UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Engenharia de Controle e Automação / Engenharia Mecatrônica
Sistemas Embarcados II / Sistemas Digitais para Mecatrônica
Prof. Éder Alves de Moura

Trabalho Final 01 – Drone 2D

Amanda Abhigail Bonilla Trochez	11711EMT027
Glênio Simião Ramalho	11611EMT008
Ísis Germiniani Teixeira	11811EMT019
Leonardo Mortari Borba	11611EMT001
Neila Cristina Morais Silva	12011EAU015

Uberlândia-MG
Março/2022

SUMÁRIO

1 INTRODUÇÃO

Este trabalho consiste na simulação de um drone em 2D, em ambiente python, utilizando das modelagens desenvolvidas ao longo do semestre. O drone deverá ser movido por meio de waypoints e utilizando o teclado. Para essa implementação, utilizou-se a biblioteca pygame.

2 BIBLIOTECAS UTILIZADAS

Nesta seção serão apresentadas as bibliotecas utilizadas no código da simulação.

2.1 PYGAME

A biblioteca Pygame é utilizada para o desenvolvimento de jogos, utilizando linguagem python e é baseada em SDL (Simple DirectMedia Layer). Por ser uma biblioteca relativamente simples, e com códigos em C e assembly para funções básicas, é amplamente utilizada. Além disso, possui portabilidade com diversos sistemas operacionais e é modular, além de não necessitar de uma GUI para acessar todas as suas funções. Fornece acesso a áudios, teclados, controles, mouses e hardwares gráficos via OpenGL e Direct3D.

Figura 1 - Logo Pygame

2.2 PIL (Python Imaging Library)

A biblioteca PIL é utilizada para manipulação de imagens em ambiente python. Em nosso trabalho, foi utilizada para redimensionar os planos de fundo e imagens para que ficassem de acordo com o tamanho da tela que utilizamos.

```
# Resizing background image to match the screen size
image = Image.open('Imagens/ghibli_background_main.jpg')
image = image.resize((tela_larg, tela_alt))
image.save('Imagens/ghibli_background_main_resized.jpg')

# Resizing menus images
# Play image
image_play_width = 200 # Set width size
image_play_height = 72 # Set height size
image = Image.open('Imagens/image_play.png')
image = image.resize((image_play_width, image_play_height))
image.save('Imagens/image_play_resized.png')
# Credits image
image_credits_width = 350 # Set width size
image_credits_height = 80 # Set height size
image = Image.open('Imagens/image_creditos.png')
image = image.resize((image_credits_width, image_credits_height))
image.save('Imagens/image_creditos_resized.png')
```

Figura 2 - Implementação da biblioteca PIL no trabalho

2.3 NumPy (Numerical Python)

Esta biblioteca é utilizada para fazer cálculos numéricos no ambiente python, por meio de arrays multidimensionais que são processados por meio de rotinas já predefinidas. Foi utilizada no nosso trabalho para realizar os cálculos do nosso sistema de controle PD.

3 CLASSES

Nesta seção serão apresentadas as classes utilizadas no código e seu funcionamento.

3.1 SCREEN

Nesta classe, definimos os parâmetros para a configuração da tela e a sua atualização. Na função “__init__”, setamos as configurações iniciais, tal qual o plano de fundo, altura e largura. Na função “update_screen”, os status do jogo são atualizados de acordo com a movimentação do drone, recebendo como argumentos a posição do drone.

```
def update_screen(self):
        # Screen configuration
        self.screen.fill((0, 0, 0))  # Clean the last screen to update the frames
        self.screen.blit(self.background, (0, 0))  # Load the bg at the (0, 0) position of the screen


        # Fonte
        fonte = pygame.font.SysFont('arial', 15, True, True)
        # Destino
        texto = f'Destino do drone: ({mx_real}, {my_real})'
        texto_formatado = fonte.render(texto, True, (255, 255, 255))
        self.screen.blit(texto_formatado, (10, 10))
        # Posição Atual
        texto = f'Posição atual: {position_}'
        texto_formatado = fonte.render(texto, True, (255, 255, 255))
        self.screen.blit(texto_formatado, (10, 30))
        # Velocidade Atual
        texto = f'Velocidade atual: {velocidade}'
        texto_formatado = fonte.render(texto, True, (255, 255, 255))
        self.screen.blit(texto_formatado, (10, 50))
        # Angulo Atual
        texto = f'Ângulo: {angle_}'
        texto_formatado = fonte.render(texto, True, (255, 255, 255))
        self.screen.blit(texto_formatado, (10, 70))
```

Figura 3 - Função update_screen

3.2 DRONE

Nesta classe, definimos as configurações iniciais do drone (posição, velocidade e angulação). Além disso, é nela que é feita a atualização da posição do drone, uma vez que a imagem é carregada a cada frame da simulação.

3.3 DRONE CONTROL

Nesta classe é realizado o controle do drone por meio da função execute_PID, que será posteriormente explicada. Primeiramente são definidos os parâmetros iniciais na função “__init__” e, depois, são definidos os controles de movimentação por mouse e teclado. Em seguida, na função pid_control, são utilizados o estado atual do drone, os erros de posição e ângulo para gerar a movimentação, utilizando o PID de acordo com o que programamos, até que o drone atinja a posição desejada.

```
def pid_control(self, destiny_x, destiny_y):
    self.eP = np.array([destiny_x - self.real_pos['x'], destiny_y -
self.real_pos['y']])
    if np.abs(self.eP[0]) > 0.2 or np.abs(self.eP[1]) > 0.2 or
np.abs(self.ePhi) > 0.1:
        self.x, self.eP, self.ePhi = execute_PID(self.x, [destiny_x,
destiny_y], t)
        # Converting from real coordinate to screen coordinate
        self.posH, self.posV = self.x[2] + larg / 2, alt - 100 - self.x[3]

        # Updating states vector
        self.angle = self.x[6]*180/np.pi
        self.v1, self.v2 = self.x[4], self.x[5]
        self.w1, self.w2 = self.x[0], self.x[1]
        self.ang_vel = self.x[7]

        self.position = [self.posH, self.posV]
        self.drone.drone_update(self.position, self.angle)

        ################ Print drone's status
        global position_, angle_, velocidade
        position_ = (self.x[2], self.x[3])
        angle_ = self.angle
        velocidade = (self.v1, self.v2)

        return True

    else:
        self.real_pos = {'x': -(larg / 2 - self.posH), 'y': alt - 100 -
self.posV}
        self.posH, self.posV = self.x[2] + larg / 2, alt - 100 - self.x[3]
        self.eP = np.array([destiny_x - self.real_pos['x'], destiny_y -
self.real_pos['y']])
        self.drone.drone_update(self.position, self.angle)
        return False
```

Figura 4 - Função pid_control

O controle pelo mouse é bem simples, o ponto da tela onde o mouse foi clicado é repassado como o ponto destino do drone que se torna a entrada do PID.

```
def mouse_control(self, destiny_x, destiny_y):
    return self.pid_control(destiny_x, destiny_y)
```

Figura 5 - Função mouse_control

Já o controle por teclado funciona de maneira semelhante, só que o ponto destino é definido por um passo de 100 pixeis na direção da tecla que foi apertada, por exemplo, apertando a tecla direcional direita ou A, o ponto destino se tornará 100 pixeis para a direita. Após definir o destino, a função “pid_control” é chamada.

```
def key_control(self):
    self.keys = pygame.key.get_pressed()
    self.real_pos = {'x': -(larg / 2 - self.posH), 'y': alt - 100 - self.posV}

    destiny_x, destiny_y = self.real_pos['x'], self.real_pos['y']
    if self.keys[pygame.K_LEFT] or self.keys[pygame.K_a]:
        destiny_x = self.real_pos['x'] - 100.0
    if self.keys[pygame.K_RIGHT] or self.keys[pygame.K_d]:
        destiny_x = self.real_pos['x'] + 100.0
    if self.keys[pygame.K_UP] or self.keys[pygame.K_w]:
        destiny_y = self.real_pos['y'] + 100.0
    if self.keys[pygame.K_DOWN] or self.keys[pygame.K_s]:
        destiny_y = self.real_pos['y'] - 100.0

    self.pid_control(destiny_x, destiny_y)
```

Figura 6 - Função key_control

3.4 GAME

Nesta classe, a simulação é executada e todos os métodos importante são executados infinitamente para que o simulador funcione. É definido o clock, para controlar o fps, e, desta forma, a tela é atualizada.

```python
def run(self):
    global t, FPS
    FPS = 600
    auto_move = False
    global mx_real, my_real
    mx_real, my_real = 0, 0
    while True:
        self.clock.tick(FPS)  # Game FPS
        t = self.clock.get_time() / 1000
        self.screen.update_screen()
```

Figura 7 - Atualização da tela de acordo com o clock

```python
for event in pygame.event.get():
    # To quit the game
    if event.type == QUIT:
        pygame.quit()
        exit()

    if event.type == pygame.MOUSEBUTTONDOWN:
        auto_move = True
        # Get the destiny's position from mouse click
        mx, my = pygame.mouse.get_pos()
        # Transform the mouse click point in real coordinates
        mx_real, my_real = -(larg / 2 - mx), alt - 100 - my
        # print(mx_real, my_real)

if auto_move:
    auto_move = self.control.mouse_control(mx_real, my_real)
else:
    self.control.key_control()

pygame.display.update()
```

Figura 8 - Eventos e funções de controle

4 CONTROLE DO DRONE

Para o controle do drone, foi utilizado um controlador do tipo PD. Que recebe o estado atual do drone, e com, as equações apresentadas no material disponibilizado pelo professor, calcula o estado futuro e os erros. A malha de controle apresentada é a seguinte.:

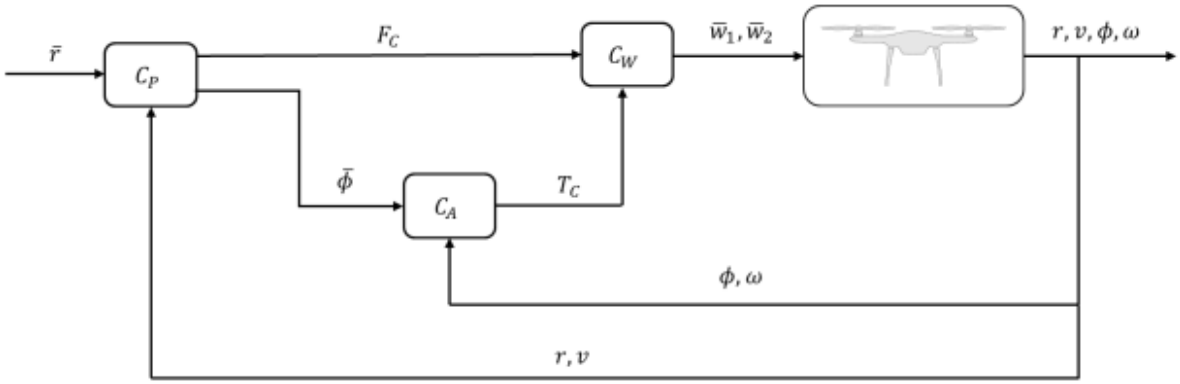

Figura 9 - Malha de controle do drone

Fonte: Material disponibilizado pelo professor Éder

Para a resolução de EDOs, foi utilizado o método de Runge-Kutta de quarta ordem, de modo a calcular o estado futuro.

O controlador e todos os métodos utilizados estão descritos no código PID.py.

5 MAIN

Neste código é definido o loop principal do game, ou seja, um while que atualiza a tela até que a mesma seja fechada pelo usuário. Além disso, nesta seção está definida também a GUI (graphical user interface) utilizada no simulador.

A GUI é formada por uma tela inicial (Figura 10), a tela de status do drone (Canto superior esquerdo da figura 11) e uma tela de créditos, que mostra as informações dos criadores do jogo (Figura 12).

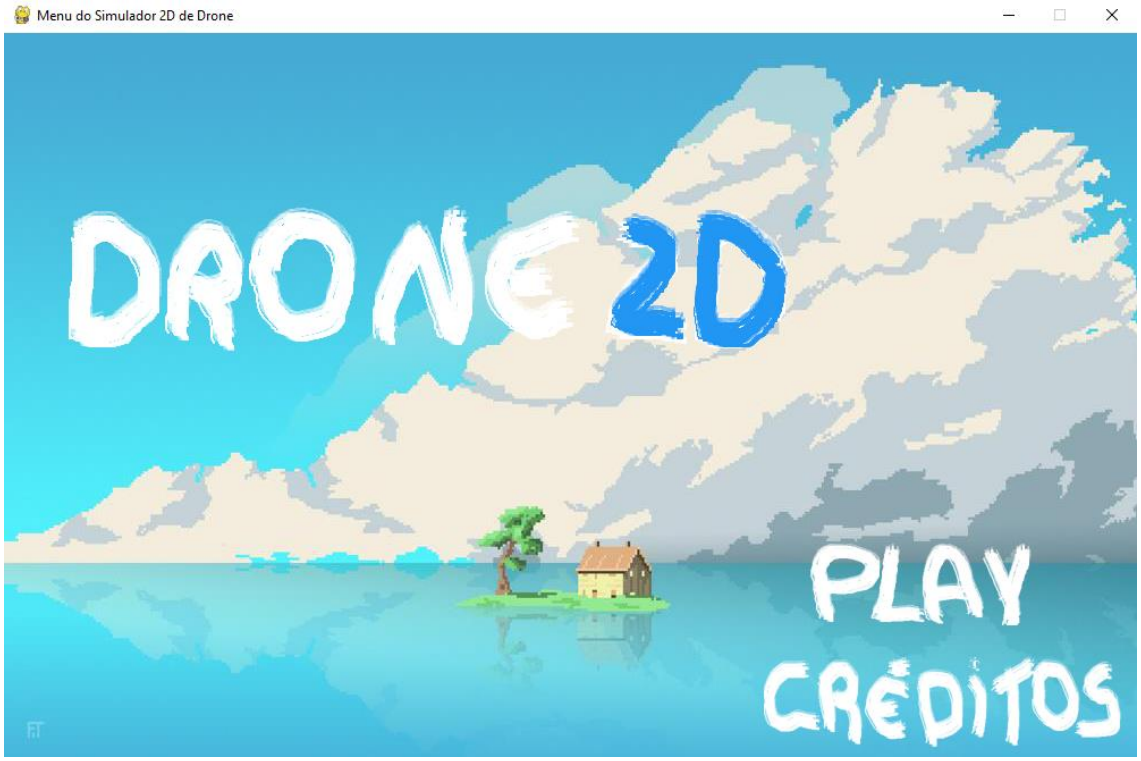

Figura 10 - GUI do simulador

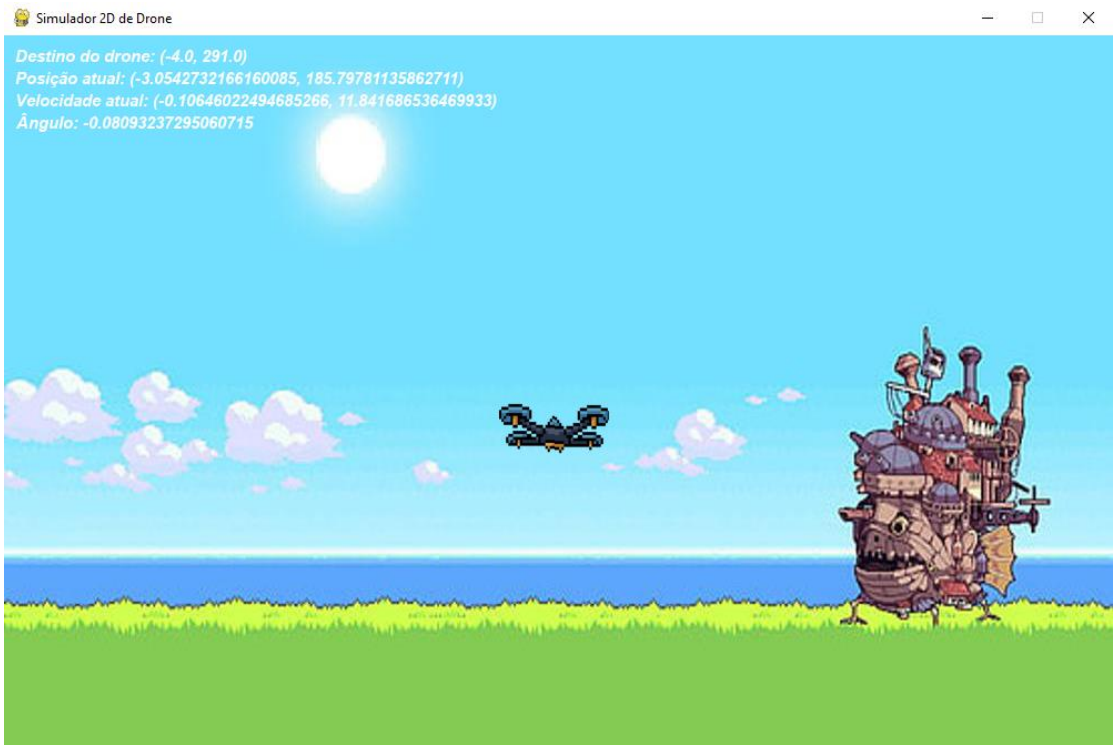

Figura 11 – Simulador em funcionamento

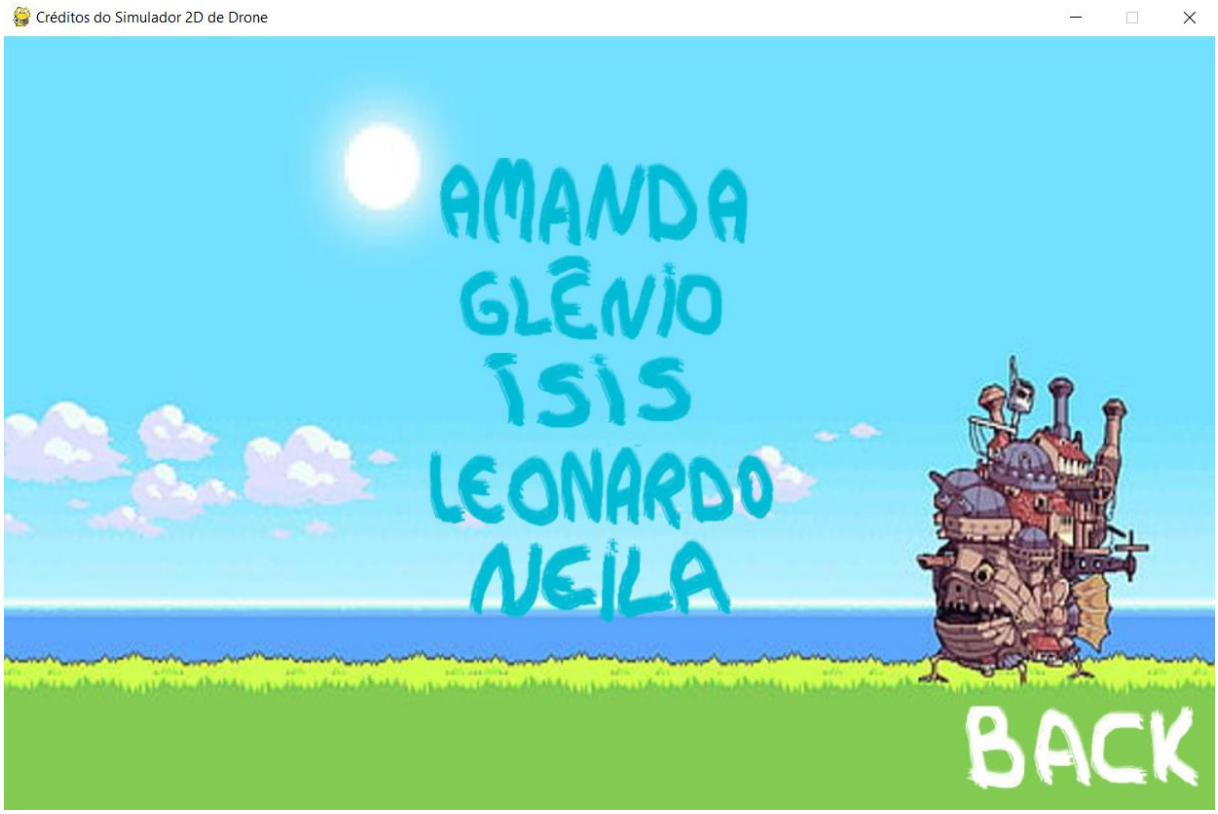

Figura 12 - Página de créditos

BIBLIOGRAFIA

[1] PYGAME ORG. Pygame.org, 2021. Disponível em: <http://pygame.org/>. Acesso em: 03 de março de 2022.

[2] PyGame: A Primer on Game Programming in Python. Disponível em: < https://realpython.com/pygame-a-primer/>. Acesso em: 03 de março de 2022.

[3] Python Imaging Library. Disponível em: <https://pt.wikipedia.org/wiki/Python_Imaging_Library>. Acesso em: 03 de março de 2022.

ANEXO A – MODELAGEM DO DRONE

Universidade Federal de Uberlândia
Engenharia de Controle e Automação / Engenharia Mecatrônica
Sistemas Embarcados II / Sistemas Digitais para Mecatrônica
Prof. Éder Alves de Moura
Trabalho Final 01 – Drone 2D

Introdução

O uso de drones na atualidade tem crescido muito. Hoje, sua aplicação varia da simples recreação até o uso em aplicações comerciais, industriais e militares. A Figura 1 apresenta um exemplo de um quadricóptero, uma das versões de veículos multirotores mais populares.

Figura 1 – DJI Mavic 2.

Este roteiro apresenta a modelagem matemática do bicóptero, simulando o movimento de um drone em um plano 2D. Esta simplificação visa simular um sistema de controle embarcado que controlará o movimento, simplificado, de um drone. Esse movimento consiste de sua posição e atitude (orientação).

Modelagem da cinemática e dinâmica do movimento

A Figura 2 apresenta o conjunto de forças básicas que atuam sobre o drone. Considere o sistema de coordenadas $\mathcal{F}_b$ que é atrelado ao corpo e passa pelo centro de massa do veículo. No centro de massa temos a força peso $\vec{P}$ e duas forças $\vec{F}_1$ e $\vec{F}_2$, aplicadas pelos rotores, atuando a uma distância l do centro de massa.

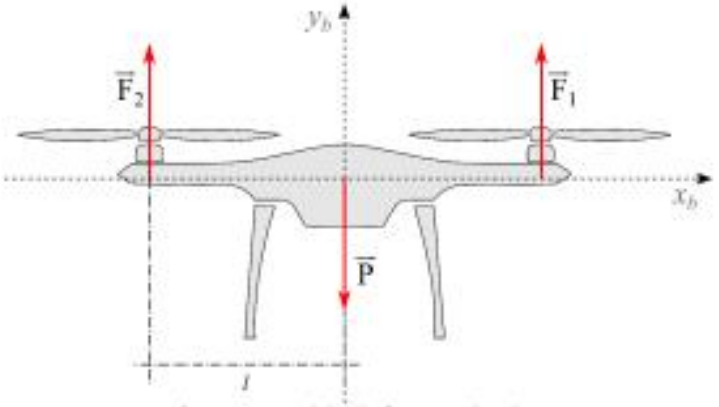

Figura 2 – Modelo de forças aplicadas.

Esse modelo apresenta simplificações para facilitar a implementação, mas permite entender o processo de simulação e controle, que podem ser utilizados em um sistema embarcado. A Figura 3 apresenta uma versão mais detalhada das forças e torque resultante, que serão utilizadas para a modelagem da dinâmica e cinemática de movimento.

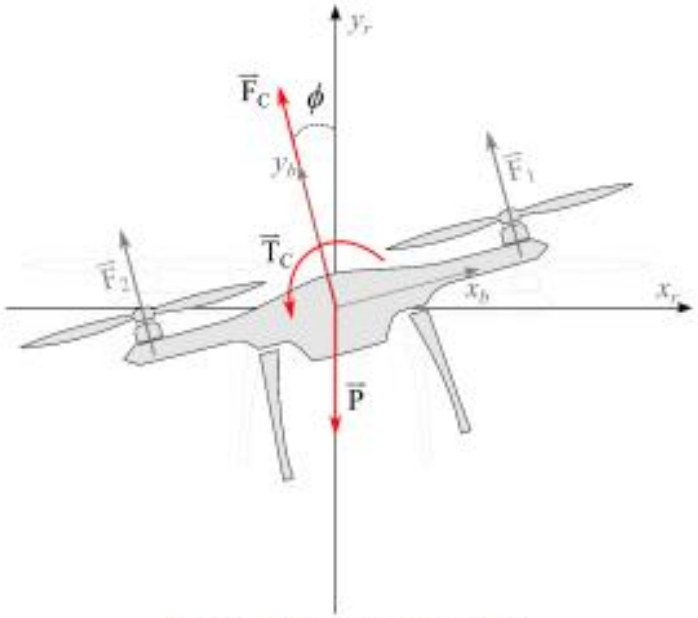

Figura 3 – Sistemas de coordenadas.

A cinemática e dinâmica de movimento do drone 2D são modeladas pelas equações:

$$\dot{w} = \frac{1}{\tau}(-w + \bar{w}) \quad (1)$$
$$\dot{r} = v \quad (2)$$
$$\dot{v} = \frac{1}{m}\left(D^{R/B}(\phi)F_C + P\right) \quad (3)$$
$$\dot{\phi} = \omega \quad (4)$$
$$\dot{\omega} = \frac{1}{I_z}T_C \quad (5)$$

onde $x = [w^T \quad r^T \quad v^T \quad \phi \quad \omega]^T \in \mathbb{R}^8$ sendo que:

- $w = [w_1 \quad w_2]^T \in \mathbb{R}^2$ é a velocidade de rotação dos rotores;
- $r = [x_r \quad y_r]^T \in \mathbb{R}^2$ é a posição;
- $v = [v_x \quad v_y]^T \in \mathbb{R}^2$ é a velocidade linear;
- $\phi \in \mathbb{R}$ é a atitude;
- $\omega \in \mathbb{R}$ é a velocidade angular;

- $F_C = [0 \quad F_1 + F_2]^T \in \mathbb{R}^2$ é a força de controle;
- $T_C \in \mathbb{R}$ é o torque de controle; e
- $D^{R/B}(\phi) \in \mathbb{R}^{2\times 2}$ é matriz de rotação.

Para a compreensão completa, também precisamos determinar as seguintes relações:

$$F_i = k_f \cdot \omega_i^2, i = 1 \text{ e } 2. \tag{6}$$

$$F_C = \begin{bmatrix} 0 \\ F_1 + F_2 \end{bmatrix} \tag{7}$$

$$T_C = l \cdot (F_1 - F_2) \tag{8}$$

$$D^{R/B}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \tag{9}$$

$$P = m \cdot g \tag{10}$$

e as constantes: l – distância de aplicação da força; k_f – constante de proporcionalidade de força; e τ – constante de tempo de resposta do rotor; m – massa; I_z – momento de inércia; e g – constante de aceleração gravitacional; são determinadas empiricamente.

Sistema de Controle

A Figura 4 ilustra o efeito de força que aparecerá ao se determinar uma velocidade de rotação para cada um dos rotores. Controlando as forças individuais de cada um dos rotores é possível controlar a força de controle resultante F_C e o torque de controle resultante T_C, como apresentado na Figura 3.

Figura 4 – Conversão entre a velocidade de rotação e as forças aplicadas por cada um dos rotores.

Com F_C é possível controlar o movimento linear e com T_C é possível controlar o movimento angular do drone. Nessa solução, adotaremos a estratégia apresentada na Figura 5 para implementação do sistema de controle para o nosso drone.

Universidade Federal de Uberlândia
Engenharia de Controle e Automação / Engenharia Mecatrônica
Sistemas Embarcados II / Sistemas Digitais para Mecatrônica
Prof. Éder Alves de Moura
Trabalho Final 01 – Drone 2D

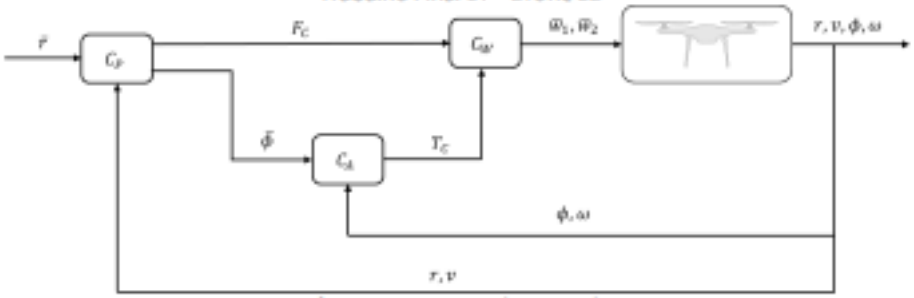

Figura 5 – Sistema de controle.

Parâmetros do modelo

Para o sistema em questão, serão adotados os seguintes parâmetros do modelo a ser simulado:

- $m = 0{,}250\ [kg]$ – Massa total do drone;
- $I_x = 2 \cdot 10^{-4}\ [kg \cdot m^2]$ – Momento de inércia de rotação
- $g = 9{,}81\ [m/s^2]$ – Constante de aceleração da gravidade;
- $l = 0{,}1\ [m]$ – Distância entre o centro de massa e o ponto de atuação da força dos motores;
- $w_{max} = 15000\ [rpm]$ – Velocidade máxima de rotação dos rotores;
- $k_f = 1{,}744 \cdot 10^{-8}$ – Constante de força;
- $\tau = 0{,}005$ – Constante de tempo.