École Polytechnique

PSC INF 12

Surveillance de zone par réseau de drones

Rapport final

Paul Mortamet
Thibault Vignon
Louis Proffit
Louis Stefanuto
X2019

Tuteur :
M. Vincent Jeauneau

Coordinateur :
M. Emmanuel Haucourt

26 avril 2021

Table des matières

Remerciements

Nous tenons particulièrement à remercier :

— notre tuteur Vincent Jeauneau, chez MBDA, pour l'aide qu'il a pu nous apporter grâce à sa connaissance du sujet, tout en nous incitant à être créatifs et à choisir par nous-mêmes les grandes voies à explorer ;
— Emmanuel Haucourt, notre coordinateur du LIX, pour son suivi de notre travail et sa lucidité en juin dernier face à nos objectifs initiaux qui étaient trop ambitieux et mal définis.

Introduction et motivation

Le 9 juillet 2016, la police de Dallas tue un tireur embusqué à l'aide d'un drone explosif. En janvier 2019, une attaque de drones tue six officiers de l'armée loyaliste Yéménite. Le 14 septembre de la même année, des drones armés ravagent un site pétrolier en Arabie Saoudite et détruisent près de la moitié des capacités de production de la société pétrolière, soit 5% des capacités de production mondiales de pétrole brut. Ces quelques exemples parmi tant d'autres illustrent **l'émergence des drones dans les stratégies d'attaque militaires mais aussi terroristes, ainsi que la puissance destructrice de cette technologie.**

Démocratisé à travers une minituarisation des technologies et une commercialisation massive, les drones deviennent une arme parfaitement adaptée aux actes de terrorisme. Ils constitutent donc **une menace pour les sites stratégiques militaires ou civils autant que les concerts publics, les manifestations urbaines, les évènements sportifs ou les meetings politiques.**

De nos jours, aucun système de défense anti-aérien déployé n'a été pensé pour contrer une attaque de drones de taille réduite ou artisanaux. Trop petits, ces robots ne sont pas détectés par les moyens de surveillance militaires actuels et le nombre d'assaillants sature rapidement les moyens de neutralisation. De plus, un système de défense anti-aérien militaire n'est pas utilisable pour contrer des éventuelles attaques lors d'un meeting politique ou une manifestation. Il faut donc **développer des moyens de défenses adaptés aux caractéristiques techniques des drones (leur petite taille, leur nombre, leur vitesse ...) mais aussi adaptable à tous les scénarios d'attaque possibles (militaire et civil).**

En tant qu'élèves-officiers des armées françaises, **mettre en œuvre nos compétences académiques pour apporter une partie de solution à cet enjeu majeur pour les armées et notre pays dans sa globalité**, nous semble crucial. Conscients de l'ampleur du problème, notre PSC se focalise donc sur une solution possible à ce problème qui est **l'utilisation d'un nuage de drones pour défendre une zone potentiellement ciblée**. Cette technique permet d'utiliser la mobilité, la maniabilité, la modularité des drones contre eux-mêmes tout en essayant d'apporter une organisation tactique conséquente, qui est le point faible des attaques de drones terroristes.

1 Etapes et composantes de notre travail

"Surveillance de zone par réseau de drones" est un sujet très large, qui laisse le champ libre pour de nombreuses interprétations et modélisations.

- Le drone : quelles sont ses capacités : vitesse, inertie, capteurs.. A-t-il une autonomie ? Ses capteurs ont-ils des directions priviliégiées ? Doit-il intercepter une menace une fois qu'il l'a détectée, ou laisse-t-il cette mission à une autre entité ? Plus généralement, le "réseau de drones" comporte il beaucoup moins de membres qu'il y a de points à surveiller, environ autant, beaucoup plus ?
- La zone : est-elle infinie, bornée ? Comporte-t'elle des obstacles ? Doit-elle être surveillée uniformément, en certains points uniquement ? Peut-on la diviser en sous-zones ?

En fonctions des réponses à ces questions, les solutions apportées diffèrent. Pour pouvoir traiter correctement le sujet, nous avons du faire des choix, éliminer certains points, pour se concentrer sur l'essentiel. Nous avons cherché à produire les solutions les plus modulaires possibles pour qu'elles puissent facilement être adaptées à une situation précise.

1.1 Représentation de l'espace

Les différentes modélisations du drone influencent moins la solution finale que ne le font les différentes modélisations de la zone. La manière de décrire le terrain change beaucoup la façon de résoudre le problème, notament via les algorithmes utilisables. Voici comment :

1.1.1 Représentation discrète

Dans une représentation discrète, l'espace est modélisé par une grille. Chaque case (2D) ou cube (3D) est défini par un état. Par exemple, pour dans une modélisation simple, ils peuvent être "vide", "drone", "obstacle", "checkpoint" parfois "ennemi"... L'avantage de cette représentation est une très bonne synthèse de l'information. On peut stocker toute celle-ci en une seule matrice. Elle permet également d'utiliser des algorithmes comme le A*, qui est assez efficace dans cette situation. En contrepartie, c'est une représentation contraignante, dans la mesure ou la grille est toujours plus grossière que l'on ne le voudrait. Elle ne permet pas non plus une bonne visualisation, puisque celle-ci ne peut pas se faire de manière continue.

C'est la première représentation que nous avons utilisé, mais que nous avons vite abandonné, pour passer aux modélisations suivantes. Dans la Figure 1, les carrés noirs sont des obstacles, les cercles noirs des checkpoints et le trait est un chemin optimal trouvé par algorithme génétique. Nous détaillerons ces méthodes par la suite.

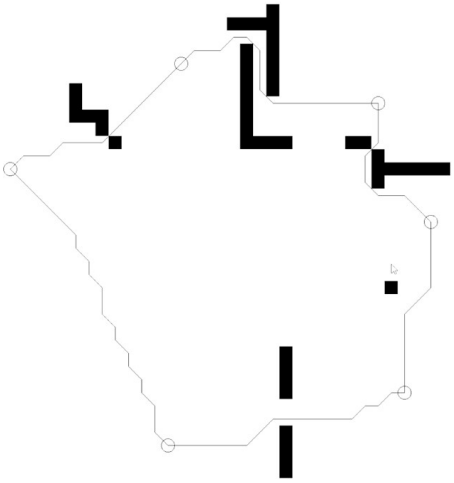


Figure 1 – Représentation discrète de l'espace 2D en 2D, java

1.1.2 Représentation continue

Dans une représentation continue, les différents éléments sont repéres par un jeu de coordonnées (2D, 3D...) continues. Par exemple, un drone ou un checkpoint est représenté par un point, un obstacle est représenté par un polygone, un disque, une sphère... C'est la représentation la plus flexible puisqu'elle permet un nombre de configurations uniquement limité par la précision machine. En contrepartie, cela exclue un certain nombre d'algorithmes comme le A*. Certaines de ses alternatives sont tout aussi efficaces (TRA* par exemple), mais bien plus difficiles à implémenter. Cela limite aussi la quantité d'information utilisable, puisqu'on ne peut définir un état partout dans la zone. Cela exclue entre autres des méthodes comme celle du potentiel (qui sera présentée plus loin), qui nécessite de définir en tout point de l'espace une grandeur réelle.

Une représentation continue est présentée en Figure 2. En 2D, les petits disques sont des checkpoints, les grands sont des drones et les traits définissent le parcours du drone entre les checkpoints de même couleur.

1.1.3 Représentation semi-continue

Pour obtenir un compromis entre les représentations discrètes et les représentations continues, dans certains cas, nous avons choisi d'utiliser une représentation semie-continue de la situation. Par exemple, pour la méthode par potentiels (voir en 2.4, nous définissons un potentiel dans tout l'espace. Ceci étant incompatible avec une représentation continue, on ne définit ce potentiel que sur une grille de l'espace (2D, 3D...), discrète, donc. Cela ne nous empèche pas d'étendre le potentiel en tout point de l'espace. En effet, on peut localiser tout point de l'espace dans un triangle formé de trois points de la grille, et déduire le potentiel en fonction de celui en ces trois points. Cette continuité du potentiel est très utile, à la fois pour définir définir la position du drone (qui peut être continue), que pour évaluer sa trajectoire (par du lancer de rayons, des marches aléatoires...).

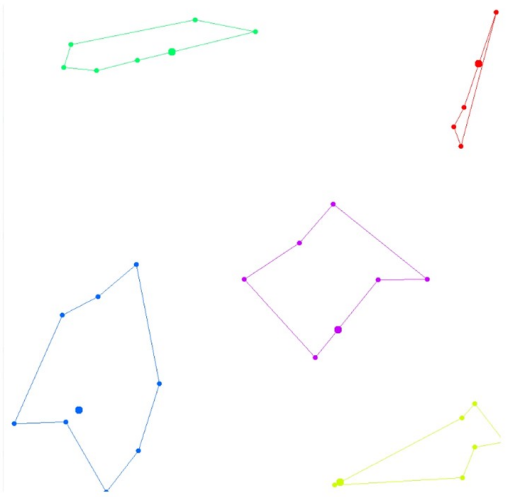


FIGURE 2 – Représentation continue de l'espace 2D en 2D, java

Une représentation semie-continue du potentiel est donnée en Figure 3. La grille et sa triangulation est affichée. A des fins de lisibilité, le pas est très large (seulement 50×50 points). Pour notre utilisation, nous avons un pas largement plus fin : typiquement 200×200. Les zones noires sont les obstacles, les zones rouge sont les minimas de potentiel et les zones vertes sont les maximas. Leur utilisation sera détaillée en partie 2.4.

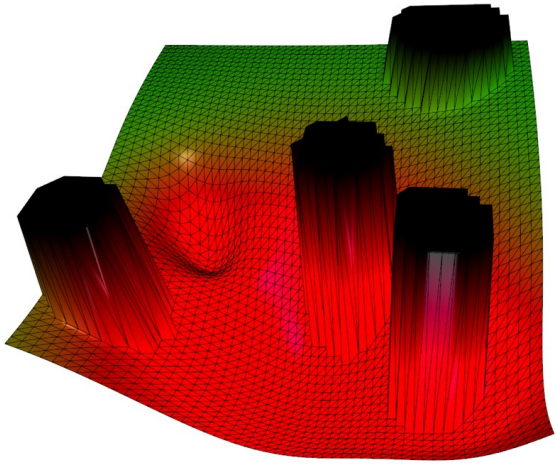


FIGURE 3 – Représentation semie-continue du potentiel 2D en 3D, c++

1.2 Calcul de chemin

La présence d'obstacles dans une zone nous a paru être un élément incontournable de notre sujet. Il est dfficile d'imaginer surveiller une zone complètement ouverte... Nous avons donc eu besoin d'algorithmes de calcul de chemin. La groupe de PSC de la promotion X2018

dont nous avons repris le travail avait concentré ses efforts sur ces algorithmes. Dans la continuité, nous avons utilisé une grande partie de notre temps en début d'année pour de tels algorithmes.

1.2.1 A* et ses extensions

Nous avons tout d'abord travaillé avec une représentation discrète de l'espace, et cherché à traiter le problème de calculer un chemin aussi court que possible passant une fois par chacun des checkpoints et prenant en compte les obstacles. Pour ce faire, nous nous sommes assez naturellement tournés vers l'algorithme A*. Cet algorithme calcule le chemin optimal entre deux nœuds d'un graphe pondéré. Il est basé sur l'algorithme de Dijkstra mais utilise une heuristique pour guider sa recherche vers les nœuds les plus prometteurs. L'heuristique que nous avons utilisée est la somme de la distance déja parcourue et de la distance euclidienne jusqu'au but : en effet, malgré les obstacles, il est souvent efficace de se diriger dans la direction du checkpoint recherché. Nous avons implémenté A* en Java sur notre grille de cases, initialement en stockant les nœuds à visiter dans une simple liste chaînée, puis dans une file de priorité triée selon l'heuristique pour gagner en rapidité. Sur des exemples réalistes, l'heuristique et la file de priorité permettent de rendre A* bien plus rapide que Dijkstra pour l'utilisation que nous en faisons : lors de tests pratiques nous avons ainsi constaté une complexité en temps au moins aussi bonne que $\mathcal{O}(n \log n)$ où n est le nombre de cases du chemin optimal.

TRA* est une extension de l'algorithme A*, adaptée à une réprésentation continue de l'espace. Cet algorithme se base sur une triangulation de l'espace (d'ou le nom) et des obstacles. Celle-ci forme un graphe, qui remplace la grille du simple A*. L'avantage de cet algorithme est qu'il optimise l'utilisation de l'espace : quelle que soit leur taille, les zones sont représentées de la même manière. En contrepartie, l'implémentation est plus difficile. De plus, une manière de calculer la triangulation de Delaunay (et son extension, Laguerre), d'obstacles en 2/3 dimensions, passe justement par une discrétisation de l'espace et d'un traitement via une grille. Cela reste cependant un pré-traitement, qui n'impacte pas la complexité à l'exécution.

Nous avons codé une implémentation de TRA* pour des obstacles polygonaux en 2D, dont les résultats sont détaillés dans le rapport intermédiaire. Nous mentionnons dans celui-ci les problèmes qu'il restait à résoudre. Nous ne nous y sommes pas attelés, nottament parce que nos solutions ont pris une autre direction après ce rapport.

1.2.2 Algorithme génétique

Une fois que nous disposions d'algorithmes rapides pour relier quasi-optimalement deux points en contournant les obstacles, il nous restait à résoudre un problème du voyageur de commerce pour relier nos n checkpoints de manière efficace. Hors de question de tester l'ensemble des $(n-1)!$ configurations possibles, car cela limiterait drastiquement nos options en termes de nombre de checkpoints. Nous avions donc le choix entre une méthode issue de l'optimisation mathématique comme le recuit simulé ou diverses métaheuristiques basées sur

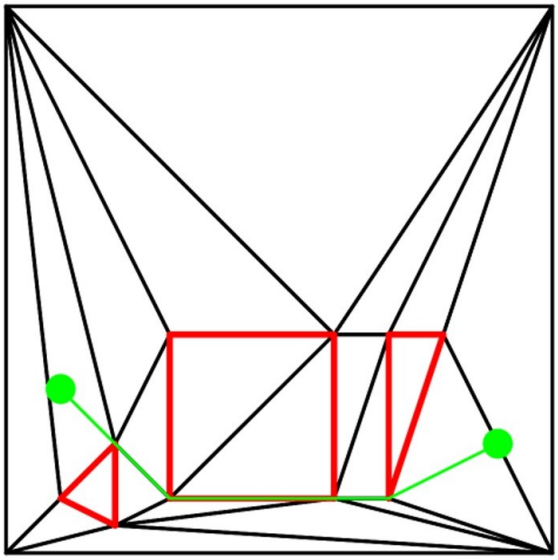


FIGURE 4 – Exemple d'utilisation du TRA*

des populations, comme les algorithmes génétiques ou les algorithmes inspirés des colonies de fourmis. Nous avons choisi d'implémenter d'abord un algorithme génétique.

La structure des algorithmes génétiques est toujours plus ou moins la même : on commence avec une population initiale de solutions générées aléatoirement. Dans notre cas, une solution est un ordre dans lequel on visite les n checkpoints avant de revenir au checkpoint initial. On exécute ensuite un certain nombres d'étapes où on conserve les meilleures solutions et on en génère de nouvelles solutions à partir de celles dont on dispose : cette phase de 'reproduction' des solutions est celle qui donne son nom à l'algorithme. Cette succession d'étapes doit en théorie permettre de converger vers l'optimum.

La principale subtilité réside dans la manière dont on fait se "reproduire" les solutions entre elles. Il faut trouver un mécanisme qui préserve dans une certaine mesure la structure des deux solutions, tout en vérifiant la condition que chaque checkpoint doit être visité exactement une fois. Nous avons choisi le mécanisme illustré ci-dessous : on coupe la première "solution parent" à une place choisie au hasard. On forme la 'solution enfant' en conservant la portion du premier parent qui se trouve avant la coupe, puis en complétant les checkpoints manquants dans l'ordre dans lequel ils apparaissent dans la deuxième solution.

Puisqu'on sélectionne à chaque étape les meilleures solutions, il est naturel de stocker la population de solutions sous la forme d'une file de priorité. Nous avons donc implémenté une file de priorité ordonnée selon la longueur totale de la solution. Pour disposer à chaque instant de celle-ci, au début de l'exécution nous exécutons $\frac{n^2}{2}$ fois l'algorithme A* pour remplir une matrice de coûts qui contient la distance à parcourir entre chaque paire de checkpoints.

Une fois que nous disposions de l'ensemble du back end (A* et algorithme génétique) et du front end (interface graphique), nous avons conduit de nombreux tests. D'abord des tests purement visuels (le parcours proposé semble-t-il efficace à première vue ?), qui ont permis de résoudre plusieurs bugs, et qui ont fini par donner des résultats satisfaisants. Puis des tests statistiques, à l'aide de la classe ScatterChart de JavaFX.

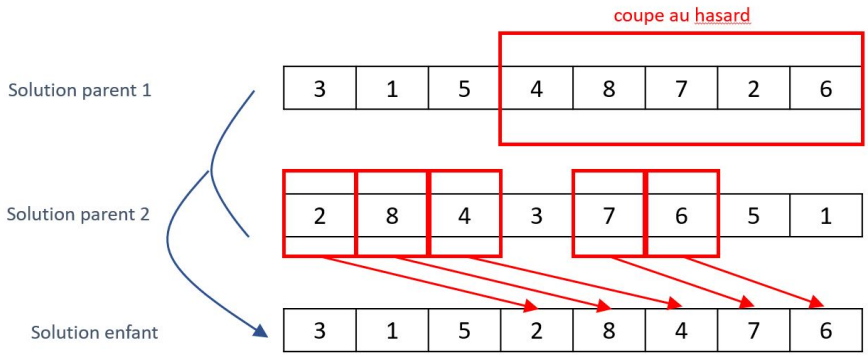


FIGURE 5 – Notre mécanisme de reproduction des solutions

Nous avons codé une fonction qui génère une grille aléatoire avec un certain nombre de checkpoints et une certaine proportion d'obstacles, puis exécute l'algorithme de calcul de parcours sur cette grille avec diverses valeurs des paramètres de l'algorithme génétique : nombre d'étapes et taille de la population. Ci-dessous des tracés de la longueur de la solution trouvée, en fonction de la population à nombres d'étapes constant, puis en fonction du nombre d'étapes à population constante. On constate que la qualité de la solution s'améliore à peu près linéairement avec l'augmentation de la population, mais que l'augmentation du nombre d'étapes a un effet bien moins marqué. Ceci pourrait signifier que le mécanisme de 'reproduction' de notre algorithme génétique est peu efficace. Nous avons cherché d'autres méthodes pour effectuer le croisement entre deux parcours de checkpoints, mais n'avons rien trouvé de plus satisfaisant. Nous nous sommes donc dits qu'un algorithme génétique n'était pas forcément adapté à l'utilisation que nous voulions en faire, et c'est en grande partie pour cette raison que nous avons changé de cap et implémenté un recuit simulé.

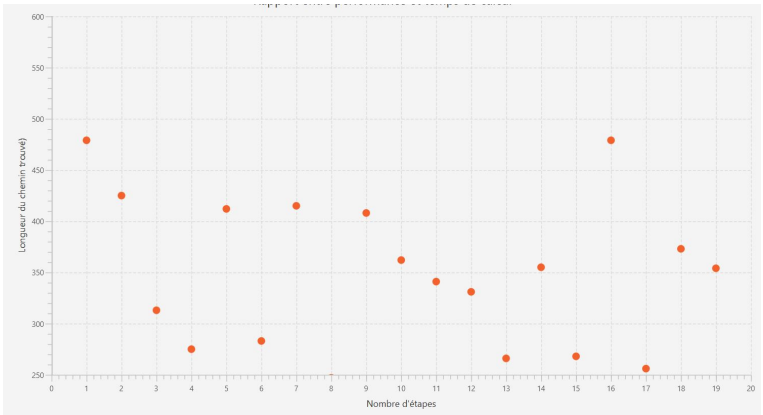


FIGURE 6 – Longueur de la solution en fonction du nombre d'étapes

1.2.3 Recuit simulé

Avec le même objectif de trouver un plus court chemin, nous avons choisi d'utiliser un algorithme de recuit simulé. C'est un algorithme aléatoire itératif qui permet de minimiser

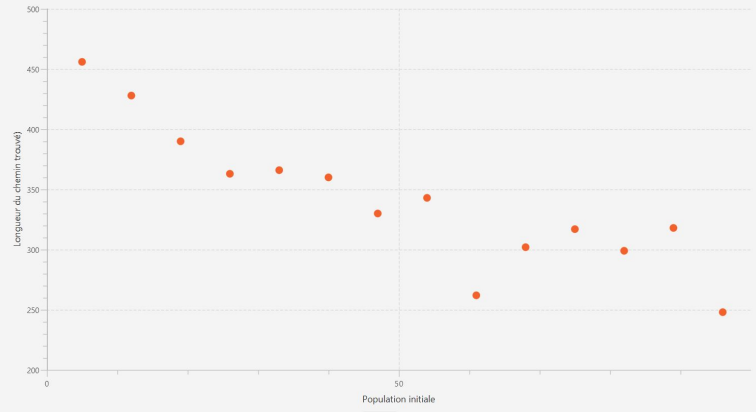


FIGURE 7 – Longueur de la solution en fonction de la population initiale

une fonction de coût. Dans notre cas, la fonction de coût est la longueur totale du chemin. L'implémentation que nous avons effectuée est détaillée en section 2.2. Elle produit de bons résultats.

Nous avons également essayé d'utiliser cet algorithme pour une résolution directe du problème, dans le cas d'une modélisation par checkpoints. La méthode choisie était d'attribuer à chaque drone un ensemble de checkpoints à visiter, avec un certain ordre, et que le drone aille d'un checkpoint au suivant dans cet ordre. La fonction de coût est la somme des taux de rafraichissement des checkpoints : le temps entre deux visites du drone à ce checkpoint. En minimisant la fonction de coût, on pouvait alors trouver une bonne répartition des checkpoints aux drones, et en même temps un bon ordre de parcours des checkpoints pour chaque drone.

Mais pour cette méthode, l'espace de départ (l'ensemble des attributions et ordres possibles) est trop grand. L'algorithme de recuit simulé est incapable de converger vers une solution cohérente. Nous avons fait évoluer cette méthode en introduisant un algorithme de clustering, pour finalement produire la solution que nous présenterons en 2.2.

1.2.4 RRT

Nous avons donc aussi développé un algorithme RRT (Rapidly Exploring Tree) puisque cette approche différait des autres algorithmes et nous permettaient d'utiliser des outils comme le Kd-Tree qui ont pu nous inspirer par la suite. Cet algortihme est aussi réputé pour être très efficace et puissant dans ces versions élaborées. En effet, il existe de nombreuses versions de cet algorithme qui permettent d'affiner le résultat mais le principe reste le même :

1. Tirer un point P_{new} au hasard dans l'espace.
2. Trouver le point $P_{nearest}$ le plus proche de lui parmi ceux déjà dans l'arbre.
3. S'ils sont à une distance supérieure à une distance maximale d_{max} définie au départ, alors il faut rapprocher le nouveau point tiré jusqu'à d_{max} de $P_{nearest}$.
4. Rajouter P_{new} dans l'arbre, relié à $P_{nearest}$.

Nous produisons donc des arbres comme suit dans le but de relier nos deux zones rouges. Le but de nos travaux sur les RRT a été de pousser au maximum l'optimalité des chemins trouvés et la vitesse de convergence de l'algorithme. Nous avons donc produit un *RRT*-Informed* (voir plus loin) parfaitement adapté à nos besoins.

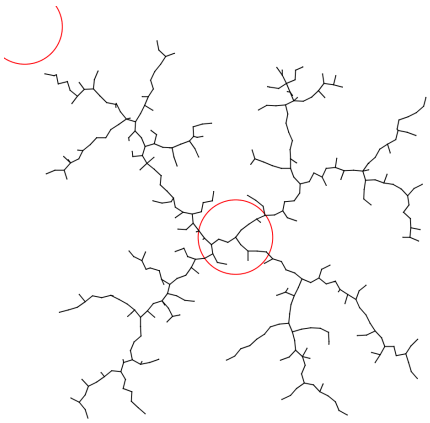


FIGURE 8 – RRT sur $[0.0, 1.0] \times [0.0, 1.0]$ entre les points $(0.5, 0.5)$ et $(0.1, 0.9)$

Comme l'illustre la figure précédente, le RRT produit un chemin où les virages forment des angles autour de 90 degrés : nous produisons donc un chemin peu optimisé et difficilement exécutable dans la réalité. Pour pallier à cet inconvénient, nous sommes passé au RRT* qui reprend le principe du RRT mais ajoute simplement une étape :

1. Tirer un point P_{new} au hasard dans l'espace.
2. Trouver le point le plus proche $P_{nearest}$ de lui parmi ceux déjà dans l'arbre.
3. S'ils sont à une distance supérieure à une distance maximale d_{max} définie au départ, alors rapprocher le nouveau point tiré jusqu'à d_{max} de $P_{nearest}$.
4. Inspecter dans un rayon r défini au départ tous les points autour de P_{new} et regarder si un d'eux pourraient le relier à la racine de l'arbre plus rapidement :
   (a) Si oui, rajouter P_{new} dans l'arbre relié au point trouvé.
   (b) Sinon, rajouter P_{new} dans l'arbre, relié à $P_{nearest}$.
5. Dans ce cercle de rayon r, si un des points pourrait passer par P_{new} pour raccourcir son chemin jusqu'à la racine alors le faire.

Cette simple étape permet de limer l'aspect du chemin et de se débarrasser des angles à 90° qui faisaient la caractéristique du RRT. De cet fait, notre algorithme produit un chemin final entre deux zones plus court et donc optimisé. Dans l'exemple de la figure 9, le chemin est trouvé en moyenne en 240ms.

Nous avons donc réussi à optimiser le chemin trouvé, comme le montre le chemin vert sur la figure précédente . En effet, lorsque l'espace est libre il va en diagonale jusqu'à la prochaine contrainte et il contourne cette dernière en effectuant une déviation serrée autour de l'obstacle. Ainsi, le chemin va au plus vite dans l'espace en naviguant autour des obstacles.

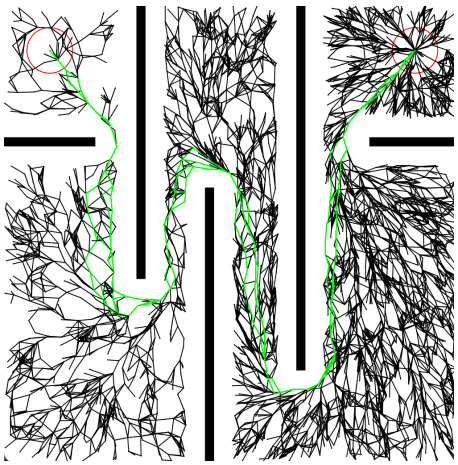
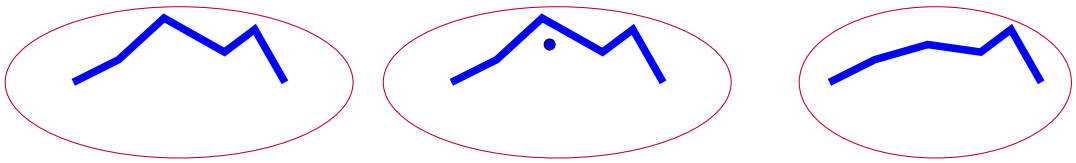


FIGURE 9 – RRT* avec obstacles : obtientention d'un arbre à l'aspect "filaire"

Nous avons ensuite cherché à améliorer la vitesse de convergence, en implémentant un *RRT*-Informed*. Il repose sur le théorème suivant : une fois qu'un chemin est trouvé entre les deux points, si ce chemin venait à être améliorer, ce ne serait que par l'ajout d'un point se situant dans une ellipse où :

- Les foyers sont le point de départ et d'arrivée.
- Le grand axe vaut la longueur du chemin précédemment trouvé.

La figure suivante illustre le procédé :



Ainsi, une fois qu'un chemin est trouvé, les points sont tirés aléatoirement dans l'ellipse et non dans l'espace entier ce qui réduit l'espace de recherche et donc accélère la convergence. Notons bien évidemment que notre algorithme ne fonctionne pas en tirant un point dans l'espace tout entier jusqu'à ce qu'il soit bien dans notre ellipse. Cette méthode est contre productive puisque l'intérêt de cette méthode est bien que plus l'on améliore notre chemin plus l'ellipse se rétrecit, plus l'espace de recherche de points est petit, plus l'on améliore vite notre chemin. Or, cette méthode produit l'effet inverse : plus l'on améliore notre chemin, plus l'on met de temps à tirer nos points. C'est pourquoi nous avons programmé notre outil de la manière suivante :

- Un boolean *pathFound* est initialisé à False,
- Si $pathFound == false$, on tire dans l'espace tout entier,
- Si $pathFound == true$, on tire selon les coordonnées de l'ellipse,

Notre RRT final trouve un chemin optimal en 157 ms en moyenne. Ces données ont été calculées dans la même configuration que lors du calcul des données du RRT* précédemment évoqué.

1.3 Représentation graphique et UI

Au fur et à mesure de nos avancées, et de nos cours, nous avons été ammenés à utiliser trois solutions graphiques. Deux d'entre elles ont déjà été developpées dans le rapport intermédiaire, elles le seront donc succintement ici.

1.3.1 Draw

La classe $Draw$ est une bibliothèque graphique pour Java, développée par deux professeurs de l'université de Princeton. Elle permet d'instancier en une ligne un objet graphique sur lequel on peut dessiner ce que l'on veut (en pratique, toutes les méthodes de la bibliothèque swing, sur laquelle draw s'appuie). Elle est donc très simple d'utilisation, et très générique, ce qui la rend lente. Elle nous à néamoins été utile pour afficher des situations simples, aider au débogage, et pour les rendus graphiques statiques.

1.3.2 LocalGraphics

$LocalGraphics$ est une classe développée par nos soins, spécifiquement pour l'affichage de drones, de checkpoints et de chemins. Elle réagit aux interactions souris/clavier pour l'ajout de drones ou de checkpoints. Elle est beaucoup plus pauvre en fonction que la classe $Draw$, mais nettement plus rapide. Pour la méthode par clustering qui sera dévelopée plus tard, nous pouvons afficher convenablement 10 images par seconde, contre moins de 2 avec $Draw$. Elle nous à été utile pour l'affichage dynamique en Java.

1.3.3 3D - INF443

Nous avons également mis à profit notre cours d'informatique graphique de P3, INF443 (en C++), en réutilisant les méthodes et templates vus en TD. En traduisant les programmes Java en C++, ce qui est un peu fastidieux, nous avons pu profiter d'un affiche 3D, fluide (60fps+), et interactif (possibilité de déplacer la caméra, visuel de drone et pas simple point/carré/sphère.

Au delà de l'intérêt esthétique, la clarté du rendu graphique est important pour l'amélioration de nos programmes. Certains bugs sont plus faciles à trouver avec un mouvement continu qu'avec 3 ou 4 fps.

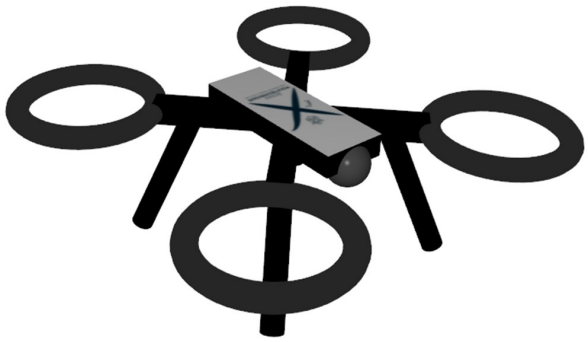


FIGURE 10 – Visuel de drone utilisé pour le rendu 3D

1.4 Tests

Pour quantifier la qualité de nos solutions, et plus particulièrement leur qualité relative, nouq avons mis en place un certain nombre de tests. Nous avons du trouver des critères mesurables d'évaluation des méthodes, effectuer les mesures puis analyser les résultats.

1.4.1 Positionnement du problème

Un des grands points forts de notre sujet d'étude est son vaste champ de solutions. Mais qui dit de nombreuses implémentations, dit également des performances très variées à comparer, des avantages et des inconvénients à mettre en lumière selon les situations.

Nous nous sommes alors demandés quels paramètres et grandeurs constituaient des indicateurs pertinents des performances du système. Voici par exemple, une liste non exhaustive de problématiques rencontrées durant cette phase de réflexion :

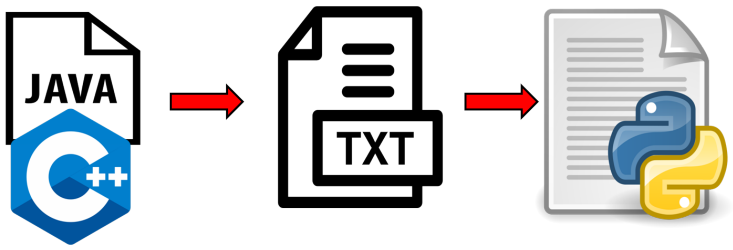


FIGURE 11 – Schéma récapitulatif de la chaîne d'acquisition des données

Paramètre mesuré	**Intérêt**	**Problématiques**
Satisfaction en un point	Permet ensuite de déterminer les points les moins visités = les les plus sensibles du dispositif. Les drones se répartissent-ils bien la zone ou y a-t-il régulièrement des zones d'ombre ? si oui est-ce toujours les mêmes ?	Facile à implémenter pour les méthodes à checkpoints, mais comment l'adapter aux méthodes continues ?
Satisfaction globale	Autant le calcul de la pire insatisfaction permet d'observer la présence de zones d'ombre, autant le calcul de la satisfaction globale (moyenne des satisfactions) permet de quantifier à quel point l'ensemble de la zone est en moyenne visitée.	Calcul d'une moyenne = nécessite la valeur pour les différents échantillons. Revient au même problème que précédemment pour les cas continus.
Distance moyenne aux voisins	Permet d'observer si les drones ont tendance à voler en essaim compact ou à s'éloigner les uns des autres pour se répartir le territoire.	Pas forcément très intéressant pour les méthodes de clustering car les drones restent généralement seuls dans leur territoire. Impossible à utiliser si les zones sont différentes : 2D e 3D sont incompatibles par exemple.

1.4.2 Recueil des données

Le recueil des données se fait en temps réel. Le traitement, lui, est différé. En effet, nous collectons les données sous forme d'objet Java pendant la simulation, puis nous le convertissons en fichier texte à la fin de celle-ci. Cela permet de stocker ces données, et de les lire/analyser avec d'autres outils que Java. Python notamment, via un notebook Jupyter nous à semblé être la meilleure solution.

Pour décorréler la partie acquisition/analyse/tracé, nous aurions pu utiliser des outils comme les fichiers JSON (JavaScript Object Notation), adaptés à ce genre de situation. Dans

un souci d'économie de temps et de moyens, sachant que les .txt fonctionnaient très bien, nous ne nous sommes pas lancés dans cette voie.

1.4.3 Perspectives d'amélioration

La récolte et l'analyse de données sur nos algorithmes pourrait être encore améliorée :

Automatisation de la génération des graphiques ? Actuellement le tracé des courbes nécessite toujours une intervention de l'utilisateur : faire tourner la simulation, téléverser le fichier .txt sur le Jupyter Notebook, puis l'afficher avec mathplotlib. Une automatisation de cette chaîne avec production d'un pdf en sortie fluidifierait l'exploitation des résultats.

Affichage en temps réel de certaines informations utiles ? Nous pensons notamment à la satisfaction globale de l'ensemble des checkpoints (ou de la map) qui serait un bon indicateur de la performance en temps réel. Un tel ajout toucherait cependant bien plus que la simple partie statistiques avec notamment des évolutions majeures des outils graphiques.

Etudier plus de performances ? Le principal défi des tests reste de trouver des dénominateurs communs aux méthodes pour pouvoir les comparer. De fait, nous aurions pu mesurer plus de performances pour certaines solutions. Mais l'intérêt s'avère limité si ces données ne sont pas collectables/comparables avec celles d'autres approches.

Le champ des ajouts possibles reste vaste : regarder si le drone adopte des trajectoires très brisées ou plutôt rectilignes, estimer la vitesse moyenne des drones si jamais on avait ajouté la prise en compte des effets d'accélération/décélération ...

2 Méthodes fonctionnelles

2.1 Méthode par conflit

Le fonctionnement de la méthode par conflit est intégralement détaillé dans le rapport intermédiaire, et n'a pas été modifié depuis. Nous ne présenterons donc ici qu'un résumé.

Le méthode par conflit se base sur une représentation de la zone par checkpoints. Sa principale caractéristique est qu'elle ne nécessite pas d'autorité centrale. Chaque drone prend ses décisions, et peut communiquer avec les autres "sur un pied d'égalité". Pour cela, il dispose de l'information donné par les checkpoints : leur instatisfaction.

L'insatisfaction d'un checkpoint est une fonction croissante du dernier instant ou un drone l'a visité. Il n'y a pas de contrainte à priori sur cette fonction (autre que la croissance), et tous les checkpoints n'ont pas forcément la même. Cela permet de décrire avec plus de finesse l'importance relative de ces checkpoints, la manière dont ils doivent être surveillés...

Pour prendre sa décision, le drone évalue le checkpoint qu'il a le plus intérêt à visiter. "L'intérêt" du drone à visiter le checkpoint est une fonction décroissante de la distance entre les deux, et croissante de l'insatisfaction du checkpoint. Il n'y a pas d'autres contrainte sur la fonction, et rien n'empèche les drones d'avoir des fonctions d'intérêt différentes, pour traduire des différences entre ceux-ci.

Une fois qu'il a choisi son checkpoint, le drone doit vérifier qu'il est le seul à l'avoir pour cible. Quand c'est le cas, il se dirige vers lui. Quand ce n'est pas le cas, il est en conflit avec un autre drone. C'est celui des deux qui est le plus proche du checkpoint qui l'emporte. Le perdant reprend une décision, parmi tous les checkpoints qui lui sont encore permis. Tant qu'il y a plus de checkpoints que de drones, chaque drone est garanti d'avoir une cible disponible.

Cette méthode à l'avantage d'être rapide. Chaque drone peut calculer son interêt pour tous les checkpoints en parrallèle, et les conflits se résolvent rapidement. Elle est également facile à implémenter, et offre un beaucoup de degrés de personnalisation à l'utilisateur. De plus, pour améliorer la coordination entre les drones, on peut envisager de faire prendre aux drones une décision plus anticipatrice : par exemple, choisir ses deux prochaines cibles au lieu de juste la prochaine. Il faut alors définir la notion de conflit entre deux listes de cibles, et plus seulement deux cibles.

Mais empiriquement, on observe un certain nombre de défauts à cette méthode. Par exemple, la coordination entre les drones n'est pas très bonne, nous l'illustrerons en 2.5.2.

En résumé, cette méthode est une méthode simple, robuste, permettant aux drones de prendre des décisions décentralisées, qui constituent un optima local parmi les décision possibles. Mais elle doit être rafiné pour garantir de meilleurs résultats, nottament en anticipant plus le mouvement des drones.

Dans la Figure 12, les cercles noirs sont des checkpoints, les disques verts sont des drones. Le parcours des drones ne s'affiche pas, donc l'image est moins parlante qu'une visualisation dynamique. Celle-ci sera proposée pour la soutenance orale.

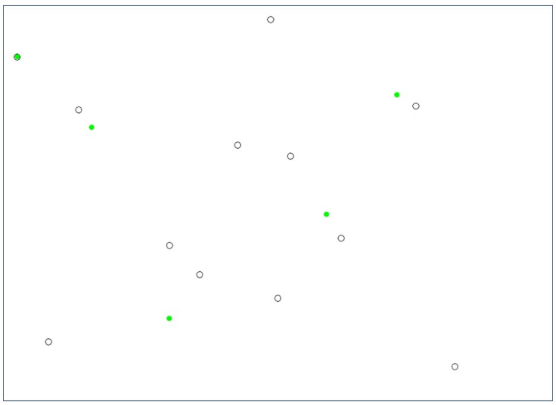


FIGURE 12 – Méthode par conflits

2.2 Méthode avec clustering

Après notre rapport intérmédiaire, les conseils de notre tuteur nous ont orienté vers des méthodes de clustering. Ce genre d'algorithme est en effet assez adapté à notre besoin : diviser l'ensemble des checkpoints en clusters, puis faire patrouiller chacun de nos drones dans son cluster. Séparer les zones d'action des drones permet d'utiliser leurs pleins potentiels.

Nous avons rapidement formulé deux exigences vis-à-vis de cette méthode :

- Elle doit être adaptable : ajouter ou retirer un drone ou un checkpoint du processus doit être peu coûteux en calcul.
- Elle doit être performante : le parcours d'un drone au sein de son cluster doit être optimisé. C'est exactement un problème TSP (traveler salesman problem).

Les exigences du clusteringet du TSP nous ont conduit à utiliser deux algorithmes : k-means et un recuit simulé (qui sera utilisé de deux manières). Le calcul se divise en trois phases : le calcul du clustering, le calcul des TSP, et l'attribution des clusters. L'ajout ou le retrait d'un drone ou d'un cluster nécessite de refaire une ou plusieurs de ces étapes. Voici le détail

2.2.1 Clustering

L'algorithme k-means (plus précisement, l'algorithme de Lloyd) est le plus connu des algorithmes de clustering. Nous en avons fait une implémentation détaillée en annexe. L'algorithme traite une liste de clusters et de checkpoints. La fonction principale est la fonction $improveOneStep$, qui associe chaque checkpoint au cluster le plus proche. La "position" d'un cluster est la moyenne des positions des checkpoints qu'il contient. En ajoutant ou en retirant un checkpoint à son cluster, on met à jour automatiquement cette moyenne.

Notre algorithme de k-means ne nécessite pas vraiment d'initialisation, puisqu'on peut être ammené à l'appliquer en cours de simulation. Il doit donc se baser par défaut sur la configuration précédente. La première configuration générée associe un checkpoint au cluster le plus proche. Si un cluster est vide, on lui associe automatiquement ce checkpoint. Cette méthode est similaire à une initialisation de Forgy, mais "à l'envers", puisqu'on n'associe pas un

centre de cluster à un checkpoint aléatoire, mais un checkpoint à un cluster (un peu mieux qu'aléatoirement).

L'avantage de cette formulation de l'algorithme de Lloyd est que la fonction $improveOneStep$ est une sorte de couteau suisse, que l'on peut utiliser à tout moment pour améliorer la situation globale. Elle remplit l'exigence d'adaptabilité, et donne des résultats pertinents. Néanmoins, on constate que ceux-ci se dégradent lors du passage de la 2D à la 3D.

Nous avons également développé un algorithme de clustering hiérarchique, détaillé en annexe. Il évalue la distance entre deux clusters comme la plus grande distance entre deux checkpoints de ces clusters. Il commence en attribuant à chaque checkpoint un cluster, puis il fusionne les clusters les plus proches jusqu'à ce qu'il y en ait autant que de drones. Il utilise pour cela une structure de union-find. Il donne de très bons résultats, souvent meilleurs que l'algorithme de k-means, particulièrement en 3D. Mais il n'est pas très adaptable, puisqu'il nécessite un calcul complet pour être utilisé.

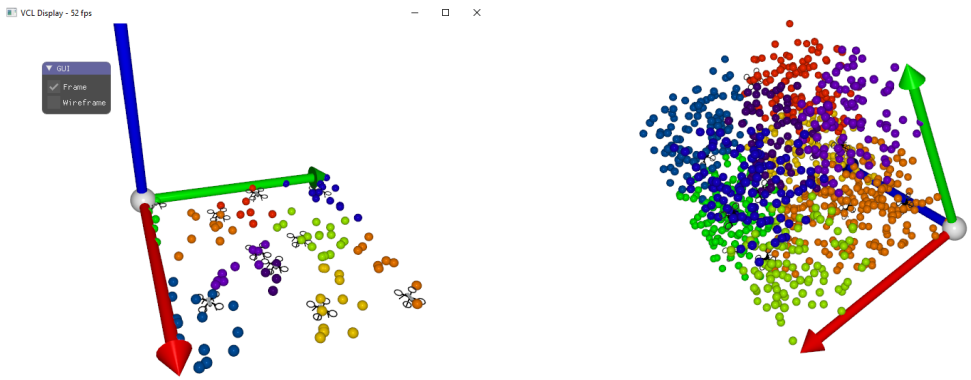


FIGURE 13 – Application du clustering sur des cas 2D et 3D.

2.2.2 TSP

Une fois que les checkpoints ont été attribués aux clusters, il faut trouver un chemin optimal dans ce cluster. Pour cela, on utilise un algorithme de recuit simulé dont l'implémentation figure en annexe.

Cette implémentation se base sur une interface, qui propose quatre fonctions :

- $getSize()$, renvoie la taille du problème, pour estimer le nombre d'itérations nécessaires. Empiriquement, on observe qu'un bon ordre de grandeur est de faire 1000 fois plus d'itérations qu'il n'y a de checkpoints. On peut aussi stoper l'algorithme dès lors qu'un certain nombre d'itérations se sont produits sans modifications. Empiriquement, on estime ce paramètre à 5 fois la taille du problème
- $modificationFunction()$, propose une modification de la configuration actuelle. En l'occurence, celle-ci consiste à parcourir un sous-chemin du chemin dans l'autre sens. On propose cette modification en tirant aléatoirement les deux extrémités de ce sous-chemin, ce qui coûte un $O(1)$.

— $improvementFunction()$, estime l'amélioration de la modification proposée. Étant donné la proposition de modification, cela ne coûte que quatre calculs de distance, soit un $O(1)$.
— $commitFunction()$, effectue la modification. La structure de chemin choisie est une double $HashMap$. Une qui associe les checkpoints à leur indice de passage et l'autre qui fait l'inverse. Contrairement à une liste comme $ArrayList$ ou $LinkedList$ on peut accéder à un checkpoint à partir de son indice et trouver l'indice d'un checkpoint en $O(1)$. Contrairement à un tableau, on peut ajouter et retirer un checkpoint en $O(1)$. Enfin, changer l'ordre de parcours d'un sous-chemin prendun temps proportionnel à la longueur de ce chemin.

L'implémentation permet de personnaliser les paramètres de valeur et de décroissance de la température. Empiriquement, on trouve qu'une décroissance linéaire est meilleure, même si mathématiquement, c'est la décroissance logarithmique qui est optimale.

2.2.3 Attribution

La dernière étape de calcul est l'attribution des drones à leur cluster. Il y a autant de drones que de clusters, donc cela revient à un problème de matching. Des algorithmes existent pour en calculer, comme la procédure de Gale-Shapley, mais nous avons choisi de réutiliser notre algorithme de recuit simulé. L'implémentation par interface rend cela très facile. On cherche à minimiser la distance totale entre un drone et son cluster. Les modifications considérées sont un échange de cluster entre deux drones. Les fonctions $modificationFunction()$, $improvementFunction()$ et $commitFunction()$ sont toutes en $O(1)$.

2.2.4 Ajout ou retrait

Conformément à l'exigence d'adaptabilité, l'ajout ou le retrait d'un drone ou d'un checkpoint doit nécessiter le minimum de calcul. Voici un tableau résumant les opérations effectuée lors de chacune de ces opérations :

	Drone	**Checkpoint**
Ajouter	Un nouveau cluster est crée, les étapes de k-means, TSP et atribution sont reprises	Le checkpoint est ajouté au cluster le plus proche, puis on effectue un recuit simulé sur ce cluster
Retirer	Le cluster du drone est fusionné avec le cluster le plus proche, puis les étapes de k-means TSP et attribution sont reprises	Le checkpoint est retiré de son cluster, puis on effectue un recuit simulé sur ce cluster. Si le cluster est vide, on refait les étapes de k-means

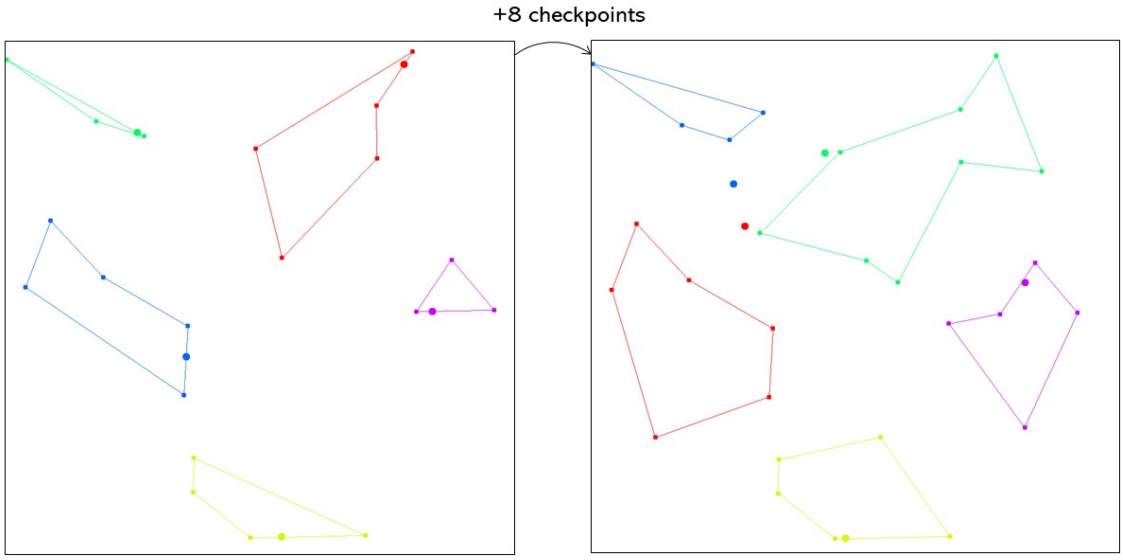


FIGURE 14 – Résultat du clustering initial, et après l'ajout de 8 checkpoints, avec *LocalGraphics*

2.3 Méthode par territoires

En cherchant des méthodes de résolution de notre problème, nous sous sommes concentrés à la fois sur l'utilisation de solutions existantes, connues, déjà testées et éprouvées, et à la fois sur des solutions innovantes, adaptées spécifiquement à notre situation

Nous avons alors pensé à nous inspirer du vivant, qui fort de quelques années d'évolution disposait sûrement d'une réponse adaptée à notre problème. Notre regard s'est alors arrêté sur la façon dont les animaux marquent leur territoire pour éloigner d'autres espèces. C'est ainsi qu'est née l'approche par territoires, où chaque agent s'octroie une zone propre et s'adapte en fonction de la présence ou non d'autres agents aux alentours.

Le principe de notre modèle est simple : un nombre d'individus appelés « walkers » (marcheurs) se déplacent dans un espace en laissant des traces de leur passage. Ces traces durent un certain temps noté T_{AS} pour « active scent time » (temps d'activité olfactive) au bout duquel les autres « walkers » peuvent à nouveau passer.

Cette approche innovante a plusieurs forces. Premièrement, elle permet d'avoir un équilibre naturel des agents dans l'espace sans laisser de zones libres. En effet, d'autres modélisations des territoires animaux voient l'apparition de « no man's land » qui seraient une faille dans notre système de défense.

Par ailleurs, de nombreux paramètres sont ajustables en fonction de l'espace à couvrir ainsi que de l'utilisation voulue.

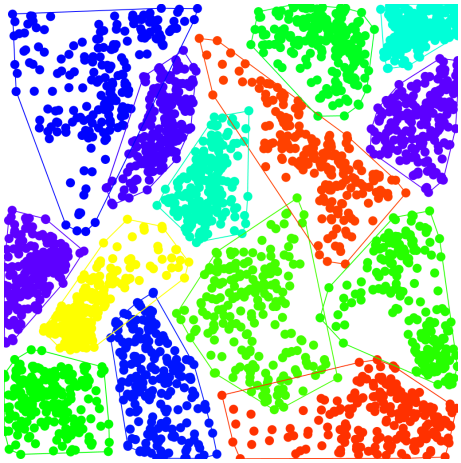


FIGURE 15 – Exemple d'un rendu de notre algorithme avec 14 drones se répartissant seuls sur une carte rectangulaire.

- *Durée T_{AS} des marquages* : Les drones peuvent être divisés en deux groupes, ceux qui restent sur une zone déterminée, par exemple autour d'un point important de la zone à couvrir, tandis que d'autres sont plus libres et patrouillent librement. Pour cela, il suffit de donner un T_{AS} petit aux "électrons libres" et un élevé aux autres.
- *Marche aléatoire des drones* : Les drones se déplacent selon une loi mathématique, avec ou sans mémoire (chaîne de MARKOV) que l'on représente en général par une marche aléatoire. Le choix de la marche caractérise directement le comportement des drones . Il est aussi possible de distinguer les marches aléatoires de chaque drone.
- *Calcul des territoires* : Une fois que l'algorithme a terminé ou régulièrement lorsque qu'il tourne, il affiche les territoires de chaque drone. Plusieurs méthodes sont possibles mais la plus simple et celle qui a été choisie pour les exemples et celle du calcul de l'enveloppe convexe des points de passage du drone. D'autres méthodes sont possibles comme celle du cercle minimal (cercle d'aire minimale contenant tous les points du territoire) qui permettent par la suite d'utiliser au mieux le système. En effet, si l'on reprend l'exemple précédent où l'on divise les drones en deux groupes, les "fixes" et les "électrons libres", utiliser la méthode du cercle minimal n'est pas optimale puisque les poins sont très étalés, dans un long rectangle très fin par exemple.

Cette modélisation permet une répartition des drones dans l'espace de manière aléatoire, ce qui est un point extrêmement intéressant dans les systèmes de défense puisque cela apporte de l'imprédictibilité dans les déplacements. Une imprédictibilité relative puisque l'utilisateur peut adapter les paramètres pour jouer avec et la construction de l'algorithme assure que les drones ne se rencontre jamais. Les drones ont donc une autonomie dans leur déplacement mais un certain ordre est instauré par le groupe.

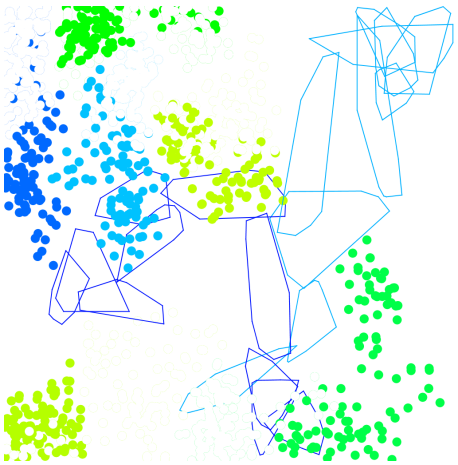


FIGURE 16 – Deux électrons libres où l'on n'affiche que les "convex hull" tous les 10 pas. Nous voyons qu'ils vont occuper la place laissée libre par les autres.

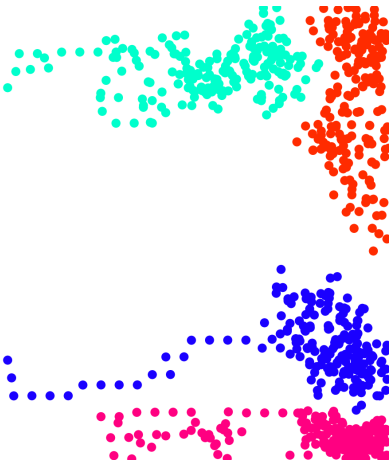


FIGURE 17 – Exemple d'ajustement des paramètres pour que les drones suivent une seule direction privilégiée. On distingue que deux suivent une trajectoire totalement horizontale à partir d'un certain moment alors qu'un autre balaye sa zone de manière verticale.

2.4 Méthode avec potentiel

La dernière méthode que nous avons développée s'inspire de la physique, et de la notion d'énergie potentielle. Nous avons souhaité traduire l'importance relative des parties de la zone par un potentiel, et laisser les drones se diriger vers les minimas de ce potentiel, comme le feraient des particules dans un champ gravitationnel. "L'intérêt" d'une zone peut être décrit sur une echelle continue.

L'esprit de cette méthode est de faire se déplacer les drones sur cette carte, en ne les faisant intéragir qu'avec le potentiel. Pour qu'ils se coordonnent, il faut donc qu'ils puissent se transmettre de l'information via le potentiel. Chaque drone observe le potentiel pour se déplacer, puis modifie celui-ci pour informer les autres de sa position et sa trajectoire. On distingue donc deux phases :

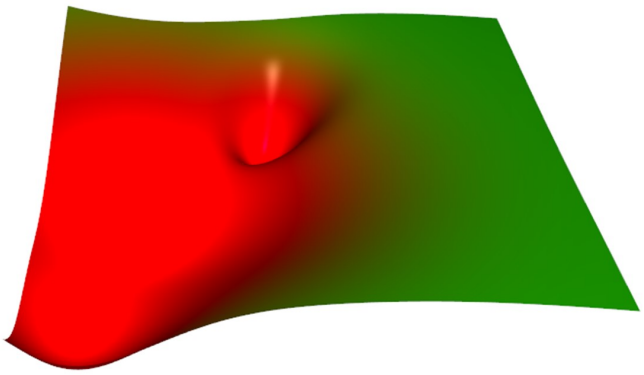


FIGURE 18 – Exemple de potentiel sur une carte $[0, 1] \times [0, 1]$

— Décision : pour chaque drone, le choix d'un déplacement à partir de sa position et du potentiel.
— Information : la mise à jour du potentiel à partir des nouvelles positions des drones.

Nous proposons plusieurs solutions possibles pour chacune des deux phases.

2.4.1 Décision

Trois implémentations ont été réalisées :

— En s'inspirant d'un algorithme de descente de gradient, pour "minimiser le potentiel", donc trouver la zone la plus intéressante, le drone prend la direction du gradient du potentiel en sa position. Le potentiel étant décrit par une grille, il suffit de trouver le triangle sur lequel il se trouve, et de calculer le "gradient de ce triangle" avec un peu de géométrie.
— Le drone va échantilloner l'intervalle $[0, 2\pi]$. Pour chaque angle, il trace un "rayon" de longueur fixée par l'utilisateur, sur le potentiel, puis il estime la valeur moyenne du potentiel le long de ce rayon. Si le rayon touche un obstacle ou un bord, il rebondit orthogonalement à l'obstacle. Il suffit ensuite de choisir la direction qui minimise la valeur moyenne du potentiel sur le rayon.
— Presque comme l'implémentation précédente, le drone ne lance plus un rayon mais une marche aléatoire depuis sa position. La marche aléatoire à un nombre d'étapes fixé par l'utilisateur. Le déplacement depuis une position vers une autre peut se faire selon une loi uniforme sur l'angle, ou en choisissant un angle "proche" (distribution gaussienne par exemple), de l'angle précédent. Cela favorise des trajectoires plus rectilignes.

Voici un tableau résumant les paramètres dont le contrôle est laissé à l'utilisateur :

Gradient	Vitesse du drone
Rayon	Vitesse du drone Nombre de directions de $[0, 2\pi]$ considéres Longueur du rayon Nombre de points de mesure du potentiel sur le rayon
Marche aléatoire	Vitesse du drone Nombre de directions de $[0, 2\pi]$ considéres Nombre de marches aléatoires lancées par direction Nombre d'étapes de la marche aléatoire Distribution probabiliste de la marche aléatoire (uniforme ou gaussien)

Voici un tableau résumant les avantages et les inconvénients de ces implémentations :

	Avantages	**Inconvénients**
Gradient	Temps de calcul	Manque d'anticipation. Risque de blocage sur le bord ou vers un obstacle. Incompatible avec certaines techniques de la phase d'information (voir la suite).
Rayon	Bonne anticipation	Ne peut pas faire un "petit écart" pour éviter un obstacle et rejoindre un potentiel très bas. Temps de calcul, particulièrement en 3D.
Marche aléatoire	Très bonne anticipation	Beaucoup de paramètres à calibrer, Temps de calcul, particulièrement en 3D.

2.4.2 Information

A chaque étape, le drone doit mettre à jour le potentiel pour informer le reste des drones de son parcours. Cette étape est cruciale, et permet de s'adapter à des situations spécifiques, en fonction du besoin de l'utilisateur.

Pour notre implémentation, nous avons décidé de conserver en mémoire deux potentiels : le potentiel initial et le potentiel courant. Seul le deuxième est considéré par les drones pour la phase de décision. Le premier est utilisé pour la phase d'information.

Lors de celle-ci, le drone remet à zéro le potentiel dans un certain rayon autour de lui. Ce rayon pourrait correspondre à la portée de ses capteurs. On pourrait aussi imaginer augmenter le potentiel en fonction de la distance au drone, pour rendre en compte d'une mesure moins fiable du drone au voisinage de cette portée. Cela permettrait également de modéliser l'environnement de la mesure : si une zone est complètement dégagée, le drone peut "voir" loin, et remettre à zéro une grande partie du potentiel. Au contraire, dans un environnement

plus complexe (espace confiné, présence de fumée...), on ne remet le potentiel à zéro que dans un proche voisinage du drone. Il faut également faire attention à ne pas remettre le potentiel à zéro à travers un obstacle si les capteurs du drones ne le permettent pas (obstacle opaque...).

Sur tout le reste de la carte, le potentiel diminue géométriquement vers la valeur du potentiel initial. Les zones les plus importantes voient donc leur potentiel diminuer plus vite que les autres, pour signifier qu'elles exigent d'être visitées plus souvent. Un grand paramètre fait revenir très vite le potentiel vers 0, et à tendance à maintenir les drones dans les zones le plus importantes. Cela peut conduire à négliger des zones moins importantes, mais nécessitant tout de même des visites : Figure 20. Au contraire, un paramètre faible à tendance à uniformiser le passage des drones sur la carte. Une zone visitée mettra longtemps à être visitée à nouveau, indépendamment de son importance : Figure 20.

Les paramètres possibles de la méthode :

- Type de remise à zéro du potentiel autour du drone
- Type de décroissance du potentiel sur la carte

Les paramètres de notre méthode :

- Rayon de remise à zéro du potentiel autour du drone.
- Coefficient de diminution (géométrique) du potentiel sur la carte.

Voici deux exemples d'utilisation

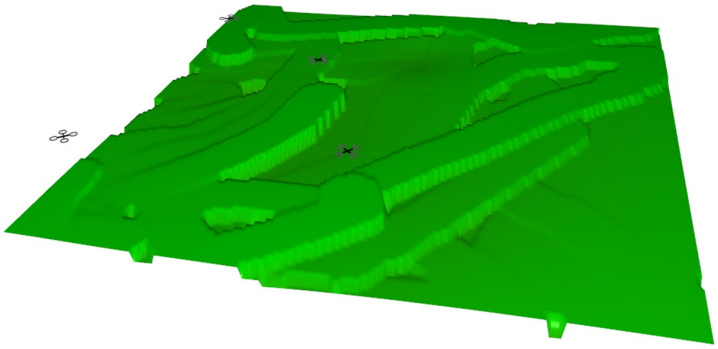


Figure 19 – Potentiel à long terme avec une petite décroissance

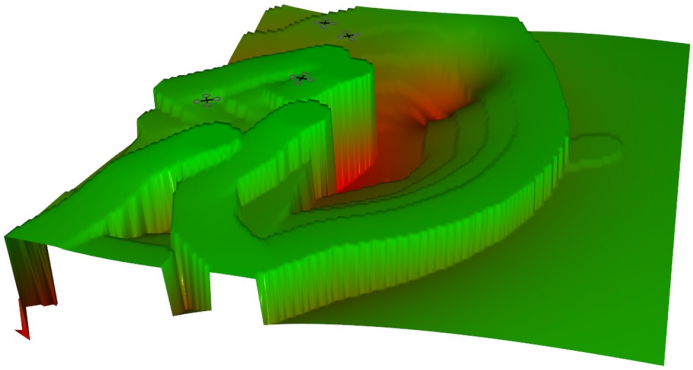


FIGURE 20 – Potentiel au cours du temps avec une grande décroissance

2.5 Analyse des résultats

2.5.1 Méthodes testées

La première méthode testée est la méthode par conflit. Le choix de cette méthode pour les tests repose sur plusieurs raisons :

- **Simplicité relative du code :** Méthode plus naïve que les autres donc plus simple à modifier ;
- **Des données facilement récupérables :** Structure de checkpoints, donc discrète.
- **Un code déjà terminé :** Nous implémentions les méthodes 3D dans le même temps. Cette méthode n'était donc plus vouée à changer drastiquement.

Une fois implémenté, nous avons choisi d'étendre les tests aux autres méthodes. La traduction des codes depuis Java a été complexe (mais instructive) : nous n'utilisions C++ que depuis quelques semaines en INF442/INF443. La fin du projet approchant, nous avons décidé de nous focaliser sur la solution la plus atypique (par potentiel) afin d'en évaluer l'efficacité sur cas concret.

2.5.2 Méthode par conflit

Le grand avantage de la méthode par conflit est sa rapidité d'exécution et de mise en place. Cependant, les tests répétés ont mis en lumière une faiblesse de cette méthode : les drones ont tendance à se regrouper pour certains environnements.

Pour illustrer ce phénomène, voici par exemple une situation à 4 zones à forte concentration de checkpoints surveillée par un réseau de 4 drones.

On intuite naturellement que la meilleure façon de surveiller ce territoire est d'attribuer une zone à chaque drone, comme pour les techniques par clustering. Or, on constate qu'après un certain temps, la configuration du réseau converge régulièrement vers une structure d'essaim visitant tour à tour les quatre zones. Ce biais dans la méthode implique alors une hausse

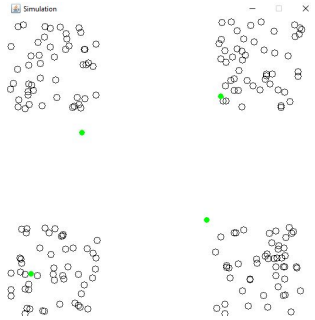


FIGURE 21 – Répartition initiale

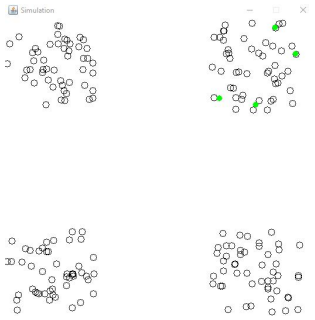


FIGURE 22 – Répartition pour temps longs

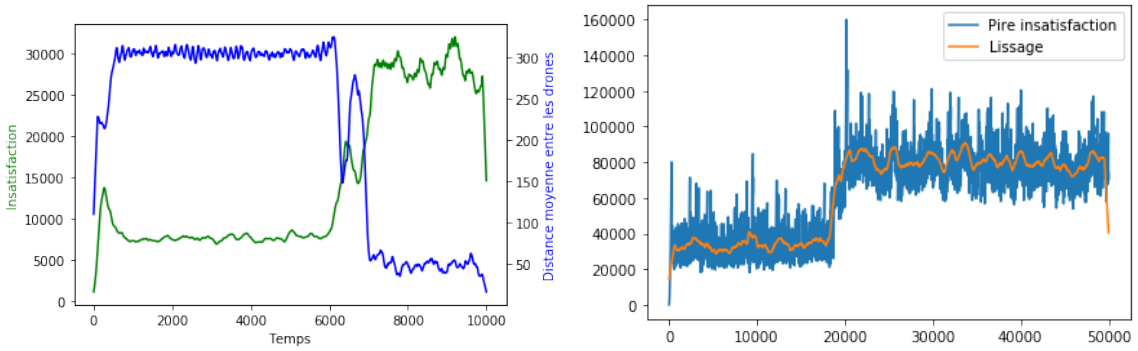


FIGURE 23 – Evolution temporelle de l'insatisfaction et de la distance moyenne entre drones (à gauche), et de la pire insatisfaction (à droite).

drastique de l'insatisfaction globale de la zone, car les forces de surveillance sont moins bien réparties. Ce problème pourrait être résolu par l'ajout de forces répulsives entre drones, ou en ajoutant plus d'anticipations pour les drones.

2.5.3 Méthode par potentiel

La couverture de la méthode par potentiel est globalement satisfaisante. Le territoire est arpenté dans sa totalité, sans zone d'ombre majeure à signaler. La représentation visuelle permet de le voir sans même rédiger de tests. C'est entre autres pour cela que nous ne l'avons pas développée en 3D, ou la représentation du potentiel est impossible. Toutefois, force est de constater que la répartition des forces est loin d'être optimale.

Similairement à la méthode par conflit, les drones tendent parfois à adopter des structures de front de drones. Ils surfent alors sur une vague de potentiel et balaient une zone jusqu'à rupture de la structure (généralement lors de la rencontre d'un bord ou d'un obstacle). Ce comportement anodin devient problématique pour les drones très proches qui suivent alors des trajectoires quasi identiques. On a ainsi une paire de drones qui fait le travail d'un seul. Cependant, ces comportements sont beaucoup plus rares que pour la méthode par conflit.

La vision à distance est également un point faible de cette approche. De fait, les drones "voient mal derrière les montagnes". Avant chaque déplacement, chaque drone tente plusieurs marches aléatoires et choisit celle qui minimise le potentiel sur son chemin. Or les chemins passant par des bosses de potentiel sont directement désavantagés, même si une vallée conséquente se cache derière. Les petits espaces non visités le restent donc parfois longtemps. Seul le passage très proche d'un drone permet de les actualiser. Les bords de carte et les contours d'obstacles sont typiquement des zones où un ennemi pourrait se cacher mais où les drones vont finalement moins souvent. A cette faiblesse, qui touche plus l technique par rayons que les marches aléatoires, nous avons considéré plusieurs parades :

- augmenter la longueur des marches aléatoires inspectées par le drone avant sa prise de décision = coût algorithmique ;
- faire décroître davantage le potentiel autour de ces zones = risque que ces lieux soient sur-inspectés.
- Biaiser la distribution aléatoire utilisée pour les marches aléatoires, en fonction de la présence d'un obstacle proche.

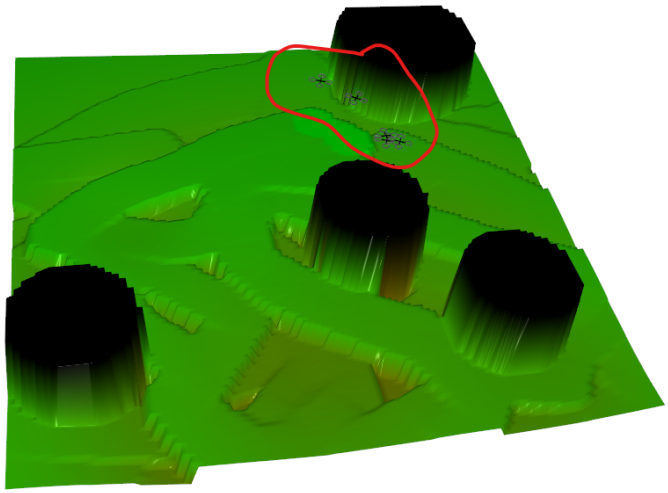


FIGURE 24 – Regroupement de drones faute de répulsion.

2.5.4 Limites des comparaisons

Dépendance des performances en fonction des cartes Les performances des solutions dépendent fortement de la géométrie des terrains. Ainsi, dans une situation spécifique, une approche peut être bien plus performante qu'une autre, alors que sur une autre carte, les rôles s'inversent. L'exemple des 4 zones vu plus haut illustre parfaitement ce genre de situation.

Si on souhaite estimer quelle approche est "en moyenne" la meilleure, une solution envisageable est de tester sur un set assez large de maps pour lisser les performances. Cela présuppose de savoir "traduire" ces cartes dans les formats spécifiques à chaque méthode.

La forte sensibilité à la géométrie des cartes mène à penser que tirer des généralités est peu pertinent. Il n'existe pas de méthode ultime, qui écrase les autres dans tous les cas. Il existe seulement des méthodes plus adaptées à des usages donnés.

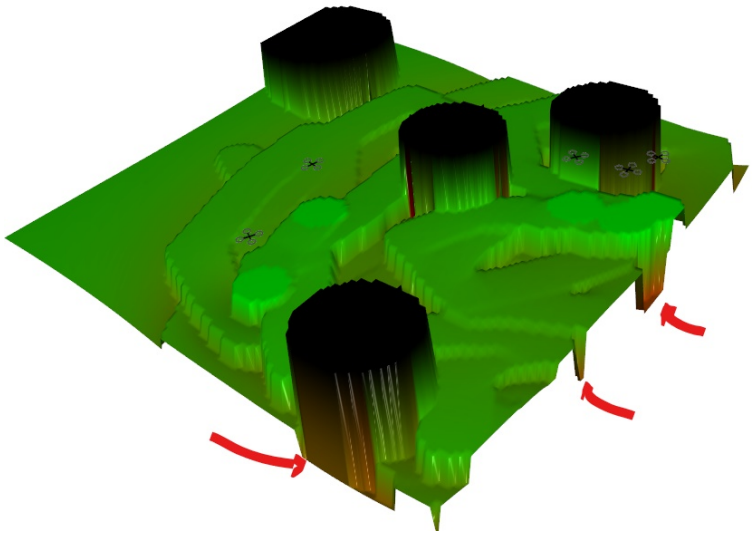


FIGURE 25 – Zones moins bien surveillées

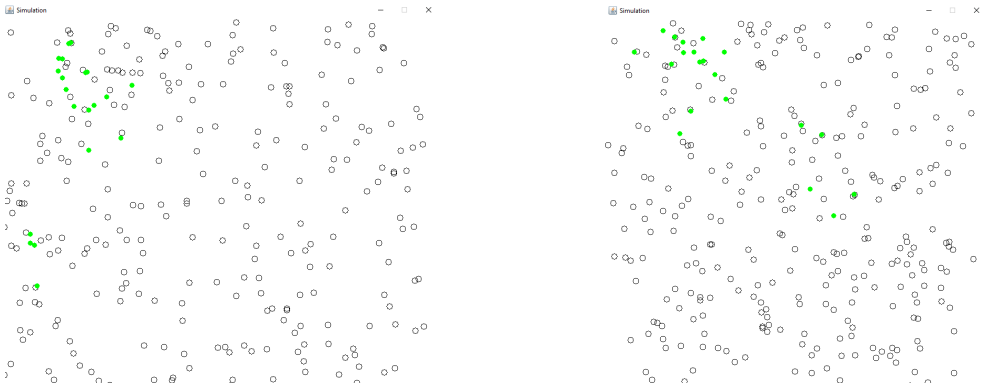


FIGURE 26 – Autres situations où les drones se regroupent.

Méthodes par potentiel VS méthodes par checkpoints L'intérêt de comparer ces deux types de méthodes est discutable. Ces dernières n'ont en effet pas exactement les mêmes objectifs. Dans une méthode par checkpoint, c'est le taux de rafraichissement de quelques points qui nous intéresse. Tout ce qui se passe hors du chemin est donc non considéré. A contrario, dans une méthode par potentiel, l'objectif est d'optimiser l'actualisation globale de la carte et donc de tous ses points.

On a donc d'une part une solution qui privilégie la surveillance de points clefs, tandis que l'autre privilégie la surveillance de l'ensemble de la zone. Les objectifs étant différents, les performances le sont également. Mais l'un n'est pas meilleur que l'autre pour autant. Tout dépend des critères définis par le cadre d'utilisation.

Problématiques d'échelle et de dimensionnement Lors de nos premières implémentations, les drones se déplaçaient dans des espaces discrétisés de $N \times N$ (avec N généralement compris entre 100 et 1000). Dans nos dernières méthodes 3D (potentiel ou checkpoint), les drones se déplacent dans un cube $[0, 1]^3$. Ces différences d'échelle sont à prendre en considération pour pouvoir comparer les distances.

Calculs d'insatisfaction ponctuelle et globale Quand la méthode utilise des checkpoints, l'évaluation de l'insatisfaction est assez aisée. Chaque checkpoint possède de fait un score variable dans le temps qui sert à la détermination des chemins des drones.

Lorsqu'on s'affranchit du système de checkpoints, le calcul devient beaucoup plus complexe. Effectivement, même si pour les potentiels, l'IHM permet d'intuiter visuellement la tendance globale, estimer l'insatisfaction du système nécessite de connaître la satisfaction en un point ou en une zone. Un parcours de l'intégralité du potentiel à chaque étape est nécessaire.

2.6 Quelle méthode pour quels cas ?

Les parties précédentes montrent combien répondre à cette question est complexe. Toutefois, après plusieurs mois de travail, certaines approches nous paraissent plus adaptées que d'autres pour un système de défense opérationnel.

La méthode par potentiel est sûrement celle qui répond le mieux au cahier des charges de la surveillance de zone. Globale, elle assure une couverture complète du terrain et non pas un nombre fini de positions. Modulable, elle tolère l'ajout de nouvelles fonctionnalités : gestion des caractéristiques du drone, gestion de la vision loin/proche, gestion de l'importance des zones. Imprévisible, elle est impossible à anticiper par l'ennemi, même par observation sur temps long.

Les méthodes par checkpoints également un intérêt certain :
- si on dispose d'un nombre de drones limité car mieux optimisées (dans notre cas) ;
- si l'objectif est d'assurer la surveillance de points clé du dispositif, même si elles restent faillibles par la prédictibilité de leurs rondes.

3 Méthodologie de travail et enseignements tirés

3.1 Organisation du travail en groupe

Notre projet est constitué quasi-exclusivement de programmes informatiques. Coder un programme donné est une tâche assez individuelle, puisqu'il est impossible d'écrire à plusieurs sur un même bloc de code en même temps. Cependant, avant l'implémentation de chaque approche, chaque algorithme, il y a une phase de réflexion collective.

Nous avons réalisé deux phases principales de brainstorming : une au début du PSC tout au long du mois de septembre, et une au moment de la rédaction du rapport intermédiaire fin janvier.

En septembre, nous avons commencé par étudier ensemble le rapport de PSC du groupe d'élèves de la promotion 2018 qui a inspiré notre projet. Nous en avons retiré deux leçons majeures :

— réaliser des tests sur des vrais drones est très compliqué et prend un temps démesuré par rapport à son intérêt : nous avons donc décidé dès septembre, sur les conseils de M. Haucourt, de ne réaliser que des simulations informatiques. Cette décision s'est avérée encore plus justifiée au vu des restrictions liées au Covid dans les mois qui ont suivi
— le problème de l'optimisation de la couverture de la zone par le réseau de drones est trop complexe pour qu'une analyse théorique puisse tirer des conclusions sur l'approche algorithmique à adopter : pour comparer les méthodes, le plus intéressant est de réaliser des tests et simulations

Fin janvier, avant de commencer la rédaction de notre rapport intermédiaire, nous nous sommes réunis pour faire le point sur ce que nous avions accompli jusqu'alors, et surtout pour brainstormer les nouvelles approches possibles et les nouvelles voies à explorer dans notre projet.

En dehors de ces deux moments, nous avons adopté l'organisation suivante pour coordonner notre travail : tous les mercredis, à quelques exceptions près, nous nous sommes retrouvés pour faire le point sur l'avancée du travail de chacun et résoudre ensemble les problèmes rencontrés. S'il est plus facile d'implémenter un programme seul qu'à plusieurs, pour débugger le code il est souvent très utile d'avoir un regard extérieur. Ainsi, si nos réunions ont parfois duré quelques dizaines de minutes seulement lorsque nos codes avançaient bien, il nous est aussi arrivés de faire des séances de débuggage mutuel et collectif de plusieurs heures.

Ces points de situation hebdomadaires nous ont permis de faire constamment émerger des nouvelles idées, puisque la créativité est stimulée avant tout par la discussion. Pour autant, nous avons du prendre garde à ne pas laisser tomber les projets en cours de route lorsque l'un d'entre nous proposait une nouvelle idée, qui de par sa nouveauté semble toujours plus attractive qu'elle ne l'est réellement.

Un autre aspect exigeant en termes d'organisation provient de l'emboîtement des différentes 'briques' informatiques de nos solutions. En effet, nous avons segmenté ce qui se passe entre l'input du problème et le rendu de la simulation associée en plusieurs étapes et programmes intermédiaires. Ainsi, le travail de chacun n'était jamais indépendant de celui des autres : un algorithme de recherche de chemins sur une grille ne sert à rien sans une interface graphique de type grille, et vice versa. Le diagramme de Gantt ci-dessous illustre ce phénomène. Nous avons donc du coordonner notre travail pour réaliser simultanément ces différentes 'briques', en devant parfois nous réorganiser en cours de route puisque telle partie était plus ardue à implémenter que ce que nous avions anticipé.

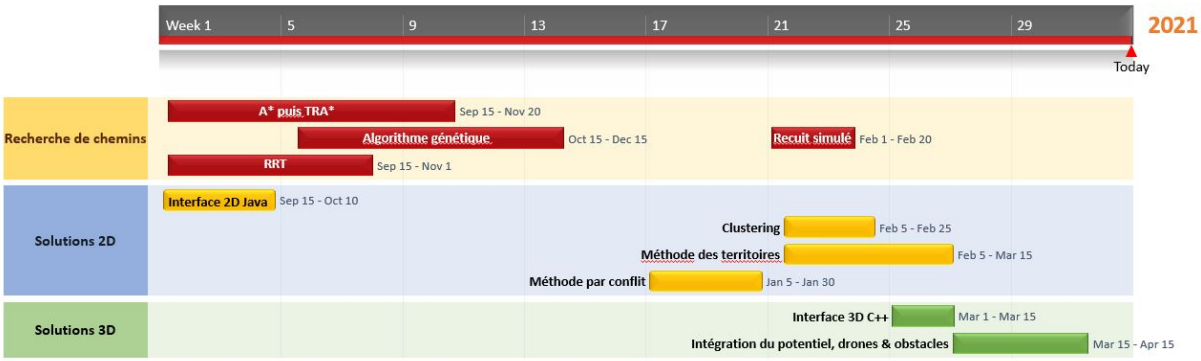


FIGURE 27 – Diagramme de Gantt des étapes de notre projet

3.2 Collaboration sur le code avec Git

Chacun d'entre nous avait déjà eu l'occasion de mener un projet informatique, mais nous n'avions pas l'expérience de développement informatique en groupe de plus de 2 sur un projet unique. Rapidement, des problèmes de coordination sont arrivés. Certaines versions des codes avaient des fonctionnalités que d'autres n'avaient pas, les versions n'étaient pas compatibles entre elles, les échanges par mail des nouveaux fichiers nous amenaient à confondre les versions et en conserver des obsolètes. De plus, sur un sujet ouvert ou les idées arrivent le plus souvent en cours de route, nous avons eu souvent l'occasion de modifier un code existant pour y incorporer une nouvelle fonctionnalité. Mais si celle-ci s'avère ne pas être bonne, ou ne pas fonctionner, le fait de revenir en arrière prend beaucoup de temps, et, souvent, ne ramène pas au point de départ. Nous avons donc perdu beaucoup de temps à refaire fonctionner des codes qui fonctionnaient auparavant, ce qui est assez frustrant et pas très efficace.

Au début de 2021, nous avons décidé de nous attaquer à ce problème en apprenant à utiliser le logiciel de gestion de version Git. Nous avons créé un répertoire GitHub pour notre PSC, ce qui nous a permis de gérer bien plus efficacement notre code et d'automatiser le partage des programmes. En prenant l'habitude de télécharger la version à jour en commençant à coder (git pull) et d'uploader nos modifications à la fin (git commit et git push), nous avons réussi à échapper aux soucis de versions incompatibles et obsolètes. Puisque GitHub sauvegarde tous les commits, nous avons également pu revenir en arrière sans problème lorsque nous avons fait des modifications hasardeuses dans nos programmes.

3.3 Un enseignement majeur : l'importance de la structure et la modularité du code

Depuis que nous faisons de l'informatique, on nous parle de l'importance de la structure du code et des avantages de découper le code en de nombreux modules indépendants. Cependant, n'ayant jamais travaillé sur un projet informatique de grande envergure, nous ne nous étions pas rendu compte concrètement de l'impact que cela a sur l'efficacité du travail en commun. Au fur et à mesure des de l'apparition des problèmes, nous avons appris à mieux anticiper nos futurs besoins et à standardiser l'articulation de nos programmes afin d'éviter de nombreuses heures de débuggage collectif.

Nous avons appliqué le principe de modularité du code dans la suite de notre travail : par exemple, lorsque nous avons codé un recuit simulé pour les parcours de chemins au sein d'un cluster, nous avons créé une *InterfaceRecuit* adaptée à tout problème d'optimisation, et nous l'avons appliquée aux problèmes précis qui nous intéressaient dans les autres classes.

3.4 Hard skills et soft skills travaillés

Notre PSC nous a d'abord permis d'acquérir un certain nombre de compétences 'dures' dans le domaine de l'informatique. En effet, nous avons beaucoup progressé en algorithmique, puisque nous avons étudié de nombreuses approches différentes du sujet de notre PSC et à chaque fois réfléchi à quels algorithmes nous pouvions appliquer et à comment les implémenter efficacement. Nous avons également travaillé l'informatique graphique, en 2D en Java intialement puis en 3D en C++. De plus, nous avons découvert le benchmarking de programmes, domaine que nous n'avions jamais exploré au cours de notre parcours académique : nous avons appris, par tâtonnements, à construire des tests standardisés pour comparer des algorithmes qui étaient parfois très différents. Enfin, nous avons appris à combiner l'utilisation de plusieurs langages et types de fichiers différents : lorsque nous avons comparé la performance de nos algorithmes, nous avons manié simulanément Java et C++ dans lesquels étaient codés les algorithmes, des fichiers .txt dans lesquels on exportait les données des tests, et un notebook Python pour traiter les données recueillies.

Nous avons également travaillé des compétences qui relèvent plus des 'soft skills' : l'élaboration et le suivi d'un cahier des charges, la planification d'un projet et son adaptation en fonction des imprévus, la canalisation des idées qui fusent en un plan cohérent n'en sont que les exemples les plus notables. Nous savions déjà tous travailler en groupe, mais pas sur un projet aussi long et avec autant d'étapes interdépendantes.

Conclusion

De nos travaux ressort la diversité des solutions pour répondre à un problème bien plus large qu'il ne nous était paru en septembre dernier. Clustering, algorithmes génétiques, recuit simulé, checkpoints, méthodes par potentiel ou conflit, découpage par territoires... Le champ des possibles est large pour orchestrer efficacement un réseau de drones.

Plus qu'une connaissance fine de l'état de l'art, ce projet scientifique commun a accru nos compétences techniques et nous a sensibilisés aux difficultés inhérentes à la programmation en groupe. Travailler pendant plusieurs mois sur un code commun nous a révélé tout l'intérêt des outils de collaboration et l'importance d'une gestion robuste des versions.

Ce PSC ouvre de nombreuses perspectives d'approfondissement. Lorsqu'à mi-parcours plusieurs de nos solutions étaient déjà fonctionnelles, nous avons pris le parti d'en approfondir l'optimisation afin de présenter des solutions plus abouties, plus efficientes et visuellement plus développées. Mais une multitude pistes restent à explorer. La détection d'un ennemi, sa poursuite, sa capture … sont tout autant de sujets passionnants qui ne demandent qu'à être étudiés et pour lesquels les angles d'approche semblent tout aussi nombreux.

Références

[1] Watson, B. Against the drones : how to stop weaponized consumer drones. Defense One, mars 2020. Disponible sur : https ://www.defenseone.com/feature/against-the-drones/

[2] Nedjah, N. & De Macedo Mourelle,L. Swarm Intelligent Systems. Studies in Computational Intelligence, Volume 26, 2006.

[3] Alighanbari, M. Task Assignment Algorithms for Teams of UAVs in Dynamic Environments. Thèse de Master, MIT, 2004.

[4] Bethke, B. Persistent Vision-Based Search and Track Using Multiple UAVs. Thèse de Master, MIT, 2005.

[5] M. R. Brust, G. Danoy, P. Bouvry, D. Gashi, H. Pathak and M. P. Gonçalves. Defending Against Intrusion of Malicious UAVs with Networked UAV Defense Swarms. 2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops).

[6] Naderi, Kourosh & Rajamäki, Joose & Hämäläinen, Perttu. (2015). RT-RRT* : a real-time path planning algorithm based on RRT*. 113-118. 10.1145/2822013.2822036.

[7] An Optimizing and Differentially Private Clustering Algorithm for Mixed Data in SDN-Based Smart Grid Chen Y.Patel V. M.[...]Kose U. IEEE Access (2018)