Universidade Federal de Uberlândia
Faculdade de Engenharia Mecânica
Curso de Graduação em Engenharia Mecatrônica
Disciplina de Sistemas Digitais
Professor Éder Alves Moura

Trabalho Final 1

Elaboração de uma simulação de um aeropêndulo

Enrico Sampaio Bonela	11721EMT007
Guilherme Salomão Agostini	11721EMT003
Jadson Silva Souza	11721EMT017
Leonardo França De Carvalho	11821EMT012
Rafael Lins Nobre	11811EMT002
Vitor Hugo Vasconcelos De Melo	11821EMT006

UBERLÂNDIA

2022

Sumário

1 Introdução

A simulação de um processo traz à tona todo o comportamento de um sistema e quais são suas características que influenciam no seu uso no mundo real, por isso tem se tornado cada vez mais comum no mundo da engenharia a simulação de projetos mecânicos antes de sua execução, visando economia e eficiência.

Tendo em mente importância das simulações, o objetivo desse trabalho é realizar a simulação de um controle de posição aplicado a um Aeropêndulo.

O Aeropêndulo é um elemento eletro-mecânico que consegue realizar oscilações numa haste rígida em torno de um eixo de rotação, muito similar a um pênculo simples.

A rotação acontece de acordo com o empuxo gerado pelo acionamento de um motor com hélice que é fixado na haste.

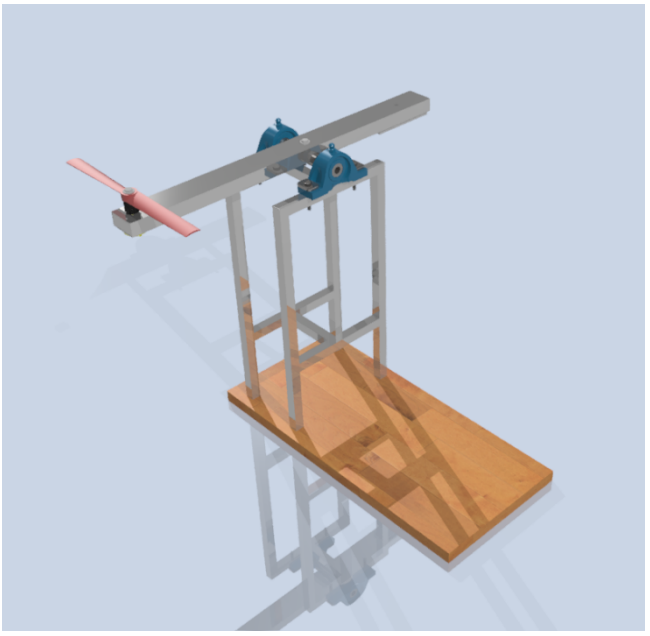

Figura 1 - Modelo 3D de aeropêndulo desenvolvido em CAD

2 Objetivos e procedimento

O trabalho visa implementar uma simulação utilizando a linguagem Python para trabalhar a dinâmica de um Aeropêndulo e do seu sistema de controle. Essa implementação será feita utilizando a biblioteca Pygame e envolve a simulação das leis da física, modelagem dinâmica e linearização e as estratégias de controle por traz do processo.

2.1 Bibliotecas Utilizadas

A principal Biblioteca utilizada para desenvolvimento do jogo foi a Pygame. Essa biblioteca se caracteriza por ser uma biblioteca de jogos multiplataforma feita para trabalhar em conjunto com a linguagem Python, baseada em SDL. Seu objetivo é ser focada em desenvolvimento de jogos e interfaces gráficas e ela consegue fornecer acesso aos hardwares disponíveis como: teclados, mouses, controles externos e controles gráficos.

Também foi utilizada a biblioteca Numpy. Essa biblioteca fornece um conjunto de funções e operações matemáticas de alto nível suportando o processamento de arranjos e matrizes multidimensionais e funções de alta complexidade. Essa biblioteca serve de suporte para trabalhar com as equações de modelagem do problema proposto.

3 Implementação

Foram implementados dois códigos para a realização da simulação, um código descrevendo o modelo e outro código para realizar a simulação propriamente dita.

3.1 Código do modelo: model.py

```
#
Importando
bibliotecas
              from scipy.integrate import odeint
              import numpy as np
              from random import uniform
              import matplotlib.pyplot as plt


              #%% Parametros do Sistema
              Lh = 0.32 #metros
              Kh = 2.12829*(10 ** -5)#N/(rad/s)²
              I = 0.0264 #kgm²
              m = 0.3182 # kg
              g = 9.81 #m/s²
              b = 0.006856 #(rad/s)^(-1)

              #%% Dinamica do aeropendulo

              # Definindo dinamica da planta
              #Esta funcao segue a documentacao da funcao odeint :
              https://www.youtube.com/watch?v=PfXJWa4TrXY (17-09-2020)
              def din_aeropendulo(y, t, v):#dinamica do aeropendulo y = theta_pp / t=
              tempo discreto /v = rotacao do motor
              thetaf,theta_pf = y #y é uma lista com [theta,theta_p]
              theta_pf_e_ppf = [ theta_pf , ((Lh*Kh/I)*(v ** 2) -
              (Lh*m*g/I)*np.sin(thetaf) - (b/I)*theta_pf) ]#aceleracao angular
              return theta_pf_e_ppf #velocidade angular e aceleracao angular

              if __name__ == '__main__':
```

```
import tqdm
#%%======= Parametros de simulacao
# Parametros de simulacao
Ta = 1e-3 # intervalo de amostragem
Tsim = 80 #tempo final
kend = int(Tsim/Ta) # quantidade de iteracoes

# scopes
#Equilibrio
theta_eq=np.zeros(kend)+2*np.pi/9 #40 graus [rad] theta_eq utilizado para
converter theta em phi
omega_eq=np.zeros(kend)+np.sqrt((m*g*np.sin(theta_eq))/Kh) #[rad/s]
omega_eq utilizado para converter omega para v
#velocidades
omega =np.zeros(kend) # velocidade de rotacao do motor real
v =omega-omega_eq # velocidade de rotacao do motor linearizada
v_p=np.zeros(kend)#acao de controle proporcinal
v_i=np.zeros(kend)#acao de controle integral
v_d=np.zeros(kend)#acao de controle derivativo
#angulos
theta = np.zeros(kend)+0*np.pi/180# Angulo do pendulo com a estrutura real
[rad]
phi = theta-theta_eq# Angulo do pendulo com o ponto de equilibrio [rad]
theta_med = np.zeros(kend)+0*np.pi/180 # Angulo do pendulo com a estrutura
phi_med=theta_med-theta_eq# Angulo do pendulo linearizado

theta_p = np.zeros(kend) # velocidade de aeropendulo real
theta_pp = np.zeros(kend) # aceleracao do aeropendulo real
e = np.zeros(kend) #Erro do sistema
e_d = np.zeros(kend) #Erro do sistema
e_i = np.zeros(kend) #Erro do sistema
phi_ref = np.zeros(kend) # Angulacao requerida ou referenciada linearizada
theta_ref=np.zeros(kend)#Angulo de referencia
#Referencias
theta_ref[:]=0*np.pi/180 #0 [rad]
phi_ref[:]=theta_ref[:]-theta_eq # [rad]
#Parametros do controlador
#Zigler Nichols critico
kcr=12.05#ganho critico
Pcr=1.2189#periodo critico

kp = 0.6*kcr
Ti=0.5*Pcr
Td=0.125*Pcr
ki=kp/Ti
kd=kp*Td

#PARAMETROS AJUSTADOS
kp = 2*kcr
Ti=0.049*Pcr
Td=1.5*Pcr
ki=kp/Ti
kd=kp*Td



#%%======================Loop percorrendo o tempo do experimento
for k in tqdm.tqdm((range(kend-1))):
```

```
if(Ta*k >= 0):#REFERENCIA
theta_ref[k] = 40*np.pi/180#angulo de referencial real [rad]
phi_ref[k] = theta_ref[k]-theta_eq[k] #angulo para o controlador[rad]
if(Ta*k >= 20):
theta_ref[k] = 30*np.pi/180#angulo de referencial real [rad]
phi_ref[k] = theta_ref[k]-theta_eq[k] #angulo para o controlador[rad]
if(Ta*k >= 40):
theta_ref[k] = (30+(k*Ta-40)*1)*np.pi/180#angulo de referencial real [rad]
phi_ref[k] = theta_ref[k]-theta_eq[k] #angulo para o controlador[rad]
if(Ta*k >= 60):
theta_ref[k] = 50*np.pi/180#angulo de referencial real [rad]
phi_ref[k] = theta_ref[k]-theta_eq[k] #angulo para o controlador[rad]
#CONTROLADOR

# Calculo da integral e da derivada do erro
e[k] = phi_ref[k] - phi_med[k]
e_d[k] = (e[k] - e[k - 1]) / Ta # derivada do erro
e_i[k] = e_i[k - 1] + e[k] * Ta # integral do erro
# Calculando as acoes de controle e o controle total
v_p[k] = e[k]*kp
v_d[k] = e_d[k]*kd

#anti-windup
if (v[k-1]==0 or v[k-1]==375):
v_i[k] =0
else:
v_i[k] = e_i[k]*ki

v[k] = v_p[k]+v_d[k]+v_i[k]#soma das acoes de controle

omega[k]=v[k]+omega_eq[k]#rotacao controlador->linear

#ATUADOR
omega[k] = min(omega[k], 375) # maximo 375 rad/s
omega[k] = max(omega[k], 0) # minimo 0

#SISTEMA NAO LINEAR
sol = odeint(din_aeropendulo, [theta[k], theta_p[k]], [ Ta*k, Ta*(k+1) ],
args= (omega[k-150],))#args recebe a velocidade real do motor (ou seja 150
ms atrasado)
theta[k+1]=sol[1,0]
phi[k+1]=theta[k+1]-theta_eq[k+1]
theta_p[k+1] = sol[1,1]
theta_med[k+1] = theta[k+1]# + uniform(-0.02, 0.02)
phi_med[k+1]=theta_med[k+1]-theta_eq[k+1]
#%%======================Plotando resultado:

plt.figure()
plt.plot(np.linspace(0, Tsim,kend) , theta*180.0/np.pi, lw =2.0 ,
color="k", label = "Posicao aeropendulo")
theta_ref[-1]=theta_ref[-2]
plt.plot(np.linspace(0, Tsim,kend) , theta_ref*180.0/np.pi, lw =2.0 ,
color="r", label = "Referencia-angulo")
plt.xlabel("Tempo [s]")
plt.ylabel("Angulo [graus]")
plt.title("Simulacao do aeropendulo")
plt.legend()
plt.grid()
plt.show()
```

```
plt.figure()
omega[-1]=omega[-2]
plt.plot(np.linspace(0, Tsim,kend), omega , lw = 2.0, color = "b", label =
"Velocidade rotacao das helices")
plt.xlabel("Tempo [s]")
plt.ylabel("Velociade de rotação [rad/s]")
plt.title("Simulacao do aeropendulo - velocidade de rotação - ação de
controle")
plt.legend()
plt.grid()
plt.show()
```

3.2 Código da simulação: simulation.py

```
#
Importando
bibliotecas
from scipy.integrate import odeint
import numpy as np
from random import uniform
import matplotlib.pyplot as plt
import pygame
import math

#%% Parametros do Sistema
Lh = 0.32 #metros
Kh = 2.12829*(10 ** -5)#N/(rad/s)²
I = 0.0264 #kgm²
m = 0.3182 # kg
g = 9.81 #m/s²
b = 0.006856 #(rad/s)^(-1)

#%% Dinamica do aeropendulo

# Definindo dinamica da planta
#Esta funcao segue a documentacao da funcao odeint :
https://www.youtube.com/watch?v=PfXJWa4TrXY (17-09-2020)
def din_aeropendulo(y, t, v):#dinamica do aeropendulo y = theta_pp / t=
tempo discreto /v = rotacao do motor
thetaf,theta_pf = y #y é uma lista com [theta,theta_p]
theta_pf_e_ppf = [ theta_pf , ((Lh*Kh/I)*(v ** 2) -
(Lh*m*g/I)*np.sin(thetaf) - (b/I)*theta_pf) ]#aceleracao angular
return theta_pf_e_ppf #velocidade angular e aceleracao angular

if __name__ == '__main__':
#VARIABLES
width, height = 800, 400 # Tamanho da tela
Out = False # Estado(byte) que controla se o jogo continua ou nao. Se
verdadeiro: jogo encerra.
acceleration = False # Estado que controla o calculo da dinamica do
pendulo
length = 0 # Comprimento L entre a bola e o rolamento
angle = 0 # Angulo
vel = 0 # Velocidade
Aacc = 0 # Aceleracao
length_draw = 300;
TSim = 0;
```

```
#COLORS
white = (255,255,255)
black = (0,0,0)
gray = (150, 150, 150)
Dark_red = (150, 0, 0)


class ball(object):

def __init__(self, XY, radius): # Set ball coordenates and radius
self.x = XY[0]
self.y = XY[1]
self.radius = radius

def draw(self, bg): # Draw circle and line based on XY coordinates
pygame.draw.lines(bg, black, False, [(width/2, 50), (self.x, self.y)], 2)
pygame.draw.circle(bg, black, (self.x, self.y), self.radius)
pygame.draw.circle(bg, Dark_red, (self.x, self.y), self.radius - 2)
pygame.display.set_caption(f'Aeropendulo Simulation Game t={int(TSim)}')


def grid(): # Draw a grid behind the pendulum
for i in range(10):
deg2rad = math.pi/180
pygame.draw.lines(background, gray, False, [(int(width / 2), 50),
(int(width / 2) + width*math.cos(10*i*deg2rad), 50 +
height*math.sin(10*i*deg2rad))])
pygame.draw.circle(background, black, (int(width/2), 50), 5)

def angle_Length(): # Send back the length and angle at the first click on
screen
length = math.sqrt(math.pow(pendulum.x - width/2, 2) + math.pow(pendulum.y
- 50, 2))
angle = math.asin((pendulum.x - width/2)/ length)
return (angle, length)

def get_path(angle, length): # with angle and length calculate x and y
position
pendulum.x = round(width/2 + length * math.sin(angle))
pendulum.y = round(50 + length * math.cos(angle))

def redraw(): # Clean up the screen and start a new grid and new frame of
pendulum with new coordinates
background.fill(white)
grid()
pendulum.draw(background)
pygame.display.update()

#BEFORE START
# carregando fonte

pygame.init()
font = pygame.font.SysFont(None, 55)
pygame.display.set_caption('Aeropendulo Simulation Game t=0')
background = pygame.display.set_mode((width, height))
clock = pygame.time.Clock()

pendulum = ball((int(width / 2),300), 15) # Comeca com o pendulo invisivel
equilibrado no meio da tela para cima
```

```
#%%======= Parametros de simulacao
# Parametros de simulacao
Ta = 1e-3 # intervalo de amostragem
kend = 2 # quantidade de iteracoes

# scopes
#Equilibrio
theta_eq=np.zeros(kend)+2*np.pi/9 #40 graus [rad] theta_eq utilizado para
converter theta em phi
omega_eq=np.zeros(kend)+np.sqrt((m*g*np.sin(theta_eq))/Kh) #[rad/s]
omega_eq utilizado para converter omega para v
#velocidades
omega =np.zeros(kend) # velocidade de rotacao do motor real
v =omega-omega_eq # velocidade de rotacao do motor linearizada
v_p=np.zeros(kend)#acao de controle proporcinal
v_i=np.zeros(kend)#acao de controle integral
v_d=np.zeros(kend)#acao de controle derivativo
#angulos
theta = np.zeros(kend)+0*np.pi/180# Angulo do pendulo com a estrutura real
[rad]
phi = theta-theta_eq# Angulo do pendulo com o ponto de equilibrio [rad]
theta_med = np.zeros(kend)+0*np.pi/180 # Angulo do pendulo com a estrutura
phi_med=theta_med-theta_eq# Angulo do pendulo linearizado

theta_p = np.zeros(kend) # velocidade de aeropendulo real
theta_pp = np.zeros(kend) # aceleracao do aeropendulo real
e = np.zeros(kend) #Erro do sistema
e_d = np.zeros(kend) #Erro do sistema
e_i = np.zeros(kend) #Erro do sistema
phi_ref = np.zeros(kend) # Angulacao requerida ou referenciada linearizada
theta_ref=np.zeros(kend)#Angulo de referencia
#Referencias
theta_ref[:]=0*np.pi/180 #0 [rad]
phi_ref[:]=theta_ref[:]-theta_eq # [rad]

#Parametros do controlador
#Zigler Nichols critico
kcr=12.05#ganho critico
Pcr=1.2189#periodo critico

kp = 0.6*kcr
Ti=0.5*Pcr
Td=0.125*Pcr
ki=kp/Ti
kd=kp*Td

#PARAMETROS AJUSTADOS
kp = 2*kcr
Ti=0.049*Pcr
Td=1.5*Pcr
ki=kp/Ti
kd=kp*Td

angle = 0
while not Out:
TSim = TSim + Ta
for event in pygame.event.get(): # Coleta de eventos do pygame
```

```
if event.type == pygame.QUIT: #Quando fecha a janela
Out = True
if event.type == pygame.MOUSEBUTTONDOWN: #Quando clica o mouse em algum
local
pendulum = ball(pygame.mouse.get_pos(), 15)
angle, length = angle_Length()



#REFERENCIA
theta_ref[0] = angle # *np.pi/180#angulo de referencial real [rad]
phi_ref[0] = theta_ref[0]-theta_eq[0] #angulo para o controlador[rad]

#CONTROLADOR

# Calculo da integral e da derivada do erro
e[0] = phi_ref[0] - phi_med[0]
e_d[0] = (e[0] - e[1]) / Ta # derivada do erro
e_i[0] = e_i[1] + e[0] * Ta # integral do erro
# Calculando as acoes de controle e o controle total
v_p[0] = e[0]*kp
v_d[0] = e_d[0]*kd

#anti-windup
if (v[1]==0 or v[1]==375):
v_i[0] =0
else:
v_i[0] = e_i[0]*ki

v[0] = v_p[0]+v_d[0]+v_i[0]#soma das acoes de controle

omega[0]=v[0]+omega_eq[0]#rotacao controlador->linear

#ATUADOR
omega[0] = min(omega[0], 375) # maximo 375 rad/s
omega[0] = max(omega[0], 0) # minimo 0

#SISTEMA NAO LINEAR
sol = odeint(din_aeropendulo, [theta[0], theta_p[0]], [ 0, Ta ], args=
(omega[0],))#args recebe a velocidade real do motor (ou seja 150 ms
atrasado)
#Movimento no tempo:
theta[1] = theta[0]
phi[1] = phi[0]
theta_p[1] = theta_p[0]
theta_med[1] = theta_med[0]
phi_med[1] = phi_med[0]

theta[0]=sol[1,0]
phi[0]=theta[0]-theta_eq[0]
theta_p[0] = sol[1,1]
theta_med[0] = theta[0]# + uniform(-0.02, 0.02)
phi_med[0]=theta_med[0]-theta_eq[0]

#Atualizando graficos
get_path(theta[0], length_draw)
redraw()
```

```
pygame.quit()
```

3.3 Respostas

Esses códigos retornam os gráficos de velocidade do motor e posição angular do aeropêndulo, bem como a simulação interativa do modelo.

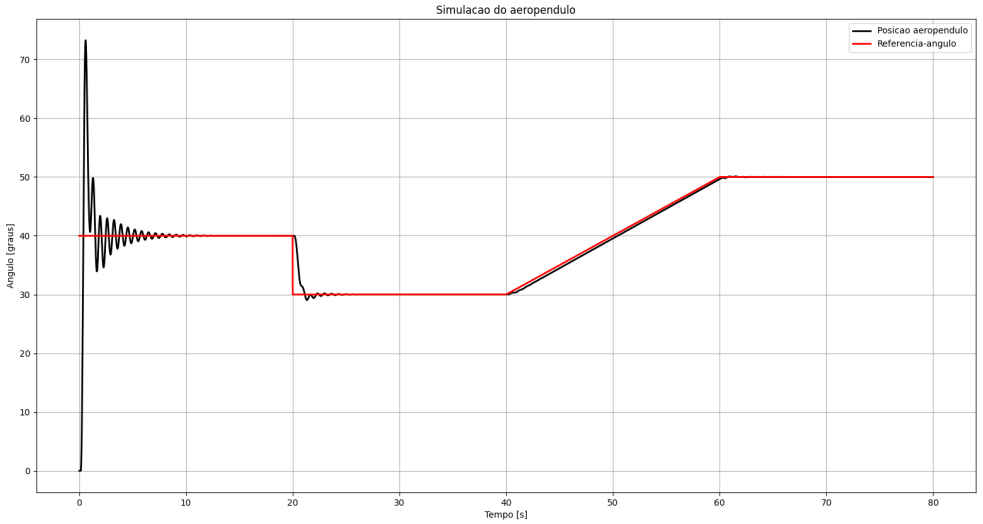

Figura 2 - posição angular do aeropêndulo

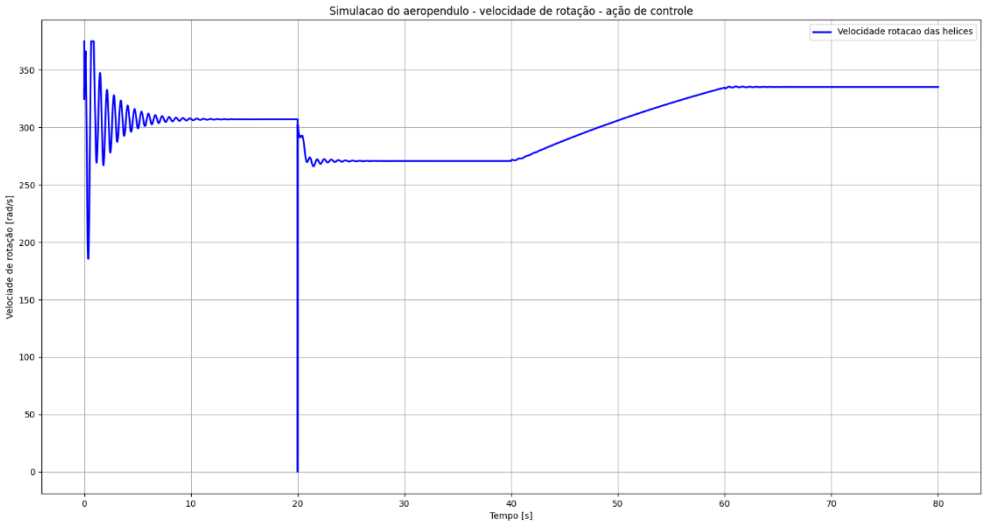

Figura 3 - velocidade do motor

4 Conclusão

Com a elaboração desse projeto ficou evidente a importância e utilidade das simulações de modelos reais no mundo da engenharia, bem como fortalecendo o conhecimento referente a linguagem python de programação.

Embora não tenha sido possível a construção do modelo real, a simulação pode ser considerada uma excelente aproximação visto que ela considera perturbações externas e saturação do motor.