Très bon rapport, merci !

18/20

MTH8408: Rapport Final

Remis pour le 30 avril, 2017

Professeur Dominique Orban

André Phu-Van Nguyen, 1525972

Contents

1 Description de la problématique

1.1 Objectifs du projet

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 [9] qui propose un contrôleur et un générateur de trajectoires pour un véhicule aérien multirotor. Plus précisément, nous faisons une implémentation seulement de la partie génération de trajectoire par une méthode d'optimisation numérique.

Dans un premier temps nous utilisons le langage interprété Matlab pour faciliter la modélisation. Puis ensuite nous passons au langage C++ qui est plus approprié pour l'implémentation en simulation et sur un véhicule réel. Finalement nous comparons plusieurs solveurs incluant OOQP, qpOASES, Gurobi, IPOPT et ALGlib pour trouver le plus performant pour notre problématique. Une validation de la trajectoire est aussi réalisée dans l'environnement de simulation Gazebo.

1.2 Trajectoires pour multirotors

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.

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:

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

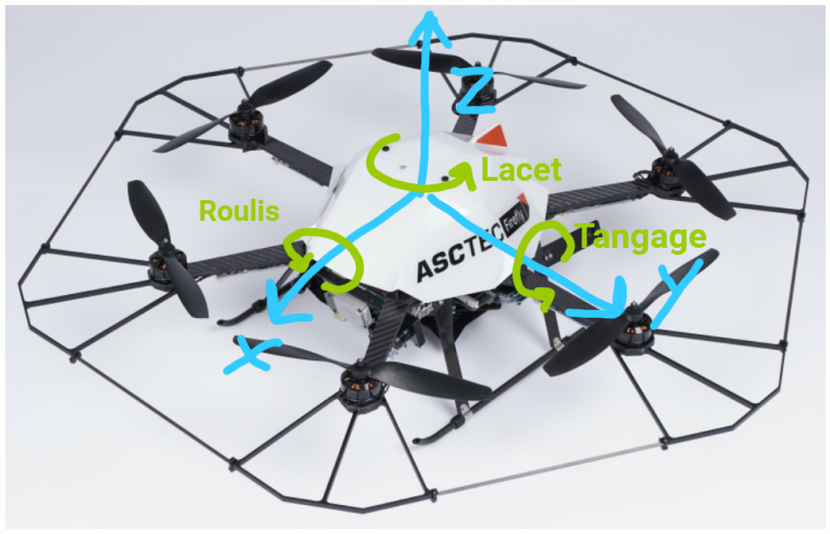


Figure 1: Systèmes de coordonnées d'un hexacoptère. L'angle de lacet est la rotation du véhicule autour de son axe Z, c'est-à-dire quand le véhicule tourne sur lui même.

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:

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

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

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

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 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 (2) 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.

2 Structure du problème d'optimisation

2.1 Problème d'optimisation quadratique

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 [10, 2].

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 avec contraintes d'égalité linéaires:

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

Par exemple si $n = 6$ nous avons un polynôme p représentant un segment de trajectoire:

$$p = c_6 t^6 + c_5 t^5 + c_4 t^4 + c_3 t^3 + c_2 t^2 + c_1 t + c_0 \tag{5}$$

Donc le vecteur $c = [c_6, c_5, c_4, c_3, c_3, c_2, c_1, c_0]^T$ est le vecteur des coefficients des polynômes. Nous notons aussi que la partie linéaire $f^T c$ est nulle.

2.1.1 Construction de la matrice hessienne

Pour un axe de déplacement p et un waypoint j, Richter et Bry [10, 2] indiquent que le processus de construction de la matrice H peut être automatisé par la formule suivante:

$$H_{pj} = \begin{cases} 2\Big(\prod_{m=0}^{r-1}(i-m)(l-m)\Big)\frac{\tau^{i+l-2r+1}}{i+l-2r+1} : & i \geq r \text{ ou } l \geq r \\ 0 : & \text{autrement} \end{cases} \tag{6}$$

où i et l sont les index de rangée et de colonne et τ est la durée du segment de trajectoire.[1] Concrètement, cela donne lieu à une matrice de la forme suivante:

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

où t_j et t_{j+1} sont les temps de départ et de fin d'un segment de trajectoire.

[1] Il faut faire une rotation de 180 degrés sur la matrice résultante pour avoir les puissances plus hautes dans les index plus bas.

La matrice Hessienne finale est construite en concaténant en diagonale les matrices H_{pj} pour chaque waypoint jusqu'à $j = m$ pour former H de (4).

$$H_x = \begin{bmatrix} H_{x1} & & & \\ & H_{x2} & & \\ & & \ddots & \\ & & & H_{xm} \end{bmatrix} \tag{8}$$

2.1.2 Construction des contraintes linéaires

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 [2](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

$$A = \begin{bmatrix} A_0 \\ A_\tau \end{bmatrix}, \quad b = \begin{bmatrix} b_0 \\ b_\tau \end{bmatrix} \tag{9}$$

$$A_{0_{rn}} = \begin{cases} \prod_{m=0}^{r-1}(r-m): & r = n \\ 0: & \text{autrement} \end{cases} \tag{10}$$

$$b_{0_r} = P^{(r)}(0) \tag{11}$$

$$A_{\tau_{rn}} = \begin{cases} \left(\prod_{m=0}^{r-1}(n-m)\right)\tau^{n-r}: & n \geq r \\ 0: & \text{autrement} \end{cases} \tag{12}$$

$$b_{\tau_r} = P^{(r)}(\tau) \tag{13}$$

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 [2]. 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:

$$A = \begin{bmatrix} A_0^0 & 0 & 0 & 0 & \dots & 0 & 0 \\ -A_\tau^0 & A_0^1 & 0 & 0 & \dots & 0 & 0 \\ 0 & -A_\tau^1 & A_0^2 & 0 & \dots & 0 & 0 \\ 0 & 0 & -A_\tau^2 & A_0^3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \ddots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & -A_\tau^{m-1} & A_0^m \\ 0 & 0 & 0 & 0 & 0 & 0 & A_\tau^m \end{bmatrix} \tag{14}$$

$$b = \begin{cases} \text{Valeur de la dérivée du polynôme } P \text{ imposée :} & \text{Si imposé} \\ 0 : & \text{autrement} \end{cases} \tag{15}$$

Exemple de matrice de contrainte

Prenons pour exemple la situation où en x nous avons les waypoints $0, 1$ et 2 et les temps $t = [0, 1, 2]$ nous obtenons:

$$A_x c - b_x = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 30 & 20 & 12 & 6 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\ 120 & 60 & 24 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & -6 & 0 & 0 & 0 \\ 360 & 120 & 24 & 0 & 0 & 0 & 0 & 0 & 0 & -24 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_{16} \\ c_{15} \\ c_{14} \\ c_{13} \\ c_{12} \\ c_{11} \\ c_{10} \\ c_{16} \\ c_{15} \\ c_{14} \\ c_{13} \\ c_{12} \\ c_{11} \\ c_{10} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = 0$$

2.2 Problème d'optimisation des temps de segment

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

$$\min \quad f(T) \tag{16}$$

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

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

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

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

$$\nabla_{g_i} f = \frac{f(T + hg_i) - f(T)}{h} \tag{17}$$

où h est une "petite" valeur et le vecteur colonne.[2] Donc, pour chaque direction du gradient, il faut rerésoudre le problème d'optimisation (2), ce qui explique pourquoi nous devons utiliser cette méthode numérique au lieu d'une méthode plus performante tel que la différentiation automatique.

$$g_i = \begin{cases} 1: & \text{à la position } i \\ \frac{-1}{m-1}: & \text{autrement} \end{cases} \tag{18}$$

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

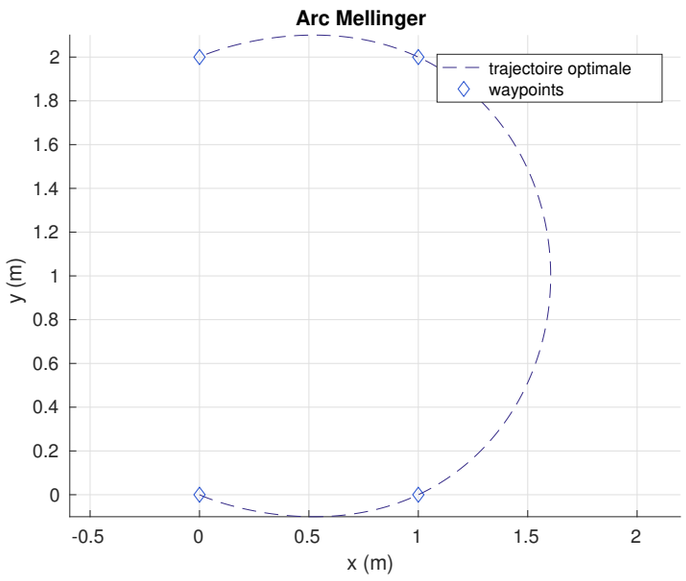
$$\nabla f = \begin{bmatrix} \nabla_{g_1} f \\ \nabla_{g_2} f \\ \nabla_{g_3} f \end{bmatrix} = \begin{bmatrix} \frac{f(T+hg_1)-f(T)}{h} \\ \frac{f(T+hg_2)-f(T)}{h} \\ \frac{f(T+hg_3)-f(T)}{h} \end{bmatrix}$$

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

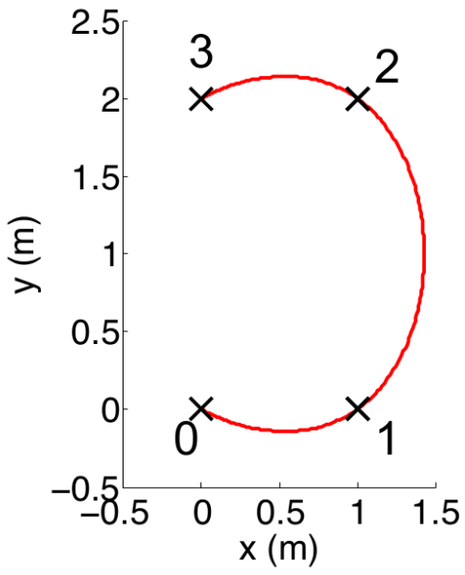
3 Implémentation

3.1 Prototype initial en Matlab

Le prototype initial 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.



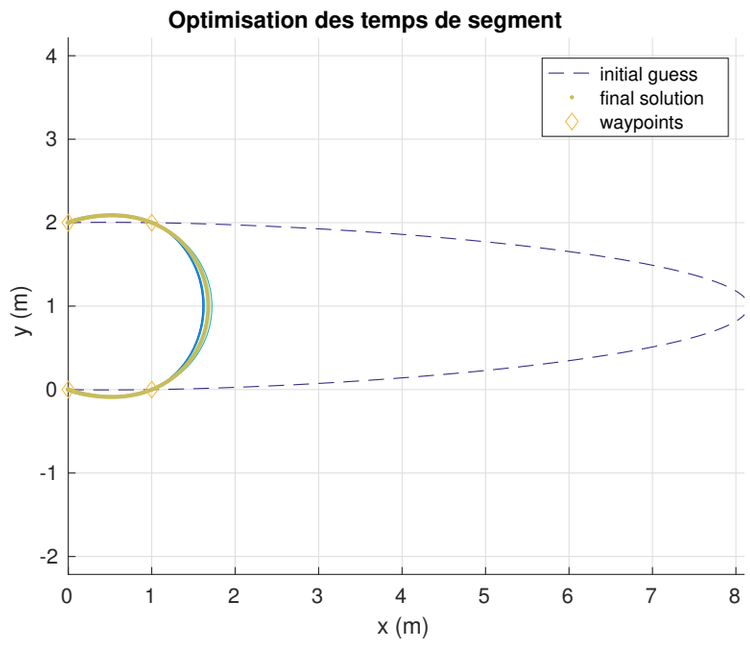
(a) Notre résultat



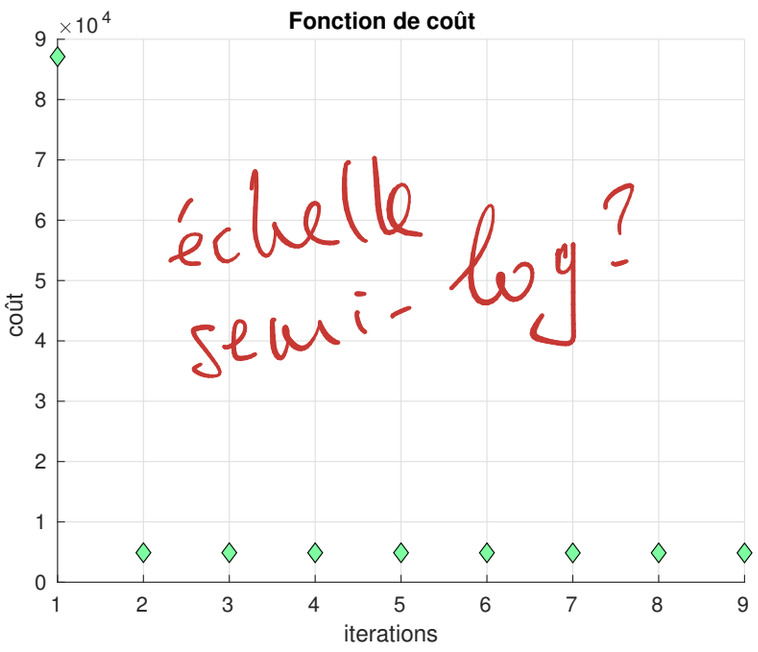
(b) Résultat de Mellinger [9]

Figure 2: Comparaison de résultats

Nous voyons dans la figure (2) que nous obtenons presque les mêmes résultats que dans [9]. La différence entre les deux solutions pourrait être reliée aux paramètres d'optimisation de `quadprog` ou les paramètres de tolérance sur la solution. D'ailleurs, Mellinger ne spécifie pas quel solveur il a utilisé pour résoudre le problème QP.



(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

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. Suite à un grand saut

d'une ordre de grandeur, le changement à la fonction de coût devient mineur. Notre méthode semble s'être approché de la solution en seulement une itération mais au final, l'exécution s'arrête après 7 itérations, le même nombre que Mellinger. Nous remarquons que le nombre d'itératiosn dépend grandement de la valeur de h choisie. 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.
```

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.

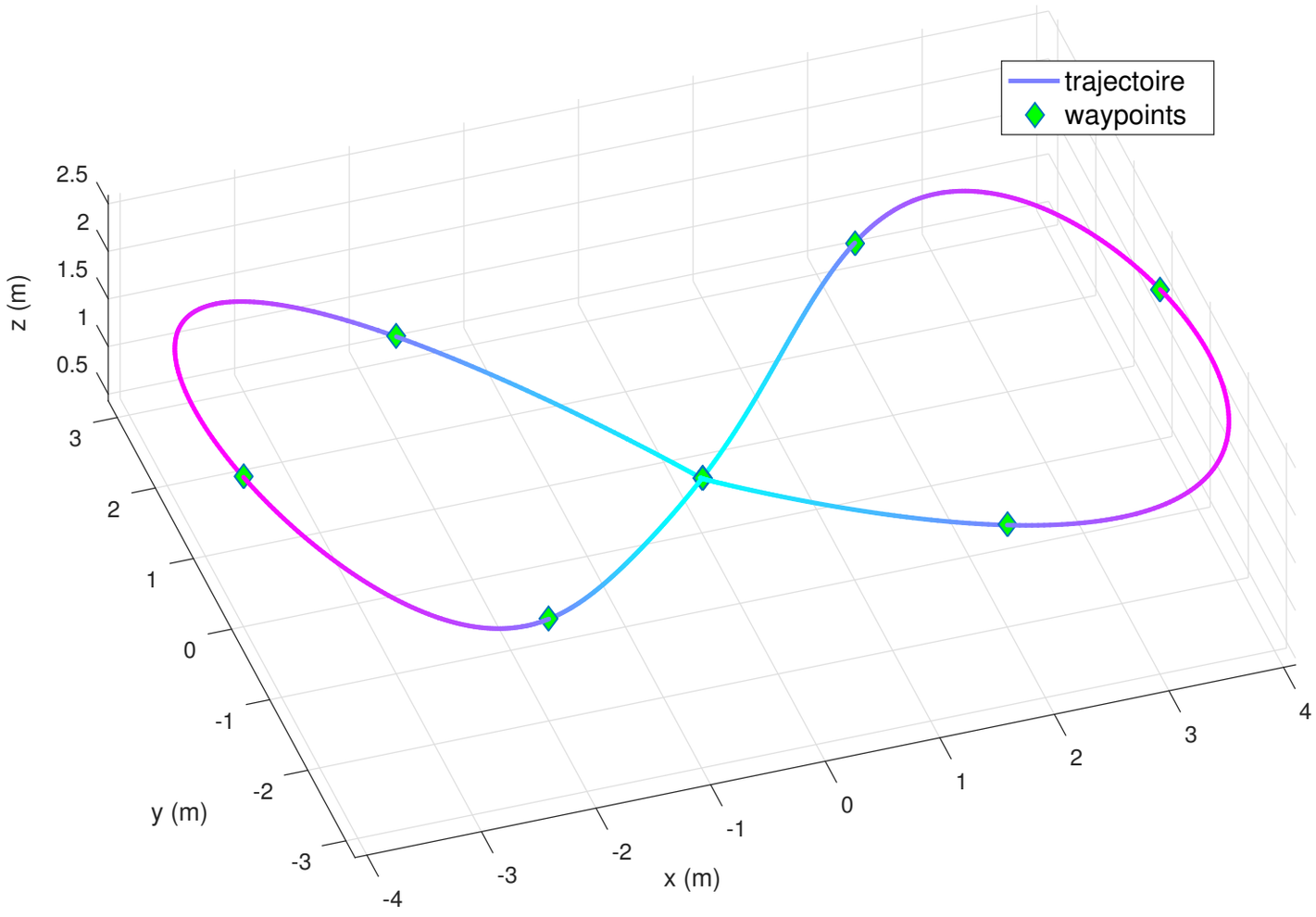
3.1.1 Interprétation de la solution non-dimensionelle

Dans un second test nous générons une trajectoire en forme de "8" avec un variation en hauteur et une allocation égale de 1 seconde à chaque segment, sans optimisation des temps de segment. Le résultat de cette génération est présenté dans la figure (4) dans laquelle la variation de la couleur représente la hauteur en Z. Dans l'annexe (A.1) nous voyons dans la trace d'exécution que `quadprog` ne prend qu'une seule itération pour résoudre le problème QP dans chaque dimension. c'est normal.

Examinons maintenant la solution du problème QP dans la dimension x. En transformant le vecteur c de 54 éléments en une matrice de $n+1 \times m+1$ *i.e.* le nombre de coefficients des polynômes par le nombre de waypoints incluant les conditions initiales. Nous avons la matrice dans l'équation (19).

Donc, chaque ligne représente les coefficients d'un polynôme non-dimensionné dans le temps. Notons que le dernier facteur de chaque polynôme est toujours la valeur d'un des waypoints puisque quand $t = 0$ la trajectoire s'évalue au coefficient c_0 tel que présenté dans l'équation (5).

Mellinger explique que pour toute solution optimale non-dimentionnée $\tilde{r}^*(\tau)$ où τ est le temps non-dimensionné, il est possible de faire une mise à l'échelle temporelle et spatiale de la solution. Par exemple Si nous voulons



On ne voit pas grand chose...

Figure 4: Trajectoire en forme de "8"

rallonger les temps d'arrivé à chaque waypoint par un facteur α nous pouvons appliquer le changement de variable $t_i = \alpha\tau_i$ ce qui donne lieu à une nouvelle solution

$$r^*(t) = \tilde{r}^*(t/\alpha)$$

Nous pouvons profiter de cette propriété pour satisfaire des contraintes *a posteriori* sur les accélérations demandées au véhicule. Par exemple dans la figure (6) nous voyons que la solution initiale nous demande des accélérations de près de $7.5m/s^2$ en x et presque $13m/s^2$ en y. En faisant une mise à l'échelle de la solution avec $\alpha = 2$, nous doublons le temps alloué à la trajectoire et nous obtenons des accélérations plus raisonables sur le véhicule à un maximum de $2m/s^2$ en x et $3.15m/s^2$ en y

$$c = \begin{bmatrix} 2.3370 & -8.2665 & 7.9295 & 0 & 0 & 0 & 0 \\ 0.2622 & 0.4434 & 1.6518 & -4.2073 & -0.0332 & 4.4075 & 2.0000 \\ 0.1200 & -0.3732 & -0.0640 & 1.5901 & -2.2431 & -1.0297 & 4.0000 \\ -0.0206 & 0.0881 & -0.1308 & 0.0010 & 0.2103 & -2.1480 & 2.0000 \\ 0.0206 & -0.0358 & -0.0000 & -0.0540 & 0.0000 & -1.9308 & 0 \\ -0.1200 & 0.3465 & 0.1308 & 0.0010 & -0.2103 & -2.1480 & -2.0000 \\ 0.2622 & -1.1297 & 0.0640 & 1.5901 & 2.2431 & -1.0297 & -4.0000 \\ -2.3370 & 5.7554 & -1.6518 & -4.2073 & 0.0332 & 4.4075 & -2.0000 \end{bmatrix} \tag{19}$$

Figure 5: Solution finale en x pour la trajectoire de la figure (4)

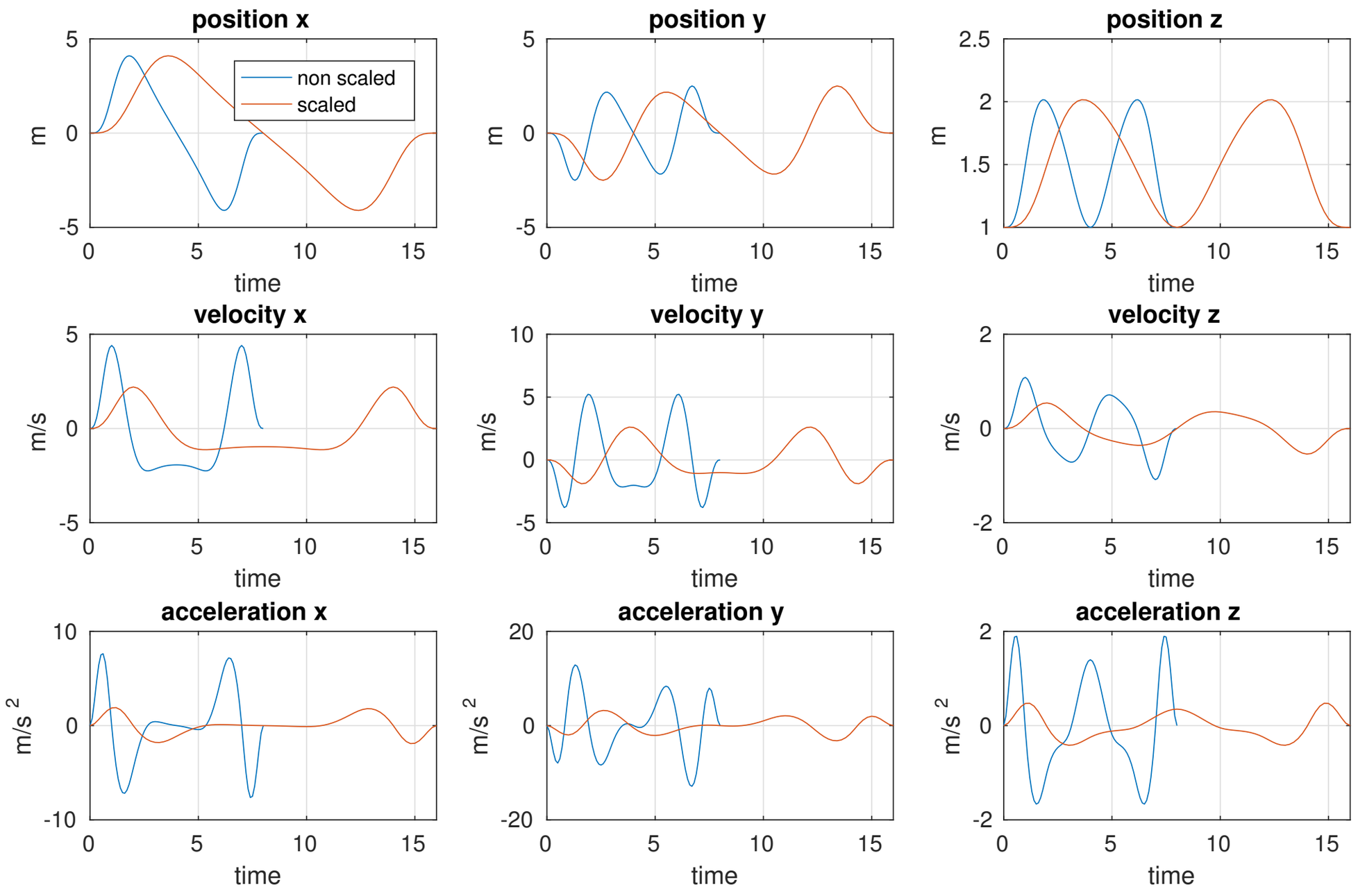


Figure 6: Comparaison entre la solution non-dimensionnelle et celle mise à l'échelle avec $\alpha = 2$

3.2 Implémentation finale en C++

L'intérêt de passer au C++ est d'éventuellement nous permettre d'utiliser le générateur de trajectories sur un vrai véhicule opérant avec le Robot Operating System (ROS) ou du moins, un véhicule simulé avec ROS dans l'environmment de simulation Gazebo. À l'aide de la librairie d'algèbre linéaire Eigen [5] le transfert de l'essentiel du code a été relativement simple.

3.2.1 Comparaison de solveurs QP

Nous avons structuré le code C++ à fin de pouvoir comparer plusieurs solveurs QP dont: OOQP [4], Gurobi [6], qpOASES [3] et le *Bound and Linear Equality and Inequality Constraint* (BLEIC) de ALGLIB[1]. Nous commençons par reprendre l'exemple de génération d'une trajectoire en "8" avec les mêmes waypoints que ceux de la figure (4) sans une optimisation sur l'allocation des temps de segment. Pour calculer la valeur finale de la fonction objectif nous faisons une sommation de la valeur de la fonction de coût ($x^T Hx$) dans les trois dimensions x, y et z. Le nombre d'itérations est la somme des itérations requises pour résoudre le problème dans les 3 dimensions. Le temps d'exécution est mesuré à partir du début de l'optimisation et n'inclut pas le temps requis pour bâtir le problème. Pour assurer la cohérence de nos résultats, le temps d'exécution présenté est une moyenne sur 100 essais.

Nous voyons dans le tableau (1) les résultats de nos tests. Nous voyons que OOQP performe beaucoup mieux que les autres solveurs et résout les trois problèmes QP en une seule itération de la même façon que

`quadprog`. Les méthodes simplex de Gurobi prennent plus d'itérations mais sont comparables en termes de temps d'exécution à la méthode barrière par défaut. qpOASES et BLEIC sont les pires en termes de performance, se basant sur la méthode de l'ensemble des contraintes actives. Dans notre problème, cette méthode n'a pas de but puisque toutes nos contraintes sont actives car notre problème ne comporte que des contraintes d'égalitées. Nous remarquons tout de même que tous les solveurs réussissent à se rendre au même minimum, la différence en `quadprog` et les autres résultats peut se résumer à une erreur d'arrondissement.

Solveur	Type	n. itér. $[x, y, z]$	$f(x)$	Temps exéc. (ns)
Matlab `quadprog`	Primal-Dual point intérieur	[1 1 1]	59 859.09	10 071 990
OOQP	Primal-Dual point intérieur	[1 1 1]	59 859.10	230 939
Gurobi	Barrière	[10 10 7]	59 859.10	2 786 326
	Primal-Simplex	[7 7 7]	59 859.10	1 940 684
	Dual-Simplex	[65 65 65]	59 859.10	2 375 172
qpOASES	Online active set	[6 6 6]	59 856.10	3 896 080
ALGLIB QP-BLEIC	Active set	[[7 11] [7 10] [8 11]]	59 856.10	93 946 093

Table 1: Comparaison des solveurs QP. L'algorithme QP-BLEIC donne les "outer iterations" et les "inner iterations".

3.2.2 Comparaison des solveurs non-linéaires

Pour remplacer `fmincon` en C++ nous avons eu recours à BLEIC et à la méthode *Sequential Least-Squares Quadratic Programming* NLopt [7]. Malheureusement nous voulions aussi tenter d'autres algorithmes tel que BFGS et le Lagrangien augmenté mais nous n'avons pas réussis à les faire fonctionner. À la lumière des résultats du tableau (1) nous avons utilisé OOQP comme solveur interne pour évaluer $f(x)$.

Dans le tableau (2) nous voyons que l'algorithme BLEIC est plus performant en termes d'itérations que `fmincon` pour un temps d'exécution deux ordres de grandeurs plus petit. NLopt SLSQP performe beaucoup d'itérations mais avec un temps équivalent à BLEIC. Cependant, la chose la plus important à remarquer ici est le fait que cette optimisation faite par dessus l'optimisation QP permet d'atteindre une valeur de fonction beaucoup plus petite, peu importe l'algorithme utilisé.

Solveur	Type	n. itér.	$f_0(x)$	$f(x)$	Temps exéc. (ms)
Matlab `fmincon`	Point intérieur	17	59 859.09	9 354.66	11 965.80
ALGlib BLEIC	Active set	12	59 859.10	9 354.70	219
NLopt	SLSQP	59	59 859.10	9 358.01	212

Table 2: Comparaison des solveurs non-linéaires

4 Résultats en simulation

À fin de vérifier que nos trajectoires sont bel et bien réalisables (qu'un multirotor puisse les voler) nous avons branché notre implémentation C++ à Robot Operating System et nous avons simulé un multirotor dans l'application Gazebo. Puisque l'écriture du contrôleur de vol ne faisait pas parti du projet, nous avons repris le *Model Predictive Controller* de Kamel et al. [8]. Une vidéo complète du vol peut être visionnée ici `https://www.youtube.com/watch?v=3_fYaVvONS4`. Nous voyons dans la figure (7) que l'horizon de prédiction du MPC (ligne rouge) s'accote à la trajectorie commandée (ligne verte). Ceci voudrait dire que la trajectoire commandée est réalisable par le véhicule. La distance entre les deux courbes pourrait être réduite en augmentant le paramètre α tel que discuté dans la section (3.1.1).

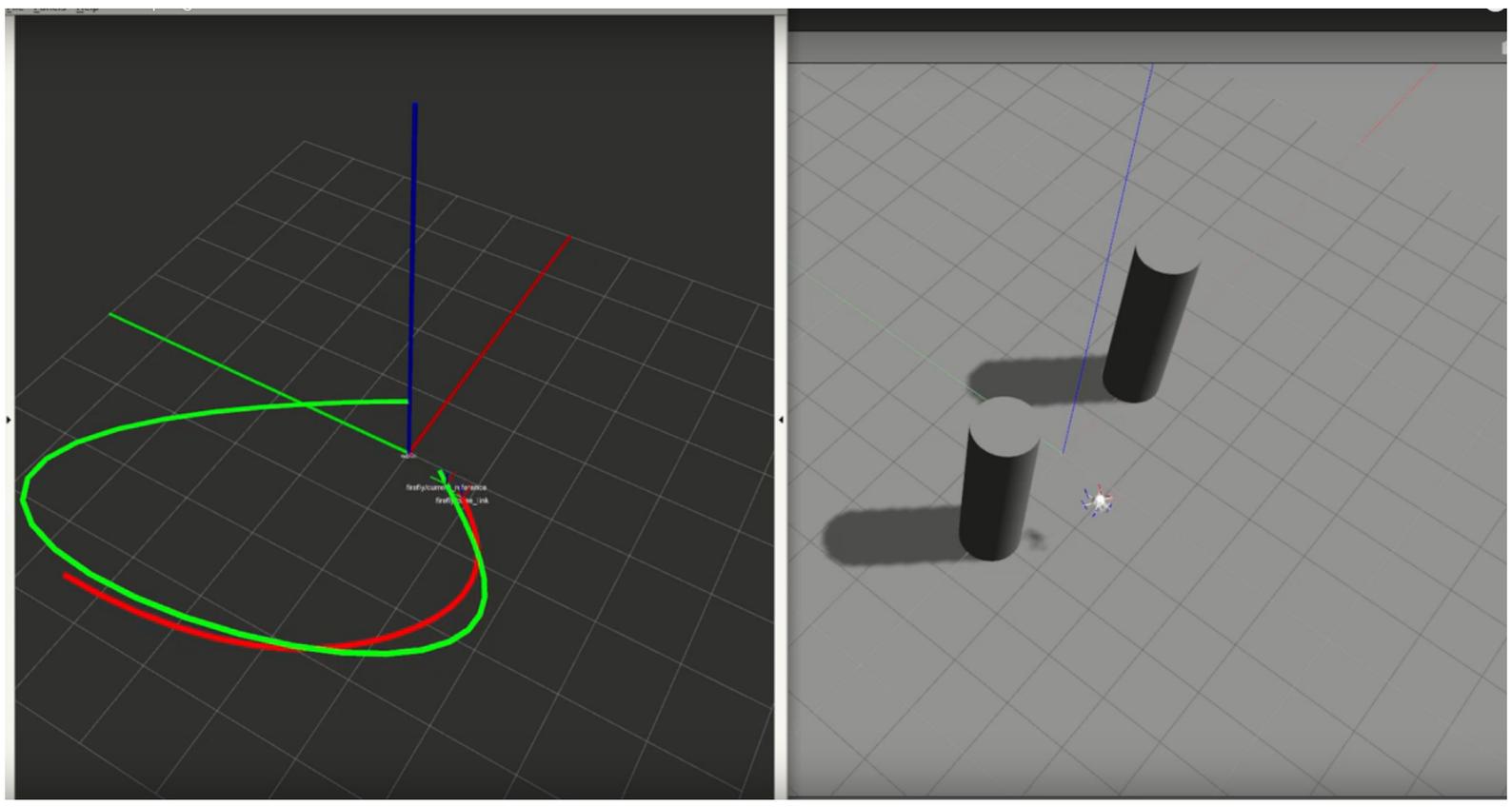


Figure 7: Vol d'un hexacoptère en simulation

5 Conclusion

En conclusion nous avons presque réussis à atteindre tous nos objectifs initiaux soit de reproduire la méthode de Mellinger et d'implémenter le système en C++. Faute de temps nous n'avons pas pu implémenter le système sur un vrai véhicule mais les tests en simulations sont assez prometteurs pour la suite.

En somme, nous avons implémenté grâce à plusieurs solveurs l'optimisation d'un problème QP en Matlab et en C++ et l'optimisation d'une fonction non-linéaire également en Matlab et en C++. Grâce à plusieurs librairies gratuites trouvées en ligne, nous avons pu aussi comparer plusieurs solveurs pour trouver la méthode d"optimisation qui convient le mieux à notre problème. Qui fini par être la méthode du point intérieur d'OOQP et la méthode d'active set de NLopt. Par contre, nous croyons bien que si nous avions réussis à faire fonctionner l'une des autres méthodes de NLopt, nos conclusions seraient différentes.

Finalement, nous en profitons aussi pour mettre notre implémentation à source ouverte [3] pour qu'elle puisse servir dans le contexte de la recherche en robotique aérienne.

[3] Ici `https://github.com/andre-nguyen/mth8408/tree/master/projet`

Références

[1] S. Bochkanov. Alglib, 2017.

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

[3] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[4] E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Trans. Math. Softw.*, 29(1):58–81, Mar. 2003.

[5] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[6] I. Gurobi Optimization. Gurobi optimizer reference manual, 2016.

[7] S. G. Johnson. The nlopt nonlinear-optimization package. 2017.

[8] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart. Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system. In A. Koubaa, editor, *Robot Operating System (ROS) The Complete Reference, Volume 2*. Springer. (to appear).

[9] 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.

[10] 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.

Annexes

A Traces d'exécution

A.1 Génération de la trajectoire en forme de "8"

```
>> main

 Iter             Fval  Primal Infeas    Dual Infeas  Complementarity
    0    2.461440e+05   4.800000e+02   4.546657e+04     0.000000e+00
    1    7.964709e+03   2.273737e-13   5.456968e-12     0.000000e+00

Optimization completed: The relative dual feasibility, 2.105312e-16,
is less than options.OptimalityTolerance = 1.000000e-08, the complementarity measure,
0.000000e+00, is less than options.OptimalityTolerance, and the relative maximum constraint
violation, 6.315935e-16, is less than options.ConstraintTolerance = 1.000000e-08.

Optimization Metric                                    Options
relative dual feasibility =   2.11e-16            OptimalityTolerance =   1e-08 (default)
complementarity measure =   0.00e+00              OptimalityTolerance =   1e-08 (default)
relative max(constraint violation) =   6.32e-16    ConstraintTolerance =   1e-08 (default)


 Iter             Fval  Primal Infeas    Dual Infeas  Complementarity
    0    2.461440e+05   4.800000e+02   4.584848e+04     0.000000e+00
    1    2.143702e+04   1.705303e-13   1.182343e-11     0.000000e+00

Optimization completed: The relative dual feasibility, 4.561509e-16,
is less than options.OptimalityTolerance = 1.000000e-08, the complementarity measure,
0.000000e+00, is less than options.OptimalityTolerance, and the relative maximum constraint
violation, 4.736952e-16, is less than options.ConstraintTolerance = 1.000000e-08.

Optimization Metric                                    Options
relative dual feasibility =   4.56e-16            OptimalityTolerance =   1e-08 (default)
complementarity measure =   0.00e+00              OptimalityTolerance =   1e-08 (default)
relative max(constraint violation) =   4.74e-16    ConstraintTolerance =   1e-08 (default)


 Iter             Fval  Primal Infeas    Dual Infeas  Complementarity
    0    2.461440e+05   4.800000e+02   4.099967e+04     0.000000e+00
    1    5.278115e+02   8.526513e-14   4.547474e-12     0.000000e+00

Optimization completed: The relative dual feasibility, 1.754427e-16,
is less than options.OptimalityTolerance = 1.000000e-08, the complementarity measure,
0.000000e+00, is less than options.OptimalityTolerance, and the relative maximum constraint
violation, 2.368476e-16, is less than options.ConstraintTolerance = 1.000000e-08.

Optimization Metric                                    Options
relative dual feasibility =   1.75e-16            OptimalityTolerance =   1e-08 (default)
complementarity measure =   0.00e+00              OptimalityTolerance =   1e-08 (default)
relative max(constraint violation) =   2.37e-16    ConstraintTolerance =   1e-08 (default)

total cost: 2.992954e+04
Elapsed time is 0.015659 seconds.
```

A.2 Gurobi Primal Simplex

```
Optimize a model with 50 rows, 56 columns and 236 nonzeros
Model has 80 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 4e+02]
  Objective range  [0e+00, 0e+00]
  QObjective range [1e-02, 3e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 4e+00]
Presolve removed 11 rows and 11 columns
Presolve time: 0.00s
Presolved: 39 rows, 45 columns, 208 nonzeros
Presolved model has 69 quadratic objective terms

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.4000000e-01   0.000000e+00   0.000000e+00      0s
Warning: Markowitz tolerance tightened to 0.125
       0    3.0832262e+08   0.000000e+00   6.295325e+09      0s
       7    7.9663854e+03   0.000000e+00   0.000000e+00      0s

Solved in 7 iterations and 0.00 seconds
Optimal objective  7.966385402e+03
iters 7
Optimize a model with 50 rows, 56 columns and 236 nonzeros
Model has 80 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 4e+02]
  Objective range  [0e+00, 0e+00]
  QObjective range [1e-02, 3e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 2e+00]
Presolve removed 11 rows and 11 columns
Presolve time: 0.00s
Presolved: 39 rows, 45 columns, 208 nonzeros
Presolved model has 69 quadratic objective terms

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    8.0000000e-02   0.000000e+00   0.000000e+00      0s
Warning: Markowitz tolerance tightened to 0.5
       0    1.6125969e+08   0.000000e+00   4.552371e+09      0s
       7    2.1440601e+04   0.000000e+00   0.000000e+00      0s

Solved in 7 iterations and 0.00 seconds
Optimal objective  2.144060111e+04
iters 7
Optimize a model with 50 rows, 56 columns and 236 nonzeros
Model has 80 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 4e+02]
  Objective range  [0e+00, 0e+00]
  QObjective range [1e-02, 3e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 2e+00]
Presolve removed 11 rows and 11 columns
Presolve time: 0.00s
Presolved: 39 rows, 45 columns, 208 nonzeros
Presolved model has 69 quadratic objective terms

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    9.5000000e-02   0.000000e+00   0.000000e+00      0s
```

```
Warning: Markowitz tolerance tightened to 0.125
       0    1.2203121e+07   0.000000e+00   1.252409e+09      0s
       7    5.2800402e+02   0.000000e+00   0.000000e+00      0s

Solved in 7 iterations and 0.00 seconds
Optimal objective  5.280040205e+02
iters 7
fval 59859.1
avg exec time: 2169561 ns

Process finished with exit code 0
```

A.3 Gurobi Dual Simplex

```
/home/andrephu-vannguyen/Documents/school/mth8408_ws/src/mth8408/projet/ros/an_min_snap_traj/cmake-build-debug/devel/lib/an_min_
Optimize a model with 50 rows, 56 columns and 236 nonzeros
Model has 80 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 4e+02]
  Objective range  [0e+00, 0e+00]
  QObjective range [1e-02, 3e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 4e+00]
Presolve removed 11 rows and 11 columns
Presolve time: 0.00s
Presolved: 39 rows, 45 columns, 208 nonzeros
Presolved model has 69 quadratic objective terms

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.4000000e-01   0.000000e+00   0.000000e+00      0s
      65    7.9663854e+03   0.000000e+00   0.000000e+00      0s

Solved in 65 iterations and 0.00 seconds
Optimal objective  7.966385402e+03
iters 65
Optimize a model with 50 rows, 56 columns and 236 nonzeros
Model has 80 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 4e+02]
  Objective range  [0e+00, 0e+00]
  QObjective range [1e-02, 3e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 2e+00]
Presolve removed 11 rows and 11 columns
Presolve time: 0.00s
Presolved: 39 rows, 45 columns, 208 nonzeros
Presolved model has 69 quadratic objective terms

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    8.0000000e-02   0.000000e+00   0.000000e+00      0s
      65    2.1440601e+04   0.000000e+00   0.000000e+00      0s

Solved in 65 iterations and 0.00 seconds
Optimal objective  2.144060111e+04
iters 65
Optimize a model with 50 rows, 56 columns and 236 nonzeros
Model has 80 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 4e+02]
  Objective range  [0e+00, 0e+00]
```

```
  QObjective range [1e-02, 3e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 2e+00]
Presolve removed 11 rows and 11 columns
Presolve time: 0.00s
Presolved: 39 rows, 45 columns, 208 nonzeros
Presolved model has 69 quadratic objective terms

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    9.5000000e-02   0.000000e+00   0.000000e+00      0s
      65    5.2800402e+02   0.000000e+00   0.000000e+00      0s

Solved in 65 iterations and 0.00 seconds
Optimal objective  5.280040205e+02
iters 65
fval 59859.1
avg exec time: 2459506 ns

Process finished with exit code 0
```

A.4 Gurobi Barrier

```
Optimize a model with 50 rows, 56 columns and 236 nonzeros
Model has 80 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 4e+02]
  Objective range  [0e+00, 0e+00]
  QObjective range [1e-02, 3e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 4e+04]
Presolve removed 11 rows and 11 columns
Presolve time: 0.00s
Presolved: 39 rows, 45 columns, 208 nonzeros
Presolved model has 69 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 61
 AA' NZ     : 2.730e+02
 Factor NZ  : 7.160e+02
 Factor Ops : 1.120e+04 (less than 1 second per iteration)
 Threads    : 1

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   7.97935603e+03 -7.97887603e+03  7.77e-11 3.82e+01  0.00e+00     0s
   1   7.97908529e+03 -7.80709902e+03  7.58e-08 3.78e+01  0.00e+00     0s
   2   7.97881305e+03 -7.63246353e+03  1.04e-07 3.74e+01  0.00e+00     0s
   3   7.97846464e+03 -7.40607153e+03  2.03e-07 3.69e+01  0.00e+00     0s
   4   7.97748917e+03 -6.75352845e+03  3.19e-07 3.53e+01  0.00e+00     0s
   5   7.97614012e+03 -5.80012065e+03  6.63e-07 3.30e+01  0.00e+00     0s
   6   7.97356474e+03 -3.76541655e+03  1.28e-06 2.81e+01  0.00e+00     0s
   7   7.97068429e+03 -9.58692733e+02  2.11e-06 2.14e+01  0.00e+00     0s
   8   7.96766724e+03  3.58040908e+03  3.72e-06 1.05e+01  0.00e+00     0s
   9   7.96656436e+03  7.96638102e+03  3.55e-06 1.05e-05  0.00e+00     0s
  10   7.96638539e+03  7.96638540e+03  1.59e-10 6.37e-11  0.00e+00     0s

Barrier solved model in 10 iterations and 0.00 seconds
Optimal objective 7.96638539e+03
```

```
Optimize a model with 50 rows, 56 columns and 236 nonzeros
Model has 80 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 4e+02]
  Objective range  [0e+00, 0e+00]
  QObjective range [1e-02, 3e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 2e+00]
Presolve removed 11 rows and 11 columns
Presolve time: 0.00s
Presolved: 39 rows, 45 columns, 208 nonzeros
Presolved model has 69 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 61
 AA' NZ     : 2.730e+02
 Factor NZ  : 7.160e+02
 Factor Ops : 1.120e+04 (less than 1 second per iteration)
 Threads    : 1

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   2.14468667e+04 -2.14467067e+04  2.17e-10 5.76e+01  0.00e+00     0s
   1   2.14467856e+04 -2.11274741e+04  7.78e-08 5.72e+01  0.00e+00     0s
   2   2.14467099e+04 -2.08279330e+04  1.54e-07 5.68e+01  0.00e+00     0s
   3   2.14465255e+04 -2.00877424e+04  2.24e-07 5.58e+01  0.00e+00     0s
   4   2.14461716e+04 -1.86270436e+04  5.88e-07 5.38e+01  0.00e+00     0s
   5   2.14455006e+04 -1.56931892e+04  8.09e-07 4.99e+01  0.00e+00     0s
   6   2.14445477e+04 -1.10642076e+04  2.15e-06 4.37e+01  0.00e+00     0s
   7   2.14435920e+04 -5.66229415e+03  4.42e-06 3.64e+01  0.00e+00     0s
   8   2.14415784e+04  1.73901431e+04  4.22e-06 5.44e+00  0.00e+00     0s
   9   2.14407501e+04  2.14405971e+04  1.47e-06 5.44e-06  0.00e+00     0s
  10   2.14406011e+04  2.14406011e+04  1.02e-10 2.55e-11  0.00e+00     0s

Barrier solved model in 10 iterations and 0.00 seconds
Optimal objective 2.14406011e+04

Optimize a model with 50 rows, 56 columns and 236 nonzeros
Model has 80 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 4e+02]
  Objective range  [0e+00, 0e+00]
  QObjective range [1e-02, 3e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 2e+00]
Presolve removed 11 rows and 11 columns
Presolve time: 0.00s
Presolved: 39 rows, 45 columns, 208 nonzeros
Presolved model has 69 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 61
 AA' NZ     : 2.730e+02
 Factor NZ  : 7.160e+02
 Factor Ops : 1.120e+04 (less than 1 second per iteration)
 Threads    : 1

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   5.28436369e+02 -5.28246369e+02  3.26e-11 9.93e+00  0.00e+00     0s
```

```
   1   5.28408327e+02 -4.91423019e+02  5.90e-08 9.59e+00  0.00e+00     0s
   2   5.28383712e+02 -4.57971824e+02  6.35e-08 9.27e+00  0.00e+00     0s
   3   5.28328719e+02 -3.78694453e+02  1.89e-07 8.53e+00  0.00e+00     0s
   4   5.28232947e+02 -2.20520321e+02  3.89e-07 7.04e+00  0.00e+00     0s
   5   5.28109249e+02  5.69630373e+01  6.75e-07 4.43e+00  0.00e+00     0s
   6   5.28024076e+02  5.28003549e+02  7.72e-07 4.43e-06  0.00e+00     0s
   7   5.28004019e+02  5.28004021e+02  6.46e-11 2.39e-11  0.00e+00     0s

Barrier solved model in 7 iterations and 0.00 seconds
Optimal objective 5.28004019e+02

fval 59859.1
avg exec time: 4461093 ns

Process finished with exit code 0
```

A.5 qpOASES

```
iter addB remB addC remC   hom len       tau      stat     bfeas     cfeas     bcmpl     ccmpl      Tmin
   0         0             1.00e+00  2.22e-25  1.11e-16  0.00e+00  4.14e-25  0.00e+00  5.03e-34  1.00e+00
   1         7             1.21e+00  2.83e-13  7.77e-16  0.00e+00  2.17e-19  0.00e+00  2.00e-22  1.00e+00
   2         1             1.21e+00  3.91e-10  4.16e-15  0.00e+00  3.47e-17  0.00e+00  7.54e-19  1.00e+00
   3        28             1.21e+00  1.53e-10  3.33e-15  0.00e+00  5.55e-17  0.00e+00  2.32e-19  1.00e+00
   4        21             1.21e+00  1.67e-10  6.30e-15  0.00e+00  7.63e-17  0.00e+00  3.50e-19  1.00e+00
   5        14             1.21e+00  3.60e-11  5.69e-15  0.00e+00  9.71e-17  0.00e+00  5.02e-19  1.00e+00
   6                       0.00e+00  1.00e+00  1.46e-11  0.00e+00  3.91e-14  0.00e+00  1.79e-11  1.00e+00

iter addB remB addC remC   hom len       tau      stat     bfeas     cfeas     bcmpl     ccmpl      Tmin
   0         0             1.00e+00  2.22e-25  1.67e-16  0.00e+00  1.81e-25  0.00e+00  6.30e-34  1.00e+00
   1         7             1.21e+00  2.83e-13  5.55e-16  0.00e+00  7.59e-19  0.00e+00  3.56e-22  1.00e+00
   2         1             1.21e+00  3.91e-10  3.90e-15  0.00e+00  5.55e-17  0.00e+00  2.35e-19  1.00e+00
   3        28             1.21e+00  1.53e-10  3.89e-15  0.00e+00  1.04e-16  0.00e+00  4.53e-19  1.00e+00
   4        21             1.21e+00  1.67e-10  5.11e-15  0.00e+00  2.78e-17  0.00e+00  3.62e-19  1.00e+00
   5        14             1.21e+00  3.60e-11  2.50e-15  0.00e+00  5.55e-17  0.00e+00  1.99e-19  1.00e+00
   6                       0.00e+00  1.00e+00  3.64e-11  0.00e+00  7.11e-14  0.00e+00  9.08e-11  1.00e+00

iter addB remB addC remC   hom len       tau      stat     bfeas     cfeas     bcmpl     ccmpl      Tmin
   0         0             1.00e+00  2.22e-25  2.50e-16  0.00e+00  1.03e-24  0.00e+00  2.96e-34  1.00e+00
   1         7             1.21e+00  2.83e-13  1.33e-15  0.00e+00  3.52e-19  0.00e+00  2.07e-22  1.00e+00
   2         1             1.21e+00  3.91e-10  3.33e-15  0.00e+00  3.67e-17  0.00e+00  1.24e-19  1.00e+00
   3        28             1.21e+00  1.53e-10  7.22e-15  0.00e+00  1.18e-16  0.00e+00  5.49e-19  1.00e+00
   4        21             1.21e+00  1.67e-10  3.11e-15  0.00e+00  8.33e-17  0.00e+00  3.49e-19  1.00e+00
   5        14             1.21e+00  3.60e-11  1.17e-14  0.00e+00  1.82e-17  0.00e+00  3.25e-19  1.00e+00
   6                       0.00e+00  1.00e+00  1.82e-12  0.00e+00  3.55e-14  0.00e+00  2.16e-12  1.00e+00
fval 59859.1
avg exec time: 4954292 ns
```