Université d'Angers - Institut des sciences et Techniques de l'Ingénieur

Mémoire

Master 2 Recherche : Systèmes Dynamiques et Signaux

Le Bouter Sébastien

20/06/2008

Table des matières

PARTIE 3: COUPLAGE GENERATION DE TRAJECTOIRE/ SUIVI DE TRAJECTOIRE

CHAPITRE 1
Introduction

I.1 - Contexte du sujet :

L'apparition des premiers drones volant sans pilote humain à bord date de la fin de la seconde guerre mondiale. Tout d'abord utilisés comme cibles militaires pour l'entraînement au combat, ils furent ensuite employés pour des missions de reconnaissance dans les années 1960 (drones Firebee et Lightning Bug). Leur utilisation pour des missions de surveillance s'est ensuite répandue lors de nombreux conflits. Depuis, de nombreux drones ont été développés, et leur usage à des fins civiles a également débuté. Les drones occupent ainsi une place de plus en plus importante dans les milieux aéronautiques civils et de la défense.

Toutefois, si le potentiel d'applications semble effectivement très élevé, ce n'est pas sans soulever certaines difficultés fondamentales qui, à défaut d'être résolues, pénaliseraient gravement une utilisation optimale et courante des drones. On peut notamment penser à l'autonomie (énergétique ou décisionnelle, la sûreté de fonctionnement, l'insertion dans le trafic aérien, etc.).

C'est dans ce cadre que l'Office National d'Etudes et Recherches Aérospatiales (ONERA) et plus particulièrement son Département PRospective et Synthèse (DPRS) se propose d'apporter des solutions. L'ONERA mène de nombreux projets en partenariat avec des industriels afin d'apporter des résultats qui devraient permettre, à terme, de donner aux drones miniatures une autonomie satisfaisante. On peut notamment citer le projet RESSAC (Recherche Et Sauvetage par Système Autonome Coopérant) dont l'objectif est de produire une démonstration de capacités d'autonomie de contrôle du vol, de conduite de mission, d'acquisition et de traitement d'informations pour la décision. Un autre exemple est le projet européen MAVDEM (Mini Aerial Vehicle DEMonstrator) dont l'objectif est l'étude, la conception et le test d'un drone aérien

miniature hautement performant (vol stationnaire, vitesse de croisière élevée et économique), fiable, facile d'utilisation et peu cher.

En parallèle plusieurs études ont été menées à l'ONERA sur les drones miniatures à voilure tournante. En effet, l'utilisation de ce type de drones est limitée à cause de leur grande sensibilité aux rafales, et de leur besoin de réactivité lors de la prise de décisions par rapport à leur environnement physique. L'ensemble de ces phénomènes impacte grandement sur leur autonomie. Ainsi, dans le cadre d'une thèse ([2], Sylvain BERTRAND 2004-2007), de nouveaux algorithmes ont été développés pour le guidage pilotage des drones miniatures en environnement perturbé (stabilisation et suivi de trajectoire en présence de perturbations aérologiques et en l'absence de mesures de vitesses du véhicule, nouvelles techniques de commande prédictive et adaptative). De plus, des travaux menés en ce moment à l'ONERA (thèse de Walid ACHOUR) reprennent la problématique du vol en milieu perturbé (estimation du vent, minimisation de son influence sur la trajectoire, recherche de chemin optimal).

Mon Projet de Fin d'Etudes se positionne en jonction de ces deux thèses.

I.2 - Finalité du PFE :

Le système considéré lors de ce projet est un drone miniature à voilure tournante. Pour rallier l'objectif on établit en général une trajectoire de référence déterminée avant la mission. Ces drones sont amenés à réaliser des missions dans des environnements incertains, de manière automatique, où la position des objets n'est pas connue à priori.. Dans le cas d'un environnement incertain, cette approche est donc inadéquate. La trajectoire à suivre doit être générée ou modifiée en ligne à partir des informations délivrées par les

Figure 1 : exemple de drone miniature à voilure tournante (quadrirotor)

capteurs embarqués. Les algorithmes de génération de trajectoire et de commande doivent donc être capables de prendre en compte, en ligne, ces nouvelles informations.

L'objectif de ce projet sera donc d'étudier des méthodes de génération de trajectoire et de commande et de créer un code de simulation permettant de mettre en œuvre le couplage entre ces deux parties

PARTIE 1 :

GENERATION DE TRAJECTOIRE

CHAPITRE 2 :
Le problème et sa décomposition

Dans un premier temps, on va considérer l'environnement connu à priori et comme statique, c'est-à-dire que les obstacles sont invariants dans le temps. De plus, on néglige les contraintes de temps imposées par la faible puissance de calcul embarqué dans le drone. L'idée sous-jacente est de se faire une bonne idée globale du fonctionnement du système. On a ainsi un découpage du problème global en plusieurs sous-problèmes plus facilement analysables.

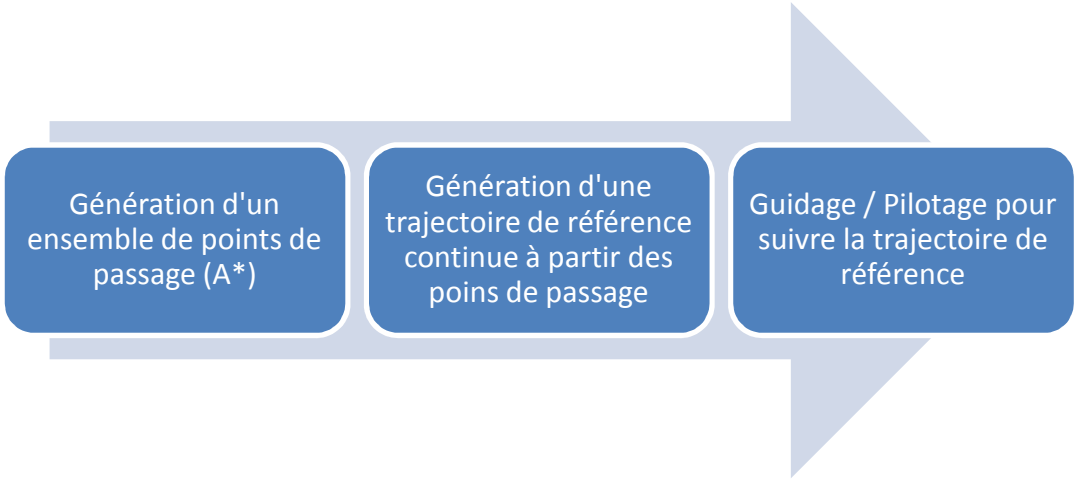


Figure 2 : Synoptique du fonctionnement

Dans un deuxième temps, on va faire d'autres hypothèses qui conduiront à l'amélioration des différents blocs :

- Les obstacles ne sont plus exactement ceux connus à priori du drone
- Le drone doit trouver un chemin dans un temps imparti

On essaiera aussi différents types de commande (retour d'état statique, commande prédictive) afin d'étudier le comportement du drone dans différents cas et d'analyser les performances en terme de suivi de trajectoire.

CHAPITRE 3 :
Recherche de chemin dans un graphe

III.1 - Modélisation de l'environnement :

Le drone miniature considéré est amené à évoluer dans l'espace 3D. Dans un premier temps, on va le discrétiser de manière régulière pour le modéliser. Le graphe obtenu sera ainsi facilement implémentable et utilisable par MATLAB. D'autres méthodes existent, plus complexes mais aussi plus efficaces, basées sur la discrétisation non-uniforme de l'environnement [3]: décomposition en cellules trapézoïdales pour les environnements dont les obstacles sont polygonaux, maillage de Voronoi ou autre.

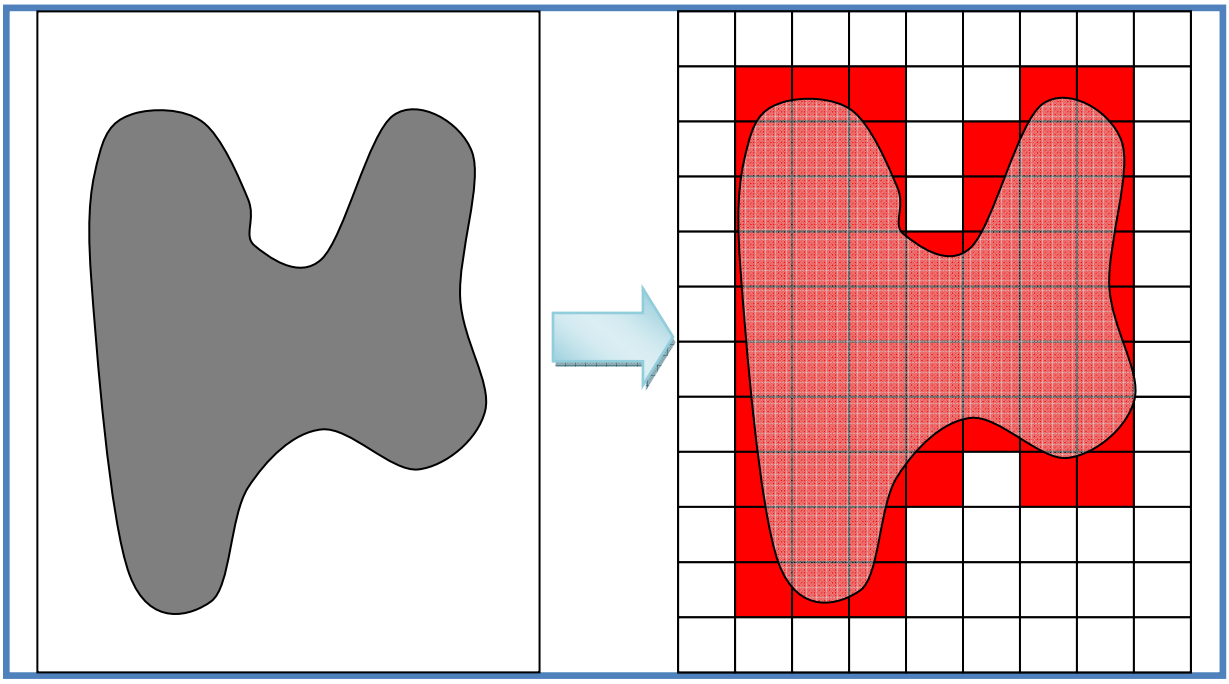


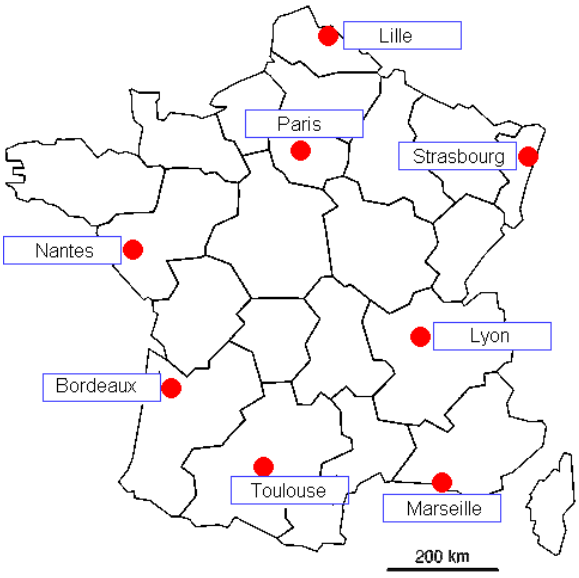
Figure 3 : Discrétisation de l'espace et d'un obstacle (généralisable à un obstacle en trois dimensions)

Le résultat est chargé dans une matrice dont les coefficients représenteront « l'état » de la case :

Valeur	Etat
-1	Obstacle
0	Départ (start)
1	Arrivée (goal)
2	Case franchissable

L'objectif de la première partie du problème consiste à trouver un chemin entre le départ (0) et l'arrivée (1) en évitant les obstacles (-1).

III.2 - Introduction à la recherche de chemin :



Supposons qu'on se donne une carte avec un point de départ et un point d'arrivée, tous deux connus. On peut modéliser cette carte par un graphe, dont les arcs représentent les chemins, le poids de ces arcs la longueur des chemins et les nœuds les intersections des chemins.

Figure 4 : Carte de France avec certaines grandes villes

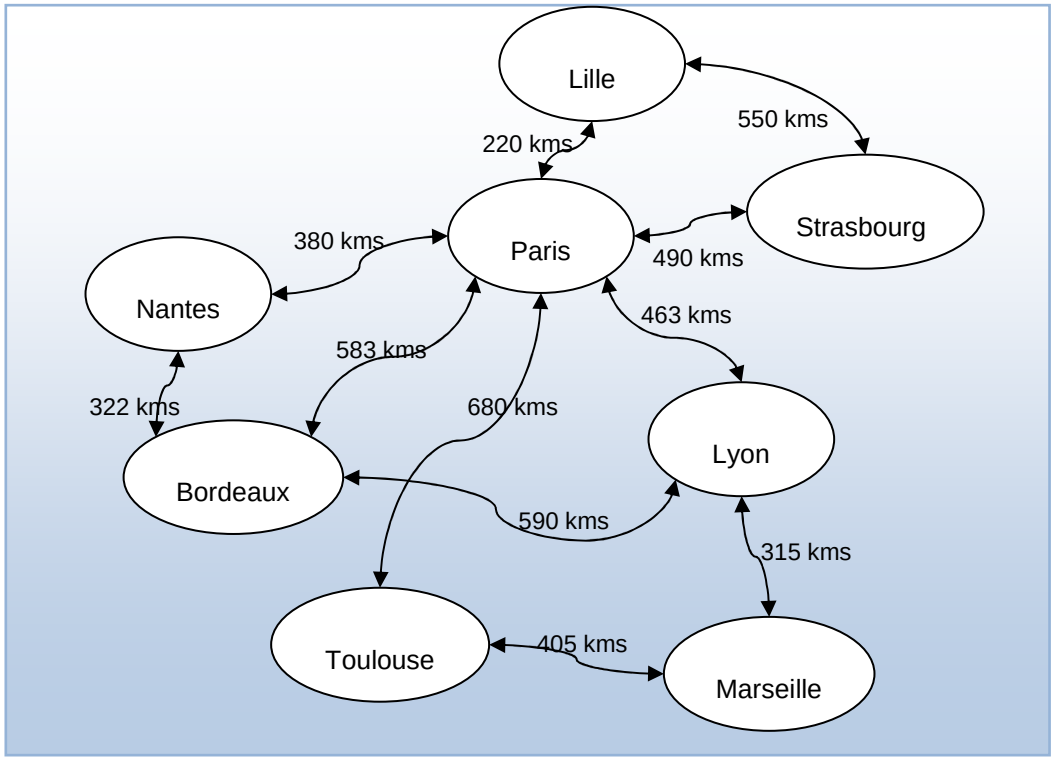


Figure 5 : Modélisation possible de la carte de France

Il existe de nombreux algorithmes qui vont nous permettre de trouver, lorsqu'il existe, le chemin reliant deux points dans ce graphe. Un des plus connu est l'algorithme de Dijkstra [4], qui va servir de point de départ à notre étude.

III.3 - L'algorithme de Dijkstra

i. Fonctionnement :

L'algorithme de Dijkstra est un algorithme itératif permettant de générer l'arbre couvrant minimum d'un graphe à partir d'un de ses sommets. Pour l'implémenter, on utilise deux listes : OPEN, dans laquelle on va mettre les nœuds qui sont candidats pour notre chemin, ainsi que CLOSED dans laquelle se trouveront les nœuds qu'on a déjà sélectionnés lors d'une précédente itération (i.e. qui étaient à une itération i, les meilleurs candidats proposés par OPEN).

A chaque nœud (appelé aussi état ou sommet, noté « s ») est associée une fonction « g » représentant le coût du chemin. On note s' un successeur de *s.* Ce coût est la somme des coûts des transitions « c(s,s') » entre les états permettant d'arriver au nœud s :

$$g(s) = \sum_{i=1}^{n} c(s_{i-1}, s_i)$$

A chaque itération de l'algorithme, on va regarder dans la liste OPEN le nœud possédant la fonction de coût la plus faible. On passe alors ce nœud s dans CLOSED et pour chacun des successeurs « s' » de « s », si le coût g(s) + c(s,s') est inférieur à g(s') alors g(s') = g(s) + c(s,s') et on insère ce nœud dans OPEN. Cette étape permet de garantir l'optimalité (au sens de la longueur) du chemin qu'on trouvera [4].

On réitère le processus jusqu'à ce qu'OPEN soit vide.

0: *Initialisation* pour tout s ≠ s_{start} g(s) = +∞
1: g (s_{start}) = 0 ;
2: OPEN = {s_{start}}
3: TANT QUE (OPEN non vide)
4: enlever le nœud s de OPEN avec le coût g minimal
5: ajouter s à CLOSED_list
6: POUR chaque successeur s' de s
7: SI (g(s') > g(s) + c(s,s'))
8: g(s') = g(s) + c(s,s')
9: Insérer s' à OPEN

Pseudo code 1 : calcul des valeurs de g par itérations successives de l'algorithme de Dijkstra

ii. Résultats

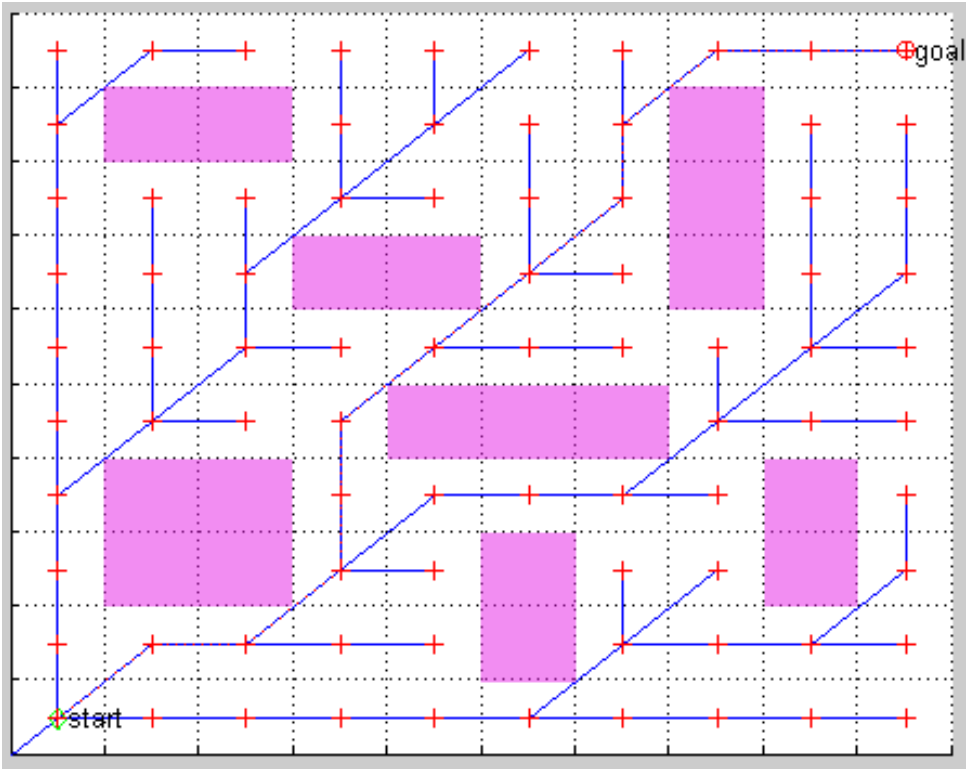


Figure 6 : Exemple de résultat avec l'algorithme de Dijkstra.

Le résultat de l'application de l'algorithme de Dijkstra sur un graphe est présenté en figure 6. La fonction de coût pour cette figure ainsi que pour le reste du document est la distance euclidienne. Les croix rouges représentent les cellules qui ont été visitées lors du déroulement de l'algorithme. On représente en bleu les chemins de longueur minimum permettant de se rendre aux différents nœuds du graphe. Ce résultat est obtenu après 82 itérations.

On se place maintenant dans un environnement dénué d'obstacles (de taille 200x100). On place le point de départ au centre de la carte et l'arrivée à l'extrémité Est de celle-ci. On lance l'algorithme qu'on arrête au bout de 11000 itérations. Le résultat est présenté en figure 8. On constate que la propagation de la recherche se fait de manière concentrique autour du point de départ et que le chemin jusqu'à s_{goal} n'est pas encore trouvé après 11000 itérations alors que la longueur du chemin optimal est de 100.

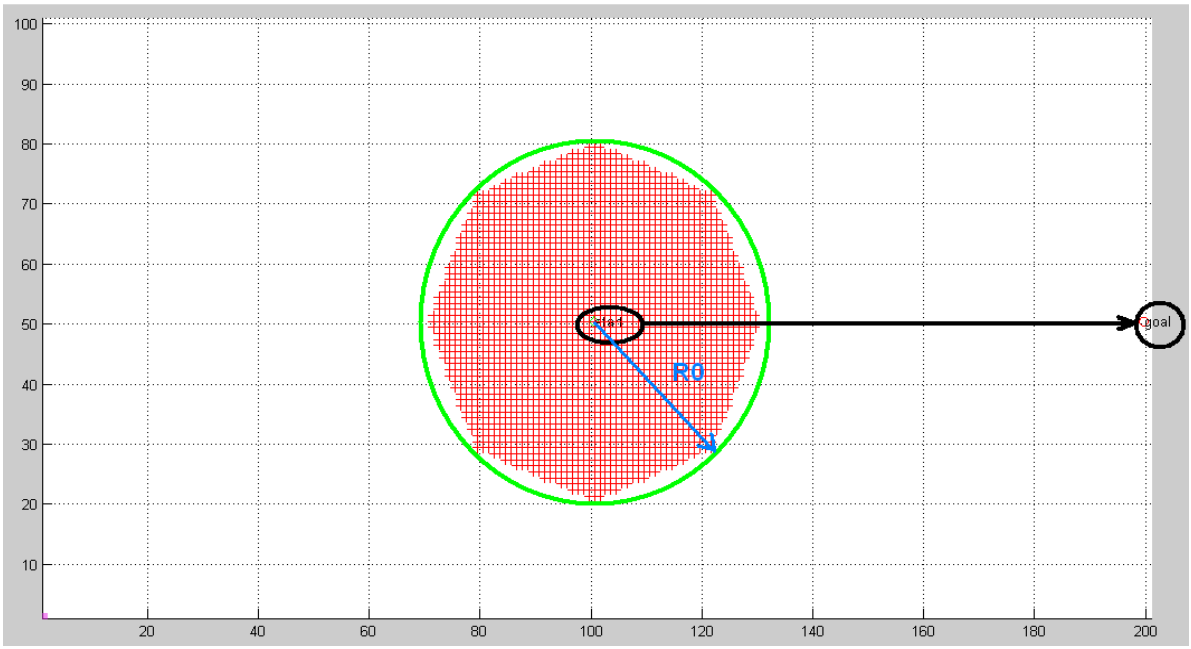


Figure 7 : Résultat après 2500 itérations sur une carte sans obstacles

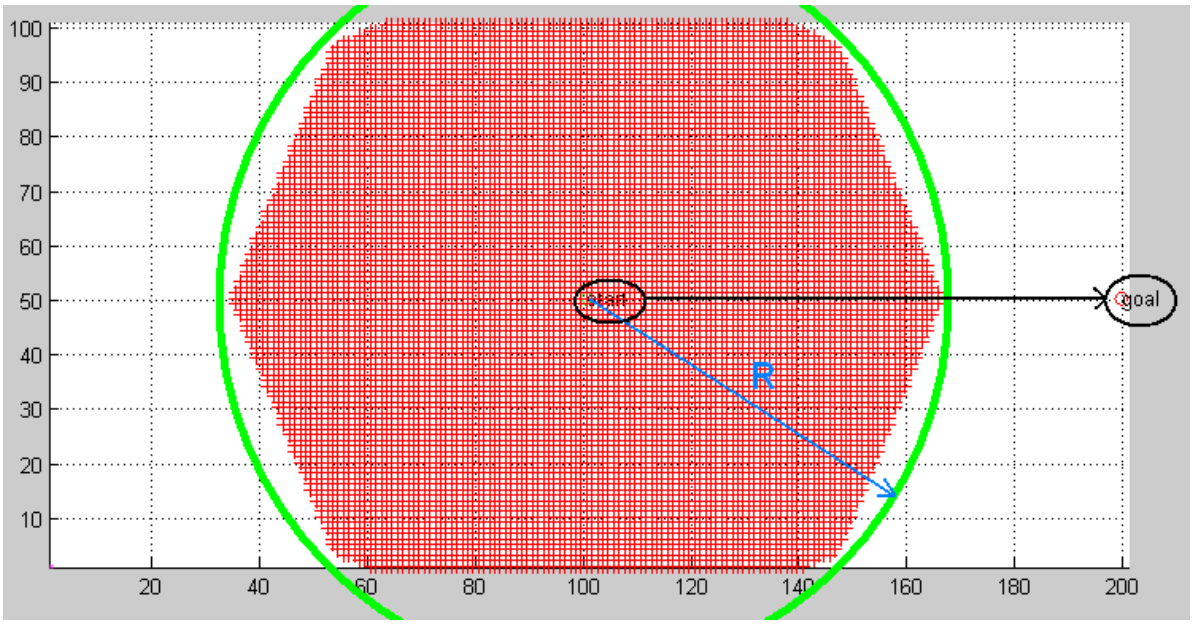


Figure 8 : Résultat après 11000 itérations sur une carte sans obstacles

Ainsi, même si l'algorithme de Dijkstra est optimal au sens de la taille du chemin pour relier deux points, il ne l'est sûrement pas au sens du temps mis pour trouver la solution.

Dans un drone miniature où la puissance de calcul embarquée est limitée, il est nécessaire d'obtenir une solution dans un temps plus court. Ce besoin a conduit à l'étude de nombreux algorithmes, dont le A*, qui fait l'objet des prochains paragraphes.

III.4 - Algorithme A*

i. Principe

La donnée supplémentaire dont nous disposons dans le cas de la planification de mission de notre mini-drone est la connaissance du point d'arrivée s_{goal} de la mission. Il serait donc judicieux de proposer une méthode qui permet de chercher dans la direction du but plutôt que d'effectuer une recherche omnidirectionnelle comme le fait l'algorithme de Dijkstra.

L'idée est d'introduire dans l'algorithme de Dijkstra, en plus de la fonction de coût g, une fonction permettant de « favoriser » les nœuds plus proches de s_{goal} [5].

On va appeler cette fonction heuristique, notée h, telle que

$$h : S \rightarrow \mathbb{R}+,$$
$$s \rightarrow a > 0, \ \forall s \neq s_{goal}$$

qui vérifie

$$[\ h(s1) < h(s2)\ ] \Leftrightarrow [\ s1\ strictement\ plus\ proche\ de\ s_{goal}\ que\ s2\ ]$$
$$h(s_{goal}) = 0$$

Typiquement, on prendra pour heuristique la distance euclidienne.

On introduit donc une nouvelle fonction key(s) qui servira de critère pour le choix du successeur dans la liste OPEN.

$$key(s) = g(s) + h(s)$$

De plus, il n'est plus nécessaire de dérouler l'algorithme jusqu'à ce que la liste OPEN soit vide. On va arrêter la recherche dès qu'on aura atteint s_{goal}.

ii. Pseudo code

On modifie en conséquence le pseudo code précédent afin de prendre en compte le nouveau critère de coût :

0: Initialisation; pour tout s ≠ $s_{start,}$ g(s) = +∞
1: g (s_{start}) = 0 ;
2: OPEN = {s_{start}}
3: ***TANT QUE*** ***(s_{goal} pas visité par A*)***
4: ***enlever le nœud de OPEN avec le plus petit key(s)***
5: ajouter s à CLOSED_list
6: POUR chaque successeur s' de s
7: SI (g(s') > g(s) + c(s,s'))
8: g(s') = g(s) + c(s,s')
9: ***Insérer s' dans OPEN ou mettre celui déjà présent à jour avec le nouveau key(s') = g(s') + h(s')***

*Pseudo code 2 : calcul des valeurs de g par itérations successives de l'algorithme A**

On a ici une formulation classique de la recherche de chemin par l'algorithme A*. On précise que les nœuds de OPEN sont choisis en fonction de leur key(s). Si g(s) est le coût du meilleur chemin trouvé entre s_{start} et le nœud s, et h(s) une estimation du coût du meilleur chemin entre le nœud s et s_{goal}, alors key(s) est une estimation du coût du meilleur chemin entre s_{start} et s_{goal} passant pas le nœud s. Si la fonction h est admissible, c'est-à-dire qu'elle ne surestime pas le coût minimum du chemin entre s et s_{goal}, alors l'algorithme A* garantie l'obtention d'un chemin optimal [1]. De plus, si la fonction h est consistante, c'est-à-dire que $\forall\, s, s' \in S, \text{tels que } s' \in \text{succ}(s), h(s) \leq c(s, s') + h(s')$, alors aucun nœud n'est regardé plus d'une fois. La mise à jour de la ligne 9 permet d'assurer une décroissance de g du successeur s' du nœud s.

Lorsque la recherche est finie, on reconstruit la solution : initialisation du chemin à s_{goal} puis, pour le dernier nœud s_i du chemin reconstruit, $s_{i-1} = \arg min_{s' \in pred(s_i)} (g(s') + c(s', s_i))$ jusqu'à ce que $s_{i-1} = s_{start}$.

iii - Illustration :

Cet exemple illustre sur un graphe simple le principe de fonctionnement de l'algorithme A* :

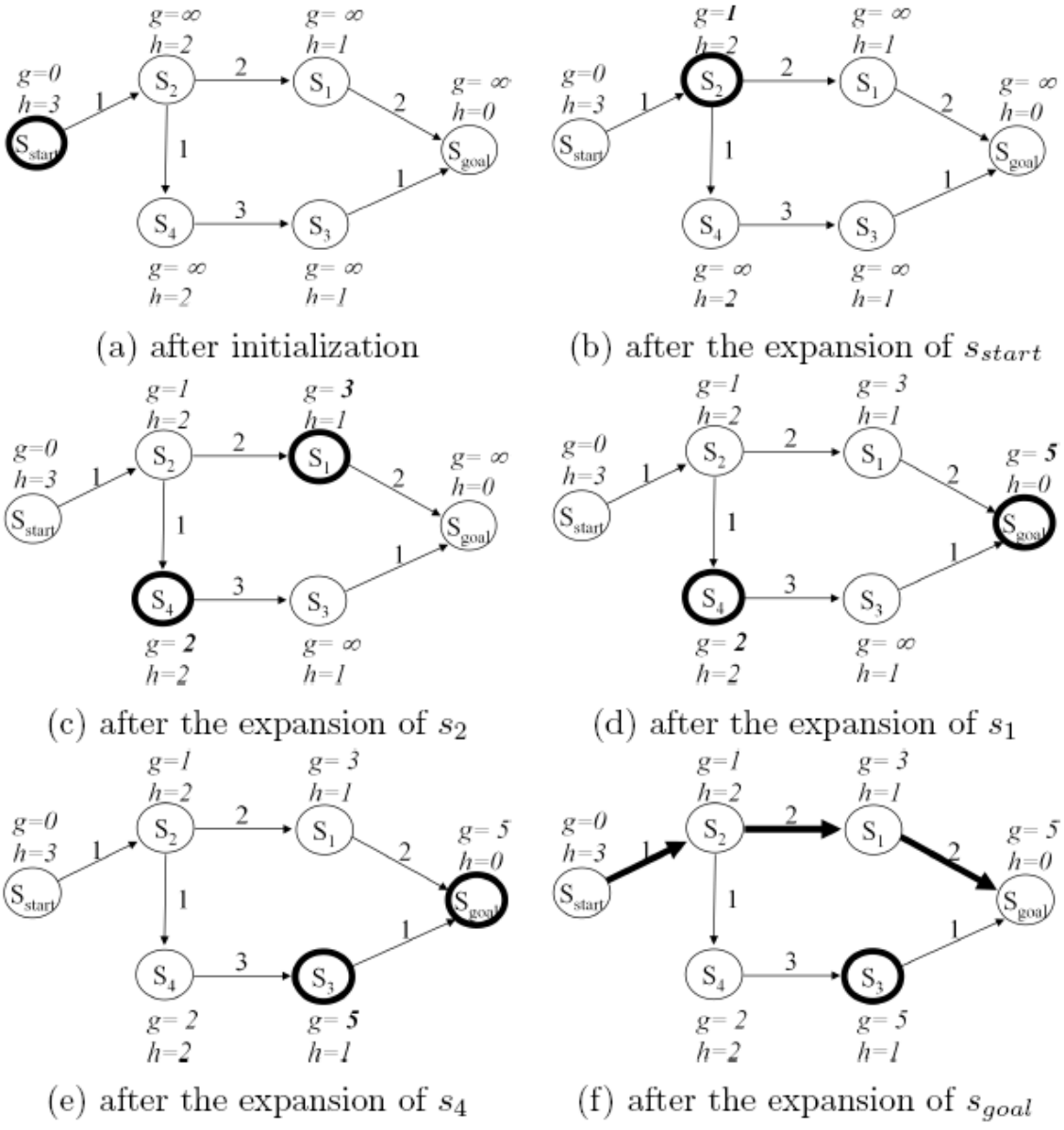


Figure 9 : [1] Exemple de fonctionnement de A. Les nœuds avec des contours en gras sont dans OPEN. Les valeurs de g qui viennent de changer sont en gras. Après que s_{goal} ait été trouvé, le chemin est affiché en gras.*

iiii - Résultats

Afin d'illustrer les avantages de cet algorithme, reprenons l'exemple précédent d'une carte sans obstacles, avec s_{start} en (100,50) et s_{goal} en (200,50). On remarque que le chemin est trouvé beaucoup plus rapidement (100 itérations contre 20000 avec l'algorithme de Dijkstra).

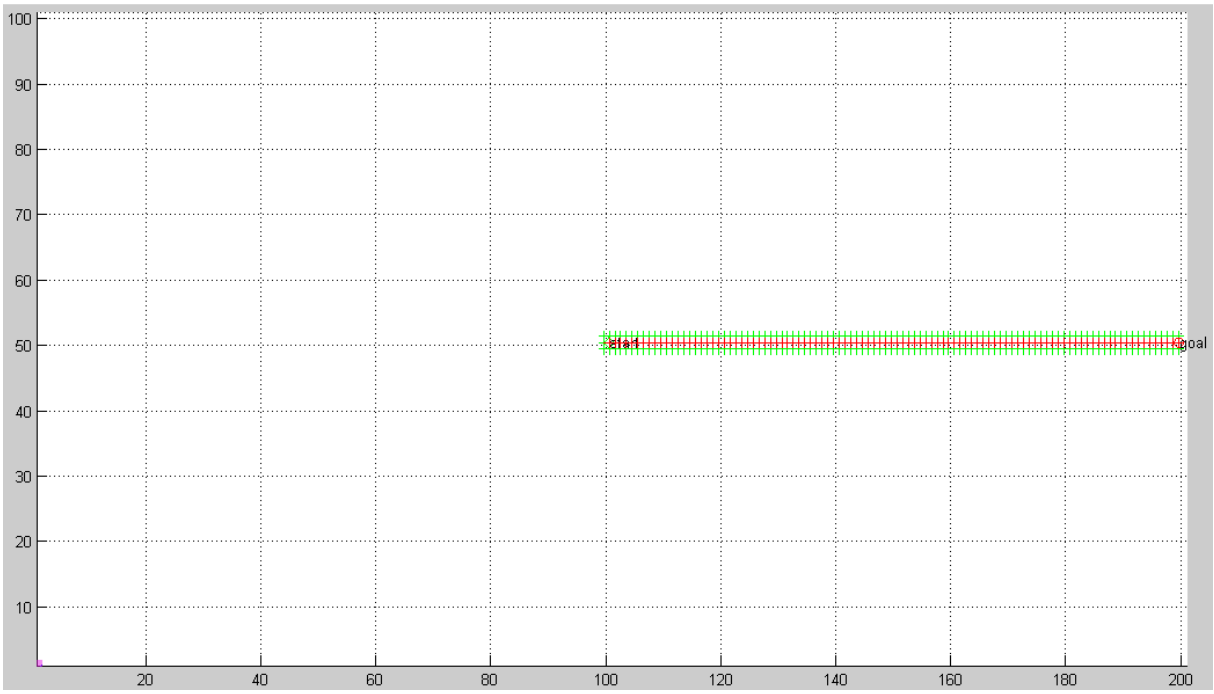


Figure 10 : Résultat après 100 itérations sur une carte sans obstacles

Les croix rouges correspondent aux nœuds présents dans CLOSED et les croix vertes à ceux présents dans OPEN, i.e. qui sont candidats à la poursuite de l'algorithme. On constate qu'on trouve bien le chemin le plus direct (optimal au sens de la longueur) et que la recherche s'est faite exclusivement dans le sens de s_{goal}.

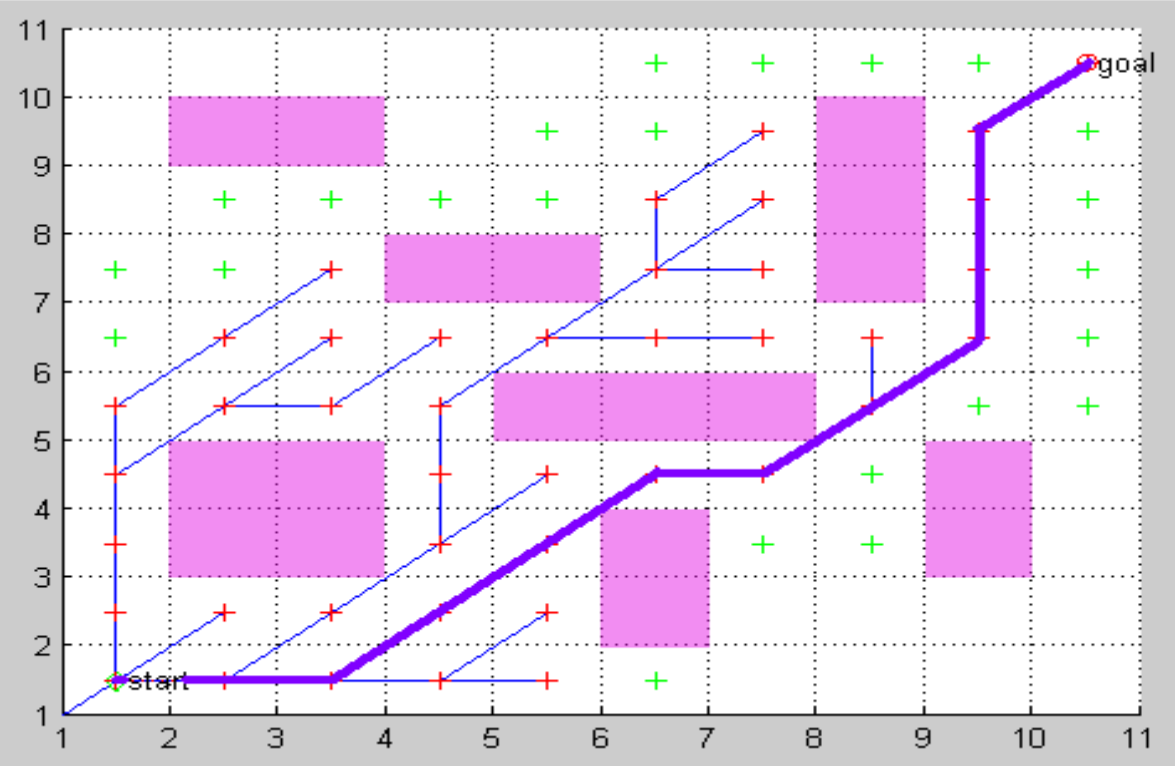


Figure 11 : Résultats du A sur un graphe simple*

Reprenons l'exemple 6, en y appliquant l'algorithme A*. Le chemin retourné par l'algorithme est représenté en mauve, les relations « nœud père / nœud fils » en bleu. Le résultat est retourné après 41 itérations, soit la moitié de ce qui était nécessaire à l'algorithme de Dijkstra pour trouver une solution. De plus, nous avons la garantie que le chemin trouvé est le chemin optimal (au sens de la distance parcourue).

Pour un mini-drone, la puissance de calcul embarquée est limitée et le gain en temps de calcul réalisé par l'algorithme A* devient alors un atout majeur.

III.5 - Généralisation

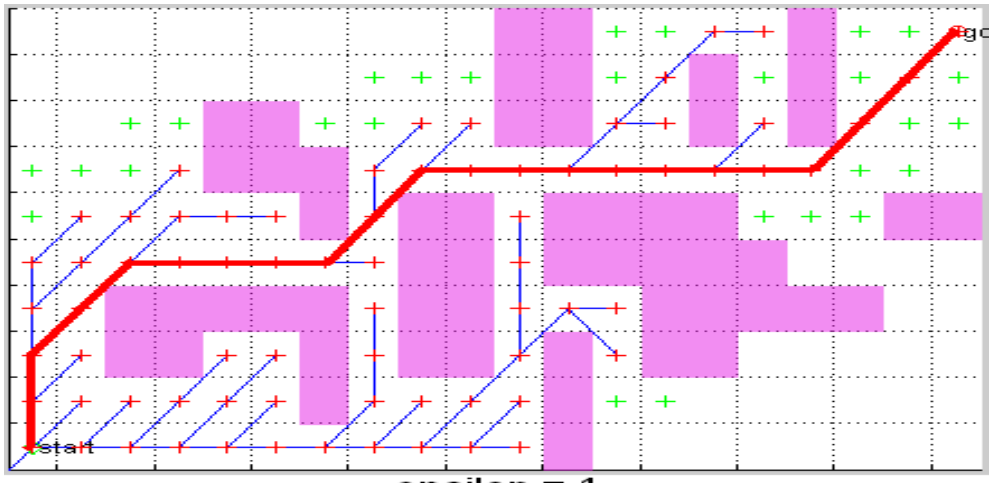
Nous avons présenté une fonction key(s) = g(s) + h(s), avec g(s) la longueur du chemin de s_{start} au nœud courant s, et h(s) l'heuristique, c'est-à-dire une estimation de la distance entre le nœud s et s_{goal}. Cette fonction va permettre de déterminer l'ordre dans lequel les nœuds vont être regardés par A*. Afin de précipiter la recherche vers s_{goal}, on pourrait modifier l'expression de key(s) afin d'apporter plus de poids à l'heuristique. On note désormais

$$key(s) = g(s) + \varepsilon . h(s).$$

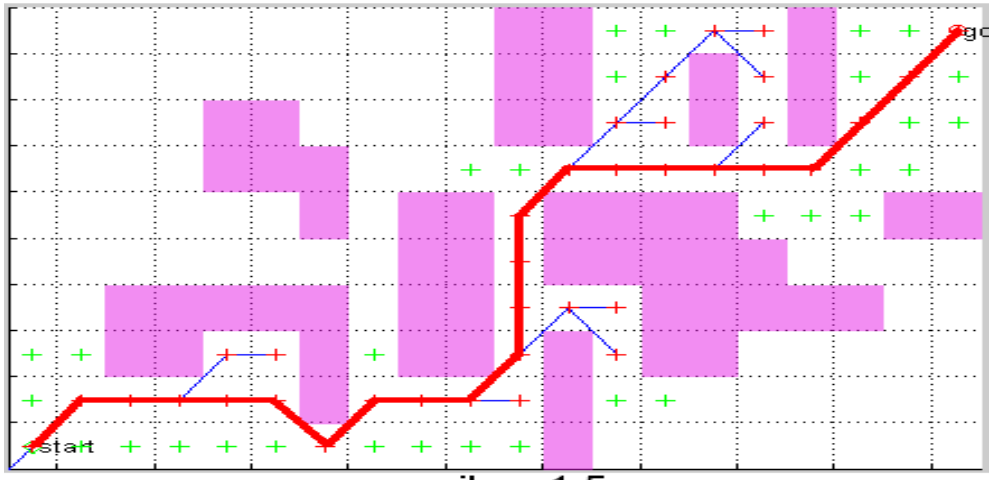
$\varepsilon = 0$ correspond à une recherche non-orientée [5] (comme l'algorithme de Dijkstra), alors que $\varepsilon = 1$ permet de se ramener à l'algorithme A*. ε>1 défini un nouvel algorithme, le **A* pondéré**.

Cet algorithme permet de réduire de façon significative le nombre d'itérations nécessaires à l'obtention d'une solution dans bien des cas. Cependant, comme le montre l'exemple présenté sur la figure 12, nous n'avons plus de garantie d'optimalité de la solution retournée par l'algorithme. Néanmoins l'utilisation d'une heuristique qui ne surestime pas le coût pour aller du nœud courant au nœud s_{goal} permet d'avoir un contrôle sur la « sous-optimalité » du chemin entre s_{start} et s_{goal}. En effet, pour une telle heuristique (distance euclidienne par exemple) nous aurons [6] :

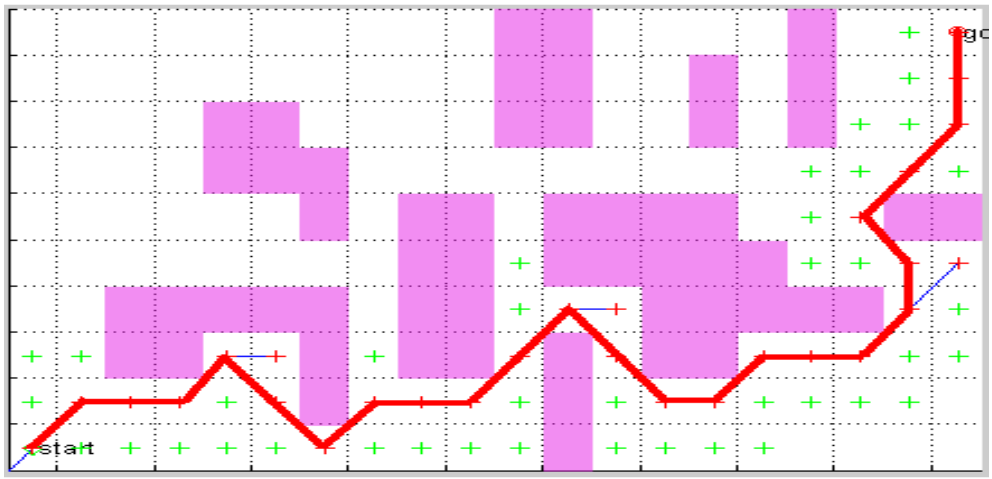
$$\mathbf{longueurChemin}_{\varepsilon>1} < \varepsilon * \mathbf{longueurCheminOptimal}$$



epsilon = 1



epsilon=1.5



epsilon = 5

Figure 12 : exemples de résultats fournis par l'algorithme A pondéré pour différentes valeur de epsilon*

Les résultats pour chaque cas sont présentés dans le tableau ci-dessous :

Epsilon	**0**	**1**	**1,5**	**5**
Nombre d'itération	145	72	36	28
Longueur chemin	23.8995	23.8995	25.3137	29.7990

Tableau récapitulatif en fonction de EPSILON

Ainsi, dans notre cas, l'utilisation d'un algorithme A* ca permettre de diminuer par 2 le nombre de cellules visitées par rapport à l'algorithme de Dijkstra pour obtenir un chemin qui sera quand même optimal. Si on n'a aucune contrainte en termes de distance à parcourir, mais une obligation de trouver le plus rapidement possible une solution, on peut augmenter la pondération sur l'heuristique (epsilon = 1,5 et 5).

Fixer Epsilon à 1,5 revient à tolérer une sous-optimalité de 50% sur la distance parcourue pour rallier s_{goal}. On constate dans cet exemple que le drone va mettre deux fois moins de temps pour obtenir un chemin par rapport à un algorithme A* classique (donc 4 fois moins de temps que l'algorithme de Dijkstra) pour obtenir un chemin qui ne sera, dans notre exemple, que 5,92% plus long.

CHAPITRE 4

Application à un environnement 3D

Rappelons que nous considérons dans notre étude un drone miniature aérien et qu'il est par conséquent amené à évoluer dans un espace à trois dimensions. On adapte donc les codes précédents de façon à produire un chemin défini comme une liste de waypoints $\mathbb{S} = \{ \ (S_x^i, S_y^i, S_z^i) \ \ \backslash \ \ i\epsilon[1{:}n], avec\ n\ le\ nombre\ de\ waypoints\}$.

Dans l'exemple suivant, l'obstacle est formé d'un « U » légèrement surélevé dans un coin et d'un cube (au centre du « U ») pour $z = 2$ à $z = 3$.

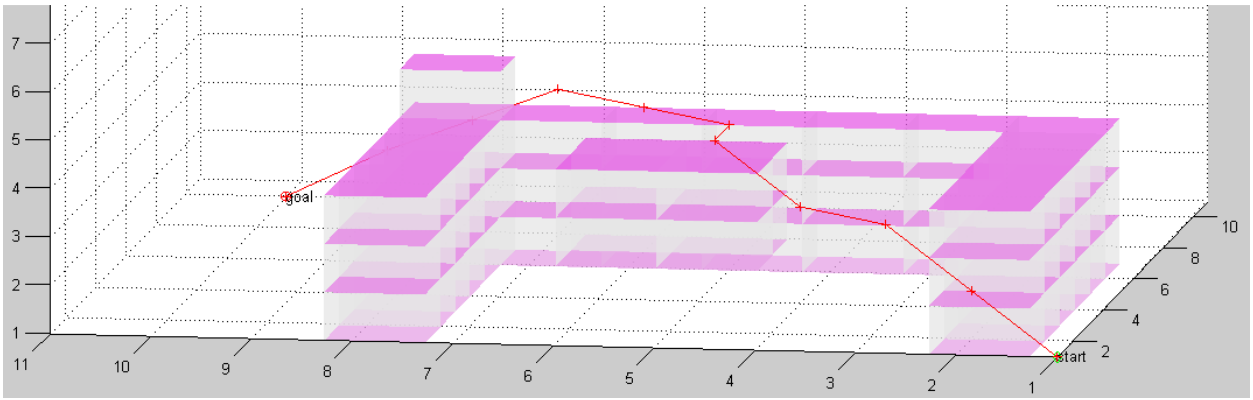


Figure 13 : génération des waypoints en 3 dimensions avec $\epsilon = 1$.

(longueur du chemin : 13.9 // nombre de cellules visitées par l'**algorithme A*** : 146)

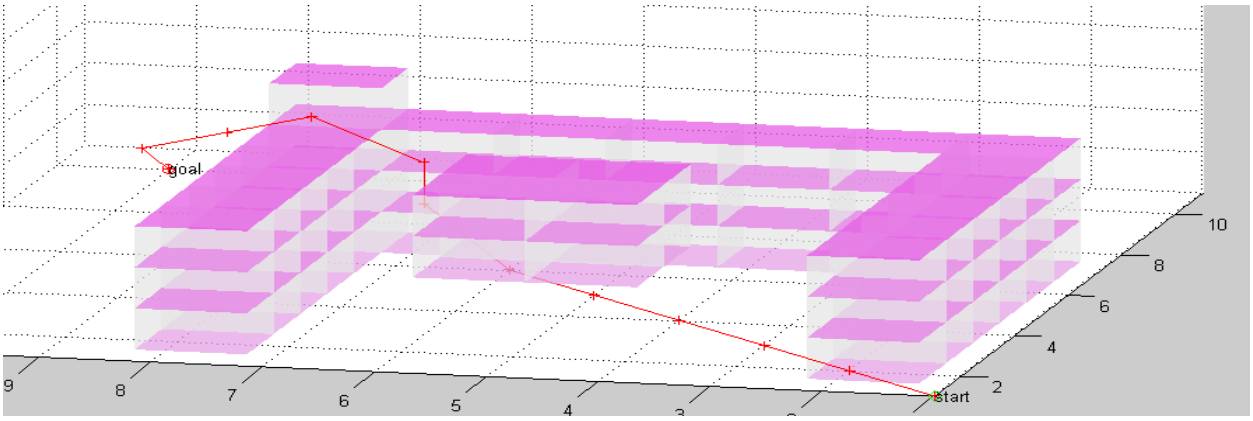


Figure 14: génération des waypoints en 3 dimensions pour $\epsilon = 2.5$.

(longueur du chemin : 16.1 // nombre de cellules visitées par l'**algorithme A*** : 24)

On obtient des résultats similaires en trois dimensions à ceux obtenus en deux dimensions (réduction du nombre de cellules visitées et donc du temps nécessaire à l'obtention du chemin, sous-optimalité…)

CHAPITRE 5
Génération de trajectoire continue

L'objectif de notre étude est de commander un drone pour qu'il parte d'une position connue s_{start} à priori pour rallier un objectif s_{goal}, connu lui aussi. Les algorithmes présentés dans le chapitre précédent permettent de générer une trajectoire définie en termes de points de passage (ou waypoints). Or, nous voulons fournir une trajectoire continue au drone pour pouvoir utiliser certaines lois de commande. Nous avons choisi de nous intéresser aux courbes de Bézier afin de la générer cette trajectoire de référence à partir de la liste des waypoints.

V- 1 : Introduction aux courbes de Bézier :

Les courbes de Bézier ont été développées par Pierre Bézier, ingénieur à la Régie Renault afin de définir des courbes complexes dont on ne connait pas a priori l'expression analytique. Ces courbes sont générées à partir d'une série de points, appelés points de contrôle, et des polynômes de Bernstein. Elles sont principalement utilisées en infographie pour la réalisation de surfaces complexes et continues afin de minimiser la taille nécessaire au stockage de ces informations.

Dans notre cas, la série de points considérée est une liste de N points générée par l'algorithme de recherche de chemin.

V-2 - Définition :

On note m le degré de la courbe de Bézier, tel que $m = N - 1$. Il existe plusieurs façons de définir une courbe de Bézier : vectorielle, matricielle, paramétrique [7]. Nous allons nous intéresser plus particulièrement à la formulation paramétrique puisqu'elle nous permettra de définir une trajectoire de référence en position pour chaque coordonnée spatiale.

On définit les polynômes de Bernstein comme suit :

$$B_i^m = \binom{m}{i} u^i (1-u)^{m-i}, \quad u \in [0;1]$$

Avec

$$\binom{m}{i} = \begin{cases} C_m^i = \dfrac{m!}{i!\,(m-i)!}, & si\ 0 \le i \le m \\ 0, & sinon \end{cases}$$

Soit une courbe de Bézier formée par N points de contrôle.Notons S_i les coordonnées du i-ème point de contrôle. L'ensemble des points P appartenant à cette courbe est donnée par [6]:

$$P(u) = \sum_{i=0}^{m} B_i^m(u)\, S_i, \qquad u \in [0;1]$$

On obtient alors la forme paramétrique qui nous intéresse en réécrivant l'équation :

$$P(u) = \begin{cases} x^r(u) = \sum_{i=0}^{m} B_i^m(u)\, S_{x_i} \\ y^r(u) = \sum_{i=0}^{m} B_i^m(u)\, S_{y_i} \\ z^r(u) = \sum_{i=0}^{m} B_i^m(u)\, S_{z_i} \end{cases} , pour\ u \in [0,1]$$

On pourra ainsi passer en référence la position en x, en y, en z.

V-3 – Quelques propriétés :

Les polynômes de Bernstein confèrent aux courbes de Bézier des propriétés particulières, parmi lesquelles deux sont importantes pour notre application :

Propriété 1 : Une courbe de Bézier est de classe C^m sur [0,1] (i.e. continue et dérivable m fois).

Ceci va permettre de fournir des consignes de position et de vitesse (dérivée première) continues à la partie commande du drone miniature.

Propriété 2 : Une courbe de Bézier passe par le premier et le dernier point de contrôle, respectivement S_0 et S_N et est tangente aux vecteurs S_0S_1 et $S_{N-1}S_N$.

NB : On peut trouver les démonstrations dans [6].

V-4- Exemple

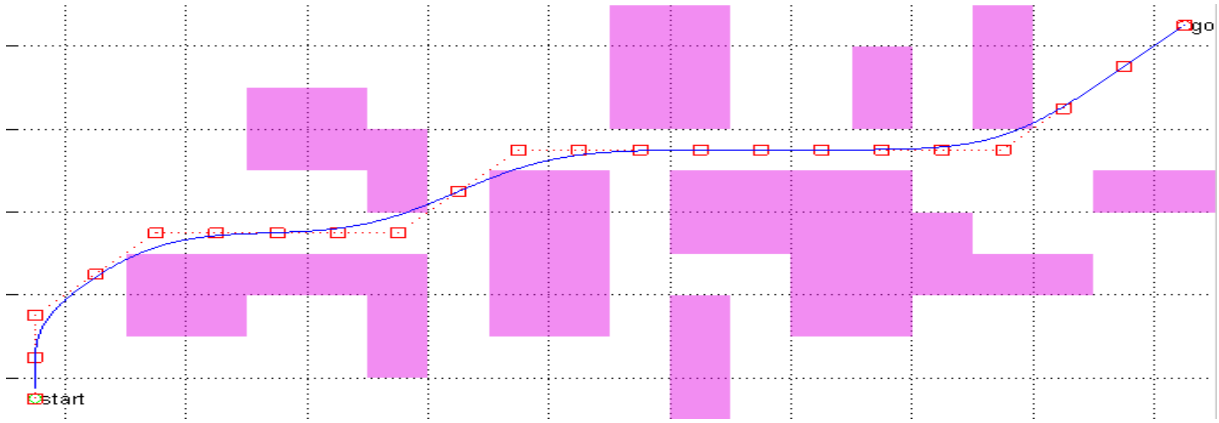


Figure 15 : exemple de génération de trajectoire continue

On représente ici en bleu la trajectoire qui sera passée en référence à notre commande. On remarquera cependant que nous n'avons pas de garantie sur le fait que la trajectoire intersecte ou pas les obstacles.

Le travail effectué pour générer les courbes de Bézier est bien entendu applicable à un environnement en trois dimensions.

Conclusion de la partie 1

Pour résumer ce qui a été fait dans cette partie, nous avons vu que l'algorithme de Dijkstra permet d'obtenir, pour un graphe donné l'arbre couvrant minimum, et donc le chemin le plus court entre la position actuelle et le lieu où le drone doit se rendre. Toutefois, pour un drone miniature, où la puissance de calcul embarquée est très limitée, il est nécessaire de trouver un chemin reliant la position actuel au lieu de réalisation de la mission plus rapidement qu'en explorant tout le graphe.

C'est pourquoi nous avons étudié l'algorithme A*. Celui-ci ajoute dans l'évaluation de l'ordre de traitement des cellules candidates une fonction appelée heuristique qui permet de favoriser les cellules plus proches de l'objectif (typiquement la distance euclidienne entre la cellule considérée et l'objectif). On effectue ainsi une recherche orientée. On réduit considérablement le nombre de cellules visitées et donc le temps nécessaire pour trouver une solution. De plus, à condition d'utiliser une heuristique qui ne surestime pas les distances, on peut garantir l'optimalité (au sens de la longueur) du chemin parcouru.

Afin d'obtenir une solution encore plus rapide, on peut pondérer l'heuristique dans la fonction de coût par une valeur strictement supérieure à 1. Cela permet de réduire encore le nombre de cellules visitées pour trouver une solution. En contrepartie, le chemin trouvé n'est plus le plus court, même si on peut quand même quantifier cette sous-optimalité.

Après avoir trouvé l'ensemble des points de passage permettant de relier la position actuelle à l'arrivée, on utilise des courbes de Bézier afin de fournir une référence continue à la partie commande du drone. Cette méthode est relativement commode puisque l'ensemble des points de passages vont correspondre directement aux points de contrôle des courbes de Bézier. De plus, excepté le manque de contrôle sur une possible intersection de la courbe générée et des obstacles, ces fonctions bénéficient de propriétés qui les rendent intéressantes pour la partie guidage/pilotage (courbes régulière C^N sur leur ensemble de définition)

La section 'suivi de trajectoire' est abordée en deuxième partie de ce mémoire.

PARTIE 2 :

COMMANDE
SUIVI DE TRAJECTOIRE

CHAPITRE 6

Modélisation du drone miniature à voilure tournante.

Afin de modéliser précisément un drone miniature à voilure tournante, il est nécessaire de connaître beaucoup de caractéristiques spécifiques à notre véhicule : aérodynamique du véhicule, dynamique des actionneurs, caractéristiques mécaniques et type de configuration (quadri rotor, hélicoptère classique…).

Pour conserver un propos le plus généraliste possible, le modèle que nous utiliserons sera un modèle dynamique de type mécanique du solide, dont la représentation peut être commune à tous les drones.

VI.1 - Notations et mouvement du drone :

On considère notre drone comme un corps rigide. On note :

- M sa masse
- G sont centre de gravité
- I son tenseur d'inertie
- On définit aussi deux repères :
- R_I un repère inertiel $\{O, e_1, e_2, e_3\}$ avec e_3 dirigé vers le bas.
- R_B un repère lié au véhicule $\{O, e_1^B, e_2^B, e_3^B\}$ où e_1^B pointe vers l'avant du véhicule et e_3^B définit l'axe vertical du véhicule et pointe vers le bas. e_2^B complète les vecteurs précédents pour former une base orthonormée directe.

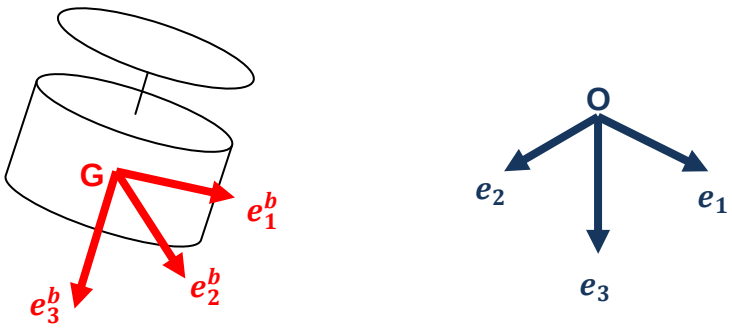


Figure 16 : Présentation des différents référentiels utilisés

Le passage d'un repère à l'autre se fait via 3 rotations successives. On retiendra l'écriture ***roulis*** $(\boldsymbol{\phi})$***- tangage*** $(\boldsymbol{\theta})$***- lacet*** $(\boldsymbol{\psi})$ pour distinguer les rotations respectivement autour de e_1^b, e_2^b, e_3^b. La matrice de rotation s'exprime alors comme produit de trois matrices :

$$R(\phi,\theta,\psi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Ou encore, avec cX = cos(X) et sX = sin(X)

$$R(\phi,\theta,\psi) = \begin{pmatrix} c\theta\, c\psi & -c\phi\, s\psi + s\theta\, c\psi\, s\phi & s\psi\, s\phi + s\theta\, c\psi\, c\phi \\ c\theta s\psi & -c\psi\, s\psi + s\theta\, s\psi\, s\phi & -c\psi\, s\phi + s\theta\, c\phi\, s\psi \\ -s\theta & c\theta\, s\phi & c\theta\, c\phi \end{pmatrix}$$

On a alors, en notant $R = R(\psi,\theta,\psi)$, les relations de changement de base :

$$\begin{cases} [u]_B = R^T [u]_I \\ [u]_I = R\ [u]_B \end{cases}$$

Afin de décrire les mouvements du drone miniature aérien, on définit :

- $\xi = [\, x\ \ y\ \ z\,]^T$ la position du centre de gravité du drone dans le repère inertiel.
- $v = \left[\, v_x\ v_y\ v_z \right]^T$ ses vitesses dans le repère inertiel.
- ϕ, θ, ψ respectivement le roulis, tangage, lacet
- $\Omega = \left[\omega_p\ \omega_q\ \omega_r\right]^T$ le vecteur vitesse de rotation instantané dans le repère lié au véhicule

VI.2 - Equations d'état du système :

On considère le système « drone miniature ». L'ensemble des forces qui s'appliquent au véhicule est représenté sur la figure 15 :

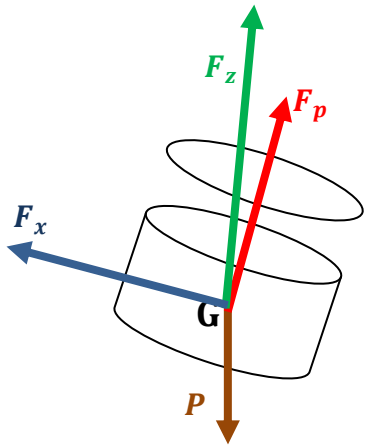


Figure 17 : Forces qui s'appliquent sur le drone miniature.

Avec :

- F_p la poussée générée par la propulsion du drone miniature
- F_a la résultante des forces aérodynamiques, ou encore $F_a = F_x + F_z$, avec
  - o F_x la trainée
  - o F_z la portance
- $P = m.g.e_3$ le poids du drone

F_{ext} la résultante des forces extérieures de perturbation du drone (vent, rafale)
Dans le cas d'un drone miniature, la majeure partie des missions se font à faible vitesse (vol quasi-stationnaire), on peut par conséquent faire l'hypothèse suivante :

(hyp 1) La résultante des forces aérodynamiques ainsi que la poussée seront selon l'axe e_b^3 [2]. Cette résultante sera notée, dans le repère inertiel :

$$F_a + F_p = -T\,R\,e_3, \qquad avec\ T = \|F_p + F_a\|$$

(Et R la matrice de changement de repère précédemment introduite)

De plus, le drone miniature est soumis à un ensemble de couples :

- Γ_1, dû aux actionneurs du drone miniature.
- Γ_2, dû aux phénomènes aérodynamiques et gyroscopiques produits par la rotation des rotors
- M_{ext} engendré par les forces extérieures de perturbation

On note

$$\Gamma = \Gamma_1 + \Gamma_2$$

On néglige le terme de couplage entre la dynamique de rotation et celle de translation créée par l'apparition de forces issues de la réalisation du couple Γ_1 par les actionneurs du véhicule.

En considérant l'hypothèse précédente, on a :

$$\begin{cases} \dot{\xi} = v \\ \dot{v} = -\frac{1}{m}.T.R.e_3 + g.e_3 + \frac{1}{m}.F_{ext} \\ \dot{R} = R.\Omega_x \\ I.\dot{\Omega} = -\Omega_x I\Omega + \Gamma + M_{ext} \end{cases}$$

Avec

$$\Omega_x = \begin{bmatrix} 0 & -\omega_r & \omega_q \\ \omega_r & 0 & -\omega_p \\ -\omega_q & -\omega_p & 0 \end{bmatrix}$$

On considère comme commande T et Γ.

L'avantage de ce modèle est de ne pas tenir compte de l'architecture mécanique du drone. On pourra ensuite spécifier plus en détail ces commandes en les reliant directement aux commandes des actionneurs, selon la configuration du véhicule.

CHAPITRE 7
COMMANDE DU DRONE MINIATURE

VII.1 - Etat de l'art sur la commande des drones miniatures :

Plusieurs types de commandes ont déjà été utilisés pour guider et piloter les drones aériens. On reprend ici une partie de l' état de l'art effectué dans [2].

i - Retour d'état linéaire :

Une linéarisation de notre système d'état autour d'un point de fonctionnement permet l'utilisation de régulateurs de type PID. On retrouve ce type d'approche dans [11].

ii - Commande par extension dynamique :

La linéarisation entrée-sorties fait partie des commandes par extension dynamique. Elle n'est cependant applicable qu'à des systèmes à déphasage minimal. Des travaux relatifs à cette méthode peuvent être trouvés dans [10].

Une autre méthode est d'utiliser une commande par backstepping en considérant notre système comme une chaîne d'intégrateur :

$$\begin{cases} \dot{\xi} = f_1(v) \\ \dot{v} = f_2(R) \\ \dot{R} = f_3(R, \Omega) \\ \dot{\Omega} = f_4(\Omega, \Gamma) \end{cases}$$

avec $\forall\, i \in [1,2,3,4], f_i de\ classe\ C^1$.

Toutefois, ces méthodes restent difficiles à mettre en place compte tenu des difficultés d'obtention de mesure de T et de $\dot{T}$.

En pratique, la mesure de la translation est donnée par un capteur de type caméra ou GPS, et la rotation généralement par une centrale inertielle. Or, pour ces types de capteurs, la fréquence d'obtention de mesures de la translation est bien inférieure à celle de la rotation, et la commande ne peut être calculée qu'en se basant sur la fréquence la plus faible. Il y a donc un réel problème pour appliquer les méthodes précédemment citées.

Il faut alors proposer une commande qui permette de séparer les cadences de mesure, comme la commande hiérarchique.

iii - Commande hiérarchique :

On va dans un premier temps considérer que $T.R.e_3$ est notre commande de la dynamique de translation. On construit ensuite un retour d'état pour fixer une valeur $(T.R.e_3)^d$ à ce vecteur de commande. On suppose que la dynamique des actionneurs permettant de réaliser T est beaucoup plus rapide que celles régissant la translation et la rotation du drone, ce qui permet d'avoir $\forall\, t \in \mathbb{R}^+,\ T^d = T$, et donc $(T.R.e_3)^d = T(R.e_3)^d = T.R^d.e_3$.

On fixera ensuite la consigne pour le lacet ψ^d ce qui permettra de calculer la valeur de la matrice de rotation R^d qui tiendra leu de consigne pour la dynamique de rotation.

VII – 2 : Commande hiérarchique :

Grâce à la commande hiérarchique, on va pouvoir scinder notre système en deux sous-systèmes et construire ensuite une loi de commande pour chacun d'eux.

i -Stratégie de commande pour le suivi de trajectoire

Rappelons l'objectif de notre étude. Un drone miniature évolue dans un milieu en trois dimensions. Ainsi, à partir de sa perception de l'environnement, il détermine une série de points de passage qui vont lui permettre de générer une trajectoire continue (via les courbes de Bézier). Cette courbe servira de trajectoire de référence à notre drone miniature pour rallier l'objectif de sa mission. On notera ξ^d l'ensemble des positions de référence au cours du temps, $v^d = \dot{\xi}^d$ et $\dot{v}^d = \ddot{\xi}^d$ ses dérivées temporelles, et ψ^d le lacet désiré.

Le système « drone miniature » est un système sous-actionné : il dispose de quatre entrées $(T, \Gamma_1\Gamma_2, \Gamma_3)$ pour six degrés de liberté $(\phi, \theta, \psi, x, y, z)$. On en pourra donc théoriquement stabiliser que 4 sorties indépendantes. Cependant, le choix de ces sorties ne peut être arbitraire [15]. On choisira comme sorties x, y, z, ψ, ce qui correspond à l'utilisation automatique du drone, et on cherchera donc à stabiliser le système autour de la position $\xi^d(t) = [x^d, y^d, z^d]$ et du lacet $\psi^d(t)$.

Loi de commande en position :

On veut déterminer une loi de commande qui permette de faire converger ξ vers ξ^d et v vers v^d. On aura donc

$$T.R^d.e_3 = f(\xi, v, \xi^d, v^d, \dot{v}^d)$$

On extrait T via le calcul de la norme de f et on a la direction:

$$R^d.e_3 = \frac{1}{T} f(\xi, v, \xi^d, v^d, \dot{v}^d)$$

$T = 0$ correspond à un mouvement de chute libre du véhicule. Ce type de mouvement est exclu pour notre application, et on peut donc diviser par T.

Loi de commande en attitude :

La dynamique de R^d dépend à la fois de l'état de la dynamique de translation et de la référence. En effet, le lacet $\psi^d(t)$ peut être choisi non constant, et la référence en position ξ^det ses dérivées successives apparaissent dans le calcul de R^d. La vitesse de rotation désignée $\dot{\Omega_d}$ définie par $\dot{R^d} = R^d . \Omega_x^d$ peut alors être décomposée en

$$\Omega^d = \Omega^d(X) + \Omega^d(t)$$

Avec $\Omega^d(X)$ dépend de l'état de la dynamique de translation (X) et $\Omega^d(t)$ ne dépend que de la trajectoire de référence.

On cherchera, pour la dynamique de rotation à synthétiser une loi de commande permettant de faire converger R vers R^d et Ω vers Ω^d de la forme :

$$\Gamma = h\,(R, R^d, \Omega, \Omega^d)$$

On a alors le schéma de commande suivant, qui permet de prendre en compte les références $\xi^d, v^d, \dot{v}^d, \psi^d$ pour le suivi de trajectoire.

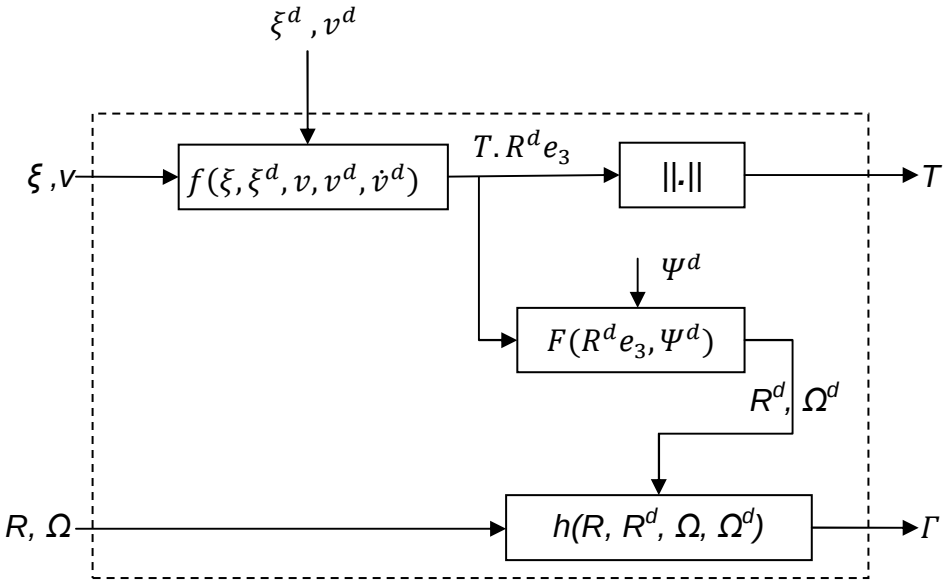


Figure 18: [2] Principe de la commande hiérarchique pour le suivi de trajectoire.

Hypothèses pour la synthèse des lois de commande.

En considérant une séparation des échelles de temps, on considère la dynamique de rotation comme beaucoup plu rapide que la dynamique de translation. On peut alors synthétiser $T.R^d.e_3$ en supposant que :

(hyp2) : la convergence de la dynamique de rotation est beaucoup plus rapide que la dynamique de translation, et donc $R = R^d$.

De plus, la composante $\Omega^d(t)$ est liée à la dynamique de la référence et peut générer une erreur si elle est négligée. On fait donc l'hypothèse suivante :

(hyp3) : La dynamique de translation est beaucoup plus lente que la dynamique de rotation et donc $\Omega^d(X) = 0$ *pour le suivi de trajectoire* $\Omega^d = \Omega^d(t)$.

Sous ces hypothèses, on peut synthétiser les lois de commande qui serviront à suivre la trajectoire de référence. Une manière d'obtenir une preuve de stabilité est l'utilisation de la théorie des perturbations singulière [2, 12, 13, 14].

VII-3 Commande par retour d'état statique:

i- Synthèse de la loi de guidage :

On va s'intéresser dans un premier temps à la commande en position. D'après le système d'état présenté en VI-2, le sous-système considéré pour la synthèse de la loi de guidage est le suivant :

$$\begin{cases} \dot{\xi} = v \\ \dot{v} = -\frac{1}{m}.T.R^d.e_3 + g.e_3 + T.(R - R^d)e_3 \end{cases}$$

Sous l'hypothèse 2, $R = R^d$, le système précédent devient :

$$\begin{cases} \dot{\xi} = v \\ \dot{v} = -\frac{1}{m}.T.R^d.e_3 + g.e_3 \end{cases}$$

Pour le suivi de trajectoire, on pose :

$$\begin{cases} \epsilon_1 = \xi - \xi^d \\ \epsilon_2 = \dot{\epsilon_1} = v - v^d \end{cases}$$

On reformule alors le système précédent pour le suivi de trajectoire :

$$\begin{cases} \dot{\epsilon_1} = \epsilon_2 \\ m\dot{\epsilon_2} = -T.R^d.e_3 + m.g.e_3 - m.\dot{v^d} \end{cases}$$

On introduit la variable δ pour le backstepping telle que

$$\delta = \frac{m}{k_1}\epsilon_2 + \epsilon_1$$

Soit la fonction de Lyapunov candidate :

$$\mathbb{L}_1 = \frac{1}{2}\epsilon_1^T\epsilon_1 + \frac{1}{2}\delta^T\delta$$

La dérivée de cette fonction vaut

$$\dot{\mathbb{L}_1} = -\frac{k_1}{m}\epsilon_1^T\epsilon_1 + \delta^T\left\{\left[\frac{k_1}{m}\delta + \frac{m}{k_1}ge_3 - \frac{m}{k_1}\dot{v^d}\right] - \frac{T}{k_1}R^de_3\right\}$$

En choisissant

$$\boxed{-TR^d e_3 = k_1\left\{-\frac{2k_1}{m}\delta_1 - \frac{m}{k_1}ge_3 + \frac{m}{k_1}\dot{v}_d\right\},\quad avec\ k_1 > 0}$$

On obtient

$$\dot{\mathbb{L}_1} = -\frac{k_1}{m}\epsilon_1^T\epsilon_1 - \frac{k_1}{m}\delta^T\delta = -\frac{k_1}{m}\mathbb{L}_1 \leq 0$$

ce qui garantit la stabilité exponentielle du système sans l'erreur d'orientation.

ii- Synthèse de la loi de pilotage :

Après avoir traité la partie guidage du drone, nous allons nous intéresser à la commande en attitude. On rappelle la dynamique de rotation :

$$\begin{cases} \dot{R} = R.\Omega_x \\ I.\dot{\Omega} = -\Omega_x I\Omega + \Gamma \end{cases}$$

On prend en compte l'hypothèse 3 ($\Omega^d = \Omega^d(t) \gg \Omega^d(X)$) pour les calculs qui suivent. La partie guidage fournit R^d comme consigne pour la rotation.

On introduit $\tilde{R} = (R^d)^T R$. Le système précédent devient alors :

$$\begin{cases} \dot{\tilde{R}} = -\Omega_x^d.\tilde{R} + \tilde{R}\Omega_x \\ I.\dot{\Omega} = -\Omega_x I\Omega + \Gamma \end{cases}$$

On introduit pour le backstepping les variables suivantes :

$$\delta_2 \triangleq \Omega - \Omega^d + Vex\left(\Pi_{a_{\tilde{R}}}\right), \qquad avec \begin{cases} Vex\begin{pmatrix} 0 & a_{12} & a_{13} \\ -a_{12} & 0 & a_{23} \\ -a_{13} & -a_{23} & 0 \end{pmatrix} = \begin{bmatrix} a_{32} \\ a_{13} \\ a_{12} \end{bmatrix} \\ \Pi a_{\tilde{R}} = \dfrac{\left(\tilde{R} - \tilde{R}^T\right)}{2} \end{cases}$$

Avec cette notation le système devient :

$$\begin{cases} \dot{\tilde{R}} = -\Omega_x^d.\tilde{R} + \tilde{R}\Omega_x \\ I.\dot{\delta_2} = -\Omega_x I\Omega + \Gamma - \mathrm{I}\,\dot{\Omega}^{\mathrm{d}} + \mathrm{I}\,\mathrm{Vex}(\dot{\Pi}_{\mathrm{a}\tilde{\mathrm{R}}}) \end{cases}$$

On considère aussi la fonction de Lyapunov candidate :

$$\mathbb{L}_2 = \frac{1}{2}Tr\left(I_3 - \tilde{R}\right) + \frac{1}{2}\delta_2^T I\delta_2$$

où Tr désigne la trace d'une matrice.

La dérivée de $\mathbb{L}_2$ [2] vaut :

$$\dot{\mathbb{L}_2} = \delta_2^T Vex\left(\Pi_{a_{\tilde{R}}}\right) - \left\|Vex\left(\Pi_{a_{\tilde{R}}}\right)\right\|^2 + \delta_2^T\left(-\Omega_x I\Omega + \Gamma - I.\dot{\Omega}^d + I.Vex\left(\dot{\Pi}_{a_{\tilde{R}}}\right)\right)$$

En choisissant :

$$\boxed{\Gamma = \Omega_x I\Omega + I\dot{\Omega}^d - I.Vex\left(\dot{\Pi}_{a_{\tilde{R}}}\right) - Vex\left(\Pi_{a_{\tilde{R}}}\right) - k_2\delta_2, \qquad avec\ k_2 > 0}$$

On a alors

$$\dot{\mathbb{L}_2} = -\|Vex(\Pi a_{\tilde{R}})\|^2 - k_2\|\delta_2\|^2 \leq 0$$

Ce qui garantit la stabilité exponentielle du sous-système permettant la synthèse de la loi de pilotage.

iii- Stabilité globale du système :

Principe général :

$$\begin{cases} \dot{\epsilon_1} = \epsilon_2 \\ m\dot{\epsilon_2} = -T.R^d.e_3 + m.g.e_3 - m.\dot{v^d} + T.(R - R^d)e_3 \\ \dot{\tilde{R}} = -\Omega_x^d.\tilde{R} + \tilde{R}\Omega_x \\ I.\dot{\delta_2} = -\Omega_x I\Omega + \Gamma - \mathrm{I}\,\dot{\Omega}^{\mathrm{d}} + \mathrm{I\ Vex}(\dot{\Pi}_{\mathrm{a}\tilde{\mathrm{R}}}) \end{cases}$$

On a stabilité exponentielle des deux sous-systèmes encadrés. De plus, $(R - R^d)e_3$ converge exponentiellement vers 0 et T reste borné. Par conséquent, $T(R - R^d)e_3$ est borné et converge exponentiellement vers 0.

Au final, ceci va nous garantir la stabilité exponentielle de tout le système [12].

Un exemple de suivi de trajectoire sera présenté dans la prochaine section.

VII-4 Exemple de suivi de trajectoire :

Afin d'illustrer les lois de commandes présentées dans le ***chapitre VII-3,*** on donne à la partie commande du drone miniature une trajectoire de référence prédéfinie (figure 17). Cette trajectoire se découpe en trois parties successives:

- Montée verticale du drone de (0, 0, 0) jusque (0, 0, 2)
- Ralliement du point (4, 4, 2) à altitude constante
- Progression en hélice jusqu'à l'objectif en (10, 4, 4)

Les points de départ s_{start} et d'arrivée s_{goal}sont respectivement placés en (0, 0, 0) et en (10, 4, 4).

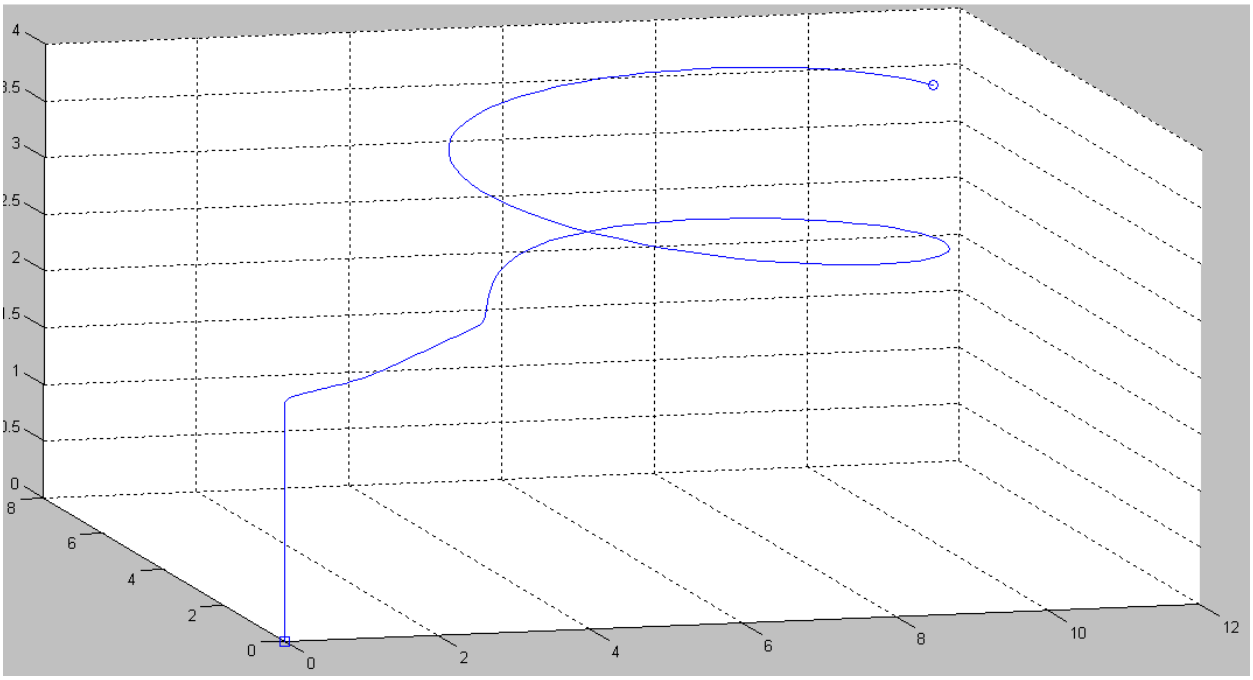


Figure 19 : exemple de trajectoire de référence pour la partie commande du drone.

On applique alors les lois de commande précédemment établies pour suivre cette trajectoire. Les résultats sont présentés en figure 20. La courbe bleue représente la trajectoire effective du drone et la courbe rouge représente la trajectoire de référence.

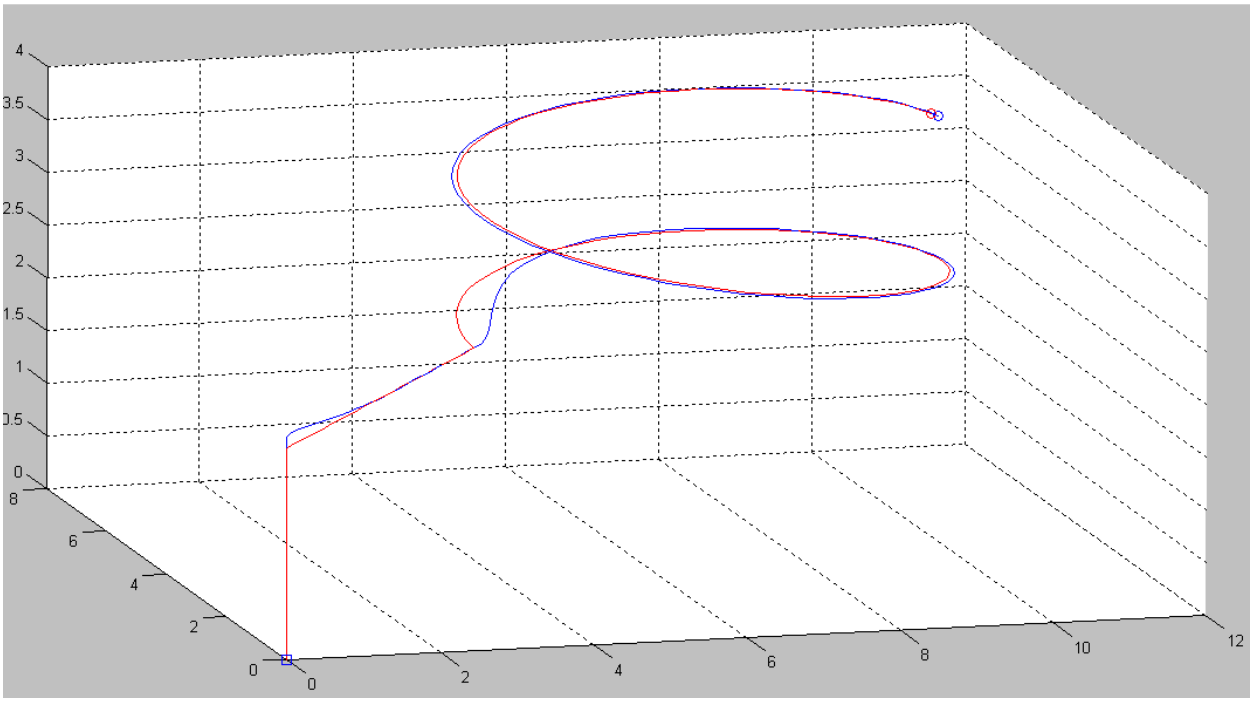


Figure 20 : résultats du suivi de trajectoire

La stratégie de commande proposée dans le chapitre VII permet bien un suivi de la trajectoire de référence.

De plus, il est possible de proposer une référence pour le lacet du drone indépendamment des autres paramètres. Ainsi, on pourra suivre avec une caméra un cap précis pendant l'évolution du drone.

Conclusion de la partie 2

Dans cette partie, l'objectif était de déterminer des lois de commande permettant le suivi de trajectoire.

En supposant que la dynamique de rotation est beaucoup plus rapide que la dynamique de translation, il est possible de découper le système d'état complet en deux sous-systèmes grâce au principe de la commande hiérarchique. Pour chacun des sous-systèmes ainsi générés, on peut synthétiser des lois de commande pour le guidage puis pour le pilotage par backstepping grâce aux hypothèses 2 et 3.

Ces commandes permettent d'avoir une stabilité exponentielle du système et donc d'assurer un bon suivi de la trajectoire de référence.

Le but de la partie suivante sera de coupler la génération de trajectoire précédemment décrite avec l'utilisation de ces lois de commande permettant le suivi de cette trajectoire. On prendra aussi en compte les incertitudes du milieu (découverte de nouveaux obstacles au cours de la mission a priori non connus du drone) afin d'effectuer des corrections en ligne sur la trajectoire de référence.

PARTIE 3 :

COUPLAGE

GENERATION DE TRAJECTOIRE / COMMANDE

CHAPITRE 8
PRINCIPE DU COUPLAGE ET SIMULATION

VIII-1 - Principe :

On va dans un premier temps exposer les grandes lignes du fonctionnement du couplage entre l'algorithme de génération de trajectoire et la commande.

Notre drone miniature va posséder une connaissance à priori de son environnement. On stocke les données connues dans une variable : MAP_connue. Il s'agit simplement d'une cartographie de l'environnement tel que connu par le drone. Il possède de surcroît des capteurs (caméra ou scanner laser par exemple) lui permettant d'acquérir des informations sur le milieu dans lequel il évolue.

A partir de la connaissance de MAP_connue dont il dispose à l'instant t_0, date de début de la mission, le drone va générer une première trajectoire de référence grâce à l'algorithme A* présenté en première partie. Il va alors, grâce aux commandes décrites dans la partie 2 suivre cette trajectoire.

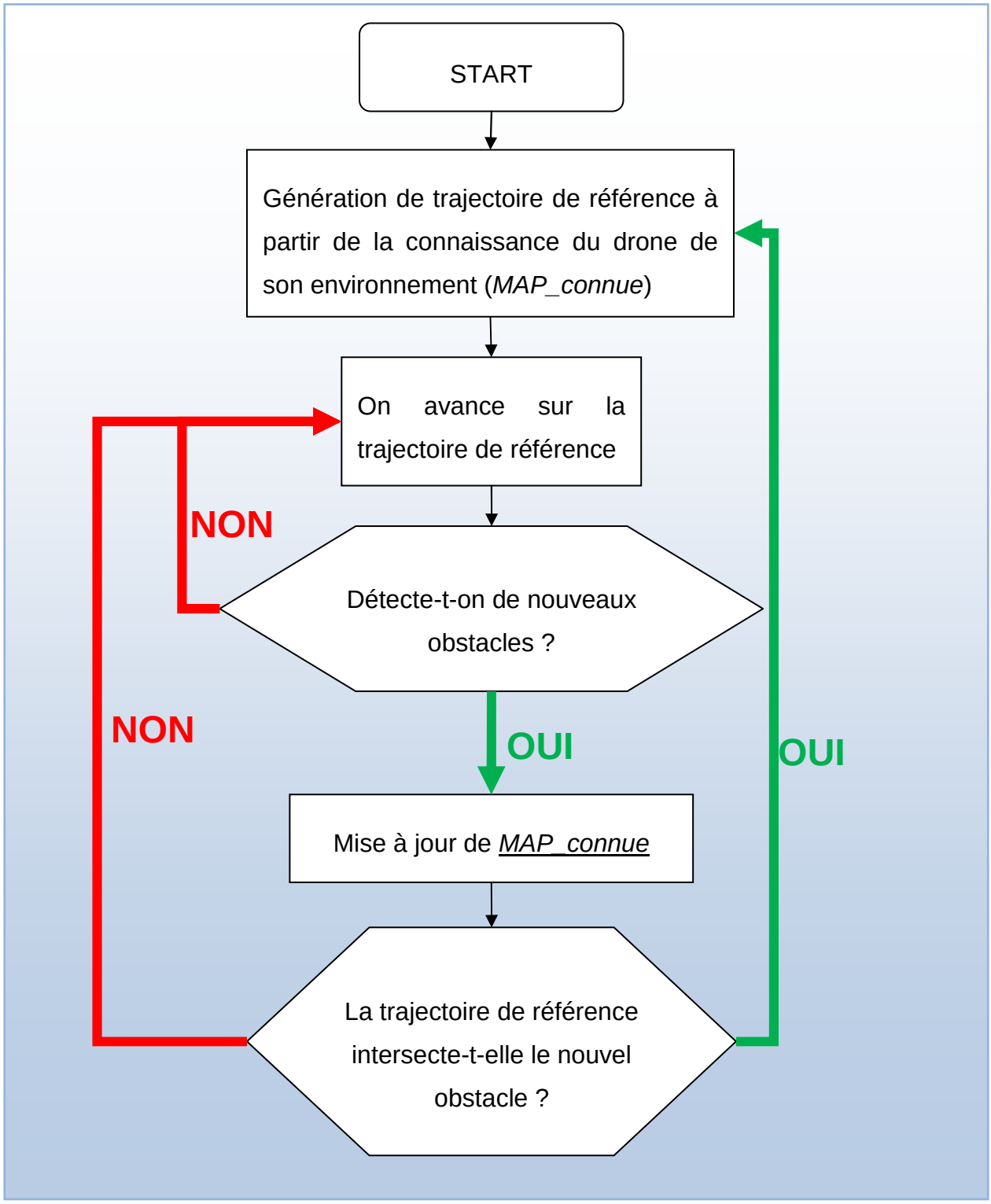
A l'instant t de son évolution, les capteurs du drone vont détecter un obstacle qui était jusqu'alors inconnu du véhicule (i.e. ne figurant pas dans MAP_connue). On va dans un premier temps mettre MAP_connue à jour de façon à ce que le drone garde en mémoire la position de ce nouvel obstacle. Ensuite, on va tester si la trajectoire générée intersecte cet obstacle ou pas.

- Si NON, on poursuit sur la trajectoire de référence précédemment définie.
- Si OUI, on va regénérer une trajectoire en réappliquant l'algorithme de génération de trajectoire à MAP_connue et en prenant comme point de départ la position actuelle du drone. Cette nouvelle trajectoire servira de nouvelle référence au drone.

On poursuit ainsi jusqu'au lieu de réalisation de la mission s_{goal}.

De plus, on souhaite que le drone rejoigne son objectif à une vitesse moyenne prédéfinie V_{moy}.

VIII-2 - Synoptique du fonctionnement :



Synoptique 1 : Principe de fonctionnement de la gestion des obstacles. ***NB*** *: On arrête la boucle lorsqu'on a fourni à la commande du drone le dernier point de la courbe de référence générée.*

VIII-3 - Modélisation :

On reprend la modélisation de l'espace et des obstacles déjà présentée en partie 1 (chapitre 3). La cartographie complète de l'environnement (cases franchissables et obstacles) est stockée sous forme d'un tenseur MAP_reelle. C'est à lui qu'on va se référer pour savoir s'il y a un nouvel obstacle dans le champ capteur. On désigne par MAP_connue la cartographie de l'environnement tel que connu par le drone. On reprend la notation :

- -1 : obstacle
- 2 : case franchissable
- 0 : s_{start}
- 1 : s_{goal}

De plus, on impose plusieurs contraintes pour la simulation:

- Le pas de temps qui va donner la période d'échantillonnage des signaux est fixé (pas_temps).
- Une vitesse moyenne de parcours V_moy au drone plutôt qu'un temps fini pour réaliser la mission. Cela permet, en cas de génération de trajectoires nombreuses, de ne pas augmenter la vitesse du drone à chacune d'elles.
- Une portée capteur ($portée$) qui va donner la distance maximale à laquelle on peut détecter un obstacle. Ainsi, $portée = 2$ va permettre au drone d'avoir des informations sur son environnement deux cases autours de sa position actuelle. On suppose que l'on a un capteur omnidirectionnel.

CHAPITRE 9

Exemple simple (pour illustration du principe)

On considère l'environnement suivant :

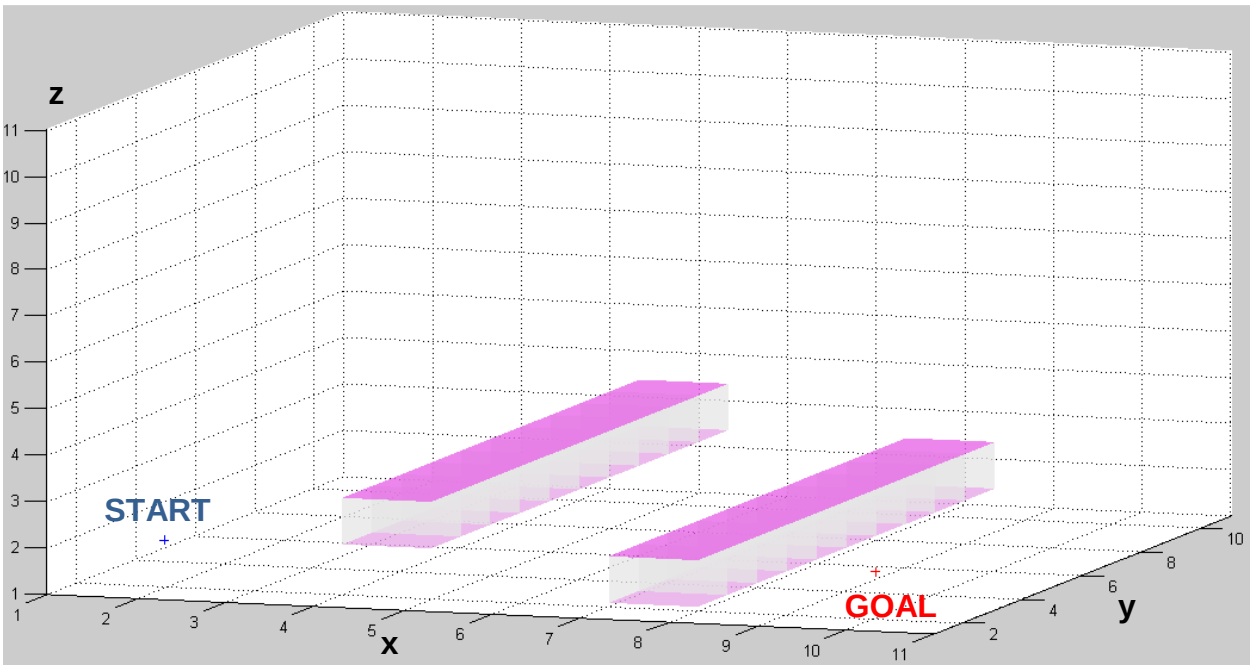


Figure 21 : environnement réel avec positions de départ et objectif.

On a ici l'environnement réel du drone. Il s'agit de deux poutres placées aux coordonnées (5, Y, 3) et (8, Y, 2). Le point de départ s_{start}de la mission est représenté par la croix bleue en (2, 2, 2) et l'objectif s_{goal}est placé en (10, 2, 2) et représenté par la croix rouge.

On considère que MAP_connue, l'ensemble des données connues a priori par le drone, est vide, c'est-à-dire que pour le drone, il n'y a pas d'obstacles entre la position de départ et l'objectif. La taille de capteur est fixée à deux, c'est-à-dire que pour chacune de ses positions, il est capable de percevoir les obstacles deux cases autour de lui, dans toutes les directions.

Le drone va effectuer une première génération de trajectoire au début de sa mission. Il va retourner une ligne droite comme trajectoire de référence puisqu'a priori il n'y a pas d'obstacle entre sa position et s_{goal}. Il va alors suivre cette trajectoire et va mettre à jour la carte qu'il possède de son environnement (MAP_connue) au fur et à

mesure qu'il perçoit de nouveaux obstacles. La première poutre n'est pas sur la trajectoire et ne donne pas lieu à une nouvelle génération de trajectoire. Par contre, lorsqu'il va voir la deuxième poutre, le drone va alors modifier son comportement pour éviter ce nouvel obstacle. C'est ce qu'on peut constater sur la figure 22, qui présente l'ensemble des trajectoires générées au cours de la mission.

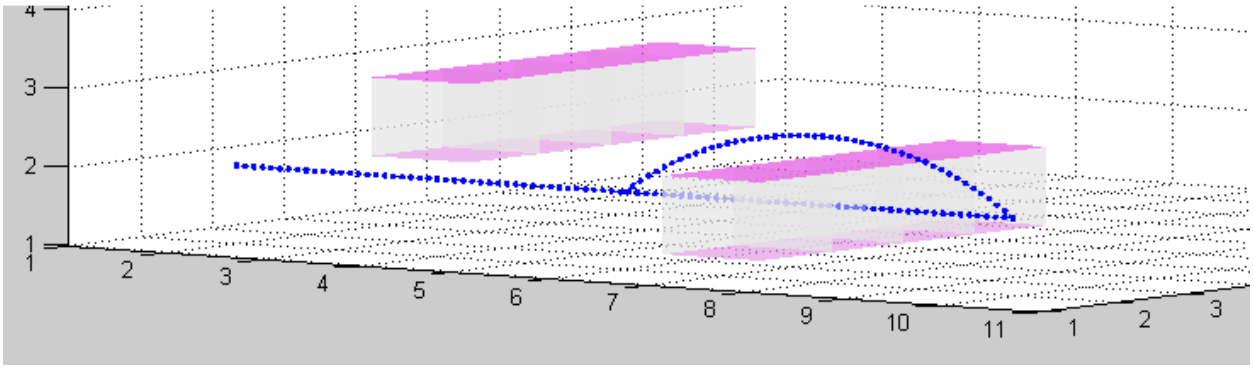


Figure 22 : exemple de génération de trajectoire(les obstacles affichés sont ceux connus par le drone, i.e. qui sont dans MAP_{connue}

La deuxième génération de trajectoire s'est faite en (6, 2,2).

On remarque aussi que le drone a pris en compte les obstacles qui se situaient à portée de capteurs puisque toutes les cellules des poutres présentées en figure 22 n'ont pas été découvertes.

Avec un point de départ du drone volontairement différent de celui qu'on va utiliser pour générer la première trajectoire de référence, on obtient le suivi présenté en figure 23 :

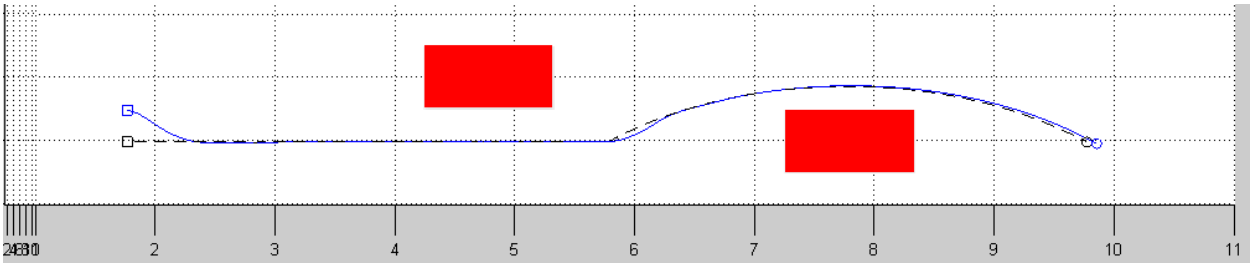


Figure 23 : suivi de trajectoire

On a une première phase d'accrochage à la trajectoire de référence puis un suivi avec une erreur très faible. Ensuite, la regénération de la trajectoire de référence (cf figure 23) va donner lieu au même phénomène : accrochage puis suivi.

CHAPITRE 10

Exemple concret : mission de pénétration dans un bâtiment via une fenêtre.

X.1 : Couplage avec connaissance totale de l'environnement

On considère maintenant un nouvel environnement :

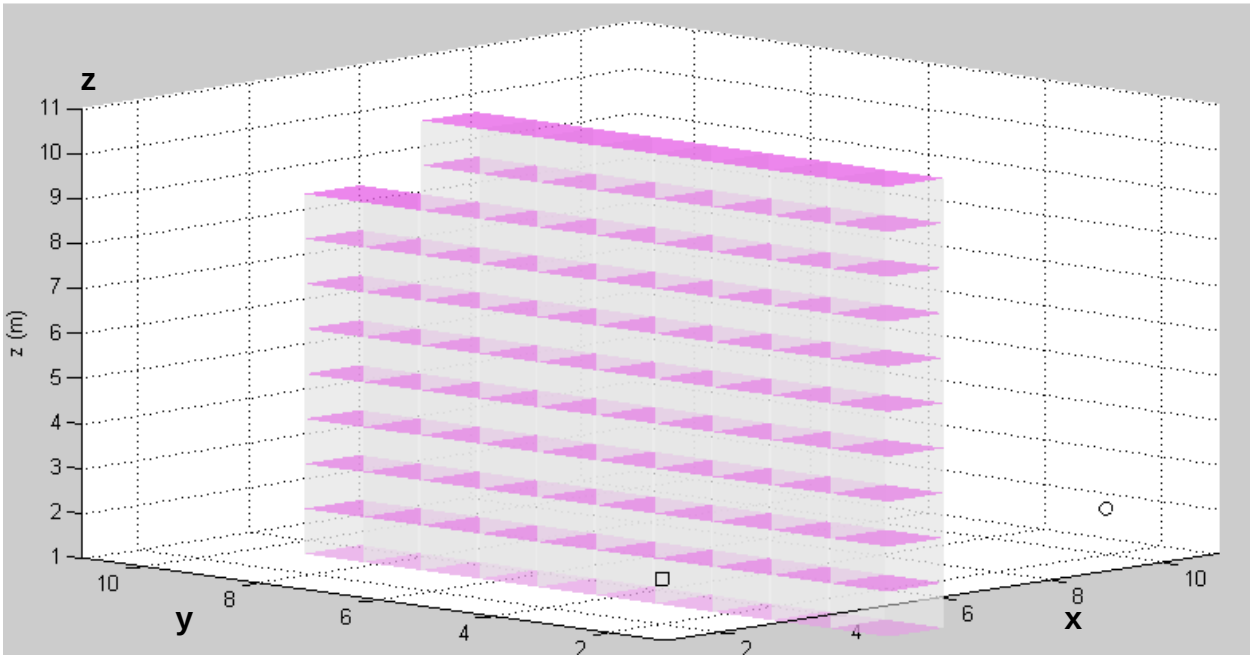


Figure 24 : pan de mur avec ouverture (fenêtre) en haut a gauche.

Ici, le but est de faire partir le drone du sol d'un coté du mur et de le faire rejoindre son objectif de l'autre côté, sachant que la seule possibilité pour lui est de passer par l'ouverture en haut à gauche.

Premièrement, supposons qu'on dispose d'un capteur de grande taille qui permette d'avoir une visibilité sur l'ensemble de l'obstacle. Par conséquent, MAP_connue reprend la cartographie réelle. Le drone va alors générer une première trajectoire de référence au début de sa mission, et ne devrait pas en re-générer pendant sa mission. Pour avoir un résultat cohérent, l'obstacle a été volontairement « gonflé », c'est-à-dire qu'on a considéré toutes les cases voisines de l'obstacle comme

infranchissables et qu'elles ont été ajoutées à MAP_connue. Cela revient à fixer une distance de sécurité pour éviter qu'un écart par rapport à la trajectoire de référence (due à une rafale par exemple) ne le fasse heurter le mur. Le résultat de la simulation est présenté en figure 25.

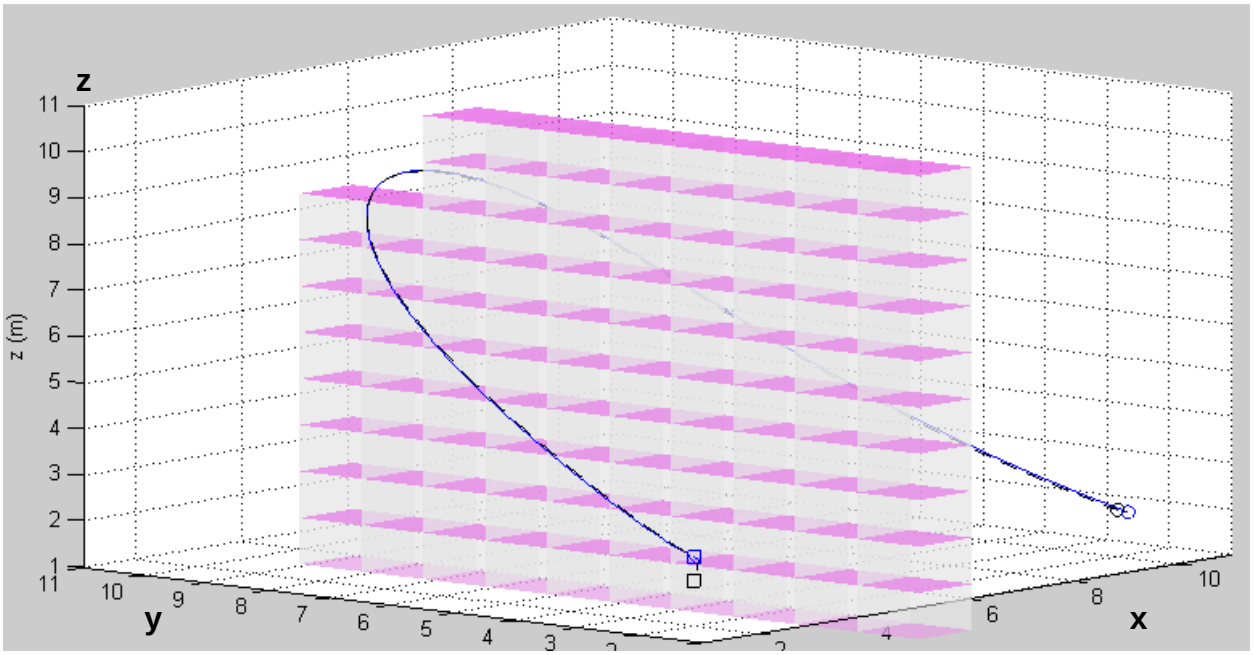


Figure 25: trajectoire de référence et suivi de cette trajectoire ($portée = 10$)

Le drone n'a généré qu'une trajectoire lors de sa mission et a bien réussi à rallier l'objectif fixé.

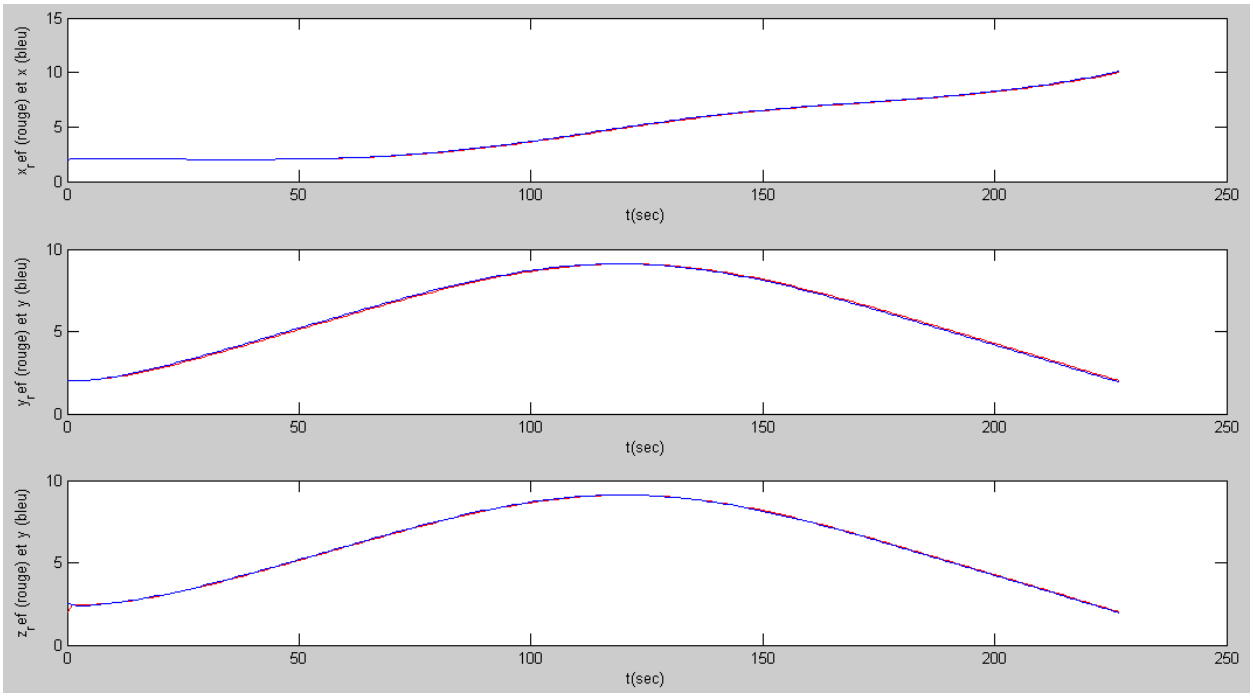


Figure 26: courbes de référence et réelle des positions en fonction du temps($portée = 10$)

On observe aussi que les performances en termes de suivi de trajectoire sont correctes. De plus, on a choisi $V_{moy} = 0.2\, m.s^{-1}$. Le résultat de la simulation nous donne $V_{moy} = 0.202\, m.s^{-1}$, ce qui est aussi tout à fait cohérent.

X.2 : Couplage sans connaissance a priori de l'environnement

i - Faible taille de capteur (portée = 2)

En pratique, le capteur peut avoir une taille plus limitée. Pour tester la validité de la méthode, on va se placer dans un cas assez défavorable : dans ce qui suit, on prend une **taille de capteur de 2**. Ainsi, la génération de trajectoire sera plus souvent sollicitée.

La figure 25 présente la trajectoire effectuée par le drone au cours de sa mission, avec une vitesse moyenne fixée à 0.2. La trajectoire bleue (continue) représente la trajectoire réalisée par le drone et la courbe noire (pointillée) correspond à la trajectoire de référence remise à jour au cours de la mission et de la découverte de l'environnement par le véhicule.

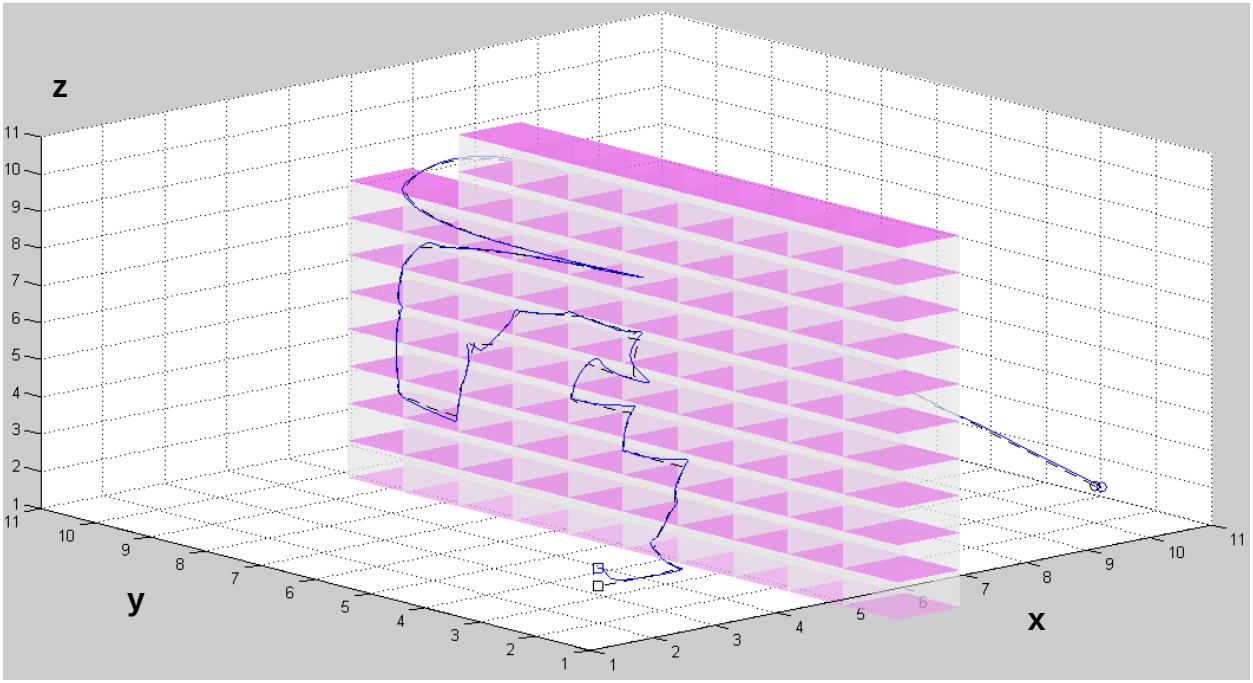


Figure 27 : trajectoire effective du drone suivie lors de sa mission (portée capteur = 2)

Les résultats sont conformes à ce qu'on attend du drone dans un tel environnement. Les générations de trajectoire sont nombreuses mais le drone parvient à rallier l'objectif de sa mission.

La figure 28 présente l'ensemble des trajectoires générées au cours de la mission. Il y a 46 courbes de référence générées et l'obstacle n'est pas entièrement découvert.

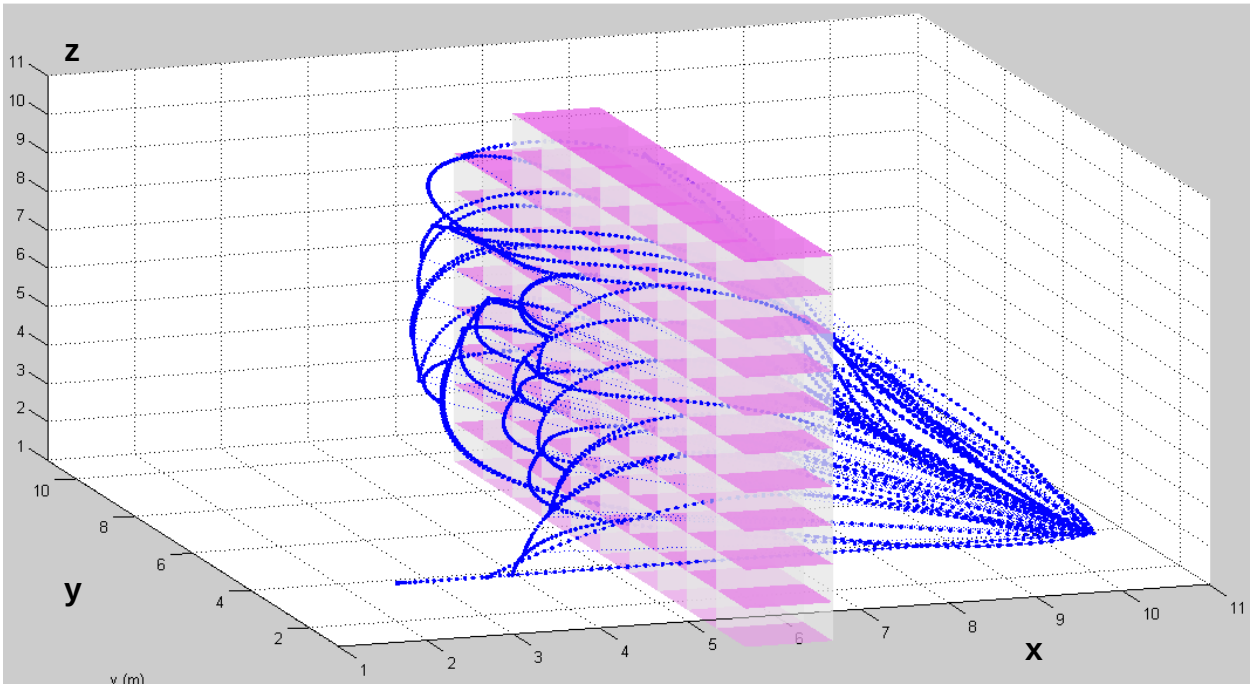


Figure 28: ensemble des trajectoires de références générées pendant la mission (portée capteurs = 2).

ii- Taille de capteur intermédiaire (portée = 5)

On va maintenant comparer les résultats de la simulation précédente avec ceux obtenus pour une portée capteur supérieure. Dans cet exemple, on prend $portée = 5$. La vitesse moyenne du drone est conservée ($V_{moy} = 0.2\ m.s^{-1}$). Les résultats de la simulation sont présentés en figures 29 et 30.

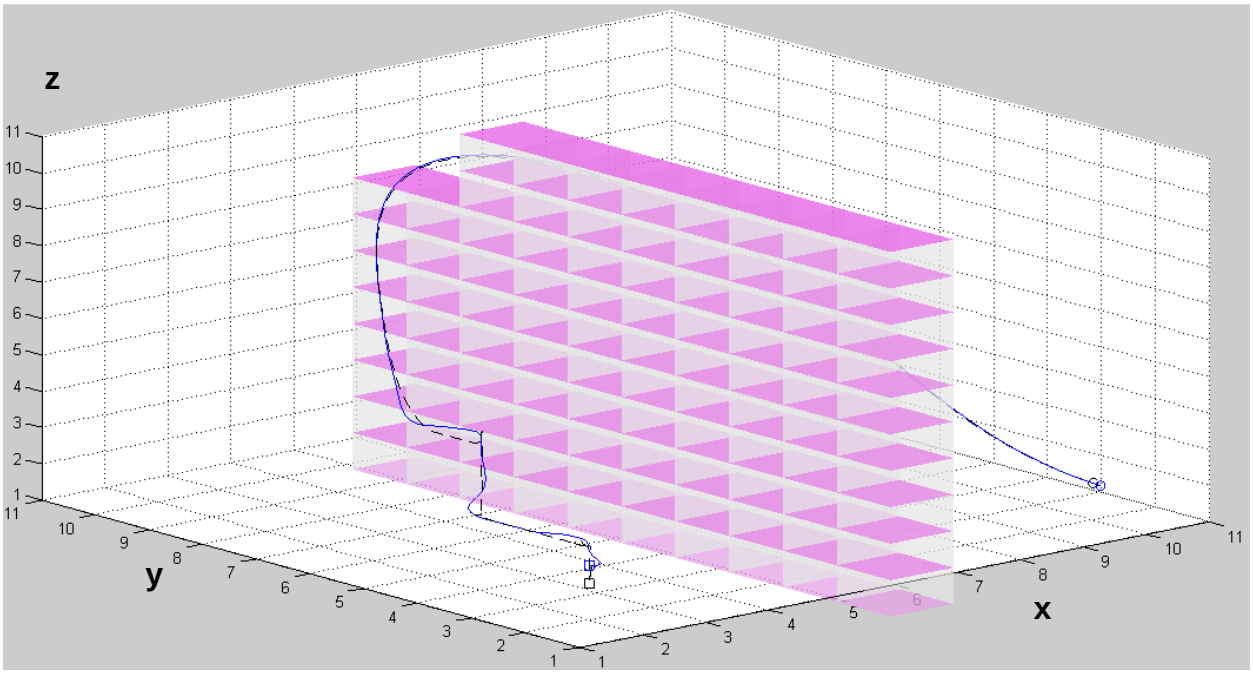


Figure 29 : trajectoire effective du drone suivie lors de sa mission

On remarque qu'une portée plus grande permet d'avoir une meilleure connaissance du milieu dans lequel le drone évolue et par conséquent de planifier avec plus de justesse la trajectoire de référence.

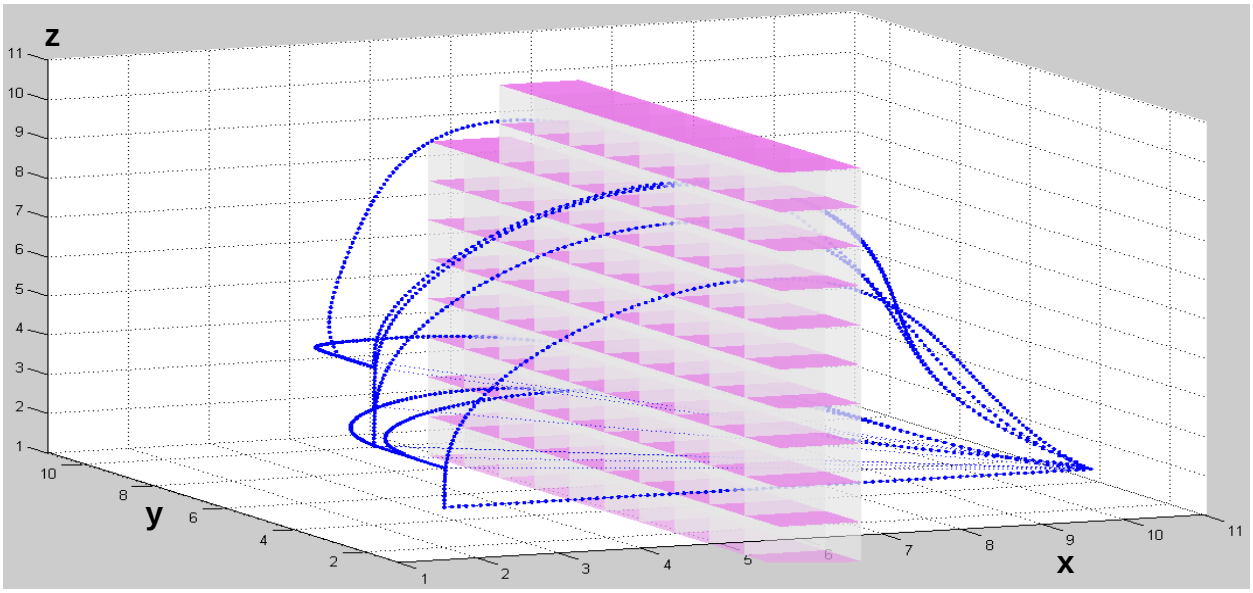


Figure 30 : ensemble des trajectoires de références générées pendant la mission

Contrairement à l'exemple précédent, l'obstacle a été entièrement découvert et le drone génère moins de trajectoires (seulement 11 dans notre cas), i.e. fait moins de mises à jour au cours de sa mission.

Conclusion de la partie 2

L'objectif de cette partie était de coupler les algorithmes de génération de trajectoires présentés en première partie avec les lois de commande synthétisées en deuxième partie.

Le principe retenu est le suivant. Le drone possède une cartographie a priori de l'environnement et s'en sert, au début de la mission, pour générer une première trajectoire de référence. Grâce aux lois présentées en deuxième partie, ce drone va suivre la référence. Des capteurs embarqués lui permettent d'obtenir de nouvelles informations sur son environnement et d'effectuer une mise à jour de sa connaissance du milieu, afin de regénérer si cela est nécessaire une trajectoire de référence.

Nous avons vu que la trajectoire générée et le nombre de regénération de trajectoires dépendait beaucoup de la portée des capteurs embarqués sur le drone. En effet, plus le capteur permet d'avoir des informations lointaines, plus le drone aura le temps de modifier son comportement de façon à éviter les obstacles.

Toutefois, même dans un cas très défavorable comme celui présenté dans X.2.i (portée capteur faible et très mauvaise connaissance de l'environnement au début de la mission)l'approche développée permet au drone miniature de réussir à trouver l'entrée du bâtiment et donc de réaliser sa mission.

CONCLUSIONS ET PERSPECTIVES

Les drones sont classés selon leur taille et leurs capacités, les générations les plus récentes tendant naturellement vers une miniaturisation et une augmentation des capacités (autonomie décisionnelle et durée de mission accrue). La première génération de drones correspond à un aéronef volant en espace ouvert. Encore au stade de recherche, la deuxième génération rassemble les drones (les drones miniatures) capables d'évoluer en zone urbaine. La troisième génération de drones (les micro-drones) concerne les drones capables d'évoluer à l'intérieur de bâtiments.

Ces drones doivent répondre à de nombreuses contraintes. Eviter les obstacles de manière automatique à l'aide de capteurs fait partie de ces contraintes, et a été abordé dans ce mémoire. Grace à ce qui a été présenté, le drone peut appréhender et modéliser son environnement pour s'y adapter et prendre des décisions.

Bien entendu, les algorithmes présentés sont perfectibles par l'utilisation de nouvelles méthodes de génération de trajectoire et de commande récemment développées. Certaines de ces méthodes feront l'objet d'une étude ultérieure dans le cadre de mon Projet de Fin d'Etudes.

Plusieurs axes ont déjà été proposés. Pour la génération de trajectoire, on s'intéressera à des algorithmes de type ARA* ou D* [1] qui permettent de réutiliser les données de la précédente génération de trajectoire afin d'économiser encore en temps de calcul, et donc de mieux prendre en compte les contraintes liées à la puissance embarquée dans le drone. Pour la partie guidage/pilotage du drone, pour le suivi de trajectoires, une méthode de synthèse des lois de commande par commande prédictive contractante (MPC) [2] sera étudiée.

D'autres thèmes seront aussi étudiés comme le raccordement des courbes de Bézier lors des générations de trajectoires, une approche plus réaliste de la perception de l'environnement par un capteur de type caméra, ou encore la particularisation du modèle utilisé pour une configuration de véhicule donnée (quadri rotor par exemple) via la mise en place des lois de commande reliant le modèle présenté à la dynamique des actionneurs.

Références

[1] : M. Likhachev, *Search-based Planning for Large Dynamic Environments*, phD thesis, Carnegie Mellon University, Pittsburg, 2005

[2]: S. Bertrand, *Commande de drone miniature à voilure tournante*, thèse ONERA-DPRS, I3S Université de Nice Sophia Antipolis, 2007

[3]: J-C Latombe. *Robot Motion Planning*, Klurver Academic, 1991

[4]: E.W.Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numer ische Mathematik 1, pp 267-271, 1959.

[5]: S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach,* Englewood Cliffs, NJ : Prentice Hall, 1995

[6]: P. Bézier, *Essai de definition numérique des courbes et des surfaces expérimentales*, 1977

[7] : B.Masson, *Trajectoire de Préguidage d'un Intercepteur*, Projet de Fin d'Etudes à l'ONERA-Chatillon, 2006

[8] : R.Mahony and T.Hamel, Robust Trajectory Tracking for a Scale Model Autonomous Helicopter, in International Journal of Robust and Nonlinear Control, 14, pp 1035-1059, 2004

[9] : J.Hauser S.Sastry and G.Meyer, Nonlinear Control Design for Slightly Nonminimum Phase System : Application to V/STOL Aircraft, in Automatica, 28 :4, pp 665-679, 1992

[10] : T.J.Koo and S.Sastry, Output Tracking Control Design of a Helicopter Model Based on Approximate Linearization, in Proceeding of the 37th IEEE Conference on Decision and Control, Tampa, Florida, USA, 1998.

[11] : W.E.Dixon, E.Zergeroglu, D.M.Dawson and M.W.Hannan, Global Adaptative Partial State Feedback Tracking Control of Rigid-Link Flexible-Joints Robots, in Robotica, 18, pp 325-336, 2000

[12] :H.K.Khalil Nonlinear Systems, 1rst Edition, Macmillan, 1992.

[13] : P.V.Kokotovic, H.K.Khalil and J.O'Reily, Singular Perturbation Methods in Control : Analysis and Design, Academon Press, 1986

[14] : A.Saberi and H.Khalil, Quadratic-Type Lyapunov Function for Singularity Pertubated Systems, in IEEE Transactions on Automatic Control, 29 :6, pp 542-550, 1984

[15] : T.Hamel and R.Mahony, Image Based Visual Control for a Class of Aerial Robotic Systems, Automatica 2007, doi :10.1016/j.automatica.2007.03.030.