# Simulation d'un essaim de drones autonomes pour la reconnaissance en milieu urbain

*Pôle Intelligence Artificielle*

Projet n°7

Année 2025


**Encadrants :**

Wassila Ouerdane

Jean-Philippe Poli

**Élèves :**

Alice Paré

Arthur De Bom Van Driessche

Clément Carragoso

Joseph Servigne



# 1 Introduction

## 1.1 Client

Notre entreprise partenaire est la start-up Drakkair, composée de trois étudiants/chercheurs : Nicolas Rosal CEO, Clément Mauget CTO et Kenzy Barakat stagiaire (qui est en 3ème année à CentraleSupélec et notre interlocuteur).

Drakkair est une start-up naissante. Le client développe un système d'essaim de drone pour l'exploration de bâtiment, de tunnel etc. L'entreprise en est à ses débuts, elle n'a encore rien produit et n'a pas encore de clients.

Le projet ne comprend pas pour l'instant d'autres parties prenantes majeures du côté de Drakkair. Le client envisage toutefois de "futurs" partenaires comme l'armée ou une entreprise du secteur du BTP.

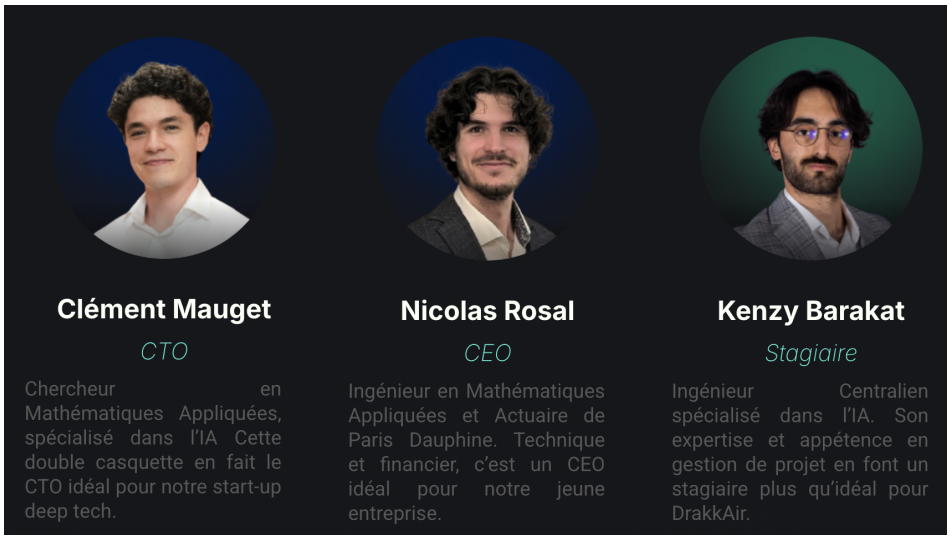


FIGURE 1 – Membres de Drakkair

## 1.2 Problème posé

Certains endroits sont trop complexes pour être cartographiés par un humain (zones dangereuses ou inaccessibles). Les drones semblent donc être une bonne alternative pour se déplacer dans ces environnements. De plus, très peu de modèles actuels permettent une exploration avec plusieurs drones.

Divers papiers de recherche étudient déjà la problématique de la reconnaissance d'environnements par un essaim de drones. Mais aucun environnement de test unifié n'existe, et ceux développés par les équipes de chercheurs ne sont pas adaptés à l'industrie.

Le client cherche donc un algorithme d'exploration permettant de diriger un essaim de plusieurs drones, de manière décentralisée, afin qu'ils puissent communiquer entre eux sans passer par une entité extérieure, se répartir les zones à explorer, cartographier un environnement inconnu et générer des trajectoires. Pour ce faire, il souhaite que soient implémentés plusieurs algorithmes en Python, pour ensuite établir un benchmark et les comparer dans environnement virtuel.

## 1.3 Bénéfice apporté au client

A l'issue de notre travail le client souhaite bénéficier d'une base solide pour pouvoir implémenter son système multi drone. Pour cela il attend disposer des codes en python complets, clairs, commentés, et utilisables des algorithmes issus des deux articles scientifiques distincts (racer et dppm) qui ne comportaient que du pseudo code ou du C++.

Ainsi à l'issue de notre étude le client pourra reposer son choix d'algorithmes d'exploration sur des critères et métriques clairs pouvant être testé et visualisé dans un environnement virtuel de simulation dont la base nous a été fourni par lui.

Ce projet ambitieux n'a pas pu être mené entièrement à son terme dans la limite des 5 mois dont nous disposions et nous détaillerons plus loin les choix que nous avons été amenés à faire.

# 2 État de l'art — Exploration autonome par UAVs

## 2.1 Problématique et cadre

L'exploration autonome consiste à cartographier un espace inconnu en maximisant le *gain d'information* tout en minimisant temps, énergie et risques. Pour les UAVs, ces objectifs heurtent les contraintes dynamiques (autonomie, accélération), la perception limitée et la complexité topologique 3D. Ce chapitre retrace l'évolution des stratégies depuis l'approche *frontier-based* (1997) jusqu'aux architectures temps-optimales (FUEL), décentralisées (RACER), puis légères et robustes (DPPM).

## 2.2 Collecte méthodique des sources

Une recherche systématique (IEEE Xplore, ArXiv 1997–2024) avec les mots-clés *UAV exploration, frontier, NBV, coverage path, multi-robot* a abouti à 86 articles. Les critères de sélection :

1. Impact (citations $> 100$ pour travaux $<2015$, $>25$ sinon) ;
2. Pertinence méthodologique (frontière, NBV, trajectoire, multi-UAV) ;
3. Validation expérimentale (simulation + tests réels).

Au final, 13 références structurantes.

## 2.3 Analyse des contributions majeures

### 2.3.1 Frontier-Based : concept fondateur

**Principe** : sélection du voxel libre adjacent à l'inconnu (frontière) le plus proche. Extensions pour le vol rapide. **Limites** : stratégie locale, re-visites fréquentes, contraintes dynamiques ignorées.

### 2.3.2 Méthodes NBV et hybrides

**Échantillonnage de viewpoints**, évaluation explicite du gain d'information. Bircher utilise RRT, Meng un TSP. Optimisation post-traj. avec CHOMP.

### 2.3.3 Optimisation de trajectoires

Usenko et Zhou introduisent des trajectoires dynamiquement faisables. Replanning B-spline $<$ 40 ms, vol à 4 m/s dans couloirs étroits.

### 2.3.4 FUEL : exploration temps-optimale mono-UAV

FUEL introduit une structure hiérarchique en 3 niveaux et un graphe incrémental (FIS). Gain d'un facteur 3 à 8 sur le temps d'exploration.

### 2.3.5 RACER : collaboration décentralisée

RACER étend FUEL à $< 10$ UAVs avec allocation pair-à-pair (CVRP, LKH-3), hiérarchie locale + couloirs anti-collision. 30 % de gain en temps, 10× moins de paquets que mTSP.

### 2.3.6 DPPM : hiérarchie décentralisée ultra-légère

DPPM propose une abstraction topologique de l'espace (EIS) avec des polytopes pseudo-convexes. Planification locale avec TSP, repositionnement global par itération de valeur. MPPI utilisé pour générer des trajectoires temps-continu sûres et efficaces. Meilleures performances que RACER avec 3× moins de bande passante.

| **Critère** | Frontier | NBV/Hybride ccccccc | FUEL | RACER | DPPM |
|---|---|---|---|---|---|
| Décision | Greedy locale | Sampling+TSP | ATSP hiérarchique | CVRP pair-à-pair | Décision locale + reposition |
| Trajectoire | A* discret | Snap post-proc. | B-spline $T_{min}$ | B-spline + avoid | MPPI temps-continu |
| Réactivité | 0,5 Hz | $\tilde{1}$ Hz | 5 Hz | 5 Hz + inter. | 10 Hz |
| Scalabilité | Mono | Mono | Mono | $\leq 10$ UAVs | $\leq 10$ UAVs |
| Communication | — | — | — | Locale async. | $\sim$15 KB/s |
| Complexité | O(N logN) | O(Samples) | NP-difficile | NP pairwise | polytope + TSP local |

## 2.4 Synthèse comparée

## 2.5 Lacunes et positionnement du projet

— Fusion de cartes robuste sans GNSS encore partiellement résolue.
— Obstacles mobiles peu pris en compte.
— Coût algorithmique élevé pour $N > 100$ clusters (CVRP).
Notre projet vise à adresser la fusion de cartes (1) par un SLAM collaboratif léger, compatible DPPM.

## 2.6 Perspectives de recherche

— Perception active multi-modalité (lidar + caméra).
— Planification par apprentissage distribué (MAPF-RL).
— Interopérabilité ROS 2 (DDS, découverte dynamique).

## Conclusion

En trois décennies, l'exploration UAV est passée de l'heuristique locale (frontier) à l'optimisation hiérarchique (FUEL), puis à la coopération décentralisée (RACER) et enfin à la coordination ultra-légère et efficace (DPPM). L'avenir réside dans l'apprentissage distribué, l'adaptation dynamique et la fusion multi-capteurs.

# 3 Description du travail réalisé

Phrase descriptive pour dire qu'on a fait deux implémentations dppm et racer :

Notre travail à essentiellement consister, dans un premier temps à comprendre les articles scientifiques et à s'approprier leurs multiples concepts notamment à travers leurs sources, mais également en contactant leurs auteurs par exemple. Une fois cette première phase terminée, nous avons implémenté les algorithmes présentés et fait le choix après consultation avec le client et nos enseignants de se limiter à ce cadre et de ne pas traiter la partie test et benchmark quand il était clair que la première phase suffisement complexe necessitait plus de temps.

## 3.1 Environnement virtuel

Nous avons utilisé un environnement virtuel codé avec le module pybullet en python afin de tester nos implémentations. Le module permet d'obtenir facilement la position d'un point dans l'espace, et d'effectuer des raycast pour simuler la camera d'un drone et détecter un obstacle.

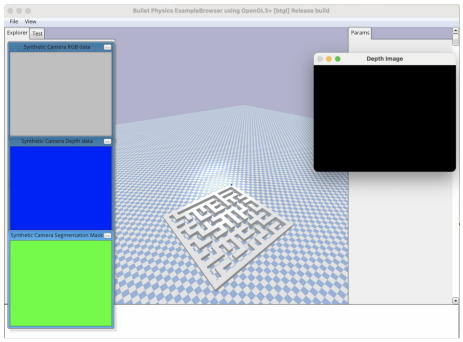


FIGURE 2 – Labyrinthe dans l'environnement virtuel

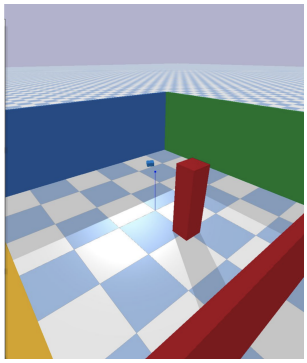

FIGURE 3 – Murs simples dans l'environnement virtuel

## 3.2 DPPM

### 3.2.1 Introduction

Nous avons fait le choix d'implémenter l'algorithme DPPM comme solution au problème d'exploration d'un espace par un essaim de drone.

Aucun code n'est disponible en ligne. Nous avons contacté par mail les auteurs de cet article, qui nous ont fait comprendre que leur code n'était pas open source. Nous sommes donc partis de zéro pour cette implémentation.

Notre première implémentation considère un environnement en 3D et un seul drone. Cette partie a été réalisé par Alice Paré et Arthur De Bom Van Driessche.


FIGURE 4 – Schéma de la structure de DPPM

### 3.2.2 Informations partagées entre les drones

Dans le cadre de l'exploration coopérative multi-UAV, il est essentiel que chaque drone puisse exploiter les informations environnementales découvertes par les autres. Pour cela, une structure d'information légère est proposée afin d'abstraire l'environnement, et chaque UAV maintient un graphe topologique clairsemé représentant la structure spatiale de l'environnement global.

Les noeuds du graphe sont des EIS et les arêtes entre deux noeuds représentent un chemin entre deux EIS.

### 3.2.3 Génération d'EIS

A chaque déplacement, le drone crée un EIS avec sa position pour centre.

**Calcul du polytope pseudo convexe** L'élément central de l'exploration est la **carte de l'espace libre connu**, représentée par un *polytope pseudo-convexe* construit autour de la position courante $p$ d'un UAV.

A chaque déplacement, un ensemble de points $p_{s,k}$ est échantillonné uniformément par le drone autour de $p$, avec :

$$K = K_{xy} \times K_z, \quad k = 1, 2, \ldots, K$$

Pour chaque point, un lancer de rayon est effectué selon la carte d'occupation $\mathcal{M}$ :
— Si le rayon atteint une cellule libre : $p_{s,k} \in S_{Free}$
— Si le rayon est bloqué : le dernier point libre est ajouté à $S_{NFree}$
On définit l'ensemble total de points détectés :

$$S = S_{NFree} \cup S_{Free}$$

Chaque point $s_k \in S$ est projeté sur une sphère unité centrée en $p$ :

$$\hat{s}_k = p + \frac{s_k - p}{\|s_k - p\|}, \quad \hat{S} = \{\hat{s}_k\}$$

L'algorithme `Quickhull` est utilisé pour générer une enveloppe convexe $\hat{S}_{Mesh} \subseteq \hat{S}$, puis les points correspondants dans l'espace initial sont retrouvés :

$$S_{Mesh} = \left\{ s_k \mid s_k = f^{-1}(\hat{x}_k),\ \hat{x}_k \in \hat{S}_{Mesh} \right\}$$

Le polytope pseudo-convexe $\mathcal{P}$ est alors défini comme l'enveloppe de $S_{Mesh}$.

Chaque UAV construit périodiquement une structure d'information appelée *EIS (Exploration Information Structure)*, qui est diffusée aux autres UAVs. Elle contient les données suivantes :
— $p$ : position actuelle de l'UAV
— $\mathcal{P}$ : polytope pseudo-convexe local
— $P_{FC}$ : centres des clusters de frontières détectés
— $K_{FC}$ : nombre de points dans chaque cluster
Cette abstraction permet une exploration coopérative décentralisée plus efficace.

**Clusterisation** L'objectif est ici de construire l'attribut pfc de l'EIS.

### 3.2.4 Mise à jour du graphe

À chaque fois qu'il scanne son environnement, le drone va mettre à jour le graphe partagé. Pour cela, s'il est à une distance suffisante des autres nœuds du graphe, il construit d'abord un EIS comme expliqué précédemment. Puis il ajoute ce nouveau nœud au graphe et créer les arêtes nécessaires entre les nœuds. Pour cela, on parcourt les nœuds voisins et on vérifie s'ils sont dans le nouveau polytope.

Une fois le graphe mis à jour, c'est au tour des frontières de l'être. Dans chaque cluster du nouvelle EIS on vérifie si le ratio de point déjà exploré ( ie : contenues dans un polytope voisin) est supérieur à une constante Kk et le cas échéant le cluster est retiré des frontières de l'EIS et $K_{FC}$ est mise à jour.

**is_point_inside_polytope :** Comme vous avez pu le remarquer, beaucoup des fonctions présentées précédemment utilisent une fonction annexe : is_point_inside_polytope

Cela a été un des points de blocage majeur du projet : comment savoir si un point est dans un polytope ? En effet, il existe beaucoup de solutions et de bibliothèques python pour des polytopes convexe, mais ce n'est pas le cas ici. Pour résoudre cela, nous nous sommes inspirés de l'article : Y. Gao et al., "Meeting-merging-mission : A multi-robot coordinate framework for large-scale communication-limited exploration," in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2022, pp. 13700–13707 cité dans notre papier. Les polytopes étant issus du ray casting de l'environnement depuis une unique position du drone, ils sont étoilés : tout point peut être relié au centre par une droite. Ainsi, après

création du polytope, on dispose d'un mesh découpé en triangle, il suffit alors dans chaque triangle de relier les trois sommets au centre pour décomposer notre polytope en un ensemble de tétraèdre. Ces figures étant convexes, on peut alors utiliser une méthode classique. En l'occurrence pour des tétraèdres, on utilise la méthode des volumes signés dans point_in_tetrahedron :

**Méthode des volumes signés :** Soit un tétraèdre défini par les points $A, B, C, D$, et un point $P$ à tester.

1. Calculer le volume signé du tétraèdre principal :

$$V_0 = \text{vol}(A, B, C, D)$$

2. Calculer les 4 sous-volumes formés avec le point $P$ :

$$V_1 = \text{vol}(P, B, C, D)$$
$$V_2 = \text{vol}(A, P, C, D)$$
$$V_3 = \text{vol}(A, B, P, D)$$
$$V_4 = \text{vol}(A, B, C, P)$$

3. Vérifier les signes :
   Si tous les volumes $V_1, V_2, V_3, V_4$ ont le **même signe que** $V_0$, alors :

$$\boxed{P \text{ est à l'intérieur du tétraèdre } ABCD}$$

### 3.2.5 Déplacement du drone

Le déplacement du drone repose sur une implémentation fidèle de l'algorithme DPPM (Decentralized Planning using a lightweight Information Structure) présenté dans DPPM. Celui-ci s'articule autour d'une stratégie locale et incrémentale de construction de graphe topologique fondée sur des structures appelées *EI* (Exploration Information), représentant localement l'espace libre via des polytopes pseudo-convexes.

Chaque itération du cycle de planification suit deux phases :

— **Exploration initiale (Algo 1)** : le drone génère un nouvel élément EI basé sur la géométrie observable (via `generate_EI`), et le graphe est mis à jour par ajout de sommet et création d'arêtes traversables. Ce processus local conserve la compacité de l'information échangée, suivant la philosophie DPPM.

— **Planification locale (Algo 2)** : une fois un EI ajouté, le drone identifie les clusters de frontières `ei.pfc` non explorés. Un filtrage (`ratio_check`) élimine ceux déjà vus par des polytopes voisins. Ensuite, des viewpoints candidats sont échantillonnés sur le polytope (`generate_candidate_viewpoints_from_poly` et ceux offrant la meilleure visibilité (maximisation de la frontière perçue) sont sélectionnés de façon probabiliste. Un plus court chemin parmi ces viewpoints est ensuite déterminé via une heuristique TSP (`select_best_sequence`).

Le drone se déplace alors successivement vers chaque viewpoint de la séquence optimale. À chaque pas, un nouvel EI est généré, ce qui permet de mettre à jour le graphe global de l'espace libre. Ce mécanisme offre une planification rapide, réactive et robuste aux obstacles, tout en assurant une faible redondance d'exploration grâce au filtrage spatial. Les appels à `is_vector_free_cached` utilisent un test de raycast pour vérifier la navigabilité locale sans obstacle. Le tout est implémenté dans PyBullet, avec visualisation 2D/3D dynamique.

Ce déplacement local, inspiré du module DPPM original, garantit un bon compromis entre complexité embarquée, efficience géométrique et faible coût de communication : autant d'objectifs critiques dans les systèmes multi-UAV décentralisés.

### 3.2.6 Passage à plusieurs drones

L'approche DPPM a été conçue dès l'origine pour être étendue à des systèmes multi-UAV de manière décentralisée et scalable. Cette extension repose sur deux piliers : (i) l'indépendance des modules de planification locaux, et (ii) la synchronisation opportuniste par échange restreint d'informations topologiques. Cette philosophie est directement reprise dans notre architecture expérimentale.

Dans le cas multi-drone, chaque agent maintient son propre graphe topologique local qu'il construit via les modules `Algo1` (exploration initiale) et `Algo2` (planification locale) décrits précédemment. Le partage d'information entre drones s'effectue via la transmission de leurs structures `EI` ou de segments du graphe, uniquement lorsqu'un autre agent est détecté à proximité (par broadcast ou localisation relative). Ce paradigme évite le recours à une fusion centralisée de cartes volumineuses, assurant ainsi une faible bande passante réseau.

Côté implémentation, l'extension multi-agent repose sur l'appel répété de la boucle `explore_viewpoints` pour chaque drone avec ses propres identifiants, états internes et graphes. L'algorithme prévoit également un raffinement de viewpoints (`refined_sample_view_point`) permettant d'éviter la redondance d'observation entre UAVs proches, en intégrant leurs positions et zones de perception connues. Même si cette étape reste encore partiellement codée, sa structure permet l'intégration aisée de politiques de non-recouvrement inspirées du CVRP décentralisé utilisé dans RACER RACER.

Enfin, l'architecture permet de tester des comportements de coopération implicite : par exemple, deux drones partagent passivement les zones explorées en mettant à jour leurs graphes respectifs à la réception d'un nouvel `EI`. Cette synchronisation par fusion légère permet à chaque agent de réorienter sa planification locale sans supervision globale, conformément aux principes DPPM. Cette approche ouvre la voie à une scalabilité effective jusqu'à une dizaine d'UAVs, sous réserve de contraintes GNSS et de gestion de collision.

## 3.3 RACER

### 3.3.1 Introduction

Nous avons décidé d'implémenter l'algorithme présenté dans l'article FUEL afin d'assister notre client. Ce papier décrit une première version de l'exploration autonome pour un drone unique. L'article RACER, initialement fourni par notre client, constitue une extension de FUEL à un essaim de drones, tout en apportant d'autres améliorations.

Malgré l'existence d'un dépôt Git public pour l'implémentation de RACER, nous avons choisi de repartir de zéro. En effet, ce dépôt était mal structuré, et les fonctions étaient principalement codées en `C`, ce qui compliquait la compréhension et la maintenance du code.

Par ailleurs, nous avons décidé que la reconstruction des différents environnements se ferait en deux dimensions, afin de simplifier le traitement des données. Ainsi, nous proposons de livrer à notre client une première version de l'algorithme FUEL, compréhensible et maintenable.

Nous détaillons ci-après les différentes étapes clés de l'implémentation.

Dans une première partie, nous aborderons la phase de reconnaissance par le drone des objets détectés par son *lidar*. Dans une seconde partie, nous traiterons les données ainsi recueillies, que nous regrouperons dans une structure nommée *Frontier Information Structure* (FIS). Enfin, nous discuterons de la phase d'exploration du drone dans son environnement.

### 3.3.2 Sampling-Based Exploration

L'algorithme FUEL se base sur un type d'algorithme connu pour l'exploration autonome, nommé *sampling-based exploration*. Cet algorithme repose sur l'échantillonnage de points de vue vers lesquels le drone doit se déplacer. Ces points de vue sont situés à la frontière entre la zone déjà explorée par le drone et la zone encore inexplorée (voir Figure 5).

On notera ces frontières **FIS** pour *Frontiere-Incremental-Structure*. La mémoire des zones explorées et inexplorées est stockée dans une *voxel grid map*. Ainsi, le parcours consiste à actualiser ces **FIS** à chaque nouvelle mesure du capteur.

Par ailleurs, nous verrons plus tard que les **FIS** ne représentent pas uniquement des frontières, mais constituent également une structure de données contenant d'autres informations.

### 3.3.3 Frontiere-Incremental-Structure

Comme dit précédemment la FIS contient bien plus qu'un tableau de pixel pour décrire une frontière.
Une $\text{FIS}_i$ est créee lorsqu'un nouveau cluster de frontière $F_i$ est détecté (c'est-à-dire que le drone à actualisé une zone inexplorée). Elle stocke toutes les cellules $C_i$ appartenant à la frontière ainsi que la

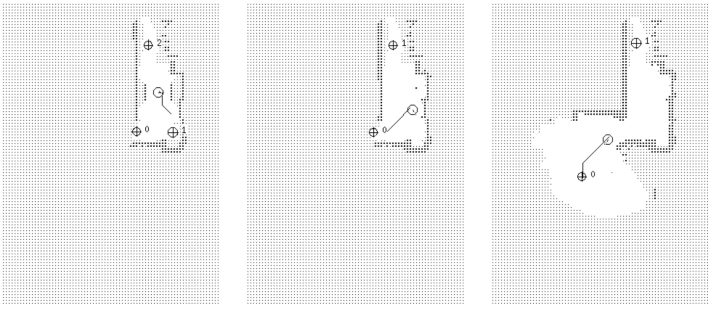

FIGURE 5 – Exemple de frontière FIS entre zones explorées et inexplorées.

position moyenne $p_{\text{avg},i}$ des cellules $C_i$. La boîte englobante alignée sur les axes (Axis-Aligned Bounding Box, AABB) $B_i$ du cluster est également calculée afin d'accélérer la détection des changements de frontière . Pour répondre aux besoins de la planification de l'exploration , des points de vue candidats $\text{VP}_i$ sont générés autour du cluster.

Ainsi, la structure **FIS** est récapitulée dans le tableau 1 ci-dessous.

| **Données** | **Explication** |
|---|---|
| $C_i$ | Liste des indices des cellules de la frontière |
| $p_{\text{avg},i}$ | Position moyenne des voxels du cluster |
| $B_i$ | Volume englobant aligné sur les axes (AABB) |
| $\text{VP}_i$ | Liste des points de vue pour observer $p_{\text{avg},i}$ |

TABLE 1 – Structure de données FIS : résumé des éléments stockés.

### 3.3.4 Notre solution

Pour proposer un code maintenable à notre client, nous avons choisi d'implémenter deux classes principales : `Drone` et `FIS`. Chacune remplit un rôle bien défini dans le système, en encapsulant les comportements et données associés au drone ou à la gestion des frontières d'exploration.

*Classe `Drone` Cette classe représente le drone autonome. Elle contient notamment :

— la position actuelle du drone,

— les fonctions de déplacement et de mise à jour de la carte,

— une interface avec les capteurs pour récupérer les observations.

*Classe `FIS` La classe `FIS` gère la structure de données décrivant les frontières d'exploration. Elle contient :

— pour attribut les données du tableau 1.

— les méthodes de mise à jour en fonction des nouvelles mesures.

Nous détaillerons certaines de ces méthodes plus en détail dans les parties suivantes.

Conversion des données capteurs en carte voxel

La première étape, indépendante de l'algorithme d'exploration, consiste à convertir l'environnement réel perçu par le drone (sa position et la mesure de son capteur) en coordonnées discrètes dans la *voxel grid map*. Cette carte est représentée sous forme d'un tableau 2D d'indices $(i, j)$, correspondant à des cellules de taille fixe. Pour cela, nous avons implémenté deux fonctions essentielles : `world_to_grid`, qui convertit des coordonnées du monde réel vers la grille, et `grid_to_world`, qui effectue l'opération inverse. Nous avons fixé une échelle de conversion de 1 mètre réel pour 30 pixels dans la grille. Les valeurs stockées dans la grille sont codées comme suit : `0` pour les cellules inexplorées, `1` pour les cellules libres, et `2` pour les obstacles détectés. Cette étape permet ainsi de maintenir une carte actualisée de l'environnement, directement exploitable par les algorithmes de planification.

Détection de frontière et clustering

Ensuite, à partir de projections de vecteurs dans la grille, nous obtenons une représentation de la vision du drone lorsqu'il est immobile. Par ailleurs, nous intégrons les différentes structures de frontière **FIS**. Cela nous permet d'obtenir, dans la grille, la Figure 7.

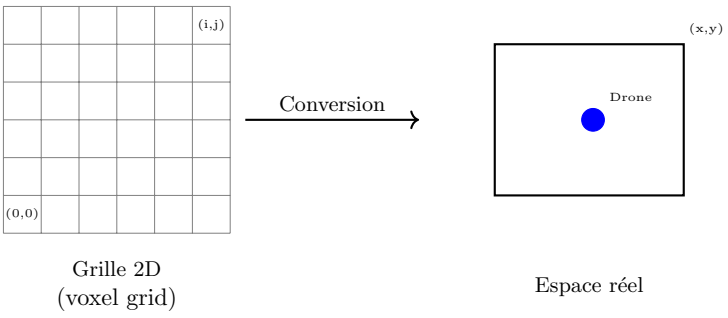


FIGURE 6 – Conversion entre l'espace réel du drone et la grille 2D utilisée pour la cartographie.

Comme on peut le constater sur cette figure, plusieurs FIS ont été construites sur une même frontière (par exemple, celle située à gauche). Ce comportement résulte de l'utilisation de la fonction `find_cluster(cellules, max_range=0.3)`, qui regroupe des cellules (indices de grille) spatialement proches en clusters, en respectant une distance maximale définie par le paramètre `max_range`.

Cette étape facilite le traitement ultérieur des données, notamment la suppression des FIS une fois celles-ci explorées. Le livrable final de cette étape est la fonction `update_voxel_grid_map` de la classe `Drone`, qui intègre l'ensemble de ces mécanismes pour gérer la mise à jour de la grille lorsque le drone est immobile.

L'élimination des clusters de frontière inutiles ou déjà explorés est une étape essentielle. Une implémentation naïve consisterait à comparer chaque cellule avec toutes les autres, ce qui induit une complexité en $\mathcal{O}(n^2)$. Nous avons conçu une approche plus efficace, en profitant des boîtes englobantes alignées sur les axes (*Axis-Aligned Bounding Boxes*, AABB) des FIS. À chaque mesure du capteur, notre algorithme s'exécute comme suit :

```
Bm   = compute_AABB()
L    = FIS_intersection(Bm)
L_r  = check_and_remove(L)
remove_FIS(L_r)
```

Cette méthode réduit la complexité à $\mathcal{O}(n)$, ce qui permet un traitement rapide même en présence d'un grand nombre de structures `FIS`. Une illustration schématique de cet algorithme est présentée dans la Figure 7.

Une fois la gestion des frontières effectuée, celles-ci doivent être explorées. Pour cela, nous proposons de calculer le milieu de chaque nouvelle frontière et de le stocker dans l'attribut `average_position`, afin que le drone puisse observer cette cible.

Notre algorithme procède ensuite à un échantillonnage uniforme de points en 2D autour de cette position cible, représentant les emplacements potentiels où le drone peut se déplacer. Chaque point de vue ainsi généré est ensuite transmis à la méthode `_select_best_viewpoint`, qui permet de déterminer le meilleur point d'observation pour le drone.

La sélection du point de vue optimal repose sur une pondération entre la position actuelle du drone et les différents points de vue, ainsi qu'entre ces points de vue et la position cible.

Enfin, le drone maintient en mémoire une liste de points qu'il doit explorer. Pour planifier efficacement son trajet entre ces points, nous avons choisi d'utiliser l'algorithme **A* (*A-star*)**, qui permet de trouver un chemin optimal en tenant compte à la fois du coût déjà parcouru et d'une estimation heuristique du coût restant.

Voici un schéma de notre algorithme : Figure 8.

### 3.3.5 Le déroulement des tâches

Comme on le voit dans la Figure 8, plusieurs bulles noires sont présentes. Chaque bulle correspond à une tâche réalisée par Clément Carragosso et Joseph Servigne. Joseph était responsable de l'ensemble des tâches.

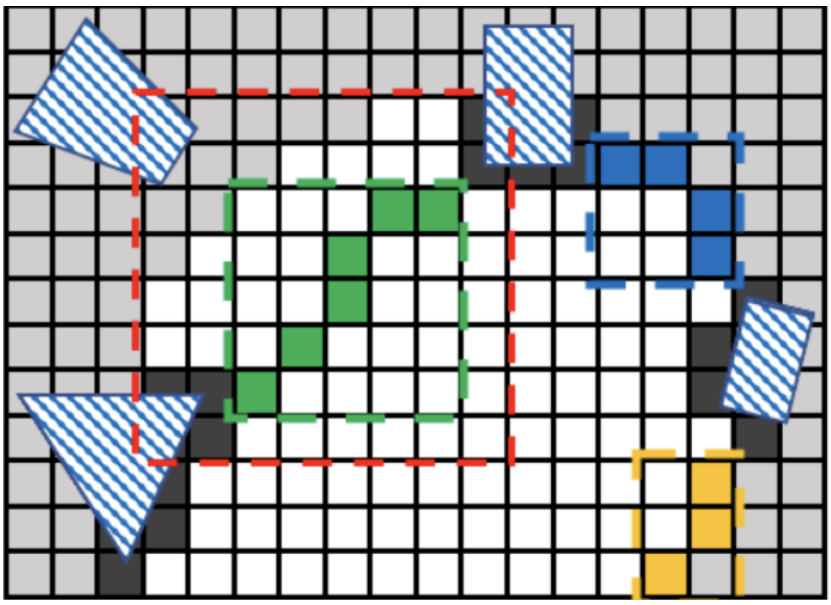

FIGURE 7 – Algorithme rapide d'élimination des clusters redondants via AABB.

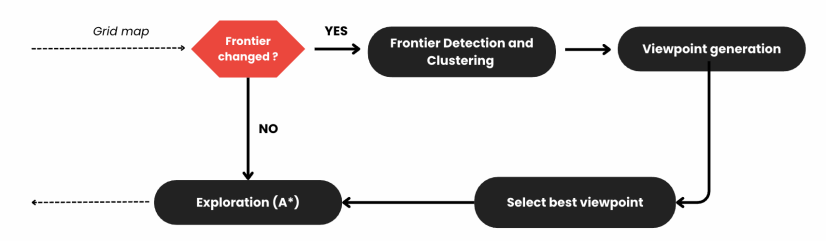


FIGURE 8 – Schéma complet de notre algorithme

Clément a d'abord travaillé pendant environ deux semaines sur la réflexion et l'implémentation du *clustering* des frontières. Il s'est ensuite penché pendant environ trois semaines sur la sélection du meilleur point de vue, puis a consacré une semaine à l'intégration de l'algorithme A* en l'adaptant à notre structure de données.

Joseph s'est chargé du reste des tâches (cf Table 5).

# 4 Conclusion et perspectives

## 4.1 Bilan du projet

Ce projet nous a permis d'approfondir notre compréhension des algorithmes d'exploration multi-robots, et plus particulièrement des méthodes décentralisées s'appuyant sur une planification locale. L'implémentation complète de l'algorithme DPPM et les tests dans un environnement simulé ont renforcé nos compétences en géométrie algorithmique, simulation physique (via PyBullet), planification de trajectoires et visualisation dynamique.

Nous avons également appris à transformer des articles scientifiques complexes — souvent limités à du pseudocode ou à des descriptions théoriques — en modules de code réutilisables, robustes et documentés. Ce travail de mise en œuvre nous a forcés à clarifier de nombreuses zones d'ombre dans les papiers initiaux, ce qui a parfois nécessité des hypothèses ou des ajustements empiriques.

## 4.2 Enseignements et valeur ajoutée

Ce projet a offert une expérience concrète de collaboration avec une start-up en phase amont, dans un contexte de R&D encore exploratoire. La liberté de conception donnée par le client nous a permis d'exercer notre autonomie et de faire preuve d'initiative. En retour, nous avons produit un code modulaire, compatible avec d'autres approches comme RACER, et un environnement de test prêt à être étendu.

Pour le client, ce projet représente une première implémentation pratique des algorithmes DPPM et RACER dans un cadre cohérent, avec visualisation et interfaces simplifiées. Il constitue une base technique utile pour benchmarker les performances de différentes approches d'exploration décentralisée, et pour préparer les tests futurs en conditions plus réalistes.

## 4.3 Limites et perspectives

En raison des contraintes temporelles (5 mois, dont une partie consacrée à la montée en compétence), nous avons volontairement limité notre périmètre à une exploration mono-drone basée sur DPPM. Le passage à plusieurs drones a été amorcé, mais la coordination inter-agents reste à implémenter intégralement (fusion de carte, évitement, synchronisation des cycles).

Plusieurs pistes restent ouvertes :

- **Extension à $N$ drones** avec gestion distribuée de la topologie globale, partage des EI et négociation des viewpoints ;
- **Benchmark complet** entre DPPM et RACER dans des environnements variés (obstacles dynamiques, scènes complexes) ;

## Conclusion

Ce projet nous a apporté une expérience précieuse à l'interface entre recherche algorithmique et développement embarqué. Nous avons conçu un système modulaire, documenté et fonctionnel, qui répond aux besoins exprimés par Drakkair tout en posant les bases d'une poursuite ambitieuse du développement multi-UAV dans les mois à venir.

# 5 Compléments

| **Tâche** | **Description / Méthode** | **Durée** | **Difficultés rencontrées** | **%** |
|---|---|---|---|---|
| Compréhension de l'article DPPM | Relecture approfondie de l'article pour mieux comprendre la structure EI et les algorithmes | 2 semaines | Article dense, abréviations, langage mathématique lourd | 50% |
| Échantillonnage de points 3D | Tirage aléatoire uniforme dans l'espace libre | 1 semaine | Echantillonner uniformément dans l'espace | 100% |
| Quickhull en C++ | Intégration de QuickHull avec Python via subprocess | 2 semaines | Utilisation du C++, appeler du C++ depuis Python, comprendre un code en fait par quelqu'un d'autre | 100% |
| Algo 1 | Génération de EI et ajout au graphe | 1 semaine | Compréhension de l'algorithme du papier | 50% |
| Algo 2 | Échantillonnage et filtrage de viewpoints | 2 semaines | Compréhension de l'algorithme du papier | 50% |
| Adaptation à la simulation | Manipulation de l'environnement PyBullet | 1 semaine | Apprendre à utiliser le module | 100% |

Table 2 – Bilan des tâches réalisées par Alice Paré.

| **Tâche** | **Description / Méthode** | **Durée** | **Difficultés rencontrées** | **%** |
|---|---|---|---|---|
| Compréhension article DPPM | Étude en profondeur des algorithmes et structures de graphe | 2 semaines | Formalisme complexe | 50% |
| Appartenance à un polytope | Vérification géométrique via inclusion dans faces triangulées | 2 semaines | Comprendre le problème géométrique, les polytopes n'étant pas complexes | 100% |
| Algo 1 | Ajout de sommets et arêtes au graphe via EI | 1 semaine | Compréhension de l'algorithme du papier | 50% |
| Algo 2 | Implémentation du ratio-check et viewpoints filtrés | 2 semaines | Compréhension de l'algorithme du papier | 50% |
| Algo 2 | Implémentation de la plannification du meilleur chemin parmi les viewpoints | 2 semaines | Compréhension de l'algorithme du papier | 50% |

Table 3 – Bilan des tâches réalisées par Arthur De Bom Van Driessche.

| **Tâche** | **Description / Méthode** | **Durée** | **Difficultés rencontrées** | **%** |
|---|---|---|---|---|
| Compréhension FUEL et RACER | Analyse des méthodes hiérarchiques et multi-UAV pour exploration rapide | 1.5 semaine | Concepts avancés (CVRP, ATSP, hgrid) | 50% |
| Clustering de frontières | Regroupement de triangles en clusters dans les EI | 2 semaine | Robustesse aux formes irrégulières | 100% |
| Parcours A* | Implémentation pour graphe discret de navigation | 1 semaine | Gestion des poids dynamiques | 100% |
| Sélection de viewpoints | Évaluation probabiliste et chemin optimal simplifié (TSP) | 3 semaine | Réglage de la fonction de reward | 50% |

Table 4 – Bilan des tâches réalisées par Clément Carragoso.

| **Tâche** | **Description / Méthode** | **Durée** | **Difficultés rencontrées** | **%** |
|---|---|---|---|---|
| Compréhension FUEL et RACER | Étude détaillée des articles pour anticiper le passage à plusieurs UAV | 1.5 semaine | Terminologie complexe, nombreux cas limites | 50% |
| Traduction voxel grid | Transformation de l'environnement en carte voxelisée utilisable | 2,5 semaine | Maillage non aligné, pertes de précision | 100% |
| Déplacement du drone | Coordination des fonctions d'exploration et mises à jour du graphe | 2 semaine | Suivi temps réel du drone | 100% |
| Méthodes principales | Orchestration du cycle complet (init + exploration + MAJ) | 3 semaine | Intégration inter-modules parfois instable | 80% |

Table 5 – Bilan des tâches réalisées par Joseph Servigne.