Très bon travail,
merci !
20-

MTH8408: Rapport intermédiaire

Remis pour le 26 mars, 2017

Professeur Dominique Orban

André Phu-Van Nguyen, 1525972

Rappel de la problématique

Ce projet consiste en l'étude et l'implémentation de l'article *Minimum Snap Trajectory Generation and Control for Quadrotors* par Daniel Mellinger et Vijay Kumar [3] qui propose un contrôleur et un générateur de trajectoires pour un véhicule aérien multirotor. Plus précisément, nous faisont une implémentation seulement de la partie génération de trajectoire par une méthode d'optimisation numérique.

Lors du suivi d'une trajectoire, une solution triviale est souvent utilisée qui consiste en l'interpolation en ligne droite entre chaque point de cheminement ou *waypoint* (Mellinger utilise plutôt l'apellation *keyframe*). Ceci est inneficace car la courbure infinie à chaque waypoint oblige le quadricoptère à s'arrêter avant de passer au prochain waypoint. Mellinger propose donc de modéliser une trajectoire optimale par un polynôme défini par parties entièrement lisse à travers les différents waypoints tout en satisfaisant des contraintes sur les vitesses et accélérations possibles du véhicule. Ce problème est résolu en le reécrivant en problème d'optimisation quadratique.

Tout dabord Mellinger démontre que la dynamique d'un quadricoptère a la propriété dêtre différentiellement plat. C'est-à-dire que les états et les entrées peuvent être exprimées par les sorties du système et leurs dérivées. Nous avons donc le vecteur de sorties plates

$$\sigma = [x, y, z, \psi]^T$$

où $r = [x, y, z]^T$ est la position du centre de masse dans le système de coordonnées du monde et ψ l'angle de lacet. Rappelons nous que dans un repère main droite centré sur un corps rigide, l'axe x pointe vers l'avant, y vers la gauche et z vers le haut. Les angles de rotation autour de ces axes sont le roulis (*roll*), tangage (*pitch*) et lacet (*yaw*) respectivement. À fin de garder le projet simple, nous ne considérons pas les angles de roulis et de tangage dans le problème mais nous savons qu'il est possible de le faire pour exécuter des manoeuvres accrobatiques tel que voler à travers une fenêtre inclinée.

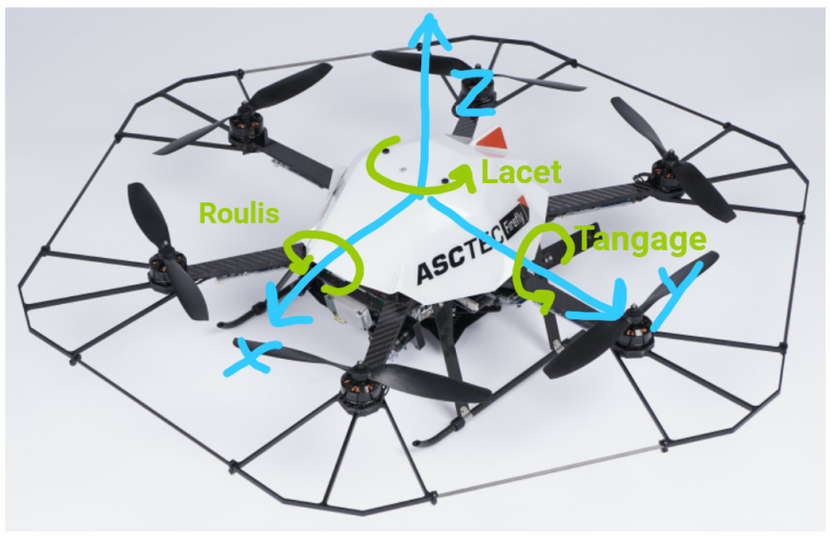


Figure 1: Systèmes de coordonnées d'un hexacoptère. L'angle de lacet est la rotation du véhicule autour de son axe Z, c'est-à-dire quand le véhicule tourne sur lui même.

Une trajectoire est définie comme était une courbe lisse dans l'espace des sorties plates:

$$\sigma(t) : [t_0, t_m] \to \mathbb{R}^3 \times SO(2)$$

où t_0 et t_m sont les temps de début et de fin de la trajectoire, m correspond au nombre d'intervales de temps entre chaque waypoint et $SO(2)$ est le groupe spécial orthogonal. En pratique, une trajectoire est plutôt

décrite par un polynôme défini par parties:

$$\sigma_T(t) = \begin{cases} \sum_{i=0}^{n} \sigma_{Ti1} t^i & t_0 \leq t < t_1 \\ \sum_{i=0}^{n} \sigma_{Ti2} t^i & t_1 \leq t < t_2 \\ \dots & \\ \sum_{i=0}^{n} \sigma_{Tim} t^i & t_{m-1} \leq t < t_m \end{cases} \tag{1}$$

où n est l'ordre du polynôme et m est encore le nombre d'intervales de temps.

Étant donné que l'on veut optimiser la dérivée d'ordre $k_r = 4$ (le *snap*) de la position au carré et la dérivée d'ordre $k_\psi = 2$ (accélération angulaire) de langle de lacet ψ au carré, nous avons le problème d'optimisation:

$$\min \int_{t_0}^{t_m} \mu_r \left\| \frac{d^4 \boldsymbol{r}_T}{dt^4} \right\|^2 + \mu_\psi \ddot{\psi}_T^2 dt \tag{2}$$

$$\begin{aligned} \text{sous contraintes} \quad & \boldsymbol{r}_T(t_i) = \boldsymbol{r}_i && i = 0, \dots, m \\ & \psi_T(ti) = \psi_i && i = 0, \dots, m \\ & \tfrac{d^p x_T}{dt^p}|_{t=t_j} = 0 \text{ ou libre}, && j = 0, m; p = 1, 2, 3, 4 \\ & \tfrac{d^p y_T}{dt^p}|_{t=t_j} = 0 \text{ ou libre}, && j = 0, m; p = 1, 2, 3, 4 \\ & \tfrac{d^p z_T}{dt^p}|_{t=t_j} = 0 \text{ ou libre}, && j = 0, m; p = 1, 2, 3, 4 \\ & \tfrac{d^p \psi_T}{dt^p}|_{t=t_j} = 0 \text{ ou libre}, && j = 0, m; p = 1, 2 \end{aligned}$$

où $\boldsymbol{r}_T = [x_T, y_T, z_T]^T$ et $r_i = [x_i, y_i, z_i]$.

Changements à la problématique

Puisque les implémentations initiales ne fonctionnaient pas (la trajectoire semblait diverger et ne respectait pas les contraintes imposées) nous avons fait quelques changements à la problématique pour simplifier les choses pour au moins obtenir une solution fonctionnelle avant de l'améliorer. Tout d'abord nous avons retiré pour l'instant l'angle de lacet du problème. Ensuite nous avons remarqué que Mellinger indique dans son article que les états sont découplées dans leurs fonction de coût et leurs contraintes alors en pratique nous pouvons séparer le problème en plusieurs sous problèmes d'optimisation. Au final la génération de trajectoire se fait donc par l'optimisation de trois problèmes quadratiques pour les dimensions x, y et z. Par exemple pour la trajectoire en x nous avons

$$\min \int_{t_0}^{t_m} \mu_r \left\| \frac{d^4 \boldsymbol{x}_T}{dt^4} \right\|^2 dt \tag{3}$$

$$\begin{aligned} \text{sous contraintes} \quad & \boldsymbol{x}_T(t_i) = \boldsymbol{x}_i && i = 0, \ldots, m \\ & \tfrac{d^p x_T}{dt^p}|_{t=t_j} = 0 \text{ ou libre}, && j = 0, m; p = 1, 2, 3, 4 \end{aligned}$$

où $\boldsymbol{x}_T$ est le vecteur contenant les coefficients des polynômes représentant la trajectoire dans la dimension x.

Structure du problème d'optimisation

Puisque l'article de Mellinger comporte très peu de détails et ne discute de la manière de poser le problème que vaguement, nous avons été obligé de faire le développement des équations à la main et au final nous avons aussi été obligé de consulter d'autres articles (Bry et Richter) citant celui de Mellinger pour mieux comprendre comment faire. Le dévelppement mathématique suivant est donc un mélange de notre propre travail et de ce qui est écrit dans [4, 1].

Considérons le problème en une dimension p, où $p = x$, y ou z. Nous pouvons reformuler le problème en tant que problème d'optimisation quadratique en réécrivant les coefficients des polynômes en un vecteur c de dimension $4m \times 1$, où m est le nombre de waypoints en excluant les conditions initiales. Nous obtenons la forme standard:

$$\begin{aligned} \min \quad & c^T H c + f^T c \\ \text{s. c.} \quad & Ac = b \end{aligned} \tag{4}$$

Avant de construire la matrice H nous devons développer les équations des polynômes représentant un segment de trajectoire i.e. la trajectoire entre deux waypoints. En réglant l'ordre du polynôme à $n = 6$ nous avons pour un axe de déplacement p en 3D où $p = x$, y ou z:

$$\begin{aligned} \boldsymbol{p} &= c_6 t^6 + c_5 t^5 + c_4 t^4 + c_3 t^3 + c_2 t^2 + c_1 t + c_0 \\ \frac{d\boldsymbol{p}}{dt} &= 6c_6 t^5 + 5c_5 t^4 + 4c_4 t^3 + 3c_3 t^2 + 2c_2 t + c_1 + 0 \\ \frac{d^2\boldsymbol{p}}{dt^2} &= (6 \cdot 5)c_6 t^4 + (5 \cdot 4)c_5 t^3 + (4 \cdot 3)c_4 t^2 + (3 \cdot 2)c_3 t + 2c_2 + 0 + 0 \\ \frac{d^3\boldsymbol{p}}{dt^3} &= (6 \cdot 5 \cdot 4)c_6 t^3 + (5 \cdot 4 \cdot 3)c_5 t^2 + (4 \cdot 3 \cdot 2)c_4 t + (3 \cdot 2)c_3 + 0 + 0 + 0 \\ \frac{d^4\boldsymbol{p}}{dt^4} &= (6 \cdot 5 \cdot 4 \cdot 3)c_6 t^2 + (5 \cdot 4 \cdot 3 \cdot 2)c_5 t + (4 \cdot 3 \cdot 2)c_4 + 0 + 0 + 0 + 0 \end{aligned}$$

Prendre la norme euclidienne au carré de la position est équivalent à prendre le carré de chaque polynôme. Donc pour un polynôme p représentant un axe x, y ou z nous avons:

$$\begin{aligned} \left\| \frac{d^4\boldsymbol{p}}{dt^4} \right\|^2 &= \Big((6 \cdot 5 \cdot 4 \cdot 3)c_6 t^2 + (5 \cdot 4 \cdot 3 \cdot 2)c_5 t + (4 \cdot 3 \cdot 2)c_4 \Big)^2 \\ &= (6 \cdot 5 \cdot 4 \cdot 3)^2 c_6^2 t^4 + (6 \cdot 5 \cdot 4 \cdot 3)(5 \cdot 4 \cdot 3 \cdot 2)c_6 c_5 t^3 + (6 \cdot 5 \cdot 4 \cdot 3)(4 \cdot 3 \cdot 2)c_6 c_4 t^2 \\ &+ (5 \cdot 4 \cdot 3 \cdot 2)^2 c_5^2 t^2 + (6 \cdot 5 \cdot 4 \cdot 3)(5 \cdot 4 \cdot 3 \cdot 2)c_6 c_5 t^3 + (5 \cdot 4 \cdot 3 \cdot 2)(4 \cdot 3 \cdot 2)c_5 c_4 t \\ &+ (4 \cdot 3 \cdot 2)^2 c_4^2 + (6 \cdot 5 \cdot 4 \cdot 3)(4 \cdot 3 \cdot 2)c_6 c_4 t^2 + (5 \cdot 4 \cdot 3 \cdot 2)(4 \cdot 3 \cdot 2)c_5 c_4 t \\ &= (6 \cdot 5 \cdot 4 \cdot 3)^2 c_6^2 t^4 + 2(6 \cdot 5 \cdot 4 \cdot 3)(5 \cdot 4 \cdot 3 \cdot 2)c_6 c_5 t^3 + 2(6 \cdot 5 \cdot 4 \cdot 3)(4 \cdot 3 \cdot 2)c_6 c_4 t^2 \\ &+ (5 \cdot 4 \cdot 3 \cdot 2)^2 c_5^2 t^2 + 2(5 \cdot 4 \cdot 3 \cdot 2)(4 \cdot 3 \cdot 2)c_5 c_4 t \\ &+ (4 \cdot 3 \cdot 2)^2 c_4^2 \end{aligned} \tag{5}$$

Ensuite, pour un waypoint j nous avons le temps de départ t_j et le temps d'arrivée au prochain waypoint

t_{j+1}, nous pouvons écrire le résultat de l'intégrale:

$$\begin{aligned}\int_{t_j}^{t_{j+1}} \left\| \frac{d^4\boldsymbol{p}}{dt^4} \right\|^2 dt = \int_{t_j}^{t_{j+1}} & (6\cdot5\cdot4\cdot3)^2 c_6^2 t^4 + 2(6\cdot5\cdot4\cdot3)(5\cdot4\cdot3\cdot2)c_6c_5t^3 \\ & + 2(6\cdot5\cdot4\cdot3)(4\cdot3\cdot2)c_6c_4t^2 + (5\cdot4\cdot3\cdot2)^2c_5^2t^2 + 2(5\cdot4\cdot3\cdot2)(4\cdot3\cdot2)c_5c_4t \\ & + (4\cdot3\cdot2)^2c_4^2\ dt \\ & = 360^2c_6^2\frac{1}{5}t^5\Big|_{t_j}^{t_{j+1}} + 2\cdot360\cdot120c_6c_5\frac{1}{4}t^4\Big|_{t_j}^{t_{j+1}} + 2\cdot360\cdot24c_6c_4\frac{1}{3}t^3\Big|_{t_j}^{t_{j+1}} \\ & + 120^2c_5^2\frac{1}{3}t^3\Big|_{t_j}^{t_{j+1}} + 2\cdot120\cdot24c_5c_4\frac{1}{2}t^2\Big|_{t_j}^{t_{j+1}} t_{j+1} \\ & + 24^2c_4^2t\Big|_{t_j}^{t_{j+1}}\end{aligned} \tag{6}$$

ok

Avec le résultat en (6) on peut maintenant poser une partie de la matrice H de (4).

Tel qu'indiqué précédemment, le vecteur c contient tous les coefficients de tous les polynômes de chaque segment de la trajectoire et de chaque degré de liberté. La partie de c correspondant à un axe p est donc $[c_6,\ c_5,\ c_4,\ c_3,\ c_2,\ c_1,\ c_0]^T$. Par conséquent, nous pouvons déduire la matrice H_{pj} correspondante pour un waypoint j

$$H_{pj} = \begin{bmatrix} 360^2\frac{1}{5}t^5\Big|_{t_j}^{t_{j+1}} & 2\cdot360\cdot120\frac{1}{4}t^4\Big|_{t_j}^{t_{j+1}} & 2\cdot360\cdot24\frac{1}{3}t^3\Big|_{t_j}^{t_{j+1}} & 0&0&0&0 \\ 2\cdot360\cdot120\frac{1}{4}t^4\Big|_{t_j}^{t_{j+1}} & 120^2\frac{1}{3}t^3\Big|_{t_j}^{t_{j+1}} & 2\cdot120\cdot24\frac{1}{2}t^2\Big|_{t_j}^{t_{j+1}} & 0&0&0&0 \\ 2\cdot360\cdot24\frac{1}{3}t^3\Big|_{t_j}^{t_{j+1}} & 2\cdot120\cdot24\frac{1}{2}t^2\Big|_{t_j}^{t_{j+1}} & 24^2t\Big|_{t_j}^{t_{j+1}} & 0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \end{bmatrix} \tag{7}$$

Je ne sais pas si ce bloc de zéros peut causer des difficultés mais vous pourriez essayer de le remplacer par $\begin{bmatrix}\delta&\delta&\delta\end{bmatrix}$ où $\delta\approx10^{-8}$ ou 10^{-7}.

Bien que Mellinger ne le présente pas dans son article, Richter et Bry [4, 1] indiquent que ce processus peut être automatisé par la formule suivante:

$$H_{pj} = \begin{cases} 2\Big(\prod_{m=0}^{r-1}(i-m)(l-m)\Big)\frac{\tau^{i+l-2r+1}}{i+l-2r+1} : & i \geq r \text{ ou } l \geq r \\ 0 : & \text{autrement} \end{cases} \tag{8}$$

où i et l sont les index de rangée et de colonne et τ est la durée du segment de trajectoire.

✓

Structure de la matrice H finale

Une fois le développement précédent fait, nous pouvons concaténer en diagonale les matrices H_{pj} pour chaque waypoint jusqu'à $j = m$, pour chaque waypoint jusqu'au dernier waypoint $j = m$ pour former H de (4).

$$H_x = \begin{bmatrix} H_{x1} & & & \\ & H_{x2} & & \\ & & \ddots & \\ & & & H_{xm} \end{bmatrix} \tag{9}$$

De plus, par le développement précédent en (6), nous pouvons voir que le terme linéaire $f^T c$ de (4) est nul.

Structure des matrices de contraintes A et b

Dans ce problème nous utilisons les matrices de contraintes d'égalité de deux façons. Premièrement pour imposer la valeur d'une certaine dérivée à un certain moment et deuxièmement pour imposer la continuité entre deux segments de trajectoire.

Dans notre code, nous avons appelé ceci les contraintes de départ et d'arrivée et les contraintes de continuité. Encore une fois, nous suivons la formulation de Bry [1](car Mellinger donne peu de détails dans son article) où pour un segment de trajectoire allant du temps 0 au temps τ nous pouvons contruire A et b de telle manière que

$$A = \begin{bmatrix} A_0 \\ A_\tau \end{bmatrix}, \quad b = \begin{bmatrix} b_0 \\ b_\tau \end{bmatrix} \tag{10}$$

$$A_{0_{rn}} = \begin{cases} \prod_{m=0}^{r-1}(r-m): & r = n \\ 0: & \text{autrement} \end{cases} \tag{11}$$

Est-ce que cela veut dire qu'il y a des lignes entières de zéros dans A?

$$b_{0_r} = P^{(r)}(0) \tag{12}$$

$$A_{\tau_{rn}} = \begin{cases} \left(\prod_{m=0}^{r-1}(n-m)\right)\tau^{n-r}: & n \geq r \\ 0: & \text{autrement} \end{cases} \tag{13}$$

$$b_{\tau_r} = P^{(r)}(\tau) \tag{14}$$

Notons que A_0 et A_τ ont $r+1$ rangées (une par dérivée incluant la dérivée 0) et n colonnes (une par coefficient du polynôme). La notation $P^{(r)}(0)$ indique le polynôme dérivé au degré r et évalué au temps 0. Si jamais la dérivée est sans contrainte, par exemple si nous ne boulons pas imposer la valeur de l'accélération à un certain waypoint, il suffit simplement d'ommettre la ligne correspondante de la matrice A.

En appliquant les équations précédentes, nous imposons à chaque waypoint une valeur fixe pour la position, la vitesse, le *jerk* et le *snap*. Tant à l'arrivée qu'au départ de celui ci. Par contre, si jamais l'utilisateur n'impose pas de contraintes sur ces dérivées, il faut tout de même ajouter des contraintes de continuité pour imposer que la valeur calculée (par le processus d'optimisation) pour cette dérivée soit continue au waypoint. Pour ce faire, il suffit de mettre égal les deux polynômes représentant les segments de part et d'autre d'un waypoint. Soit le polynôme dérivé r fois allant au waypoint j, $P_j^{(r)}$ et le prochain polynôme partant du waypoint j, $P_{j+1}^{(r)}$ nous avons la contrainte

$$P_j^{(r)} = P_{j+1}^{(r)}$$
$$-P_{j+1}^{(r)} + P_j^{(r)} = 0$$

Par inspection, nous pouvons recouvrir la forme de A finale proposée par Bry [1]. Prenons la notation A_0^j la matrice de contraintes pour un début de trajectoire au waypoint j et de même pour A_τ^j la matrice de

contraintes pour une fin de trajectoire au waypoint j. Nous avons au final:

$$A = \begin{bmatrix} A_0^0 & 0 & 0 & 0 & \dots & 0 & 0 \\ -A_\tau^0 & A_0^1 & 0 & 0 & \dots & 0 & 0 \\ 0 & -A_\tau^1 & A_0^2 & 0 & \dots & 0 & 0 \\ 0 & 0 & -A_\tau^2 & A_0^3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \ddots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & -A_\tau^{m-1} & A_0^m \\ 0 & 0 & 0 & 0 & 0 & 0 & A_\tau^m \end{bmatrix} \tag{15}$$

$$b = \begin{cases} \text{Valeur de la dérivée du polynôme } P \text{ imposée :} & \text{Si imposé} \\ 0 : & \text{autrement} \end{cases} \tag{16}$$

Structure du deuxième problème d'optimisation

Mellinger montre que si jamais le temps d'arrivé à chaque segment n'importe pas, il est possible dans un deuxième temps de résoudre un deuxième problème d'optimisation à fin de trouver la meilleur répartition de temps possible entre chaque segment de la trajectoire.

Au lieu de considérer les temps d'arrivés t_i à un waypoint i, nous pouvons plutôt considérer le temps de durée d'un segment de trajectoire $T_i = t_i - t_{i-1}$ et $T = [T_0, T_1, ..., T_i]$. Soit $f(T) = f_x(T) + f_y(T) + f_z(T)$, la somme des valeurs de fonction de coût après la résolution du problème (3) dans chaque dimension. Nous résolvons maintenant le problème

$$\min \quad f(T) \tag{17}$$

$$\begin{aligned} \text{s. c.} \quad & \textstyle\sum T_i = t_m \quad && i = 1, \dots, m \\ & T_i \geq 0 && i = 1, \dots, m \end{aligned}$$

où $T_i = t_i - t_{i-1}$ sont les temps alloués à chaque segment de la trajectoire. Il est relativement trivial de reécrire le problème sous la forme "standard" (il y a standard et standard...)

$$\begin{aligned} \min \quad & f(x) \\ \text{s. c.} \quad & Ax = b \\ & \text{lb} \leq x \end{aligned}$$

A est de taille $1 \times m$ avec des 1 partout $b = t_m$ le temps d'arrivé au dernier waypoint et lb$= 0$.

Par contre, ce qui est un peu moins trivial est la façon de fournir le gradient de $f(T)$ à la fonction d'optimisation. Mellinger le calcule numériquement par l'équation

$$\nabla_{g_i} f = \frac{f(T + hg_i) - f(T)}{h} \tag{18}$$

où h est une "petite" valeur et le vecteur colonne[1]

$$g_i = \begin{cases} 1 : & \text{à la position } i \\ \frac{-1}{m-1} : & \text{autrement} \end{cases} \tag{19}$$

f est-elle une fonction Matlab ? Si oui, ce serait sans doute beaucoup mieux d'évaluer ∇f par différentiation automatique.

[1] Mellinger utilse en fait $\frac{-1}{m-2}$ car son m inclut aussi les conditions initiales.

par exemple, dans le cas où $m = 3$ nous aurions

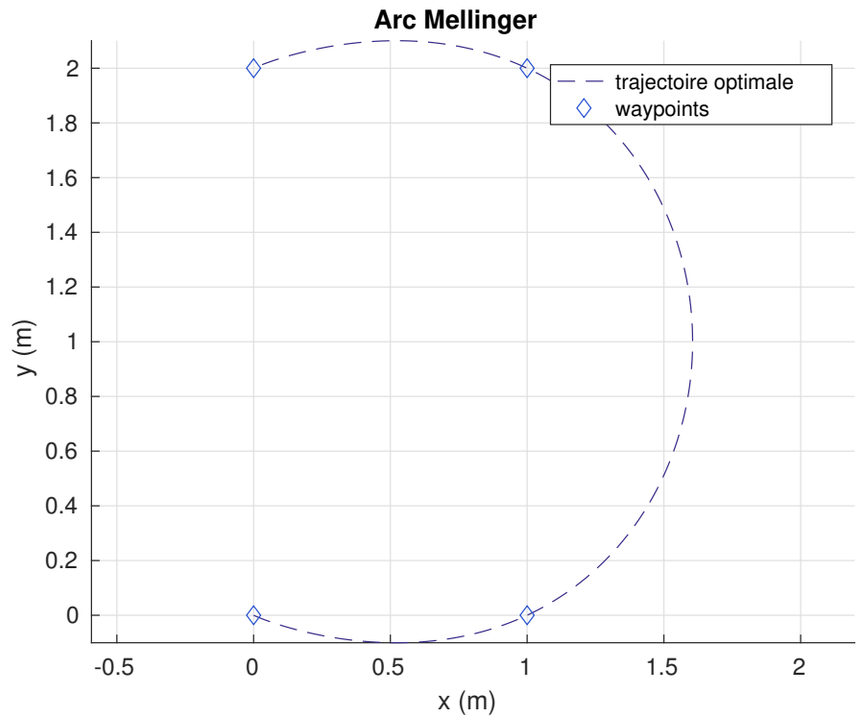
$$g_1 = \begin{bmatrix} 1 \\ -\frac{1}{2} \\ -\frac{1}{2} \end{bmatrix}, \; g_2 = \begin{bmatrix} -\frac{1}{2} \\ 1 \\ -\frac{1}{2} \end{bmatrix} \text{ et } g_3 = \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ 1 \end{bmatrix}$$

et le gradient final serait

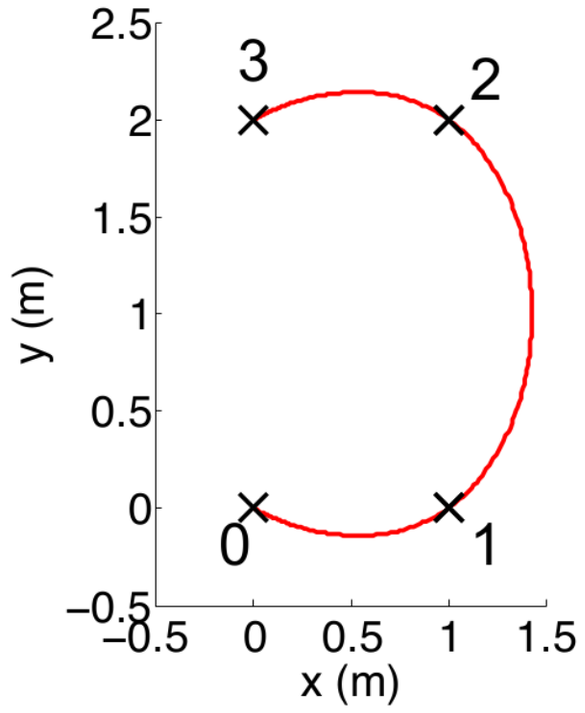
$$\nabla f = \begin{bmatrix} \nabla_{g_1} f \\ \nabla_{g_2} f \\ \nabla_{g_3} f \end{bmatrix} = \begin{bmatrix} \frac{f(T+hg_1)-f(T)}{h} \\ \frac{f(T+hg_2)-f(T)}{h} \\ \frac{f(T+hg_3)-f(T)}{h} \end{bmatrix}$$

Implémentation

L'implémentation s'est fait au moyen de Matlab et des solveurs `quadprog` pour la génération de trajectoire et `fmincon` pour l'optimisation des temps de trajectoire. Pour commencer nous avons reproduit les résultats de Mellinger en faisait une trajectoire simple à travers les points $(0, 0)$, $(1, 0)$, $(1, 2)$ et $(0, 2)$ avec une allocation de temps égale à chaque segment de trajectoire et aucune contrainte sur les derivées, outre celles de continuité.



(a) Notre résultat

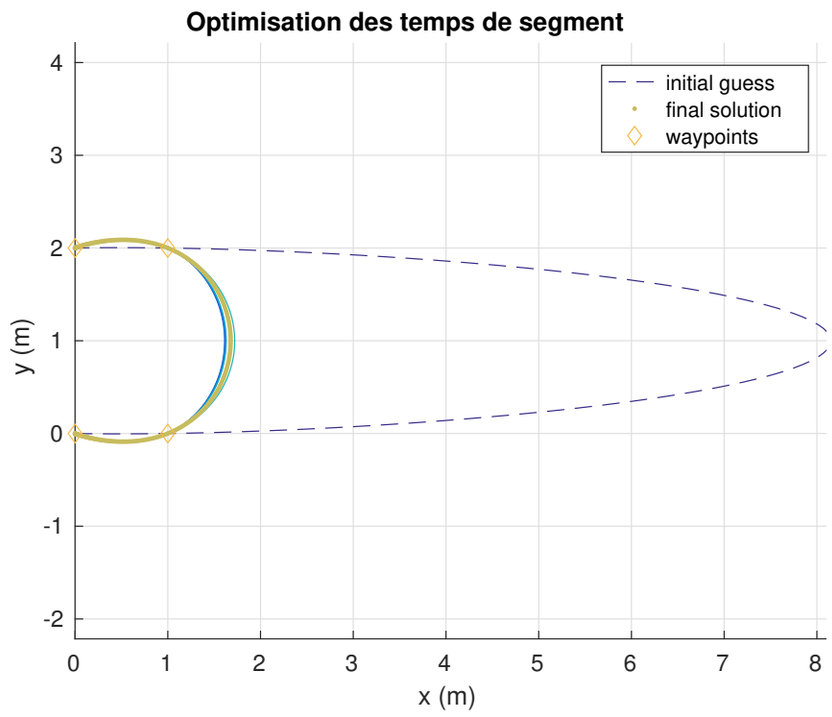


(b) Résultat de Mellinger [3]

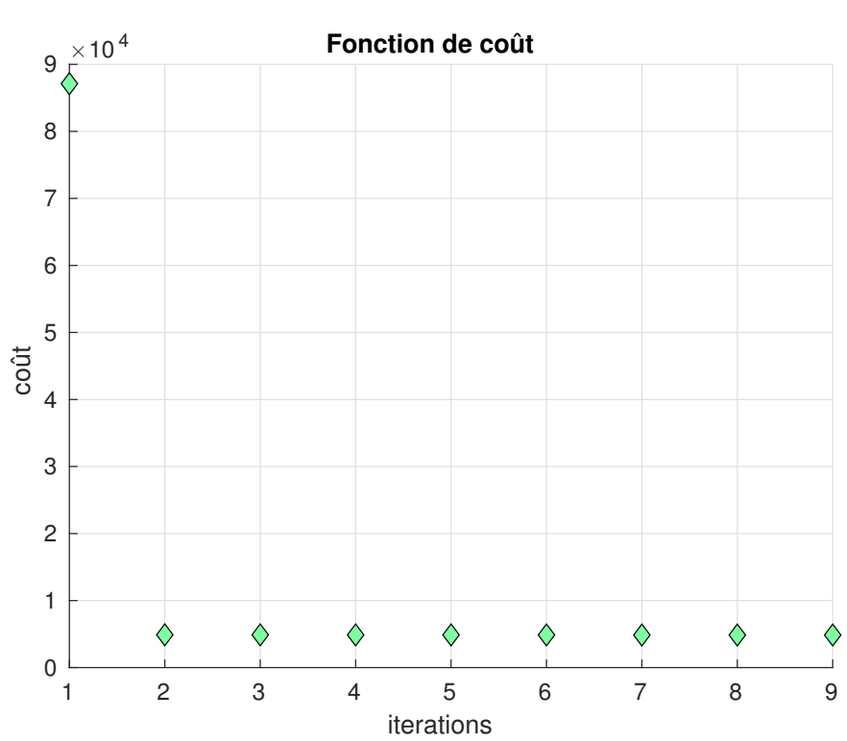
Figure 2: Comparaison de résultats

Comment Mellinger a-t-il/elle résolu le problème ?

Nous voyons dans la figure (2) que nous obtenons presque les mêmes résultats que dans [3]. La différence entre les deux solutions pourrait être reliée aux paramètres d'optimisation de `quadprog` ou les paramètres de tolérance sur la solution.



(a) Résultat des itérations, l'allocation de temps initiale donnait beaucoup trop de temps au segment du milieu.



(b) Valeur de la fonction de coût à chaque itération

pas de variation ?

Figure 3: Comparaison de résultats d'optimisation des temps de segment

??? Dans la figure (3) nous voyons le résultat de l'optimisation des segments de temps. Notre méthode converge beaucoup plus rapidement que celle de Mellinger mais dans le même nombre d'itérations; la raison n'est pas claire mais nous supposons que cela a rapport avec la valeur de h utilisé pour calculer la dérivée et le choix

de solveur et de ses paramètres. Mellinger ne spécifie pas le solveur qu'il a utilisé, seulement qu'il utilise une descente de gradient avec une recherche linéaire en *backtracking*; ce qui diffère de la méthode du point intérieur utilisé par `fmincon`. Nous voyons aussi dans le bloc suivant la sortie de console de Matlab lors de l'optimisation.

```
>> main_optimt

                                                First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
    0       1    8.711304e+04    0.000e+00    5.069e+05
    1       3    4.893364e+03    4.166e-12    3.105e+05    1.219e+00
    2       4    4.888882e+03    0.000e+00    3.882e+03    4.061e-03
    3       9    4.879789e+03    4.441e-16    1.242e+03    8.132e-03
    4      13    4.879264e+03    4.441e-16    1.193e+03    4.046e-03
    5      17    4.874708e+03    0.000e+00    1.210e+03    6.915e-02
    6      18    4.860743e+03    0.000e+00    4.416e+01    3.115e-02
    7      19    4.860729e+03    0.000e+00    3.209e+00    5.396e-04

Optimization stopped because the relative changes in all elements of x are
less than options.StepTolerance = 1.000000e-10, and the relative maximum constraint
violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

Optimization Metric                                          Options
max(abs(delta_x./x)) =   6.25e-11                    StepTolerance =   1e-10 (default)
relative max(constraint violation) =   0.00e+00   ConstraintTolerance =   1e-06 (default)

Elapsed time is 4.171527 seconds.
```

on n'est pas vraiment à l'optimalité On a laissé de $\pm 10^5$.

Nous voyons que le processus prend étonamment longtemps (4.17 secondes) quoi que le tout a été calculé sur un vieux laptop avec beaucoup d'applications ouvertes.

Finalement, pour démontrer le fonctionnement de notre application dans toutes les dimensions, nous générons une trajectoire en slalom contenant aussi des variations en altitude présenté dans la figure (4).

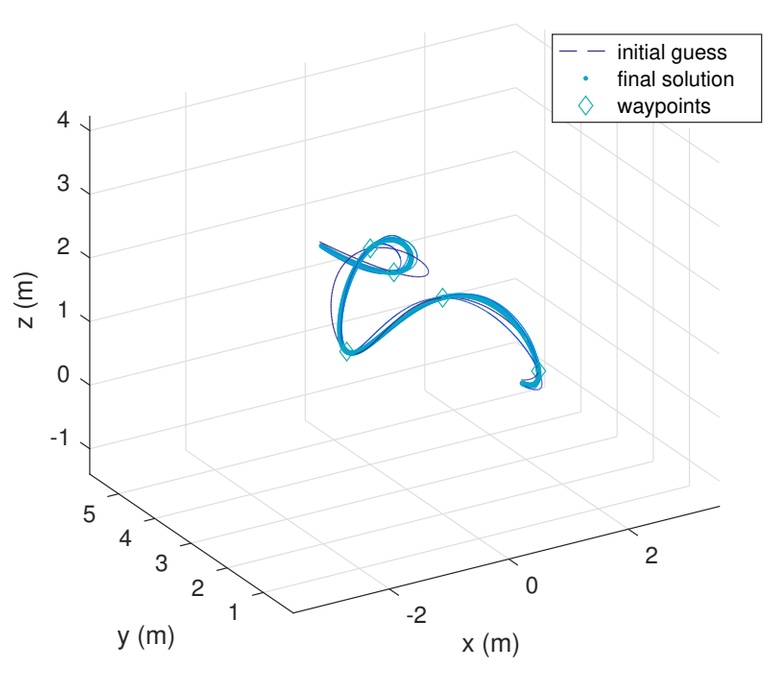


Figure 4: Slalom en 3 dimensions

Nondimentionalisation

L'une des particularités de la solution de la trajectoire optimale est qu'elle est nondimensionelle. Nous pouvons donc la dilater ou la contracter à notre guise dans le temps et dans l'espace pour l'adapter à nos besoins. Nous pourrions par exemple avoir besoin de satisfaire des contraintes sur l'accélération maximale que le véhicule multirotor peut fournir. Ces contraintes ne peuvent être inclusent dans le problème d'optimisation quadratique car elles sont non-linéaire, mais elles peuvent être satisfaites *a posteriori* en dilatant la trajectoire, tel que proposé par Loianno [2].

pourquoi ne pas résoudre un problème non-linéaire ?

D'ailleurs, c'est l'une des raisons pourquoi les implémentation initiales de l'algorithme ne fonctionnaient pas, n'ayant pas compris que les trajectoires étaient nondimensionelles nous tracions les courbes de t_i à t_{i+1} alors qu'en fait, il faut les tracer de 0 à 1.

Résumé de ce qui a été accompli

En somme voici ce qui a été accompli:

1. Développement mathématique pour la fonction de coût et la équations de contraintes du problème d'optimisation quadratique.
2. Développement mathématique pour le problème d'optimisation pour la répartition des temps de segment.
3. Implémentation dans Matlab et reproduction des résultats de Mellinger.
4. Traçage de la solution nondimentionnée

Résumé de ce qui reste à faire

Voici un résumé avec un échéancier des choses qui restent à faire. Nous aimerions passer plus rapidement possible à une implémentation qui pourrait se brancher à un simulateur pour confirmer que les trajectoires peuvent en effet être exécutées par un véhicule multirotor.

1. (2-3 jours) Traçage de la solution dimentionnée et satisfaisant les contraintes sur la force que les moteurs d'un multirotor peut fournir.
2. (1.5 semaine) Implémentation du problème d'optimisation quadratique en Python à l'aide de la librairie CVXOPT `http://cvxopt.org/`. Sera probablement assez lent !
3. (1 semaine) Implémentation du problème de réallocation des temps en Python à l'aide du solveur IPOPT (ou peut-être CVXOPT aussi). Pourquoi passer à Python ?
4. (1 semaine) Implémentation sur simulateur.
5. S'il reste du temps, ou si une tâche prend moins de temps que prévu, implémentation sur un vrai véhicule. ✓

Références

[1] A. P. Bry. *Control, estimation, and planning algorithms for aggressive flight using onboard sensing*. PhD thesis, Citeseer, 2012.

[2] G. Loianno, C. Brunner, G. McGrath, and V. Kumar. Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu. *IEEE Robotics and Automation Letters*, 2(2):404–411, 2017.

[3] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, May 2011.

[4] C. Richter, A. Bry, and N. Roy. *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, pages 649–666. Springer International Publishing, Cham, 2016.