Universidade Federal de Uberlândia

Faculdade de Engenharia Mecânica
Sistemas Digitais para Mecatrônica

Quadricóptero em espaço 2D

André Tommasello Ramos - 11511EMT025
Gabriel Renato Oliveira Alves - 11621EMT007
Nicolas Bruno Santos Pereira - 11621EMT013
Ricardo Henrique da Silva Assis - 11611EMT013
Victor Spini Paranaiba - 11611EMT005
Professor: Dr. Éder Alves de Moura

Uberlândia, 4 de outubro de 2021

Sumário

1 Introdução

Controlando um quadricóptero no espaço 2D. A simulação comanda o quadricóptero para vários pontos de passagem a cada poucos segundos.

A trajetória é calculada usando diferentes geradores de trajetória: snap, jerk, aceleração e velocidade mínimos. Após o término de cada trajetória, alguns segundos se passarão antes que o quadricóptero se mova para o próximo gerador de trajetória. Todas as trajetórias são traçadas com antecedência usando vários tons de cinza. A trajetória real é exibida em amarelo. Os pontos de trajetória são representados em vermelho como pequenas cruzes.

Além disso a simulação conta com um modo manual, onde a posição do quadricóptero é controlada a partir das teclas direcionais do teclado em tempo real.

Este link leva a um breve vídeo explicando sucintamente o código do simulador

2 Metodologia

Este relatório descreve como o simulador é construído usando a teoria de controle moderna e design de controlador de espaço de estado.

2.1 Requerimentos

Para o desenvolvimento do projeto foi necessário um versão atualizada do Python, no caso a utilizada foi a 3.8.

Algumas bibliotecas foram utilizadas ao longo da implementação dentre elas vale destaque:

- `pymunk` uma biblioteca de física 2d que pode ser usada para simular dinâmicas de corpo rígido 2D no Python (BLOMQVIST, 2007).
- `pyglet` uma biblioteca multimídia para Python, destinada ao desenvolvimento de jogos e outros aplicativos visualmente ricos. Ela oferece suporte a janelas, manipulação de eventos da interface do usuário, joysticks, gráficos OpenGL, carregamento de imagens e vídeos e reprodução de sons e música (PYGLET..., 2016).
- `numpy` uma biblioteca Python que fornece um objeto de matriz multidimensional, vários objetos derivados (como matrizes e matrizes mascaradas) e uma variedade de rotinas para operações rápidas em matrizes, incluindo matemática, lógica, manipulação de forma, classificação, seleção, E/S, transformadas discretas de Fourier, álgebra linear básica, operações estatísticas básicas, simulação aleatória, etc (HARRIS et al., 2020).
- `matplotlib` uma biblioteca abrangente para a criação de visualizações estáticas, animadas e interativas em Python (HUNTER, 2007).
- `sympy` uma biblioteca Python para matemática simbólica (MEURER et al., 2017).

Dentre essas, bibliotecas nativas como `math`, `logging` e `enum` também foram utilizadas.

2.2 Descrição do sistema

A tarefa de controle consiste em rastrear referências para as posições y e z mantendo-se θ constante e igual a zero. Com esse propósito, manipulam-se as entradas u_1 e u_2 que são função das forças F_1 e F_2. Essas variáveis de entrada encontram-se apontadas na Figura 2.1.

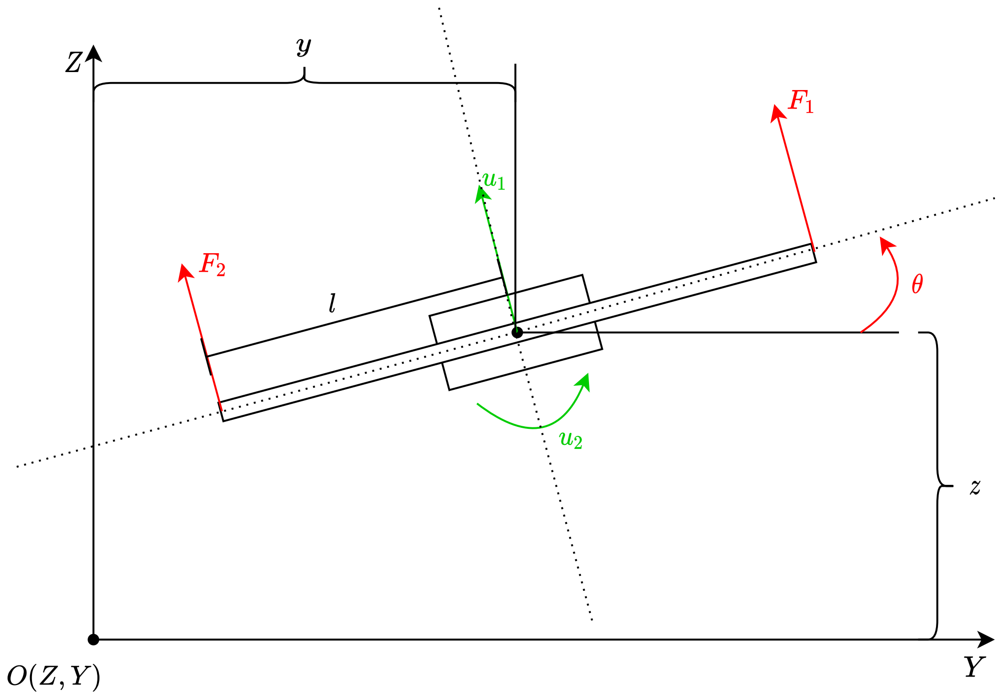

Figura 2.1: Ilustração das variáveis controladas u_1 e u_2

O modelo não linear a tempo contínuo do sistema é definido como

$$m\ddot{y}(t) = -(F_1(t) + F_2(t))\sin\theta(t) \tag{2.1}$$

$$m\ddot{z}(t) = (F_1(t) + F_2(t)) - mg \tag{2.2}$$

$$I\ddot{\theta}(t) = l(F_1(t) - F_2(t)) \tag{2.3}$$

Sendo a representação do modelo linear no espaço de estados a tempo contínuo desse sistema

$$\dot{x}(t) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{\bar{u}_1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{m} & 0 \\ 0 & \frac{1}{I} \end{bmatrix} u(t) \tag{2.4}$$

$$\tag{2.5}$$

em que $x = [y \;\; z \;\; \theta \;\; \dot{y} \;\; \dot{z} \;\; \dot{\theta}]^T$ e $u = [\tilde{u}_1 \;\; \tilde{u}_2]^T$. Os valores e as descrições das constantes encontram-se na Tabela 2.1. O símbolo "~" indica valores relativos aos valores de equilíbrio.

O controle total aplicado no sistema é então definido como

$$\begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = u(t) + \begin{bmatrix} \bar{u}_1 \\ \bar{u}_2 \end{bmatrix} \tag{2.6}$$

em que $[\bar{u}_1 \ \bar{u}_2]^T$ são os valores de equilíbrio para as entradas definidos como

$$\begin{bmatrix} \bar{u}_1 \\ \bar{u}_2 \end{bmatrix} = \begin{bmatrix} mg \\ 0 \end{bmatrix} \tag{2.7}$$

Por fim a relação entre as entradas de controle e as forças geradas pelos motores definida como

$$\begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} F_1(t) + F_2(t) \\ l(F_1(t) - F_2(t)) \end{bmatrix} \tag{2.8}$$

Os valores e as descrições das constantes encontram-se na Tabela 2.1

Tabela 2.1: Descrição dos parâmetros do modelo.

Constante	Significado	Valor
I	Momento de inércia total	0,001 kg$\cdot m^2$
l	Comprimento dos braços do quadricóptero	0,25 m
m	massa do quadricóptero	0,25 Kg
g	aceleração gravitacional	9,8 m^2/s

2.3 Sistema de controle

Consiste em um controlador que paire o quadricóptero em uma determinada posição. Neste caso, as acelerações e velocidades desejadas serão zero em regime permanente. Ignorando qualquer erro, isto é, foi presumido que o quadricóptero não será afetado por fontes externas, como o vento, por isso não será considerada ação integral na equação do controlador.

Do modelo (2.4) temos que as dinâmicas lineares são

$$\ddot{y} = -g\theta \tag{2.9}$$

$$\ddot{z} = \frac{\tilde{u}_1}{m} \tag{2.10}$$

$$\ddot{\theta} = \frac{\tilde{u}_2}{I} \tag{2.11}$$

Para criar as equações de controle PD, aplica-se um ganho proporcional ao erro de distância e um ganho derivado ao erro de velocidade e ainda considerando realimentação negativa, as acelerações são definidas como

$$\ddot{y}(t) = \ddot{y}_r(t) + K_{d,y}(\dot{y}_r(t) - \dot{y}(t)) + K_{p,y}(y_r(t) - y(t)) \tag{2.12}$$

$$\ddot{z}(t) = \ddot{z}_r(t) + K_{d,z}(\dot{z}_r(t) - \dot{z}(t)) + K_{p,z}(z_r(t) - z(t)) \tag{2.13}$$

$$\ddot{\theta}(t) = K_{d,\theta}(\dot{\theta}_r(t) - \dot{\theta}(t)) + K_{p,\theta}(\theta_r(t) - \theta(t)) \tag{2.14}$$

em que o sub-escrito r indica valores de referência.

Então substituindo as equações das acelerações nas dinâmicas lineares é possível definir a equação das entradas parciais

$$\theta_r(t) = -\frac{1}{g}(\ddot{y}_r(t) + K_{d,y}(\dot{y}_r(t) - \dot{y}(t)) + K_{p,y}(y_r(t) - y(t))) \tag{2.15}$$

$$\tilde{u}_1(t) = m(\ddot{z}_r(t) + K_{d,z}(\dot{z}_r(t) - \dot{z}(t)) + K_{p,z}(z_r(t) - z(t))) \tag{2.16}$$

$$\tilde{u}_2(t) = \ddot{\theta}_r(t) + K_{d,\theta}(\dot{\theta}_r(t) - \dot{\theta}(t)) + K_{p,\theta}(\theta_r(t) - \theta(t)) \tag{2.17}$$

Obtêm-se as equações do controle total somando os valores de equilíbrio nas equações das entradas parciais

$$\theta_r(t) = -\frac{1}{g}(\ddot{y}_r(t) + K_{d,y}(\dot{y}_r(t) - \dot{y}(t)) + K_{p,y}(y_r(t) - y(t))) \tag{2.18}$$

$$u_1(t) = m(g + \ddot{z}_r(t) + K_{d,z}(\dot{z}_r(t) - \dot{z}(t)) + K_{p,z}(z_r(t) - z(t))) \tag{2.19}$$

$$u_2(t) = \ddot{\theta}_r(t) + K_{d,\theta}(\dot{\theta}_r(t) - \dot{\theta}(t)) + K_{p,\theta}(\theta_r(t) - \theta(t)) \tag{2.20}$$

2.4 Resposta ao degrau unitário

Para ajustar os valores dos ganhos realizou-se simulações em relação ao modelo não linear considerando como referencia um degrau unitário para as posições y e z, na Figura 2.2 são apresentados os estados quando o quadricóptero está se movendo de (0,0) para (1,1).

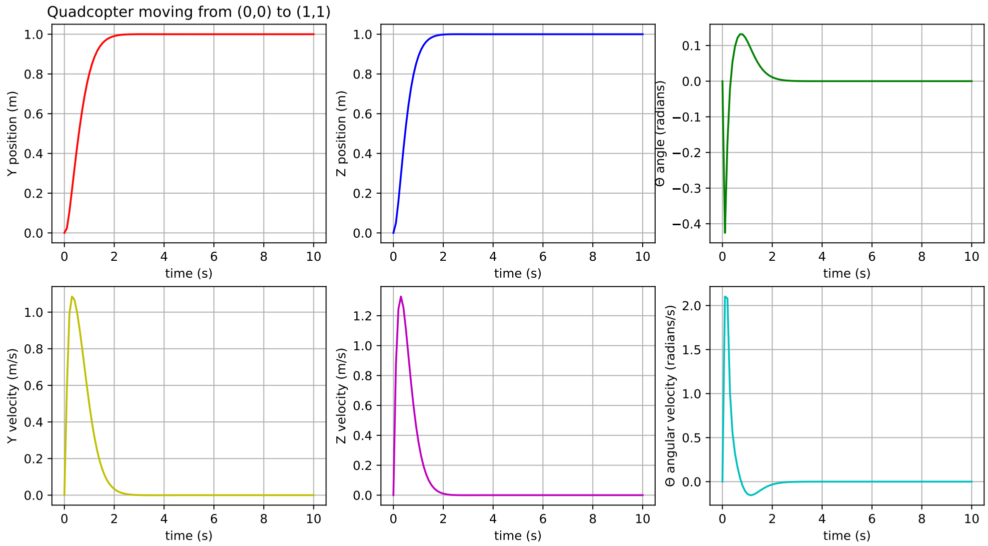

Figura 2.2: Resposta ao degrau unitário

2.5 Trajetória miníma

Supondo que o trajeto entre dois pontos p_1 e p_2 entre os intervalos t_1 e t_2 seja definido como

$$x(t) = f(t) \tag{2.21}$$

E sejam a velocidade, a aceleração, o jerk e o snap definidos, respectivamente, como

$$v(t) = \dot{x}(t) \tag{2.22}$$
$$a(t) = \ddot{x}(t) \tag{2.23}$$
$$j(t) = x^{(3)}(t) \tag{2.24}$$
$$s(t) = x^{(4)}(t) \tag{2.25}$$

A trajetória será miníma se

$$\min x(t) = \int_{t_1}^{t_2} (x^{(n)})^{(2)} dt = 0 \tag{2.26}$$

em que n indica o tipo de trajetória miníma, conforme Tabela 2.2.

Tabela 2.2: Tipos de minimização de trajetória.

n	Tipo de minimização
1	Velocidade
2	Aceleração
3	Jerk
4	Snap

Agora supondo uma trajetória com snap mínimo, segue que

$$\min x(t) = \int_{t_1}^{t_2} (x^{(4)})^{(2)} dt = 0 \tag{2.27}$$

$$\min x(t) = \int_{t_1}^{t_2} (x^{(8)}) dt = 0 \tag{2.28}$$

isto é verdade se a trajetória for definida como um polinômio de sétima como

$$x(t) = c_7 t^7 + c_6 t^6 + c_5 t^5 + c_4 t^4 + c_3 t^3 + c_2 t^2 + c_1 t + c_0 \tag{2.29}$$

uma vez que $x^{(8)}(t) = 0$, então (2.27) é mínimo e igual a zero.

A tabela 2.3 mostra a relação do tipo de trajetória miníma com a ordem do polinômio que define a trajetória.

Tabela 2.3: Ordem do polinômio que define a trajetória de acordo com o tipos de minimização de trajetória.

Tipo de minimização	Ordem do polinômio	Numero de coeficientes
Velocidade	1	2
Aceleração	3	4
Jerk	5	6
Snap	7	8

2.5.1 Gerando a trajetória

Para um conjunto de 2 pontos a e b onde o tempo na posição a é t_1, e o tempo na posição b é t_2, assumimos que a velocidade e todas as derivadas de tempo adicionais serão zero (ou seja, o objeto após o trajetória começa e termina em repouso):

Tabela 2.4: Condições de contorno.

	Posição	Velocidade	Aceleração	Jerk
$t = t_1$	a	0	0	0
$t = t_2$	b	0	0	0

Seja a trajetória definida como em (2.30), isso significa que teremos 8 incógnitas, da álgebra linear é conhecido que serão necessárias 8 equações linearmente independentes para determinar os coeficientes do polinômio que resultara como a trajetória de snap mínima no período de tempo determinado. Sabendo-se as condições de contorno é possível avaliar as equações de posição, velocidade, aceleração e jerk nos tempos inicial e final para gerar as equações necessárias. Observe que isso será em 1 dimensão, mas podemos repetir o processo para cada dimensão - por exemplo, se quisermos gerar uma trajetória no espaço 3D, o processo será o mesmo para x, y e z e podemos reutilizar a matriz A para cada dimensão.

As equações são as seguintes:

$$
\begin{aligned}
a &= c_7t_1^7 + c_6t_1^6 + c_5t_1^5 + c_4t_1^4 + c_3t_1^3 + c_2t_1^2 + c_1t_1 + c_0 \\
0 &= 7c_7t_1^6 + 6c_6t_1^5 + 5c_5t_1^4 + 4c_4t_1^3 + 3c_3t_1^2 + 2c_2t_1 + c_1 \\
0 &= 42c_7t_1^5 + 30c_6t_1^4 + 20c_5t_1^3 + 12c_4t_1^2 + 6c_3t_1 + 2c_2 \\
0 &= 210c_7t_1^4 + 120c_6t_1^3 + 60c_5t_1^2 + 24c_4t_1 + 6c_3 \\
b &= c_7t_2^7 + c_6t_2^6 + c_5t_2^5 + c_4t_2^4 + c_3t_2^3 + c_2t_2^2 + c_1t_2 + c_0 \\
0 &= 7c_7t_2^6 + 6c_6t_2^5 + 5c_5t_2^4 + 4c_4t_2^3 + 3c_3t_2^2 + 2c_2t_2 + c_1 \\
0 &= 42c_7t_2^5 + 30c_6t_2^4 + 20c_5t_2^3 + 12c_4t_2^2 + 6c_3t_2 + 2c_2 \\
0 &= 210c_7t_2^4 + 120c_6t_2^3 + 60c_5t_2^2 + 24c_4t_2 + 6c_3
\end{aligned}
$$

No formato matricial temos

$$
\underbrace{\begin{bmatrix}
1 & t_1 & t_1^2 & t_1^3 & t_1^4 & t_1^5 & t_1^6 & t_1^7 \\
0 & 1 & 2t_1^1 & 3t_1^2 & 4t_1^3 & 5t_1^4 & 6t_1^5 & 7t_1^6 \\
0 & 0 & 2 & 6t_1^1 & 12t_1^2 & 20t_1^3 & 30t_1^4 & 42t_1^5 \\
0 & 0 & 0 & 6 & 24t_1^1 & 60t_1^2 & 120t_1^3 & 210t_1^4 \\
1 & t_2 & t_2^2 & t_2^3 & t_2^4 & t_2^5 & t_2^6 & t_2^7 \\
0 & 1 & 2t_2^1 & 3t_2^2 & 4t_2^3 & 5t_2^4 & 6t_2^5 & 7t_2^6 \\
0 & 0 & 2 & 6t_2^1 & 12t_2^2 & 20t_2^3 & 30t_2^4 & 42t_2^5 \\
0 & 0 & 0 & 6 & 24t_2^1 & 60t_2^2 & 120t_2^3 & 210t_2^4
\end{bmatrix}}_{A}
\underbrace{\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix}}_{c}
=
\underbrace{\begin{bmatrix} a \\ 0 \\ 0 \\ 0 \\ b \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{p}
\tag{2.30}
$$

Ou seja o sistema é simplificado para

$$Ac = p \tag{2.31}$$

Os coeficientes são determinados fazendo

$$c = A^{-1}p \tag{2.32}$$

Desta forma sabendo-se as posições e os tempo para atingir cada posição é possível determinar o polinômio que minimiza uma trajetória.

Com os coeficientes determinados a cada instante de amostragem sera passado para ao controlador as posições velocidades e acelerações de referencia, determinadas no um instante de amostragem a frente ao instante atual.

Referências Bibliográficas

BLOMQVIST, V. *Pymunk: A easy-to-use pythonic rigid body 2d physics library (version 6.2.0)*. 2007. Disponível em: <https://www.pymunk.org>.

HARRIS, C. R. et al. Array programming with NumPy. *Nature*, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>.

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

MEURER, A. et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, v. 3, p. e103, jan. 2017. ISSN 2376-5992. Disponível em: <https://doi.org/10.7717/peerj-cs.103>.

PYGLET: a cross-platform windowing and multimedia library for Python (version 1.5.21). 2016. Disponível em: <https://github.com/pyglet/pyglet>.