'exploration) de surfaces à l'aide d'un capteur de profondeur et un capteur d'odométrie. Toutefois, il est intéressant de mentionner l'existence d'une littérature riche beaucoup plus spécialisée sur la reconstruction 3D en général [190, Chapitre 5], [191, 192] présentant des algorithmes plus lourds et plus complexes qui amélioreraient les résultats de reconstruction en post-traitement. Le maillage produit en temps réel permet de fournir de nouvelles informations sur la performance d'une méthode d'exploration autonome au-delà des critères d'exploration volumétriques traditionnels.

i. `https://unity.com/fr`
ii. `https://www.unrealengine.com/`

Évaluer les méthodes d'exploration autonome sur la qualité des surfaces géométriques qu'elles produisent nécessite des modèles d'environnements/scènes 3D (i) sans trou pour éviter des comportements de vol aberrants, passages au travers de murs notamment en intérieur et (ii) réalistes en termes de complexité structurelle, de taille (volume) et de géométrie. Les données dérivées de la CAO (Conception Assistée par Ordinateur) en particulier, constituent un standard par rapport à ces critères. Souvent conçues par l'homme à partir de données réelles (e.g. nuage de points acquis par un LiDAR), ces données sont généralement privilégiées quand elles sont disponibles dans un environnement contrôlé et simulé pour l'exploration autonome [3, 166]. Actuellement, les méthodes d'exploration autonome parues sont souvent sous-évaluées à cause de l'utilisation d'un jeu de données expérimental dérivé de la CAO trop petit (2 ou 3 environnements) [116, 118, 168]. Pour palier au manque de données CAO, d'autres méthodes d'exploration [120, 121] choisissent d'emprunter des ensembles de données dérivés de la reconstruction RGB-D. Les jeux de données dérivés de la reconstruction RGB-D [193-197] suffisent pour évaluer des méthodes d'exploration autonomes embarquées sur des robots non volants naviguant exclusivement en 2D [120, 121]. Ce n'est pas le cas pour les robots volants qui naviguent sur les trois dimensions et qui pourraient facilement atteindre et traverser des régions non scannées. De plus, il arrive que les données issues de méthodes de reconstruction RGB-D soient sujettes à des problèmes d'artefacts structurels liés au capteur utilisé et à la méthode de reconstruction. Ces artefacts engendrent des défauts dans le maillage produit voir des parties manquantes dans la scène reconstruite. Ces problèmes invalident ce type de données car ils peuvent biaiser les méthodes d'exploration autonome et leur évaluation.

C'est pourquoi, dans FLYBO, nous proposons un jeu de données public, réunissant onze environnements "étanches", sans artefact et hautement structurés, dérivés de la CAO. Ceci constitue le premier ensemble de données et le premier point de référence public réutilisable, consacré aux systèmes d'exploration autonomes, basés sur les drones quadrirotors.

En résumé, les travaux présentés dans ce chapitre apportent les contributions suivantes :

— *Système* : FLYBO, le premier environnement de simulation de référence unifié spécialement conçu pour l'évaluation des drones quadrirotors autonomes pour des tâches d'exploration volumétrique et de reconstruction de surface.

— *Jeu de données* : L'ensemble comprend 11 jeux de données de complexité structurelle et de taille croissante dérivés de la CAO, qui représentent des situations réalistes en intérieur et en extérieur.

— *Méthodes* : FLYBO comprend 7 des principales méthodes d'exploration autonome extraites de l'état de l'art, y compris des méthodes sans code public utilisant nos propres implémentations.

— *Outils* : Afin d'évaluer de manière juste et précise la partie exploration des systèmes existants, nous fournissons un cadre expérimental qui rassemble des outils partagés par toutes les méthodes. Notre système permet également de générer des résultats tant quantitatifs que qualitatifs au cours de l'exploration.

— L'ensemble des informations telles que le jeu de données, les méthodes intégrées et le code du projet sont disponibles en ligne sur la page web du projet `www.flybo.org`. Le

code de FLYBO ainsi qu'un guide d'installation est disponible sur GitHub : `https://github.com/anthonybrunel/FLYBO`.

3.3 Jeu de données

Le jeu de données inclus dans FLYBO réunit 11 modèles de scène 3D au format OBJ. Quelques modifications ont été appliquées sur chaque modèle pour garantir l'accessibilité de toutes les salles, l'absence de trou ou de passage indésirable. Chaque scène est délimitée par une boite englobante limitant le processus d'exploration dans un volume bien défini. Ces scènes modélisent des environnements ou des scénarios plausibles que les algorithmes d'exploration autonomes auraient à traiter dans un scénario réel d'exploration et d'inspection avec des niveaux de difficulté modérés à très difficiles. Nous listons et décrivons chaque environnement ci-dessous :

— **Maze.** Il s'agit de l'environnement proposant la structure la plus simple parmi l'ensemble des scènes proposées. Il permet d'isoler les comportements d'exploration sur une scène d'intérieure, sans objets, comprenant plusieurs embranchements et chemins possibles. La littérature utilise très souvent les labyrinthes pour évaluer les performances des méthodes d'exploration. Le Maze est constitué d'un seul niveau, avec un sol et des murs verticaux plats. Le volume couvert par la boite englobante de l'environnement est de $20m \times 20m \times 2.5m$.

— **Offices A.** Cet environnement présente une structure plus complexe que le Maze: il se compose de deux salles connectées par un couloir. L'une des salles est un espace de travail de type open-space meublé. Parmi les meubles, nous retrouvons des bureaux, des chaises, des ordinateurs, des casiers de rangement et une imprimante. La boite englobante de cet environnement est de $22m \times 28m \times 3.4m$.

— **Offices B.** Cet environnement est équipé de quelques bureaux, ordinateurs et divers équipements typiques d'une installation industrielle. Il présente plusieurs étages et par rapport aux deux environnements précédemment s'étend en hauteur avec une boite englobante de $8m \times 20m \times 12m$.

— **Offices C.** Il s'agit d'un environnement de type locaux de travail très grand (+500 m^2, $30m \times 16m \times 7.4m$) et avec une complexité structurelle plutôt réaliste. Ces locaux s'agencent sur deux étages et se composent de nombreuses salles (#23), parmi lesquelles on retrouve des bureaux, des salles de réunion, des cuisines/cafétérias, des vestiaires, des salles de douche. Chaque local est meublé. Le mobilier se compose de : tables, ordinateurs, chaises, casiers...

— **Facility A.** Il s'agit de locaux appartenant à une installation industrielle. Cet environnement est constitué de nombreux équipements industriels typiquement : des tuyaux, des canalisations, des chemins de câbles, des gaines, des réservoirs...

— **Facility B.** Cette installation industrielle est deux fois plus grande que Facility A et présente une structure similaire. Facility B est agencée en deux salles connectées par

une porte avec des équipements industriels tels que des canalisations, des réservoirs, des pompes...

— **Facility C.** Cette installation industrielle est très volumineuse et structurellement complexe. On y retrouve des équipements industriels similaires à ceux de FACILITY A et B. FACILITY C se compose de 5 salles connectées et s'étend sur une surface de 478 m^2. Cette installation modélise un environnement typique de ce qu'il est possible de retrouver dans une campagne d'exploration et d'inspection.

— **Warehouse.** Il s'agit d'un grand entrepôt industriel. Cet entrepôt contient un grand espace vide central entouré de nombreux équipements industriels.

— **Platform A et B.** Deux grands espaces extérieurs industriels où l'on trouve une quantité d'objets encombrants et d'équipements. Ces deux installations présentent de nombreuses zones complexes ou il est difficiles de manœuvrer pour un drone.

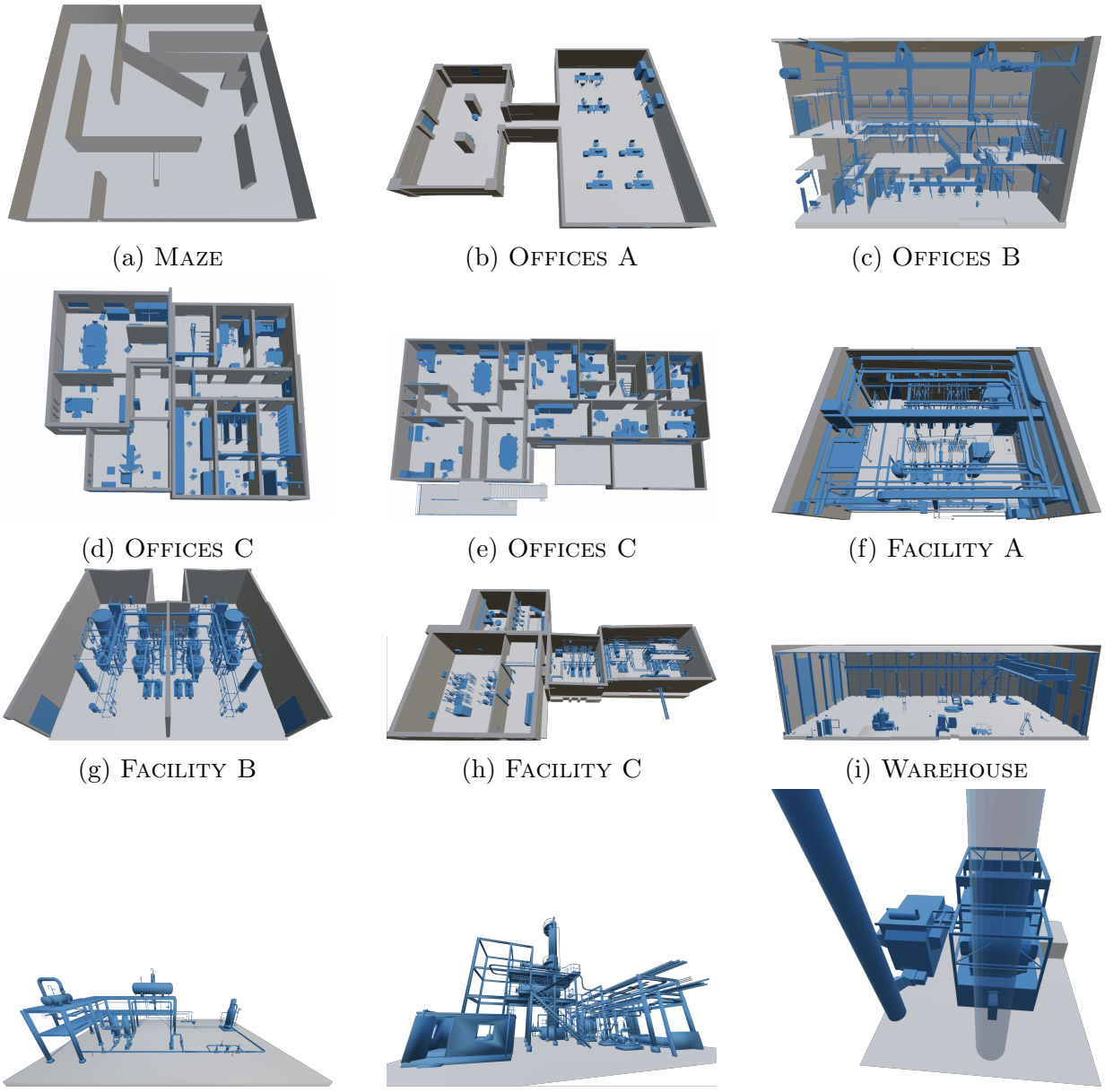
— **Powerplant.** Cet environnement d'extérieur est à l'origine une centrale à très grande échelle disponible comme modèle open-source et utilisée par plusieurs publications scientifiques indépendantes afin d'évaluer les performances des méthodes d'exploration autonome par drone volant [3, 69, 118, 166]. Le découpage exact du modèle 3D considéré pour l'évaluation n'était pas explicité et pourrait potentiellement différer entre les divers travaux l'ayant utilisé. Nous abordons ce problème en fournissant un découpage fixe à ce modèle avec les paramètres expérimentaux (e.g. échelle) correspondants. En termes de complexité d'exploration, POWERPLANT. est l'environnement le plus volumineux de notre jeu de données avec une boite englobante de $33m \times 31m \times 29m$.

Les données statistiques telles que : la surface en m^2, le volume en m^2, le découpage de la boite englobante, le nombre de salles, le nombre d'étages, le nombre de triangles des maillages, le type d'environnement (intérieur ou extérieur) la complexité structurelle et la taille de l'environnement de chaque scène sont synthétisés Table 3.1. En guise d'illustration, la Figure 3.2 montre une vue d'ensemble de chaque modèle 3D (environnement) dans lequel le robot explorateur évolue. Additionnellement, pour enrichir la visualisation, des vues immersives sont proposées Figure 3.3.

	Maze	Offices A	Offices B	Offices C	Facility A	Facility B
Surface (m^2)	400	470	120	534	115	323
Volume (m^3)	1000	1410	1443	1602	690	1450
Boite englobante (m)	$20 \times 20 \times 2.5$	$22 \times 28 \times 3.4$	$8 \times 20 \times 12$	$30 \times 16 \times 7.4$	$12 \times 14 \times 6$	$19 \times 17 \times 4.6$
#Salles	-	2	4	23	1	2
#Niveaux	1	1	4	2	1	1
#Nombre de triangles	130	6910	227k	857k	373k	1120k
Complexité structurelle	Faible	Faible	Intermédiaire	Élevée	Élevée	Élevée
Échelle	Petit	Intermédiaire	Intermédiaire	Grand	Intermédiaire	Petit
Intérieur / extérieur	Indoor	Indoor	Indoor	Indoor	Indoor	Indoor

	Facility C	Warehouse	Platform A	Platform B	Powerplant
Surface (m^2)	478	800	384	480	1023
Volume (m^3)	2868	9600	3456	12000	26598
Boite englobante (m)	$38 \times 22 \times 6$	$40 \times 20 \times 12$	$16 \times 24 \times 9$	$24 \times 20 \times 25$	$33 \times 31 \times 26$
#Salles	5	1	-	-	-
#Niveaux	2	2	3	3	1
#Nombre de triangles	1072k	793k	149k	470k	2000
Complexité structurelle	Elevée	Elevée	Intermédiaire	Elevée	Faible
Taille	Grand	Grand	Grand	Grand	Grand
Intérieur / extérieur	Intérieur	Intérieur	Extérieur	Extérieur	Extérieur

Table 3.1 – Détails et statistiques des 11 environnements qui composent le jeu de données de FLYBO.



(a) Maze (b) Offices A (c) Offices B

(d) Offices C (e) Offices C (f) Facility A

(g) Facility B (h) Facility C (i) Warehouse

(j) Platform A (k) Platform B (l) Powerplant

Figure 3.2 – Vue d'ensemble des environnements utilisée dans notre jeu de données

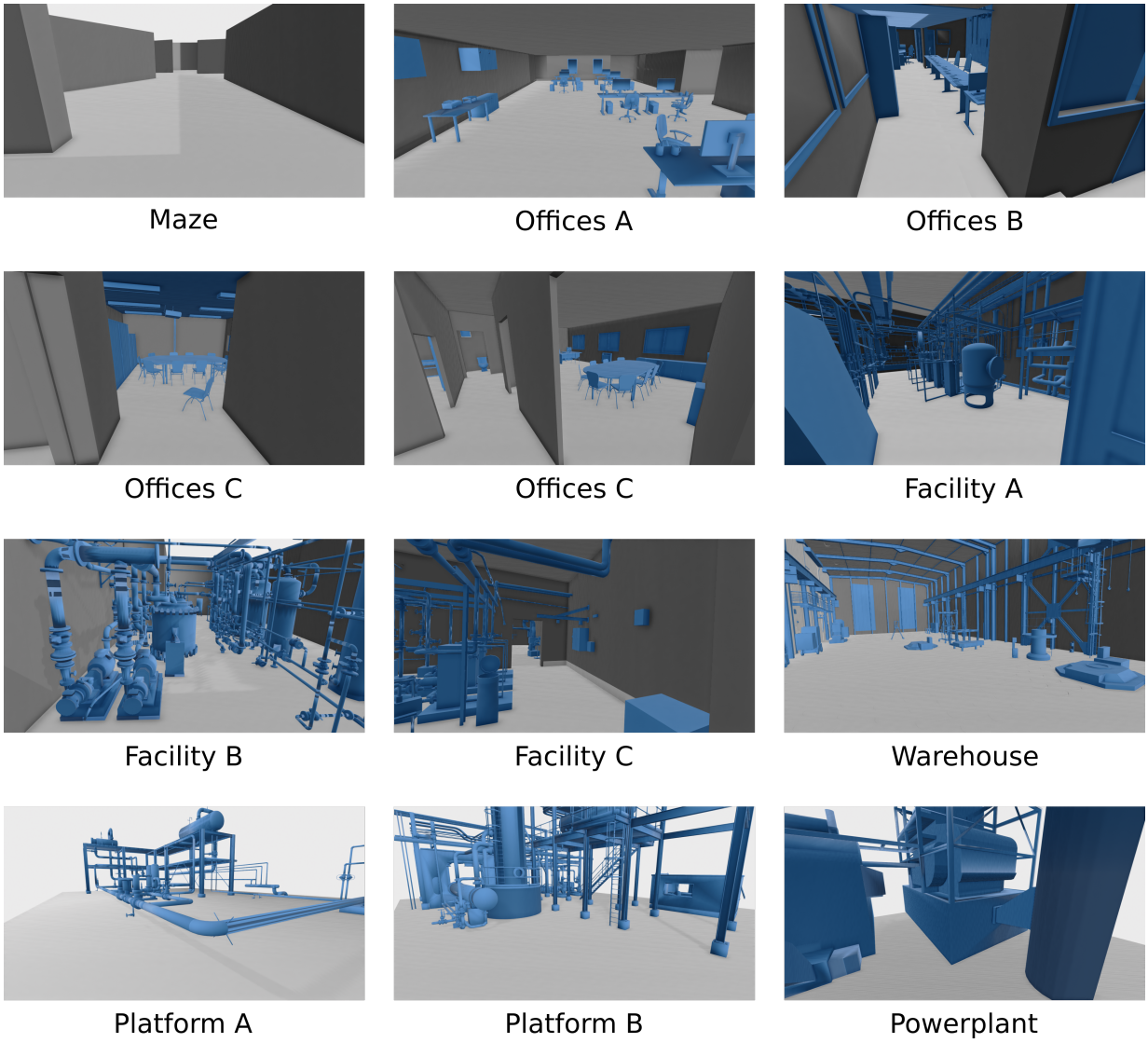


FIGURE 3.3 – Vues immersives des scènes constituant FLYBO.

3.4 Environnement de simulation

3.4.1 Vue d'ensemble

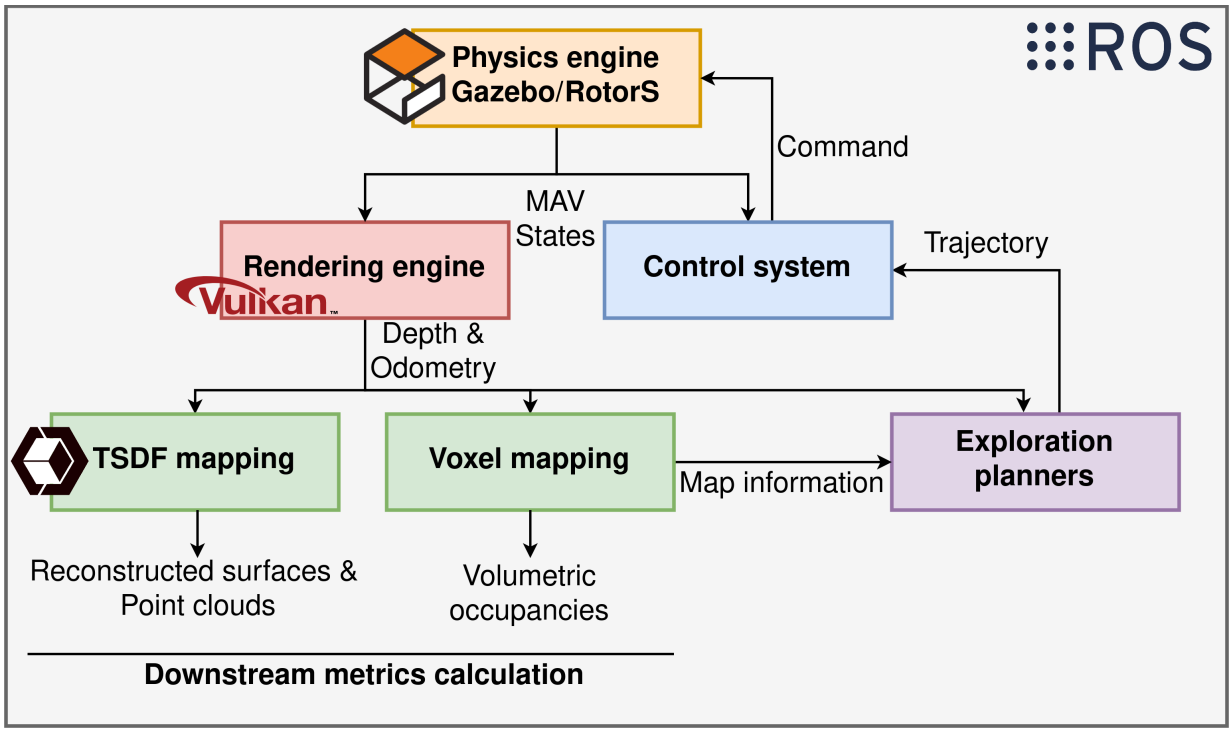


FIGURE 3.4 – Diagramme des composants de l'environnement de simulation FLYBO

Dans cette partie, nous proposons un environnement de simulation unifié (FLYBO) spécifiquement conçu pour l'exploration autonome par drone quadrirotor. FLYBO réunit les composants requis et communs à de nombreuses routines de simulation utilisées pour évaluer les méthodes d'exploration autonome telles que celles proposées dans [3, 69, 116-118]. Un composant peut être considéré comme un ou plusieurs processus exécutables. Chaque processus est nécessaire au fonctionnement de FLYBO et aux méthodes d'exploration autonome intégrée. La Figure 3.4 synthétise dans un diagramme les informations transmises/reçues ainsi que les composants. Ci-dessous, nous listons et décrivons brièvement tous ces composants :

— ***Robot Operating System*** (ROS) [198] : ROS est un intergiciel open-source qui regroupe plusieurs fonctionnalités facilitant le développement d'applications flexibles et modulables pour le robotique. Parmi ces fonctionnalités, on retrouve des collections d'outils et de bibliothèques qui facilitent notamment la création d'applications visant à simuler le comportement complexe des robots. Les processus ROS (aussi appelé nœud) en cours d'exécution sont représentés dans une architecture de type graphe. Chaque composant (e.g. moteur de rendu, méthode de cartographie, méthode d'exploration autonome...) inclus dans FLYBO correspond à un ou plusieurs

nœuds de ce graphe. Les nœuds sont des processus exécutables qui utilisent ROS pour communiquer entre eux via des *topics*[iii] en informant le processus ROS maître de leurs intentions. En tant qu'intention, un nœud peut transmettre des données à un *topic* (on parle de *publisher*[iv]) ou réceptionner des données (on parle de *subscriber*[v]). Le processus ROS maître permet d'enregistrer et de suivre tous les *topics*, les échanges, et les nœuds initialisés et exécutés. Ce système de communication inter-processus simple est une des fonctionnalités qui rend flexible et modulable notre environnement de simulation. Chaque composant de FLYBO communique avec les autres composants via cette fonctionnalité. De plus, ROS fournit un moyen facile de visualisation (RVIZ) permettant de visualiser diverses informations telles que la position du robot, la cartographie voxelique, les chemins empruntés par le robot, etc.

— **Gazebo**[199] – **RotorS** [98] : Gazebo est un simulateur 3D, cinématique, dynamique, permettant de simuler divers types de robots. Gazebo supporte divers moteurs physiques performants tels que : Open Dynamics Engine (ode) [vi], bullet [vii] ou bien simbody [viii]. RotorS est un simulateur modulable utilisant Gazebo pour modéliser divers types de drones aériens (hélices, structure, ailes...) notamment les drones quadrirotors. L'usage de Gazebo + RotorS dans FLYBO assure une simulation réaliste de la dynamique du drone quadrirotor et publie sur un *topic* l'état du drone en temps réel (>100Hz). L'état du drone est calculé par le moteur physique de Gazebo en fonction de divers paramètres structurels du drone (e.g. envergure, taille des hélices...) et de l'état de chaque moteur (rotation des hélices). L'état du drone correspond à la position et l'orientation du drone. RotorS interface un système de commandes bas niveau pour contrôler la vitesse de rotation des hélices dans la simulation et utilise l'état restitué par le moteur physique pour simuler un capteur d'odométrie idéale. La Figure 3.5 illustre un modèle 3D proposé par **RotorS** [98] et utilisé dans FLYBO.

— **Moteur de rendu** : Le moteur de rendu est un processus qui permet de simuler la vision perçue par une caméra de profondeur. Ce processus restitue une image de profondeur synchronisée avec l'odométrie du drone quadrirotor (transmis par RotorS). Pour augmenter le réalisme de la simulation, nous bruitons l'image de profondeur, le positionnement et l'orientation du drone quadrirotor. Nous fournissons plus de détail concernant ce composant en sous-section 3.4.2.

— **Système de contrôle** : Le système de contrôle de FLYBO utilise les travaux de Faessler et al. [100] évoqué au Chapitre 1 et dont le projet est open-source [ix]. Pour rappel, c'est une architecture de contrôle en cascade composée de deux lois de contrôle proportionnel-dérivée. Cette architecture de contrôle peut assurer le suivi d'une tra-

iii. Un *topic* est un système de transport d'information fonction avec un système d'abonnement et de publication par lequel les nœuds peuvent communiquer (publier ou lire les informations transmises).
iv. Un *publisher* est un nœud qui publie des données dans un *topics*.
v. Un *subscriber* est un nœud qui souscrit a un *topic* pour lire les données qui y transitent.
vi. www.ode.org
vii. github.com/bulletphysics/bullet3
viii. simtk.org/projects/simbody/
ix. github.com/uzh-rpg/rpg_quadrotor_control

jectoire paramétrique de référence et transforme une information de position de vélocité et d'accélération en commande moteur (vitesse angulaire de chaque moteur). La commande est transmise via un *topic* auquel simulateur RotorS est connecté. RotorS répercute la commande sur le comportement physique et dynamique du drone quadrirotor (rotation des hélices, déplacement) simulé dans Gazebo.

— **Cartographie d'occupation** : La cartographie d'occupation reprend la méthode de cartographie présentée dans le Chapitre 2. Grâce à la cartographie d'occupation, nous estimons le volume découvert pendant l'exploration. Dès qu'un voxel étiqueté *'occupé'* est observé, le volume total découvert est augmenté de r^3, r étant la résolution de la carte d'occupation. De plus, les méthodes d'exploration, quand c'est possible, peuvent réutiliser les informations calculées dans la carte d'occupation en souscrivant aux *topics* correspondants.

— **TSDF – Reconstruction 3D** : La reconstruction 3D de l'environnement passe par une grille de voxels de résolution plus fine que celle utilisée par la méthode de cartographie d'occupation voxélique. La reconstruction 3D avec la méthode la TSDF [66] utilise uniquement les voxels à proximité des surfaces (dans l'intervalle de distance de troncature). La TSDF de chaque voxel est estimée quand celui-ci est observé par la caméra de profondeur. Nous fournissons davantage de détails concernant ce composant sous-section 3.4.3.

— **Méthode d'exploration autonome** : En se connectant aux *topics* et en considérant quelques adaptations pour être équitable avec nos expériences (e.g. même façon de construire la trajectoire paramétrique [82] transmise au système de contrôle, des vitesses max et accélération max identique à nos expériences) n'importe quelle méthode d'exploration autonome par drone quadrirotor peut être intégrée à FLYBO. Ceci permet de comparer de nouvelles méthodes aux méthodes déjà intégrées et évaluées. Chaque méthode peut prendre en entrée les informations de cartographie d'occupation (e.g. carte d'occupation, état des voxels), l'image de profondeur (si nécessaire) et l'odométrie. Elle restitue en sortie une trajectoire paramétrique de référence qui s'exécute une fois transmise au système de contrôle.

Synthétiquement, FLYBO est un simulateur basé sur ROS pour drones quadrirotors visant à évaluer les méthodes d'exploration et de reconstruction autonome. La dynamique réaliste du drone quadrirotor est modélisée grâce à la bibliothèque RotorS [98] et au simulateur Gazebo [199]. Le système de contrôle [100] transmet les commandes moteurs nécessaires au suivi de la trajectoire de référence. Les trajectoires estimées par les méthodes d'exploration, et transmise au système de contrôle, sont converties en trajectoires paramétriques sous contrainte (vitesse et accélération maximales) calculée grâce à [82]. Le contrôle des collisions est assuré par une sphère centrée sur la position du drone. Le moteur de rendu utilise l'API Vulkan [x] et simule une caméra de profondeur à 20 Hz synchronisée avec l'odométrie idéale transmise par RotorS. Toutes les scènes considérées sont intégrées comme des maillages au format OBJ dans notre moteur de rendu. Les données telles que

x. www.vulkan.org

FIGURE 3.5 – Modèle 3D du drone quadrirotor utilisé dans les simulations

le maillage final, le nuage de points des surfaces et le volume exploré, sont continuellement enregistrées durant la simulation.

Dans la suite de cette section, nous portons une attention particulière sur le moteur de rendu et la reconstruction 3D. Les autres composants tels que ROS, Gazebo, RotorS et l'architecture de contrôle sont des composants et des bibliothèques devenues standard quand il s'agit de simulation robotique et de drones quadrirotors.

3.4.2 Caméra de profondeur

Pour simuler la caméra de profondeur, nous avons développé un composant qui, en fonction d'un modèle de scène 3D représentée par un maillage triangulaire au format OBJ, la position et l'orientation idéale du drone dans le repère monde de la simulation (transmise par RotorS sur un *topic*), nous simulons une image de profondeur idéale via l'interface de programmation graphique Vulkan.

La caméra de profondeur que nous simulons reprend le modèle sténopé (sans distorsion). Le modèle sténopé se compose des paramètres intrinsèques suivants :

— r_h la résolution horizontale de l'image,

— r_v la résolution verticale de l'image,

— p_h la coordonnée horizontale de projection du centre optique sur l'image,

— p_v la coordonnée verticale de projection du centre optique sur l'image,

— f la distance focale (exprimée en pixel).

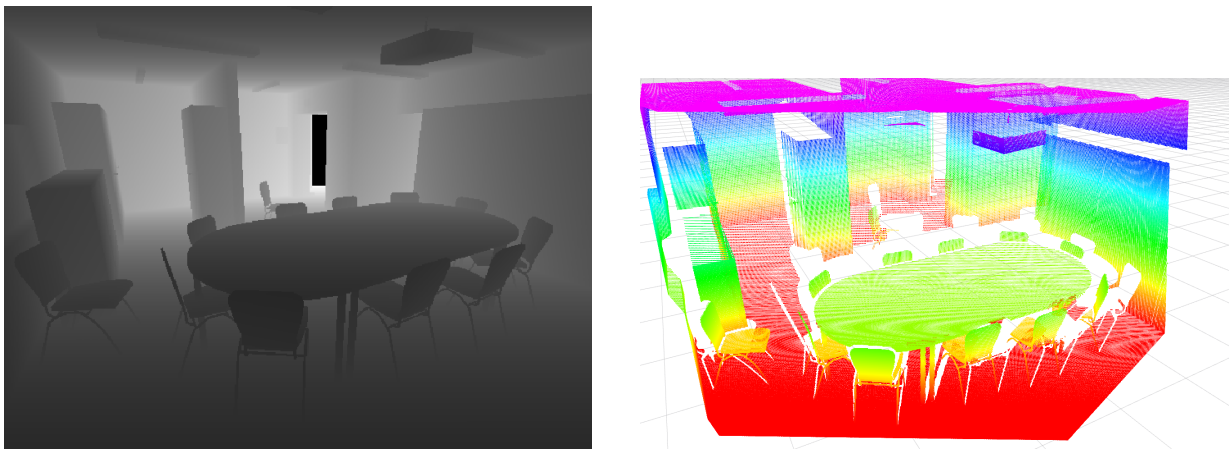
Nous calculons r_v, f, p_h et p_v en fonction de r_h, du champ de vision vertical (fov_v) et horizontal (fov_h) exprimé en radians choisis par l'utilisateur comme suit :

$$\begin{aligned} r_v &= \frac{r_h}{\frac{\tan(fov_h * 0.5)}{\tan(fov_v * 0.5)}}, \\ f &= \frac{r_v}{2 * \tan(fov_v * 0.5)}, \\ p_h &= r_h * 0.5, \\ p_v &= r_v * 0.5. \end{aligned} \tag{3.1}$$

Ces paramètres permettent de calculer la position 3D (x, y, z) dans le repère caméra d'un pixel de coordonnée (u, v) et de valeur de profondeur d (contenue dans le pixel) de l'image de profondeur :

$$\begin{aligned} x &= \frac{(u - p_h) * d}{f}, \\ y &= \frac{(v - p_v) * d}{f}, \\ z &= d. \end{aligned} \tag{3.2}$$

De plus, nous limitons la distance de vision de la caméra (dans nos expériences cette limite est fixée à 5 (m) ou 7 (m)). La Figure 3.6 montre une image de profondeur idéale d'un bureau et le nuage de points résultant de la projection des mesures de profondeur.

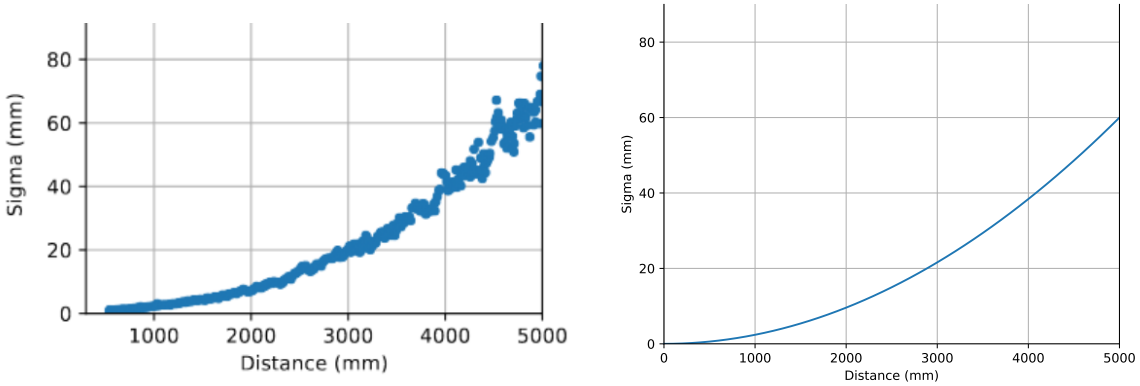


(a) (b)

FIGURE 3.6 – (a) image de profondeur idéale simulée, (b) le nuage de points construit à partir de (a). Les points sont colorés graduellement en fonction de la coordonnée z

Enfin, pour accentuer le réalisme de la simulation, nous ajoutons : (1) du bruit sur l'image de profondeur et (2) du bruit sur l'odométrie idéale transmise par RotorS.

1. Simulation de bruit sur l'image de profondeur.



(a) (b)

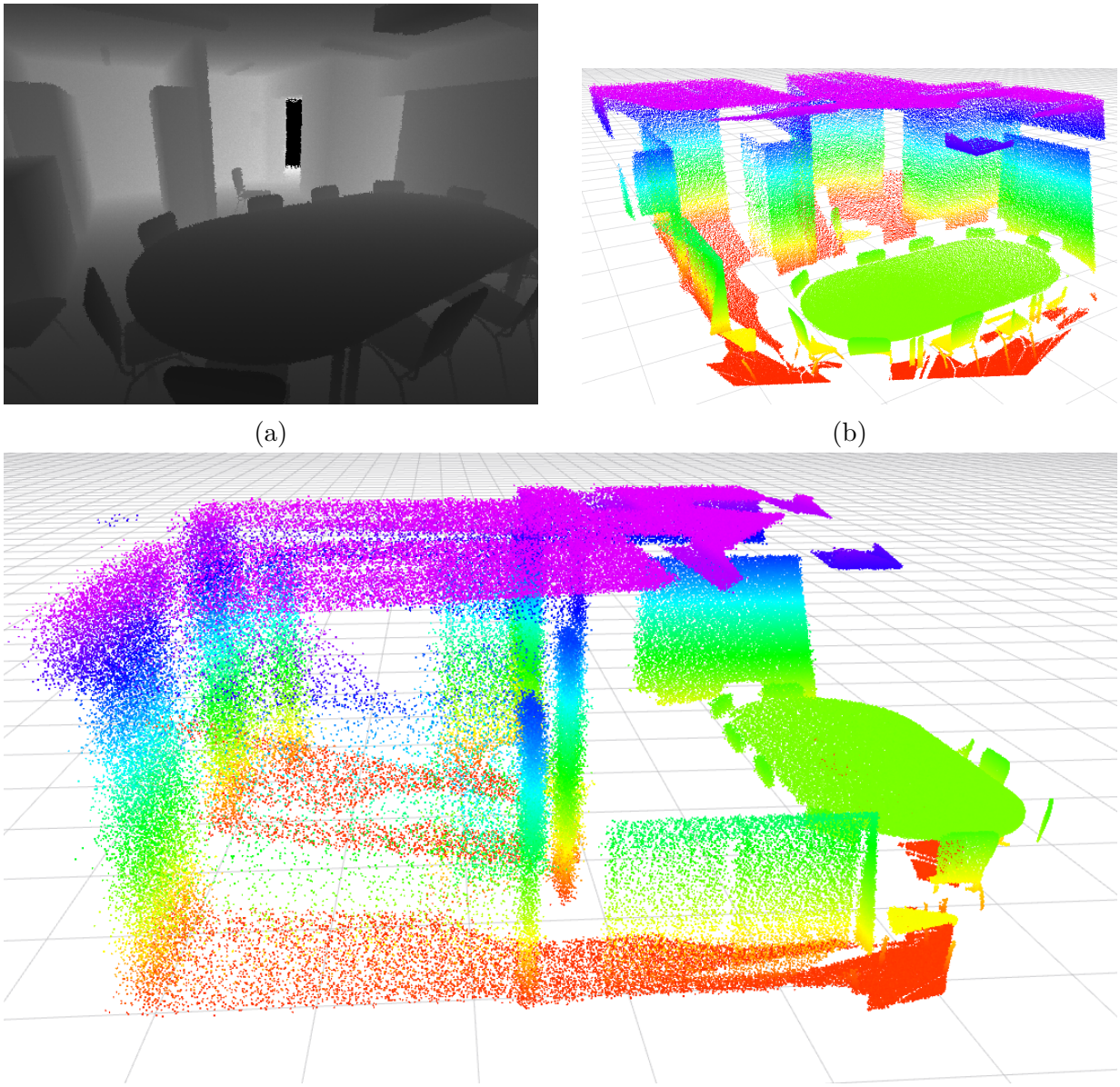
FIGURE 3.7 – (a) mesures de l'erreur RMS (Root Mean Squared) du capteur RealSense R200 (b) l'erreur RMS simulé via la formule $0.0024 * z^2$

Le modèle de bruit que nous utilisons dans FLYBO s'appuie sur les travaux de Keselman et al. [200] (Intel) qui ont caractérisé le comportement du capteur de profondeur RealSense R200. Ce capteur utilise un système de caméras stéréoscopiques proche infrarouge couplé avec un projecteur laser proche infrarouge pour augmenter la texture sur les surfaces faiblement texturées. Dans ces travaux, Keselman et al. en mesurent les performances via la méthode de l'ajustement du meilleur plan sur les données restituées par le capteur quand il observe une mire planaire (e.g. mur). L'erreur RMS est calculée avec une décomposition en valeur singulière des données de profondeur dans l'espace 3D, la plus petite valeur singulière correspond à l'erreur RMS du meilleur ajustement de plan. L'erreur de profondeur du capteur RealSense R200 est quadratique en fonction de la distance observable (voir Figure 3.7(a) extrait de [200]). La Figure 3.7(b) est la courbe issue de la formule quadratique qui suit ces mesures, soit $0.0024 * z^2$.

Nous reprenons donc le modèle de bruit utilisé par Schmid et al. [117] qui simule un bruit gaussien $\hat{z}$ quadratique en fonction d'une donnée de profondeur idéale (z) :

$$\hat{z} = z_{ideale} + \mathcal{N}(0, 0.0024 * z^2).$$

Additionnellement, nous ajoutons des décalages aléatoires par pixel sur les coordonnées image (u, v), comme cela est proposé par Handa et al. [201] pour simuler le bruit d'une caméra RGB-D dans le cadre d'une évaluation pour l'odométrie visuelle et la reconstruction 3D. Ce décalage suit aussi une distribution gaussienne $(\hat{u}, \hat{v}) = E(u + \mathcal{N}(0, 0.5), v +$



(a) (b)

(c)

FIGURE 3.8 – (a) image de profondeur avec bruit, (b) le nuage de point construit à partir de (a) et (c) une vue de côté. Les points sont colorés graduellement en fonction de la coordonnée z.

$\mathcal{N}(0, 0.5)$) avec $\hat{u}$ et $\hat{v}$ les nouvelles coordonnées bruitées et E la fonction d'arrondie à l'entier le plus proche dans l'image de profondeur. En guise d'illustration, après application du bruit, la Figure 3.8 montre l'image de profondeur et le nuage de points résultant.

2. Simulation du bruit de localisation.

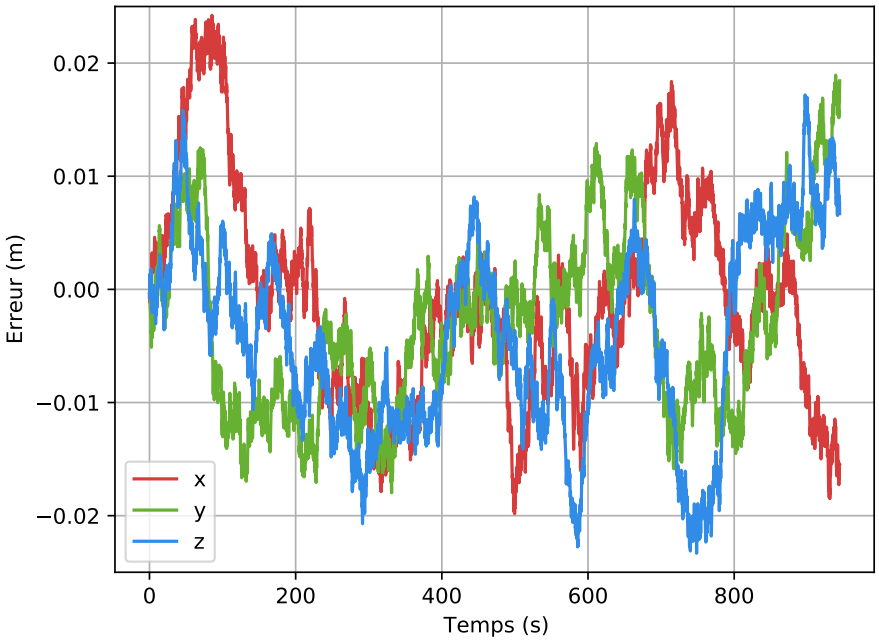


FIGURE 3.9 – Évolution de l'erreur de positionnement au cours d'une campagne d'exploration autonome simulée.

Pour simuler l'erreur de localisation, la position et l'orientation idéales transmises par le simulateur RotorS sont décalées par un bruit de type marche aléatoire (*random walk*). Ce modèle de bruit est comparable à celui que l'on retrouve dans des méthodes de localisation performantes (e.g. VI-SLAM ou GPS RTK). Le modèle est borné de sorte à éviter une trop grande dérive. Le problème de dérive n'est pas considéré par la majorité des méthodes de cartographie d'occupation et d'exploration/reconstruction autonome. À titre d'illustration, la Figure 3.9 montre l'évolution de l'erreur durant 1000 secondes qui décale le positionnement en (x, y, z) du drone. Dans cette illustration et dans nos expériences, nous utilisons une incertitude de positionnement de 2.5 cm. Concernant l'erreur de rotation (tangage, roulis, lacet), nous utilisons une valeur d'incertitude de 0.5° pour chaque angle.

3.4.3 Reconstruction 3D

L'évaluation de la qualité de reconstruction 3D est régie par deux critères : (i) la qualité du maillage reconstruit et (ii) l'efficacité de la reconstruction 3D (analogue à l'efficacité d'exploration en m^3/s ici nous mesurons le pourcentage de surfaces reconstruit à un instant t par rapport au modèle 3D idéal). Pour faire cette évaluation (notamment l'efficacité de la reconstruction) la simulation passe par la construction d'un modèle 3D en temps réel. La construction d'un modèle 3D prend en entrée les images de profondeur provenant du capteur simulé afin de les fusionner en un seul modèle 3D. La fusion se fait via la méthode de la TSDF [66] (présentée dans le Chapitre 1) dans une structure de données adaptée pour les calculs GPU. L'usage du GPU est devenu standard pour accélérer des calculs hautement parallélisables. Pour la reconstruction 3D temps réel, nous utilisons la méthode développée dans la bibliothèque Open3D [xi] [202]. Cette méthode utilise une structure de données de type table de hachage. Plus particulièrement, Open3D reprend les travaux de [203] qui propose de diviser grossièrement l'espace 3D en blocs (stratégie globalement éparse). Les blocs à proximité de surfaces observées sont alloués dans une table de hachage et référencés par une coordonnée 3D. Chaque bloc alloué est une grille dense de voxels englobant l'espace nécessaire pour le calcul de TSDF (stratégie localement dense). Cette double stratégie offre deux avantages : (i) elle préserve la localité des informations dans le but de représenter efficacement les surfaces contiguës des objets peuplant la scène et (ii) elle se concentre uniquement sur les blocs à proximité des surfaces, offrant un gain en mémoire. Par ailleurs, cette représentation est bien adaptée pour le calcul hautement parallèle d'un GPU.

Grâce à cette méthode, nous pouvons extraire, à intervalles de temps réguliers, un maillage ou un nuage de points répartis sur les surfaces du maillage via la méthode dite du "Marching Cube" [67], lors d'une expérience d'exploration. Ce nuage de points ou ce maillage permet d'évaluer l'efficacité de la reconstruction 3D au cours de la simulation et la qualité finale de la reconstruction. Comme nous l'avons vu dans les chapitres 1 et 2, la TSDF est affectée par deux paramètres (i) la résolution des voxels et (ii) la distance de troncature. Ces deux paramètres affectent la qualité de la reconstruction, la vitesse d'insertion et l'espace occupé en mémoire. Le choix de ces paramètres est fait en fonction de l'envergure des scènes 3D utilisée et de la machine de calcul sur laquelle la simulation s'exécute. Dans nos expériences, nous utilisons une résolution de voxel de 3 cm et une distance de troncation de 9 cm (pour toutes les scènes contenues dans FLYBO). Ce paramétrage nous permet de produire des maillages suffisamment précis tout en étant sûr que nous ne dépassons pas l'espace mémoire de la machine utilisée (dans nos expériences le GPU équipé sur notre machine dispose de 6 GO de VRAM). Il est aussi important de noter que la distance de troncature est plus grande que l'étendue du bruit appliqué sur l'image de profondeur (c.f. Figure 3.7). Ceci permet de réduire le bruit dans le maillage produit. Des valeurs trop faibles peuvent induire du bruit et des trous dans le maillage produit.

xi. www.open3d.org

3.5 Métriques d'évaluation

L'exploration autonome a pour objectif initial de construire une cartographie voxélique d'un environnement inconnu, mais elle peut aussi être associée à une campagne de reconstruction 3D surfacique. En pratique, au terme d'une campagne d'exploration et de reconstruction, la qualité du modèle 3D reconstruit peut constituer un facteur décisif de la réussite de cette campagne, ceci indépendamment de l'efficacité et de la rapidité du processus d'exploration. Une campagne peut être un échec si les données collectées ne sont pas suffisamment précises et complètes (environnement entièrement reconstruit) pour utiliser la reconstruction après l'exploration. Nous proposons de joindre à FLYBO un ensemble de métriques permettant d'évaluer l'efficacité et la qualité de l'exploration autonome d'une part et de la reconstruction surfacique au cours de l'exploration (reconstruction de surface en ligne) d'autre part. Les métriques mentionnées dans cette partie peuvent être moyennées sur plusieurs expériences pour tenir compte de la nature stochastique des tâches et des méthodes d'exploration.

3.5.1 Métriques d'exploration volumique

L'objectif principal d'une méthode d'exploration autonome est d'observer entièrement un environnement délimité arbitrairement (c.f. Chapitre 1). Étant donné $\mathcal{V}_t \in \mathbb{R}^3$ une carte d'occupation organisée dans une grille de voxel extraite à un instant t de l'expérience d'exploration, le volume découvert $V(\mathcal{V}_t)$ est calculé en multipliant $|\mathcal{L}|$ le cardinal de l'ensemble $\mathcal{L}$, constitué des voxels étiquetés comme *'libre'* et *'occupé'*, par la résolution r^3 d'un voxel :

$$\mathcal{L} = \{v \in \mathcal{V} \mid \mathcal{E}(v) \in \{\text{'libre', 'occupé', }\}\} \tag{3.3}$$

$$V(\mathcal{V}_t) = |\mathcal{L}| * r^3, \tag{3.4}$$

où $\mathcal{E}(v)$ est l'étiquette de v. Ceci nous permet d'extraire deux métriques d'exploration volumique standard [3, 69, 118, 121, 166] :

— **L'efficacité d'exploration volumique** exprimée en m^3/s calculée en mesurant régulièrement le volume (toutes les 100 ms dans nos expériences).

— **La complétude d'exploration volumique** est définie comme le ratio du volume obtenu par une méthode à un temps donné par rapport à un volume de référence au même moment. Cette métrique permet de comparer la capacité d'une méthode à explorer la globalité de l'environnement par rapport à celle de la méthode de références. Dans nos expériences, le volume de référence correspond au volume le plus élevé atteint par une méthode à un temps t (t peut être différent du temps alloué pour l'expérience, cela dépend de la rapidité des méthodes les plus performantes).

3.5.2 Métriques de reconstruction surfacique en ligne

Pour évaluer la capacité d'une méthode à reconstruire son environnement, nous utilisons les métriques standards d'évaluation couramment utilisées sur de grandes scènes 3D dans le domaine de la reconstruction 3D [121, 192, 204]. Le maillage est reconstruit via la TSDF calculées par notre composant de reconstruction 3D (c.f. Section 3.4). Les métriques utilisées pour évaluer les performances sur la tâche de reconstruction 3D d'une méthode d'exploration sont listées ci-dessous, nous proposons ensuite de les détailler.

— **L'efficacité en termes de rappel géométrique** : exprimé comme le pourcentage de surface correctement reconstruite en fonction du temps. La performance en termes de rappel géométrique se mesure en ligne (durant l'exploration) à intervalle de temps régulier.

— **Le F-score** : mesure la justesse de reconstruction à l'issue d'une campagne d'exploration. Le F-score se calcule comme la moyenne harmonique entre le rappel et la précision du maillage reconstruit.

— **La RMSE**: mesure l'erreur de reconstruction géométrique à l'issue d'une campagne d'exploration. La RMSE est exprimée en mètre (m).

Ces métriques sont calculées soit en fonction d'un nuage de points denses idéal noté $\mathcal{G}$, soit d'un maillage idéal noté $\mathcal{S}$.

Comme nous travaillons avec de grandes scènes, il est d'usage de simplifier les données produites par la reconstruction 3D en les sous-échantillonnant en nuage de points [192]. Notre méthode de reconstruction construit un maillage précis des surfaces observées, mais peut aussi fournir un nuage de points très dense appartenant à ces surfaces. Ce nuage de points est sous-échantillonné avec une grille de résolution τ ($\tau = 2$ cm pour nos expériences). Si plusieurs points appartiennent à la même cellule, c'est la moyenne de leurs coordonnées qui est retenue. Nous appelons $\mathcal{M}$ le nuage de points résultant de cet échantillonnage.

La performance en termes de rappel géométrique. De manière identique à de la classification, le rappel de $\mathcal{M}$ par rapport à $\mathcal{G}$ mesure combien de points (candidats) pertinents de $\mathcal{M}$ ont été correctement reconstruits (sélectionnés). Si nous prenons un point $g \in \mathcal{G}$, nous considérons que g est correctement reconstruit s'il existe un point $r \in \mathcal{M}$ tel que la distance quadratique entre r et g ($\delta(r, g)$) est inférieur au seuil choisi ϵ :

$$\mathcal{A} = \{g \in \mathcal{G} \mid \exists r \in \mathcal{M},\ \delta(r, g) < \epsilon\}. \tag{3.5}$$

En ce sens, un point g est considéré comme pertinent s'il est suffisamment proche d'un point de $\mathcal{M}$.

Le rappel R s'exprime comme le ratio entre le cardinal des points pertinent $\mathcal{A}$ et celui de $\mathcal{G}$:

$$R(\mathcal{M}, \mathcal{G}, \epsilon) = \frac{|\mathcal{A}|}{|\mathcal{G}|}. \tag{3.6}$$

Une illustration des points pertinents (vert) et ceux non pertinents (rouge) de $\mathcal{G}$ par rapport au nuage de points $\mathcal{M}$ extrait durant une expérience d'exploration est donnée Figure

3.10(a). Lors d'une expérience d'exploration, nous mesurons régulièrement le rappel (toutes les 20 s). Ces mesures permettent de calculer l'efficacité de rappel. La performance en termes de rappel géométrique se présente dans un graphique qui montre l'évolution du rappel en fonction du temps. Le graphique permet de contrôler et de comparer la rapidité de chaque méthode d'exploration autonome à reconstruire l'environnement.

La précision de reconstruction géométrique. La précision est utilisée dans le calcul du F-score. La précision de $\mathcal{M}$ par rapport à $\mathcal{G}$ dépend du nombre de points sélectionnés qui sont pertinents. En reprenant les notations données pour définir le rappel, nous considérons qu'un point r est pertinent s'il existe un point $g \in \mathcal{G}$ tel que la distance quadratique entre r et g ($\delta(r, g)$) est inférieur au seuil choisi ϵ :

$$\mathcal{B} = \{r \in \mathcal{M} \mid \exists g \in \mathcal{G},\ \delta(r, g) < \epsilon\}. \tag{3.7}$$

Un point r est pertinent (correctement reconstruit) s'il est suffisamment proche d'un point de $\mathcal{G}$. La précision P s'exprime comme le ratio entre le cardinal des points pertinents $\mathcal{B}$ et celui de $\mathcal{M}$:

$$P(\mathcal{M}, \mathcal{G}, \epsilon) = \frac{|\mathcal{B}|}{|\mathcal{M}|}. \tag{3.8}$$

La précision quantifie l'exactitude de la reconstruction. La Figure 3.10(b) illustre les points pertinents (vert) et non pertinents (rouge) de $\mathcal{M}$ extraits durant une expérience d'exploration.

Le F-score. La précision et le rappel seuls ne sont pas suffisants pour mesurer les performances et la validité des résultats de reconstruction obtenues par une méthode d'exploration. La précision seule peut être maximisée en produisant un ensemble très petit de points précisément reconstruit. Le rappel seul peut être maximisé en couvrant densément l'espace avec des points. Le F-score agrège la précision P et le rappel R entre un nuage de points reconstruit $\mathcal{M}$ restitué en fin d'expérience et le modèle idéal $\mathcal{G}$ correspondant de la manière suivante :

$$F(\mathcal{M}, \mathcal{G}, \epsilon) = \frac{2 \times P(\mathcal{M}, \mathcal{G}, \epsilon) \times R(\mathcal{M}, \mathcal{G}, \epsilon)}{P(\mathcal{M}, \mathcal{G}, \epsilon) + R(\mathcal{M}, \mathcal{G}, \epsilon)}, \quad \text{(F-score \%)}. \tag{3.9}$$

Dans FLYBO, le F-score mesure la justesse de reconstruction géométrique d'une méthode d'exploration sur la tâche de reconstruction. Un F-score élevé ne peut être obtenu que par un score de rappel et de précision élevé. Une précision ou un rappel de 0 entraine un F-score de 0. Ceci signifie qu'une méthode d'exploration autonome performante sur la tâche de reconstruction de surfaces en ligne est une méthode qui a précisément et entièrement reconstruit son environnement.

Pour ces métriques l'utilisation directe des modèles de CAO (maillage $\mathcal{S}$), échantillonnés en nuages de points uniformes n'est pas adéquate, en particulier pour la mesure de rappel qui tient compte de toutes les surfaces géométriques pouvant être reconstruites. Il est fréquent, dans un modèle CAO, que certaines surfaces ne puissent être mesurées. Pour

éviter d'introduire des biais importants dans notre évaluation, nous ne devons considérer que les surfaces pouvant être perçues par le drone. Pour ce faire, nous exécutons un mode de cartographie d'exploration interactive sans ajout de bruit et contrôlé manuellement. Lors de l'utilisation, les déplacements de la caméra sont réalisés avec la souris et le clavier comme dans un jeu vidéo en première personne. Les données de profondeur sont converties en nuage de points et agrégées dans une grille éparse de résolution 1 cm. Cela permet de récupérer uniquement les parties extérieures perceptibles des scènes de CAO et de les stocker sous forme de nuages de points denses et très précis afin de calculer les métriques considérées (rappel, précision, F-score). De même que pour $\mathcal{M}$, le nuage de points résultant de notre mode de cartographie d'exploration interactive est sous-échantillonné avec une grille de résolution $\tau = 2\ cm$.

Le seuil ϵ est choisi pour être suffisamment grand afin que les métriques ne soient pas affectées par le sous-échantillonnage des nuages de points, mais également, suffisamment petit pour obtenir des résultats pertinents. Un seuil trop élevé aurait pour effet d'associer des points entre le nuage de points reconstruit ($\mathcal{M}$) et idéal ($\mathcal{G}$) appartenant à des surfaces différentes. Un seuil trop faible n'associerait aucun point entre $\mathcal{M}$ et $\mathcal{G}$ (0% de rappel, de précision et de F-score). Comme dans [192], nous choisissons un seuil de distance ϵ par défaut de $\epsilon = 2 \times \tau$. Ce seuil présente un juste compromis entre l'influence portée par le sous-échantillonnage et l'obtention de résultats pertinents. Il est important de rappeler que le sous-échantillonnage avec une grille de résolution τ assure une répartition homogène des points. Une répartition spatiale homogène des points est nécessaire car elle évite des biais potentiels sur les métriques venant de zones (surfaces) surreprésentées en point 3D.

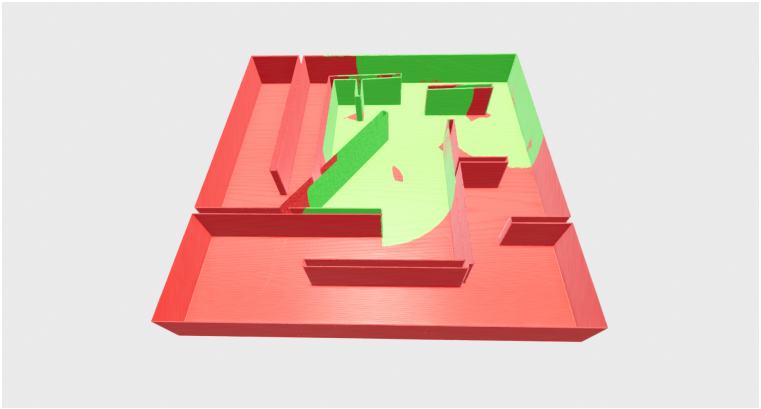
La RMSE. Nous utilisons la racine de l'erreur quadratique moyenne (RMSE) pour mesurer l'écart entre un nuage de points $\mathcal{M}$ reconstruit et un modèle 3D idéal $\mathcal{S}$ constitué de face triangulaire. L'écart grandi si $\mathcal{M}$ est imprécis. Nous définissons l'erreur quadratique d'un point r comme la distance orthogonale quadratique $\delta(r, s)$ au plan triangle $s \in \mathcal{S}$ le plus proche (ou arrête si la projection orthogonale tombe sur une arête) :

$$e(r, \mathcal{S}) = \min_{s \in \mathcal{S}} \delta(r, s). \tag{3.10}$$

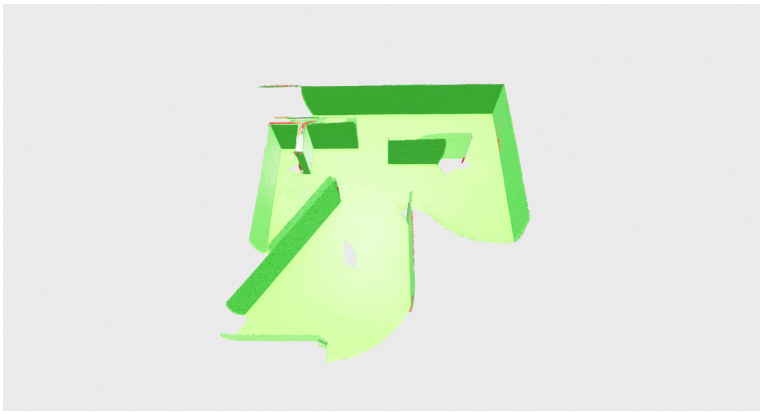
La justesse de $\mathcal{M}$ par rapport à $\mathcal{S}$ est mesurée par la RMSE :

$$\text{RMSE}(\mathcal{M}, \mathcal{S}) = \sqrt{\frac{1}{|\mathcal{M}|} \sum_{r \in \mathcal{M}} e(r, \mathcal{S})}. \tag{3.11}$$

La procédure complète qui permet de calculer la RMSE est détaillé dans [204]. Dans notre cas nous utilisons la procédure implémentée dans CloudCompare.



(a)



(b)

FIGURE 3.10 – Deux nuages de points illustrant via colorisation (a) le rappel avec le nuage de points idéal $\mathcal{G}$ et (b) la précision avec le nuage de points reconstruit $\mathcal{M}$ extrait durant l'exploration. En vert les points jugés correctement reconstruits (ceux dont la distance est inférieure au seuil de selection ϵ) et en rouge les points dont la distance est supérieure au seuil de selection ϵ.

3.6 Méthodes implantées dans FLYBO

Dans FLYBO, sont disponibles sept méthodes de l'état de l'art : Classic [2], Rapid [69], NBVP [116], AEP [118], ESM_{IPP} [117], ESM_{Recon} [117] et SplatPlanner [3]. Ces sept méthodes couvrent les trois stratégies d'exploration autonome que nous pouvons trouver dans la littérature, i.e les stratégies basées sur les frontières, les stratégies basées sur l'échantillonnage et les stratégies hybrides.

Nous avons intégré ces méthodes dans l'objectif d'obtenir des comparaisons équitables sur les tâches d'exploration et reconstruction. Pour certaines méthodes, en l'absence d'implémentation publique, cela nécessite une ré-implémentation complète, c'est le cas de Classic [2] et Rapid [69]. Pour les autres méthodes seulement modifiées afin qu'elles soient compatibles à FLYBO. Nous les avons éventuellement améliorées si nous percevons des

erreurs d'ordre techniques d'implémentation (e.g. NBVP [116]).

Cette section vise à expliquer les raisons pour lesquelles les méthodes ont été intégrées dans FLYBO. En parallèle, nous détaillons succinctement les implémentations que nous avons réalisées et les modifications apportées aux implémentations déjà publiées par les auteurs des méthodes.

3.6.1 Méthodes basées sur la notion de frontières

De nombreuses méthodes découlent du concept de frontière. Classic [2] est la première méthode introduisant l'usage de ce concept pour l'exploration autonome. Cette méthode développée en 1997 par Yamauchi et al., continue à montrer des résultats compétitifs même quand elle est adaptée aux drones quadrirotors [69]. C'est pourquoi nous pensons pertinent de faire l'effort de proposer une implémentation publique de cette méthode et de l'intégrer dans FLYBO.

La seconde méthode basée sur les frontières proposée dans FLYBO est Rapid [69]. Rapid [69] est une amélioration directe de Classic [2] développée pour le drone quadrirotor. Cette méthode introduit en plus du mode de vol "classic" (mode proposé par Classic [2]) un second mode de vol qui vise à accélérer la vitesse d'exploration. Des résultats et des comparaisons rapportés par les auteurs, cette méthode semble constituer une très bonne méthode de référence pour représenter les méthodes basées sur les frontières (dans FLYBO). Les expériences et les résultats que nous proposons Chapitre 4 le confirmeront.

À notre connaissance, aucune implémentation de Classic [2] et Rapid [69] pour drones quadrirotors n'est disponible publiquement. C'est pourquoi, nous proposons notre propre implémentation. Nous détaillons ci-dessous les éléments principaux de ces deux méthodes telles que nous les avons implémentées.

Classic [2]. La méthode procède en visitant l'amas de frontières disponibles le plus proche à chaque itération de l'algorithme, jusqu'à ce que toutes les frontières atteignables par le robot soient visitées. L'implémentation que nous proposons étend, pour une exploration et une navigation 3D, la version initialement proposée par Yamauchi et al. [2]. La cartographie d'occupation proposée dans FLYBO est suffisante pour être utilisée par Classic [2]. Nous listons ci-dessous les étapes de la procédure de Classic [2] et proposons ensuite quelques détails complémentaires sur les choix algorithmiques des étapes :

1. Les voxels frontières transmis par la cartographie d'occupation sont regroupés en amas. Le voxel situé approximativement au centre de l'amas constitue un objectif potentiel à visiter.
2. Le chemin vers l'objectif le plus proche est calculé avec l'algorithme de Djikstra [68] en utilisant la carte d'occupation transmise par le module de cartographie.
3. Le chemin sélectionné menant à un amas est simplifié (pour faciliter son suivi) et optimisé en trajectoire paramétrique [82].
4. La trajectoire est exécutée par le système de contrôle. La procédure est réitérée tant qu'il reste des frontières atteignables.

(1.) Les frontières sont regroupées en amas à l'aide d'une méthode de croissance de région qui étend de façon incrémentale les amas à partir d'un ensemble de points 3D répartis uniformément dans une grille régulière de résolution κ. Plus précisément, ce processus utilise un regroupement local de type "k-means" et tient compte de la connectivité spatiale entre les voxels frontières. Des détails supplémentaires sur cet algorithme sont fournis dans [205]. La taille d'un amas est régulée par le paramètre κ. Dans nos expériences, nous utilisons une valeur de κ fixe égale à 1.5 m. Ce choix est fait empiriquement en fonction des explications proposées par les auteurs de Classic [2]. Une valeur de κ trop petite revient à considérer les frontières une par une (sans regroupement). Dans le cas d'une exploration 3D cette stratégie n'est pas efficace, notamment quand les frontières sont situées au-dessus du robot. Une valeur de κ trop grande réduit le nombre d'amas, et donc peut fortement augmenter la distance à l'objectif le plus proche. Avec κ égal à 1.5 m la méthode obtient des résultats pertinents et comparables à ceux rapportés dans la littérature. Nous précisons que même si nous construisons des groupes de frontières suffisamment grands, toutes les frontières atteignables sont considérées et seront explorées si le temps limite de l'expérience d'exploration le permet.

(2.) Un chemin construit restitué par l'algorithme de Djikstra est considéré comme valide seulement s'il respecte certaines conditions, sinon la recherche de chemin continue. Nous avons deux conditions (i) que la navigation soit sans risque pour le robot (même quand la trajectoire arrive proche d'un objectif) et (ii) que le centre de l'amas atteint par le chemin soit observable. Pour résoudre (i) nous contraignons l'algorithme de Djikstra à ne choisir que des voxels dont la distance au voxel occupée le plus proche est supérieure au rayon d'une sphère de collision dont le diamètre est égal à l'envergure du drone choisi dans les expériences.

Dès qu'un chemin est restitué par l'algorithme de Djikstra, nous cherchons une position sur le chemin qui permette au robot d'observer la frontière située au centre de cet amas (l'objectif). L'objectif est ensuite retiré de la liste des frontières. S'il n'existe aucune position qui permette d'observer l'objectif, nous cherchons un autre chemin. Nous précisons que, la trajectoire s'arrête avant le voxel frontière (à une distance égale au rayon de la sphère de collision) ce qui évite d'arriver trop près de voxels inconnus potentiellement occupés.

Rapid [69]. La méthode introduit un second mode de fonctionnement "rapid" que nous avons dû implémenter et qui s'ajoute au mode de Classic [2]. En mode "rapid" le robot priorise les frontières dans le champ de vision du drone afin de maintenir la vitesse de ce dernier tout en gardant un comportement réactif aux obstacles. Le mode "rapid" est actif quand il existe une frontière (non occultée) dans le champ de vision du drone. Nous récupérons les frontières dans le champ de vision du drone via une simple opération de lancer de rayon. En parcourant la liste des frontières récupérées (frontières perceptibles par le drone), nous cherchons celle qui minimise le changement de direction et de vitesse. Comme dans Rapid [69], la vitesse se calcule en fonction de la distance qui sépare le drone d'une frontière sélectionnée : plus cette frontière est proche, plus la vitesse est réduite. Ceci permet un comportement réactif aux potentiels obstacles encore inconnus. La vitesse

maximale est atteinte quand la distance entre le drone et la frontière est égale à la distance maximale de perception du capteur. Le système de contrôle implanté dans FLYBO permet le contrôle du drone en vitesse en régulant la vitesse de rotation des moteurs en fonction d'une vitesse transmise par notre implémentation de Rapid [69].

Nous venons de résumer succinctement le fonctionnement de notre implémentation de Rapid [69]. Puisque la méthode est conçue pour le drone quadrirotor, l'implémentation que nous proposons n'apporte aucune modification, autre que celle nécessaire à l'intégration dans FLYBO, par rapport aux explications données par les auteurs [69]. Vous trouverez davantage de détail sur cette méthode dans [69].

3.6.2 Méthodes basées sur l'échantillonnage

Concernant les méthodes basées sur l'échantillonnage, FLYBO en intègre trois dont NBVP [116]. NBVP [116] est une méthode précurseuse de la stratégie d'exploration autonome basée sur l'échantillonnage. Cette méthode est très souvent utilisée comme référence dans le domaine de l'exploration autonome avec un drone volant [3, 117, 118, 150, 168].

Nous avons aussi intégré ESM_{IPP} [117] dont les résultats rapportés par les auteurs sont supérieurs à ceux de NBVP [116]. De plus, ESM_{IPP} [117] n'est pas sujet aux problèmes de terminaison prématurée que rencontre NBVP [116]. Nous verrons dans le Chapitre 4 que cette méthode est l'une des plus performantes sur certaines des métriques proposées.

Enfin, les auteurs de ESM_{IPP} [117] proposent une autre version de leur méthode qui priorise d'abord l'exploration des surfaces puis la qualité de reconstruction 3D puis le volume inconnu sous le nom de ESM_{Recon} [117]. Nous trouvons pertinent de l'ajouter dans FLYBO puisque, au terme d'une campagne d'exploration et de reconstruction, la qualité du modèle 3D résultant peut constituer un facteur d'échec ou de réussite de la mission. En comparant les résultats obtenus par ESM_{Recon} [117] sur les métriques de FLYBO, il sera possible de fournir un premier avis sur la réelle utilité de l'usage des surfaces dans la fonction qui guide l'exploration.

Nous détaillons ci-dessous les modifications et adaptations faites aux implémentations publiques de NBVP [116], de ESM_{IPP} [117] et de ESM_{Recon} [117] lors de leur intégration dans FLYBO.

NBVP [116] L'implémentation publiée par les auteurs de NBVP [116] (que nous nommons dans cette partie uniquement NBVP_{legacy} [116] [xii]) utilise initialement Octomap [51] (méthode de cartographie d'occupation) dans le même fil d'exécution ("thread") que l'algorithme d'exploration. Nous avons pu observer que ceci constitue un goulot d'étranglement, Octomap mettant longtemps à traiter un flux continu d'image de profondeur acquise par le capteur. Nous avons comparé expérimentalement la version originale NBVP_{legacy} [116] et celle de FLYBO (NBVP_{flybo}) qui est notre intégration du code avec des "thread" parallèles de calcul distinct pour Octomap et la méthode d'exploration. Dans cette expérience, se déroulant durant 900 secondes dans l'environnement Maze, nous considérons une résolution

xii. `https://github.com/ethz-asl/nbvplanner`

d'image de profondeur de 450×600, et une fréquence d'acquisition d'images de 20 Hz. Les résultats de ces expériences sont fournis dans la Figure 3.11 et le tableau Table 3.2. Ces résultats montrent un écart de performances entre les deux implémentations en faveur de NBVP$_{flybo}$. Cet écart peut être en partie expliqué par le temps moyen pris par itération entre NBVP$_{legacy}$ [116] et NBVP$_{flybo}$ [116].

Le comportement d'exploration résultant de l'utilisation de deux threads distincts, au lieu d'un seul, rend l'implémentation conceptuellement plus proche de celle des autres méthodes d'exploration et surtout d'AEP [118]. Nous pensons qu'il s'agit d'une erreur technique dans l'implémentation publique qui a artificiellement réduit les performances de la méthode. Nous nous contentons d'utiliser notre modification (NBVP$_{flybo}$ [116]) qui rend l'architecture de toutes les méthodes d'exploration autonome comparables et homogènes. Les problèmes de performance décrits ici ont également été documentés dans les travaux publiés dans [69].

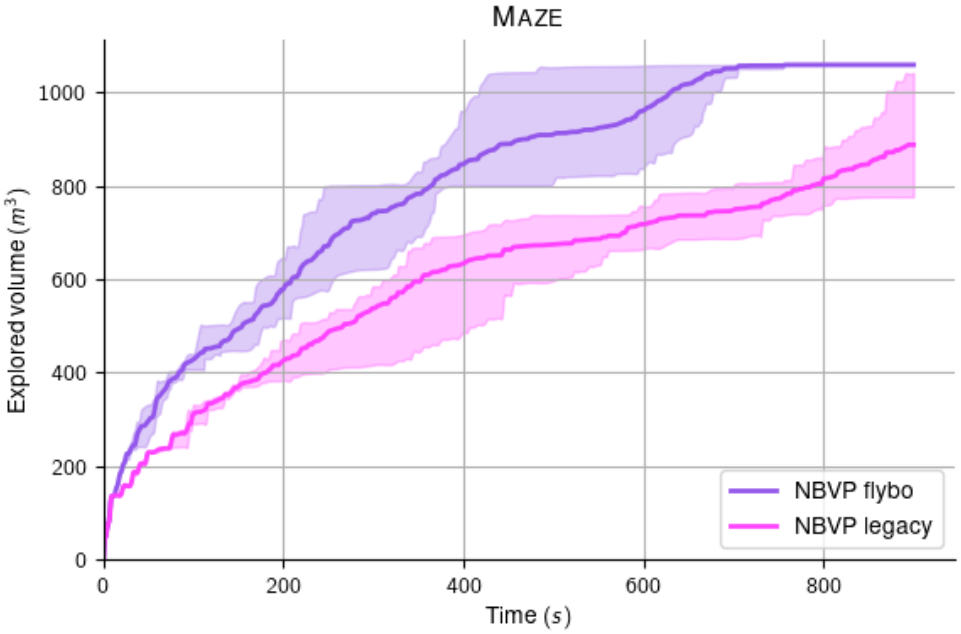


FIGURE 3.11 – Efficacité volumique de NBVP$_{flybo}$ [116] (utilisant notre modification) et NBVP$_{legacy}$ [116] (publique): Volume exploré en fonction du temps moyennés sur cinq expériences dans l'environnement MAZE. L'intervalle coloré autour des courbes correspond au volume minimum et maximum obtenu sur les cinq executions.

Nous ajoutons que NBVP [116] dépend de manière inhérente à la méthode de cartographie d'occupation Octomap [51]. Il est peu pratique de transférer l'algorithme d'exploration à notre cartographie d'occupation qui repose sur une grille régulière. C'est pourquoi nous gardons l'usage d'Octomap [51] au sein de la méthode.

	#Itérations	Temps moyen Par iter. (ms)	Taille tot. du chemin (m)
NBVP$_{flybo}$ [116]	187 ± 18	708 ± 437	258 ± 30
NBVP$_{legacy}$ [116]	103 ± 4	5606 ± 2285	144 ± 6

TABLE 3.2 – Statistiques comparatives extraites des expériences réalisées sur la scène MAZE au temps $t = 900s$. Nous rapportons pour NBVP$_{flybo}$ [116] et NBVP$_{legacy}$ [116] le nombre d'itérations et le temps de calcul moyen par itération de leur algorithme ainsi que la distance totale parcourue par le drone.

ESM$_{IPP}$ et ESM$_{Recon}$ [xiii]. Les seules modifications apportées à l'implémentation initiale publiée par les auteurs est l'usage de notre module de cartographie d'occupation et la conversion des trajectoires restituées par ESM en trajectoires paramétriques [82]. Nous n'avons observé aucune différence significative de résultat entre l'intégration d'ESM dans FLYBO et l'implémentation initiale publiée par les auteurs.

3.6.3 Méthodes hybrides

Concernant les méthodes hybrides, nous avons intégré dans FLYBO AEP [118] et SplatPlanner [3].

AEP [118] se situe comme une amélioration de NBVP [116]. Les principales modifications et optimisations proposées par AEP [118] sont une réduction des temps de calcul et une solution au problème de terminaisons prématurées. Les améliorations sont confirmées par les résultats rapportés dans la littérature [117, 118].

SplatPlanner [3] décrit Chapitre 2, est intégré pour son originalité par rapport aux autres méthodes présentes dans FLYBO et son efficacité d'exploration volumique. Des premiers résultats que nous rapportons sur la tâche d'exploration autonome volumique dans [3], SplatPlanner semble plus performant que ESM$_{IPP}$ [117] et AEP [118].

AEP [118]. La seule modification apportées à l'implémentation publique d'AEP [118] [xiv] est la conversion des trajectoires calculées par la méthode en trajectoires paramétriques [82]. Nous ajoutons que comme pour NBVP [116], AEP [118] dépend de manière inhérente de la méthode de cartographie d'occupation Octomap [51]. Vu qu'il est peu pratique de transférer l'algorithme d'exploration à notre cartographie d'occupation, nous gardons l'usage d'Octomap [51] au sein d'AEP [118]. Nous n'avons observé aucune différence significative de résultat entre l'intégration d'AEP [118] dans FLYBO et l'implémentation initiale publiée par les auteurs.

SplatPlanner [3]. SplatPlanner ne nécessitait aucune modification par rapport à l'implémentation initiale que nous avions fait pour notre premier article [3].

xiii. `https://github.com/ethz-asl/mav_active_3d_planning`
xiv. `https://github.com/mseln/aeplanner`

3.7 Conclusion

Dans ce chapitre, nous avons présenté FLYBO : un environnement de simulation unifié étudié pour évaluer des systèmes d'exploration autonomes avec un drone quadrirotor équipé d'une caméra de profondeur. Notre contribution propose (i) 11 environnements réalistes 3D de complexité et de taille variées allant d'une difficulté modérée à très élevée, (ii) des métriques permettant de comparer l'efficacité et la qualité de plusieurs méthodes d'exploration autonome d'une part et de reconstruction lors de l'exploration d'autre part, (iii) des composants communs aux systèmes d'exploration autonomes afin d'évaluer équitablement, précisément et dans les mêmes conditions leurs performances d'exploration dans un environnement de simulation contrôlé et (iv) sept méthodes d'exploration autonome extraites de l'état de l'art, avec y compris des méthodes sans implémentations publiques, utilisant donc nos propres implémentations et dont chacune sont intégrées et factorisées dans FLYBO.

Dans le chapitre suivant, nous présentons les résultats issus de nombreuses expériences que nous avons réalisées évaluant les sept méthodes implantées dans FLYBO. Leur robustesse et leur efficacité sont évaluées avec les métriques présentées dans ce chapitre. Nous montrons également la faisabilité pratique de notre système d'exploration autonome SplatPlanner [3] lors d'un vol réel avec un drone quadrirotor.

Chapitre 4

Expériences et Résultats

4.1 Introduction

Nous venons de présenter FLYBO, un environnement de simulation étudié pour tester et évaluer les méthodes d'exploration autonome utilisant un drone quadrirotor et une caméra de profondeur. Dans ce chapitre, nous présentons les résultats d'expériences réalisées au cours des recherches ayant mené à cette thèse.

Ces expériences permettent, dans un premier temps, d'évaluer et de comparer en simulation les sept méthodes d'exploration autonome de l'état de l'art inclus dans FLYBO : Classic [2], Rapid [69], NBVP [116], AEP [118], ESM$_{IPP}$ [117], ESM$_{Recon}$ [117] et SplatPlanner [3]. Notez que le temps de calcul total requis pour évaluer rigoureusement les 7 méthodes considérées sur nos 11 scènes a nécessité 122,5 heures de calcul sans interruption. Cette évaluation a nécessité préalablement de correctement paramétrer et vérifier le bon fonctionnement de chaque méthode sur chaque scène. En termes de jeu de données et de métriques, les expériences en simulation proposées dans ce chapitre sont au moins équivalentes, voire supérieur (en termes de quantité) aux travaux les plus rigoureux du domaine (e.g [69, 117, 118, 168]). Les environnements tels que Office C ou Facility C sont significativement plus complexes et réalistes que ceux habituellement utilisés par les méthodes parues récemment [117, 118, 166, 168] ce qui rend nos évaluations plus complètes.

Dans la seconde partie de ce chapitre, nous décrivons la réalisation d'une exploration autonome faite en condition de vol réel avec un drone quadrirotor. Nous présentons les ré-

sultats d'exploration obtenus (carte d'occupation, trajectoire empruntée et maillages produits). Cette expérience permet de tester l'usage pratique de SplatPlanner [3].

4.2 Évaluation des méthodes d'exploration autonome en environnement simulé

Les méthodes listées en introduction de ce chapitre sont évaluées sur deux tâches : (i) la tâche d'exploration volumique et (ii) la tâche de reconstruction de surfaces en ligne. Nous ajoutons aussi une comparaison qualitative supplémentaire sur la tâche d'exploration volumique entre SplatPlanner [3] et FFI [166] une méthode qui n'est pas dans FLYBO. Mis à part cette comparaison qualitative, SplatPlanner et les autres méthodes intégrées à FLYBO sont évaluées équitablement grâce à un protocole expérimental identique et maitrisé. Avec FLYBO, nous simulons le comportement d'un drone quadrirotor instrumenté d'une caméra de profondeur et d'un capteur d'odométrie. Les deux capteurs sont étalonnés et rigidement attachés au drone. L'objectif est d'explorer et de reconstruire entièrement les scènes du jeu de données présenté dans le Chapitre 3. Les performances des méthodes d'exploration sont mesurées avec les métriques décrites dans le Chapitre 3. Nous les listons ci-dessous :

1. **Pour l'exploration volumique, les métriques sont :**
   - efficacité d'exploration volumique,
   - complétude d'exploration volumique.
2. **Pour la reconstruction de surfaces en ligne, les métriques sont :**
   - efficacité en termes de rappel géométrique (Rappel),
   - justesse (F-score) et erreur (RMSE) de reconstruction géométrique.

Pour tenir compte de la nature stochastique des méthodes et de la simulation, les expériences sont exécutées plusieurs fois (5 par méthodes et par scène) et les métriques sont moyennées. C'est une pratique courante dans le domaine [3, 69, 118, 166]. Les métriques d'efficacité sont présentées dans des graphiques qui montrent respectivement le volume exploré et le rappel en fonction du temps. Les trois autres métriques sont présentées dans des tableaux, dans lesquels, pour chaque méthode, nous donnons le score obtenu par scène moyenné sur les cinq exécutions, la moyenne globale des scores obtenus sur l'ensemble des expériences et le rang moyen. Le rang moyen est simplement la moyenne des classements d'une méthode obtenus sur l'ensemble des expériences pour une métrique donnée.

Dans un premier temps, nous analysons les résultats des expériences obtenus en classant ces résultats par métrique. Dans un second temps, nous proposons une synthèse comparative générale, nous permettant de conclure sur un classement qualitatif des méthodes d'explorations étudiées.

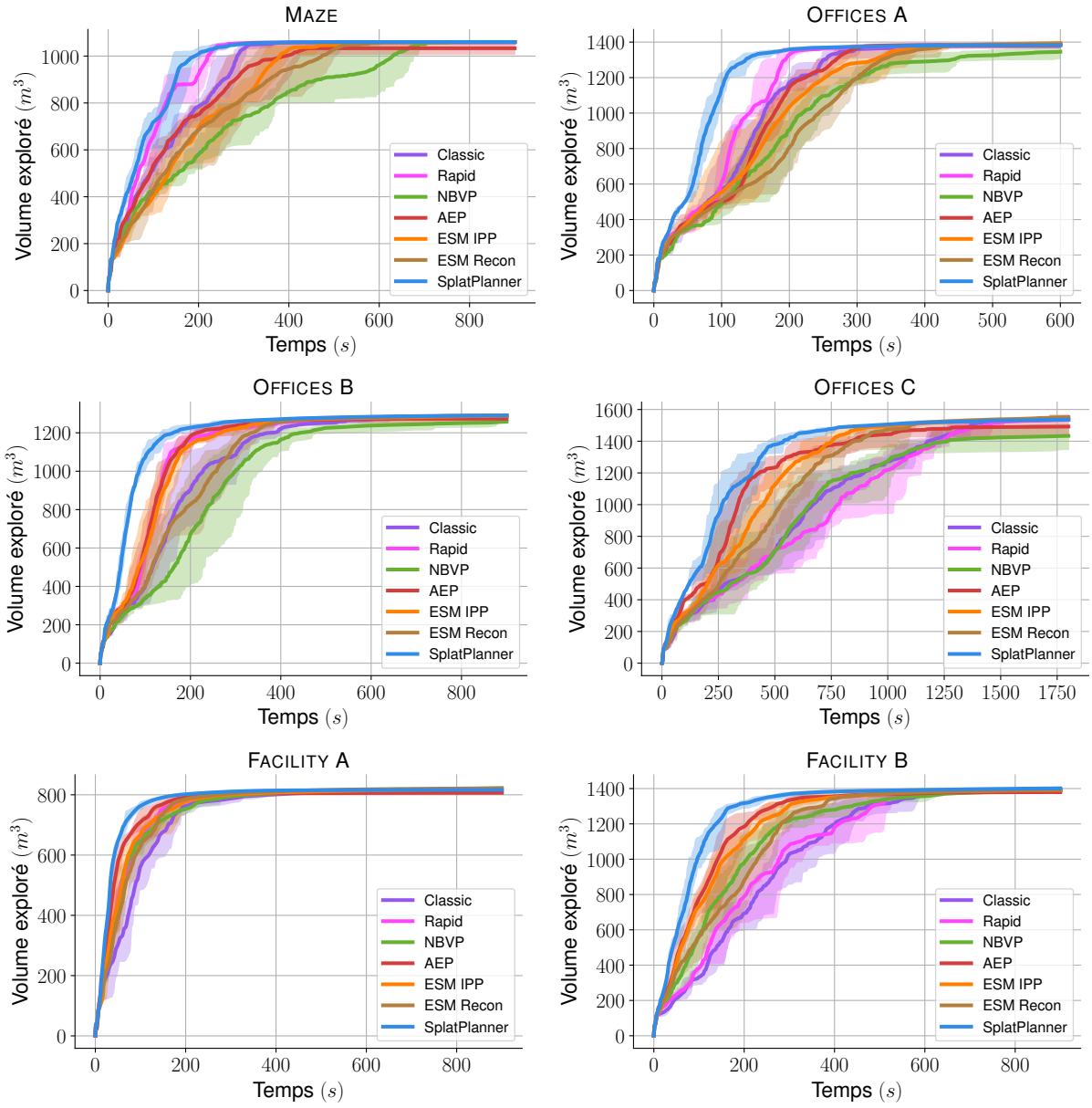


FIGURE 4.1 – **Efficacité d'exploration volumique.** Les graphiques de cette Figure représentent la quantité de volumes exploré (en m^3), par les méthodes d'exploration, en fonction du temps (en s) moyenné sur 5 exécutions par scène. Les noms des scènes sont spécifiés au-dessus des graphiques. L'intervalle coloré correspond au volume exploré minimum et maximum obtenus sur les 5 exécutions.

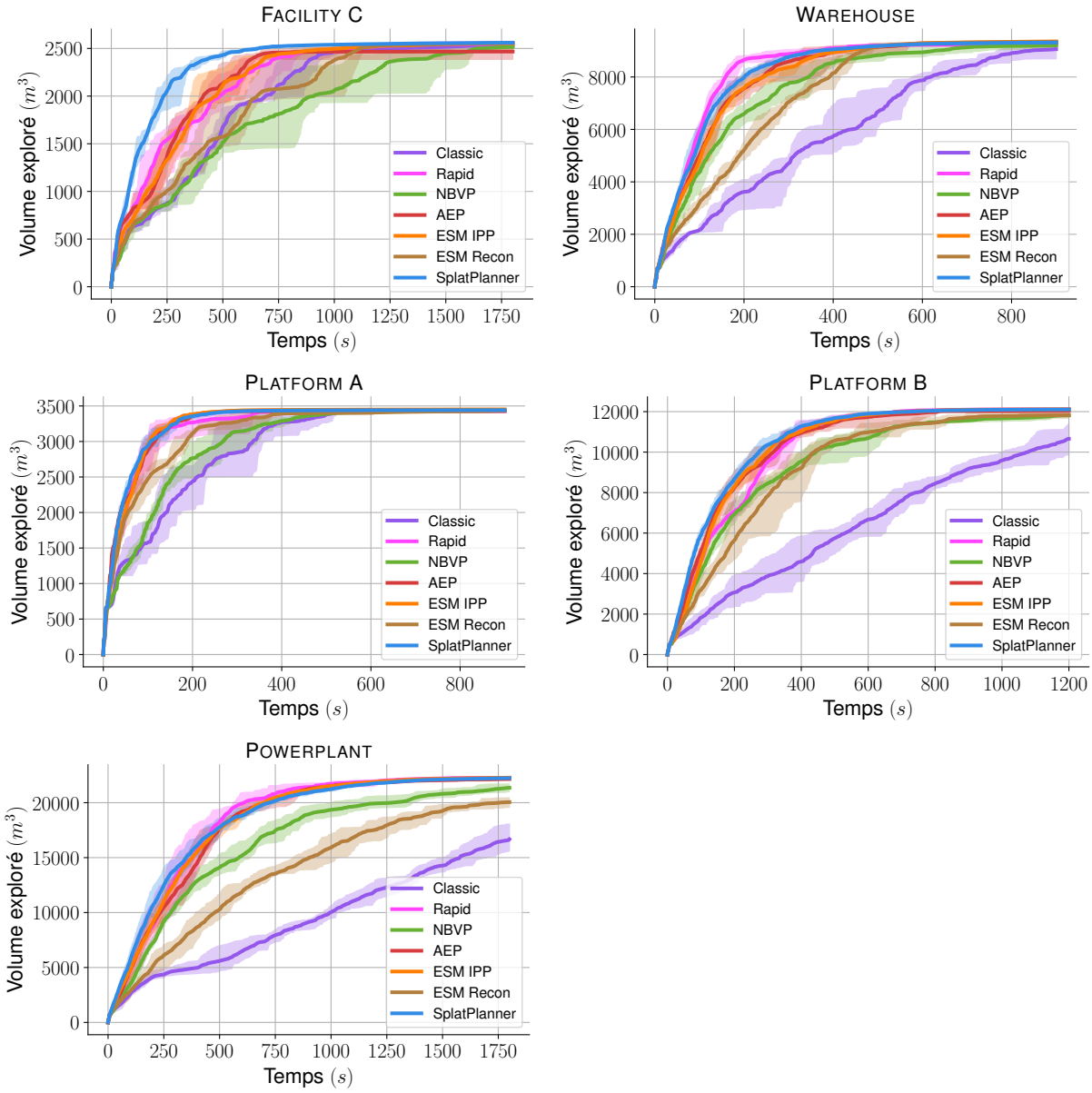


FIGURE 4.2 – **Efficacité d'exploration volumique.** Les graphiques de cette Figure représentent la quantité de volumes exploré (en m^3), par les méthodes d'exploration, en fonction du temps (en s) moyenné sur 5 exécutions par scène. Les noms des scènes sont spécifiés au-dessus des graphiques. L'intervalle coloré correspond au volume exploré minimum et maximum obtenus sur les 5 exécutions.

4.2.1 Efficacité d'exploration volumique

Dans cette sous-section, nous discutons des résultats obtenus sur la métrique d'efficacité d'exploration volumique. La première tendance qui se dégage des données présentées Figure

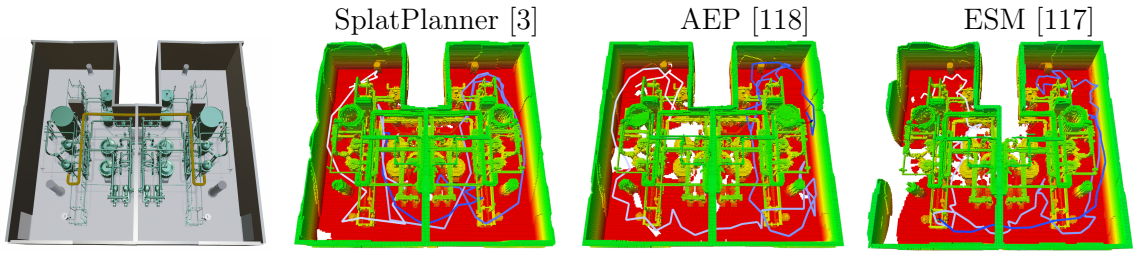
4.1 et Figure 4.2 est que certaines méthodes maintiennent des performances régulières, c'est-à-dire équivalentes sur chaque scène. C'est le cas pour les méthodes suivantes : Splatplanner [3], AEP [118], NBVP [116] et ESM$_{IPP}$ [117].

Classic [2] et Rapid [69] présentent des performances irrégulières et montrent parfois des difficultés significatives sur certaines scènes. Les difficultés de Classic [2] sont visibles sur les scènes très volumineuses ayant de grandes variations d'altitude (e.g. Warehouse, Platform A–B, Powerplant). Ces difficultés se caractérisent par une différence notable d'efficacité (e.g. Figure. 4.2 – Warehouse, Platform B, Powerplant). Les contraintes de perception (champ de vision frontal) ont tendance à produire des frontières directement au-dessus ou au-dessous du drone. Ces frontières sont priorisées par Classic [2] mais sont souvent peu utiles pour enrichir la carte d'occupation en construction. Un autre facteur réduisant les performances de Classic [2] dans les scènes volumineuses aussi, mentionné par [69], concerne l'algorithme de Dijkstra [68]. La complexité de cet algorithme pour trouver un chemin est directement corrélée à la taille de l'environnement et la résolution de la carte d'occupation. En dépit de ces faiblesses, Classic [2] présente des performances compétitives sur les autres scènes par rapport aux aux méthodes concurentes, et notamment des résultats sensiblement meilleurs que AEP [118] et ESM$_{IPP}$ [117] sur l'environnement Maze et Office A.

Les résultats montrent que Rapid [69] est en difficulté en présence d'espaces étroits et complexes (de nombreux objets) comme c'est le cas dans les scènes Office C et Facility B. Dans de telles scènes, l'algorithme Rapid [69] a tendance à alterner trop fréquemment entre la méthode d'exploration traditionnelle de Classic [2] et l'heuristique qui maintient une vitesse constante. Typiquement, sur Office C, Rapid [69] est parfois moins efficace que Classic [2] en rapidité d'exploration. À l'inverse, Rapid [69] est avantagé dans les espaces volumineux (e.g. Warehouse, Platform A et Powerplant) ou en présence de longs couloirs (e.g. Maze). Dans ces types de scènes, la méthode permet au drone de naviguer à vitesse maximale sans difficulté ni changement fréquent de stratégie. Sur les scènes suivantes : Warehouse, Platform A, Powerplant et Maze, l'efficacité d'exploration volumique de Rapid [69] se positionne comme l'une des meilleures méthodes d'exploration.

ESM$_{Recon}$ [117] utilise une fonction gain qui priorise en premier les voxels proches des surfaces déjà reconstruites pour les améliorer tandis que les autres méthodes cherchent à enrichir au plus vite la carte d'occupation en construction. En raison de l'usage de cette fonction de gain, ESM$_{Recon}$ [117] montre une efficacité d'exploration volumique en général inférieure à celle des autres méthodes. Ceci est visible en particulier sur des scènes très volumineuses et plutôt constituées d'espace libre (e.g Warehouse et Powerplant).

NBVP [116], AEP [118] et ESM$_{IPP}$ [117] ont la particularité d'obtenir des progressions d'exploration régulière quelle que soit la scène. À première vue NBVP [116] est la moins efficace des trois, pénalisée par le temps de calcul de son algorithme que nous trouvons un peu trop lent en termes de temps de calcul (800 ms). AEP [118] et ESM$_{IPP}$ [117] ont des résultats sensiblement identiques avec parfois une évolution du volume exploré en fonction du temps légèrement plus rapide pour AEP [118]. De ce fait AEP [118] semble légèrement plus efficace sur cette métrique qu'ESM$_{IPP}$ [117]. Toutefois, sur les graphiques représentés



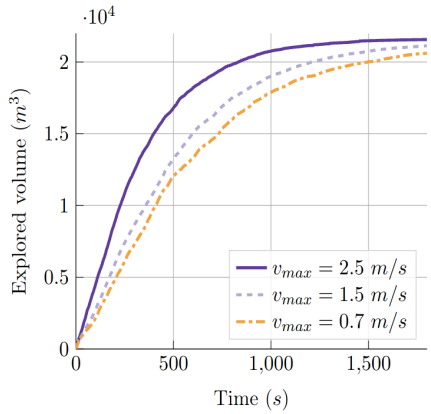
(a) Model CAO. (b) Cartes produites à $t = 300s$. Les voxels sont colorés en fonction de leur altitude.

FIGURE 4.3 – SplatPlanner [3] explore l'environnement complexe de FACILITY B plus efficacement et en relevant plus de détails géométriques que les alternatives de l'état de l'art.

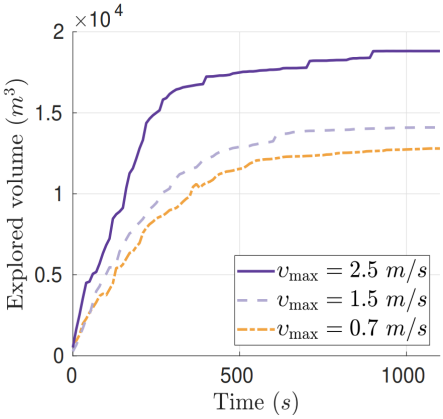
sur les Figures 4.1 et 4.2 nous pouvons observer qu'ESM$_{IPP}$ [117] obtient d'excellents scores finaux d'exploration tandis que AEP [118] et NBVP [116] sont souvent en dessous du meilleur score.

SplatPlanner [3] présente de manière significative et constante des performances en rapidité d'exploration équivalente (c.f. Figure 4.1 et Figure 4.2 – MAZE, WAREHOUSE, PLARFORM A-B, POWERPLANT) voire supérieure (c.f. Figure 4.1 et Figure 4.2 – OFFICE A-C, FACILITY A-C) aux méthodes concurrentes dans cette évaluation. Pour illustrer ce propos, nous présentons Figure 4.3(a) les trajectoires empruntées ainsi que la cartographie voxélique de FACILITY B après 290 s d'expérience, réalisée par SplatPlanner [3], AEP [118] et ESM$_{IPP}$ [117] (les trois meilleures méthodes selon nos résultats sur cette scène). Nous pouvons observer sur la Figure 4.3(b) que SplatPlanner [3] produit une carte d'occupation plus complète qu'AEP [118] et ESM$_{IPP}$ [117]. D'autres comparaisons de SplatPlanner [3] avec ces deux méthodes sont disponibles dans une vidéo que vous pouvez visualiser à l'adresse suivante : `https://youtu.be/DCcfA2HB1GI?t=92`.

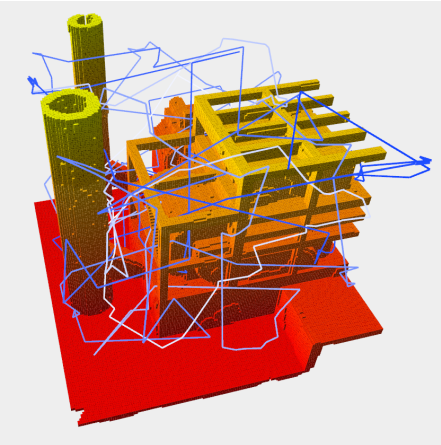
À ces expériences, nous ajoutons une comparaison qualitative de SplatPlanner [3] et de FFI [166]. En l'absence d'implantation publique de FFI [166], nous présentons Figures 4.4(b) et 4.4(d) les performances et les illustrations rapportées par les auteurs tandis que les Figures 4.4(a) et 4.4(c) sont les résultats de SplatPlanner [3]. La comparaison de ces quatre Figures montre une évolution de la courbe du volume exploré en fonction du temps pour SplatPlanner [3] plus rapide que FFI [166] pour les différentes vélocités (c.f. Figure 4.4(a), 4.4(b)), excepté sur les 500 premières secondes où FFI [166] semble légèrement plus rapide. Le faible écart sur les 500 premières secondes peut s'expliquer par une différence d'optimisation des vitesses du drone le long des trajectoires. Notez que les auteurs n'ont rien spécifié quant à la méthode de calculs des trajectoires paramétriques qu'ils ont employée. Celle-ci peut être meilleure que celle utilisée dans FLYBO [82]. Mais, au-delà de cette limite, les résultats que nous obtenons sont en faveur de SplatPlanner [3] qui réalise à une efficacité d'exploration volumique supérieure après les 500 premières secondes d'exploration. Cette tendance est également confirmée qualitativement par des trajectoires mieux réparties dans l'espace exploré et une carte d'occupation plus complète



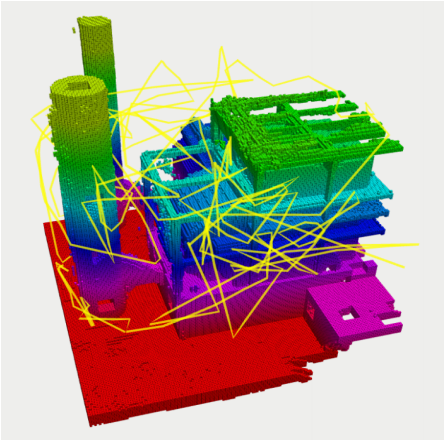
(a) SplatPlanner [3] **(Notre méthode)**.



(b) FFI [166] (Capture de leur publication).



(c) SplatPlanner [3] **(Notre méthode)**.



(d) FFI [166] (Capture de leur publication).

FIGURE 4.4 – Comparaison de SplatPlanner [3] avec FFI [166] sur la scène POWERPLANT en termes d'efficacité d'exploration volumique (a–b), de trajectoires correspondantes et cartes d'occupation produites (c–d). SplatPlanner [3] réalise une exploration plus rapide après 500 s Il suit une trajectoire plus simple et produit une carte d'occupation plus complète sur l'ensemble des vitesses maximales considérées.

(Figure 4.4(c), 4.4(d)).

4.2.2 Complétude d'exploration volumique

Les résultats de complétude d'exploration volumique sont rapportés sur le tableau référencé Table 4.1. ESM$_{IPP}$ [117] obtient le meilleur rang moyen avec 2.2. Nous justifions cette performance par l'efficacité d'exploration suffisante d'ESM$_{IPP}$ [117] lui permettant de compléter entièrement l'exploration des scènes dans les temps impartis et par la structure arborescente utilisée qui couvre entièrement l'espace navigable. Cette stratégie semble encore plus avantageuse par rapport aux autres méthodes dans des espaces plutôt volumineux tels que Warehouse, Platform B et Powerplant car elle permet d'optimiser très efficacement les chemins d'explorations.

Le score d'ESM$_{IPP}$ [117] est suivi par celui de SplatPlanner [3] et de ESM$_{Recon}$ [117] avec respectivement un rang moyen de 2.6 et 2.9. Fondamentalement ESM$_{IPP}$ [117] et ESM$_{Recon}$ [117] ont la même stratégie d'exploration excepté pour la fonction de gain (l'un utilise le volume inconnu, l'autre priorise les surfaces). Cette différence ralentit ESM$_{Recon}$ [117] dans son exploration et par conséquent réduit son score sur les scènes volumineuses. Quand il reste peu de volume non exploré, SplatPlanner [3] peut construire des trajectoires moins efficaces, pénalisant partiellement la méthode sur la métrique de complétude par rapport à la stratégie d'ESM$_{IPP}$ [117].

Parmi toutes les méthodes considérées dans cette évaluation NBVP [116] et AEP [118] obtiennent les scores les plus faibles (rang moyen respectif de 5.8 et 5.5). Le principal problème de ces deux méthodes est la priorité mise sur l'exploration volumique par rapport à la qualité du résultat final de l'exploration. Dans leur fonctionnement AEP [118] et NBVP [116] ne considèrent parfois pas les régions avec peu de voxels inconnues pour accélérer l'exploration. Les conséquences de cette "optimisation" sont des régions mal explorées. Le problème de terminaison prématurée est un facteur supplémentaire qui réduit les résultats d'NBVP [116] (c.f. Table 4.1 – Office C). Nous ajoutons aussi que le temps pris par NBVP [116] entre chaque calcul d'une trajectoire d'exploration réduit ses résultats de complétude d'exploration volumique vu que la méthode ne termine pas l'exploration dans le temps imparti (c.f. Table 4.1 – Powerplant, Platform B).

		Classic [2]	Rapid [69]	NBVP [116]	AEP [118]	ESM$_{IPP}$ [117]	ESM$_{Recon}$ [117]	SplatPlanner [2]	**Plateau (s)**
Complétude Vol.	Maze	0.9992	0.9988	0.9925	0.9750	0.9997	**1.0**	0.9998	700
	Offices A	0.9945	0.9940	0.9648	0.9911	**1.0**	0.9953	0.9923	600
	Offices B	0.9957	0.9987	0.9685	0.9882	0.9962	0.9988	**1.0**	750
	Offices C	0.9926	0.9912	0.9258	0.9656	0.9976	**1.0**	0.9935	1650
	Facility A	0.9868	0.9900	0.9973	0.9836	0.9961	**1.0**	0.9959	700
	Facility B	0.9911	0.9932	0.9866	0.9863	0.9925	0.9927	**1.0**	750
	Facility C	0.9883	0.9950	0.9774	0.9642	0.9971	0.9974	**1.0**	1650
	Warehouse	0.9681	0.9861	0.9834	0.9962	**1.0**	0.9974	0.9952	900
	Platform A	0.9933	0.9934	0.9887	0.9939	**1.0**	0.9863	0.9976	550
	Platform B	0.8795	0.9995	0.9748	0.9930	**1.0**	0.9758	0.9990	1200
	Powerplant	0.7484	0.9999	0.9587	0.9945	**1.0**	0.9001	0.9984	1800
	Moyenne	0.9579	0.9945	0.9744	0.9847	**0.9981**	0.9858	0.9974	
	Rang	5.3	3.7	5.8	5.5	**2.2**	2.9	2.6	

Table 4.1 – **Complétude d'exploration volumique.** Score pris au plateau des courbes de volume exploré en fonction du temps, relatif au meilleur score obtenu (moyenné sur 5 expériences).

Classic [2] obtient des résultats supérieurs à ceux d'AEP [118] et NBVP [116] ce qui est une performance compte tenu de la simplicité de l'algorithme. Les résultats de Classic [2] chutent sur les scènes très volumineuses et ayant de fortes variations d'altitude (e.g Warehouse, Platform B, Powerplant). Ceci est expliqué par la faible efficacité d'exploration volumique de la méthode sur ce type d'environnement. Rapid [69] corrige l'inefficacité de Classic [2] grâce à son second mode de fonctionnement. Cet avantage se répercute sur le score obtenu par Rapid [69] sur cette métrique (3.7 pour Rapid [69] et 5.3 Classic [2]).

Sur la Figure 4.4(d) nous pouvons qualitativement observer que FFI [166] obtient des résultats inférieurs à ceux de SplatPlanner [3] . Les performances de cette méthode semblent fortement dépendantes du paramètre v_{max}. Selon les auteurs, cette faiblesse est due à la fonction de gain qu'ils utilisent. Sur les résultats rapportés, plus v_{max} est faible, plus FFI [166] a tendance à rester dans des régions à proximité de la position courante du drone. Ceci est problématique pour explorer des scènes volumineuses (e.g. Powerplant).

4.2.3 Efficacité en termes de rappel géométrique

Dans cette sous-section, nous discutons des résultats obtenus sur la métrique d'efficacité en termes de rappel de reconstruction géométrique. Les résultats de cette métrique sont présentés Figures 4.5 et 4.6. Comme pour la métrique d'efficacité en termes de volume exploré, nous pouvons observer que certaines méthodes maintiennent une évolution de rappel régulier quelle que soit la scène et particulièrement SplatPlanner [3], NBVP [116], AEP [118], ESM$_{IPP}$ [117] et ESM$_{Recon}$ [117]. Par opposition Classic [2] et Rapid [69] présentent des performances irrégulières, inférieures aux méthodes concurrentes sur certaines scènes telles que Warehouse, Platform B et Powerplant pour Classic [2] et Office C et Facility C pour Rapid [69]. Sur les autres scènes Classic [2] et Rapid [69] ont des performances compétitives par rapport aux autres méthodes. Rapid [69] possède la meilleure progression en termes de rappel géométrique sur la scène Warehouse.

Parmi les méthodes ayant des performances régulières, NBVP [116] est en général la méthode la moins performante, en raison des défauts mentionnés précédemment. SplatPlanner [3] obtient une progression de rappel en général meilleure que les méthodes ayant des performances régulières, en particulier sur les scènes Maze, Offices A, Offices B, Offices C, Facility A, Facility B, Facility C, Warehouse. Nous pouvons constater sur les Figures 4.5 et 4.6 qu'ESM$_{IPP}$ [117] et AEP [118] ont une évolution de rappel quasiment équivalente. L'évolution du rappel géométrique de NBVP [116], AEP [118], ESM$_{IPP}$ [117], Classic [2], Rapid [69] et SplatPlanner [3] semble être corrélé avec l'évolution de l'exploration volumique.

Concernant ESM$_{Recon}$ [117], sur certaines scènes, en particulier les plus volumineuses comme Platform B et Powerplant, cette méthode obtient une progression en termes de rappel géométrique supérieure à sa progression en termes de volume exploré. Sur ces deux scènes, pendant que les autres méthodes d'exploration dirigent le drone vers des régions inconnues, ESM$_{Recon}$ [117] se concentre majoritairement sur les surfaces de la scène. Ainsi, les performances en termes de rappel géométrique d'ESM$_{Recon}$ [117] sont équiva-

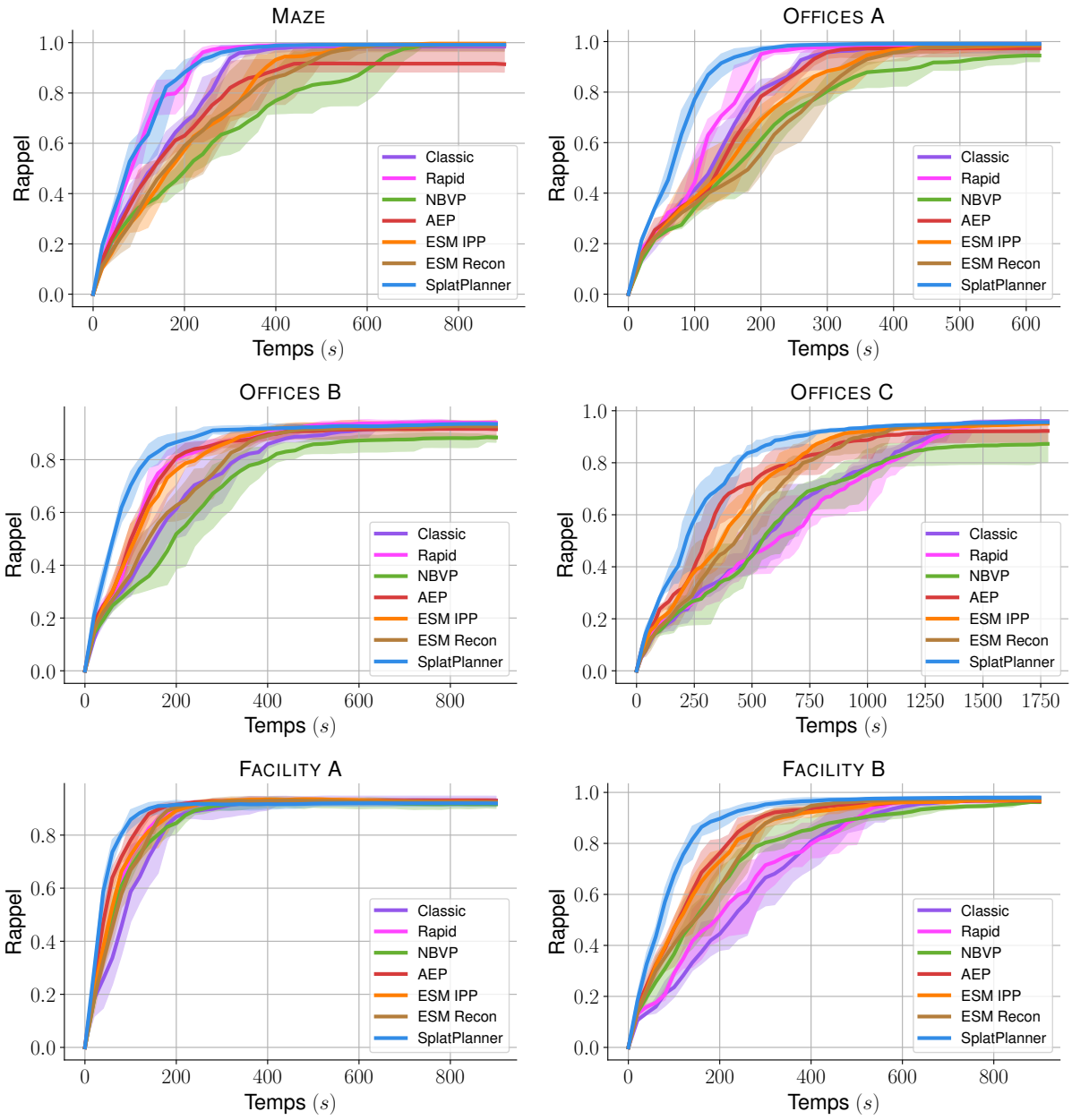


Figure 4.5 – **Efficacité en termes de rappel géométrique**. Les graphiques de cette Figure représentent le ratio de surfaces reconstruites, pour chaque méthode, en fonction du temps moyenné sur 5 exécutions par scène. Les noms des scènes sont spécifiés au-dessus des graphiques. L'intervalle coloré correspond au rappel minimum et maximum obtenus sur les 5 exécutions.

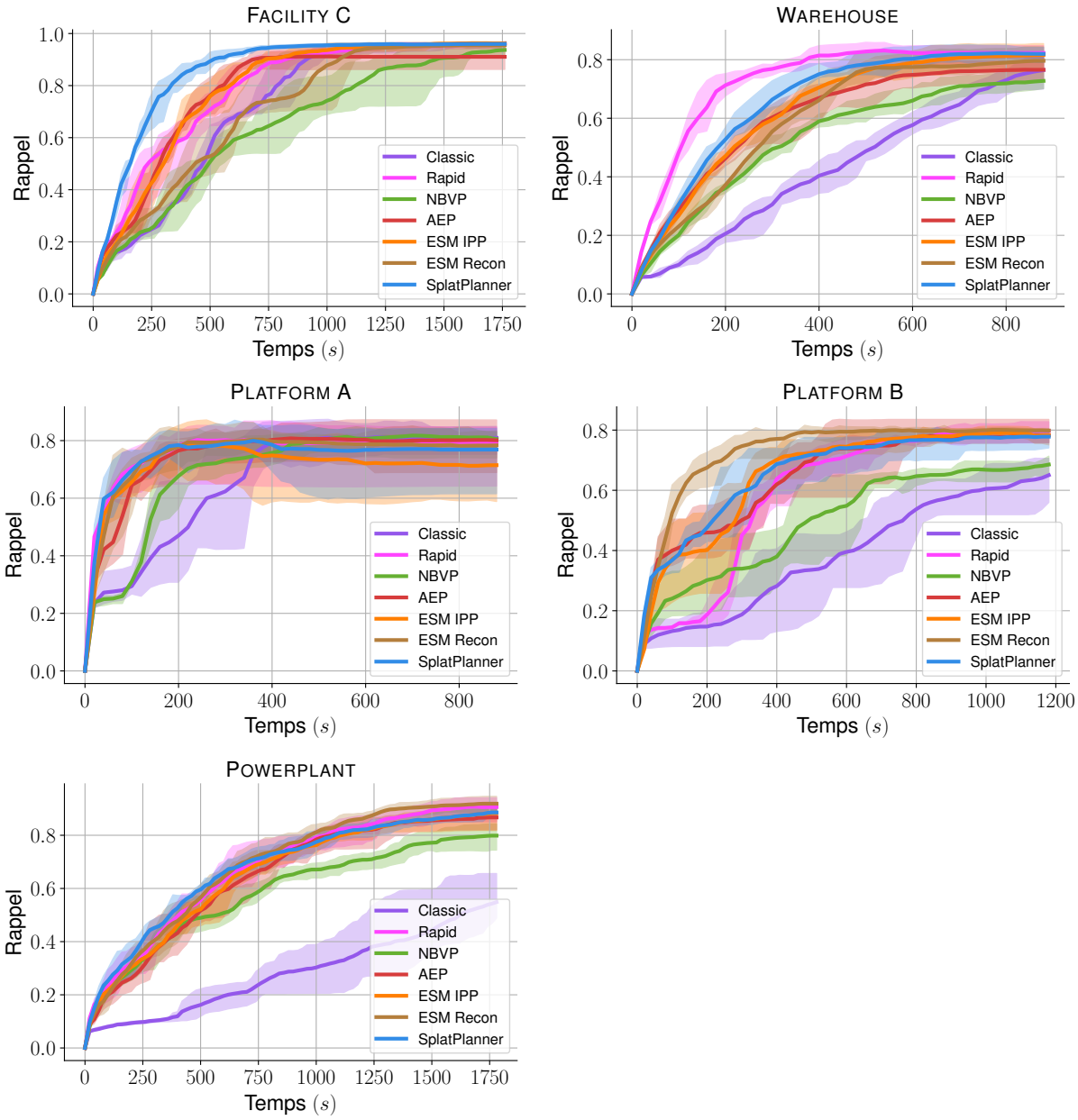


FIGURE 4.6 – **Efficacité en termes de rappel géométrique**. Les graphiques de cette Figure représentent le ratio de surfaces reconstruites, pour chaque méthode, en fonction du temps moyenné sur 5 exécutions par scène. Les noms des scènes sont spécifiés au-dessus des graphiques. L'intervalle coloré correspond au rappel minimum et maximum obtenus sur les 5 exécutions.

		Classic [2]	Rapid [69]	NBVP [116]	AEP [118]	ESM$_{IPP}$ [117]	ESM$_{Recon}$ [117]	SplatPlanner [2]
F-Score	Maze	0.9807	0.9831	0.9888	0.9362	**0.9944**	0.9862	0.9848
	Offices A	0.9792	**0.9905**	0.9609	0.9687	0.9820	0.9831	0.9891
	Offices B	0.9371	**0.9435**	0.9086	0.9182	0.9344	0.9355	0.9391
	Offices C	**0.9661**	0.9620	0.9160	0.9385	0.9575	0.9603	0.9597
	Facility A	0.9266	0.9257	0.9257	0.9132	0.9338	**0.9355**	0.9201
	Facility B	0.9459	0.9477	0.9351	0.9353	0.9429	**0.9468**	0.9527
	Facility C	0.9629	0.9574	0.9416	0.9214	**0.9613**	0.9538	0.9602
	Warehouse	0.7781	0.8097	0.7289	0.7581	0.8077	0.7927	**0.8118**
	Platform A	0.8111	0.7912	0.7780	0.7647	0.7028	**0.7965**	0.7576
	Platform B	0.6738	0.7597	0.7009	0.7616	0.7680	**0.7986**	0.7575
	Powerplant	0.6694	0.8673	0.8251	0.8484	0.8513	**0.8809**	0.8674
	Moyenne	0.8755	0.9034	0.8736	0.8786	0.8942	**0.9064**	0.9
	Rang	3.9	2.8	5.8	5.8	3.6	**2.7**	3.3
RMSE (m)	Maze	0.018 ± 0.014	0.021 ± 0.014	0.018 ± 0.013	0.023 ± 0.015	**0.016 ± 0.012**	0.018 ± 0.013	0.02 ± 0.014
	Offices A	0.019 ± 0.025	**0.018 ± 0.014**	0.019 ± 0.016	0.021 ± 0.016	**0.018 ± 0.015**	0.019 ± 0.015	**0.018 ± 0.015**
	Offices B	**0.021 ± 0.02**	**0.021 ± 0.019**	0.023 ± 0.022	0.024 ± 0.022	**0.021 ± 0.019**	0.022 ± 0.019	**0.021 ± 0.02**
	Offices C	**0.015 ± 0.015**	0.016 ± 0.015	0.017 ± 0.016	0.017 ± 0.017	0.016 ± 0.016	0.016 ± 0.015	0.016 ± 0.016
	Facility A	0.021 ± 0.021	0.022 ± 0.021	0.021 ± 0.02	0.026 ± 0.026	**0.019 ± 0.017**	0.02 ± 0.018	0.022 ± 0.022
	Facility B	0.02 ± 0.019	0.02 ± 0.021	0.021 ± 0.019	0.021 ± 0.02	0.02 ± 0.019	0.02 ± 0.019	**0.018 ± 0.017**
	Facility C	**0.02 ± 0.019**	**0.02 ± 0.018**	0.022 ± 0.02	0.025 ± 0.022	**0.02 ± 0.018**	0.021 ± 0.019	0.021 ± 0.018
	Warehouse	0.03 ± 0.034	0.029 ± 0.033	0.031 ± 0.032	0.032 ± 0.033	0.029 ± 0.032	0.029 ± 0.031	**0.028 ± 0.032**
	Platform A	**0.03 ± 0.032**	0.032 ± 0.034	0.037 ± 0.035	0.037 ± 0.037	0.036 ± 0.03	0.031 ± 0.029	0.033 ± 0.03
	Platform B	0.046 ± 0.046	0.043 ± 0.046	0.044 ± 0.045	0.043 ± 0.046	0.042 ± 0.045	**0.038 ± 0.044**	0.043 ± 0.045
	Powerplant	0.032 ± 0.028	0.034 ± 0.03	0.034 ± 0.027	0.034 ± 0.028	0.036 ± 0.03	**0.031 ± 0.027**	0.033 ± 0.028
	Rang	2.6	2.7	5.0	6.1	**2.3**	2.5	2.7

Table 4.2 – **Reconstruction de surface en ligne : Justesse et erreur de reconstruction géométrique** – Évaluation comparative des maillages produits au moyen du F-score et de la RMSE, moyennées sur 5 exécutions.

lentes, voire supérieures aux méthodes concurrentes sur ces deux scènes. L'illustration de la Figure .30 présente le résultat des surfaces produites par chaque méthode à différentes étapes de l'exploration de la scène Platform B. Dans cette illustration qualitative, nous pouvons observer qu'ESM$_{Recon}$ [117] a quasiment reconstruit toutes les surfaces de la scène à l'itération *Intermédiaire*, tandis que des surfaces sont manquantes pour toutes les méthodes concurrentes. Sur des scènes plus petites et plus étroites, la stratégie proposée par ESM$_{Recon}$ [117] ne montre pas sur la métrique d'efficacité de rappel géométrique d'amélioration particulière par rapport aux méthodes concurrentes.

4.2.4 Justesse et erreur de reconstruction géométrique

Le tableau Table 4.2 rapporte les résultats de justesse (F-score) et d'erreur de reconstruction géométrique (RMSE) de chaque méthode pour chaque scène simulée.

Concernant le F-Score, ESM$_{Recon}$ [117] obtient la meilleure performance suivie de Rapid [69], de SplatPlanner [3] et d'ESM$_{IPP}$ [117]. Le rang moyen de 3.9 obtenus par Classic [2] la situe un peu en dessous des méthodes précédemment citées. Cependant, nous précisons que le F-score de Classic [2] sur certaines scènes est affecté par ses performances de rappel surtout quand la méthode n'a pas suffisamment de temps pour terminer la reconstruction (e.g. Platform B, Poweplant). Cela affecte par conséquent son rang moyen. Sur

cette métrique NBVP [116] et AEP [118] ont des résultats plus faibles que les méthodes concurrentes.

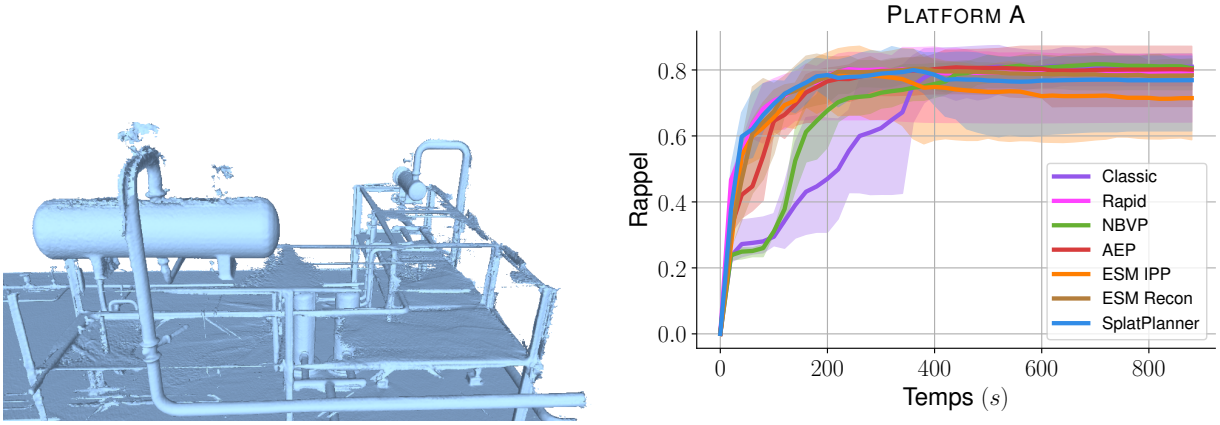
Concernant les résultats d'erreur de reconstruction géométrique (RMSE) Classic [2], Rapid [69], ESM_{IPP} [117], ESM_{Recon} [117] et SplatPlanner [3] obtiennent des résultats très similaires. Vu que la RMSE ne tient pas compte des scores de rappel, Classic [2] obtient un rang moyen (2.6) meilleur que son rang rapporté sur le F-score. AEP [118] et NBVP [116] ont des erreurs de reconstruction plus élevée que les méthodes concurrentes.

En général, la justesse et l'erreur de reconstruction géométrique d'un maillage reconstruit avec les valeurs de TSDF et l'algorithme dit du "Marching Cube" s'améliorent quand les surfaces produites sont observées : (i) de près, (ii) plusieurs fois et (iii) sous plusieurs angles. Le premier motif d'amélioration (i) s'explique car la variation aléatoire que nous ajoutons sur les valeurs de profondeur augmente en fonction de la distance des mesures. Les deux autres motifs d'amélioration, (ii) et (iii), permettent d'obtenir des valeurs de TSDF plus précises en moyennant les nombreuses mesures acquises. ESM_{Recon} [117] tient compte de ses motifs d'amélioration en calculant des trajectoires qui renforcent les voxels à proximités de surfaces déjà reconstruites. Ce qui explique un F-score élevés et une erreur de RMSE faible pour cette méthode en comparaison des méthodes concurrentes, en particulier sur les scènes Platform B et Powerplant.

La qualité des résultats de F-score et de RMSE obtenus par Rapid [69], Classic [2] et SplatPlanner [3] peuvent s'expliquer par notre implémentation qui oblige le drone à se rapprocher des frontières (en général moins de 2 m). Par conséquent, le drone se retrouve souvent à proximité de surfaces situées juste derrière les frontières. Notons tout de même que ESM_{IPP} [117] n'applique pas réellement ces trois motifs d'amélioration mais obtient des résultats équivalents aux trois méthodes. Nous pensons que cette performance est en partie due à la structure arborescente utilisée, qui couvre de manière dense, par des trajectoires possibles, l'espace navigable.

NBVP [116] et AEP [118] ont des performances en termes de F-score et RMSE inférieures aux méthodes concurrentes. Comme pour la métrique de complétude volumique, la priorité mise par ces méthodes sur l'efficacité d'exploration volumique et les problèmes de terminaisons prématurées engendrent des surfaces manquantes ou mal reconstruites. Nous proposons sur la Figure 4.9 quelques résultats qualitatifs produits à la fin de l'exploration par NBVP [116] et AEP [118] en regard de ceux obtenus par SplatPlanner [3]. Sur cette figure nous pouvons observer des surfaces manquantes sur les résultats produits par NBVP [116] et AEP [118].

Limitation : En général, en reconstruction 3D et notamment avec l'usage de la TSDF, il est difficile de produire des maillages précis et complets de structures géométriques fines. Nous montrons un exemple typique du problème engendré par des structures trop fines sur la Figure 4.7(a). Sur cette illustration (Figure 4.7(a)) le sol du dernier étage de la structure n'est pas reconstruit. Lors des expériences, les valeurs de TSDF sont constamment changées quand le drone passe en dessous puis au-dessus du sol. L'épaisseur du sol n'est pas suffisante compte tenu de la résolution de la TSDF que nous avons choisie pour l'ensemble des expériences. Vous pouvez observer sur le graphique Figure 4.7(b) un écart de rappel



(a) (b)

FIGURE 4.7 – Limitations de la TSDF pour reconstruire les surfaces fines. Impact sur le maillage et le rappel : (a) le sol d'un étage de la structure n'est pas reconstruit et (b) intervalle min. et max. variant beaucoup plus que sur les autres scènes signifiant que les surfaces sont constamment ajoutées et supprimées.

minimum et maximum (intervalle coloré) plus important que sur les autres scènes.

La reconstruction de structure fine est un problème ouvert. Nous avons volontairement laissé de côté ce problème, laissant sa résolution éventuelle pour de futures améliorations concernant FLYBO et surtout la tâche de reconstruction 3D.

4.2.5 Synthèse comparative des performances de l'état de l'art

Si nous tentons de synthétiser les résultats en F-score et en complétude d'exploration volumique, ESM_{IPP} [117], ESM_{Recon} [117] et SplatPlanner [3] sont, au sens de l'optimum de Pareto, les méthodes les plus performantes. ESM_{IPP} [117] est le meilleur choix en complétude d'exploration volumique et ESM_{Recon} [117] en F-score. SplatPlanner [3] émerge comme le meilleur compromis entre les deux métriques considérées. Sur les métriques d'efficacité en termes de volume exploré et de rappel géométrique en fonction du temps, les performances de SplatPlanner [3] sont équivalentes, voire meilleures que les méthodes concurrentes.

En conclusion, selon les résultats et analyse rapportés dans cette section, SplatPlanner [3] se positionne comme la méthode la plus performante parmi les méthodes considérées. Toutefois, suivant le cas d'usage, les autres méthodes peuvent être privilégiées. Par exemple, si le temps d'exploration est suffisant, ESM_{IPP} [117] possède la meilleure stratégie pour couvrir l'ensemble de l'espace explorable. ESM_{Recon} [117] est moins efficace mais produit des surfaces plus précises et un maillage plus complet (F-score) que les autres méthodes. Classic [2] est une méthode pertinente quand le drone navigue horizontalement (déplacement surtout 2D), mais ses performances se réduisent dès que l'exploration nécessite des variations d'altitude. Rapid [69] corrige ce problème et est parfois la méthode la plus efficace

quand le drone peut naviguer librement (e.g WAREHOUSE). Cependant, les performances de Rapid [69] chutent quand le drone navigue dans des espaces étroits (e.g. OFFICES C). AEP [118] explore efficacement les scènes, mais les résultats finaux d'exploration et de reconstruction sont peu précis et souvent incomplets. NBVP [116] est moins efficace qu'AEP [118] et possède les mêmes défauts, cependant sa simplicité constitue, comme Classic [2] et Rapid [69], des méthodes de base intéressantes pour le développement, l'amélioration et la comparaison de futures stratégies d'exploration.

Vous trouverez quelques résultats qualitatifs de maillages produit Figure 4.11 à différentes étapes de l'exploration par les méthodes évalué dans nos expériences. Les résultats de maillage sont extraits au début, au milieu et à la fin de l'exploration. Nous avons aussi ajouté une vue immersive.

Les paramètres utilisés dans ces expériences sont résumés en Annexes 4.4. Des résultats additionnels y sont aussi donnés.

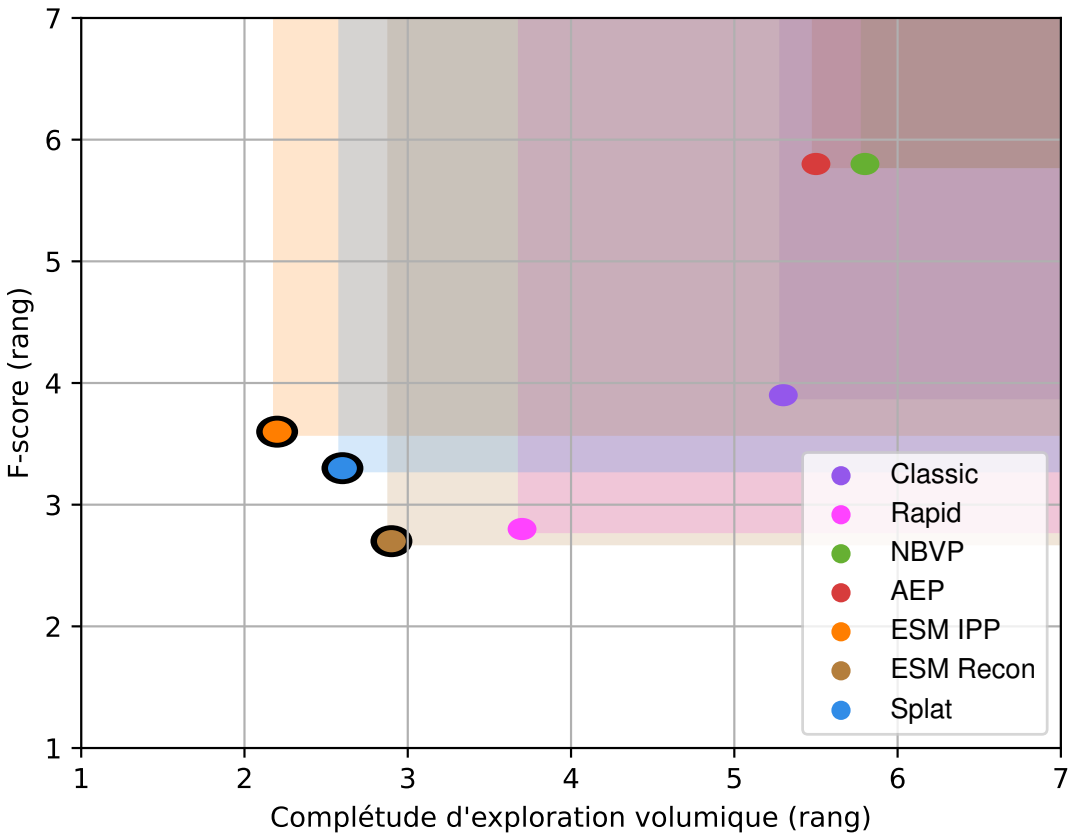
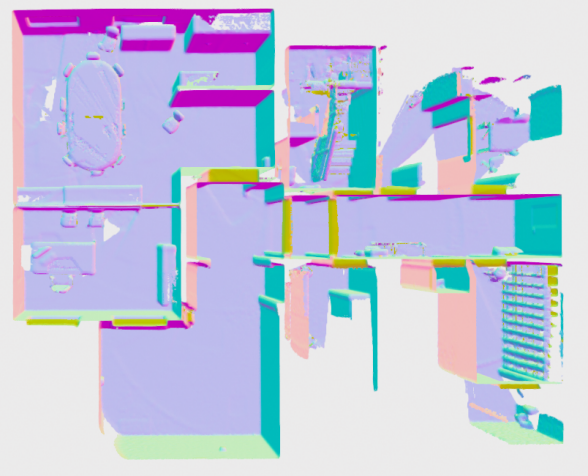
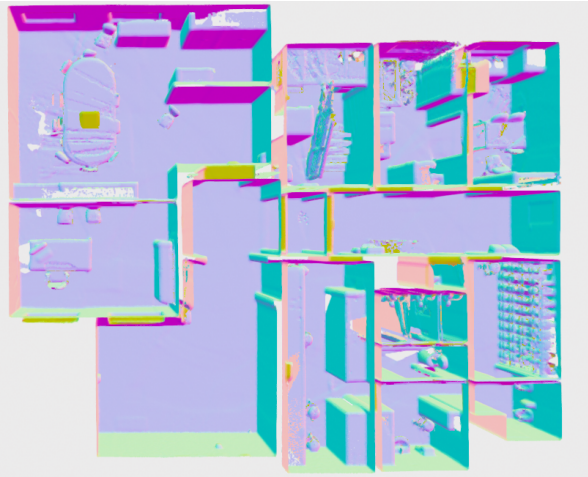


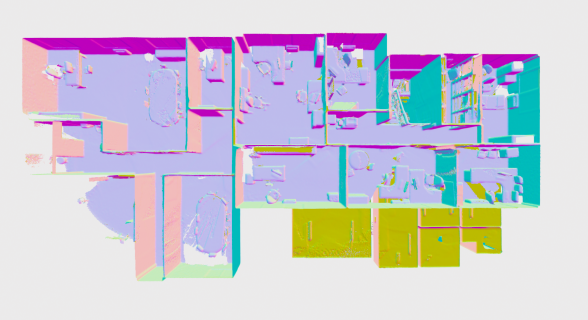
FIGURE 4.8 – Résumé de nos résultats de référence montrant les méthodes optimales selon le front de Pareto (**en cerclées en noir**) par rapport à leur classement conjoint de complétude d'exploration volumique et de précision de reconstruction en ligne (F-score) sur FLYBO.



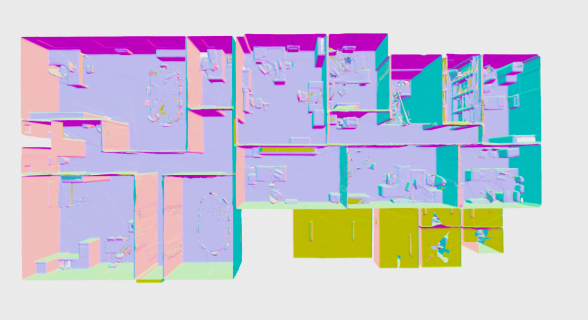
(a) Office C - NBVP [116].



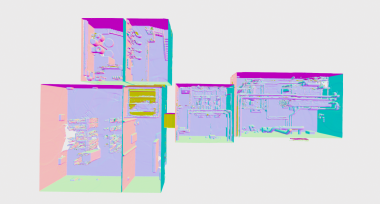
(b) Office C - SplatPlanner [3].



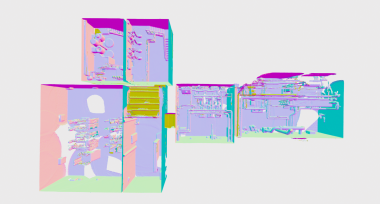
(c) Office C - AEP [118].



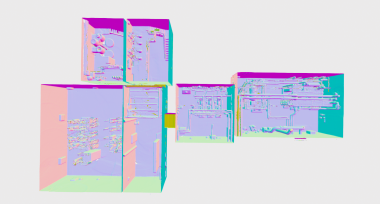
(d) Office C - SplatPlanner [3].



(e) Facility C - NBVP [116].



(f) Facility C - AEP [118].



(g) Facility C - Splat. [3].

Figure 4.9 – Quelques résultats qualitatifs qui présentent des maillages reconstruits en ligne.

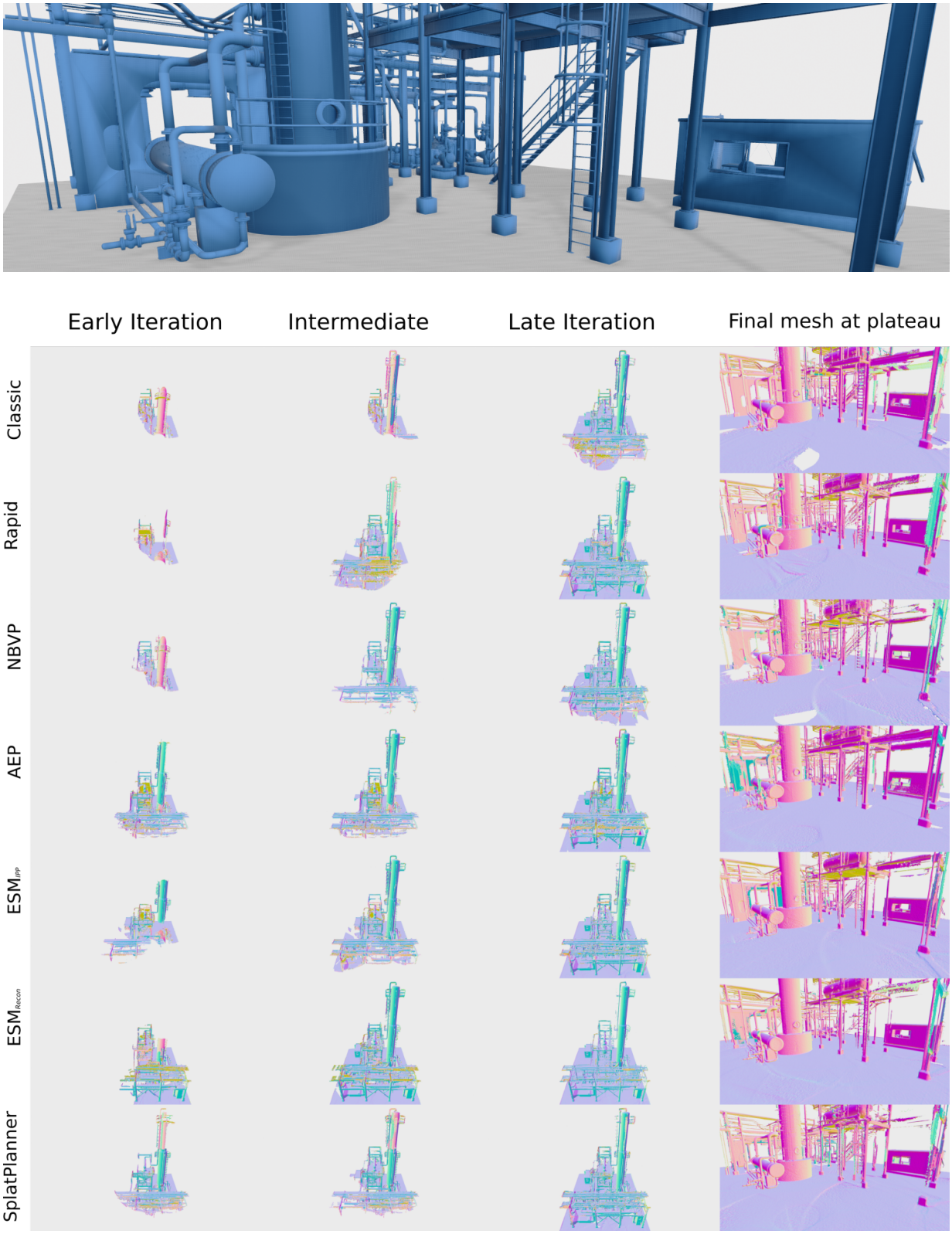


FIGURE 4.10 – Reconstructions progressives de l'environnement PLATFORM B (illustré en modèle CAO ligne du haut) en vue immersive.

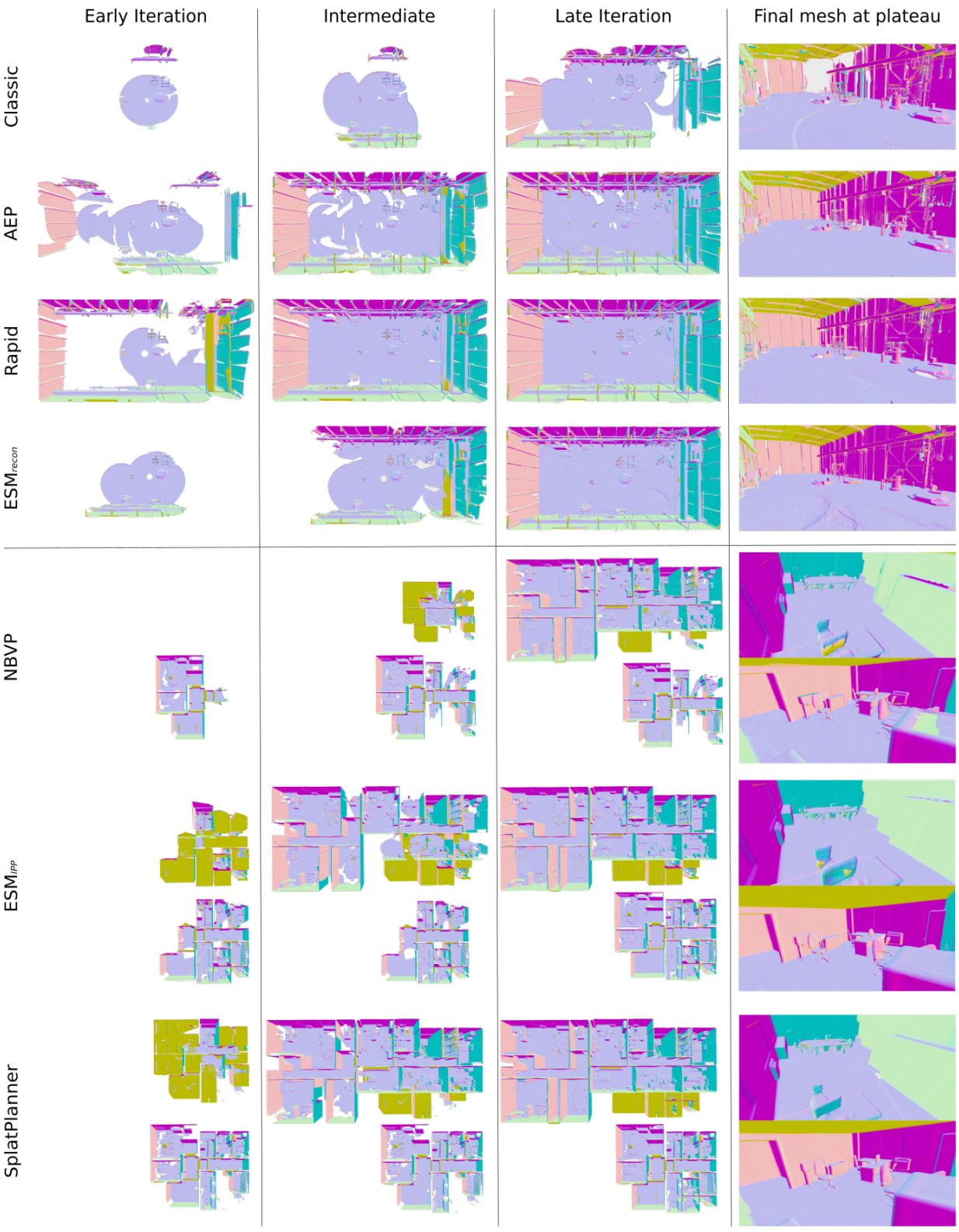


FIGURE 4.11 – **Quelques résultats qualitatifs**– Les quatre rangées supérieures montrent les résultats obtenus dans WAREHOUSE, tandis que les trois rangées inférieures montrent les bureaux des deux étages d'OFFICE C à différents stades d'exploration, en utilisant différentes méthodes..

4.3 Expérience en conditions de vol réel

Dans cette section, nous présentons une expérience d'exploration autonome avec Splat-Planner [3] en condition de vol réel avec un drone quadrirotor.

Nous commençons cette section par une description du robot utilisé, des algorithmes, des étapes d'étalonnage effectuées et des conditions expérimentales. Ensuite, nous présentons et commentons les résultats obtenus.

4.3.1 Description du drone utilisé et des conditions expérimentales

FIGURE 4.12 – Photographie du drone lors de l'expérience dans l'entrepôt industriel de Gambi-M

Description du drone utilisé. Le drone quadrirotor utilisé est équipé d'un contrôleur de vol DJI A3 [i], instrument qui contrôle la mise sous tension de certains équipements électroniques du drone (e.g. moteur). Le contrôleur de vol choisit la vitesse de rotation des moteurs en fonction d'un comportement désiré (e.g. vol stationnaire) ou d'un ordre externe venant d'une manette de contrôle ou d'un ordinateur de bord. Le DJI A3 est instrumenté d'IMU [ii] (*Inertial Measurement Unit*), d'un magnétomètre et d'un GPS (*Global Positioning System*), ce dernier n'est utilisable qu'en extérieur. Dans notre cas d'usage (en intérieur), l'IMU et le magnétomètre ne sont pas suffisants pour assurer le maintien de position en vol (en raison de données trop bruitées restituées par ces deux capteurs). Nous avons donc équipé le drone d'un ordinateur de bord (Intel NUC) raccordé par USB à une caméra de

i. `www.dji.com/fr/a3`

ii. Capteur composé d'un gyroscope et d'un accéléromètre.

profondeur (Intel D455 [iii]) et une caméra de VI-SLAM (Intel T265 [iv]). La caméra de VI-SLAM assure une localisation plus fiable que l'IMU et le magnétomètre. Les deux caméras sont attachées rigidement au drone. L'ordinateur de bord embarque des algorithmes et processus nécessaires à la réalisation de notre expérience d'exploration et de cartographie autonome. Nous listons les principaux algorithmes et processus ci-dessous :

— **Le modèle prédictif de contrôle (MPC)**: nous utilisons les travaux de Kamel et al. [108] (NMPC [v]). Le MPC convertit un ordre de déplacement (i.e. vecteur de positions, vecteur de vitesses et vecteur d'accélérations) en angles de tangage (pitch) et de roulis (roll), en vitesse angulaire de lacet et poussée. Si aucune directive n'est envoyée, le MPC transmet des directives permettant de maintenir une position de vol stationnaire.

— **La cartographie d'occupation :** nous utilisons celle proposée au Chapitre 2.

— **Méthode d'exploration employée :** nous utilisons SplatPlanner [3] présenté au Chapitre 2.

— **Divers processus d'interfaçage et de traitement de données :** typiquement, le processus qui interface la commande désirée et le contrôleur de vol DJI A3.

Les interactions entre les capteurs et les processus principaux sont schématisées dans le diagramme présenté sur la Figure 4.13.

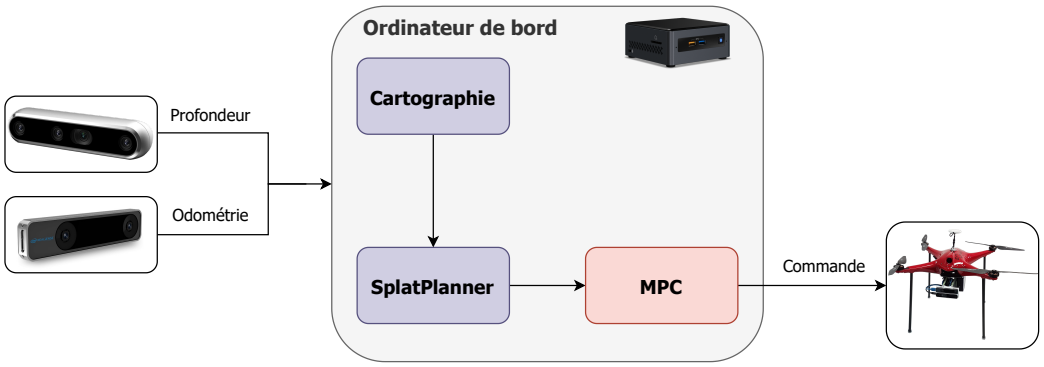


FIGURE 4.13 – Diagramme simplifié du matériel informatique, des capteurs et des processus embarqués sur le drone quadrirotor.

Phase d'étalonnage. Avant de commencer l'expérience d'exploration autonome, nous procédons à deux étapes d'étalonnage. La première assure l'estimation des paramètres extrinsèques liant les deux caméras. La seconde a pour objectif d'identifier les paramètres du système dynamique de notre drone quadrirotor.

iii. www.intelrealsense.com/depth-camera-d455
iv. www.intelrealsense.com/tracking-camera-t265
v. `https://github.com/ethz-asl/mav_control_rw`

FIGURE 4.14 – Vue d'ensemble de l'entrepôt industriel de Gambi-M prise au début de l'expérience

Les paramètres extrinsèques des deux capteurs sont estimés à l'aide de l'outil Kalibr [vi] [206] qui nécessite la prise d'une série d'images (#20) en direction d'une mire. Ces paramètres se composent d'une matrice 3×3 de rotation et d'un vecteur de translation 1×3. Ces paramètres permettent d'aligner le capteur de profondeur dans le repère du capteur de VI-SLAM et plus généralement dans le repère monde en utilisant les informations d'odométrie. Une fois cet étalonnage effectué, il est possible de construire une carte d'occupation avec notre méthode de cartographie en déplaçant les capteurs rigidement attachés au drone.

La seconde étape d'étalonnage permet d'identifier la relation entre un ordre envoyé au drone (tangage, roulis, vitesse de lacet ou poussée) et les données de localisation du capteur de VI-SLAM. Ce processus nécessite seulement un capteur d'odométrie standard (Camera T265 dans notre cas) équipé sur le drone. Les paramètres identifiés sont utilisés par le MPC [116] pour prédire, sur un horizon défini, l'ordre lissant l'effort des moteurs, respectant les contraintes physiques (e.g. vitesse maximale) choisies et l'ordre de déplacement transmis par SplatPlanner [3]. Sa et al. [207] détaillent davantage la réalisation de cet étalonnage.

Paramètres et objectif expérimentaux. L'expérience est réalisée dans l'entrepôt industriel de Gambi-M. La Figure 4.14 présente une photographie grand angle de l'entrepôt. L'objectif de l'expérience est d'obtenir une cartographie voxelique de l'entrepôt à l'aide du drone, de nos algorithmes de cartographie, de navigation et d'exploration autonome.

L'entrepôt fait approximativement $15\ m \times 10\ m \times 3.5\ m$. Divers outils et mobiliers y sont entreposés. Par souci de sécurité, et afin d'éviter tous risques de dégradations de matériel

vi. `https://github.com/ethz-asl/kalibr`

couteux, les ordres de déplacements transmis par nos algorithmes d'exploration sont limités à 2.5 m d'altitude, à 0.75 rad/s en vitesse de rotation en lacet et 0.4 m/s en vitesse linéaire. L'espace de navigation est aussi contraint (e.g. aucune trajectoire au-dessus des plans de travail). La caméra D455 possède les paramètres suivants : un angle de vue horizontal de 86° et un angle de vue vertical de 57°. La distance de perception est limitée pour cette expérience à 4 m. Toute mesure de profondeur au-delà de 4 m ne sera pas considérée dans la cartographie d'occupation utilisée par SplatPlanner [3]. Le déroulement de l'expérience, les résultats et les discussions sont fournis dans la partie suivante.

4.3.2 Résultats et Discussions

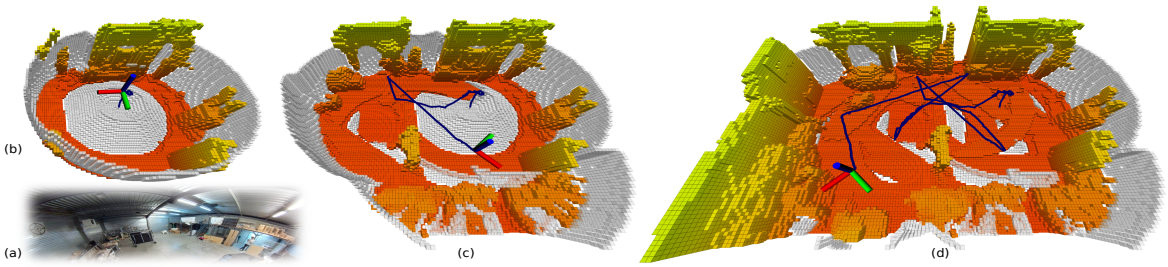


FIGURE 4.15 – Différentes étapes de l'experience : (a) $t = 54$ s, ,(b) $t = 169$ s et (c) $t = 347$ s. La carte d'occupation est graduellement enrichie grâce à notre système d'exploration autonome. Les voxels occupés sont colorés en fonction de leur hauteur, tandis que les frontières sont de couleur grise et la trajectoire empruntée par le drone est en bleue.

Déroulement de l'expérience :

1. Le drone commence par décoller pour atteindre une hauteur de 1.4 m et poursuit en tournant de 360° en lacet pour initialiser les voxels libres ou occupés qui l'entourent.
2. SplatPlanner [3] transmet des directives en temps réel au MPC pour naviguer dans l'entrepôt et ainsi enrichir la carte d'occupation.
3. Au terme de l'expérience d'une durée approximative de 350 s, nous reprenons le contrôle sur le drone pour le faire atterrir.

Résultats. La carte d'occupation, directement reconstruite pendant l'expérience, par SplatPlanner [3] prise à trois temps différents de l'expérience est présentée sur la Figure 4.15. Nous pouvons y observer, en plus des voxels occupés et des voxels frontières, le chemin emprunté par le drone. SplatPlanner [3] est exécuté sur l'ordinateur de bord, en temps réel, sans interactions extérieur pendant les 350 s que dure l'expérience.

Le graphique Figure 4.16 présente l'évolution du volume exploré en fonction du temps. Des données telles que la durée de l'expérience, la distance parcourue et le volume exploré par le drone sont rapportées sur le tableau présenté en Table 4.3. Le maillage reconstruit via

les valeurs de TSDF et l'algorithme dit du "Marching Cube" (méthode de Curless et al. [66]) est présenté Figure 4.17.

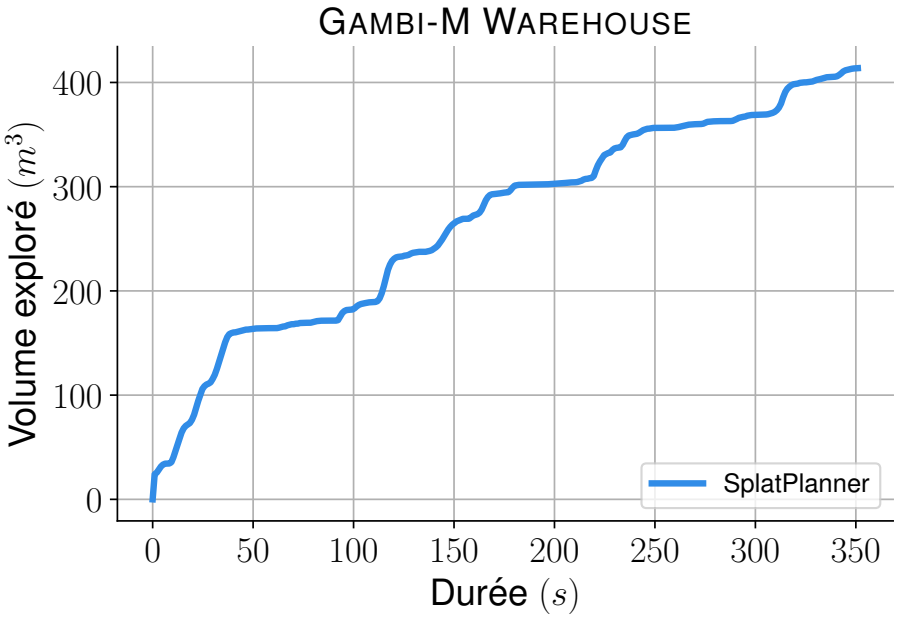


FIGURE 4.16 – Volume exploré en fonction du temps mesuré durant l'experience.

	Durée (s)	**Distance** (m)	**Volume** (m^3)
SplatPlanner [3]	350	40.33	413.71

TABLE 4.3 – Statistiques de l'expériences.

Discussion. Cette première expérience permet de valider l'utilisation en conditions de vol réel, de SplatPlanner [3] dans un environnement complexe. En comparaison des expériences d'exploration autonome réalisées en condition de vol réel référencées dans [118] et dans [166], notre expérience se déroule dans un environnement intérieur plus vaste et plus complexe.

Dans notre expérience, le drone explore 413 m^3 en 350 s c'est-à-dire une exploration volumique moyenne de 1.2 m^3/s. En guise de comparaison, cet indicateur vaut 2.9 avec la scène MAZE lors des expériences réalisées en simulation avec SplatPlanner [3]. Nous rappelons que la vitesse maximale est de 1.5 m/s pour la scène MAZE et 0.4 m/s pour cette expérience réelle. La limite de perception est de 5 m pour la scène MAZE et 4 m pour cette expérience réelle. Compte tenu de ces limites, plutôt favorables pour l'exploration du MAZE, le ratio semble légèrement en faveur de l'expérience réelle. Ceci montre aussi que notre système d'exploration fonctionne correctement en conditions de vol réelles.

Les sécurités mises sur l'espace navigable empêchent la production d'un maillage et d'une carte d'occupation complète de l'entrepôt. Ainsi, le caractère informatif de l'interprétation

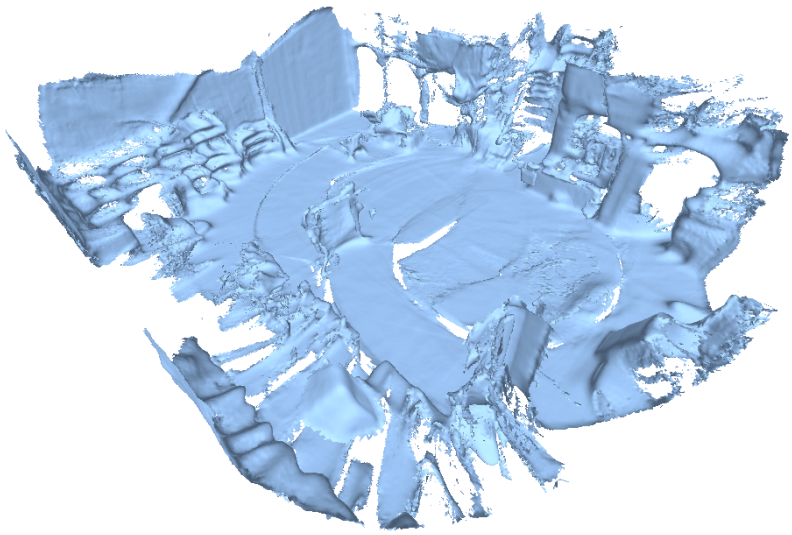


FIGURE 4.17 – Maillage résultant des données capturées par le drone reconstruit avec la méthode de curless et al. [66] (TSDF + algorithme dit du "Marching Cube").

et de la complétude du maillage produit (Figure 4.17) lors de cette expérience s'en retrouve limité. Il en est de même pour la carte d'occupation. En raison de ces sécurités et des capteurs standards utilisés, les données présentées ne sont pas aussi précises que celles qui pourraient être restituées par des relevés topographiques traditionnels faisant usage de dispositifs LiDAR/Caméra de haute précision. Cependant, cela constitue une première itération pour le projet de Gambi-M qui est d'automatiser les relevés topographiques.

Lors de cette expérience, nous avons relevé que l'envergure du drone est trop grande (actuellement de 1 m) pour un environnement intérieur. Cela rend la navigation plus difficile, risquée, voire parfois impossible à causes des zones trop étroites pour le drone et/ou des turbulences provoquées par les hélices du drone. Réduire l'envergure du drone permettrait de se rapprocher des surfaces, de faciliter ses déplacements et finalement d'accéder à bien plus d'espace libre de l'entrepôt.

Notez que, dans cette expérience, nous ne tenons pas compte de problèmes éventuels de dérive de localisation puisque le drone peut repasser souvent par des régions déjà explorées. Ainsi, la caméra T265 peut fréquemment effectuer des opérations dites de "fermetures de boucles" ce qui réduit les risques de dérive. Dans des circonstances, plus complexes pour l'algorithme de localisation (e.g. très longs couloirs), la dérive peut affecter la carte d'occupation et potentiellement la méthode d'exploration. Il existe des méthodes réduisant ce phénomène [208] mais le problème reste ouvert.

4.4 Conclusion

Dans ce chapitre, nous avons analysé les résultats des expériences réalisées durant les recherches ayant mené à cette thèse.

Dans la première section de ce chapitre, nous avons proposé de comparer les sept méthodes implantées dans FLYBO sur les deux tâches considérées dans ce dernier à savoir l'exploration volumique et la reconstruction de surfaces. Les méthodes évaluées sont les suivantes : Classic [2], Rapid [69], NBVP [116], ESM$_{Recon}$ [117] ESM$_{IPP}$ [117] , ESM$_{Recon}$ [117] ESM$_{Recon}$ [117] , AEP [118] et SplatPlanner [3]. Nos résultats ont mis en avant trois méthodes parmi les sept présentes. ESM$_{IPP}$ [117] pour maximiser la métrique de complétude d'exploration volumique, ESM$_{Recon}$ [117] pour maximiser la justesse de reconstruction et SplatPlanner [3] qui a émergé comme le meilleur compromis pour maximiser ces deux métriques. De plus, sur les métriques d'efficacité en termes de volume exploré et de rappel de reconstruction géométrique en fonction du temps, SplatPlanner a obtenu en moyenne les meilleurs résultats.

Dans la seconde section de ce chapitre, nous avons présenté une expérience d'exploration autonome utilisant SplatPlanner [3] en conditions de vol réel avec un drone quadrirotor. Cette expérience constitue pour Gambi-M une première itération vers l'automatisation des relevés topographiques par drone quadrirotor. Toutefois, les données rapportées sont encore insuffisantes pour remplacer totalement le relevé topographique traditionnel. Le projet requiert encore quelques itérations de recherche et développement pour atteindre cet objectif.

Conclusion

Conclusion

Dans ce manuscrit, nous avons étudié l'exploration et la reconstruction 3D autonome d'infrastructures intérieures et extérieures à l'aide d'un drone quadrirotor équipé d'un système de localisation ainsi que d'une caméra de profondeur. Nous avons pu évaluer les besoins algorithmiques que ce type de robot requiert pour passer d'un contrôle humain haut niveau à une complète autonomie robotique. Nous avons pris connaissance des travaux existants proposant des méthodes d'exploration autonome efficace utilisant des robots mobiles. À la lecture de ces travaux, nous avons pu identifier deux enjeux. Le premier découle de l'émergence récente de l'usage de drones quadrirotors qui induit que les méthodes d'exploration utilisant ce type de robot sont encore perfectibles. Le second découle du constat d'un manque de rigueur dans l'évaluation des méthodes d'exploration autonome proposées par la littérature scientifique. Pour chacun de ces enjeux, nous avons proposé une contribution.

Dans notre première contribution (c.f. Chapitre 1) nous nous sommes intéressés au développement d'une méthode originale d'exploration autonome et efficace (SplatPlanner [3]). Nous avons proposé une stratégie originale d'exploration hybride couplée à une méthode de cartographie d'occupation adaptée. Notre méthode réalise simultanément une estimation de trajectoires d'exploration et une reconstruction de l'environnement sous forme d'une grille 3D de voxels. Pour réaliser cette double tâche de manière efficace, nous avons proposé une approche originale s'appuyant sur un algorithme de filtrage bilatéral. Cette approche attribue un score d'intérêt aux voxels situés aux frontières des régions visitées afin de guider l'échantillonnage de nouveaux objectifs. Ce score tient compte de la densité de voxels inconnus à proximité des frontières, ce qui permet d'échantillonner des points de vue et des trajectoires d'exploration de manière plus efficace que les méthodes de la littérature auxquelles nous l'avons comparée au cours de nos expériences. Notre méthode a aussi fait l'objet d'une expérience d'exploration autonome en condition de vol réel (c.f. Chapitre 4).

Dans notre seconde contribution (c.f. Chapite 3) nous avons proposé un environnement de simulation conçu pour tester, évaluer et comparer les méthodes d'exploration autonome (FLYBO [62]). Cet environnement permet de simuler un drone quadrirotor instrumenté d'une caméra de profondeur et d'un capteur d'odométrie étalonnés, rigidement attachés au drone. Nous nous sommes intéressés à l'évaluation de deux tâches : (i) la tâche d'exploration volumique et (ii) la tâche de reconstruction de surfaces en ligne. Pour ce faire, dans notre contribution, nous avons :

— rassemblé des modèles de scènes 3D réalistes de complexité et de tailles variées,
— étudié et ajouté des métriques spécifiques aux deux tâches considérées ici,
— fournit des outils communs à l'évaluation de méthodes d'exploration autonome,

— rendu compatibles, en plus de SplatPlanner [3], six autres méthodes originales d'exploration autonome efficace.

De cette contribution, nous avons pu proposer une étude comparative exhaustive (Chapitre 4) des méthodes implantées dans FLYBO [62].

Limites et perspectives

Dans cette section, nous présentons des perspectives d'améliorations de la méthode de cartographie d'occupation, de la méthode d'exploration autonome que nous avons proposée ainsi que FLYBO.

Cartographie d'occupation et exploration autonome

La méthode de cartographie d'occupation que nous avons utilisée (c.f. Chapitre 2) fonctionne avec une grille régulière. Ce type de structure est suffisant dans les cas d'usage que nous avons présenté dans cette thèse. Cependant, comme l'efficacité de cette structure est corrélée avec la taille de l'environnement et son nombre de voxels, ses performances peuvent être moindres dans des environnements beaucoup plus volumineux que ceux que nous avons proposés dans nos expériences. Pour résoudre ce problème, une solution serait de remplacer la grille régulière par un Octree sous condition de lui adjoindre une méthode d'extraction de frontières efficace (ce n'est pas le cas en général). Une autre solution, moins triviale mais plus élégante, consisterait à découper l'espace d'exploration en une multitude de cartes d'occupation, de tailles variables, connectées et pouvant se chevaucher. Ce mécanisme de sous cartes a été proposée par Duhautbout et al. [209] pour résoudre le problème de cartographie multi-robot. Nous retrouvons aussi cette solution dans [208] pour répondre aux problèmes de dérive de localisation dans des circonstances de navigation long terme dans des environnements complexes. Dans ces travaux, les auteurs utilisent un graphe connectant les cartes d'occupation entrent-elles en fonction de leur proximité et de leur chevauchement. Ceci leur permet d'estimer un alignement global à partir du chevauchement des champs de distance euclidien (EDF). Les EDFs sont calculées en fonction des obstacles étiquetés dans les cartes d'occupations. Ces chevauchements permettent d'établir des contraintes de distance entre les différentes cartes (comparable aux méthodes de graphe utilisées en SLAM c.f. Chapitre 1). Des résultats rapportés par les auteurs [208], cette optimisation améliore les performances de localisation et réduit la dérive. Dans un contexte d'exploration long terme, diviser l'environnement en une multitude de cartes d'occupation aurait les bénéfices suivants :

1. accélération des calculs d'objectifs d'exploration, car seulement fait localement par carte. Les objectifs précédemment extraits sur d'autres cartes d'occupation (éloigné de la position du robot) peuvent être retenus et utilisés plus tard,
2. accélération du calcul de trajectoire, car aussi fait localement,
3. optimisation de l'espace mémoire en compressant et sauvegardant les cartes d'occupation explorées n'ayant plus d'utilités pour explorer l'environnement,

4. amélioration des performances de localisation et réduction de la dérive [208].

Concernant la méthode d'exploration proposée (SplatPlanner), nous avons vu que le treillis permutoèdrique est une structure utilisée dans le domaine de l'apprentissage [179, 180]. Le treillis permutoèdrique peut gérer des vecteurs caractéristiques beaucoup plus complexes en termes de dimensionnalité, qui pourrait s'ajouter au vecteur de coordonnées 3D que nous considérons. Nous laissons cette voie à des travaux futurs, par exemple, pour intégrer à notre formulation d'exploration un raisonnement sémantique ou de la détection d'objet/forme 3D. Grâce à la détection d'objets 3D, il serait possible d'exploiter leur pose et leur occupation volumique prédites pour automatiquement remplir les voxels des régions concernées. Une autre piste pourrait être l'utilisation d'une modalité supplémentaire telle que l'exploitation de l'information couleur dans un processus d'apprentissage et de segmentation sémantique à des fins d'exploration autonome.

Une composante importante de notre méthode est le calcul de trajectoires. Actuellement, ce calcul utilise l'algorithme RRT^*. Le drone navigue en suivant des trajectoires composées de segments linéaires. Il serait intéressant de remplacer l'algorithme RRT^* par un graphe de trajectoire, maintenu tout au long de l'exploration, qui couvrirait approximativement toutes les régions visitées (semblable à ESM_{IPP}). Ceci permettrait d'étudier et de comparer le gain, en particulier sur des environnements de grande envergure, là où en l'état l'algorithme RRT^* pourrait avoir des difficultés quand il reste peu d'espace exploré. Une autre piste d'amélioration serait la construction de trajectoires dynamique [168] (sans interruption aux extrémités des segments). Néanmoins, il faudrait alors étudier le gain en vitesse d'exploration, l'impact de cette stratégie sur les métriques de justesse et d'erreur de reconstruction géométrique ainsi que les potentielles augmentations de risques de collisions ou de casse de matériel en vol réel puisque le drone volerait plus vite.

L'exploration en condition de vol réel met en vue quelques perspectives de développement pour obtenir des reconstructions de surfaces plus précises et plus complètes en vue de remplacer le relevé topographique traditionnel. Pour commencer afin d'obtenir des comparaisons plus complètes avec SplatPlanner [3], en particulier sur la tâche de reconstruction, nous pourrions envisager une méthode qui tient compte des erreurs de reconstruction (e.g ESM_{Recon} [117]), ainsi qu'une campagne de relevés topographiques avec un LiDAR très précis de l'environnement à explorer. Le relevé topographique constituerait la vérité terrain.

Une autre piste que nous aimerions développer concerne l'usage de la caméra RGB. Une caméra RGB de haute définition pourrait, avec des outils tels que COLMAP [210] ou Pix4D [vii] donner des résultats de reconstruction plus fiables qu'en utilisant la TSDF. Cependant, ces outils de reconstruction utilisent des processus trop complexes en calcul pour aboutir à des processus en temps réel sur l'ordinateur de bords. Ces outils pourraient être utilisés qu'en post-traitement. Notre proposition concerne davantage l'usage de "bonnes pratiques" pour améliorer les résultats que ces outils produisent. Par exemple, Roberts et al. [211] recommande d'enregistrer des images ayant, par rapport à un objet donné, un angle d'incidence entre chacune d'elle de 5° à 15°. De même, une faible distance avec les surfaces observées peut permettre de reconstruire des structures fines. Une piste

vii. `www.pix4d.com`

très intéressante serait de réunir des "bonnes pratiques" en les généralisant dans une fonction de gain qui guiderait l'exploration et la reconstruction autonome basée image couleur. Ici, la piste de l'apprentissage pour résoudre cette tâche n'est pas à écarter aux vues des performances que ces méthodes obtiennent sur une reconstruction 3D basée image [192].

Une dernière piste d'amélioration, qui permettrait de se rapprocher de la qualité des relevés traditionnels, fait usage d'une stratégie d'exploration autonome en deux étapes. La première étape restituerait rapidement une cartographie de l'environnement, par exemple, avec l'utilisation de SplatPlanner [3]. La seconde calculerait les déplacements optimaux en vue de restituer une reconstruction complète et très précise de l'environnement dans un temps imparti. Notez que, dans un premier temps, la première étape pourrait être des données, volontairement incomplètes, acquises par des relevés topographiques traditionnels. Dans ce cas, la seconde étape viserait à améliorer et compléter les relevés topographiques réalisés.

FLYBO

L'environnement de simulation que nous avons présenté est naturellement perfectible. Nous présentons ici des pistes potentielles d'amélioration de FLYBO.

Dans un premier temps, il serait important d'envisager l'ajout de scènes 3D supplémentaires pour augmenter la variété des environnements simulés (e.g. forêt). De même, il serait intéressant d'implanter d'autres méthodes d'exploration autonome qui ne sont pas encore considérées dans FLYBO (e.g. [168]). Cela permettrait des comparaisons encore plus pertinentes.

Une piste de travail passionnante, que nous aimerions développer, réside dans l'ajout potentiel de modalités de données supplémentaires, telles que des scans LiDAR ou le rendu d'une caméra RGB simulée réaliste. Le LiDAR est un capteur souvent préféré aux caméras de profondeur dans des campagnes d'exploration et de reconstruction autonome extérieures ou à très grandes échelles [212, 213]. La caméra RGB est parfois choisie pour réaliser l'acquisition automatique et la reconstruction 3D basée image de scènes urbaines [185]. Ce type de caméra peut aussi être envisagée pour construire une image de profondeur via des méthodes d'apprentissage [214-216] pour remplacer une caméra de profondeur standard.

Les récents succès de l'apprentissage automatique dans de multiples domaines ont entrainé l'utilisation d'approche d'apprentissage pour les problèmes d'exploration autonome [119, 157] ou de planification de trajectoire [96]. Il est important de préciser que de nombreuses méthodes d'apprentissage s'intéressent aux données issues de LiDAR ou de caméra RGB pour des tâches telles que la détection d'objet 2D et/ou 3D [217-220] ou la segmentation d'image et/ou de nuage de points [179, 221]. Ces méthodes d'apprentissage pourraient être entrainées dans FLYBO et utilisées par des méthodes d'exploration autonome déjà implantées dans FLYBO. La combinaison de ce type de méthodes d'apprentissage (identification/détection) et de stratégies d'exploration efficaces peut mener la résolution de tâches telles que le secours d'individus perdus ou en situation de détresse dans un environnement hostile (e.g. forêt, haute montagne, zone contaminée, zone de combat...). Ce type de tâche pourrait être envisagée dans FLYBO.

Les méthodes d'exploration peuvent utiliser des applications dérivées de l'apprentissage pour améliorer leur rapidité d'exploration. Par exemple, une amélioration pertinente, à notre connaissance non réalisée, serait l'usage d'algorithmes de complétion de scène en ligne [222, 223] avec éventuellement l'ajout de détection et d'identification d'objets 3D. Ce type d'algorithme nécessite, pour son entrainement et son usage des scènes 3D, une caméra de profondeur, un capteur d'odométrie et une cartographie voxelique, ce dont l'environnement FLYBO dispose actuellement. L'amélioration pourrait consister à insérer les objets prédits dans la représentation voxelique de FLYBO [224, 225], ce qui éviterait au robot explorateur de devoir entièrement parcourir certaines régions de la scène.

Une branche de l'apprentissage très populaire en robotique est l'apprentissage par renforcement. Les méthodes qui s'appuient sur l'apprentissage par renforcement nécessitent généralement une quantité modérée de données (scène 3D) d'apprentissage [226] mais requièrent un moyen efficace de multiplier les expériences d'apprentissage. Étant donné la nature structurée et unifiée de FLYBO, les données et l'environnement de simulation proposés peuvent être utilisés pour soutenir le développement d'approches d'exploration autonome avec de l'apprentissage par renforcement [227, 228]. Toutefois, FLYBO requiererait le développement d'un moyen efficace pour paralléliser et multiplier les expériences d'apprentissage.

L'exploration autonome s'appuie fortement sur la génération de trajectoires continues, efficaces et optimisées en fonction de contraintes imposées par le drone quadrirotor [61, 91, 92]. Notre simulateur et le jeu de données proposé pourraient directement bénéficier aux méthodes de (re)-planification de trajectoires [91, 168, 229] pour améliorer et valider leurs performances dans des scènes simulées réalistes. Parallèlement, les méthodes d'exploration incluses dans FLYBO pourraient tirer avantage des méthodes de calcul de trajectoire en tant qu'alternative aux trajectoires linéaires par segment actuellement supportées.

FLYBO pourrait aussi être envisagé pour simuler et évaluer des méthodes d'exploration autonome multi-agent [175]. Toutefois, quelques développements techniques seraient à envisager (e.g. communication entre les agents) ainsi que l'ajout de métriques.

Les expériences réalisées avec FLYBO nous ont permis de constater des difficultés pour la reconstruction de structures fines. Ces difficultés ont induit un léger biais à nos résultats de reconstruction. Elles sont liées à la méthode utilisée (basée sur la TSDF). Il serait très bénéfique à FLYBO de travailler sur la méthode et les métriques concernées. Conjointement aux développements de la simulation d'un capteur RGB il pourrait être envisagé d'utiliser une méthode de reconstruction basée image. Une autre piste que nous pensons prometteuse pour la reconstruction 3D serait l'usage de ce nouveau paradigme de rendu 3D fondé sur les champs de radiance neuronale [230, 231].

Annexes

Listes des contributions

Les recherches scientifiques présentées dans ce manuscrit ont fait l'objet de deux publications en conférence international :

— [3] **Brunel, A.**, Bourki, A., Demonceaux, C., et Strauss, O. (2021, May). Splatplanner: Efficient autonomous exploration via permutohedral frontier filtering. In 2021 IEEE International Conference on Robotics and Automation (ICRA) (pp. 608-615).

— [62] **Brunel, A.**, Bourki, A., Strauss, O., et Demonceaux, C. (2021, December). FLYBO: A Unified Benchmark Environment for Autonomous Flying Robots. In 2021 International Conference on 3D Vision (3DV) (pp. 1420-1431).

Nous proposons également des implémentations publiques et des données mise a disposition de la recherche :

— SplatPlanner ainsi qu'une méthode de cartographie d'occupation disponible sur GitHub à l'adresse suivante : `https://github.com/anthonybrunel/splatplanner`.

— Implémentations de classic et Rapid [69] disponible sur GitHub à l'adresse suivante : `https://github.com/anthonybrunel/splatplanner`.

— Un environnement (FLYBO) qui rassemble les composants requis et communs à de nombreuses routines de simulation utilisées pour évaluer les méthodes d'exploration autonome. FLYBO est disponible sur GitHub à l'adresse suivante : `https://github.com/anthonybrunel/FLYBO`.

— Un ensemble de scène 3D réalistes téléchargeable à l'adresse suivante : `https://drive.google.com/file/d/1-RZQbAvwqtydrg2PHvfipUo-F8K7V27U/view`.

En parallèle de ce projet de thèse, et avec le soutien de Gambi-M, nous avons réalisé et encadré trois stages de fin d'étude. Le premier stage, réalisé avec Valentin PENAUD-POLGE, a permis d'explorer des méthodes de cartographie d'occupation, de localisation et de cartographie simultanées basées LiDAR ainsi que des techniques de générations de trajectoires. Le second stage, réalisé avec Sébastien PETIT, a permis d'appréhender les techniques d'asservissement de drone quadrirotors, en vue de leurs utilisations en condition de vol réel. Le dernier stage réalisé avec Weili WANG s'intéressait à l'estimation par apprentissage profond d'images de profondeur à parti d'images fisheye en niveau de gris ou en couleur. Ceci en perspective future, de compléter, voire de remplacer les caméras de profondeur traditionnelles dont les données peuvent être parfois insuffisantes pour la navigation dans certains environnements.

TABLE .4 – Liste exhaustive des paramètres expérimentaux utilisés en simulation.

	PARAMÈTRES	MAZE	OFFICES A	OFFICES B	OFFICES C	FACILITY A	FACILITY B	FACILITY C	WAREHOUSE	PLATFORM A	PLATFORM B	POWERPLANT
All	$res.\ (m)$	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.2
	$R\ (m)$	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
	$v_{max}\ (m/s)$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
	$a_{max}\ (m/s^2)$	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5
	$\omega_{max}\ (rad/s)$	1.57	1.57	1.57	1.57	1.57	1.57	1.57	1.57	1.57	1.57	1.57
	$d_{max}\ (m)$	5.0	5.0	5.0	5.0	5.0	5.0	5.0	7	7	7	7
	$fov\ [h, v]$	[90, 73.73]	[90, 73.73]	[90, 73.73]	[90, 73.73]	[90, 73.73]	[90, 73.73]	[90, 73.73]	[115, 99.3]	[115, 99.3]	[115, 99.3]	[115, 99.3]
	$image_{res}\ [h, v]$	[600,450]	[600,450]	[600,450]	[600,450]	[600,450]	[600,450]	[600,450]	[600,450]	[600,450]	[600,450]	[600,450]
	$tsdf_{res}\ (m)$	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
	$tsdf_{trunc}\ (m)$	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09
NBVP [116]	N	15	15	15	15	15	15	15	15	15	15	15
	g_{zero}	1	1	1	1	1	1	1	1	1	1	1
	λ	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	$l_{max}\ (m)$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	3.0	3.0	3.0	3.0
AEP [118]	N	15	15	15	15	15	15	15	15	15	15	15
	g_{zero}	1	1	1	1	1	1	1	1	1	1	1
	λ	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	σ^2_{tresh}	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	$l_{max}\ (m)$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	3.	3.0	3.0	3.0
ESM_{IPP} [117] , ESM_{Recon} [117]	n_{local}	15	15	15	15	15	15	15	15	15	15	15
	$r_{local}\ (m)$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	3.0	3.0	3.0	3.0
	$r_{update}\ (m)$	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	6.0	6.0	6.0
	$crop_{margin}\ (m)$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	2.0	2.0	2.0
	$crop_{min}\ (m)$	1.3	1.3	1.3	1.3	1.3	1.3	1.3	2.5	2.5	2.5	2.5
	$rewire_{l_{max}}\ (m)$	1.6	1.6	1.6	1.6	1.6	1.6	1.6	3.1	3.1	3.1	3.1
	$r_{density}\ (m)$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	2.0	2.0	2.0
	$l_{max}\ (m)$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	3.0	3.0	3.0	3.0
	$l_{min}\ (m)$	1.3	1.3	1.3	1.3	1.3	1.3	1.3	2.5	2.5	2.5	2.5
SplatPlanner [3]	$r_{thresh}\ (m)$	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5
	$\sigma_{xyz}\ (m)$	1	1	1	1	1	1	1	1	1	1	1

Résultats et comparaisons additionnels

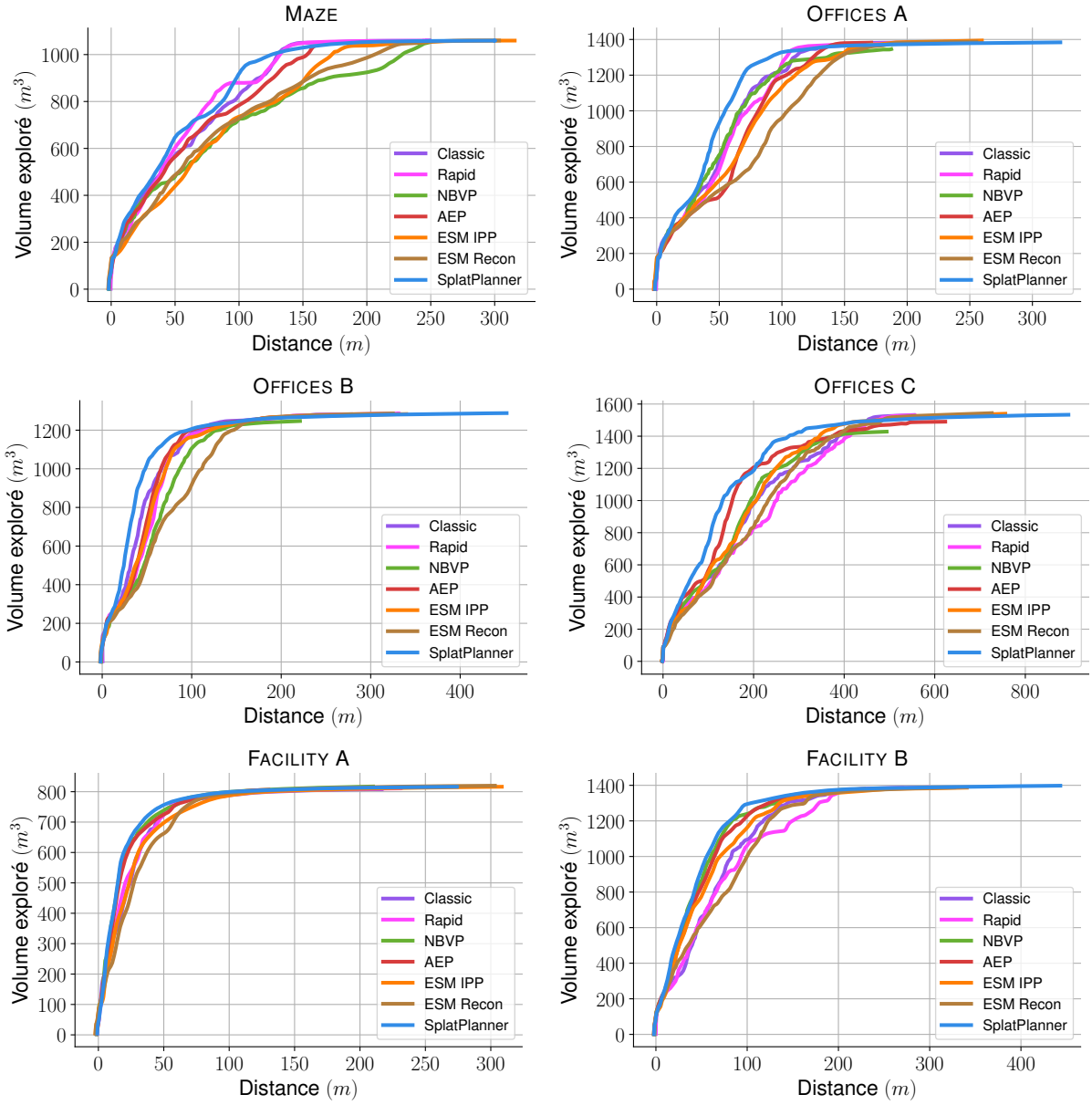


Figure .18 – Les graphiques de cette Figure représentent la quantité de volume exploré (en m^3), en fonction de la distance parcourue (en m) par chaque méthode, moyenné sur 5 exécutions par scène. Les noms des scènes sont spécifiés au-dessus des graphiques. L'intervalle coloré correspond au volume exploré minimum et maximum obtenus sur les 5 exécutions.

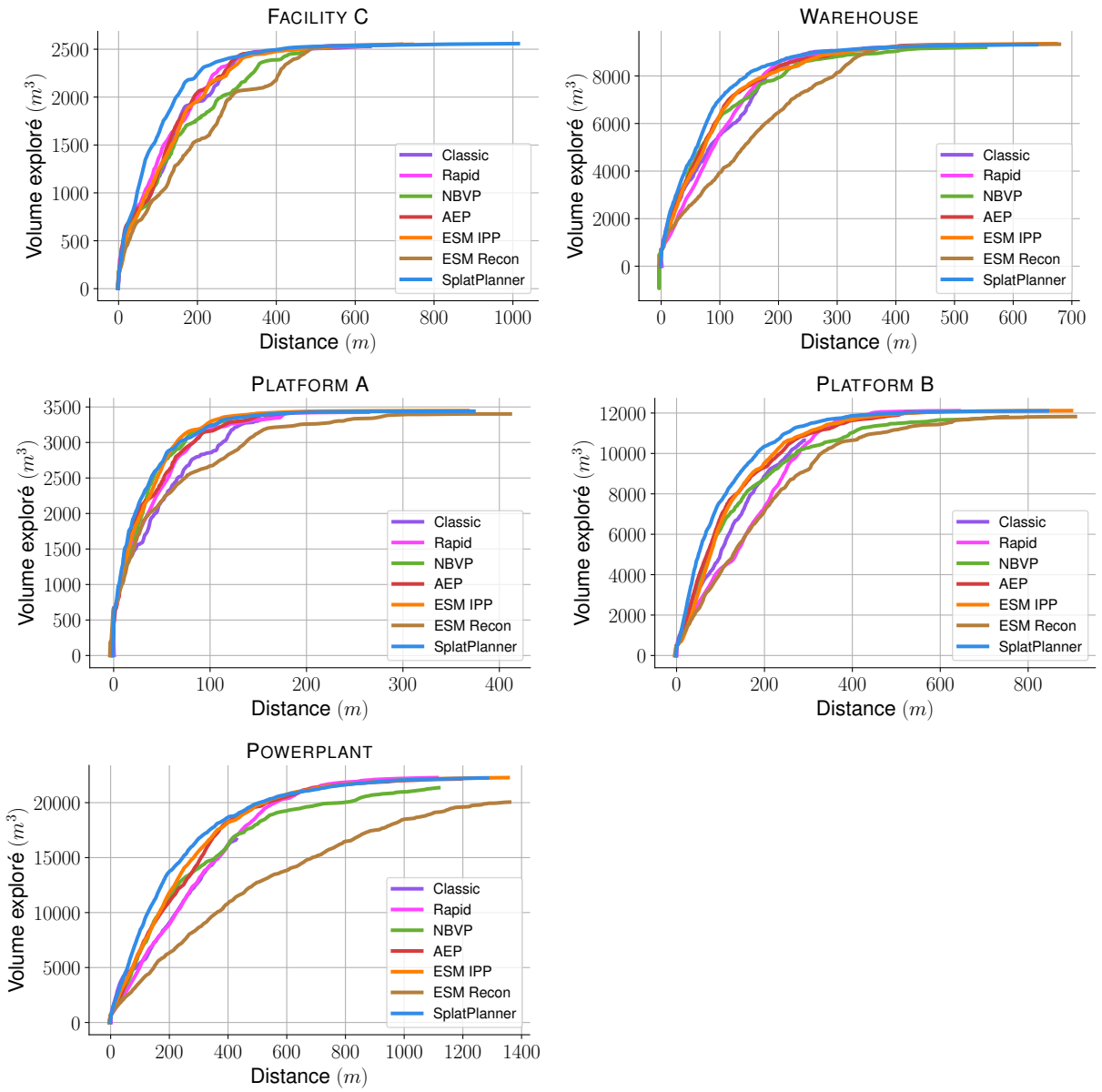


FIGURE .19 – Les graphiques de cette Figure représentent la quantité de volume exploré (en m^3), en fonction de la distance parcourue (en m) par chaque méthode, moyenné sur 5 exécutions par scène. Les noms des scènes sont spécifiés au-dessus des graphiques. L'intervalle coloré correspond au volume exploré minimum et maximum obtenus sur les 5 exécutions.

	#Itérations	**Temps moyen. par iter. (ms)**	**Distance parcourue (m)**
Classic [2]	147 ± 7	35 ± 19	292 ± 21
Rapid [69]	52 ± 6	85 ± 110	442 ± 15
NBVP [116]	196 ± 2	858 ± 83	554 ± 8
AEP [118]	173 ± 5	37 ± 9	500 ± 15
ESM_{IPP} [117]	251 ± 1	N/A	673 ± 1
ESM_{Recon} [117]	246.9 ± 1	N/A	679 ± 2
SplatPlanner [3]	61 ± 2	236 ± 69	641 ± 31

TABLE .5 – Statistiques comparatives extraites des expériences réalisées sur la scène WAREHOUSE au temps $t = 900s$. Nous rapportons pour chaque méthode le nombre d'itérations effectuées et le temps de calcul moyen par itération de leur algorithme ainsi que la distance totale parcourue par le drone.

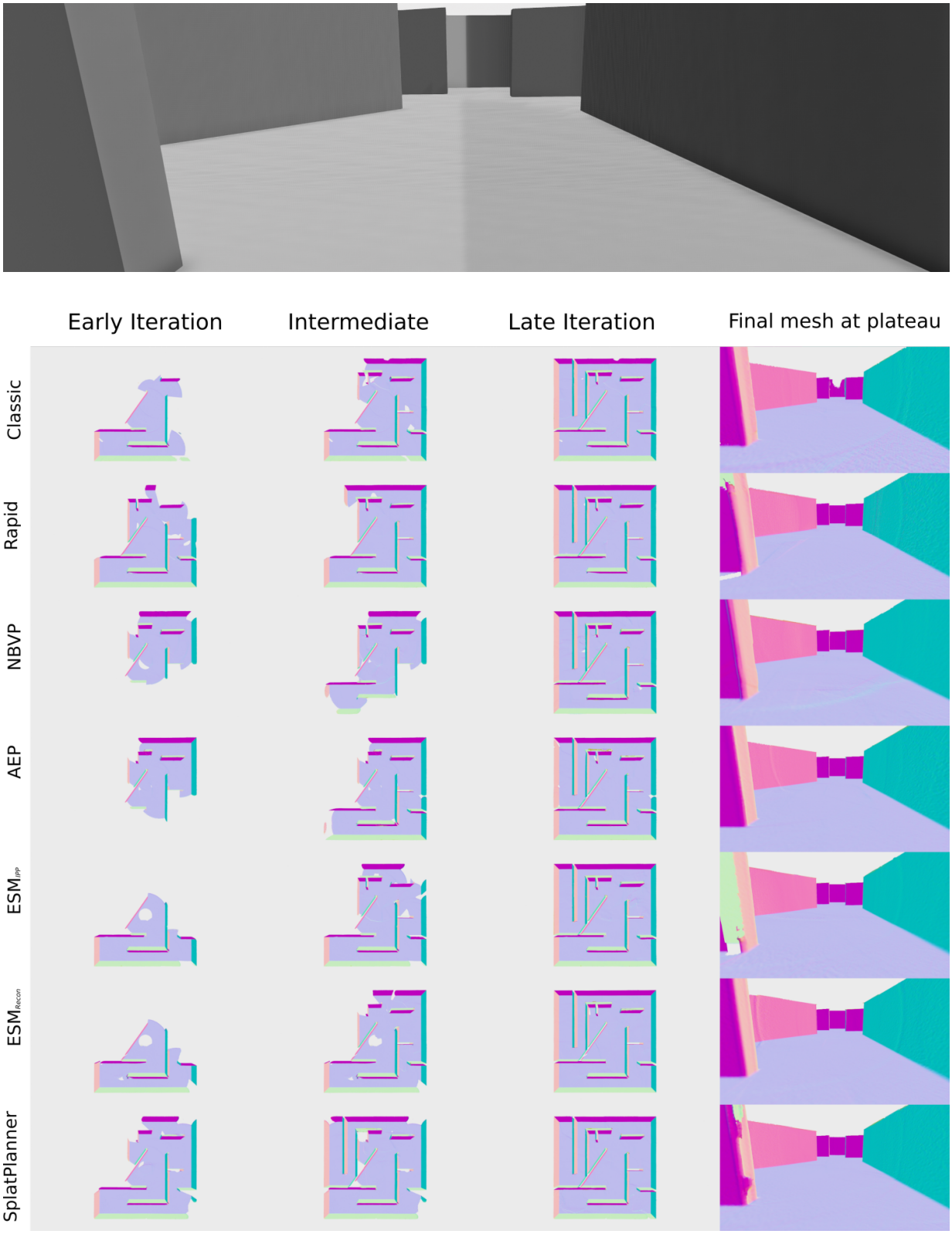


FIGURE .20 – Reconstructions progressives de l'environnement MAZE (illustré en modèle CAO ligne du haut) en vue immersive.

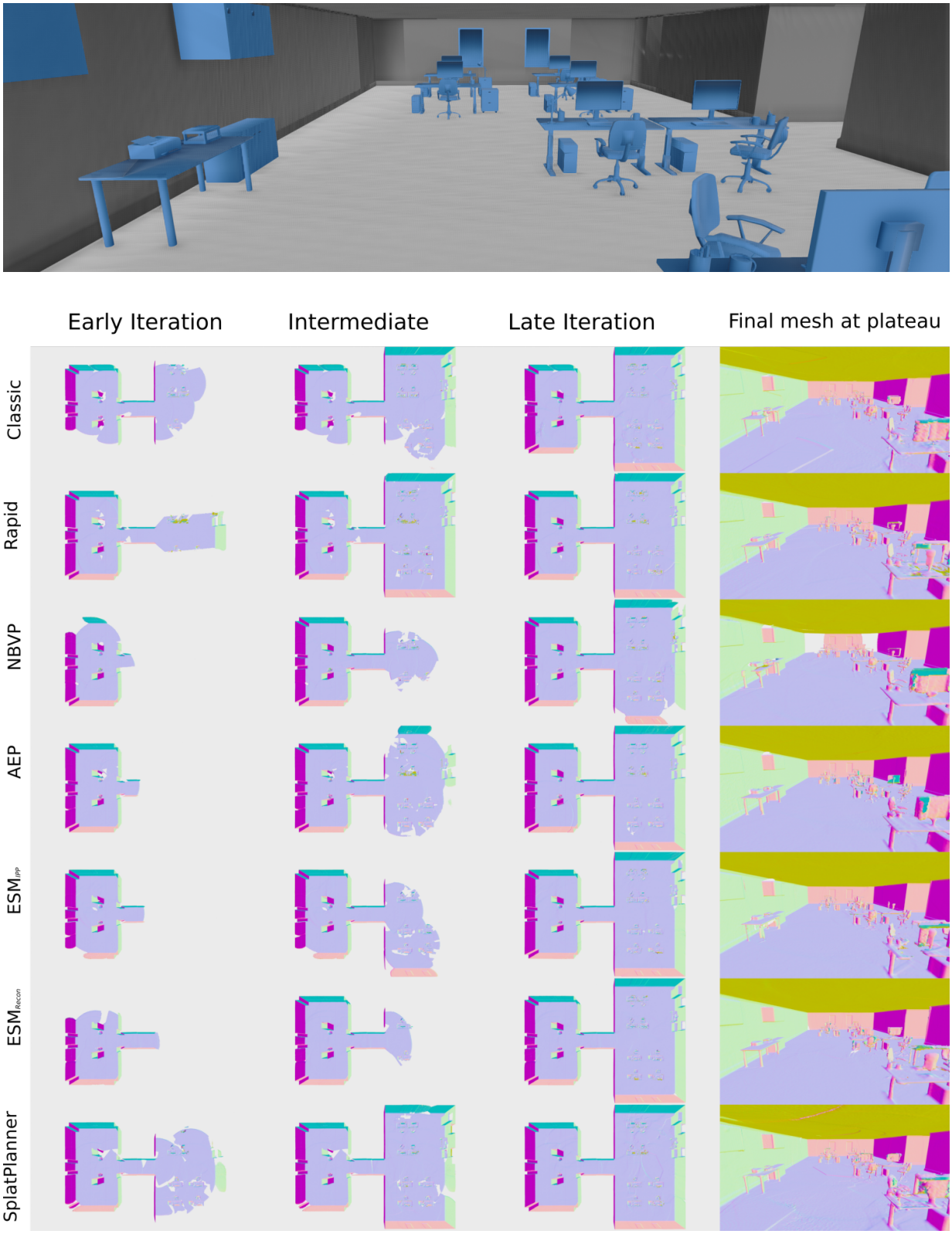


FIGURE .21 – Reconstructions progressives de l'environnement OFFICES A (illustré en modèle CAO ligne du haut) en vue immersive.

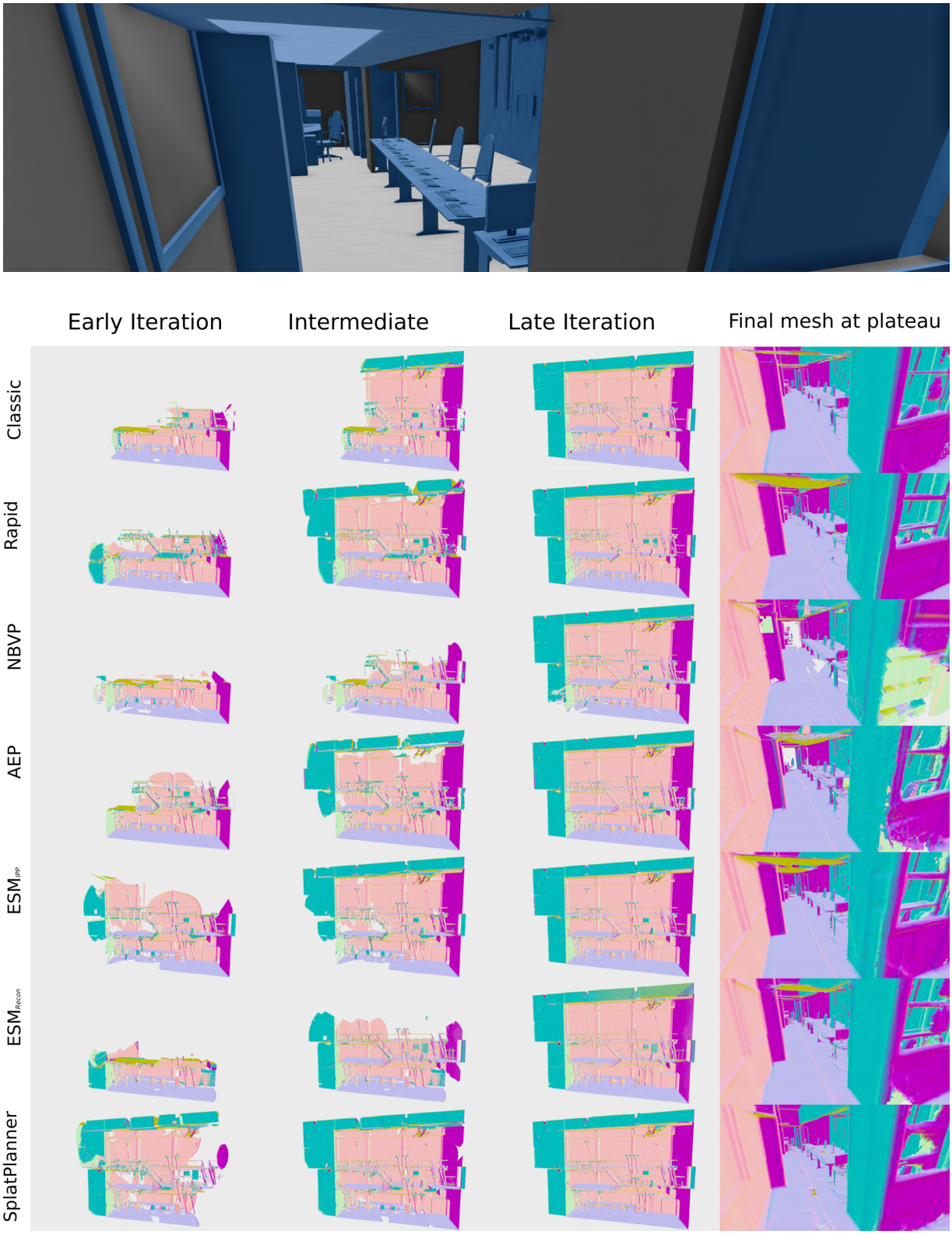


FIGURE .22 – Reconstructions progressives de l'environnement OFFICES B (illustré en modèle CAO ligne du haut) en vue immersive.

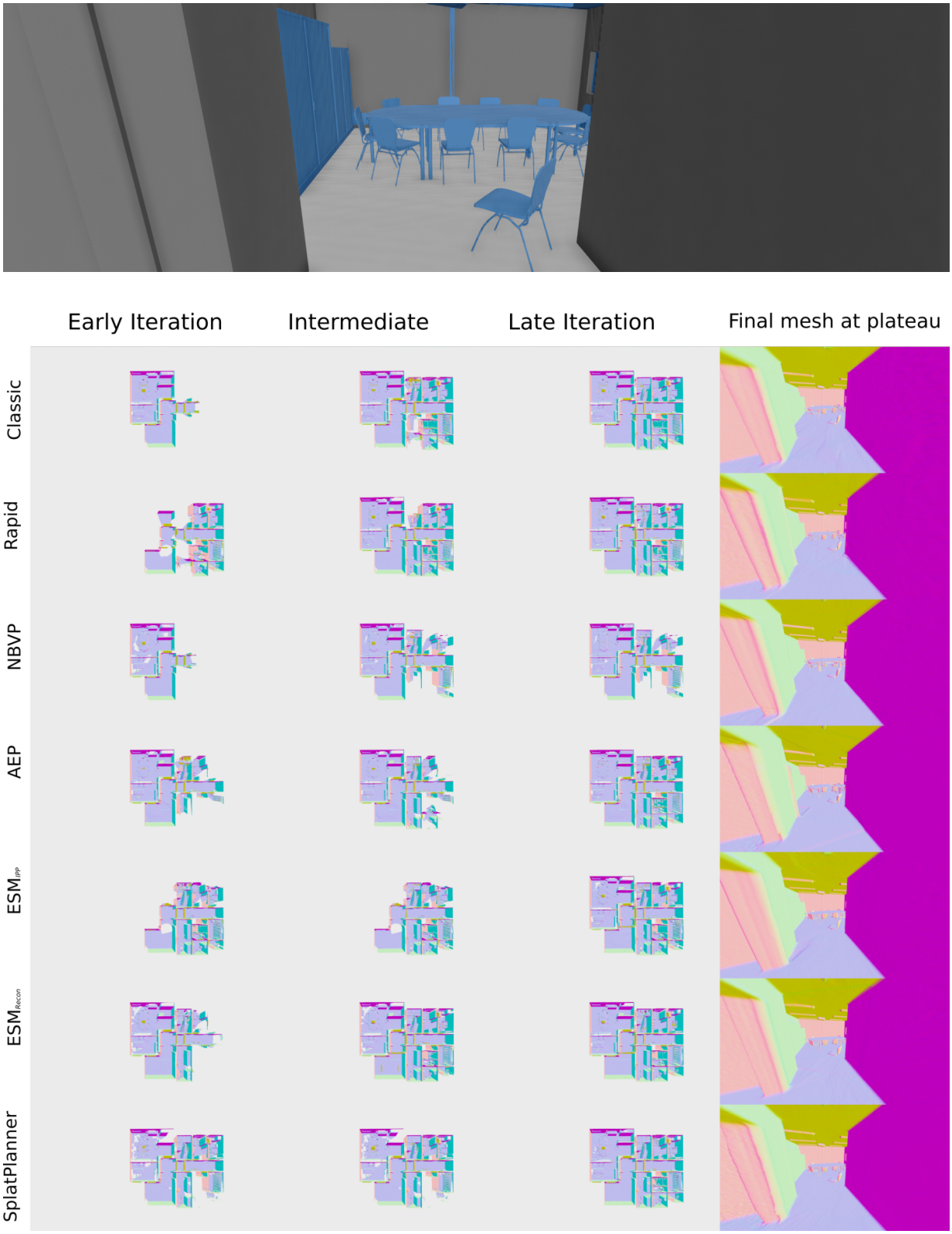


FIGURE .23 – Reconstructions progressives de l'environnement OFFICES C, rez-de-chaussée, (illustré en modèle CAO ligne du haut) en vue immersive.

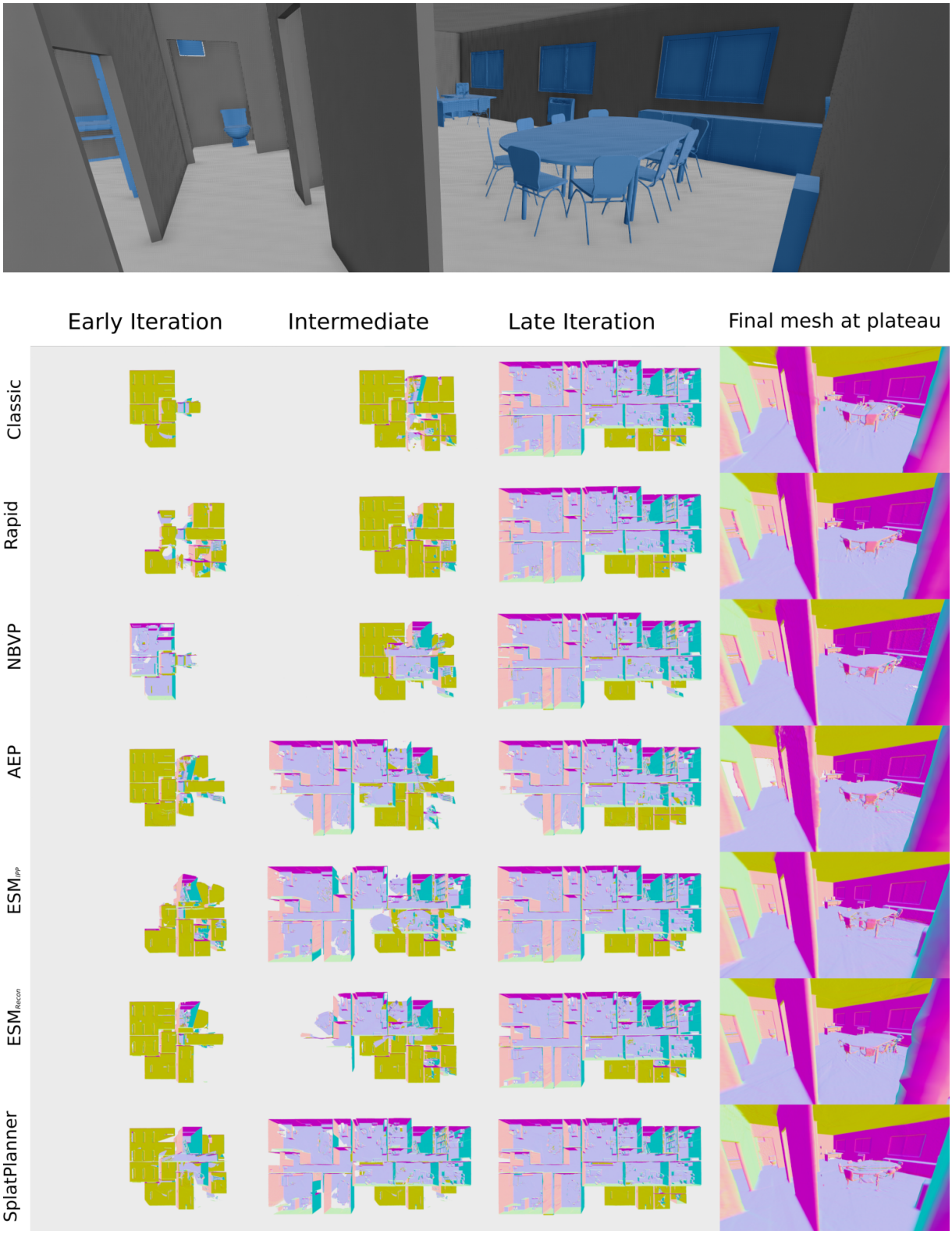


FIGURE .24 – Reconstructions progressives de l'environnement OFFICES C, première étage, (illustré en modèle CAO ligne du haut) en vue immersive.

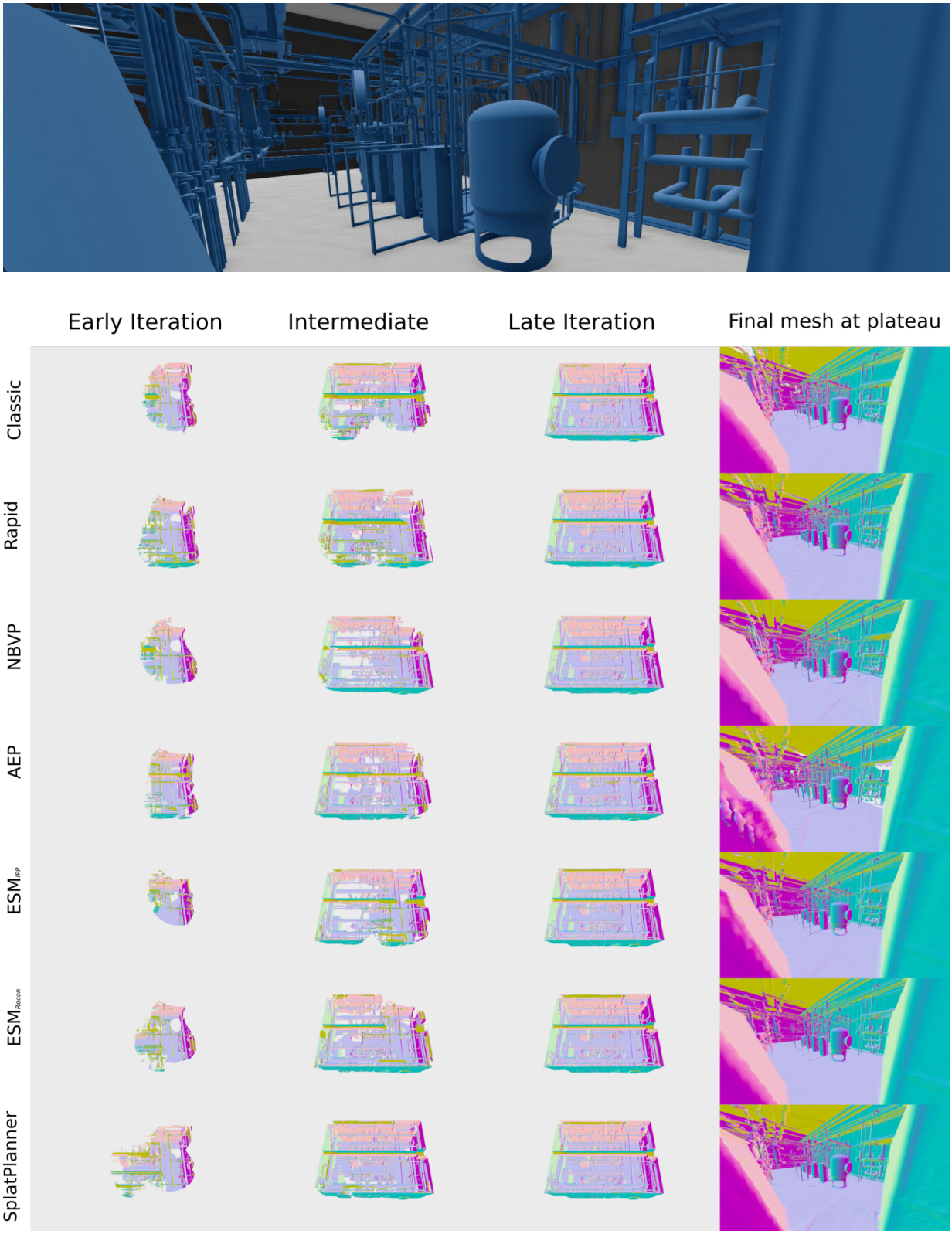


FIGURE .25 – Reconstructions progressives de l'environnement FACILITY A (illustré en modèle CAO ligne du haut) en vue immersive.

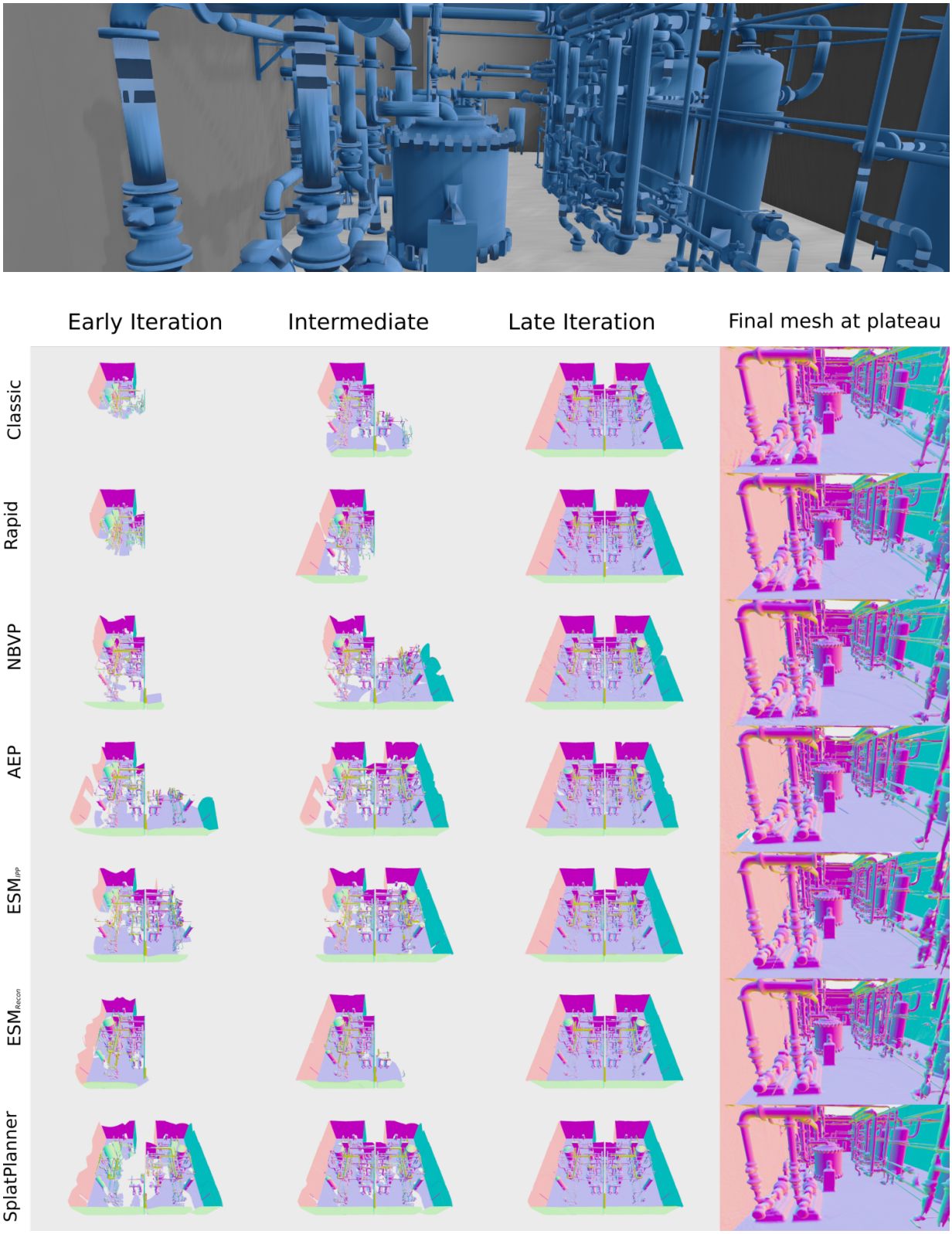


FIGURE .26 – Reconstructions progressives de l'environnement FACILITY B (illustré en modèle CAO ligne du haut) en vue immersive.

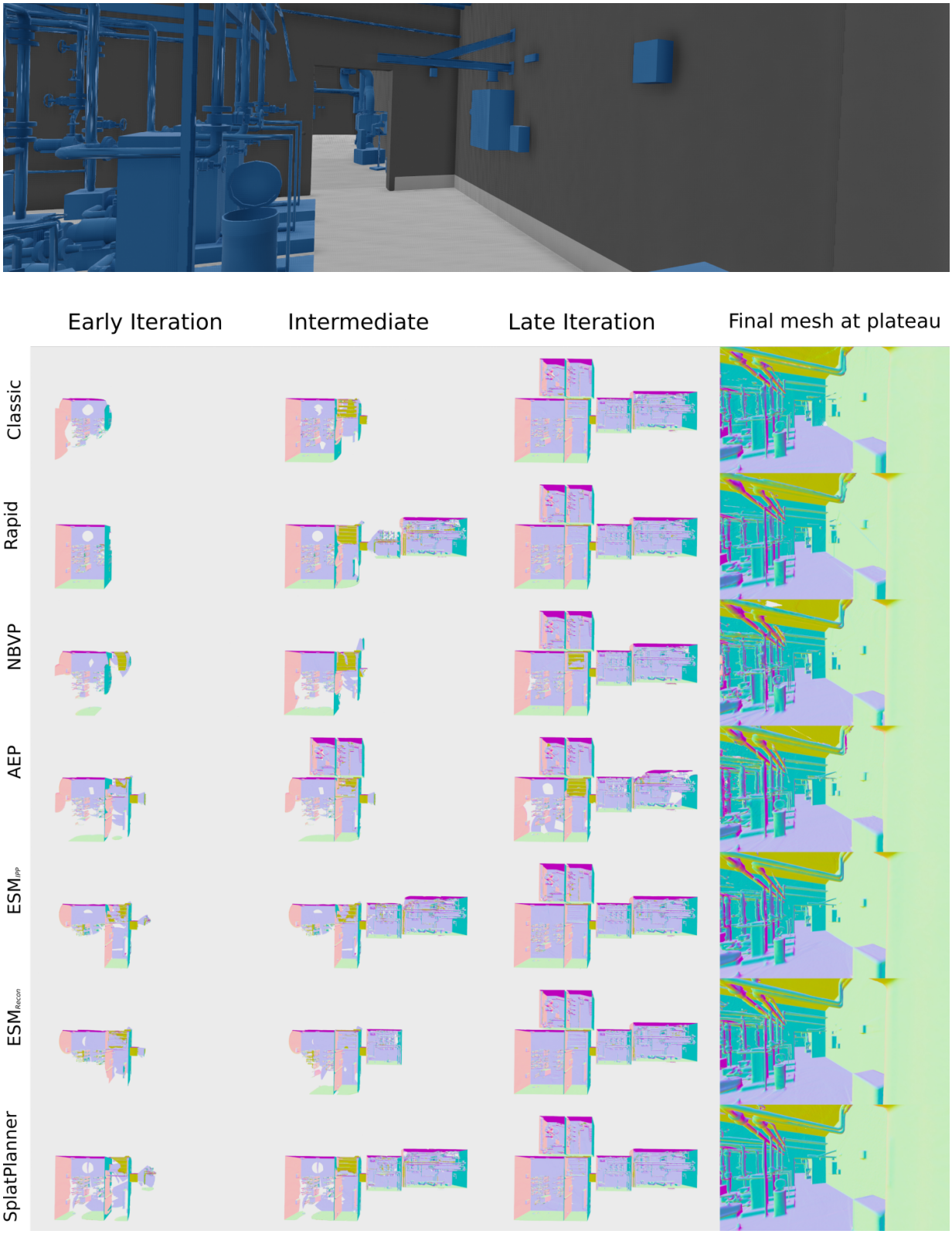


FIGURE .27 – Reconstructions progressives de l'environnement FACILITY C (illustré en modèle CAO ligne du haut) en vue immersive.

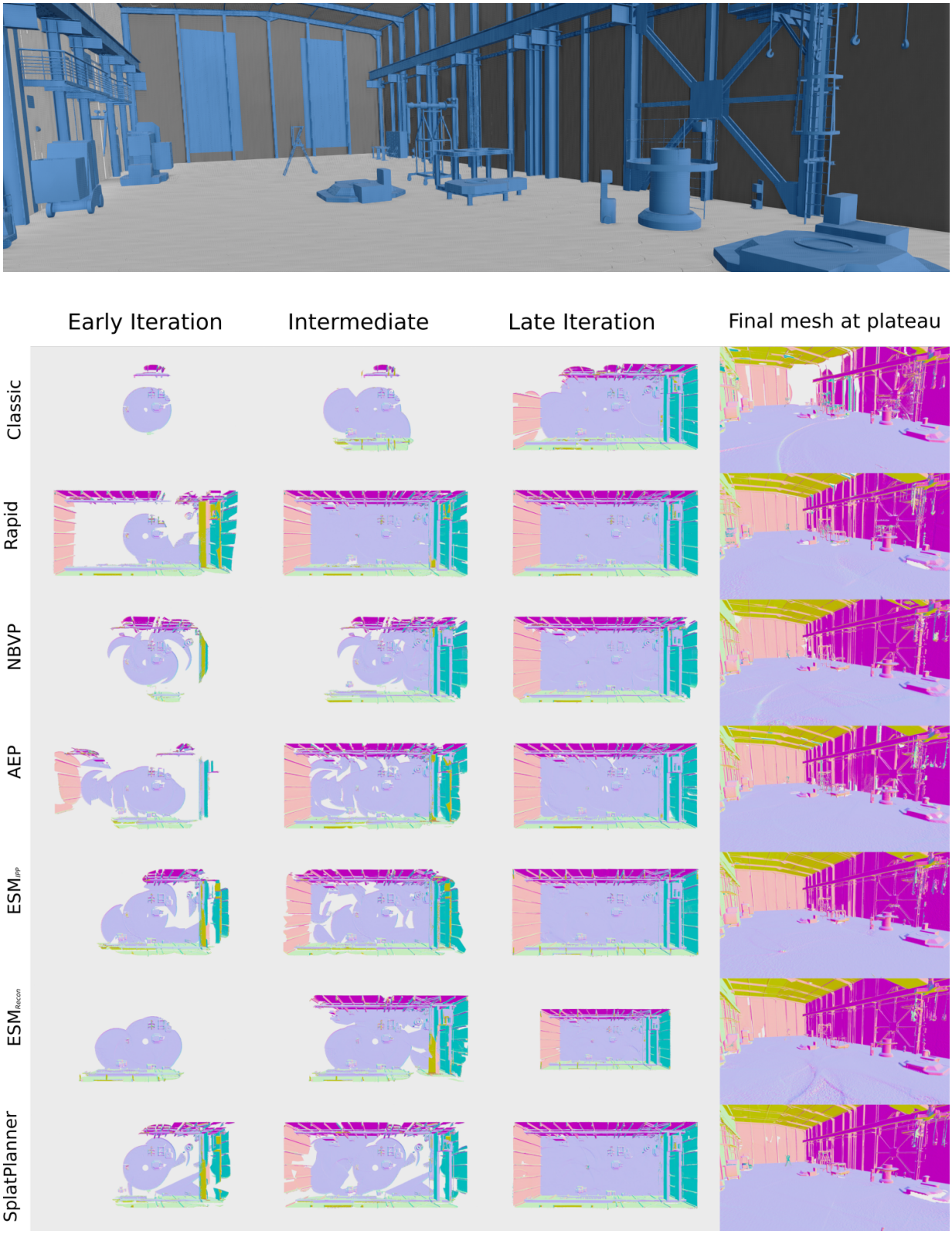


FIGURE .28 – Reconstructions progressives de l'environnement WAREHOUSE (illustré en modèle CAO ligne du haut) en vue immersive.

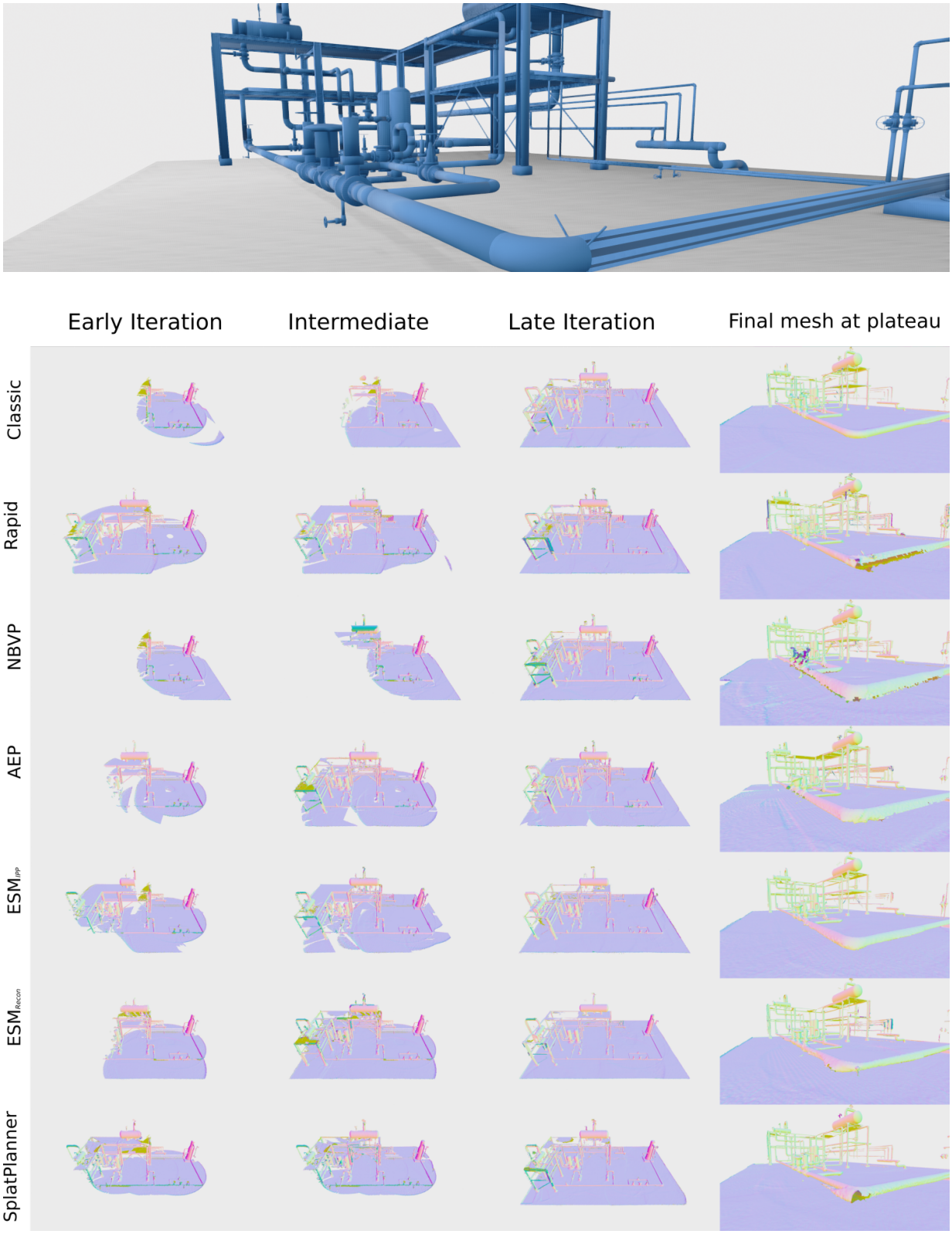


FIGURE .29 – Reconstructions progressives de l'environnement PLATFORM A (illustré en modèle CAO ligne du haut) en vue immersive.

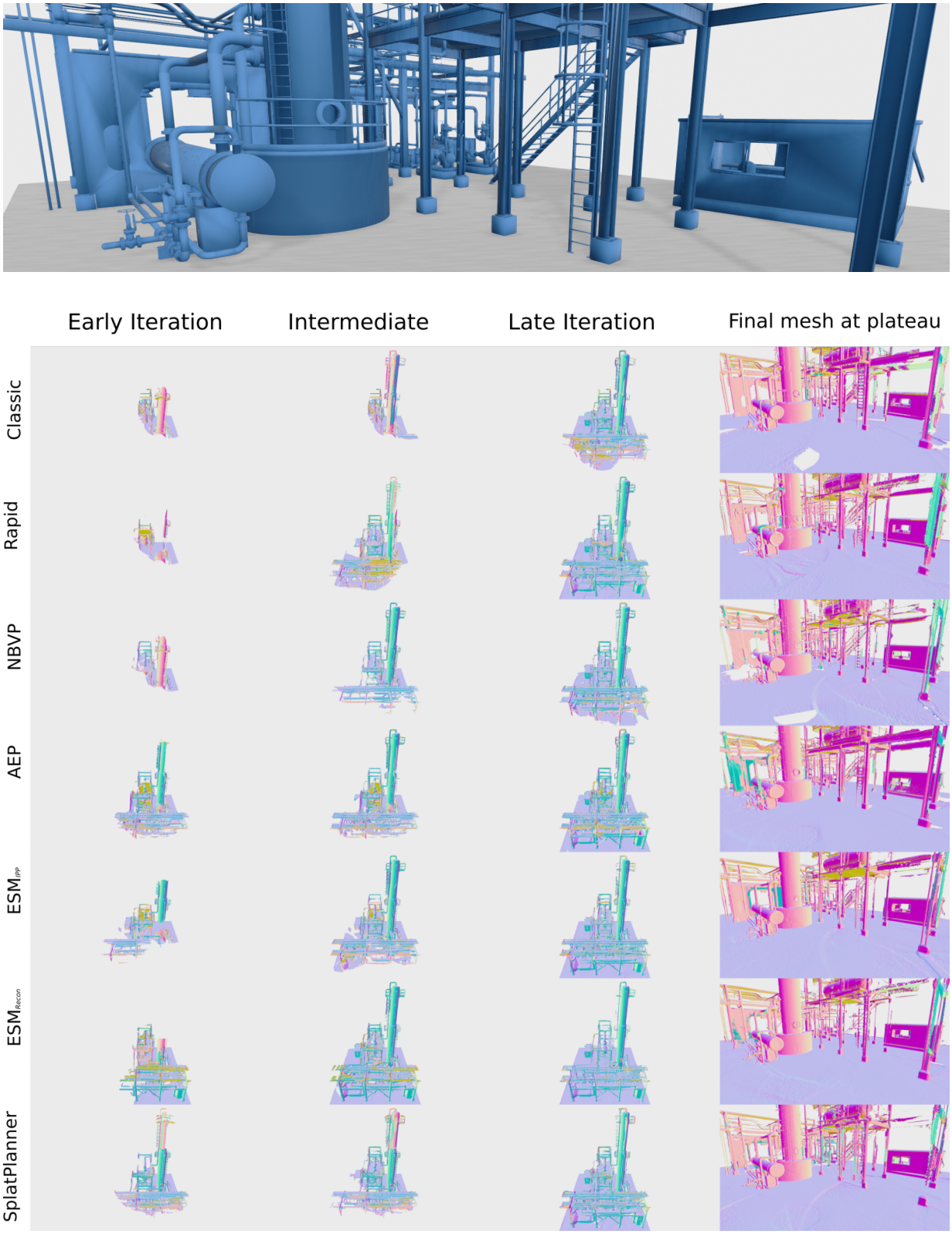


FIGURE .30 – Reconstructions progressives de l'environnement PLATFORM B (illustré en modèle CAO ligne du haut) en vue immersive.

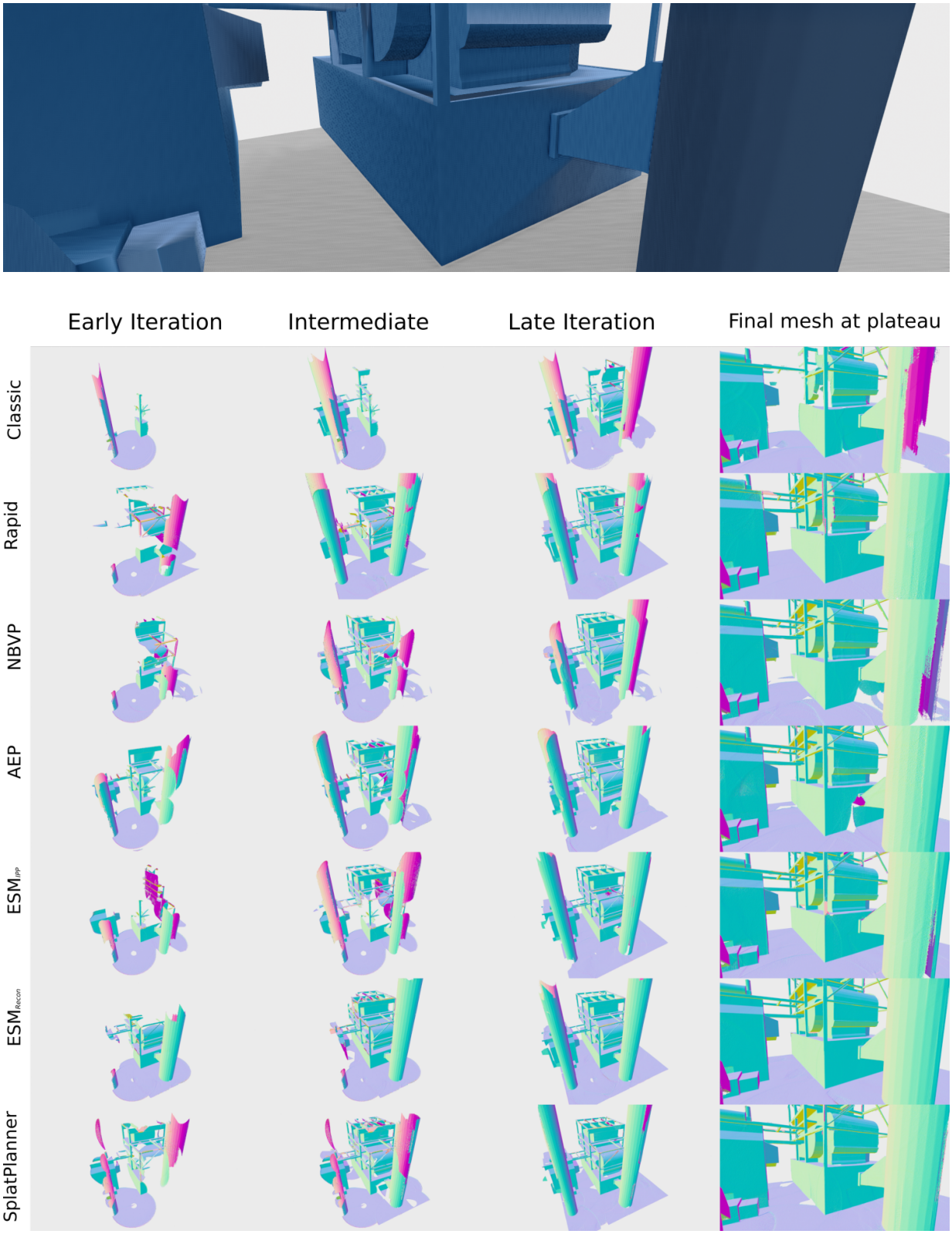


FIGURE .31 – Reconstructions progressives de l'environnement POWERPLANT (illustré en modèle CAO ligne du haut) en vue immersive.

Bibliographie

1. Maja, J. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on robotics and automation* **13,** 304-312 (1992).
2. Yamauchi, B. *A frontier-based approach for autonomous exploration* in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'* (1997), 146-151.
3. Brunel, A., Bourki, A., Demonceaux, C. & Strauss, O. *SplatPlanner: Efficient Autonomous Exploration via Permutohedral Frontier Filtering* in *IEEE International Conference on Robotics and Automation (ICRA 2021)* (2021).
4. Smith, R. C. & Cheeseman, P. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research* **5,** 56-68 (1986).
5. Durrant-Whyte, H. & Bailey, T. Simultaneous localization and mapping: part I. *IEEE robotics & automation magazine* **13,** 99-110 (2006).
6. Bailey, T. & Durrant-Whyte, H. Simultaneous localization and mapping (SLAM): Part II. *IEEE robotics & automation magazine* **13,** 108-117 (2006).
7. Dissanayake, G., Huang, S., Wang, Z. & Ranasinghe, R. *A review of recent developments in simultaneous localization and mapping* in *2011 6th International Conference on Industrial and Information Systems* (2011), 477-482.
8. Cadena, C. *et al.* Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics* **32,** 1309-1332 (2016).
9. Engel, J., Schöps, T. & Cremers, D. *LSD-SLAM: Large-scale direct monocular SLAM* in *European conference on computer vision* (2014), 834-849.
10. Mur-Artal, R., Montiel, J. M. M. & Tardos, J. D. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE transactions on robotics* **31,** 1147-1163 (2015).
11. Mourikis, A. I. & Roumeliotis, S. I. *A multi-state constraint Kalman filter for vision-aided inertial navigation* in *Proceedings 2007 IEEE International Conference on Robotics and Automation* (2007), 3565-3572.
12. Welch, G., Bishop, G. *et al.* An introduction to the Kalman filter (1995).
13. Strasdat, H., Montiel, J. & Davison, A. J. *Real-time monocular SLAM: Why filter?* in *2010 IEEE International Conference on Robotics and Automation* (2010), 2657-2664.

14. Cvišić, I., Ćesić, J., Marković, I. & Petrović, I. SOFT-SLAM: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles. *Journal of field robotics* **35,** 578-595 (2018).
15. Grisetti, G., Kümmerle, R., Strasdat, H. & Konolige, K. *g2o: A general framework for (hyper) graph optimization* in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China* (2011), 9-13.
16. Agarwal, S., Mierle, K. *et al. Ceres Solver* `http://ceres-solver.org`.
17. *GTSAM* `https://gtsam.org`.
18. Triggs, B., McLauchlan, P. F., Hartley, R. I. & Fitzgibbon, A. W. *Bundle adjustment—a modern synthesis* in *International workshop on vision algorithms* (1999), 298-372.
19. Rublee, E., Rabaud, V., Konolige, K. & Bradski, G. *ORB: An efficient alternative to SIFT or SURF* in *2011 International conference on computer vision* (2011), 2564-2571.
20. Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* **60,** 91-110 (2004).
21. Bay, H., Tuytelaars, T. & Gool, L. V. *Surf: Speeded up robust features* in *European conference on computer vision* (2006), 404-417.
22. Qin, T., Li, P. & Shen, S. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics* **34,** 1004-1020 (2018).
23. Usenko, V., Demmel, N., Schubert, D., Stückler, J. & Cremers, D. Visual-inertial mapping with non-linear factor recovery. *IEEE Robotics and Automation Letters* **5,** 422-429 (2019).
24. Klein, G. & Murray, D. *Parallel tracking and mapping for small AR workspaces* in *2007 6th IEEE and ACM international symposium on mixed and augmented reality* (2007), 225-234.
25. Newcombe, R. A., Lovegrove, S. J. & Davison, A. J. *DTAM: Dense tracking and mapping in real-time* in *2011 international conference on computer vision* (2011), 2320-2327.
26. Engel, J., Koltun, V. & Cremers, D. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence* **40,** 611-625 (2017).
27. Ranftl, R., Vineet, V., Chen, Q. & Koltun, V. *Dense monocular depth estimation in complex dynamic scenes* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 4058-4066.
28. Valgaerts, L., Bruhn, A., Mainberger, M. & Weickert, J. Dense versus sparse approaches for estimating the fundamental matrix. *International Journal of Computer Vision* **96,** 212-234 (2012).

29. Mur-Artal, R. & Tardós, J. D. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics* **33,** 1255-1262 (2017).

30. Campos, C., Elvira, R., Rodríguez, J. J. G., Montiel, J. M. & Tardós, J. D. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics* (2021).

31. Qin, T., Li, P. & Shen, S. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics* **34,** 1004-1020 (2018).

32. Lupton, T. & Sukkarieh, S. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Transactions on Robotics* **28,** 61-76 (2011).

33. Mur-Artal, R. & Tardós, J. D. Visual-inertial monocular SLAM with map reuse. *IEEE Robotics and Automation Letters* **2,** 796-803 (2017).

34. Besl, P. J. & McKay, N. D. *Method for registration of 3-D shapes* in *Sensor fusion IV: control paradigms and data structures* **1611** (1992), 586-606.

35. Low, K.-L. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina* **4,** 1-3 (2004).

36. Segal, A., Haehnel, D. & Thrun, S. *Generalized-icp.* in *Robotics: science and systems* **2** (2009), 435.

37. Shao, W., Vijayarangan, S., Li, C. & Kantor, G. *Stereo visual inertial lidar simultaneous localization and mapping* in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), 370-377.

38. Wisth, D., Camurri, M. & Fallon, M. VILENS: Visual, inertial, lidar, and leg odometry for all-terrain legged robots. *arXiv preprint arXiv:2107.07243* (2021).

39. Forster, C., Pizzoli, M. & Scaramuzza, D. *SVO: Fast Semi-Direct Monocular Visual Odometry* in *IEEE International Conference on Robotics and Automation (ICRA)* (2014).

40. Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R. & Furgale, P. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research* **34,** 314-334 (2015).

41. Bloesch, M., Omari, S., Hutter, M. & Siegwart, R. *Robust visual inertial odometry using a direct EKF-based approach* in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (2015), 298-304.

42. Matsuki, H., von Stumberg, L., Usenko, V., Stückler, J. & Cremers, D. Omnidirectional DSO: Direct sparse odometry with fisheye cameras. *IEEE Robotics and Automation Letters* **3,** 3693-3700 (2018).

43. Wang, R., Schworer, M. & Cremers, D. *Stereo DSO: Large-scale direct sparse visual odometry with stereo cameras* in *Proceedings of the IEEE International Conference on Computer Vision* (2017), 3903-3911.

44. Von Stumberg, L., Usenko, V. & Cremers, D. *Direct sparse visual-inertial odometry using dynamic marginalization* in *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), 2510-2517.
45. Rosinol, A., Abate, M., Chang, Y. & Carlone, L. *Kimera: an open-source library for real-time metric-semantic localization and mapping* in *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), 1689-1696.
46. `https://www.intelrealsense.com/tracking-camera-t265/`.
47. Zhang, J. & Singh, S. *LOAM: Lidar Odometry and Mapping in Real-time.* in *Robotics: Science and Systems* **2** (2014).
48. Shan, T. & Englot, B. *Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), 4758-4765.
49. Qin, C. *et al. LINS: A Lidar-Inertial State Estimator for Robust and Efficient Navigation* in *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), 8899-8906.
50. Shan, T. *et al. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping* in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), 5135-5142.
51. Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. & Burgard, W. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous robots* **34,** 189-206 (2013).
52. Moravec, H. & Elfes, A. *High resolution maps from wide angle sonar* in *Proceedings. 1985 IEEE international conference on robotics and automation* **2** (1985), 116-121.
53. Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R. & Nieto, J. *Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning* in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), 1366-1373.
54. Niessner, M., Zollhöfer, M., Izadi, S. & Stamminger, M. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)* **32,** 1-11 (2013).
55. Klingensmith, M., Dryanovski, I., Srinivasa, S. S. & Xiao, J. *Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially Hashed Signed Distance Fields.* in *Robotics: science and systems* **4** (2015).
56. Duberg, D. & Jensfelt, P. UFOMap: An efficient probabilistic 3D mapping framework that embraces the unknown. *IEEE Robotics and Automation Letters* **5,** 6411-6418 (2020).
57. Vespa, E. *et al.* Efficient octree-based volumetric SLAM supporting signed-distance and occupancy mapping. *IEEE Robotics and Automation Letters* **3,** 1144-1151 (2018).

58. Han, L., Gao, F., Zhou, B. & Shen, S. *Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots* in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), 4423-4430.

59. Felzenszwalb, P. F. & Huttenlocher, D. P. Distance transforms of sampled functions. *Theory of computing* **8,** 415-428 (2012).

60. Usenko, V., Von Stumberg, L., Pangercic, A. & Cremers, D. *Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer* in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), 215-222.

61. Zhou, B., Gao, F., Wang, L., Liu, C. & Shen, S. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters* **4,** 3529-3536 (2019).

62. Brunel, A., Bourki, A., Strauss, O. & Demonceaux, C. *FLYBO: A Unified Benchmark Environment for Autonomous Flying Robots* in *2021 International Conference on 3D Vision (3DV)* (2021), 1420-1431.

63. Moravec, H. P. *Sensor fusion in certainty grids for mobile robots* in *Sensor devices and systems for robotics* (Springer, 1989), 253-276.

64. Thrun, S. Probabilistic robotics. *Communications of the ACM* **45,** 52-57 (2002).

65. Loop, C., Cai, Q., Orts-Escolano, S. & Chou, P. A. *A closed-form bayesian fusion equation using occupancy probabilities* in *2016 Fourth International Conference on 3D Vision (3DV)* (2016), 380-388.

66. Curless, B. & Levoy, M. *A volumetric method for building complex models from range images* in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), 303-312.

67. Lorensen, W. E. & Cline, H. E. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics* **21,** 163-169 (1987).

68. Dijkstra, E. W. *et al.* A note on two problems in connexion with graphs. *Numerische mathematik* **1,** 269-271 (1959).

69. Cieslewski, T., Kaufmann, E. & Scaramuzza, D. *Rapid exploration with multi-rotors: A frontier selection method for high speed flight* in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), 2135-2142.

70. Hart, P. E., Nilsson, N. J. & Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* **4,** 100-107 (1968).

71. Stentz, A. *et al. The focussed d^* algorithm for real-time replanning* in *IJCAI* **95** (1995), 1652-1659.

72. Stentz, A. *Optimal and efficient path planning for partially known environments* in *Intelligent unmanned ground vehicles* (Springer, 1997), 203-220.

73. Barraquand, J. & Latombe, J.-C. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research* **10,** 628-649 (1991).
74. Sucan, I. A., Moll, M. & Kavraki, L. E. The open motion planning library. *IEEE Robotics & Automation Magazine* **19,** 72-82 (2012).
75. Amato, N. M. & Wu, Y. *A randomized roadmap method for path and manipulation planning* in *Proceedings of IEEE international conference on robotics and automation* **1** (1996), 113-120.
76. Kavraki, L. E., Svestka, P., Latombe, J.-C. & Overmars, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* **12,** 566-580 (1996).
77. LaValle, S. M. *et al.* Rapidly-exploring random trees: A new tool for path planning (1998).
78. Karaman, S. & Frazzoli, E. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* **30,** 846-894 (2011).
79. Otte, M. & Frazzoli, E. *RRT^X: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles* in *Algorithmic Foundations of Robotics XI* (Springer, 2015), 461-478.
80. Gammell, J. D., Srinivasa, S. S. & Barfoot, T. D. *$InformedRRT^*$: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic* in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2014), 2997-3004.
81. Elbanhawi, M. & Simic, M. Sampling-based robot motion planning: A review. *Ieee access* **2,** 56-77 (2014).
82. Richter, C., Bry, A. & Roy, N. *Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments* in *Robotics research* (Springer, 2016), 649-666.
83. Burri, M., Oleynikova, H., Achtelik, M. W. & Siegwart, R. *Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments* in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), 1872-1878.
84. Liu, S. *et al.* Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters* **2,** 1688-1695 (2017).
85. Harabor, D. & Grastien, A. *Online graph pruning for pathfinding on grid maps* in *Proceedings of the AAAI Conference on Artificial Intelligence* **25** (2011).
86. Tordesillas, J., Lopez, B. T. & How, J. P. *Faster: Fast and safe trajectory planner for flights in unknown environments* in *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (2019), 1934-1940.

87. Dolgov, D., Thrun, S., Montemerlo, M. & Diebel, J. Path planning for autonomous vehicles in unknown semi-structured environments. *The international journal of robotics research* **29,** 485-501 (2010).

88. LaValle, S. M. & Kuffner Jr, J. J. Randomized kinodynamic planning. *The international journal of robotics research* **20,** 378-400 (2001).

89. Webb, D. J. & Berg, J. v. d. Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints. *arXiv preprint arXiv:1205.5088* (2012).

90. Zhou, X., Wang, Z., Ye, H., Xu, C. & Gao, F. EGO-Planner: An ESDF-free Gradient-based Local Planner for Quadrotors. *IEEE Robotics and Automation Letters* **6,** 478-485 (2020).

91. Oleynikova, H. *et al. Continuous-time trajectory optimization for online UAV replanning* in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (2016), 5332-5339.

92. Ding, W., Gao, W., Wang, K. & Shen, S. An efficient b-spline-based kinodynamic replanning framework for quadrotors. *IEEE Transactions on Robotics* **35,** 1287-1306 (2019).

93. Head, J. D. & Zerner, M. C. A Broyden—Fletcher—Goldfarb—Shanno optimization procedure for molecular geometries. *Chemical physics letters* **122,** 264-270 (1985).

94. Foehn, P., Romero, A. & Scaramuzza, D. Time-optimal planning for quadrotor waypoint flight. *Science Robotics* **6,** eabh1221 (2021).

95. Song, Y., Steinweg, M., Kaufmann, E. & Scaramuzza, D. Autonomous Drone Racing with Deep Reinforcement Learning (2021).

96. Loquercio, A. *et al.* Learning high-speed flight in the wild. *Science Robotics* **6,** eabg5810 (2021).

97. Mellinger, D. & Kumar, V. *Minimum snap trajectory generation and control for quadrotors* in *2011 IEEE international conference on robotics and automation* (2011), 2520-2525.

98. Furrer, F., Burri, M., Achtelik, M. & Siegwart, R. *Rotors a modular gazebo mav simulator framework* in *Robot operating system (ROS)* (Springer, 2016), 595-625.

99. Shah, S., Dey, D., Lovett, C. & Kapoor, A. *Airsim: High-fidelity visual and physical simulation for autonomous vehicles* in *Field and service robotics* (2018), 621-635.

100. Faessler, M., Franchi, A. & Scaramuzza, D. Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories. *IEEE Robotics and Automation Letters* **3,** 620-626 (2017).

101. Lee, T., Leok, M. & McClamroch, N. H. *Geometric tracking control of a quadrotor UAV on SE (3)* in *49th IEEE conference on decision and control (CDC)* (2010), 5420-5425.

102. Lee, T., Leok, M. & McClamroch, N. H. Nonlinear robust tracking control of a quadrotor UAV on SE (3). *Asian Journal of Control* **15,** 391-408 (2013).

103. Van Nieuwstadt, M. J. & Murray, R. M. Real-time trajectory generation for differentially flat systems. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal* **8,** 995-1020 (1998).

104. Camacho, E. F. & Alba, C. B. *Model predictive control* (Springer science & business media, 2013).

105. Mueller, M. W. & D'Andrea, R. *A model predictive controller for quadrocopter state interception* in *2013 European Control Conference (ECC)* (2013), 1383-1389.

106. Bangura, M. & Mahony, R. Real-time model predictive control for quadrotors. *IFAC Proceedings Volumes* **47,** 11773-11780 (2014).

107. Kamel, M., Alexis, K., Achtelik, M. & Siegwart, R. *Fast nonlinear model predictive control for multicopter attitude tracking on so (3)* in *2015 IEEE Conference on Control Applications (CCA)* (2015), 1160-1166.

108. Kamel, M., Stastny, T., Alexis, K. & Siegwart, R. *Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System* in *Robot Operating System (ROS) The Complete Reference, Volume 2* (éd. Koubaa, A.) (Springer).

109. Kamel, M., Burri, M. & Siegwart, R. Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine* **50,** 3463-3469 (2017).

110. Nguyen, H., Kamel, M., Alexis, K. & Siegwart, R. Model Predictive Control for Micro Aerial Vehicles: A Survey. *arXiv preprint arXiv:2011.11104* (2020).

111. Sun, S., Romero, A., Foehn, P., Kaufmann, E. & Scaramuzza, D. A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight. *arXiv preprint arXiv:2109.01365* (2021).

112. Mueller, M. W. & D'Andrea, R. *Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers* in *2014 IEEE international conference on robotics and automation (ICRA)* (2014), 45-52.

113. Sun, S., Cioffi, G., De Visser, C. & Scaramuzza, D. Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events. *IEEE Robotics and Automation Letters* **6,** 580-587 (2021).

114. Darivianakis, G., Alexis, K., Burri, M. & Siegwart, R. *Hybrid predictive control for aerial robotic physical interaction towards inspection operations* in *2014 IEEE international conference on robotics and automation (ICRA)* (2014), 53-58.

115. De Crousaz, C., Farshidian, F., Neunert, M. & Buchli, J. *Unified motion control for dynamic quadrotor maneuvers demonstrated on slung load and rotor failure tasks* in *2015 IEEE International Conference on Robotics and Automation (ICRA)* (2015), 2223-2229.

116. Bircher, A., Kamel, M., Alexis, K., Oleynikova, H. & Siegwart, R. *Receding horizon "next-best-view" planner for 3D exploration* in *2016 IEEE International Conference on Robotics and Automation (ICRA)* (2016), 1462-1468.

117. Schmid, L. *et al.* An efficient sampling-based method for online informative path planning in unknown environments. *IEEE Robotics and Automation Letters* **5,** 1500-1507 (2020).

118. Selin, M., Tiger, M., Duberg, D., Heintz, F. & Jensfelt, P. Efficient autonomous exploration planning of large-scale 3-d environments. *IEEE Robotics and Automation Letters* **4,** 1699-1706 (2019).

119. Hepp, B. *et al. Learn-to-score: Efficient 3d scene exploration by predicting view utility* in *Proceedings of the European conference on computer vision (ECCV)* (2018), 437-452.

120. Xu, K. *et al.* Autonomous reconstruction of unknown indoor scenes guided by time-varying tensor fields. *ACM Transactions on Graphics (TOG)* **36,** 1-15 (2017).

121. Dong, S. *et al.* Multi-robot collaborative dense scene reconstruction. *ACM Transactions on Graphics (TOG)* **38,** 1-16 (2019).

122. Wullschleger, F. H., Arras, K. O. & Vestli, S. J. *A flexible exploration framework for map building* in *1999 Third European Workshop on Advanced Mobile Robots (Eurobot'99). Proceedings (Cat. No. 99EX355)* (1999), 49-56.

123. Schmidt, D., Luksch, T., Wettach, J. & Berns, K. *Autonomous behavior-based exploration of office environments.* in *ICINCO-RA* (2006), 235-240.

124. Thrun, S. & Bücken, A. *Integrating grid-based and topological maps for mobile robot navigation* in *Proceedings of the national conference on artificial intelligence* (1996), 944-951.

125. Santosh, D., Achar, S. & Jawahar, C. *Autonomous image-based exploration for mobile robot navigation* in *2008 IEEE International Conference on Robotics and Automation* (2008), 2717-2722.

126. Balch, T. *Avoiding the past: A simple but effective strategy for reactive navigation* in *[1993] Proceedings IEEE International Conference on Robotics and Automation* (1993), 678-685.

127. E Silva Jr, E. P., Engel, P. M., Trevisan, M. & Idiart, M. A. Exploration method using harmonic functions. *Robotics and Autonomous Systems* **40,** 25-42 (2002).

128. Holz, D., Basilico, N., Amigoni, F. & Behnke, S. *Evaluating the efficiency of frontier-based exploration strategies* in *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)* (2010), 1-8.

129. Sim, R. & Little, J. J. Autonomous vision-based robotic exploration and mapping using hybrid maps and particle filters. *Image and Vision Computing* **27,** 167-177 (2009).

130. Sim, R. & Little, J. J. *Autonomous vision-based exploration and mapping using hybrid maps and Rao-Blackwellised particle filters* in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2006), 2082-2089.

131. Yamauchi, B. *Frontier-based exploration using multiple robots* in *Proceedings of the second international conference on Autonomous agents* (1998), 47-53.

132. Bachrach, A., He, R. & Roy, N. Autonomous flight in unknown indoor environments. *International Journal of Micro Air Vehicles* **1,** 217-228 (2009).

133. Shen, S., Michael, N. & Kumar, V. *Autonomous indoor 3D exploration with a micro-aerial vehicle* in *2012 IEEE international conference on robotics and automation* (2012), 9-15.

134. Shen, S., Michael, N. & Kumar, V. Stochastic differential equation-based exploration algorithm for autonomous indoor 3D exploration with a micro-aerial vehicle. *The International Journal of Robotics Research* **31,** 1431-1444 (2012).

135. Connolly, C. *The determination of next best views* in *Proceedings. 1985 IEEE international conference on robotics and automation* **2** (1985), 432-435.

136. Tarabanis, K. A., Allen, P. K. & Tsai, R. Y. A survey of sensor planning in computer vision. *IEEE transactions on Robotics and Automation* **11,** 86-104 (1995).

137. Scott, W. R., Roth, G. & Rivest, J.-F. View planning for automated three-dimensional object reconstruction and inspection. *ACM Computing Surveys (CSUR)* **35,** 64-96 (2003).

138. Banta, J. E. *et al. Best-next-view algorithm for three-dimensional scene reconstruction using range images* in *Intelligent Robots and Computer Vision XIV: Algorithms, Techniques, Active Vision, and Materials Handling* **2588** (1995), 418-429.

139. Pito, R. *A sensor-based solution to the" next best view" problem* in *Proceedings of 13th International Conference on Pattern Recognition* **1** (1996), 941-945.

140. Pito, R. A solution to the next best view problem for automated surface acquisition. *IEEE Transactions on pattern analysis and machine intelligence* **21,** 1016-1030 (1999).

141. Kuipers, B. & Byun, Y.-T. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and autonomous systems* **8,** 47-63 (1991).

142. González-Banos, H. & Latombe, J.-C. Planning robot motions for range-image acquisition and automatic 3d model construction (1998).

143. O'rourke, J. *Art gallery theorems and algorithms* (Oxford New York, NY, USA, 1987).

144. González-Banos, H. H. & Latombe, J.-C. Navigation strategies for exploring indoor environments. *The International Journal of Robotics Research* **21,** 829-848 (2002).

145. Surmann, H., Nüchter, A. & Hertzberg, J. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Robotics and Autonomous Systems* **45,** 181-198 (2003).

146. Juliá, M., Gil, A. & Reinoso, O. A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. *Autonomous Robots* **33,** 427-444 (2012).

147. Witting, C., Fehr, M., Bähnemann, R., Oleynikova, H. & Siegwart, R. *History-aware autonomous exploration in confined environments using mavs* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), 1-9.

148. Dang, T., Khattak, S., Mascarich, F. & Alexis, K. *Explore locally, plan globally: A path planning framework for autonomous robotic exploration in subterranean environments* in *2019 19th International Conference on Advanced Robotics (ICAR)* (2019), 9-16.

149. Respall, V. M., Devitt, D., Fedorenko, R. & Klimchik, A. *Fast Sampling-based next-best-view exploration algorithm for a MAV* in *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), 89-95.

150. Batinovic, A., Ivanovic, A., Petrovic, T. & Bogdan, S. A Shadowcasting-Based Next-Best-View Planner for Autonomous 3D Exploration. *IEEE Robotics and Automation Letters* (2022).

151. Xu, Z., Deng, D. & Shimada, K. Autonomous UAV exploration of dynamic environments via incremental sampling and probabilistic roadmap. *IEEE Robotics and Automation Letters* **6,** 2729-2736 (2021).

152. Dharmadhikari, M. *et al. Motion primitives-based path planning for fast and agile exploration using aerial robots* in *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), 179-185.

153. Bourgault, F., Makarenko, A. A., Williams, S. B., Grocholsky, B. & Durrant-Whyte, H. F. *Information based adaptive robotic exploration* in *IEEE/RSJ international conference on intelligent robots and systems* **1** (2002), 540-545.

154. Sim, R., Dudek, G. & Roy, N. *Online control policy optimization for minimizing map uncertainty during exploration* in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004* **2** (2004), 1758-1763.

155. Papachristos, C., Khattak, S. & Alexis, K. *Uncertainty-aware receding horizon exploration and mapping using aerial robots* in *2017 IEEE international conference on robotics and automation (ICRA)* (2017), 4568-4575.

156. Dang, T., Papachristos, C. & Alexis, K. *Autonomous exploration and simultaneous object search using aerial robots* in *2018 IEEE Aerospace Conference* (2018), 1-7.

157. Schmid, L., Ni, C., Zhong, Y., Siegwart, R. & Andersson, O. Fast and Compute-efficient Sampling-based Local Exploration Planning via Distribution Learning. *arXiv preprint arXiv:2202.13715* (2022).

158. Kingma, D. P. & Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

159. Sohn, K., Lee, H. & Yan, X. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems* **28** (2015).

160. Stachniss, C. & Burgard, W. *Exploring unknown environments with mobile robots using coverage maps* in *IJCAI* **2003** (2003), 1127-1134.

161. Stachniss, C., Grisetti, G. & Burgard, W. *Information Gain-based Exploration Using Rao-Blackwellized Particle Filters.* in *Robotics: Science and systems* **2** (2005), 65-72.

162. Freda, L. & Oriolo, G. *Frontier-based probabilistic strategies for sensor-based exploration* in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation* (2005), 3881-3887.

163. Freda, L., Loiudice, F. & Oriolo, G. *A randomized method for integrated exploration* in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2006), 2457-2464.

164. Pravitra, C., Chowdhary, G. & Johnson, E. *A compact exploration strategy for indoor flight vehicles* in *2011 50th IEEE Conference on Decision and Control and European Control Conference* (2011), 3572-3577.

165. Oriolo, G., Vendittelli, M., Freda, L. & Troso, G. *The SRT method: Randomized strategies for exploration* in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004* **5** (2004), 4688-4694.

166. Dai, A., Papatheodorou, S., Funk, N., Tzoumanikas, D. & Leutenegger, S. *Fast frontier-based information-driven autonomous exploration with an mav* in *2020 IEEE international conference on robotics and automation (ICRA)* (2020), 9570-9576.

167. Meng, Z. *et al.* A two-stage optimized next-view planning framework for 3-D unknown environment exploration, and structural reconstruction. *IEEE Robotics and Automation Letters* **2,** 1680-1687 (2017).

168. Zhou, B., Zhang, Y., Chen, X. & Shen, S. FUEL: Fast UAV exploration using incremental frontier structure and hierarchical planning. *IEEE Robotics and Automation Letters* **6,** 779-786 (2021).

169. Simmons, R. *et al. Coordination for multi-robot exploration and mapping* in *Aaai/Iaai* (2000), 852-858.

170. Burgard, W., Moors, M. & Schneider, F. *Collaborative exploration of unknown environments with teams of mobile robots* in *Advances in plan-based control of robotic agents* (Springer, 2002), 52-70.

171. Franchi, A., Oriolo, G., Freda, L. & Vendittelli, M. *A decentralized strategy for cooperative robot exploration* in *First International Conference on Robot Communication and Coordination (ROBOCOMM 2007)* (2007), 1-8.

172. Franchi, A., Freda, L., Oriolo, G. & Vendittelli, M. The sensor-based random graph method for cooperative robot exploration. *IEEE/ASME transactions on mechatronics* **14,** 163-175 (2009).

173. Hardouin, G., Morbidi, F., Moras, J., Marzat, J. & Mouaddib, E. M. Surface-driven Next-Best-View planning for exploration of large-scale 3D environments. *IFAC-PapersOnLine* **53,** 15501-15507 (2020).

174. Hardouin, G., Moras, J., Morbidi, F., Marzat, J. & Mouaddib, E. M. *Next-Best-View planning for surface reconstruction of large-scale 3D environments with multiple UAVs* in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), 1567-1574.

175. Gao, Y. *et al.* Meeting-Merging-Mission: A Multi-robot Coordinate Framework for Large-Scale Communication-Limited Exploration. *arXiv preprint arXiv:2109.07764* (2021).

176. Kuffner, J. J. & LaValle, S. M. *RRT-connect: An efficient approach to single-query path planning* in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)* **2** (2000), 995-1001.

177. Adams, A., Baek, J. & Davis, M. A. *Fast high-dimensional filtering using the permutohedral lattice* in *Computer graphics forum* **29** (2010), 753-762.

178. Jampani, V., Kiefel, M. & Gehler, P. V. *Learning sparse high dimensional filters: Image filtering, dense crfs and bilateral neural networks* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), 4452-4461.

179. Su, H. *et al. Splatnet: Sparse lattice networks for point cloud processing* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), 2530-2539.

180. Rosu, R. A., Schütt, P., Quenzel, J. & Behnke, S. *LatticeNet: Fast point cloud segmentation using permutohedral lattices* in *Proc. of Robotics: Science and Systems (RSS)* (2020).

181. Gu, X., Wang, Y., Wu, C., Lee, Y. J. & Wang, P. *Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), 3254-3263.

182. Gao, W. & Tedrake, R. *Filterreg: Robust and efficient probabilistic point-set registration using gaussian filter and twist parameterization* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), 11095-11104.

183. Amanatides, J., Woo, A. *et al. A fast voxel traversal algorithm for ray tracing.* in *Eurographics* **87** (1987), 3-10.

184. Baek, J. & Adams, A. Some useful properties of the permutohedral lattice for Gaussian filtering. *other words* **10,** 0 (2009).

185. Zhang, H. *et al.* Continuous aerial path planning for 3D urban scene reconstruction. *ACM Transactions on Graphics (TOG)* **40,** 1-15 (2021).

186. Song, Y., Naji, S., Kaufmann, E., Loquercio, A. & Scaramuzza, D. *Flightmare: A Flexible Quadrotor Simulator* in *Conference on Robot Learning* (2020).

187. Du Montcel, T. T., Nègre, A., Gomez-Balderas, J.-E. & Marchand, N. BOARR: A benchmark for quadrotor obstacle avoidance based on ROS and RotorS (2019).

188. Isler, S., Sabzevari, R., Delmerico, J. & Scaramuzza, D. *An information gain formulation for active volumetric 3D reconstruction* in *2016 IEEE International Conference on Robotics and Automation (ICRA)* (2016), 3477-3484.

189. Newcombe, R. A. *et al. Kinectfusion: Real-time dense surface mapping and tracking* in *2011 10th IEEE international symposium on mixed and augmented reality* (2011), 127-136.

190. Szeliski, R. in *Computer Vision: Algorithms and Applications* 639-680 (Springer International Publishing, Cham, 2022). isbn : 978-3-030-34372-9. `https://doi.org/10.1007/978-3-030-34372-9_13`.

191. Schönberger, J. L., Zheng, E., Frahm, J.-M. & Pollefeys, M. *Pixelwise view selection for unstructured multi-view stereo* in *European conference on computer vision* (2016), 501-518.

192. Knapitsch, A., Park, J., Zhou, Q.-Y. & Koltun, V. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)* **36,** 1-13 (2017).

193. Chang, A. *et al.* Matterport3D: Learning from RGB-D Data in Indoor Environments. *International Conference on 3D Vision (3DV)* (2017).

194. Dai, A. *et al. Scannet: Richly-annotated 3d reconstructions of indoor scenes* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), 5828-5839.

195. Song, S. *et al. Semantic scene completion from a single depth image* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), 1746-1754.

196. Armeni, I., Sax, S., Zamir, A. R. & Savarese, S. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105* (2017).

197. Xia, F. *et al. Gibson env: Real-world perception for embodied agents* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), 9068-9079.

198. Quigley, M. *et al. ROS: an open-source Robot Operating System* in *ICRA workshop on open source software* **3** (2009), 5.

199. Koenig, N. & Howard, A. *Design and use paradigms for gazebo, an open-source multi-robot simulator* in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)* **3** (2004), 2149-2154.

200. Keselman, L., Iselin Woodfill, J., Grunnet-Jepsen, A. & Bhowmik, A. *Intel realsense stereoscopic depth cameras* in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (2017), 1-10.

201. Handa, A., Whelan, T., McDonald, J. & Davison, A. J. *A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM* in *2014 IEEE international conference on Robotics and automation (ICRA)* (2014), 1524-1531.

202. Zhou, Q.-Y., Park, J. & Koltun, V. Open3D: A modern library for 3D data processing. *arXiv preprint arXiv:1801.09847* (2018).

203. Dong, W., Lao, Y., Kaess, M. & Koltun, V. ASH: A Modern Framework for Parallel Spatial Hashing in 3D Perception. *arXiv preprint arXiv:2110.00511* (2021).

204. Cignoni, P., Rocchini, C. & Scopigno, R. *Metro: measuring error on simplified surfaces* in *Computer graphics forum* **17** (1998), 167-174.

205. Papon, J., Abramov, A., Schoeler, M. & Wörgötter, F. *Voxel Cloud Connectivity Segmentation - Supervoxels for Point Clouds* in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on* (Portland, Oregon, juin 2013).

206. Maye, J., Furgale, P. & Siegwart, R. *Self-supervised calibration for robotic systems* in *2013 IEEE Intelligent Vehicles Symposium (IV)* (2013), 473-480.

207. Sa, I. *et al. Dynamic system identification, and control for a cost-effective and open-source multi-rotor mav* in *Field and Service Robotics* (2018), 605-620.

208. Reijgwart, V. *et al.* Voxgraph: Globally consistent, volumetric mapping using signed distance function submaps. *IEEE Robotics and Automation Letters* **5,** 227-234 (2019).

209. Duhautbout, T., Moras, J. & Marzat, J. *Distributed 3D TSDF manifold mapping for multi-robot systems* in *2019 European Conference on Mobile Robots (ECMR)* (2019), 1-8.

210. Choi, S., Zhou, Q.-Y. & Koltun, V. *Robust reconstruction of indoor scenes* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), 5556-5565.

211. Roberts, M. *et al. Submodular trajectory optimization for aerial 3d scanning* in *Proceedings of the IEEE International Conference on Computer Vision* (2017), 5324-5333.

212. Schmid, L. *et al.* A unified approach for autonomous volumetric exploration of large scale environments under severe odometry drift. *IEEE Robotics and Automation Letters* **6,** 4504-4511 (2021).

213. Cao, C., Zhu, H., Choset, H. & Zhang, J. *Exploring large and complex environments fast and efficiently* in *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), 7781-7787.

214. Ranftl, R., Lasinger, K., Hafner, D., Schindler, K. & Koltun, V. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **44** (2022).

215. Ranftl, R., Bochkovskiy, A. & Koltun, V. Vision Transformers for Dense Prediction. *ICCV* (2021).

216. Zhao, C., Sun, Q., Zhang, C., Tang, Y. & Qian, F. Monocular depth estimation based on deep learning: An overview. *Science China Technological Sciences* **63,** 1612-1627 (2020).

217. Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. *You only look once: Unified, real-time object detection* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 779-788.

218. Yang, Z., Sun, Y., Liu, S., Shen, X. & Jia, J. *Std: Sparse-to-dense 3d object detector for point cloud* in *Proceedings of the IEEE/CVF international conference on computer vision* (2019), 1951-1960.

219. He, C., Zeng, H., Huang, J., Hua, X.-S. & Zhang, L. *Structure aware single-stage 3d object detection from point cloud* in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2020), 11873-11882.

220. Shi, W. & Rajkumar, R. *Point-gnn: Graph neural network for 3d object detection in a point cloud* in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2020), 1711-1719.

221. Minaee, S. *et al.* Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence* (2021).

222. Dai, A. *et al. Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), 4578-4587.

223. Dai, A. & Niessner, M. *Scan2mesh: From unstructured range scans to 3d meshes* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), 5574-5583.

224. Hou, J., Dai, A. & Niessner, M. *RevealNet: Seeing Behind Objects in RGB-D Scans* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), 2098-2107.

225. Ashour, R., Taha, T., Dias, J. M. M., Seneviratne, L. & Almoosa, N. Exploration for Object Mapping Guided by Environmental Semantics using UAVs. *Remote Sensing* **12,** 891 (2020).

226. Choudhury, S. *et al.* Data-driven planning via imitation learning. *The International Journal of Robotics Research* **37,** 1632-1672 (2018).

227. Zhu, D., Li, T., Ho, D., Wang, C. & Meng, M. Q.-H. *Deep reinforcement learning supervised autonomous exploration in office environments* in *2018 IEEE international conference on robotics and automation (ICRA)* (2018), 7548-7555.

228. Niroui, F., Zhang, K., Kashino, Z. & Nejat, G. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. *IEEE Robotics and Automation Letters* **4,** 610-617 (2019).

229. Zhou, B., Pan, J., Gao, F. & Shen, S. Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight. *IEEE Transactions on Robotics* (2021).

230. Mildenhall, B. *et al. Nerf: Representing scenes as neural radiance fields for view synthesis* in *European conference on computer vision* (2020), 405-421.

231. Müller, T., Evans, A., Schied, C. & Keller, A. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* **41,** 102:1-102:15. https://doi.org/10.1145/3528223.3530127 (juill. 2022).