**UNIVERSIDADE FEDERAL DE UBERLÂNDIA**

**FACULDADE DE ENGENHARIA MECÂNICA**



**Curso de Graduação em Engenharia Mecatrônica**

**Projeto Final I de Sistemas Digitais para Mecatrônica**

**Simulação de Drone 2D**

Prof. Éder Alves de Moura

|  |  |
| --- | --- |
| Guilherme Vitor dos Santos Rodrigues | 11321EMT009 |
|  |  |
|  |  |

**Uberlândia, Março de 2022**

**Sumário**

**[1.](#_Toc98111115)****[Objetivos](#_Toc98111115)** [3](#_Toc98111115)

[**2.** **Introdução** 4](#_Toc98111116)

[**3.** **Programação e Simulação** 5](#_Toc98111117)

[**4.** **Conclusão** 19](#_Toc98111118)

[**5.** **Referências** 20](#_Toc98111119)

[**6.** **Repositório de código** 20](#_Toc98111120)

# **Objetivos**

O trabalho a seguir tem como o objetivo a implementação de uma simulação de uma movimentação realista de um drone em 2 dimensões com o uso de um sistema de controle utilizando a modelagem cinemática e dinâmica desenvolvida ao longo do semestre. A movimentação deve ser feita por meio de waypoits e teclado.

# **Introdução**

A engenharia no mundo atual necessita do controle de sistemas tanto simples quanto complexos então é de extrema importância o estudo das várias teorias e métodos de controle para que sejam embarcados em sistemas digitais afim de criar equipamentos mais eficientes e autônomos. Dessas várias formas de controle, temos o controlador do tipo Proporcional-Integral-Derivativo(PID) que são utilizados devido à sua alta aplicação na maioria dos sistemas pela simplicidade e podendo ajustar as resposta experimental do sistema. Este será o sistema de controle utilizado neste trabalho.

# **Programação e Simulação**

A simulação feita utilizou apenas as bibliotecas instaladas como Pygame por exemplo. Para que a parte física pudesse ter sido feita, as constantes como massa e aceleração da gravidade. Após a definição destas constantes foram necessárias as funções definidas do pygame para carregamento de imagens e variáveis de tela e também do drone. As bibliotecas que foram utilizadas neste programas estão abaixo:



A biblioteca PID que executa o controle se utiliza de funções definidas nela própria que fazem a integração e o calculo do erro das variáveis de estado colocados como parâmetros. A função ***rk4*** executa esta integração utilizando a função de calculo de Runge-Kutta, já a função ***x\_dot*** recalcula as novas funções e erros dados.Estas funções se apresentam abaixo:

def rk4(tk, h, xk, uk):

    k1 = x\_dot(tk , xk , uk)

    k2 = x\_dot(tk + h/2.0, xk + h\*k1/2.0, uk)

    k3 = x\_dot(tk + h/2.0, xk + h\*k2/2.0, uk)

    k4 = x\_dot(tk + h , xk + h\*k3 , uk)

    xkp1 = xk +(h/6.0)\*(k1 + 2\*k2 + 2\*k3 + k4)

    return xkp1

def x\_dot(t, x, w\_):

    # State vector

    # x = [ w r\_xy v\_xy phi omega]' \in R^8

    #print('x: ', x)

    #print('w\_: ', w\_)

    ## Parâmetros

    w\_max = 15000. # velocidade máxima do motor

    m = 0.25 # massa

    g = 9.81 # aceleração da gravidade

    l = 0.1 # tamanho

    kf = 1.744e-08 # constante de força

    Iz = 2e-4 # momento de inércia

    tal = 0.005

    Fg = np.array([[0], [-m\*g]])

    ## Estados atuais

    w = x[0:2]

    r = x[2:4]

    v = x[4:6]

    phi = x[6]

    ome = x[7]

    ## Variáveis auxiliares

    # forças

    f1 = kf \* w[0]\*\*2

    f2 = kf \* w[1]\*\*2

    # Torque

    Tc = l \* (f1 - f2)

    # Força de controle

    Fc\_B = np.array([[0], [(f1 + f2)]])

    # Matriz de atitude

    D\_RB = np.array([[ np.cos(phi), -np.sin(phi)], [ np.sin(phi), np.cos(phi)]])

    ## Derivadas

    w\_dot = (-w + w\_)/tal

    r\_dot = v

    v\_dot = (1/m)\*(D\_RB @ Fc\_B + Fg)

    v\_dot = v\_dot.reshape(2,)

    phi\_dot = np.array([ome])

    ome\_dot = np.array([Tc/Iz])

    xkp1 = np.concatenate([ w\_dot, r\_dot, v\_dot, phi\_dot, ome\_dot ])

    return xkp1

A função que executa o Controle PID se apresenta abaixo como executePID e tem por objetivo o retorno do novo estado em que o drone estará e seu erro de posição e de ângulo. Esta função se apresenta abaixo:

def executePID(x, destino, t):

    eP = [0,0]

    destinoX = destino[0]

    destinoY = destino[1]

    # Extrai os dados do vetor

    r\_k = x[2:4]

    v\_k = x[4:6]

    phi\_k = x[6]

    ome\_k = x[7]

    # Comando de posição

    v\_ = np.array([0,0])

    #####################

    # Controle de Posição

    kpP = np.array([.05])

    kdP = np.array([0.25])

    eP[0] = destinoX - r\_k[0]

    eP[1] = destinoY - r\_k[1]

    #print(r\_k[1])

    eV = v\_ - v\_k

    eP\_ = eP

    Fx = kpP \* eP[0] + kdP \* eV[0]

    Fy = kpP \* eP[1] + kdP \* eV[1] - Fe

    Fy = np.maximum(0.2\*Fc\_max, np.minimum(Fy, 0.8\*Fc\_max))

    #####################

    # Controle de Atitude

    phi\_ = np.arctan2(-Fx, Fy)

    if np.abs(phi\_) > phi\_max:

    #print(phi\_\*180/np.pi)

        signal = phi\_/np.absolute(phi\_)

        phi\_ = signal \* phi\_max

    # Limitando o ângulo

        Fx = Fy \* np.tan(phi\_)

    Fxy = np.array([Fx, Fy])

    Fc = np.linalg.norm(Fxy)

    f12 = np.array([Fc/2.0, Fc/2.0])

    # Constantes Kp e Kd

    kpA = np.array([.75])

    kdA = np.array([0.05])

    ePhi = phi\_ - phi\_k

    eOme = 0 - ome\_k

    Tc = kpA \* ePhi + kdA \* eOme

    Tc = np.maximum(-0.4\*Tc\_max, np.minimum(Tc, 0.4\*Tc\_max))

    # Delta de forças

    df12 = np.absolute(Tc)/2.0

    # Forças f1 e f2 final f12' = f12 + deltf12

    if (Tc >= 0.0):

        f12[0] = f12[0] + df12

        f12[1] = f12[1] - df12

    else:

        f12[0] = f12[0] - df12

        f12[1] = f12[1] + df12

    # Comando de rpm dos motores

    w1\_ = np.sqrt(f12[0]/(kf))

    w2\_ = np.sqrt(f12[1]/(kf))

    #

    # Limitando o comando do motor entre 0 - 15000 rpm

    w1 = np.maximum(0., np.minimum(w1\_, w\_max))

    w2 = np.maximum(0., np.minimum(w2\_, w\_max))

    # Determinação do comando de entrada

    w\_= np.array([w1, w2])

    # Simulação um passo a frente

    x = rk4(t, h, x, w\_)

    x = rk4(t, h, x, w\_)

    x = rk4(t, h, x, w\_)

    x = rk4(t, h, x, w\_)

    x = rk4(t, h, x, w\_)

    x = rk4(t, h, x, w\_)

    x = rk4(t, h, x, w\_)

    x = rk4(t, h, x, w\_)

    return x, eP, ePhi

Após a definição de bibliotecas para o controle, tem-se o código abaixo que executado, cria uma simulação de controle 2D que ao toque do mouse segue as coordenadas deste e se move para lá. A definição de funções que executam a física do drone para sua movimentação mais realista foram necessárias. O código completa se apresenta abaixo:

# Variaveis iniciais

step = 1

w1 = 0

w2 = 0

v1 = 0

v2 = 0

mx, my = 0, 0

vel\_ang = 0

pos = [390, 420]

angulo = 0

eP = np.array([1, 1])  # Erro da posição

ePhi = 2  # Erro do angulo

estavel = False

auto\_move = False

y\_direction = 0

Fx = 0

Fy = 0

F = 0

vel\_vert = 0

vel\_horiz = 0

x = np.array([w1, w2, pos[0], pos[1], v1, v2,

             angulo \* np.pi/180., vel\_ang\*np.pi/180.])

# Dados do drone

m = 0.25

g = 9.79

P = m\*g

def acelerar(t, F, estavel):

    print(estavel)

    if estavel == False:

        # print(F)

        F += 100\*t

        if F >= 155:

            F = 155

    else:

        # print(F)

        F -= 100\*t

        # print(t)

        if F <= 0:

            F = 0

    return F

    # return F

def rotacionar(img, angulo):

    comando = pygame.key.get\_pressed()

    if comando[pygame.K\_a] and pos[1] < 400 or comando[pygame.K\_LEFT]:

        angulo += 1

    if comando[pygame.K\_z] and pos[1] < 400 or comando[pygame.K\_RIGHT]:

        angulo -= 1

    # Limitar o angulo em 35°

    if angulo > 35:

        angulo = 35

    if angulo < -35:

        angulo = -35

    drone\_rodado = pygame.transform.rotate(img, angulo)

    posr = (pos[0] - drone\_rodado.get\_width()/10,

            pos[1] - drone\_rodado.get\_height()/4)

    return drone\_rodado

def auto\_estabilizar(angulo):

    if angulo > 0:

        angulo -= 1

    else:

        angulo += 1

    return angulo

def mouse1(pos, angulo, eP, ePhi, mx, my, x, t):

    if np.abs(eP[0]) > 0.2 or np.abs(eP[1]) > 0.2 or np.abs(ePhi) > 0.2:

        x, eP, ePhi = executePID(x, [mx, my], t)

        # print(x[2:4])

        #print([mx, my])

        posH = x[2]

        posV = x[3]

        ang = np.round(x[6]\*180/np.pi)

        pos = [posH, posV]

        #print(mx, my)

        return pos, ang

    return pos, ang

def mouse2(pos, angulo, eP, ePhi, mx, my, x, t):

    if np.abs(eP[0]) > 0.2 or np.abs(eP[0]) > 0.2 or np.abs(eP[0]) > 0.2:

        x, eP, ePhi = executePID(x, [mx, my], t)

        posH = x[2]

        posV = x[3]

        #print([posH, posV])

        ang = np.round(x[6]\*180/np.pi)

        pos = [posH, posV]

        #print(mx, my)

        return True

    return False

def velocidade\_resultante(pos, F, P, m, estavel, angulo, step, vel\_vert, vel\_horiz, t, y\_direction):

    Fx = Fy = -1

    # Calculo de Força

    Fx = F \* math.sin(angulo \* math.pi/180)

    # print(Fx)

    if not estavel:

        Fy = F \* math.cos(angulo \* math.pi/180)

        # print(Fy)

        step -= 0.05

        if step <= 0:

            step = 0

    else:

        step = 5

        if Fy < P:

            Fy += 50\*t

        if Fy > P:

            Fy = 50\*t

        if P - 1 < Fy < P+1:

            Fy = P

    ax = Fx/m

    ay = (Fy - P) / m

    vel\_vert = -ay \* t + step \* y\_direction \* math.cos(angulo\*math.pi/180)

    vel\_horiz = -ax \* t - step \* math.sin(angulo\*math.pi/180)

    if vel\_vert > 10:

        vel\_vert = 10

    elif vel\_vert < -10:

        vel\_vert = -10

    if vel\_vert > 10:

        vel\_horiz = 10

    elif vel\_horiz < -10:

        vel\_horiz = -10

    #print("H", vel\_horiz)

    #print("V", vel\_vert)

    pos[0] += vel\_horiz

    pos[1] += vel\_vert

    if pos[0] <= 0 or pos[0] >= larg\_bk-90:

        pos[0] = pos[0]

    if pos[1] <= 0 or pos[1] >= altura\_bk-100:

        pos[1] = pos[1]

    # print(pos)

    return pos

def waypoints(pos, F, x, dt, angulo, y\_direction, i):

    xpos = [0, 0]

    wp = [[0., 10], [15., 25], [-30, 70]]

    pi = [500, 400]

    if np.abs(my - pos[1]) > 5:

        F = acelerar(dt, F, estavel)

        y\_direction = -1

    else:

        y\_direction = 0

    # for i in range(len(wp)):

    xpos[0] = pi[0] + wp[0][0]

    xpos[1] = pi[1] + wp[0][1]

    print(xpos)

    xpos, angulo = mouse1(pos, angulo, eP, ePhi, wp[0][0], wp[0][1], x, dt)

    # xpos = velocidade\_resultante(pos, F, P, m, estavel, angulo, step, vel\_vert, vel\_horiz, dt, y\_direction)

    return xpos, angulo

# Inicia o pygame

pygame.init()

# Carrega as imagens e define medidas

img\_fundo = pygame.image.load('fundor.jpg')

img\_fundo = pygame.transform.scale(img\_fundo, (900, 600))

img\_drone = pygame.image.load('image\_drone.png')

img\_drone = pygame.transform.scale(img\_drone, (100, 100))

# d1 = Sapo()

img\_drone\_rect = img\_drone.get\_rect()

# Cria variáveis de altura e largura de imagens

larg\_bk = img\_fundo.get\_rect().width

altura\_bk = img\_fundo.get\_rect().height

larg\_dr = img\_drone.get\_rect().width

altura\_dr = img\_drone.get\_rect().height

# Cria objetos para controlador, ambiente de simulação e drone

pygame.font.init()

# Cria variáveis de movimentação

step = 5

grav = False

# Variáveis de tela

tela = pygame.display.set\_mode((larg\_bk, altura\_bk))

pygame.display.set\_caption('Teste de movimento do drone')

# Variavel de execução

running = True

# Variaveis de posição

pos = [390, 420]

dpos = [0, 0]

# Clock e FPS

clock = pygame.time.Clock()

fps = 60

dt = (1/fps)/1000

angular = [0, 0]

font\_padrao = pygame.font.get\_default\_font()

font = pygame.font.SysFont(font\_padrao, 30)

while running:

    #clock.tick(60)

    # print(t)

    clock.tick(fps)  # Define fps

    tela.fill([0, 0, 0])  # Cria tela

    tela.blit(img\_fundo, (0, 0))  # Define imagem de fundo

    grav = True

    # Evento de fechamento de tela

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            sys.exit()

# Eventos de controle de drone

    # Estado das variáveis

    x = np.array([w1, w2, pos[0], pos[1], v1, v2,

                 angulo \* np.pi/180., vel\_ang\*np.pi/180.])

    comando = pygame.key.get\_pressed()

    # Inércia do sistema

    if not comando[pygame.K\_DOWN] and not comando[pygame.K\_UP]:

        y\_direction = 0

        grav = True

    if comando[pygame.K\_LEFT]:  # Comando para esquerda

        pos[0] -= step

        if pos[0] <= 0:

            pos[0] += step

        grav = False

        auto\_move = False

    if comando[pygame.K\_RIGHT]:  # Comando para direita

        right = True

        pos[0] += step

        if pos[0] <= 0 or pos[0] >= larg\_bk-90:

            pos[0] -= step

        grav = False

        auto\_move = False

    if comando[pygame.K\_UP]:  # Comando para cima

        pos[1] -= step

        if pos[1] <= 0 or pos[1] >= altura\_bk:  # Colisão

            pos[1] += step

        grav = False

        auto\_move = False

    if comando[pygame.K\_DOWN]:  # Comando para cima

        pos[1] += step

        if pos[1] > 400:

            angulo = angulo

            grav = False

            pos[1] = 400

        auto\_move = False

    # Comando para seguir caminho dos waypoints

    if comando[pygame.K\_w]:

        pygame.time.delay(100)

        i = 0

        pos, angulo = waypoints(pos, F, x, dt, angulo, y\_direction, i)

    # Eventos de clique do mouse

    if event.type == pygame.MOUSEBUTTONDOWN:

        auto\_move = True

        mx, my = pygame.mouse.get\_pos()

    # Posicionamento do drone nos limites da tela

    if pos[1] >= my or pos[1] > 450:

        #grav = False

        angulo = angulo

        pos[1] = pos[1]

    if angulo > 360:

        angulo = 0

    # Movimento automatico definido pelo clique do mouse

    if auto\_move:

        estavel = True

        if (my - pos[1]) < 5:

            F = acelerar(dt, F, estavel)

            y\_direction = -1

        else:

            y\_direction = 0

        angulo = auto\_estabilizar(angulo)

        pos, angulo = mouse1(pos, angulo, eP, ePhi, mx, my, x, dt)

    else:

        grav = True

    xd = 0.1\*pos[0] - 40

    yd = -(pos[1]-500)/275

    # dpos = [400, 500]

    #print(xd, yd, '=' , pos[0], pos[1])

    if pos[1] < 600:

        grav = False

    if grav is True:

        pos[1] += step

    else:

        pos[1] = pos[1]

    # Funções de movimentação do drone na tela

    # Atua na estabilização da rotação do drone em movimento

    drone\_rodado = rotacionar(img\_drone, angulo)

    pos = velocidade\_resultante(pos, F, P, m, estavel, angulo, step,

                                vel\_vert, vel\_horiz, dt, y\_direction)  # Movimenta o drone

    tela.blit(drone\_rodado, pos)  # Atualiza o drone na tela

    wp = [[0., 10], [15., 25], [-30, 70]]  # waypoints

    print(grav)

    # if np.abs(yd -wp[0][1])< 0.000001:

    #     print("Perto")

    # tp = np.array([w1, w2, xd, yd, v1, v2, angulo \* np.pi/180., vel\_ang\*np.pi/180.])

    # tp , eTP, eTPhi = executePID(tp, wp[0], dt)

    # print(yd)

    # if np.abs(tp[3] - wp[0][1]) > 8.3:

    #     pos[1] -=1

    # else:

    #     pos[1] = pos[1]

    pygame.time.delay(10)

    pygame.display.update()

# **Conclusão**

No final de todo o trabalho, não foi possível o controle do equipamento fazer os cálculos sozinho para a movimentação para os waypoints. O que foi possível foi um controle por meio do mouse que ao clicar na tela fazia o drone se movimentar sozinho.

# **Referências**

[1] – Materiais de Aula com o foco em Controle Digital

# **Repositório de código**

O código desta simulação pode ser encontrado em: https://github.com/GuilhermeVitor009/SEI-GuilhermeVitorDosSantosRodrigues/tree/main/Projeto\_Final\_01