RSI07 - Diagnostic et Tolérance aux fautes des systèmes dynamiques

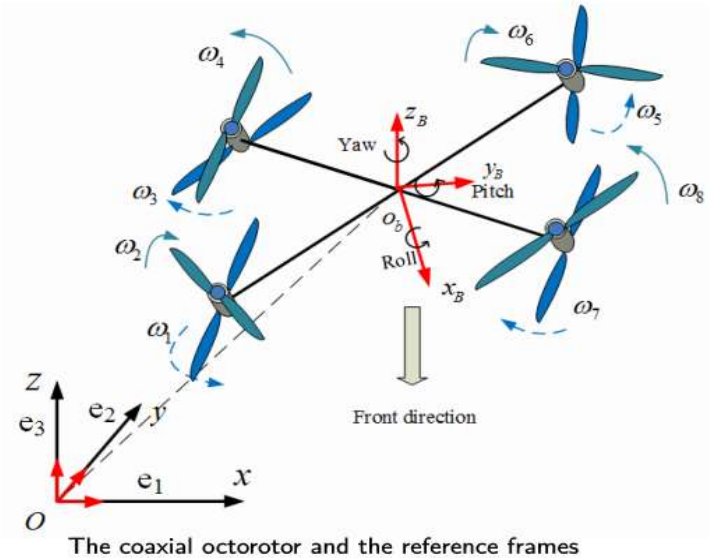
---

Travaux Pratiques

Feb. 2019

Partie 1: Modélisation et Loi de commande : (50 min)

La figure 1 donne une représentation schématique du drone octorotor et ses paramètres sont donnés dans le tableau 1.



The coaxial octorotor and the reference frames

K_f	Coefficient de Poussée	$3.5 * 10^{-5}\ N.s^2/rad^2$
K_t	Coefficient de moment	$3 * 10^{-7}\ Nm/rad^2$
M	Masse	$1.6\ Kg$
l	Longueur du bras	$0.23\ m$
I_{xx}, I_{yy}	Moments d'inertie	$0.0255\ Kg.m^2$
I_{zz}	Moment d'inertie	$0.0388\ Kg.m^2$

Table 1: Paramètres de l'octorotor

Les équations différentielles régissant le mouvement sont déduites à partir du formalisme d'Euler-Lagrange :

$$\ddot{x} = (cos\Phi\ sin\theta\ cos\psi + sin\Phi\ sin\psi)\frac{u_f}{m} \qquad \ddot{\Phi} = \frac{I_{yy}-I_{zz}}{I_{xx}}\dot{\theta}\dot{\psi} - \frac{J_r}{I_{xx}}\dot{\theta}\Omega + \frac{1}{I_{xx}}\tau_\Phi$$

$$\ddot{y} = (cos\Phi\ sin\theta\ sin\psi - sin\Phi\ cos\psi)\frac{u_f}{m} \qquad \ddot{\theta} = \frac{I_{zz}-I_{xx}}{I_{yy}}\dot{\Phi}\dot{\psi} + \frac{J_r}{I_{yy}}\dot{\Phi}\Omega + \frac{1}{I_{yy}}\tau_\theta$$

$$\ddot{z} = (cos\Phi\ cos\theta)\frac{u_f}{m} - g \qquad \ddot{\psi} = \frac{I_{xx}-I_{yy}}{I_{zz}}\dot{\Phi}\dot{\theta} + \frac{1}{I_{zz}}\tau_\psi$$

Les entrées de commande et les perturbations sont données par :

$$u_f = F_{12} + F_{34} + F_{56} + F_{78}$$

$\tau_{\Phi} = (F_{78} + F_{56} - F_{12} - F_{34}) l \frac{\sqrt{2}}{2}$

$\tau_{\theta} = (F_{34} + F_{56} - F_{12} - F_{78}) l \frac{\sqrt{2}}{2}$

$\tau_{\psi} = (\tau_2 + \tau_3 + \tau_6 + \tau_7) - (\tau_1 + \tau_4 + \tau_5 + \tau_8)$

Avec : $F_i = K_f \omega_i^2, \quad F_{ij} = \sigma(F_i + F_j), \quad \tau_i = K_t \omega_i^2$.

<u>Modèle non Linéaire</u> : (20 min)

1. Dans un premier temps, nous allons étudier le modèle non linéaire représenté par les équations du mouvement ci-dessus.
   Proposez les variables d'état. Combien d'états choisissez-vous ?
2. Compléter le modèle sur Matlab/Simulink.

<u>Commande Par saturations</u> : (30 min)

Le modèle dynamique présenté ci-dessus est non linéaire. On considère que l'angle de lacet est toujours asservi à 0 et les angles de roulis et tangage sont très petits. Si un angle α est petit (de l'ordre de quelques degrés) on peut écrire : $\sin(\alpha) \cong \alpha$ et $\cos(\alpha) = 1$. Pour la synthèse de la loi de commande, les équations de mouvement sont simplifiées comme suit :

$$m\ddot{x} = u\theta$$
$$m\ddot{y} = -u\Phi$$
$$m\ddot{z} = u - mg$$
$$\tau_{\Phi} = I_{xx}\ddot{\Phi}$$
$$\tau_{\theta} = I_{yy}\ddot{\theta}$$
$$\tau_{\psi} = I_{zz}\ddot{\psi}$$

Le tableau suivant résume la stratégie de commande :

Commande	Description
Contrôle de l'altitude	Asservir à la consigne z_d
Contrôle du lacet	Asservir l'angle de lacet à 0
Contrôle du tangage	Asservir θ à 0 avec contrainte d'asservir x à la consigne x_d
Contrôle du roulis	Asservir Φ à 0 avec contrainte d'asservir y à la consigne y_d

Étant donné que le multirotor a des contraintes physiques sur les amplitudes des entrées de commande, une commande par saturations qui permet de respecter ces contraintes a été proposée dans la littérature.

La fonction de saturation est définie comme :

$$\sigma_M(s) = \begin{cases} M & si\ s > M \\ s & si - M < s < M \\ -M & si\ s < -M \end{cases}$$

Considérons un système à 2 états (x_1, x_2) :

$$\dot{x}_1 = \alpha_1 x_2$$
$$\dot{x}_2 = \alpha_2 u'$$

En faisant le changement de variable suivant :

$$y_1 = x_1$$
$$y_2 = \alpha_1 x_2$$
$$u = \alpha_1 \alpha_2 u'$$

Le système aura la forme suivante :

$$\dot{y}_1 = y_2$$
$$\dot{y}_2 = u$$

Une loi de commande u qui a la forme

$$u = -\sum \sigma_{b_i}(K_i y_i)$$

stabilise asymptotiquement le système et borne chaque état y_i.

3. En se basant sur les équations ci-dessus, écrire la loi de commande sur le lacet.
4. Compléter la loi de commande sur le modèle Matlab/Simulink.

Partie 2: Etude de Commandabilité : (30 min)

La contrôlabilité statique de l'octorotor après l'occurrence des défaillances sur les moteurs est étudiée ci-dessous en utilisant une méthode graphique basée sur la construction de l'« Attainable Control Set » (ACS). Le ACS est un sous-espace en PRTL (Poussée, Roulis, Tangage, Lacet) qui définit les limites en poussée et moments qui peuvent être alloués lorsque les contraintes sur les vitesses des moteurs sont satisfaites. Il est obtenu en faisant correspondre les limites sur les vitesses des moteurs aux limites sur les commandes virtuelles à l'aide de la matrice B :

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -d & -d & -d & -d & d & d & d & d \\ -d & -d & d & d & d & d & -d & -d \\ -K_t/K_f & K_t/k_f & K_t/K_f & -K_t/K_f & -K_t/K_f & K_t/K_f & K_t/K_f & -K_t/K_f \end{bmatrix}$$

d = L*sqrt(2)/2

Pour évaluer la contrôlabilité de l'octorotor le polytope quadri-dimensionnel PRTL est coupé aux conditions de vol nominal : poussée P=mg, moment de lacet L=0. Si l'origine du plan R-T se trouve dans le polygone, alors l'octorotor est contrôlable.

5. En considérant que la poussée d'un moteur est contrainte à être positive et que la poussée maximale d'un moteur est 6.5N, écrire les contraintes sous la forme $A.x \leq b$ dans le cas normal.
6. Utiliser la fonction « con2vert : V = con2vert(A,b)» sur Matlab qui permet de convertir l'ensemble des inégalités de contraintes aux sommets correspondant aux intersections de ces inégalités.
7. Faire un mapping entre les deux espaces des vitesses et des commandes virtuelles à partir de la matrice d'efficacité B : « X = (BV')' ».
8. Construire le polygone P correspondant aux sommets X : « P = Polyhedron(X) ».
9. Pour trouver la contrôlabilité en roulis, tangage et lacet, utiliser la commande « slide » pour découper le polygone P aux plans L = 0 et P = mg : Q = P.slice(4,0) et T = Q.slice(1,mg).
10. Tracer T : T.plot()
11. Est-ce que l'origine (0,0) est à l'intérieur du plan R-T ? Déduire à propos de la contrôlabilité.
12. Répéter les étapes 5-11 quand les moteurs 1 et 2 sont défaillants.

Partie 3: Diagnostic de défauts : (30 min)

Un module de diagnostic à base de modèle est proposé dans cette partie. Pour la génération des résidus, l'approche à base d'un observateur non linéaire est utilisée. Soit $x = [\Phi\ \theta\ \psi\ \dot{\Phi}\ \dot{\theta}\ \dot{\psi}]^t$ le vecteur d'état, et $y = [\Phi\ \theta\ \psi]^t$ le vecteur de sortie. Un observateur de Thau sera implémenté.

13. Ecrire le système représentant l'attitude de l'octorotor sous la forme : $\dot{x}(t) = Ax(t) + Bv + H\big(x(t)\big)$.
14. Ecrire la matrice d'observabilité en utilisant la commande « obsv(A,C) » sur Matlab.
15. Calculer le rang de la matrice d'observabilité en utilisant la commande « rank » sur Matlab et déduire ainsi l'observabilité du système.

Considérons que la partie non linéaire du système représentant l'attitude du multirotor est continument différentiable et localement Lipschitzienne en γ :

$$\|H(x_1) - H(x_2)\| < \gamma\|x_1 - x_2\|$$

Alors un observateur de Thau peut être proposé pour l'estimation des états sous la forme :

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bv(t) + H\big(\hat{x}(t)\big) + K\big(y(t) - \hat{y}(t)\big)$$

$$\hat{y}(t) = C\hat{x}(t)$$

Avec K le gain de l'observateur.

16. Implémenter l'observateur sur Matlab/Simulink.
17. Simuler une défaillance sur le moteur 1.
18. Tester l'observateur dans le cas normal et en cas de défaillance. Déduire.

Partie 4: Rétablissement du système : (40 min)

Le rétablissement de l'octorotor se fait par modification du multiplexage de façon à redistribuer les efforts de commande d'une manière optimale sur les actionneurs sains. Les lois de multiplexage sont déterminées par résolution d'un problème d'optimisation multi-paramétrique hors ligne, et enregistrées dans un ' lookup table '. Le problème d'optimisation est représenté par l'équation ci-dessous :

$$min_u \frac{1}{2} u^T u$$

$$such\ that \begin{matrix} Bu = v \\ u_{min} \leq u \leq u_{max} \end{matrix}$$

avec $u = [f_1 \quad \dots \quad f_8]^T$ est le vecteur des poussées des moteurs et $v = \begin{bmatrix} u_f & \tau_\Phi & \tau_\theta & \tau_\psi \end{bmatrix}^T$.

19. Ecrire la matrice B.
20. Définir les vecteurs u, v et les paramètres y_i représentant les efficacités des moteurs i.

$$v = sdpvar(4,1)$$
$$u = sdpvar(8,1)$$
$$\gamma_i = sdpvar(1,1), \qquad i = 1, \dots, 8.$$
$$a = [v; \gamma_1; \ \gamma_2; \ \dots; \gamma_8]$$

21. Ecrire la fonction objective du problème d'optimisation :

$$Q_u = 1 * eye(8,8)$$
$$obj = (0.5) * u' * Q_u * u$$

22. Ecrire le vecteur des contraintes du problème d'optimisation :

$$F = [B * u == v, 1 \geq \gamma_i, \gamma_i \geq 0, u(i) \geq 0, \gamma_i * 6.5 \geq u(i)]$$

23. Résoudre le problème par la commande solvemp :

```
[sol,diagnostics,aux,Valuefunction,Optimizer] = solvemp(F,obj,[],a,u)
```

Ceci permet de trouver les régions des solutions en fonction des paramètres γ_i et du vecteur de commande v.

24. Trouver la solution particulière en cas du fonctionnement normal

$$assign(a, [mg; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 1])$$
value(Optimizer)

25. Trouver la solution particulière en cas de défaillance du moteur 1 ($\gamma_1 = 0$).
26. Déduire les gains de multiplexage.
27. Tester cette solution sur Simulink en simulant une défaillance sur le moteur 1 et un temps de détection de 0.1s.