Insper

Curso: Engenharia Mecatrônica

Disciplina: DRONES

Prof°: Fábio Bobrow

RELATÓRIO FINAL

Modelagem e controle de um quadricoptero

Gabriel Pinto Patrocinio – 15610367 – gabrielpp@al.insper.edu.br

Pedro Casella – 15610326 – pedroctr@al.insper.edu.br

Rodrigo Ronchail Gikas – 15610385 – rodrigo.rgk@hotmail.com

Sumário

1. Introdução

Este relatório tem como objetivo registrar todo o processo de modelagem e controle de um drone de pequeno porte, mais especificamente um quadricóptero do modelo Crazyflie 2.0, abordando desde o embasamento teórico utilizado e os experimentos em laboratório até a implementação efetiva do código de controle.

A estrutura de controle de um drone voador, desde a aquisição de dados dos sensores até o acionamento dos motores pode ser representado pelo diagrama abaixo:

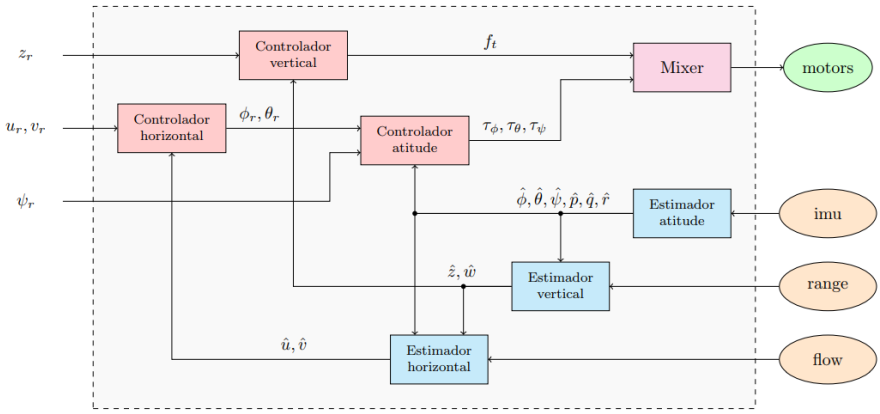


Figura 1 – Diagrama de controle do drone (imagem retirada de slide da aula 1)

2. Base teórica

Essa seção será dividida em alguns tópicos, o primeiro sendo uma breve descrição do Crazyflie 2.0 e de suas propriedades físicas e os demais a análise de cada bloco do diagrama e dos conceitos necessários para os laboratórios.

2.1. Crazyflie 2.0

O Crazyflie 2.0 é o drone quadricóptero que será utilizado ao longo deste projeto e conta com:

- Quatro motores de corrente contínua de voltagem e corrente máximas de 4,2 V e 1000 mA, respectivamente, e uma relação de 14000 rpm/V;
- Um processador ARM Cortex-M4;
- Uma unidade de medição inercial (IMU) de 10 eixos, incluindo acelerômetro, giroscópio, magnetômetro e barômetro;

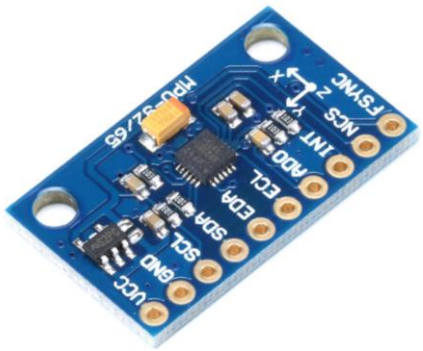


Figura 2 – MPU9250 – IMU de 9 eixos

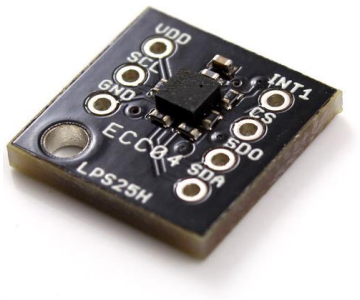


Figura 3 – LPS25H – Barômetro de alta precisão

- Uma bateria de íons de lítio de 3,7 V e 240 mAh.

Figura 4 - Crazyflie 2.0

Seu tempo de voo com a bateria cheia é de 7 minutos e sua capacidade máxima de carga é de 15 gramas.

Suas propriedades físicas estão listadas abaixo:

Tabela 1 – Dados físicos do Crazyflie 2.0

Massa (m)	$27\ g$
Momento de inércia (eixo x) (I_{xx})	$1{,}6 \cdot 10^{-5}\ kg \cdot m^2$
Momento de inércia (eixo x) (I_{yy})	$1{,}6 \cdot 10^{-5}\ kg \cdot m^2$
Momento de inércia (eixo x) (I_{zz})	$2{,}9 \cdot 10^{-5}\ kg \cdot m^2$
Distância entre os motores e os eixos (l)	$3{,}3 \cdot 10^{-2}\ m$

2.2. Mixer

O mixer é um software responsável pelo acionamento controlado do drone, recebendo como entrada uma força total e três torques, sendo estes cada um referente a um eixo de rotação (roll, pitch ou yaw), e devolvendo como saída sinais de acionamento para cada um dos motores do drone, de modo a garantir que esses parâmetros de entrada sejam alcançados.

Para o seu projeto, é necessário compreender três relações principais: como as forças de arrasto e de sustentação em cada hélice influenciam a

força e os torque totais no drone, como a rotação do eixo do motor afeta as forças sobre as hélices e qual a dinâmica entre o sinal de acionamento do motor e a rotação de seu eixo.

2.2.1. Decomposição de forças

Na imagem abaixo estão representadas todas as forças e torques atuantes sobre o drone devido à rotação dos 4 motores DC presentes em suas extremidades:

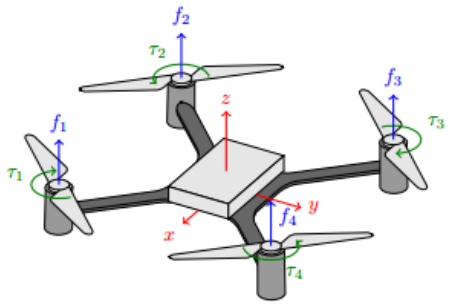


Figura 5 - Forças e torques envolvidos no quadricóptero (imagem retirada de slide da aula 2)

Entretanto, estas forças e torques podem facilmente serem transformados em uma força e três torques em relação ao sistema de coordenadas do drone, simplificando o controle e análises futuras, como pode ser visto abaixo:

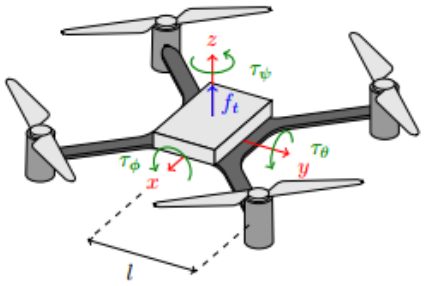


Figura 6 - Composição das forças e torques (imagem retirada de slide da aula 2)

$$\begin{cases} f_t = f_1 + f_2 + f_3 + f_4 \\ \tau_\phi = (-f_1 - f_2 + f_3 + f_4)l \\ \tau_\theta = (-f_1 + f_2 + f_3 - f_4)l \\ \tau_\psi = -\tau_1 + \tau_2 - \tau_3 + \tau_4 \end{cases} \quad \text{(Eq.1)}$$

Dito isso, é possível controlar todos os movimentos do drone escolhendo adequadamente os valores de forças e torques demonstrados acima. Por exemplo, para levantar o drone na vertical (no caso, no eixo

z), entra-se como parâmetro uma força f_t maior que a força peso, mantendo todas as entradas de torque nulas. Para ir para frente ou para trás (eixo x), entra-se com um valor de torque no eixo y e 0 nos outros parâmetros, entre outros movimentos possíveis.

2.2.2. Coeficientes de arrasto e sustentação

A força e o torque atribuídos para cada hélice na descrição acima, por sua vez, são uma representação simplificada das forças aerodinâmicas envolvidas no sistema. O deslocamento de cada pá em torno do eixo de rotação da hélice resulta em duas forças principais: uma de arrasto, resistente ao movimento pelo atrito com o ar deslocado, e uma de sustentação, provocada por uma diferença de pressão entre as partes superior e inferior da hélice. A figura a seguir mostra um esquema dessas forças:

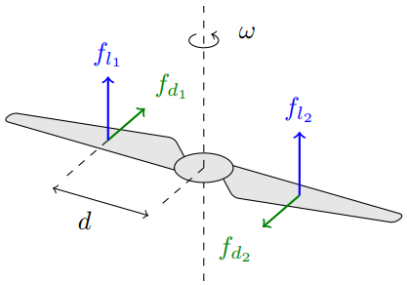


Figura 7 – Esquema de forças em uma das hélices do drone (imagem retirada de slide da aula 2)

Essas forças dependem de uma série de propriedades tanto da hélice quanto do ar em volta. São elas densidade do ar (ρ), velocidade de deslocamento das pás da hélice (v), que no caso será expressa em função da velocidade angular da hélice, logo, $v = \omega d$; área da pá perpendicular ao movimento e coeficientes de arrasto e de sustentação, respectivos às suas forças.

$$\begin{cases} f_{d_{1,2}} = \frac{1}{2}\rho(\omega d)^2 A C_d \\ f_{l_{1,2}} = \frac{1}{2}\rho(\omega d)^2 A C_l \end{cases} \quad \text{(Eq.2)}$$

Como representado na seção 2.2.1, as forças de sustentação podem ser somadas em uma única força sob o eixo de rotação e as forças de

arrasto podem ser substituídas por um único torque, considerando a distância d do centro de rotação, produzindo assim o sistema a seguir:

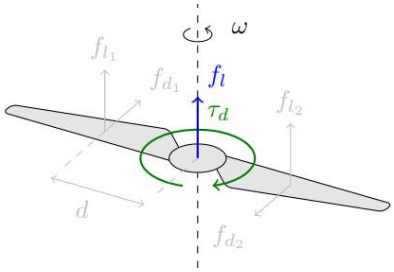


Figura 8 – Sistema de forças e torque da hélice simplificado (imagem retirada de slide da aula 2)

$$\begin{cases} f_l = 2\left(\frac{1}{2}\rho(\omega d)^2 A C_l\right) \\ \tau_d = 2\left(\frac{1}{2}\rho(\omega d)^2 A C_d\right) d \end{cases} \quad \text{(Eq.3)}$$

Sendo a velocidade angular ω o único parâmetro dessas equações que pode ser controlado e, considerando os demais termos constantes, essas expressões podem ser reescritas como

$$\begin{cases} f_l = 2\left(\frac{1}{2}\rho d^2 A C_l\right)\omega^2 \\ \tau_d = 2\left(\frac{1}{2}\rho d^2 A C_d\right) d\omega^2 \end{cases} \Rightarrow \begin{cases} f_l = k_l\omega^2 \\ \tau_d = k_d\omega^2 \end{cases} \quad \text{(Eq.4)}$$

As constantes de arrasto (k_d) e de sustentação (k_l) podem então ser obtidas experimentalmente e os processos para isso serão descritos nas seções 3.2 e 3.3 desse relatório.

2.2.3. Acionamento dos motores

Como foi mencionado anteriormente, os 4 motores do Crazyflie 2.0 são motores de corrente contínua, de modo que o controle da velocidade de rotação dos seus eixos pode ser realizado através da técnica de PWM (Pulse Width Modulation). Com esta técnica, uma série de pulsos de mesma tensão são enviados ao motor, mantendo um intervalo de tempo fixo entre cada pulso, porém regulando a duração do pulso com tensão alta. Dessa forma, a tensão efetivamente recebida pelo motor é um intermediário entre os níveis de alta e baixa tensão do pulso, dependente

da proporção entre os tempos de atuação de cada, como pode ser visto na imagem abaixo:

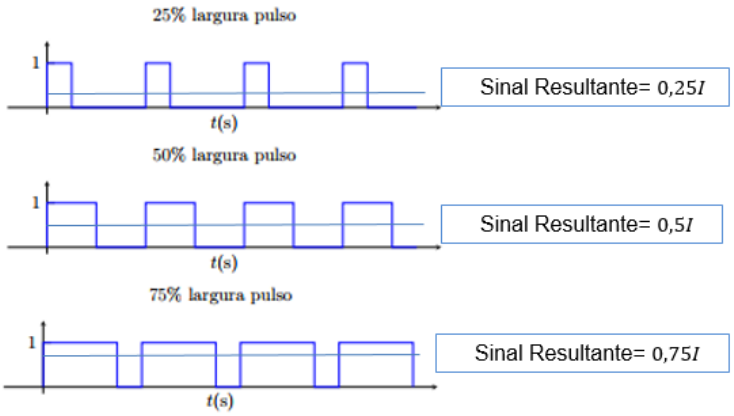
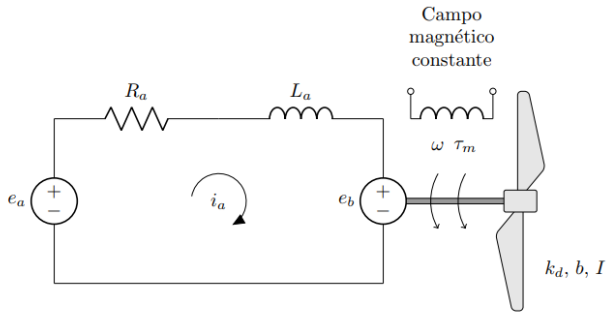


Figura 9 - Explicação PWM

Com isso, torna-se necessário avaliar a relação entre a velocidade angular do eixo do motor e o sinal PWM, o que pode ser obtida pela modelagem eletromecânica do motor:



Onde:

- e_a – Tensão de armadura (V)
- i_a – Corrente de armadura (A)
- R_a – Resistência de armadura (Ω)
- L_a – Indutância de armadura (H)
- e_b – Tensão contra-eletromotriz (V)
- ω – Velocidade angular do motor/helice (rad/s)
- τ_m – Torque do motor (N · m)
- k_d – Constante de arrasto da hélice (N · m · s^2/rad^2)
- b – Coeficiente de atrito viscoso do motor (N · m · s/rad)
- I – Momento de inércia do motor/hélice (kg · m^2)

Figura 10 – Diagrama do modelo eletromecânico de um motor do drone (imagem retirada de roteiro do laboratório 2)

$$\begin{cases} e_a - R_a i_a - L_a \dfrac{di_a}{dt} - e_b = 0 \\ \tau_m - k_d \omega^2 - b\omega = I \dfrac{d\omega}{dt} \\ \tau_m = K_m i_a \\ e_b = K_m \omega \end{cases} \qquad \text{(Eq.5)}$$

Sendo K_m a constante de torque do motor.

Após um pouco de abstração matemática, chega-se em

$$e_a = \frac{R_a k_d}{K_m}\omega^2 + \frac{R_a b + K_m^2}{K_m}\omega \qquad \text{(Eq.6)}$$

e considerando que o PWM representa uma proporção entre a tensão da armadura do motor (e_a) e a tensão da bateria (e_s)

$$PWM = \frac{e_a}{e_s} \qquad \text{(Eq.7)}$$

tem-se que

$$PWM = \frac{R_a k_d}{K_m e_s}\omega^2 + \frac{R_a b + K_m^2}{K_m e_s}\omega$$
$$PWM = \alpha\omega^2 + \beta\omega. \qquad \text{(Eq.8)}$$

Essa equação mostra não apenas que a relação entre PWM e velocidade angular em regime permanente corresponde a uma função quadrática de coeficiente independente nulo, mas também que os coeficientes restantes devem ser maiores ou iguais a zero, uma vez que todos os parâmetros que os constituem são grandezas positivas.

Em suma, realizando uma regressão a partir de dados de PWM por velocidade angular em regime permanente, pode-se estimar os valores das constantes α e β, tendo assim uma função de acionamento que pode ser aplicada na implementação do código de controle. Esse processo será descrito mais detalhadamente na seção 3.1.

2.3. Estimador de atitude

Antes de sequer poder controlar o drone, é necessário estar ciente a todo momento da sua orientação no espaço para que o seguimento de referência seja atendido de forma adequada. Essa orientação, normalmente denominada de atitude, pode ser obtida por meio de um estimador de atitude, que se utiliza das medições do acelerômetro e do giroscópio da IMU do drone para calcular os ângulos de Euler que descrevem a disposição do drone no espaço.

O acelerômetro e o giroscópio apresentam medidas imprecisas devido ao ruído dos sensores, além de cada um possuir certas limitações quanto à estimação da atitude. Desta forma, combina-se as leituras de ambos de maneira que se obtenha os dados mais corretos possíveis.

Nos subtópicos abaixo estão descritos os processos de estimação específicos para cada sensor e, posteriormente na seção 4.2.4, será mostrado como unir o resultado de ambos os sensores em uma única estimação mais acurada.

2.3.1. Estimador do acelerômetro

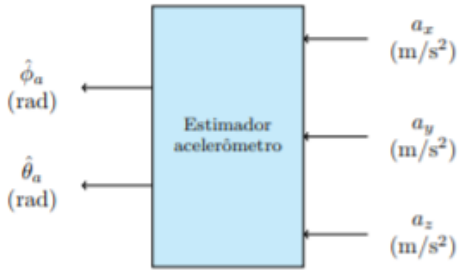


Figura 11 – Diagrama estimador do acelerômetro (imagem retirada de roteiro do laboratório 6)

Diante do fato que a aceleração da gravidade sempre está orientada para baixo, ou seja, no sentido negativo do eixo z do sistema de coordenadas fixo, e o acelerômetro mede aceleração linear em relação ao sistema de coordenadas móvel, apenas os ângulos de Euler de rolagem (*roll*) e de arremesso (*pitch*) são passiveis de serem calculados indiretamente projetando as componentes da aceleração medida no sistema de coordenadas fixo. O ângulo de guinada (*yaw*), correspondente à primeira rotação na transformação de sistemas de coordenadas em torno do eixo z, não altera as componentes da aceleração da gravidade, de modo que é impossível ser estimado deste jeito.

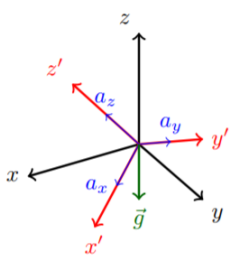


Figura 12 - Sistema de coordenadas acelerômetro (imagem retirada de slide da aula 6)

A medição da gravidade feita pelo acelerômetro pode ser representada da seguinte forma:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = R\vec{g} \qquad \text{(Eq.9)}$$

Utilizando o sistema de rotação zyx e sabendo que a aceleração da gravidade vale $[0; 0; -g]$ no sistema de coordenadas fixo, chega-se em:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} c\theta c\psi & c\theta s\psi & -s\theta \\ -c\phi s\psi + s\phi s\theta c\psi & c\phi c\psi + s\phi s\theta s\psi & s\phi c\theta \\ s\phi s\psi + c\phi s\theta c\psi & -s\phi c\psi + c\phi s\theta s\psi & c\phi c\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = g \begin{bmatrix} sin\theta \\ -\sin\phi \cdot \cos\theta \\ -\cos\phi \cdot \cos\theta \end{bmatrix} \qquad \text{(Eq.10)}$$

A partir da equação 10, pode-se então derivar os ângulos de *roll* e *pitch*:

$$\phi_a = \tan^{-1}\left(\frac{-a_y}{-a_z}\right) \qquad \text{(Eq.11)}$$

$$\theta_a = \tan^{-1}\left(\frac{a_x}{-sign(a_z) \cdot \sqrt{a_y^2 + a_z^2}}\right) \qquad \text{(Eq.12)}$$

2.3.2. Estimador do giroscópio

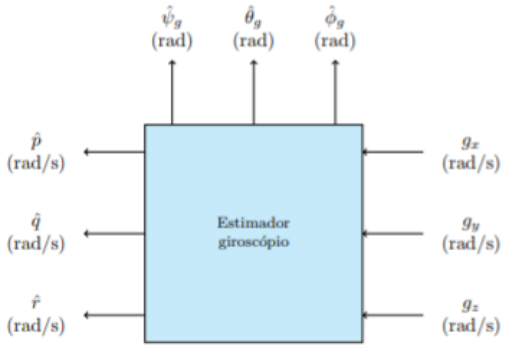


Figura 13 - Diagrama estimador do giroscópio (imagem retirada de roteiro do laboratório 6)

O estimador de atitude baseado no giroscópio, por sua vez, é capaz de estimar não apenas os ângulos de Euler, mas também as suas derivadas, empregando uma transformação de sistemas de coordenadas, para converter as medições do sensor em relação ao sistema de coordenadas móvel (do drone) para o sistema fixo (global), e um processo iterativo para obter os ângulos integrando as velocidades obtidas.

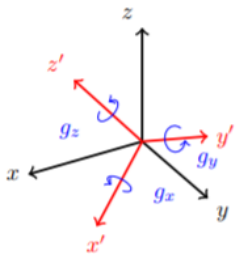


Figura 14 - sistema de coordenadas do giroscópio (imagem retirada de slide de aula 6)

Utilizando como base uma rotação zyx e uma análise da dinâmica de corpo rígido do drone, é possível calcular as velocidades angulares em termos de ângulos de Euler da seguinte forma:

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta\end{bmatrix} \cdot \begin{bmatrix}p\\ q\\ r\end{bmatrix} \qquad \text{(Eq.13)}$$

A equação acima pode então ser integrada, visando assim obter os ângulos de Euler:

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix} = \begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix}_0 + \int \begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} dt \qquad \text{(Eq.14)}$$

Ao discretizar a equação 14 para tornar possível a implementação no microcontrolador chega-se em:

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix}_k = \begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix}_{k-1} + \begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}_k \Delta t \qquad \text{(Eq.15)}$$

Substituindo a equação 13 na equação 15 e sabendo que $\begin{bmatrix}p\\ q\\ r\end{bmatrix} = \begin{bmatrix}g_x\\ g_y\\ g_z\end{bmatrix}$, chega-se por fim em:

$$\begin{bmatrix}\phi_g\\ \theta_g\\ \psi_g\end{bmatrix}_k = \begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix}_{k-1} + \begin{bmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta\end{bmatrix}_{k-1} \cdot \begin{bmatrix}g_x\\ g_y\\ g_z\end{bmatrix}_k \Delta t \qquad \text{(Eq.16)}$$

Porém, assim como o estimador do acelerômetro possui suas limitações, não sendo capaz de estimar o ângulo de guinada, o estimador do giroscópio também mostra certos problemas. Embora consiga estimar

bem as velocidades angulares, o fato de precisar integrá-las para obter os ângulos de Euler permite um acumulo progressivo de erro na estimação com o passar do tempo.

2.4. Estimador vertical

Como o próprio nome diz, e estimador vertical estimará, através de um sensor LIDAR VL53L0X (como o da Figura 15) a posição e a velocidade verticais do drone. Para isso, será utilizada uma técnica de duas etapas, sendo elas uma de predição, que utiliza o modelo do sistema para estimar os valores de posição e velocidade vertical, e uma etapa de correção, que utiliza efetivamente os valores das leituras do sensor LIDAR para estimar os valores desejados. Como os conceitos necessários para a etapa de predição serão comentados na seção 2.6.1, ela será brevemente descrita durante a implementação dos códigos, enquanto que a etapa de correção é desenvolvida mais a fundo a seguir.

Figura 15 – Sensor LIDAR VL53L0X

A posição vertical medida pelo sensor varia de acordo com a angulação do drone. Em duas dimensões este fenômeno pode ser entendido de acordo com a figura abaixo:

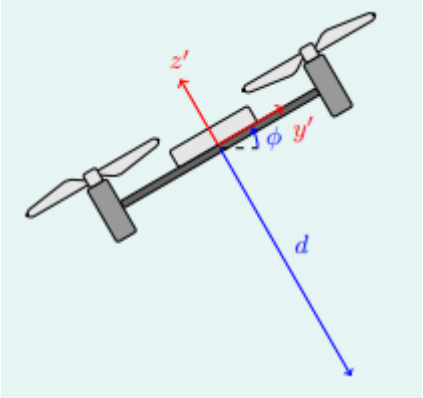


Figura 16 – Leitura do LIDAR em 2D (imagem retirada de slide da aula 8)

Percebe-se que a leitura do sensor indicará a medida *d* demonstrada na figura acima e não a altura vertical desejada. Para isso, pode-se utilizar as equações de triângulo retângulo para, com base no ângulo de rolagem estimado anteriormente na seção 2.3.1, calcular a altura *z*.

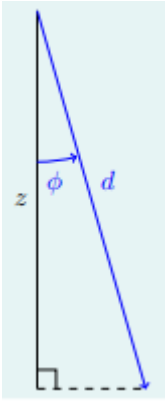


Figura 17 – Relação entre a leitura do sensor e a altura do drone em 2D (imagem retirada de slide da aula 8)

Tem-se que:

$$\cos\phi = \frac{z}{d} \quad \therefore \quad z = d \cdot \cos\phi \qquad \text{(Eq.17)}$$

Entretanto, para um caso 3D esta análise fica um pouco mais complexa. O modo que os ângulos de *roll* e *pitch* afetam a medida da medida *d* podem ser vistas na imagem abaixo:

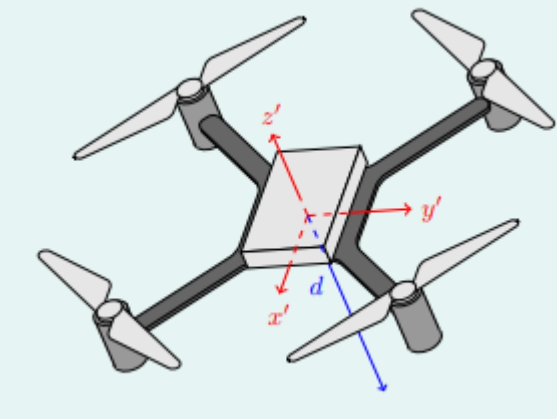


Figura 18 – Leitura do LIDAR em 3D (imagem retirada de slide da aula 8)

Para isso, pode-se de maneira similar ao caso 2D utilizar as equações do triângulo retângulo para encontrar o valor real da altura *z*.

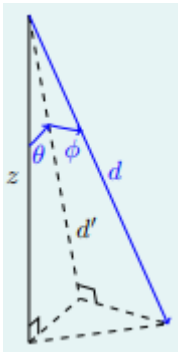


Figura 19 – Relação entre a leitura do sensor e a altura do drone em 3D (imagem retirada de slide da aula 8)

Da figura acima, têm-se a partir do estimador de atitude os ângulos θ *e* ϕ, além da medida *d* obtida pelo sensor LIDAR. Com isso, pode-se estimar o valor de *z* da seguinte forma:

$$\cos\theta = \frac{z}{d'} \Rightarrow z = d' \cdot \cos\theta \qquad \text{(Eq.18)}$$

Além disso,

$$\cos\phi = \frac{d'}{d} \Rightarrow d' = d \cdot \cos\phi \qquad \text{(Eq.19)}$$

Substituindo a equação 18 em 19, obtém-se:

$$z = d \cdot \cos\phi \cdot \cos\theta \qquad \text{(Eq.20)}$$

Com essa equação obtida, pode-se implementar a etapa de correção no estimador vertical através das medidas obtidas pelo sensor.

É importante ter em mente que a etapa de predição acontece numa frequência muito maior que a etapa de correção. Nesse caso em questão, a primeira terá uma frequência de 500Hz, enquanto a outra uma frequência de 20Hz.

2.5. Estimador Horizontal

O estimador horizontal possui uma função similar ao vertical, de estimar posição e velocidade do drone, entretanto, como o próprio nome já diz, nesse caso ele estima as posições (x, y) e velocidades horizontais (u, v).

Para conseguir obter os valores desejados, utiliza-se um sensor de fluxo óptico PMW3901 (Figura 20), que calcula a distância percorrida por um pixel de uma imagem dentro de seu campo de visão em dois instantes diferentes. Ao considerarmos que o drone está se movendo sobre uma superfície na qual os objetos estão parados, pode-se afirmar que o valor retornado pelo sensor será o deslocamento do próprio drone.

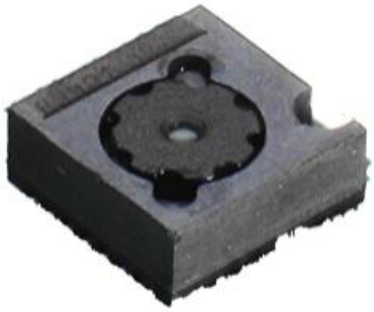


Figura 20 - Sensor de Fluxo Óptico PMW3901

A imagem abaixo apresenta o princípio de funcionamento do sensor de fluxo óptico:

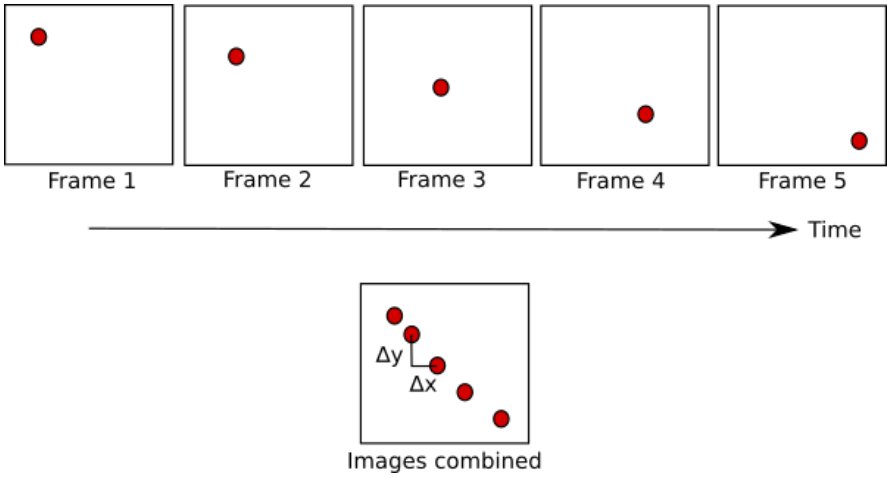


Figura 21 - Funcionamento Sensor de Fluxo Óptico - Imagem retirada de: <https://www.bitcraze.io/2017/11/optical-flow/>

Assim como no estimador vertical, nesse caso também é utilizada duas etapas para estimar a posição horizontal, uma de predição e outra de correção, com os princípios de funcionamento iguais aos já relatado anteriormente.

A etapa de predição utiliza o modelo do sistema para prever os valores de posições e velocidades desejados. O modelo utilizado está representado abaixo:

$$\begin{bmatrix} \dot{x} \\ \dot{u} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ u \end{bmatrix}$$

$$\begin{bmatrix} \dot{y} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} y \\ v \end{bmatrix}$$

Já a etapa de correção exige um pouco mais de cuidado, visto que, dependendo do nível de exatidão que se deseja ter na estimação, as equações para correção vão ficando mais complicadas.

Como já foi mencionado anteriormente, a correção utiliza os valores captados pelo sensor para estimar as posições e velocidades horizontais, entretanto, os ângulos de Roll e Pitch podem influênciar no campo de visão do sensor, alterando assim as proporções entre as distâncias medidas em pixels pelo sensor e as distâncias reais observadas, alterando assim as equações de correção, como pode ser visto nas imagens abaixo.

- Caso 1 – Ângulos de Roll e Pitch $\phi, \theta = 0$.

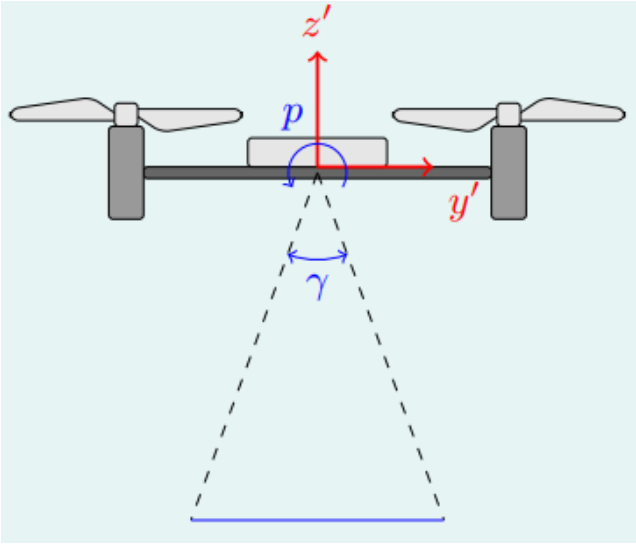


Figura 22 - Caso 1 - Pitch e Roll = 0 (imagem retirada de slide da aula 10)

- Caso 2 – Ângulos de Pitch $\theta = 0$ e Roll $\phi \neq 0$.

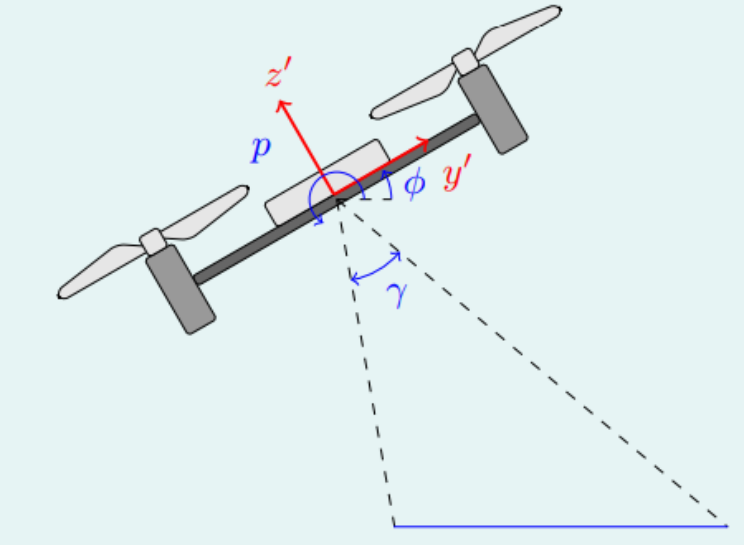


Figura 23 - Caso 2 - Pitch = 0 & Roll ≠ 0 (imagem retirada de slide da aula 10)

- Caso 3 – Ângulos de Roll e Pitch $\phi, \theta \neq 0$.

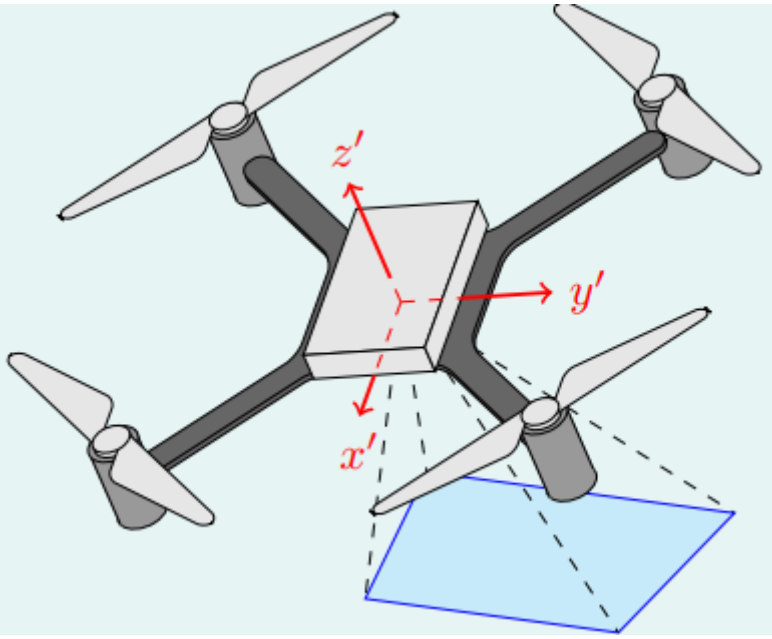


Figura 24 - Caso 3 - Roll e Pitch ≠ 0 (imagem retirada de slide da aula 10)

Devido à complexidade inserida pelos casos 2 e principalmente o 3, optou-se por simplificar o sistema e, visto que todas as análises até o momento foram realizadas em torno do ponto de equilíbrio, que levam em consideração os ângulos de Roll e Pitch valendo 0, escolheu-se implementar no estimador as equações de correção referentes ao caso 1, deduzidas a seguir.

Sabendo inclusive que não apenas os ângulos de Roll e Pitch, mas também as suas velocidades influenciam a estimação horizontal, já que que, mesmo sem deslocamento horizontal, uma rotação em torno do eixo x provoca um aparente deslocamento em y, convém calcular primeiramente as velocidades $u\,e\,v$ a partir das medições e dos parâmetros do sensor.

Considerando um ângulo de abertura do sensor γ, uma altura d e um campo de visão de lado a, desenha-se a partir da Figura 22 o seguinte triângulo:

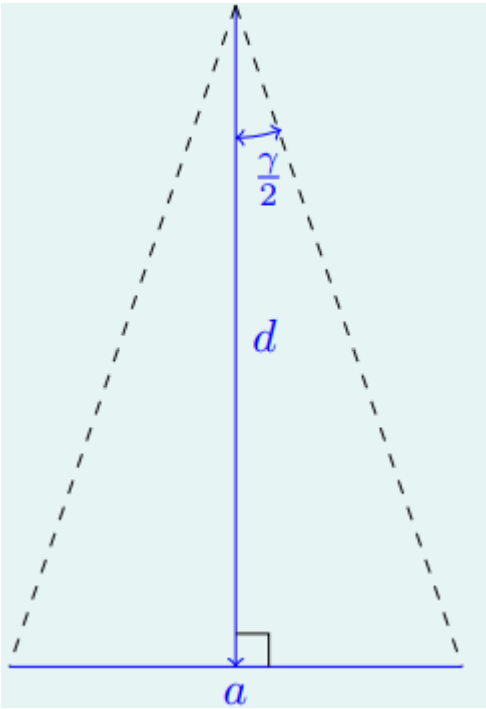


Figura 25 - Triângulo Caso 1 (imagem retirada de slide da aula 10)

A partir disso, sabe-se que:

$$\tan\left(\frac{\gamma}{2}\right) = \frac{\frac{a}{2}}{d} \rightarrow a = 2d \cdot \tan\left(\frac{\gamma}{2}\right) \qquad \text{(Eq.21)}$$

Considerando um intervalo de tempo Δt entre os dois sinais captados pelo sensor e W com sendo a resolução do sensor, obtém-se:

$$v = \frac{a}{W} \cdot \frac{p_y}{\Delta t} - p \cdot d \qquad \text{(Eq.22)}$$

Desenvolvendo a equação acima tem-se:

$$v = \frac{2d \cdot \tan\left(\frac{\gamma}{2}\right)}{W \cdot \Delta t} \cdot p_y \cdot d - p \cdot d \qquad \text{(Eq.23)}$$

Sabe-se que W e Δt são constantes, assim como $2d \cdot \tan\left(\frac{\gamma}{2}\right)$, logo pode-se compacta-las em uma única constante σ, obtendo, por fim:

$$v = \left(\sigma \cdot p_y - p\right) \cdot d \qquad \text{(Eq.24)}$$

A mesma análise pode ser feita para se obter o valor de u, chegando-se em:

$$u = (\sigma \cdot p_x + q) \cdot d \qquad \text{(Eq.25)}$$

Com essas equações em mente, pode-se por fim implementar o estimador, considerando que ambas as etapas, de predição e correção, ocorrem à uma frequência de 500Hz.

2.6. Controladores

Idealizados os estimadores, o foco do projeto passa a ser conceber e implementar os sistemas de controle que irão delimitar a dinâmica de voo do quadricoptero. Dito isso, a seguir serão apresentadas duas seções, uma dedicada a analisar brevemente a dinâmica de corpo rígido do drone, assim como a sua manipulação visando obter um sistema mais simples de se controlar, e a outra focada na descrição do método de controle selecionado para esse projeto específico.

2.6.1. Dinâmica do quadricoptero

Com o objetivo de descrever o comportamento de um drone do tipo voador no espaço, tornam-se necessárias 12 variáveis de estado: suas coordenadas *x, y* e *z* e sua orientação em ângulos de Euler (ϕ, θ e ψ) em relação a uma sistema de coordenadas fixo, dado que sempre se espera saber onde e com que orientação o drone se encontra segundo uma referência externa, e suas velocidades lineares e angulares em relação ao próprio referencial, já que isso torna a representação de forças e torques muito mais simples.

Após muita derivação matemática e análises de dinâmica e cinemática, são definidas as equações de corpo rígido do drone:

$$\begin{cases} \dot{x} = v_x' \cos\theta \cos\psi + v_y' \cos\theta \sin\psi - v_z' \sin\theta \\ \dot{y} = v_x'(-\cos\phi \sin\psi + \sin\phi \sin\theta \cos\psi) + v_y'(\cos\phi \cos\psi + \sin\phi \sin\theta \sin\psi) + v_z' \sin\phi \cos\theta \\ \dot{z} = v_x'(\sin\phi \sin\psi + \cos\phi \sin\theta \cos\psi) + v_y'(-\sin\phi \cos\psi + \cos\phi \sin\theta \sin\psi) + v_z' \cos\phi \cos\theta \\ \dot{\phi} = \omega_x' + \omega_y' \sin\phi \tan\theta + \omega_z' \cos\phi \tan\theta \\ \dot{\theta} = \omega_y' \cos\phi - \omega_z' \sin\phi q \\ \dot{\psi} = \omega_y' \sin\phi \sec\theta + \omega_z' \cos\phi \sec\theta \\ \dot{v_x}' = -\omega_y' v_z' + \omega_z' v_y' + \frac{1}{m} f_x' \\ \dot{v_y}' = -\omega_z' v_x' + \omega_x' v_z' + \frac{1}{m} f_y' \\ \dot{v_z}' = -\omega_x' v_y' + \omega_y' v_x' + \frac{1}{m} f_z' \\ \dot{\omega_x}' = -\frac{I_{zz}' - I_{yy}'}{I_{xx}'} \omega_y' \omega_z' + \frac{1}{I_{xx}'} \tau_x' \\ \dot{\omega_y}' = -\frac{I_{xx}' - I_{zz}'}{I_{yy}'} \omega_x' \omega_z' + \frac{1}{I_{yy}'} \tau_y' \\ \dot{\omega_z}' = -\frac{I_{yy}' - I_{xx}'}{I_{zz}'} \omega_x' \omega_y' + \frac{1}{I_{zz}'} \tau_z' \end{cases} \quad \text{(Eq.26)}$$

Sendo

$$\begin{cases} f_x' = mg \sin\theta \\ f_y' = -mg \sin\phi \cos\theta \\ f_z' = -mg \cos\phi \cos\theta \\ \tau_x' = \tau_\phi \\ \tau_y' = \tau_\theta \\ \tau_z' = \tau_\psi \end{cases} \quad \text{(Eq.27)}$$

Ao observar o sistema de equações acima, é possível perceber claramente sua natureza não linear, com a presença de funções trigonométricas e equações acopladas entre si, o que por sua vez dificulta consideravelmente o projeto de um controlador, já que a maior parte da literatura de controle é dedicada exclusivamente a sistemas lineares.

Uma solução para esse problema envolve linearizar a planta em torno de um ponto de equilíbrio, no caso um cenário no qual o drone se encontre pairando no ar, perfeitamente parado e alinhado com a horizontal enquanto compensa seu próprio peso:

$$x_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ \phi_0 \\ \theta_0 \\ \psi_0 \\ u_0 \\ v_0 \\ \omega_0 \\ p_0 \\ q_0 \\ r_0 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 0 \\ 0 \\ \psi_0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad u_0 = \begin{bmatrix} f_{t_0} \\ \tau_{\phi_0} \\ \tau_{\theta_0} \\ \tau_{\psi_0} \end{bmatrix} = \begin{bmatrix} mg \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Resultando assim em um sistema de equações extremamente simples e desacopladas:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \\ u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 \\ 0 & 1/I_{xx} & 0 & 0 \\ 0 & 0 & 1/I_{yy} & 0 \\ 0 & 0 & 0 & 1/I_{zz} \end{bmatrix} \begin{bmatrix} f_t \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \qquad \text{(Eq.28)}$$

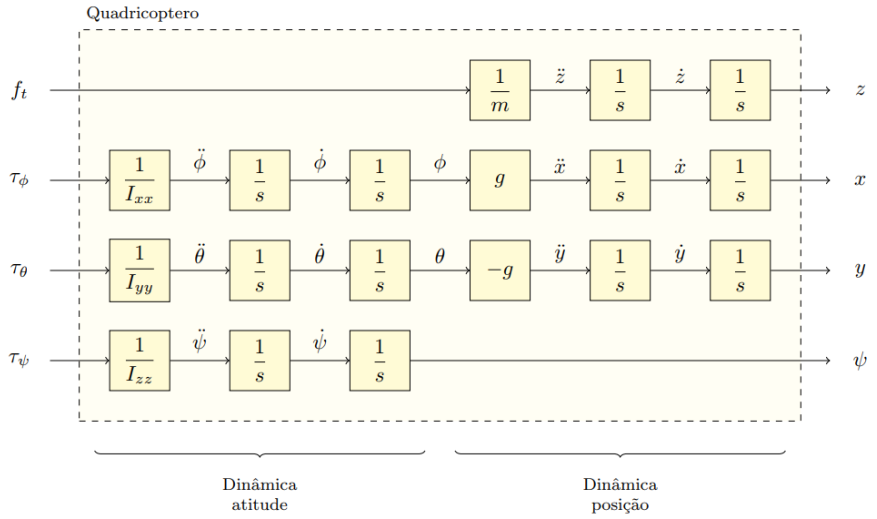


Figura 26 – Esquema da planta linearizada (imagem retirada de slide da aula 7)

Além de esse novo sistema facilitar o projeto dos controladores, já que cada posição e cada ângulo de Euler possui sua própria dinâmica com uma entrada e uma saída, pode-se notar que todas essas dinâmicas se mostram exatamente iguais, contendo sempre um duplo integrador e uma constante multiplicando a entrada. Em outras palavras, todos os

controladores podem ser projetados de forma similar, apenas alterando os ganhos para compensar essas constantes.

$$\begin{cases} x = g\dfrac{1}{s^2}\phi \\ y = -g\dfrac{1}{s^2}\theta \\ z = \dfrac{1}{m}\dfrac{1}{s^2}f_t \\ \phi = \dfrac{1}{I_{xx}}\dfrac{1}{s^2}\tau_\phi \\ \theta = \dfrac{1}{I_{yy}}\dfrac{1}{s^2}\tau_\theta \\ \psi = \dfrac{1}{I_{zz}}\dfrac{1}{s^2}\tau_\psi \end{cases} \qquad \text{(Eq.29)}$$

2.6.2. Controlador regulador linear quadrático

O método de controle idealizado foi o controlador LQR da teoria de controle moderno.

Um sistema linear pode sempre ser representado em uma estrutura matricial de **n** equações diferenciais de primeira ordem, denominada de representação em espaço de estados:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \qquad \text{(Eq.30)}$$

Como exemplo desse tipo de estrutura, a dinâmica de atitude de um eixo do quadricoptero pode ser representada pelo seguinte sistema de equações diferenciais em espaço de estados desta maneira:

$$\phi = \frac{1}{I_{xx}}\frac{1}{s^2}\tau_\phi \Rightarrow \ddot{\phi} = \frac{1}{I_{xx}}\tau_\phi$$

$$\begin{cases} \begin{bmatrix} \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \dfrac{1}{I_{xx}} \end{bmatrix} \cdot \tau_\phi \\ \phi = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \phi \\ \dot{\phi} \end{bmatrix} \end{cases} \qquad \text{(Eq.31)}$$

Assim, as matrizes A, B, C e D são:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ \frac{1}{I_{xx}} \end{bmatrix} \quad C = [1 \quad 0] \quad D = 0$$

Com a teoria de controle moderno, sabe-se que, desde que o sistema seja controlável, é possível alocar os polos da planta em malha fechada em qualquer posição do plano complexo apenas aplicando uma realimentação dos estados da planta:

$$u = K(x_r - x) \qquad \text{(Eq.32)}$$

De modo que a dinâmica do sistema passe a ser:

$$\begin{cases} \dot{x} = (A - BK)x + BKx_r \\ y = Cx \end{cases} \qquad \text{(Eq.33)}$$

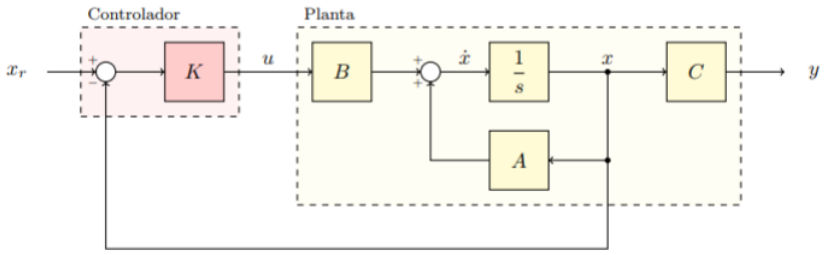


Figura 27 – Diagrama de controle realimentado para um eixo em espaço de estados (imagem retirada de slide da aula 7)

Adicionando realimentação de estados ao sistema pode-se conseguir uma resposta bastante rápida da dinâmica dos estados. Porém na prática tal feito não é recomendado pois polos muito rápidos levam a ganhos elevados e grande esforço de controle possibilitando saturação da entrada assim como desperdícios de energia. O controle LQR busca resolver esse problema permitindo obter um compromisso entre velocidade de resposta e esforço de controle. Essa eficiência é obtida pela minimização da função

$$J = \int_0^{+\infty} (x^T Q x + u^T R u) dt \qquad \text{(Eq.34)}$$

em que J é uma função de custo do controle, x é o vetor das variáveis de estado, u é o vetor das entradas do sistema e representa o esforço de controle despendido e Q e R são matrizes de pesos.

A função J é minimizada se $K = R^{-1}B^T P$, na qual a matriz P_{nxm} é a solução da equação: $A^T P + PA - PBR^{-1}B^T P + Q = 0$, conhecida como equação de Riccati. Dado que o esforço de controle e a performance do sistema são diretamente proporcionais, ou seja, quanto maior um maior o outro, o desafio é encontrar o vetor de ganhos ótimos $K = [K_1\ K_2 \ldots K_i]$ que minimiza J, proporcionando assim o melhor desempenho com o menor esforço de controle.

O software Matlab permite resolver com facilidade este problema utilizando a sintaxe $[K, P, E] = lqr(A, B, Q, R)$, sendo K o vetor de ganhos, P a solução das equações de Riccati e E os autovalores em malha fechada do regulador linear quadrático. As matrizes de pesos Q e R, passadas para a função do Matlab, são escolhidas para ponderar cada elemento do vetor de estados e do vetor de esforço de controle de acordo com a necessidade de projeto. Existe um método para definir os valores iniciais dos pesos dessas matrizes, este mostrado abaixo:

$$Q_{ii} = \frac{1}{máximo\ valor\ desejado\ de\ x_i^2}$$

$$R_{jj} = \frac{\rho}{máximo\ valor\ desejado\ de\ u_j^2}$$

O ρ é ajustado de modo que:

- Escolhendo ρ grande, enfatiza-se a economia de energia e esforço de controle às custas de um maior erro (polos se afastam, sistema responde rapidamente e tem-se alto gasto de energia).
- Escolhendo ρ pequeno, o erro é reduzido às custas de um maior esforço de controle (polos se movem pouco, sistema responde mais lentamente, energia gasta é mais baixa).

3. Metodologia

Nesta sessão serão descritos os procedimentos realizados em laboratório para a determinação de alguns parâmetros e constantes necessários para a implementação do controle.

3.1. Relação entre velocidade angular e PWM

Como foi mencionado anteriormente na seção 2.2.3, o fato do motor ser acionado por um PWM torna necessária a criação de um algoritmo que receba como parâmetro a velocidade angular desejada para as hélices e que tenha como saída o valor de PWM necessário para se obter essa rotação. Para isso, realizou-se um experimento em laboratório no qual se passava um valor de PWM pré-definido e media-se com um tacômetro (Figura 28) a velocidade angular de saída das hélices (em rpm), à fim de preencher a Tabela 2.

Figura 28 - Tacômetro utilizado (imagem retirada de roteiro do laboratório 2)

Visando minimizar erros, fez-se três medições para cada valor de PWM. Os resultados obtidos são apresentados na tabela abaixo:

Tabela 2 - Dados coletados de PWM por Velocidade angular

PWM	1	2	3	MÉDIA RPM	$\frac{rad}{s}$
0,1	9466	9885	9239	9530	997,979
0,2	13912	13245	12943	13366,67	1399,754
0,3	15267	15385	14870	15174	1589,018
0,4	17864	18297	18007	18056	1890,82
0,5	20465	20625	20432	20507,33	2147,523
0,6	22227	22191	21916	22111,33	2315,493
0,7	23241	23499	23344	23361,33	2446,393
0,8	24546	24510	24209	24421,67	2557,431
0,9	27544	27403	27079	27342	2863,248
1	27980	28128	27711	27939,67	2925,835

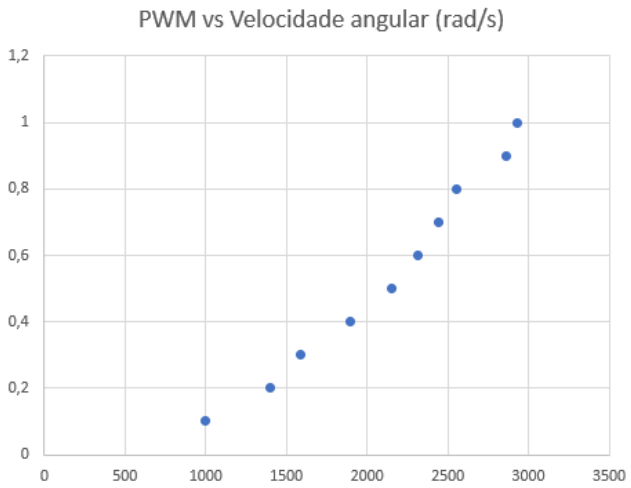


Figura 29 - Dados coletados de PWM por Velocidade angular

Os parâmetros α e β da equação (Eq.8) podem então ser estimados a partir de uma regressão quadrática. Pelas limitações de valores que esses coeficientes podem ter, descritas no final da seção 2.2.3, foi utilizada a ferramenta de ajuste de curva do Matlab:

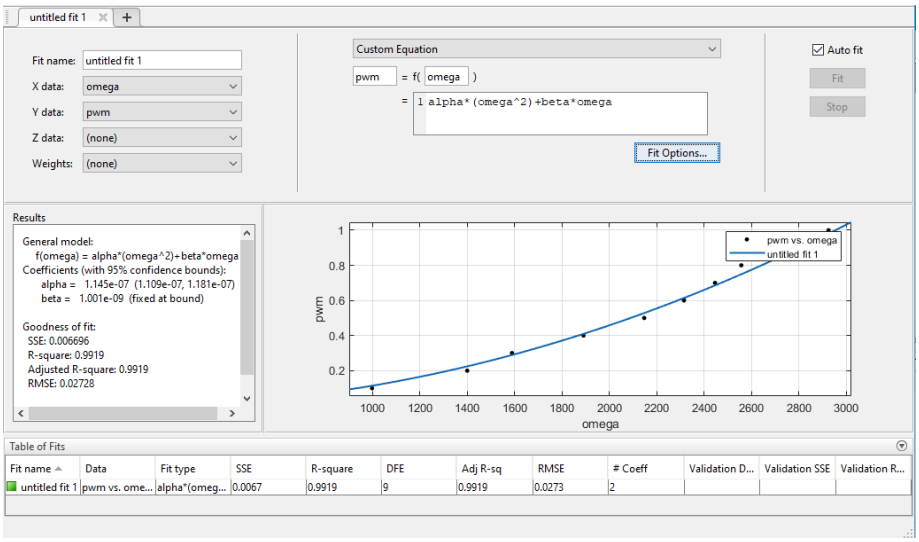


Figura 30 – Cálculo das constantes α e β usando ferramenta do Matlab

$$\begin{cases} \alpha = 1{,}145 \cdot 10^{-7} \\ \beta = 1{,}001 \cdot 10^{-9} \end{cases}$$

3.2. Coeficiente de sustentação

Para determinar este coeficiente, utilizou-se o dispositivo apresentado na figura abaixo para limitar o movimento do drone.

Figura 31 - Dispositivo de rolagem

Com esse dispositivo, sabendo-se o peso do drone, o peso da haste, as dimensões da haste e o ângulo de rolagem gerado devido à uma determinada rotação das hélices do drone é possível determinar o coeficiente de sustentação. A leitura desse ângulo de rolagem pode ser realizada na lateral do dispositivo, como é mostrado na figura abaixo:

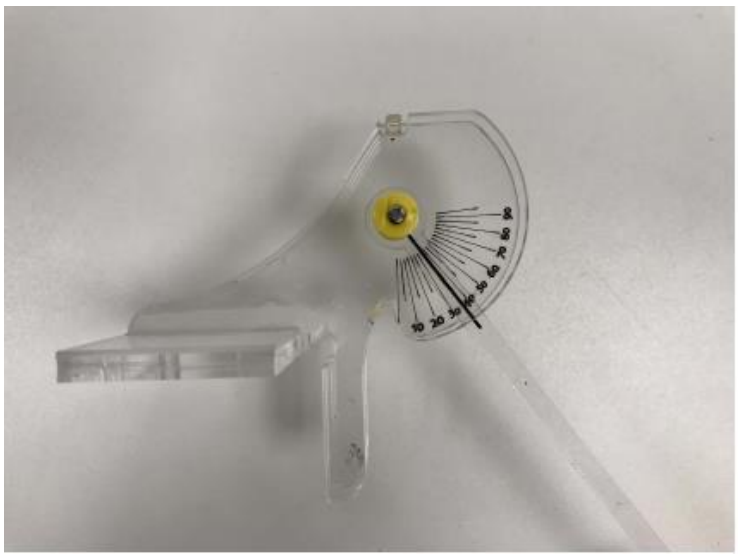


Figura 32 - Leitura do ângulo de rolagem

Neste experimento foi utilizado como entrada diversas velocidades angulares para os motores e, com isso, mediu-se as angulações de rolagem resultantes, preenchendo a tabela abaixo.

Tabela 3 - Dados coletados - Ângulo de rolagem

$\frac{rad}{s}$	1	2	3	$\phi\,(rad)$
1000	16	15	16	0,273
1200	21	21	20	0,361
1400	31	31	31	0,541
1600	40	40	40	0,698
1800	51	50	51	0,884
2000	70	70	69	1,216

Os ângulos de rolagem obtidos podem então ser utilizados para calcular a força f_l respectiva a cada velocidade angular da tabela. O diagrama abaixo demonstra o esquema das forças envolvidas no sistema e, com base nele, é obtida a relação entre f_l e ϕ:

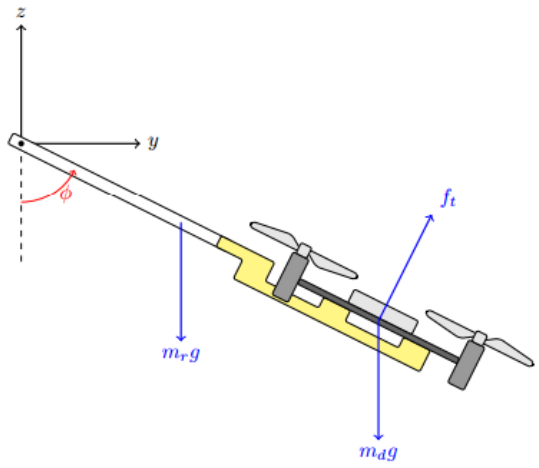


Figura 33 – Diagrama de forças do sistema drone/suporte de rolagem (imagem retirada de roteiro do laboratório 3)

$$\begin{cases} f_t l_d - m_d g l_d \sin\phi - m_r g l_r \sin\phi = 0 \\ f_t = f_1 + f_2 + f_3 + f_4 = 4f_l \end{cases}$$

$$f_l = \frac{\mathrm{g} \cdot \sin\phi}{4l_d}(m_d l_d + m_r l_r) \qquad \text{(Eq.35)}$$

Os parâmetros envolvidos na equação 35 estão presentes na tabela abaixo:

Tabela 4 - Parâmetros para equação de força f_l

Parâmetro	Valor
m_d	$30 \cdot 10^{-3}\ kg$
m_r	$10 \cdot 10^{-3}\ kg$

$\boldsymbol{l_d}$	$19 \cdot 10^{-2}\ m$
$\boldsymbol{l_r}$	$10 \cdot 10^{-2}\ m$
$\boldsymbol{g}$	$9{,}81\ m/s^2$

Utilizando os dados da Tabela 3 e os parâmetros da Tabela 4, têm-se:

Tabela 5- Dados calculados f_l

$\boldsymbol{f_l\ (N)}$
2,34E-02
3,05E-02
4,45E-02
5,56E-02
6,69E-02
8,11E-02

Por fim, com os dados da Tabela 5 e sabendo a relação entre f_l e ω, presente na equação (Eq.4), pode-se calcular a constante de sustentação k_l. Os resultados estão presentes abaixo:

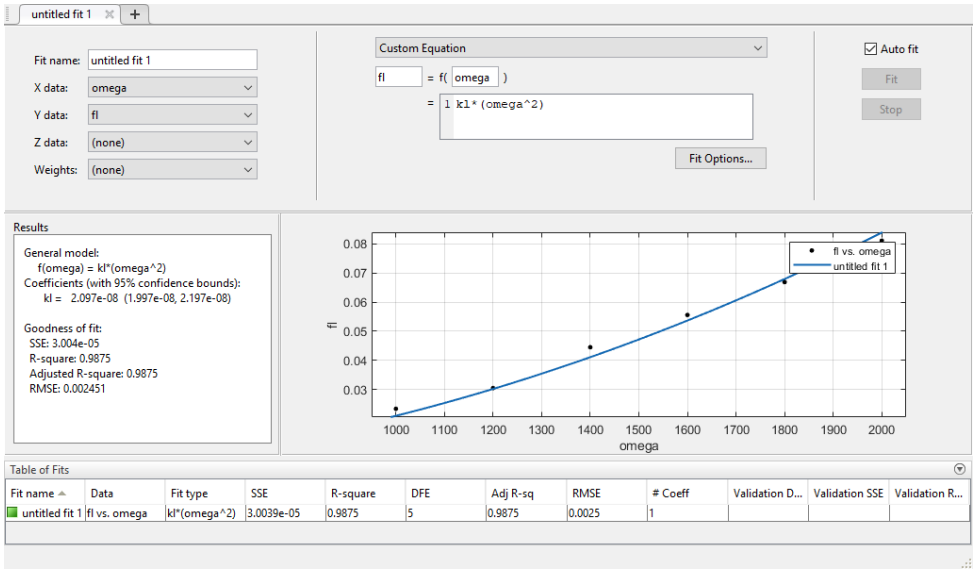


Figura 34 – Cálculo da constante k_l usando ferramenta do Matlab

$$k_l = 2{,}097 \cdot 10^{-8}\ N \cdot s^2/rad^2$$

3.3. Coeficiente de arrasto

O último coeficiente a ser determinado antes de implementar o controle é a constante de arrasto das hélices. Para tal, primeiramente o drone foi preso a uma estrutura que limita seus movimentos para apenas rotações

em torno do eixo z, denominada de dispositivo de guinagem, como mostra esta imagem:

Figura 35 – Drone preso ao dispositivo de guinagem

Essa restrição de movimento permite analisar apenas a contribuição dos torques de arrasto na rotação do drone como um todo. A ideia do experimento consiste em acionar os motores 2 e 4 do drone com uma tensão maior que a dos motores 1 e 3, fazendo com que o drone em si gire em sentido anti-horário, mesmo sentido de rotação das hélices 2 e 4, e a partir disso medir, com o auxílio de uma câmera com alta taxa de quadros por segundo, o tempo decorrido dentre cada quarto de volta do drone desde o começo do movimento até ter completado duas voltas inteiras.

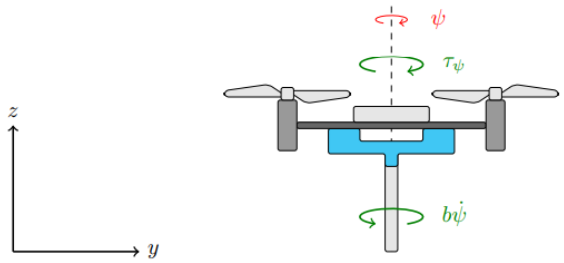
Figura 36 – Filmagem do drone rotacionando no dispositivo de guinagem

Foram definidas velocidades angulares de $1000\ rad/s$ para os motores 1 e 3 e de $2000\ rad/s$ para os motores 2 e 4 e os dados obtidos após o experimento são apresentados na tabela abaixo:

Tabela 6 – Dados Coletados – Tempo de rotação de guinagem

$\boldsymbol{\phi(rad)}$	**1**	**2**	**3**	$\boldsymbol{t(s)}$
$\mathbf{0}$	0	0	0	0
$\boldsymbol{\pi/2}$	0,31	0,33	0,38	0,34
$\boldsymbol{\pi}$	0,4	0,49	0,52	0,47
$\boldsymbol{3\pi/2}$	0,58	0,7	0,65	0,6433
$\boldsymbol{2\pi}$	0,67	0,76	0,82	0,75
$\boldsymbol{5\pi/2}$	0,8	0,85	0,87	0,84
$\boldsymbol{3\pi}$	0,88	0,95	0,97	0,9333
$\boldsymbol{7\pi/2}$	0,97	1,04	1,1	1,0367
$\boldsymbol{4\pi}$	1,1	1,12	1,16	1,1267

Sabe-se que o ângulo de guinada do drone está diretamente relacionado com o torque de arrasto das hélices e este com a velocidade angular dos motores e a constante de arrasto, como mostrado na equação (Eq.4). Contudo, não se sabe ainda a expressão exata que relaciona essas grandezas, de modo que convêm, assim como na seção anterior, realizar uma modelagem mecânica do sistema.



ψ – Ângulo de guinagem (rad)
τ_ψ – Torque total das hélices em torno do eixo z (N · m)
I_{zz} – Momento de inércia do drone em torno do eixo z (kg · m²)
b – Coeficiente de atrito viscoso do suporte (N · m · s/rad)

Figura 37 – Diagrama de torques do sistema drone/suporte de guinagem (imagem retirada do roteiro do laboratório 4)

$$I_{zz}\ddot{\psi}(t) + b\dot{\psi}(t) = \tau_\psi \qquad \text{(Eq.36)}$$

Após um pouco de abstração algébrica e transformadas de Laplace, tem-se que a expressão em função do tempo $\psi(t)$ é

$$\psi(t) = \frac{\tau_\psi}{b}t + \frac{\tau_\psi I_{zz}}{b^2}e^{-\frac{b}{I_{zz}}t} - \frac{\tau_\psi I_{zz}}{b^2} \qquad \text{(Eq.37)}$$

e considerando que

$$\tau_\psi = -\tau_1 + \tau_2 - \tau_3 + \tau_4$$
$$\tau_\psi = -k_d\omega_1^2 + k_d\omega_2^2 - k_d\omega_1^2 + k_d\omega_2^2$$
$$\tau_\psi = 2k_d(\omega_2^2 - \omega_1^2) \qquad \text{(Eq.38)}$$

a equação 37 torna-se

$$\psi(t) = \frac{2k_d(\omega_2^2 - \omega_1^2)}{b}t + \frac{2k_d(\omega_2^2 - \omega_1^2)I_{zz}}{b^2}e^{-\frac{b}{I_{zz}}t} - \frac{2k_d(\omega_2^2 - \omega_1^2)I_{zz}}{b^2}. \qquad \text{(Eq.39)}$$

Os parâmetros conhecidos dessa equação estão listados abaixo:

Tabela 7 – Parâmetros para equação de ângulo de guinagem

Parâmetro	Valor
$\boldsymbol{I_{zz}}$	$2{,}9 \cdot 10^{-5}\ kg \cdot m^2$
$\boldsymbol{b}$	$4 \cdot 10^{-5}\ N \cdot m \cdot s/rad$
$\boldsymbol{\omega_1}$	$1000\ rad/s$
$\boldsymbol{\omega_2}$	$2000\ rad/s$

Com isso e novamente com o auxílio da ferramenta de ajuste de curva do Matlab, pode então ser calculada a constante de arrasto k_d:

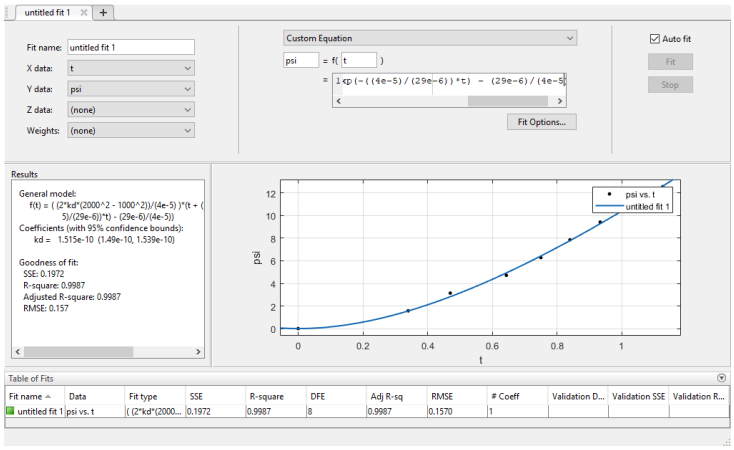


Figura 38 – Cálculo da constante k_d usando ferramenta do Matlab

$$k_d = 1{,}515 \cdot 10^{-10}\ N \cdot m \cdot s^2/rad^2$$

3.4. Ganhos dos controladores

Como mencionado na seção 2.6.2, o controle de atitude escolhido para este projeto foi o regulador linear quadrático (LQR) e os ganhos ótimos para o sistema foram obtidos através da função $lqr(A, B, Q, R)$ do Matlab, demonstrado em mais detalhas na .

$$K = \sigma[K_p \quad K_d] \qquad \text{(Eq.40)}$$

Sendo σ uma um constante responsável pelo cancelamento das constantes de entrada das dinâmicas da planta $(m;\ I_{xx}; I_{yy}; I_{zz}; g)$, possibilitando que todos os controladores sejam definidos considerando uma mesma representação em espaço de estados.

Os pesos das matrizes Q e R foram escolhidos por meio de validação experimental dos ganhos obtidos, uma vez que a planta linearizada usada neste projeto não envolve alguns fenômenos físicos relevantes, como por exemplo a dinâmica dos motores, a interdependência das rotações e o efeito do ângulo de guinada no deslocamento, dentre outros, o que impede que uma simulação válida da resposta da planta com o controlador seja efetuada.

3.4.1. Controlador de atitude

Para levantamento dos ganhos do controlador de atitude o drone foi preso a um suporte que limita seus movimentos para apenas ângulos de arremesso.

Figura 39 – Drone preso ao dispositivo de inclinação

Com isso, é possível observar o comportamento do sistema controlado de forma mais segura enquanto ainda não se tem certeza quanto ao desempenho esperado. O suporte em si possui um certo atrito viscoso, de modo que o comportamento que for observado do drone não será exatamente igual ao do drone sem estar preso no suporte, mas ainda é uma boa aproximação de qualquer forma.

Após alguns testes, as matrizes de pesos que proporcionaram os melhores resultados foram

$$\begin{cases} Q = \begin{bmatrix} 1 & 0 \\ 0 & 0{,}04 \end{bmatrix} \\ R = 10^{-4} \end{cases}$$

Isso indica que a posição angular do drone foi priorizada acima da velocidade angular e ambas acima do esforço de controle, o que significa que o sistema está procurando atingir a referência de forma mais rápida com um maior esforço nos motores, embora não se tenha notado uma saturação no acionamento dos motores.

Os ganhos proporcionados por esses pesos são então:

$$\begin{cases} K_p = 100 \\ K_d = 24{,}4 \end{cases}$$

3.4.2. Controlador vertical

Para levantamento dos ganhos do controlador vertical o drone foi preso a um suporte que permitia apenas o deslocamento vertical em z.

Figura 40 – Drone preso ao suporte vertical e estabilizado na posição de referência

Com esse sistema é possível observar, novamente, o comportamento do sistema de forma segura. Como este suporte também apresenta um certo atrito viscoso, o comportamento do drone em voo livre será de certa forma diferente, porém os ganhos encontrados podem ser considerados uma boa aproximação.

Após alguns testes, as matrizes de pesos que proporcionaram os melhores resultados foram

$$\begin{cases} Q = \begin{bmatrix} 1 & 0 \\ 0 & 0{,}04 \end{bmatrix} \\ R = 10^{-3} \end{cases}$$

Isso indica que a posição vertical do drone foi priorizada acima da velocidade no eixo z e ambas acima do esforço de controle, o que significa que o sistema procurará atingir a referência de forma mais rápida com um maior esforço nos motores, embora este ligeiramente menor do que o esforço exigido pelo controlador de atitude.

Os ganhos proporcionados por esses pesos são então:

$$\begin{cases} K_p = 32 \\ K_d = 10 \end{cases}$$

3.4.3. Controlador horizontal

Para levantamento dos ganhos do controlador horizontal o drone foi mantido completamente solto, sem o auxílio de nenhum dispositivo para restringir seus movimentos, principalmente porque a sintonia destes ganhos depende do comportamento do drone quanto à atitude à vertical.

Desta vez, o método do LQR não foi aplicado tentando obter os ganhos de melhor eficiência, uma vez que os demais ganhos e as constantes de ponderação ainda necessitam de um ajuste mais fino, de modo que se optou apenas por um chute inicial. As matrizes de pesos selecionadas foram

$$\begin{cases} Q = \begin{bmatrix} 1 & 0 \\ 0 & 0{,}444 \end{bmatrix} \\ R = 0{,}6064 \end{cases}$$

E os ganhos correspondentes são

$$\begin{cases} K_p = 1{,}28 \\ K_d = 1{,}6 \end{cases}$$

4. Implementação dos códigos

Definidas todas as constantes necessárias para a descrição do sistema, as funções de controle podem finalmente ser implementadas em código, utilizando a plataforma mbed na linguagem C++.

Cada bloco do diagrama de controle (Figura 1) será desenvolvida dentro de uma classe própria, por exemplo uma classe só do mixer, uma só do estimador de atitude e assim por diante, contendo suas próprias bibliotecas, variáveis e funções públicas e privadas, visando assim organizar a implementação e facilitar futuras alterações do código.

4.1. Mixer

São necessárias somente duas funções para criar o mixer: uma que converte a força total e os torques de rolagem, arfagem e guinagem para as velocidades angulares dos motores e uma que converte essas velocidades em PWM para acionar os motores corretamente.

A função mais simples a ser desenvolvida é a que converte velocidade angular em sinal de PWM, envolvendo apenas o emprego da equação (Eq.8) obtida na seção 2.2.3 e está disponível na Figura 47.

A outra função, por sua vez, exige um pouco de dedução matemática. A partir das equações (Eq.1) e (Eq.4) e do valor da distância entre os motores e os eixos na Tabela 1, pode-se representar relações entre a força total e os torques da seguinte forma:

$$\begin{bmatrix} f_t \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} k_l & k_l & k_l & k_l \\ -k_l l & -k_l l & k_l l & k_l l \\ -k_l l & k_l l & k_l l & -k_l l \\ -k_d & k_d & -k_d & k_d \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \qquad \text{(Eq.41)}$$

Já que é desejado obter a relação inversa, na qual o vetor com a força e os torques é a entrada e o vetor das velocidades a saída, inverte-se a matriz das constantes, resultando em

$$\begin{bmatrix}\omega_1^2\\ \omega_2^2\\ \omega_3^2\\ \omega_4^2\end{bmatrix} = \begin{bmatrix}\frac{1}{4k_l} & -\frac{1}{4k_l l} & -\frac{1}{4k_l l} & -\frac{1}{4k_d}\\ \frac{1}{4k_l} & -\frac{1}{4k_l l} & \frac{1}{4k_l l} & \frac{1}{4k_d}\\ \frac{1}{4k_l} & \frac{1}{4k_l l} & \frac{1}{4k_l l} & -\frac{1}{4k_d}\\ \frac{1}{4k_l} & \frac{1}{4k_l l} & -\frac{1}{4k_l l} & \frac{1}{4k_d}\end{bmatrix}\begin{bmatrix}f_t\\ \tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix} \quad \text{(Eq.42)}$$

Com essa equação matricial representada acima é possível implementar uma função que receba os valores de força e torques e retorne as velocidades angulares de cada um dos motores. Essa função pode ser vista na Figura 48 - Força e torques para velocidade angular.

Além disso, unindo essa função com a criada anteriormente que converte velocidade angular em PWM, tem-se uma função, disponível na Figura 49 – Força e torques para PWM, que obtém direto os valores de PWM de cada motor para que o drone realize a força e torques exigidos.

Com isso, o mixer está finalizado e pronto para ser validado. Sua estrutura de classe pode ser vista na Figura 46 – Funções públicas e privadas - Mixer.

4.2. Estimador de atitude

Depois de obtidas as equações mencionadas na seção 2.3, é possível implementá-las no microcontrolador a partir das leituras do acelerômetro e do giroscópio, no intuito de se obter a estimação de atitude do drone. A estrutura de classe desta seção está visível na Figura 50 e a descrição do processo de implementação é descrito logo a seguir.

4.2.1. Calibração do giroscópio

A função de calibração do giroscópio é necessária visto que o mesmo possui um erro sistemático constante, que pode ser diferente para cada um dos eixos e, ao integrar o valor de velocidade retornado pelo giroscópio, o erro consequentemente se propagará no resultado, para

evitar que isso ocorra utiliza-se uma função para estimar o valor desse erro.

Quando esta função for iniciada, o drone deverá estar parado, portanto, com velocidades zero nas 3 direções, desta forma, o valor de medição esperado do giroscópio será também zero, entretanto, devido ao erro mencionado, esse valor não será exatamente o esperado, por isso, tira-se uma média de 200 medições para cada um dos eixos com intervalo de 5ms e atribui este valor a três variáveis diferentes, representando o erro médio para cada um dos eixos. A equação implementada pode ser vista abaixo e o código está disponível na Figura 51:

$$\begin{cases} \tilde{p} = \dfrac{1}{200}\displaystyle\sum_{i=1}^{200} g_{x_i} \\ \tilde{q} = \dfrac{1}{200}\displaystyle\sum_{i=1}^{200} g_{i_i} \\ \tilde{r} = \dfrac{1}{200}\displaystyle\sum_{i=1}^{200} g_{z_i} \end{cases} \qquad \text{(Eq.43)}$$

4.2.2. Estimador do giroscópio

Na função do estimador do giroscópio, define-se inicialmente os valores de velocidade obtidos pela IMU para cada um dos eixos menos o erro calculado com a função de calibração, através da seguinte equação:

$$\begin{cases} \hat{p} = g_x - \tilde{p} \\ \hat{q} = g_y - \tilde{q} \\ \hat{r} = g_z - \tilde{r} \end{cases} \qquad \text{(Eq.44)}$$

Estes valores obtidos são na verdade as velocidades em cada eixo referentes aos eixos do sistema de coordenadas móvel, portanto, utiliza-se a equação (Eq.13) para calcular os valores dos ângulos em relação aos eixos do sistema de coordenadas fixo.

Vale ressaltar que o método de estimação dos ângulos pela medida do acelerômetro só retorna valores entre $\pm\,\pi\ rad$, logo, é necessário garantir que a estimação feita pelo giroscópio também permaneça dentro desse

intervalo. A solução adotada pelo grupo está disponível junto com os códigos de implementação na Figura 52.

4.2.3. Estimador do acelerômetro

O estimador do acelerômetro tem uma implementação mais simples do que o giroscópio, pois é necessário simplesmente implementar as equações (Eq.11) e (Eq.12), com a única ressalva de que uma correção de sinal na operação de raiz quadrada deve ser feita para casos em que o valor de a_z seja menor que zero. O código está disponível na Figura 53.

4.2.4. União dos estimadores

Como foi mencionado anteriormente, tanto o acelerômetro quanto o giroscópio apresentam ruídos em suas medições, dentre outras limitações de cada sensor, desta forma é necessária a criação de uma ponderação entre as leituras dos ângulos de Euler, que pode ser realizada criando uma constante de ponderação ρ que definirá qual dos sensores tem mais peso sobre aquela medida especifica.

É importante relembrar que o ângulo de guinagem é medido apenas pelo giroscópio, desta forma, não há nenhuma ponderação a ser feita nesse caso, de modo que ele estará sujeito ao acumulo de erro proveniente da estimação pelo giroscópio. As equações abaixo indicam a construção dessa função, que também podem ser vistas na Figura 54.

$$\begin{cases} \hat{\phi} = \rho\hat{\phi}_a + (1-\rho)\hat{\phi}_g \\ \hat{\theta} = \rho\hat{\theta}_a + (1-\rho)\hat{\theta}_g \\ \hat{\psi} = \hat{\psi}_g \end{cases} \qquad \text{(Eq.45)}$$

4.3. Estimador vertical

Como foi mencionado anteriormente na seção 2.4, será implementada duas etapas para o estimador vertical. A primeira delas, de predição, tem suas entradas e saídas indicadas na figura abaixo:

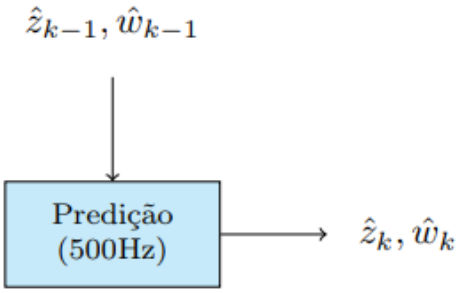


Figura 41 – Diagrama da etapa de predição do estimador vertical (imagem retirada do roteiro do laboratório 8)

Sendo,

$\hat{z}_k$ a altura *z* prevista da iteração atual de acordo com o modelo linearizado

$\widehat{w}_k$ a velocidade vertical prevista da iteração atual de acordo com o modelo linearizado

$\hat{z}_{k-1}$ *e* $\widehat{w}_{k-1}$ os valores da iteração anterior de cada um dos parâmetros acima.

Como demonstrado na seção 2.6.1, a dinâmica linearizada do sistema considerando deslocamento em *z* pode ser descrita da seguinte forma:

$$\begin{bmatrix} \dot{z} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix} \quad \text{(Eq.46)}$$

Assumindo essa relação em um domínio discreto de tempo com velocidade vertical constante, esse sistema adquire o este formato:

$$\begin{bmatrix} z \\ w \end{bmatrix}_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix}_{k-1} \quad \text{(Eq.47)}$$

Dessa forma, a etapa de predição pode ser implementada a partir dessas equações:

$$\begin{cases} \hat{z}_k = \hat{z}_{k-1} + \widehat{w}_{k-1} \Delta t_{predição} \\ \widehat{w}_k = \widehat{w}_{k-1} \end{cases} \quad \text{(Eq.48)}$$

Vale ressaltar que esse processo isoladamente só é válido quando a velocidade vertical é de fato constante, o que na prática raramente acontece. Contudo, a etapa de correção serve justamente para

compensar esse problema ao basear sua estimação no sensor de proximidade no lugar do modelo.

Essa etapa utiliza além dos valores obtidos na predição, os ângulos obtidos pelo estimador de atitude e a leitura do sensor LIDAR, como pode ser visto na figura abaixo:

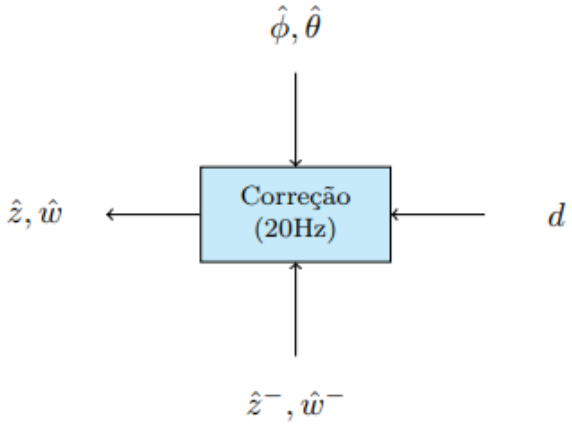


Figura 42 – Diagrama da etapa de correção do estimador vertical (imagem retirada do roteiro do laboratório 8)

A equação (Eq.20) é empregada para obter o valor aproximado da verdadeira altura do drone em relação ao solo, enquanto que a velocidade vertical é obtida integrando-se essa altura em relação ao último valor medido pelo intervalo de tempo entre as correções:

$$\begin{cases} z_m = d \cdot \cos\hat{\phi} \cdot \cos\hat{\theta} \\ w_m = \dfrac{z_m - z_{m,last}}{\Delta t_{correção}} \end{cases} \quad \text{(Eq.49)}$$

Por fim, assim como foi realizado no estimador de atitude, utiliza-se uma constante de ponderação para obter o melhor resultado para a medição, nesse caso, ponderando entre os valores obtidos na etapa de predição e de correção, seguindo as equações abaixo:

$$\begin{cases} \hat{z} = (1-\rho) \cdot \hat{z}^- + \rho \cdot z_m \\ \hat{w} = (1-\rho) \cdot \hat{w}^- + \rho \cdot w_m \end{cases} \quad \text{(Eq.50)}$$

4.4. Estimador Horizontal

Da mesma maneira que na seção anterior e como já foi discutido na seção 2.5, serão utilizados dois métodos para estimar as posições e velocidades horizontais: Predição e Correção.

A predição irá prever os valores desejados a partir do modelo linearizado. As entradas e saídas da função predição podem ser vistas na figura abaixo:

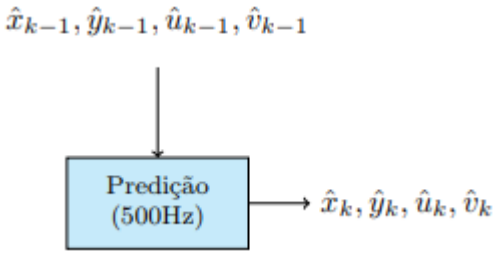


Figura 43 - Entradas e Saídas predição horizontal

Percebe-se que esta função utiliza apenas os valores estimados de uma iteração anterior para prever os valores atuais.

O modelo da dinâmica horizontal linearizada para o quadricóptero em questão, quando este está pairando é dado por:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ u \\ v \end{bmatrix} \qquad \text{(Eq.51)}$$

É importante notar que no sistema acima, as velocidades são consideradas constantes, ou seja, considera-se que o drone está sempre pairando, o que não é verdade, entretanto, é uma consideração adequada neste caso.

Ao discretizar o sistema apresentado acima para poder implementá-lo no controlador, obtém-se o seguinte sistema:

$$\begin{bmatrix} x \\ y \\ u \\ v \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ u \\ v \end{bmatrix}_{k-1} \qquad \text{(Eq.52)}$$

Com isso, a função de predição atualizará as posições e velocidades horizontais sempre que chamada.

A outra função é a de correção, que utiliza efetivamente as medidas do sensor para corrigir os valores obtidos na predição. As entradas e saídas dessa função estão presentes a seguir:

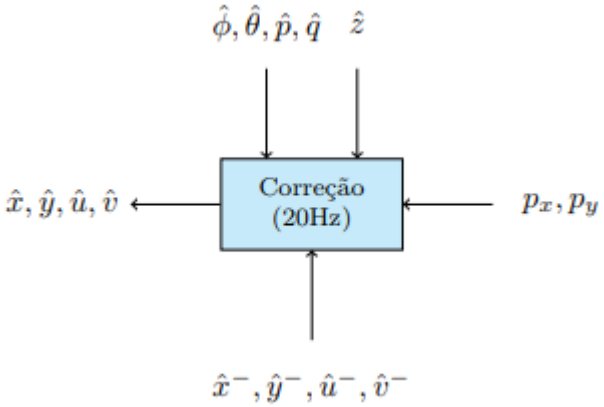


Figura 44 - Entradas e Saídas correção horizontal

Percebe-se que a correção utiliza diversos parâmetros como entrada, dentre eles os ângulos de Roll e Pitch, as velocidades angulares relativas a esses ângulos, além das posições e velocidades horizontais estimadas anteriormente e as velocidades em x e y, em unidades de pixels.

Utilizando os parâmetros fornecidos no Datasheet do sensor, e considerando que:

$$d = \frac{z}{\cos\hat{\phi}\cos\hat{\theta}} \qquad \text{(Eq.53)}$$

Pode-se obter as posições horizontais integrando as medições de velocidade, aplicando as seguintes equações:

$$\begin{cases} x_m = x_{ml} + u_m \cdot \Delta t \\ y_m = y_{ml} + v_m \cdot \Delta t \end{cases} \qquad \text{(Eq.54)}$$

Sendo x_{ml} e y_{ml} as posições da iteração anterior.

Por fim, assim como foi feito no estimador vertical, pondera-se os valores obtidos pelos dois métodos à fim de se obter uma medição mais precisa, da seguinte forma:

$$\begin{cases} \hat{x} = (1-\rho)\cdot\hat{x}^{-} + \rho\cdot x_m \\ \hat{y} = (1-\rho)\cdot\hat{y}^{-} + \rho\cdot y_m \\ \hat{u} = (1-\rho)\cdot\hat{u}^{-} + \rho\cdot u_m \\ \hat{v} = (1-\rho)\cdot\hat{v}^{-} + \rho\cdot v_m \end{cases} \quad \text{(Eq.55)}$$

Sendo ρ a constante de ponderação que leva em conta o peso de cada um dos métodos para o valor da estimação final.

4.5. Controladores

Uma vez obtidos os ganhos dos controladores, não há muito mais o que dizer sobre os controladores em si e as suas implementações. Suas classes e suas funções de controle foram construídas como mostram as figuras de Figura 61 a Figura 66 na seção Anexos.

5. Validação dos códigos

5.1. Validação do mixer

Para validação do mixer realiza-se diversos testes qualitativos, ou seja, fornece-se como entrada da função diferentes valores e observa-se o comportamento do drone para aqueles parâmetros. Os valores a serem inseridos e os comportamentos esperado estão descritos abaixo.

Inicialmente testa-se os seguintes valores:

$$\begin{bmatrix} f_t \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} 0{,}2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Com isso, espera-se que o drone exerça uma força sobre a superfície, mas não o suficiente para levantar voo. Além disso, o quadricoptero não deverá girar em nenhum dos três eixos, visto que os torques desejados são nulos.

As três matrizes de testes apresentadas a seguir têm como objetivo avaliar a rotação do drone em torno de cada um dos eixos ϕ, θ e ψ, respectivamente.

$$\begin{bmatrix} f_t \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} 0 \\ 0{,}005 \\ 0 \\ 0 \end{bmatrix}$$

Ao implementar estes valores, é esperado que apenas as hélices 3 e 4 rotacionem, o que produzirá um torque no eixo ϕ, ou seja, um movimento de rolagem (roll). Caso o sinal do valor seja modificado, as hélices 1 e 2 deverão rotacionar no lugar das outras.

$$\begin{bmatrix} f_t \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0{,}005 \\ 0 \end{bmatrix}$$

Já com esses valores, as hélices ativadas devem ser apenas a 2 e 3, produzindo um torque em θ, inclinando o drone (pitch). Outro teste que pode ser realizado é inverter o sinal do valor, que deverá resultar no movimento das hélices 1 e 4.

$$\begin{bmatrix} f_t \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0{,}001 \end{bmatrix}$$

Por último, as hélices que devem girar são as hélices 2 e 4, resultando em uma guinagem do quadricóptero. Novamente caso o sinal do torque seja invertido, as hélices acionadas serão as de número 1 e 3.

Após todos os testes serem realizados, o Mixer estará implementado e funcionando adequadamente.

5.2. Validação do estimador de atitude

Para validar o estimador de atitude implementa-se o código da Figura 67 no MATLAB. Este código realiza a leitura das variáveis de ângulos calculadas pelo estimador e plota os resultados dessas medidas em um gráfico 3D, como pode ser visto na figura abaixo.

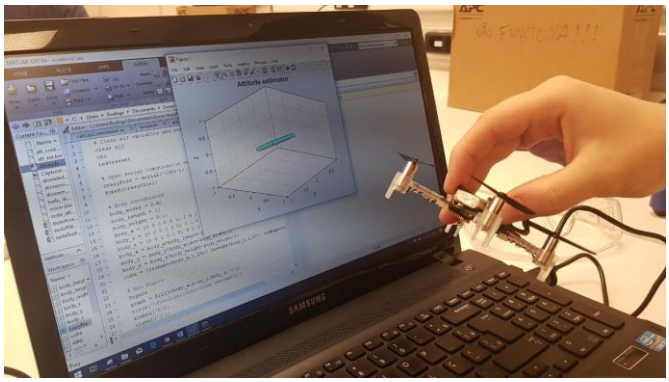


Figura 45 – Validação estimador de atitude

Desta forma, ao movimentar o drone, o gráfico atualiza automaticamente, simulando a orientação do quadricóptero, permitindo assim validar o estimador de atitude.

6. Conclusão

Como foi mencionado anteriormente, este é um relatório de um projeto em desenvolvimento, desta forma, não será possível apresentar as conclusões do projeto completo, entretanto, há como discutir os resultados acerca dos módulos já finalizados.

No caso do Mixer, primeiro módulo construído e relatado na seção 2.2, foi possível realizar a validação através dos experimentos da seção 5.1. Os resultados obtidos foram exatamente os esperados no que se refere ao comportamento qualitativo, visto que, à medida que os testes indicados na validação eram realizados, o drone apresentava o comportamento adequado.

Já o estimador de atitude que foi detalhado na seção *2.3*, foi possível realizar a validação através da implementação de um código no MATLAB que utilizava o estimador criado para plotar a posição do drone em um gráfico. O que pôde-se perceber é que o estimador funcionava de forma adequada para movimentações suaves do drone, entretanto, ao realizar movimentações muito bruscas o ângulo de *yaw* perdia a referência de posição, visto que este ângulo era medido apenas pelo giroscópio, não havendo uma ponderação que poderia corrigir este problema, como

acontece nos casos dos outros ângulos. Após algumas iterações experimentais, chegou-se em uma constante de ponderação igual a $0{,}01$.

O estimador vertical apresentado na seção 2.4, por outro lado, foi validado de forma mais simples, apenas verificando por comunicação serial se a altura estimada condiz com a distância real do drone de uma superfície e se essa altura se altera ao inclinar o drone em alguma direção. Os valores medidos condizem com o esperado, embora possua certas limitações, como por exemplo os limites de operação do sensor de proximidade e uma pequena alteração da estimação para ângulos muito grandes de rolagem e arfagem. Após algumas iterações experimentais, chegou-se em uma constante de ponderação igual a $0{,}3$.

Quanto ao estimador horizontal descrito na seção 2.5, os valores obtidos das velocidades horizontais u e v condizem bem com o esperado, identificando deslocamentos puramente lineares e cancelando de forma razoável o efeito das rotações, embora ainda haja um pouco de incerteza, talvez por causa da leitura do sensor ou pelo acúmulo de imprecisão das outras estimações. As posições horizontais x e y por outro lado apresentaram um pouco de derivação dos valores reais, talvez por dependerem apenas de integrações ao longo do processo de estimação, porém é algo que pode ser compensado pelo controlador. Após algumas iterações experimentais, chegou-se em uma constante de ponderação igual a $0{,}3$.

Os controladores vertical, horizontal e de atitude implementados segundo a seção 2.6, por sua vez, foram testados também qualitativamente. O que concluiu-se em relação aos controladores é que, utilizando o modelo linearizado do quadricoptero, não foi possível obter os ganhos de forma teórica e precisa, dada uma séria de fenômenos não representados, alguns dos quais citados na seção, portanto, os ganhos finais que foram implementados e considerados os mais adequados foram obtidos a partir de experimentos, primeiro utilizando as estruturas das figuras Figura 39 e Figura 40 e depois com o drone completamente livre tentando pairar no ar.

Estes experimentos se baseavam em implementar diferentes ganhos para o controlador e observar o comportamento do drone até que ele conseguisse se estabilizar na posição desejada, mesmo sob a influência de perturbações externas, o mais rápido possível. Ao final da implementação foram encontrados os ganhos considerados mais adequados que fizeram com que controlador conseguisse atender às expectativas, entretanto, foi difícil de validar de forma efetiva sua robustez.

Os ganhos obtidos após esse processo estão listados abaixo:

$$\begin{cases} K_{p,\phi} = 240{,}3 \\ K_{d,\phi} = 26{,}7 \\ K_{p,\theta} = 240{,}3 \\ K_{d,\theta} = 26{,}7 \\ K_{p,\psi} = 60{,}1 \\ K_{d,\psi} = 13{,}3 \\ K_{p,x} = 1{,}28 \\ K_{d,x} = 2{,}5 \\ K_{p,y} = 1{,}28 \\ K_{d,y} = 2{,}5 \\ K_{p,z} = 5{,}41 \\ K_{d,z} = 4 \end{cases}$$

Anexos

```
7  class Mixer
8  {
9      public:
10         // Class constructor
11         Mixer();
12         // Actuate motors with the desired force (N) and torques (N.m)
13         void actuate( float ft , float tau_phi , float tau_theta , float tau_psi );
14     private:
15         // Motors PWM outputs
16         PwmOut motor0, motor1, motor2, motor3;
17         // Angular velocities ( rad /s)
18         float omega0, omega1, omega2, omega3;
19         // Converts desired force (N) and torques (N.m) to angular velocities ( rad/s)
20         void ft_to_omega( float ft, float tau_phi, float tau_theta, float tau_psi );
21         // Converts desired angular velocity (rad/s) to PWM signal
22         float omega_to_pwm( float omega );
23 };
```

Figura 46 – Funções públicas e privadas - Mixer

```
9  float Mixer::omega_to_pwm(float omega)
10 {
11     float pwm = alpha*pow(omega,2) + beta*omega;
12     return pwm;
13 }
```

Figura 47 – Velocidade angular para PWM

```
15 void Mixer::ft_to_omega(float ft, float tau_phi, float tau_theta, float tau_psi)
16 {
17     omega0 = sqrt((1/(4*kl))*ft - (1/(4*kl*l))*tau_phi - (1/(4*kl*l))*tau_theta - (1/(4*kd))*tau_psi);
18     omega1 = sqrt((1/(4*kl))*ft - (1/(4*kl*l))*tau_phi + (1/(4*kl*l))*tau_theta + (1/(4*kd))*tau_psi);
19     omega2 = sqrt((1/(4*kl))*ft + (1/(4*kl*l))*tau_phi + (1/(4*kl*l))*tau_theta - (1/(4*kd))*tau_psi);
20     omega3 = sqrt((1/(4*kl))*ft + (1/(4*kl*l))*tau_phi - (1/(4*kl*l))*tau_theta + (1/(4*kd))*tau_psi);
21 }
```

Figura 48 - Força e torques para velocidade angular

```
23 void Mixer::actuate(float ft, float tau_phi, float tau_theta, float tau_psi)
24 {
25     ft_to_omega(ft,tau_phi,tau_theta,tau_psi);
26     motor_0 = omega_to_pwm(omega0);
27     motor_1 = omega_to_pwm(omega1);
28     motor_2 = omega_to_pwm(omega2);
29     motor_3 = omega_to_pwm(omega3);
30 }
```

Figura 49 – Força e torques para PWM

```
 9 class AttitudeEstimator
10 {
11     public:
12         // Class constructor
13         AttitudeEstimator();
14         // Initialize class
15         void init();
16         // Estimate Euler angles (rad) and angular velocities (rad/s)
17         void estimate();
18         // Euler angles (rad)
19         float phi, theta, psi;
20         // Angular velocities (rad/s)
21         float p, q, r;
22     private:
23         // IMU sensor object
24         MPU9250 imu;
25         // Calibrates gyroscope by calculating angular velocity bias (rad/s)
26         void calibrate_gyro();
27         // Estimate Euler angles (rad) from accelerometer data
28         void estimate_accel();
29         // Estimate Euler angles (rad) and angular velocities (rad/s) from gyroscope data
30         void estimate_gyro();
31         // Euler angles (rad) from accelerometer data
32         float phi_accel, theta_accel;
33         // Euler angles (rad) from gyroscope data
34         float phi_gyro, theta_gyro, psi_gyro;
35         // Angular velocities bias (rad/s)
36         float p_bias, q_bias, r_bias;
37 };
```

Figura 50 – Funções públicas e privadas - Estimador de atitude

```
19 void AttitudeEstimator::calibrate_gyro()
20 {
21     for(int i=0;i<200;i++){
22         imu.read();
23         p_bias += imu.gx;
24         q_bias += imu.gy;
25         r_bias += imu.gz;
26         wait_ms(5);
27     }
28     p_bias /= 200;
29     q_bias /= 200;
30     r_bias /= 200;
31 }
```

Figura 51 - Calibração do giroscópio

```
void AttitudeEstimator::estimate_gyro()
{
    p = imu.gx - p_bias;
    q = imu.gy - q_bias;
    r = imu.gz - r_bias;

    phi_gyro = phi + (p + sin(phi)*tan(theta)*q + cos(phi)*tan(theta)*r)*dt;
    theta_gyro = theta + (cos(phi)*q - sin(phi)*r)*dt;
    psi_gyro = psi + ((sin(phi)/cos(theta))*q + (cos(phi)/cos(theta))*r)*dt;

    if(phi_gyro > pi){
        phi_gyro -= 2*pi;
    }
    if(theta_gyro > pi){
        theta_gyro -= 2*pi;
    }
    if(psi_gyro > pi){
        psi_gyro -= 2*pi;
    }
}
```

Figura 52 - Estimador do giroscópio

```
void AttitudeEstimator::estimate_accel()
{
    phi_accel = atan2(-imu.ay,-imu.az);

    if(imu.az >= 0){
        theta_accel = atan2(imu.ax,-sqrt(pow(imu.ay,2) + pow(imu.az,2)));
    }else{
        theta_accel = atan2(imu.ax,sqrt(pow(imu.ay,2) + pow(imu.az,2)));
    }
}
```

Figura 53 - Estimador do acelerômetro

```
void AttitudeEstimator::estimate()
{
    imu.read();

    estimate_accel();
    estimate_gyro();

    phi = rho*phi_accel + (1.0f-rho)*phi_gyro;
    theta = rho*theta_accel + (1.0f-rho)*theta_gyro;
    psi = psi_gyro;
}
```

Figura 54 - União dos estimadores

```
class VerticalEstimator
{
    public:
        // Class constructor
        VerticalEstimator();
        // Initialize class
        void init();
        // Predict vertical position and velocity from model
        void predict();
        // Correct vertical position and velocity with measurement
        void correct(float phi,float theta);
        // Vertical position (m) and velocity (m/s) estimation
        float z,w;
    private:
        // Range sensor object
        VL53L0X range;
        // Last vertical position (m) measurement
        float z_m_last;
};
```

Figura 55 – Funções públicas e privadas – Estimador vertical

```
void VerticalEstimator::predict()
{
    z = z + w*dt_vert_pred_cont;
}
```

Figura 56 – Etapa de Predição do estimador vertical

```
void VerticalEstimator::correct(float phi,float theta)
{
    range.read();
    float z_m = range.z*cos(phi)*cos(theta);
    float w_m = (z_m - z_m_last)/dt_vert_corr;

    z = (1-rho_vert_corr)*z + rho_vert_corr*z_m;
    w = (1-rho_vert_corr)*w + rho_vert_corr*w_m;

    z_m_last = z_m;
}
```

Figura 57 – Etapa de correção do estimador vertical

```
class HorizontalEstimator
{
    public:
        // Class constructor
        HorizontalEstimator();
        // Initialize class
        void init();
        // Predict horizontal position and velocity from model
        void predict();
        // Correct horizontal position and velocity with measurements
        void correct(float phi,float theta,float p,float q,float z);
        // Horizontal positions (m) and velocities (m/s) estimations
        float x, y, u, v;
    private:
        // Flow sensor object
        PMW3901 flow;
        // Last horizontal positions (m) measurements
        float x_m_last, y_m_last;
};
```

Figura 58 – Funções públicas e privadas – Estimador horizontal

```
void HorizontalEstimator::predict()
{
    x = x + u*dt;
    y = y + v*dt;
}
```

Figura 59 – Etapa de predição do estimador horizontal

```
void HorizontalEstimator::correct(float phi,float theta,float p,float q,float z)
{
    float Cphi = cos(phi);
    float Ctheta = cos(theta);

    if( Cphi >= 0.7f && Ctheta >= 0.7f )
    {
        float d = z/(Cphi*Ctheta);

        flow.read();
        float u_m = (sigma*flow.x + q)*d;
        float v_m = (sigma*flow.y - p)*d;

        float x_m = x_m_last + u_m*dt_flow;
        float y_m = y_m_last + v_m*dt_flow;

        x = (1.0f - rho_horiz_corr)*x + rho_horiz_corr*x_m;
        y = (1.0f - rho_horiz_corr)*y + rho_horiz_corr*y_m;
        u = (1.0f - rho_horiz_corr)*u + rho_horiz_corr*u_m;
        v = (1.0f - rho_horiz_corr)*v + rho_horiz_corr*v_m;

        x_m_last = x_m;
        y_m_last = y_m;
    }
}
```

Figura 60 – Etapa de correção do estimador horizontal

```
class AttitudeController
{
    public:
        AttitudeController();

        void control(float phi_r, float theta_r, float psi_r, float phi, float theta, float psi, float p, float q, float r);

        float tau_phi, tau_theta, tau_psi;
    private:

        float single_axis_control(float angle_r, float angle, float rate, float K_angle, float K_rate, float I);
};
```

Figura 61 – Funções públicas e privadas – Controlador de atitude

```
AttitudeController::AttitudeController()
{
}

// Control torques given reference angles and current angles and angular velocities
void AttitudeController::control( float phi_r , float theta_r , float psi_r , float phi, float theta , float psi , float p, float q, float r)
{
    tau_phi = single_axis_control(phi_r, phi, p, phi_kp, phi_kd, I_xx);
    tau_theta = single_axis_control(theta_r, theta, q, theta_kp, theta_kd, I_yy);
    tau_psi = single_axis_control(psi_r, psi, r, psi_kp, psi_kd, I_zz);
}
// Control torque of a single axis given reference angles and current angles and angular velocities
// ( with given gains and /or time constants and moments of inertia )
float AttitudeController::single_axis_control( float angle_r , float angle , float rate , float K_angle , float K_rate , float I)
{
    return I*(K_angle*(angle_r - angle) - K_rate*rate);
}
```

Figura 62 – Controle de atitude

```
class VerticalController
{
    public:

        VerticalController();

        void control(float z_r, float z, float w);

        float f_t;
};
```

Figura 63 – Funções públicas e privadas – Controlador vertical

```
VerticalController::VerticalController()
{
    f_t = 0.0f;
}

void VerticalController::control(float z_r, float z, float w)
{
    f_t = m*( z_kp*( z_r - z ) - z_kd*w + g);
}
```

Figura 64 – Controle vertical

```
class HorizontalController
{
    public:

        HorizontalController();

        void control(float x_r, float y_r, float x, float y, float u, float v);

        float phi_r, theta_r;

    private:

        float control_single(float pos_r, float pos, float vel, float kp, float kd);
};
```

Figura 65 – Funções públicas e privadas – Controlador horizontal

```
HorizontalController::HorizontalController()
{
    phi_r = 0.0f;
    theta_r = 0.0f;
}

void HorizontalController::control(float x_r, float y_r, float x, float y, float u, float v)
{
    phi_r = control_single(y_r, y, v, y_kp, y_kd)/(-g);
    theta_r = control_single(x_r, x, u, x_kp, x_kd)/g;
}

float HorizontalController::control_single(float pos_r, float pos, float vel, float kp, float kd)
{
    return kp*(pos_r - pos) - kd*vel;
}
```

Figura 66 – Controle horizontal

```
% Clear all variables and screen
clear all
clc

% Open serial comunication with Crazyflie
crazyflie = serial('COM3');
fopen(crazyflie);

% Body coordinates
body_width = 0.6;
body_length = 1;
body_height = 0.1;
body_x = [0 0 0 0 0 1; 1 0 1 1 1 1; 1 0 1 1 1 1; 0 0 0 0 0 1];
body_y = [0 0 0 0 1 0; 0 1 0 0 1 1; 0 1 1 1 1 1; 0 0 1 1 1 0];
body_z = [0 0 1 0 0 0; 0 0 1 0 0 0; 1 1 1 0 1 1; 1 1 1 0 1 1];
body_x = body_x*body_length-body_length/2;
body_y = body_y*body_width-body_width/2;
body_z = body_z*body_height-body_height/2;
cube = [reshape(body_x,1,24); reshape(body_y,1,24); reshape(body_z,1,24)];

% New figure
figure
graph = fill3(body_x,body_y,body_z,'c');
title('\fontsize{16}Attitude estimator')
xlabel('X');
ylabel('Y');
zlabel('Z');
view(135,30);
axis([-1 1 -1 1 -1 1]);
grid on;
box on;

for i = 1:1000

    % Acquire Euler angles data
    fprintf(crazyflie,'%d\n',1);
    data = fscanf(crazyflie, '%f,%f,%f');

    % Convert Euler angles into rotation matrix
    R = angle2dcm(data(3),data(2),data(1),'zyx');

    % Update body coordinates
    V = R'*cube;

    % Update individual coordinates
    X=reshape(V(1,:),4,6);
    Y=reshape(V(2,:),4,6);
    Z=reshape(V(3,:),4,6);

    % Update graph data
    set(graph,'XData',X);
    set(graph,'YData',Y);
    set(graph,'ZData',Z);

    % Refresh graph
    drawnow

end

% Close graph
close all

% Close serial comunication with Crazyflie
fclose(crazyflie);
```

Figura 67 – Código MATLAB – Validação estimador de atitude

```
58 -    A = [0 1 ;0 0];
59 -    B = [0;1];
60 -    C = [1 0];
61 -    D = [0];
62
63 -    K = lqr(A,B,[1 0;0 1/25],1/10000);
```

Figura 68 – Código MATLAB – Cálculo dos ganhos dos controladores