IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

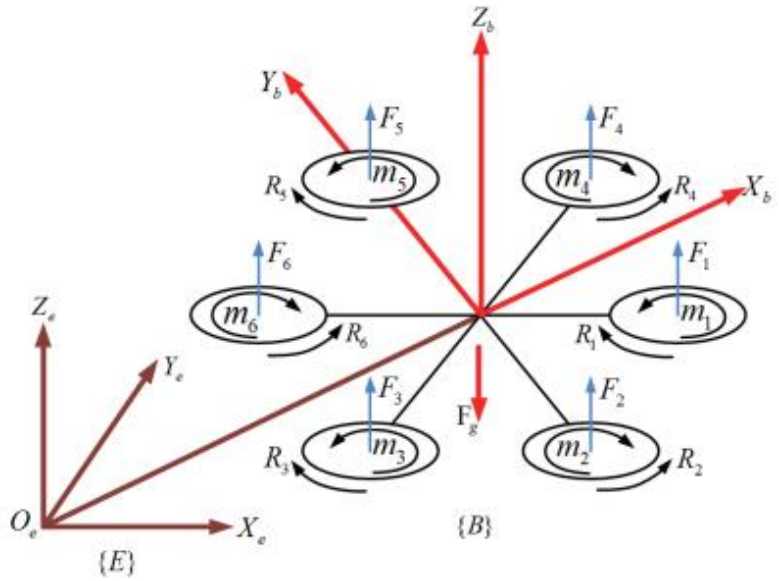
Compte Rendu

Drone hexarotor

Dynamique des robots - UE A

MAKNI Oussama

Compte Rendu de Projet : Modélisation et Simulation d’un Drone Hexarotor



Introduction

Ce projet vise à modéliser et simuler le comportement dynamique d'un drone à 6 rotors à l'aide de MATLAB. Nous avons construit un modèle dynamique et géométrique du drone en utilisant dans un premier temps les angles d'Euler pour représenter la rotation et l'orientation dans l'espace. Dans les versions ultérieures, les quaternions seront également représentés comme moyen d'améliorer la précision. Grâce à cette simulation, le comportement dynamique du drone peut être compris et les algorithmes sous-jacents peuvent être vérifiés avant leur mise en œuvre réelle.

Etapes du Projet

1. Créer un modèle mathématique des équations de mouvement d’un drone à six rotors.
2. Implémenter ces équations sous forme de code MATLAB pour simuler les positions, les vitesses, et les angles du drone.
3. Réaliser une animation 3D pour visualiser les dynamiques du drone.

Utilisation de la Convention d'Angles d'Euler ZYX

Dans ce projet de modélisation de drone à 6 rotors, nous avons utilisé la convention d'angles d'Euler ZYX pour décrire les rotations du drone. Cette convention implique une séquence de rotations autour des axes z, y, puis x. Concrètement, cela signifie que nous effectuons d'abord une rotation autour de l'axe z (yaw), suivie d'une rotation autour de l'axe y (pitch), et enfin une rotation autour de l'axe x (roll).

Nous n'avons pas utilisé d'autres conventions d'angles d'Euler, car elles peuvent introduire des ambiguïtés et des complications supplémentaires dans les calculs. Par exemple, certaines séquences peuvent conduire à l’alignement de deux des trois axes de rotation, rendant impossible la détermination de l'orientation du drone La convention ZYX évite ces problèmes et offre une solution robuste et intuitive pour modéliser les rotations du drone.

$$R = Rz(\psi)Ry(\theta)Rx(\phi)$$

$$Rz(\psi) = \begin{matrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{matrix} \quad Ry(\theta) = \begin{matrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{matrix} \quad Rx(\phi) = \begin{matrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{matrix}$$

$$R = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

Modélisation Dynamique

Les forces de portance

Les forces générées par les rotors sont modélisées par une constante de portance (β) multipliée par le carré de la vitesse de rotation des rotors.

$Pi = \beta i * ui^2$

Les couples générés par les rotors

Les couples générés par les rotors sont modélisés par une constante de couple spécifique au rotor (α) multipliée par le carré de la vitesse de rotation des rotors

$Pi = \alpha i * ui^2$

D'après les équations de Newton-Euler on a démontré :

$$m \cdot \left(\dot{V} + \Omega \times V\right) = \sum_{i=1}^{6} P_i \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - R^T \cdot \begin{pmatrix} 0 \\ 0 \\ m \cdot g \end{pmatrix} \quad (1)$$

$$I \cdot \dot{\Omega} + \Omega \times (I \cdot \Omega) = \sum_{i=1}^{6} -l_i \cdot P_i \cdot \begin{pmatrix} -\sin\left(\frac{2\pi(i-1)}{6}\right) \\ \cos\left(\frac{2\pi(i-1)}{6}\right) \\ 0 \end{pmatrix} + C_i \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (2)$$

***I* donné par :**

$$I = \begin{pmatrix} \frac{1}{12}m\left(3R^2 + h^2\right) & 0 & 0 \\ 0 & \frac{1}{12}m\left(3R^2 + h^2\right) & 0 \\ 0 & 0 & \frac{1}{2}mR^2 \end{pmatrix}$$

Le vecteur d'État :

$$\mathbf{x} = \begin{pmatrix} V \\ \Omega \\ \phi \\ \theta \\ \psi \end{pmatrix} \quad \text{Ave} \quad \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

Et d'après (1) et (2) on a démontré :

$$\dot{V} = \frac{1}{m}\left(\sum_{i=1}^{6} P_i \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - R^T \cdot \begin{pmatrix} 0 \\ 0 \\ m \cdot g \end{pmatrix}\right) - \Omega \times V$$

$$\dot{\Omega} = I^{-1} \cdot \left(\sum_{i=1}^{6} \left(-l_i \cdot P_i \cdot \begin{pmatrix} -\sin\left(\frac{2\pi(i-1)}{6}\right) \\ \cos\left(\frac{2\pi(i-1)}{6}\right) \\ 0 \end{pmatrix} + C_i \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\right) - \Omega \times (I \cdot \Omega)\right)$$

Et on note le vecteur de commande :

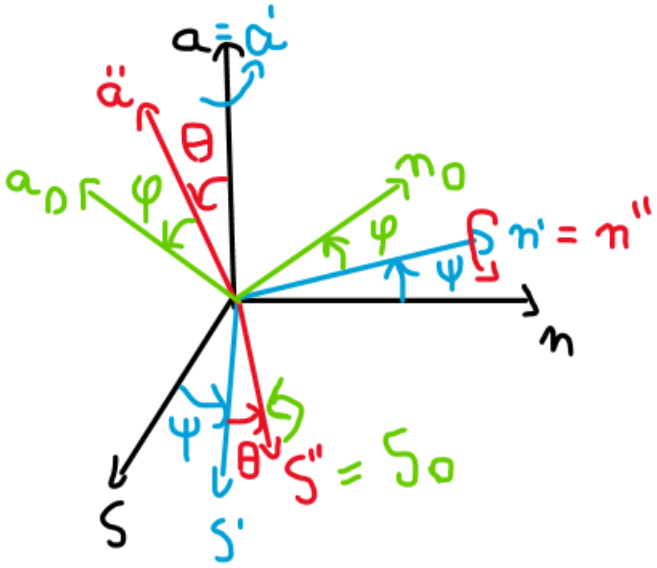
$$\mathbf{U} = \begin{pmatrix} (\dot{u}_1)^2 \\ (\dot{u}_2)^2 \\ (\dot{u}_3)^2 \\ (\dot{u}_4)^2 \\ (\dot{u}_5)^2 \\ (\dot{u}_6)^2 \end{pmatrix}$$

En utilisant la matrice jacobienne on va démontre la variation de ϕ, θ, ψ :

$$\Omega = J(\varphi, \theta, \psi) * \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = J(\varphi, \theta, \psi)^{-1} * \Omega$$

Démarche pour obtenir la matrice jacobienne :

convention d'angles d'Euler Z Y X



$$\Omega = \dot{\phi}\,\vec{S}_0 + \dot{\Theta}\,\vec{n}'' + \dot{\psi}\,\vec{a}'$$

On peut déduire initialement que : $\dot{\varphi} \times \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \Rightarrow R_x(\phi)^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

On peut déduire que : $\dot{\Theta} \times \begin{pmatrix} 0 \\ \cos\phi \\ -\sin\phi \end{pmatrix}$

$$\boldsymbol{R = Ry(\theta) * Rx(\phi)}$$

$$R = \begin{bmatrix} \cos\theta & \sin\phi\sin\theta & \sin\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \Rightarrow R^T = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ \sin\phi\sin\theta & \cos\phi & \sin\phi\cos\theta \\ \sin\theta\cos\phi & -\sin\phi & \cos\phi\cos\theta \end{bmatrix}$$

On peut déduire que : $\dot{\Psi} \times \begin{pmatrix} -\sin\theta \\ \sin\phi\cos\theta \\ \cos\phi\cos\theta \end{pmatrix}$

On peut conclure que : $\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{bmatrix} 1 & 0 & -\sin\phi \\ 0 & \cos\phi & \sin\phi * \cos\theta \\ 0 & -\sin\phi & \cos\phi * \cos\theta \end{bmatrix}^{-1} * \Omega$

Implémentation sur Matlab :

Fonction de Rotation

La fonction get_rotation calcule la matrice de rotation en fonction des angles d'Euler (phi, theta, psi).

Editor - C:\Users\oussa\OneDrive\Desktop\drone\get_rotation.m

+9 sim_quadcopter.m | test.m | test1.m | test2.m | modele_dynamique.m | simulation_

```
function R = get_rotation(phi,theta,psi)

R_x = [1    0       0; ...
       0  cos(phi) -sin(phi); ...
       0  sin(phi) cos(phi)];

R_y = [cos(theta)  0   sin(theta); ...
       0           1         0; ...
      -sin(theta) 0   cos(theta)];

R_z = [cos(psi) -sin(psi)  0; ...
       sin(psi)  cos(psi)  0; ...
       0           0       1];
R = R_z*R_y*R_x;
end

```

Modèle Dynamique

La fonction modele_dynamique décrit la dynamique du drone, incluant les forces et couples générés par les rotors, ainsi que les équations de mouvement.

Editor - C:\Users\oussa\OneDrive\Desktop\drone\modele_dynamique.m

+9 sim_quadcopter.m | test.m | test1.m | test2.m | modele_dynamique.m

```
function x_dot = modele_dynamique(t,x,u,PARAM)

V = x(1:3);
Omega = x(4:6);
phi = x(7);
theta = x(8);
psi = x(9);

g = PARAM.g;
m = PARAM.m;
l = PARAM.l;
I = PARAM.I;
alpha = PARAM.alpha;
beta = PARAM.beta;

```

```
couple_sum = zeros(3, 1);
C = alpha .* u.^2;
for i = 1:6
    couple_sum = couple_sum -l/2*beta(i)*u(i)^2 *[-sin(2*pi*(i-1)/6);cos(2*pi*(i-1)/6);0] + C(i) *[0;0;1];
end

% Résolution pour Ω̇
Omega_dot = I \ (couple_sum - cross(Omega, I * Omega));

force_sum = zeros(3, 1);
for i = 1:6
    P = [0; 0; beta(i) * u(i)^2];
    force_sum = force_sum + P;
end
% Résolution pour V̇
R = get_rotation(phi,theta,psi);
gravity = [0; 0; m * g];
V_dot = (1/m) * (force_sum - R' * gravity - cross(Omega, V));
```

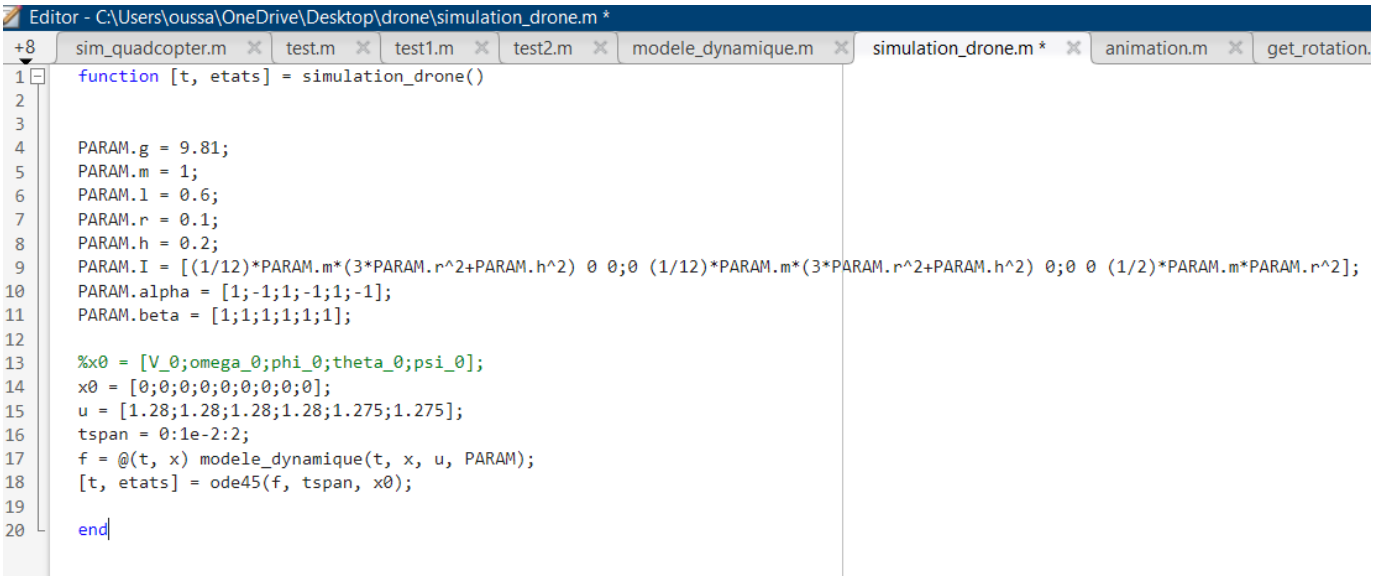
Editor - C:\Users\oussa\OneDrive\Desktop\drone\modele_dynamique.m

```
%Calcul des angles_dot phi theta psi
% Matrice Jacobienne
function J = jacobienne(phi,theta)
R_x = [1    0       0; ...
       0  cos(phi) -sin(phi); ...
       0  sin(phi) cos(phi)];

R_y = [cos(theta)  0   sin(theta); ...
       0           1         0; ...
      -sin(theta) 0   cos(theta)];


R1_t = R_x';
R2 = R_y * R_x;
R2_t = R2';
J = [[1;0;0],R1_t(:,2),R2_t(:,3)];

end
```

```
J = jacobienne(phi,theta);
angle_dot = J \ Omega;

phi_dot = angle_dot(1);
theta_dot = angle_dot(2);
psi_dot = angle_dot(3);
x_dot = [V_dot;Omega_dot;phi_dot;theta_dot;psi_dot];
end
```

Simulation

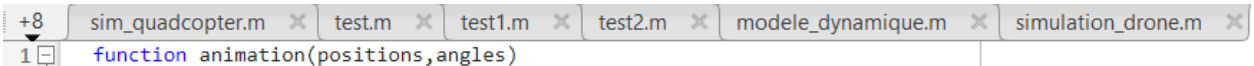
La fonction simulation_drone exécute la simulation du drone en utilisant ode45 pour résoudre les équations différentielles.




```m
function [t, etats] = simulation_drone()


PARAM.g = 9.81;
PARAM.m = 1;
PARAM.l = 0.6;
PARAM.r = 0.1;
PARAM.h = 0.2;
PARAM.I = [(1/12)*PARAM.m*(3*PARAM.r^2+PARAM.h^2) 0 0;0 (1/12)*PARAM.m*(3*PARAM.r^2+PARAM.h^2) 0;0 0 (1/2)*PARAM.m*PARAM.r^2];
PARAM.alpha = [1;-1;1;-1;1;-1];
PARAM.beta = [1;1;1;1;1;1];

%x0 = [V_0;omega_0;phi_0;theta_0;psi_0];
x0 = [0;0;0;0;0;0;0;0;0;0];
u = [1.28;1.28;1.28;1.28;1.275;1.275];
tspan = 0:1e-2:2;
f = @(t, x) modele_dynamique(t, x, u, PARAM);
[t, etats] = ode45(f, tspan, x0);

end
```

Animation

La fonction animation visualise le mouvement du drone en utilisant les positions et angles calculés.




```m
function animation(positions,angles)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
l= 1;
axle_x = [-l/2 0 0;
           l/2 0 0;
           -l/2*cos(pi/3) -l/2*sin(pi/3) 0;
           l/2*cos(pi/3) l/2*sin(pi/3) 0;
           -l/2*cos(pi/3) l/2*sin(pi/3) 0;
           l/2*cos(pi/3) -l/2*sin(pi/3) 0];


r = 0.1*l;
ang = linspace(0,2*pi);
x_circle = r*cos(ang);
y_circle = r*sin(ang);
z_circle = zeros(1,length(ang));
propeller = [x_circle',y_circle',z_circle'];


[p1,q1] = size(propeller);
[p2,q2] = size(axle_x);
[mm,nn] = size(angles);

for ii=1:mm
    x = positions(ii,1);
    y = positions(ii,2);
    z = positions(ii,3);
    phi = angles(ii,1);
    theta = angles(ii,2);
    psi = angles(ii,3);
    R = get_rotation(phi,theta,psi);
```

```
+8 | sim_quadcopter.m | test.m | test1.m | test2.m | modele_dynamique.m | simulation_drone.m | animation.m
33      for i=1:p2
34          r_body = axle_x(i,:)';
35          r_world = R*r_body;
36          new_axle_x(i,:) = r_world';
37      end
38      new_axle_x = [x y z] +new_axle_x;
39
40
41
42      for i=1:p1
43          r_body = propeller(i,:)';
44          r_world = R*r_body;
45          new_propeller(i,:) = r_world';
46      end
47      new_propeller1 = new_axle_x(1, :) + new_propeller;
48      new_propeller2 = new_axle_x(2, :) + new_propeller;
49      new_propeller3 = new_axle_x(3, :) + new_propeller;
50      new_propeller4 = new_axle_x(4, :) + new_propeller;
51      new_propeller5 = new_axle_x(5, :) + new_propeller;
52      new_propeller6 = new_axle_x(6, :) + new_propeller;
```

```
54      line(new_axle_x(1:2, 1), new_axle_x(1:2, 2), new_axle_x(1:2, 3),'Color', 'black', 'Linewidth', 2); hold on;
55      line(new_axle_x(3:4, 1), new_axle_x(3:4, 2), new_axle_x(3:4, 3),'Color', 'black', 'Linewidth', 2); hold on;
56      line(new_axle_x(5:6, 1), new_axle_x(5:6, 2), new_axle_x(5:6, 3),'Color', 'black', 'Linewidth', 2); hold on;
57      patch(new_propeller1(:, 1), new_propeller1(:, 2), new_propeller1(:, 3), 'r');
58      patch(new_propeller2(:, 1), new_propeller2(:, 2), new_propeller2(:, 3), 'g');
59      patch(new_propeller3(:, 1), new_propeller3(:, 2), new_propeller3(:, 3), 'b');
60      patch(new_propeller4(:, 1), new_propeller4(:, 2), new_propeller4(:, 3), 'c');
61      patch(new_propeller5(:, 1), new_propeller5(:, 2), new_propeller5(:, 3), 'm');
62      patch(new_propeller6(:, 1), new_propeller6(:, 2), new_propeller6(:, 3), 'y');
63      axis(1 * [-2.5 2.5 -2.5 2.5 -2.5 2.5]);
64      xlabel('x'); ylabel('y'); zlabel('z');
65      view(3)
66      pause(0.01)
67      if (ii ~= mm)
68          clf
69      end
70  end
71  end
```

Code pour lancer les fonctions :

```
+8 | sim_quadcopter.m | test.m | test1.m | test2.m | modele_dynamique.m | simulation_drone.m | animat
1   clear all;
2   close all;
3   clc;
4   [t, etats] = simulation_drone();
5
6   V = etats(:, 1:3);
7   angles = etats(:, 7:9);
8   positions = cumtrapz(t, V);
9
10  animation(positions, angles);
11
```

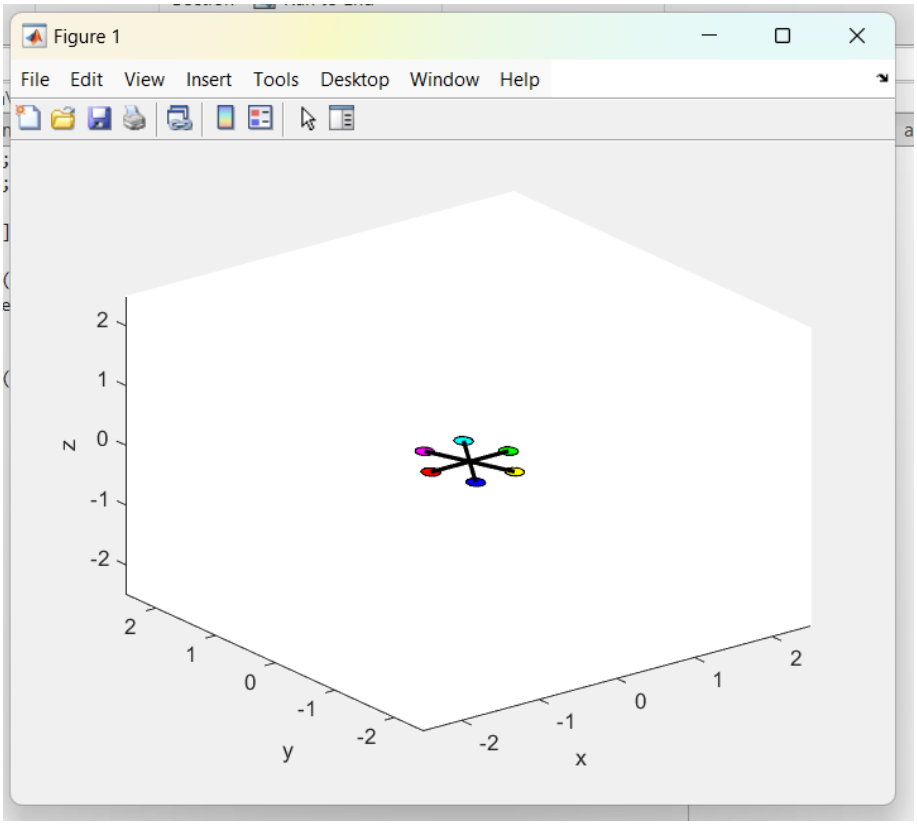


Figure 1
File Edit View Insert Tools Desktop Window Help
z
2
1
0
-1
-2
y
2
1
0
-1
-2
x
2
1
0
-1
-2

Quaternions

On va modéliser la dynamique d'un drone à 6 rotors en utilisant des quaternions pour représenter les rotations. Les quaternions sont utilisés pour éviter les problèmes de verrouillage de cardan associés aux angles d'Euler. Voici une explication détaillée des parties du code liées aux quaternions.

Conversion des angles d'Euler en quaternions

Les angles d'Euler (phi, thêta, psi) doivent être remplacés par un quaternion q= [q0, q1,q2,q3] T qui représente l'orientation.

Multiplication de quaternions

Si nous avons deux quaternions $q_a = [q_{\{a0\}}, q_{\{a1\}}, q_{\{a2\}}, q_{\{a3\}}]$ et $qb = [qb0, qb1, qb2, qb3]$ leur produit q=qa·qb est donné par :

$$q_0 = q_{a0}q_{b0} - q_{a1}q_{b1} - q_{a2}q_{b2} - q_{a3}q_{b3}$$

$$q_1 = q_{a0}q_{b1} + q_{a1}q_{b0} + q_{a2}q_{b3} - q_{a3}q_{b2}$$

$$q_2 = q_{a0}q_{b2} - q_{a1}q_{b3} + q_{a2}q_{b0} + q_{a3}q_{b1}$$

$$q_3 = q_{a0}q_{b3} + q_{a1}q_{b2} - q_{a2}q_{b1} + q_{a3}q_{b0}$$

Quaternions individuels :

1. **Rotation autour de x (ϕ) :**

$$q_x = \left[\cos\left(\frac{\phi}{2}\right), \sin\left(\frac{\phi}{2}\right), 0, 0\right]$$

2. **Rotation autour de y (θ) :**

$$q_y = \left[\cos\left(\frac{\theta}{2}\right), 0, \sin\left(\frac{\theta}{2}\right), 0\right]$$

3. **Rotation autour de z (ψ) :**

$$q_z = \left[\cos\left(\frac{\psi}{2}\right), 0, 0, \sin\left(\frac{\psi}{2}\right)\right]$$

Multiplication qx·qy :

Effectuons la multiplication entre $q_x = [q_{x0}, q_{x1}, 0, 0]$ et $q_y = [q_{y0}, 0, q_{y2}, 0]$:

$$q_{xy0} = q_{x0}q_{y0} - q_{x1} \cdot 0 - 0 \cdot q_{y2} - 0 \cdot 0 = q_{x0}q_{y0}$$

$$q_{xy1} = q_{x0} \cdot 0 + q_{x1}q_{y0} + 0 \cdot 0 - 0 \cdot q_{y2} = q_{x1}q_{y0}$$

$$q_{xy2} = q_{x0}q_{y2} - q_{x1} \cdot 0 + 0 \cdot q_{y0} + 0 \cdot 0 = q_{x0}q_{y2}$$

$$q_{xy3} = q_{x0} \cdot 0 + q_{x1} \cdot 0 - 0 \cdot q_{y0} + 0 \cdot q_{y2} = 0$$

Ainsi :

$$q_{xy} = [q_{x0}q_{y0}, q_{x1}q_{y0}, q_{x0}q_{y2}, 0]$$

Multiplication qxy·qz :

Prenons maintenant $q_{xy} = [q_{xy0}, q_{xy1}, q_{xy2}, q_{xy3}]$ et $q_z = [q_{z0}, 0, 0, q_{z3}]$:

$$q_{final0} = q_{xy0}q_{z0} - q_{xy1} \cdot 0 - q_{xy2} \cdot 0 - q_{xy3}q_{z3} = q_{xy0}q_{z0} - q_{xy3}q_{z3}$$

$$q_{final1} = q_{xy0} \cdot 0 + q_{xy1}q_{z0} + q_{xy2}q_{z3} - q_{xy3} \cdot 0 = q_{xy1}q_{z0} + q_{xy2}q_{z3}$$

$$q_{final2} = q_{xy0} \cdot 0 - q_{xy1}q_{z3} + q_{xy2}q_{z0} + q_{xy3} \cdot 0 = -q_{xy1}q_{z3} + q_{xy2}q_{z0}$$

$$q_{final3} = q_{xy0}q_{z3} + q_{xy1} \cdot 0 - q_{xy2} \cdot 0 + q_{xy3}q_{z0} = q_{xy0}q_{z3} + q_{xy3}q_{z0}$$

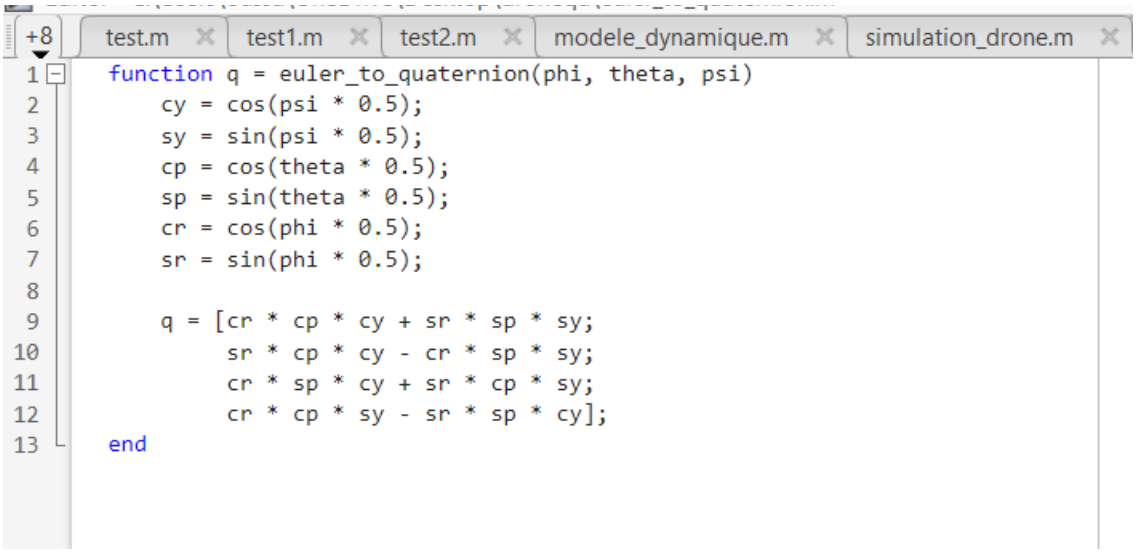
Quaternion final :

En combinant tout cela, on obtient le quaternion final qui représente la rotation autour des trois axes x, y, et z avec les angles ϕ, θ, et ψ respectivement :

$$q_{final} = [q_{final0}, q_{final1}, q_{final2}, q_{final3}]$$

$$\begin{aligned}
q_0 &= \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\
q_1 &= \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) - \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) \\
q_2 &= \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\
q_3 &= \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) - \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2})
\end{aligned}$$

Fonction euler_to_quaternion



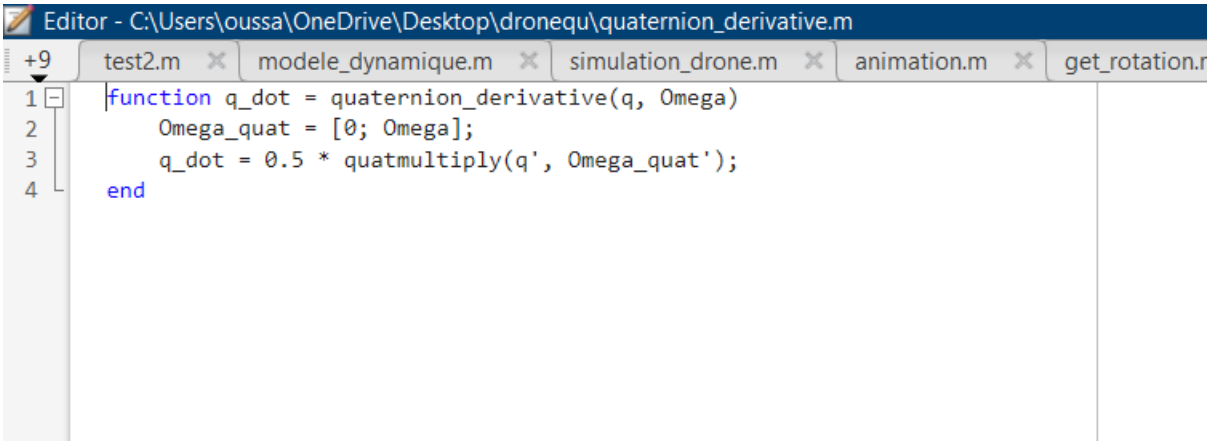
Dérivée des quaternions

La dérivée d'un quaternion (q) en fonction de la vitesse angulaire (Ω) est donnée par :

$$\dot{q} = \frac{1}{2} q \otimes \Omega_{\text{quat}}$$

Où Ωquat = [0, ωx,ωy,ωz] désigne la vitesse angulaire représentée sous forme de quaternion avec ωx, ωy, et ωz correspondant aux composantes de la vitesse angulaire dans chaque direction, et ⊗indique le produit quaternionique.

Fonction quaternion_derivative



Dynamique du modèle

Conversion d'un quaternion en une matrice de rotation :

Un quaternion q = [q0, q1, q2, q3] peut être converti en une matrice de rotation (R) comme suit :

$$R = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix}$$

La fonction quat2rotm en MATLAB convertit un quaternion en une matrice de rotation 3x3

Editor - C:\Users\oussa\OneDrive\Desktop\dronequ\modele_dynamique.m *

+10 | simulation_drone.m | animation.m | get_rotation.m | test.m | euler_to_quaternion.m | quaternion_derivative.m | modele_dyna

```
function x_dot = modele_dynamique(t, x, u, PARAM)

    V = x(1:3);
    Omega = x(4:6);
    q = x(7:10);

    g = PARAM.g;
    m = PARAM.m;
    l = PARAM.l;
    I = PARAM.I;
    alpha = PARAM.alpha;
    beta = PARAM.beta;

    couple_sum = zeros(3, 1);
    C = alpha .* u.^2;
    for i = 1:6
        couple_sum = couple_sum -l/2*beta(i)*u(i)^2 *[-sin(2*pi*(i-1)/6);cos(2*pi*(i-1)/6);0] + C(i) *[0;0;1];
    end

    % Résolution pour Ω̇
    Omega_dot = I \ (couple_sum - cross(Omega, I * Omega));

    force_sum = zeros(3, 1);
    for i = 1:6
        P = [0; 0; beta(i) * u(i)^2];
        force_sum = force_sum + P;
    end

    % Résolution pour V̇
    R = quat2rotm(q');
    gravity = [0; 0; m * g];
    V_dot = (1/m) * (force_sum - R' * gravity - cross(Omega, V));

    q_dot = quaternion_derivative(q, Omega);
    x_dot = [V_dot; Omega_dot; q_dot'];
end
```

Simulation du drone

Editor - C:\Users\oussa\OneDrive\Desktop\dronequ\simulation_drone.m *

+9 animation.m | get_rotation.m | test.m | euler_to_quaternion.m | quaternion_derivative.m | modele_dynamique.m | animate.m | simul

```m
function [t, etats] = simulation_drone()
    PARAM.g = 9.81;
    PARAM.m = 1;
    PARAM.l = 1;
    PARAM.r = 0.1;
    PARAM.h = 0.2;
    PARAM.I = [(1/12)*PARAM.m*(3*PARAM.r^2+PARAM.h^2) 0 0; 0 (1/12)*PARAM.m*(3*PARAM.r^2+PARAM.h^2) 0; 0 0 (1/2)*PARAM.m*PARAM.r^2];
    PARAM.alpha = [1; -1; 1; -1; 1; -1];
    PARAM.beta = [1; 1; 1; 1; 1; 1];

    % x0 = [V_0; omega_0; phi_0; theta_0; psi_0];
    V_0 = [0; 0; 0];
    Omega_0 = [0; 0; 0];
    q_0 = euler_to_quaternion(0, 0, 0);
    x0 = [V_0; Omega_0; q_0];

    u = [1.28; 1.28; 1.28; 1.28; 1.275; 1.275];
    tspan = 0:1e-2:2;
    f = @(t, x) modele_dynamique(t, x, u, PARAM);
    [t, etats] = ode45(f, tspan, x0);
end
```

Animation

+9 get_rotation.m | test.m | euler_to_quaternion.m | quaternion_derivative.m | modele_dynamique.m | animatic

```m
function animation(positions, quaternions)
    l=1;
    axle_x = [-l/2 0 0;
              l/2 0 0;
              -l/2*cos(pi/3) -l/2*sin(pi/3) 0;
              l/2*cos(pi/3) l/2*sin(pi/3) 0;
              -l/2*cos(pi/3) l/2*sin(pi/3) 0;
              l/2*cos(pi/3) -l/2*sin(pi/3) 0];

    r = 0.1 * l;
    ang = linspace(0, 2 * pi);
    x_circle = r * cos(ang);
    y_circle = r * sin(ang);
    z_circle = zeros(1, length(ang));
    propeller = [x_circle', y_circle', z_circle'];

    [p1, q1] = size(propeller);
    [p2, q2] = size(axle_x);
    [mm, nn] = size(quaternions);

    for ii = 1:mm
        x = positions(ii, 1);
        y = positions(ii, 2);
        z = positions(ii, 3);
        q = quaternions(ii, :);
        R = quat2rotm(q);

        for i = 1:p2
            r_body = axle_x(i, :)';
            r_world = R * r_body;
            new_axle_x(i, :) = r_world';
        end
        new_axle_x = [x y z] + new_axle_x;
```

```
        for i = 1:p1
            r_body = propeller(i, :)';
            r_world = R * r_body;
            new_propeller(i, :) = r_world';
        end
        new_propeller1 = new_axle_x(1, :) + new_propeller;
        new_propeller2 = new_axle_x(2, :) + new_propeller;
        new_propeller3 = new_axle_x(3, :) + new_propeller;
        new_propeller4 = new_axle_x(4, :) + new_propeller;
        new_propeller5 = new_axle_x(5, :) + new_propeller;
        new_propeller6 = new_axle_x(6, :) + new_propeller;

        line(new_axle_x(1:2, 1), new_axle_x(1:2, 2), new_axle_x(1:2, 3), 'Color', 'black', 'Linewidth', 2); hold on;
        line(new_axle_x(3:4, 1), new_axle_x(3:4, 2), new_axle_x(3:4, 3), 'Color', 'black', 'Linewidth', 2); hold on;
        line(new_axle_x(5:6, 1), new_axle_x(5:6, 2), new_axle_x(5:6, 3), 'Color', 'black', 'Linewidth', 2); hold on;
        patch(new_propeller1(:, 1), new_propeller1(:, 2), new_propeller1(:, 3), 'r');
        patch(new_propeller2(:, 1), new_propeller2(:, 2), new_propeller2(:, 3), 'g');
        patch(new_propeller3(:, 1), new_propeller3(:, 2), new_propeller3(:, 3), 'b');
        patch(new_propeller4(:, 1), new_propeller4(:, 2), new_propeller4(:, 3), 'c');
        patch(new_propeller5(:, 1), new_propeller5(:, 2), new_propeller5(:, 3), 'm');
        patch(new_propeller6(:, 1), new_propeller6(:, 2), new_propeller6(:, 3), 'y');
        axis(1 * [-2.5 2.5 -2.5 2.5 -2.5 2.5]);
        xlabel('x'); ylabel('y'); zlabel('z');
        view(3)
        pause(0.01)
        if (ii ~= mm)
            clf
        end
    end
end
```

Conclusion

Dans ce projet, nous avons exploré deux méthodes de représentation de la rotation des drones : les angles d'Euler (convention ZYX) et les quaternions. Les angles d'Euler simples et intuitifs sont parfaits pour les calculs de base, mais leur principale limitation (verrouillage du cardan) peut s'avérer problématique dans les scènes complexes. Les quaternions, quant à eux, fournissent des solutions plus fluides et plus précises, ce qui les rend idéaux pour les simulations avancées ou les applications en temps réel. Ce passage aux quaternions représente une avancée clé qui allie performances et robustesse pour mieux répondre aux exigences des systèmes dynamiques modernes.