Preuve de la stabilité contrôleur d'altitude Crazyflie

Modelling Physics with DAE

February 3, 2021

Outlines

Introduction

Objectifs du projet

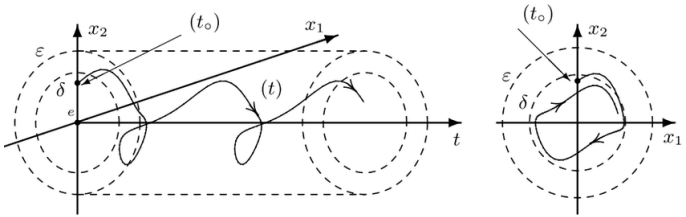
L'objectif principale de ce projet est de montrer la stabilité du contrôleur d'altitude du drone Crazyflie.

Pour cela, il faudra :

- définir un moyen de vérifier la stabilité d'un système,
- modéliser ce contrôleur,
- démontrer la stabilité du contrôleur d'altitude du drone.

Stabilité de Lyapunov

Un système est stable pour un point d'équilibre x_e au sens de Lyapunov si tout mouvement d'un système issu d'un voisinage suffisamment petit de x_e demeure au voisinage de ce point.



Stabilité de Lyapunov

Théorème de stabilité de Lyapunov:

Soit un système décrit par une équation différentielle du type $\dot{x} = f(x, t)$.
SPdG : le point d'équilibre auquel on s'intéresse est $x_e = 0$.

- S'il existe une fonction $V : (x, t) \to V(x, t)$ continûment différentiable définie positive telle que $\dot{V}$ est semi-définie négative et que $V(x, t) \xrightarrow[||x|| \to \infty]{} \infty$ alors 0 est un point d'équilibre stable.

NB : Ces conditions rappellent celles que l'énergie potentielle doit vérifer pour qu'il y ait stabilité d'un point d'équilibre d'un système physique.

Modèle

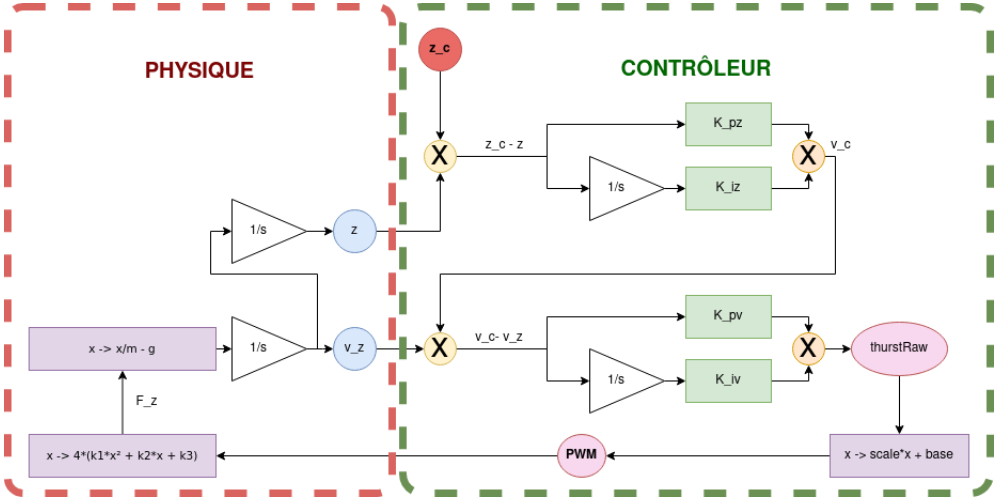
Modélisation simple du Crazyflie

- Principe fondamental de la dynamique : $\sum F = m\dot{v_z}$ donc $\dot{v_z} = \frac{F_z}{m} - g$
- Force générée par chaque hélice : $T_i = C_T\omega_i^2$
- Par symétrie : $F_z = 4 * C_T\omega^2$.
- Relation entre signal moteur et vitesse angulaire : $\omega = C_1 * PWM + C_2$
- On a donc : $F_z = (4C_TC_1^2) * PWM^2 + (8C_TC_1C_2) * PWM + 4C_TC_2^2$

Nous utiliserons cependant les valeurs trouvées expérimentalement dans [1] :

$$F_z = 4(k_1 * PWM^2 + k_2 * PWM + k_3)$$

Contrôleur d'altitude



Contrôleur d'altitude

Les équations que nous pouvons tirer du code C du firmware du Crazyflie sont les suivantes :

- $v_c = K_{pz}(z_c - z) + K_{iz} \int (z_c - z) dt$
- $thrustRaw = K_{pv}(v_c - v_z) + K_{iv} \int (v_c - v_z) dt$
- $PWM = \mathrm{Scale} * thrustRaw + \mathrm{Base}$

Système global

Ajout de : $u_1 = \int(2(z_c - z) + 0.5\int(z_c - z)dt - v_z)dt$ et $u_2 = \int(z_c - z)dt$
Nouvelles constantes : $K_p = \text{Scale} * K_{pv}$ et $K_i = \text{Scale} * K_{iv}$
Altitude centrée en zéro : $z \leftarrow z - z_c$

$$\begin{cases} \dot{z} = v_z \\ \dot{v_z} = \frac{4k_1}{m} * PWM^2 + \frac{4k_2}{m} * PWM + \frac{4k_3}{m} - g \\ \dot{u_1} = -2z + 0.5u_2 - v_z \\ \dot{u_2} = -z \end{cases}$$
$$PWM = K_p\dot{u_1} + K_i u_1 + \text{Base}$$

À l'équilibre $\dot{v_z} = 0$ donc $PWM_e \simeq 37443.66060930098$ d'après calcul numérique.

Linéarisation autour de l'équilibre

Expansion de Taylor au premier ordre :

$$\begin{cases} \Delta\dot{z} = \Delta v_z \\ \Delta\dot{v_z} = \frac{8k_1}{m} * PWM_e * \Delta PWM + \frac{4k_2}{m} * \Delta PWM \\ \Delta\dot{u_1} = -2\Delta z + 0.5\Delta u_2 - \Delta v_z \\ \Delta\dot{u_2} = -\Delta z \\ \Delta PWM = K_p\Delta\dot{u_1} + K_i\Delta u_1 \end{cases}$$

Nouveau système linéaire

Si on note $\begin{cases} \Delta x = [\Delta z \ \Delta v_z \ \Delta u_1 \ \Delta u_2]^t \\ \Delta \dot{x} = [\Delta \dot{z} \ \Delta \dot{v_z} \ \Delta \dot{u_1} \ \Delta \dot{u_2}]^t \end{cases}$ alors nous avons :

$$\begin{cases} \Delta \dot{x} = A * \Delta x \\ \Delta PWM = B * \Delta x \end{cases}$$

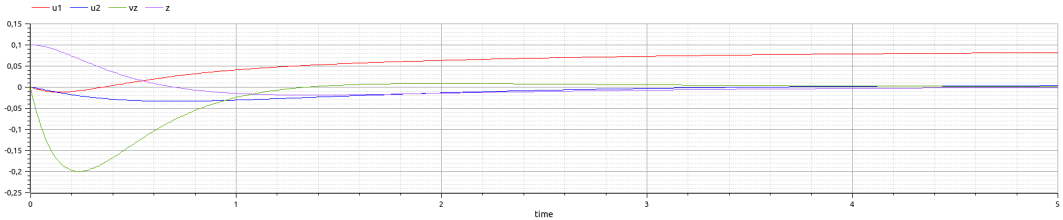
$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -2K_p * \alpha & -K_p * \alpha & K_i * \alpha & \frac{K_p}{2} * \alpha \\ -2 & -1 & 0 & 0.5 \\ -1 & 0 & 0 & 0 \end{bmatrix} \text{ avec } \alpha = \frac{8(2k_1 * PWM_e + k_2)}{m}$$

$$B = \begin{bmatrix} -2 * K_p & -K_p & K_i & 0.5 * K_p \end{bmatrix}$$

Simulation

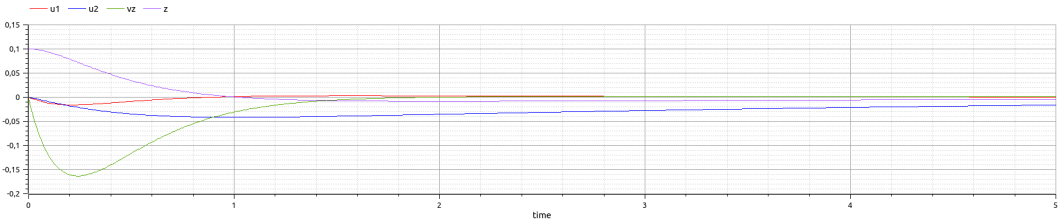
Sytème non linéaire avec Modelica

Le contrôleur d'altitude semble stable :



Sytème linéarisé avec Modelica

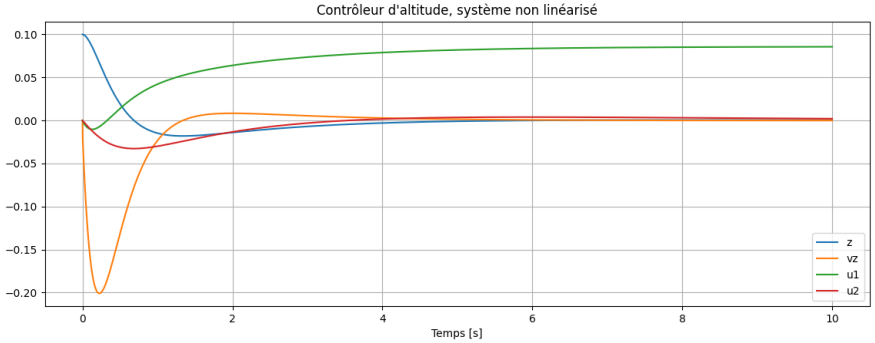
Le système linéarisé ressemble à l'original. Seul u_1 diffère :



On peut donc espérer trouver une fonction de Lyapunov pour ce système.

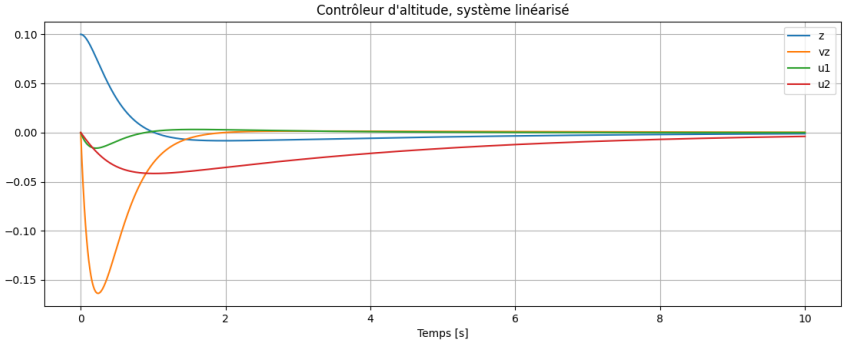
Simulation python pour le système non continue

On peut voir dans le code du firmware que le stabilisateur boucle à 1kHz ou 500Hz. J'ai donc réalisé une simulation python discrétisée d'un pas de 0.002s.



Simulation python pour le système non continue linéarisé

De même, pour le système linéarisé :



On peut donc conclure que la continuité du système est légitime.

Équation de Lyapunov

On utilise la linéarisation à l'équilibre pour construire une fonction de Lyapunov.
Supposons P et Q symétriques et définie positives telles que :

$$A^t P + PA = -Q$$

Alors la fonction $V : x \to x^t Px$ est définie positive et $-\dot{V} : x \to -\nabla VAx = x^t Qx$ aussi.
Nous devons alors résoudre cette équation dite de Lyapunov.

Équation de Lyapunov

$Q = I$ implique que

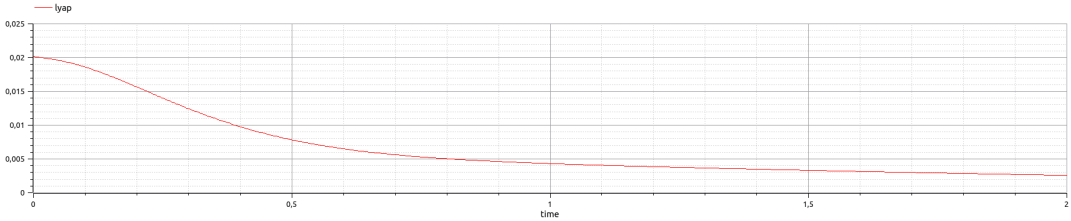
$$P = \begin{bmatrix} 2.01614335 & 0.08318215 & 0.11508536 & -1.29159107 \\ 0.08318215 & 0.07159635 & -0.08878898 & -0.04601649 \\ 0.11508536 & -0.08878898 & 1.35160137 & -0.56810997 \\ -1.29159107 & -0.04601649 & -0.56810997 & 2.44789777 \end{bmatrix}.$$

Les valeurs propres de P sont 3.65618273, 1.40504381, 0.7653299 et 0.0606824. Donc la matrice est bien définie positive.

Le système linéarisé est donc stable au sens de Lyapunov.

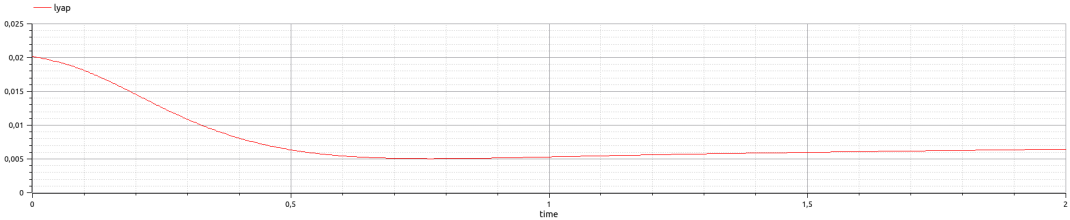
Simulation de la fonction de Lyapunov dans le système linéarisé

On peut voir que la fonction est **potentiellement** candidate. Je n'ai pas encore pu le vérifier, problème dans mon code python ($A^tP + PA = -Q$ non vérifié).



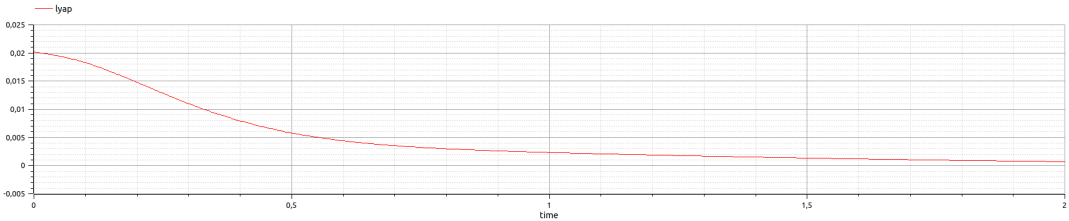
Simulation de la fonction de Lyapunov dans le système initial

On peut voir une dynamique similaire avec le système initial. Cependant on peut voir qu'à l'équilibre du système, la fonction candidate augmente et **n'est donc pas une fonction de Lyapunov**.



Simulation de la fonction de Lyapunov modifiée dans le système initial

On se rappelle que l'état u_1 différait beaucoup entre le système linéarisé et le système initial. J'ai enlevé la dernière ligne de P afin d'enlever une partie des coefficients liés à u_1. La fonction candidate **semble être une fonction de Lyapunov**. À prouver...



Conclusion

Récapitulatif

Lors de ce projet, nous avons :

- vu comment caractériser une stabilité d'un système
- modélisé le contrôleur d'altitude du Crazyflie
- simulé ce contrôleur d'altitude pour
- linéarisé ce système pour le simplifier
- trouver une fonction de Lyapunov pour le système linéarisé
- trouver une fonction candidate adéquate pour le système initial

Pour le rapport, il reste encore :

- trouver des candidats plus parlant
- prouver les candidats
- déterminer si le candidat invariant du système linéarisé est bien un invariant

Références

Références

[1] J. Förster. "System Identification of the Crazyflie 2.0 Nano Quadrocopter". In: (2015).

Je vous remercie.

Annexe

Dynamique initale

```
class Dynamique_nonLineaire
  Real z(start = 0.1,fixed=true);
  Real vz(fixed=true);
  Real u1(fixed=true);
  Real u2(fixed=true);
  Real pwm();
equation
  der(u2) = -z;
  der(z) = vz;
  pwm = 25000*der(u1) + 15000*u1 + 36000;
  der(vz) = 4/0.028*(2.130295*10^(-11)*pwm*pwm+1.032633*10^(-6)*pwm+5.484560*10^(-4))-9.81;
  der(u1) = -2*z+0.5*u2-vz;
end Dynamique_nonLineaire;
```

Dynamique linéarisée

```
class Dynamique_Lineaire
  Real z(start = 0.1,fixed=true);
  Real vz(fixed=true);
  Real u1(fixed=true);
  Real u2(fixed=true);
equation
  der(z)  = vz;
  der(vz) = -18.771099*z-9.3855495*vz+5.6313297*u1+4.69277475*u2;
  der(u1) = -2*z-vz+0.5*u2;
  der(u2) = -z;
end Dynamique_Lineaire;
```

Lyapunov dans système linéarisé

```
class Dynamique_Lineaire_Lyapunov
  Real z(start = 0.1,fixed=true);
  Real vz(fixed=true);
  Real u1(fixed=true);
  Real u2(fixed=true);
  Real lyap();
equation
  der(z)  = vz;
  der(vz) = -18.771099*z-9.3855495*vz+5.6313297*u1+4.69277475*u2;
  der(u1) = -2*z-vz+0.5*u2;
  der(u2) = -z;
  lyap = (2.01614335*z+0.08318215*vz+0.11508536*u1-1.29159107*u2)*z +
 (0.08318215*z+0.07159635*vz-0.08878898*u1-0.04601649*u2)*vz +
 (0.11508536*z-0.08878898*vz+1.35160137*u1-0.56810997*u2)*u1 +
 (-1.29159107*z-0.04601649*vz-0.56810997*u1+2.44789777*u2)*u2;
end Dynamique_Lineaire_Lyapunov;
```

Lyapunov modifiée pour le système initial

```
class Dynamique_nonLineaire_Lyapunov_Modifiee
  Real z(start = 0.1,fixed=true);
  Real vz(fixed=true);
  Real u1(fixed=true);
  Real u2(fixed=true);
  Real pwm();
  Real lyap();
equation
  der(u2) = -z;
  der(z) = vz;
  pwm = 25000*der(u1) + 15000*u1 + 36000;
  der(vz) =
 4/0.028*(2.130295*10^(-11)*pwm*pwm+1.032633*10^(-6)*pwm+5.484560*10^(-4))-9.81;
  der(u1) = -2*z+0.5*u2-vz;
  lyap = (2.01614335*z+0.08318215*vz+0.11508536*u1-1.29159107*u2)*z +
 (0.08318215*z+0.07159635*vz-0.08878898*u1-0.04601649*u2)*vz +
 (-1.29159107*z-0.04601649*vz-0.56810997*u1+2.44789777*u2)*u2;
end Dynamique_nonLineaire_Lyapunov_Modifiee;
```

Valeurs numériques des constantes du Crazyflie

m	0.028
k_1	$2.130295e-11$
k_2	$1.032633e-6$
k_3	$5.484560e-4$
Scale	1000
Base	36000
K_{pz}	2
K_{iz}	0.5
K_{pv}	25
K_{iv}	15
K_p	25000
K_i	15000

Les constantes présentées sont en SI.

Nous pouvons remarquer que k_3 et k_2 paraissent négligeables devant k_1 qui n'est lui même pas bien grand. Cependant les valeurs de PWM sont conséquentes, ce qui rend les k_i non dispensables.

Rappel : $PWM_e \simeq 37443.66060930098$