UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA MECÂNICA

Curso de Graduação em Engenharia Mecatrônica

TRABALHO FINAL I DA DISCIPLINA
SISTEMAS DIGITAIS PARA MECATRÔNICA (FEELT49081)

AEROPÊNDULO

Prof. Éder Alves de Moura

Murilo Antônio da Silva Rocioli	11811EMT015
Eduardo Marques da Silva	11721EMT018
Eduardo Apolinario Lopes	11621EMT005
Caio Augusto de Medeiros	11421EMT043
Alan Nascimento de Paula	11721EMT010
Felipe de Camargo Buffeli	11621EMT002

Uberlândia, Julho de 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. Avaliar um comportamento é como dar vida a um projeto e tentar entender seu comportamento prático. Tendo como base a importância das simulações, temos como tarefa realizar a simulação de um controle de posição aplicado a um Aeropendulo.

O Aeropêndulo é um elemento eletro-mecânico que consegue realizar oscilações numa haste rígida em torno de um eixo de rotação. Essa rotação acontece de acordo com o empuxo gerado pelo acionamento de um motor com hélice que é fixado na haste.

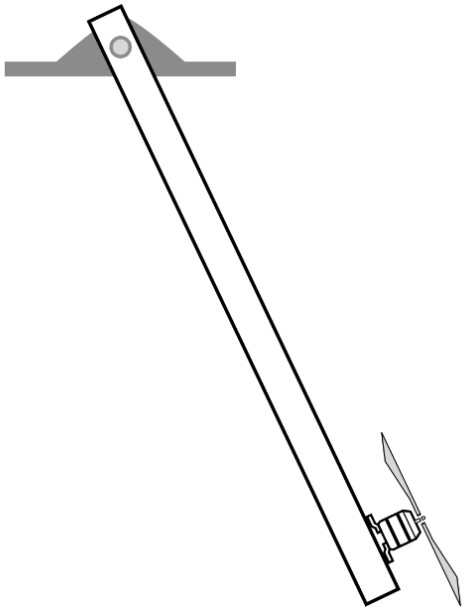

Figura 1 - Aeropêndulo

Esse sistema produz movimentos angulares e é capaz de se estabilizar em um determinado ângulo pré definido. O aeropêndulo é muito conhecido com seu vasto uso para a realização de teste de diversas técnicas avançadas de controle. Nesse caso todos os parâmetros de modelagem dinâmica e linearização foram disponibilizados no roteiro da atividade.

2. Objetivos

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.

3. 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.

São muitas as vantagens do seu uso e podem ser destacadas as facilidades que obtemos usando Pygame:

- Muitos jogos publicados com o uso da Biblioteca.
- Código relativamente pequeno e enxuto.
- Facilidade para uso em CPU's Multi Core.
- Codigo C e Assembly otimizado para funções básicas.

Figura 2 - Pygame

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.

Figura 3 - Numpy

4. Implementação

Para as simulações dinâmicas e do sistema de controle do Aeropendulo será utilizada a linguagem Python e a parte de representação visual será feita com o auxílio da biblioteca Pygame

Código utilizado para o aeropendulo:

```
import numpy as np
import matplotlib.pyplot as plt
#import pendulo

def f(t, x, u):
    # State vector
    # x = [x2 x1]^T

    x1 = x[0]
    x2 = x[1]

    # Variáveis do aeropêndulo
    l1 = .75
    l2 = 1.2
    J = 1e-2
    p = .85*9.81
    ua = .1
    x_dot = np.array( [ x2, \
                    (-p*l1/J)*np.sin(x1)   +   (-ua/J)*x2   +   (l2/J)*u   ],
dtype='float64')
    return x_dot

# Runge Kutta de 4a ordem - Metodo iterativo para melhorar precisão da
resposta
def rk4(tk, h, xk, uk):
    k1 = f(tk, xk, uk)
    k2 = f(tk + h/2.0, xk + h*k1/2.0, uk)
    k3 = f(tk + h/2.0, xk + h*k2/2.0, uk)
    k4 = f(tk + h, xk + h*k3, uk)
    xkp1 = xk + (h/6.0)*(k1 + 2*k2 + 2*k3 + k4)

    return xkp1


def saida(ang): #Definindo função de saída no pendulo
    print('FunçãoSaida')
    print(ang)
    if ang is None:
        ang = 30
    # PARÂMETROS DE SIMULAÇÃO
```

```
    h = 1e-4 # Sample time
    t = np.arange(0,5,h) # vetor tempo
    tam = len(t) # comprimento de t
    # Vetor de estados
    x = np.zeros([2, tam], dtype='float64')

    Kp = 30 # ganho proporcional
    Kd = 2e+4 # ganho derivativo
    Ki = 40 # ganho integral
    phi_ref = (ang*np.pi)/180 # ângulo de referência

    l1 =.75
    l2 = 1.2
    J = 1e-2
    p = .85*9.81
    u_eq = np.sin(phi_ref)*p*l1/l2 # ângulo de equilibrio
    # Vetor de entrada
    u = np.zeros([tam],dtype='float64') # u de equilibrio
    e = np.zeros([tam], dtype='float64') # erro de rastreamoento
    e1 = 0

    # Execução da simulação
    for k in range(tam-1):
    # u(k) será calculado aqui na simulação
    # Atualização do estado


        #CONTROLADOR PID Discreto
        e[k] = phi_ref - x[0][k]  # erro da referência

         u[k] = Kp*e[k] + Ki*(e[k] + e1) + Kd*(e[k] - e1) + u_eq # ação
de controle -> equação de diferenças

        e1 = e[k]
        x[:,k+1] = rk4(t[k], h, x[:,k], u[k]) # saída da planta


    return t, x

# Comportamento da planta
if __name__ == '__main__':
    t, x = saida()
    plt.subplot(2, 1, 1)
    plt.plot(t,x[0,:]*180/np.pi)
    plt.ylabel('$x_1$ - i ')
    plt.subplot(2, 1, 2)
    plt.plot(t,x[1,:]*180/np.pi)
    plt.ylabel('$x_2$ - q')
    plt.xlabel('t [s]')
    plt.show()
```

Assim, foi possível gerar o seguinte gráfico:

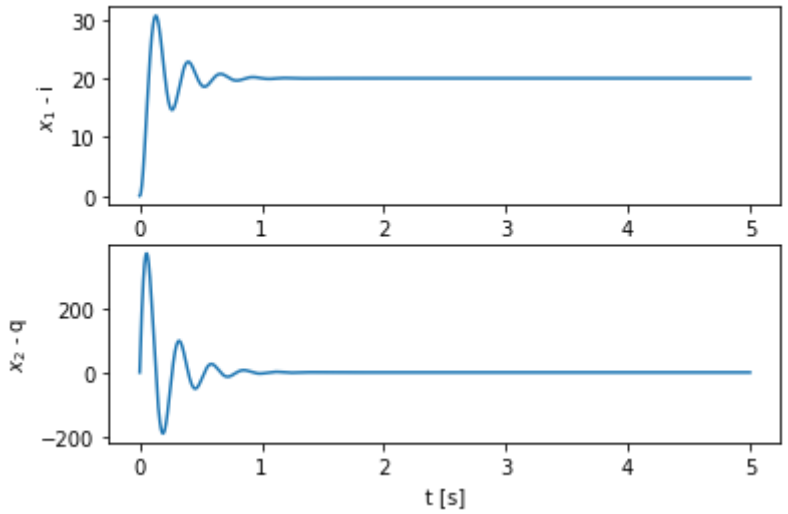

Figura 4: Gráficos gerados

Código utilizado para a simulação através da biblioteca Pygame:

```
import pygame
import math
import time
from aeropendulo import saida # import tempo e posição de
aeropendulo.py



pygame.init()

clock = pygame.time.Clock()
closed = False

width, height = (1000, 600) # largura e altura
window = pygame.display.set_mode((width, height)) # ajustando a janela
da simulação
pygame.display.set_caption('Simulação Aeropêndulo') # mudando o nome da
janela na simulação

ang = 0
user_text = ''

class massa(): # aeropêndulo
    def __init__(self):
```

```
            self.tempo, self.xx = saida(ang) # chamada de código
aeropêndulo -> tempo, posição do aeropêndulo
    def draw(self, bg, i):
        self.x = round(width/2 + 300*math.sin(self.xx[0, i]))
        self.y = round(300*math.cos(self.xx[0, i]))
        for j in range(25, width, 25): # grid
            pygame.draw.lines(bg, 'blue', False, [(j, 0), (j, height)])
        for w in range(25, height, 25): # grid
            pygame.draw.lines(bg, 'blue', False, [(0, w), (width, w)])

         pygame.draw.lines(bg, (0, 0, 0), False, [(width/2,0), (self.x,
self.y)], 4)
        pygame.draw.circle(bg, (0, 0, 0), (self.x, self.y), 25)
        pygame.draw.circle(bg, (255, 255, 255), (self.x, self.y), 23)
        pygame.draw.circle(bg, (160, 50, 50), (self.x, self.y), 10)

    def check (self,ang):
        self.tempo, self.xx =  saida(ang)

# white = (255, 255, 255)
# blue = (0, 0, 255)
# red = (255, 0, 0)
# green = (0, 255, 0)

i = 0
haste = massa()

while not closed:
    clock.tick(120)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            closed = True
        if event.type == pygame.KEYDOWN:
            print(user_text)
            # Check for backspace
            if event.key == pygame.K_BACKSPACE:

                # get text input from 0 to -1 i.e. end.
                user_text = ''

                ang = 0

            # Unicode standard is used for string
            # formation
            else:
                user_text += event.unicode
                try:
                    ang = int(user_text)
                    haste.check(ang)
                except ValueError:
                    ang = 30
```

```
    window.fill((255, 255, 255))
    haste.draw(window, i)
    i = i + 1

    pygame.display.update()

pygame.quit()
```

Abaixo, podemos ver uma captura de tela da simulação realizada com um ângulo de 40º:

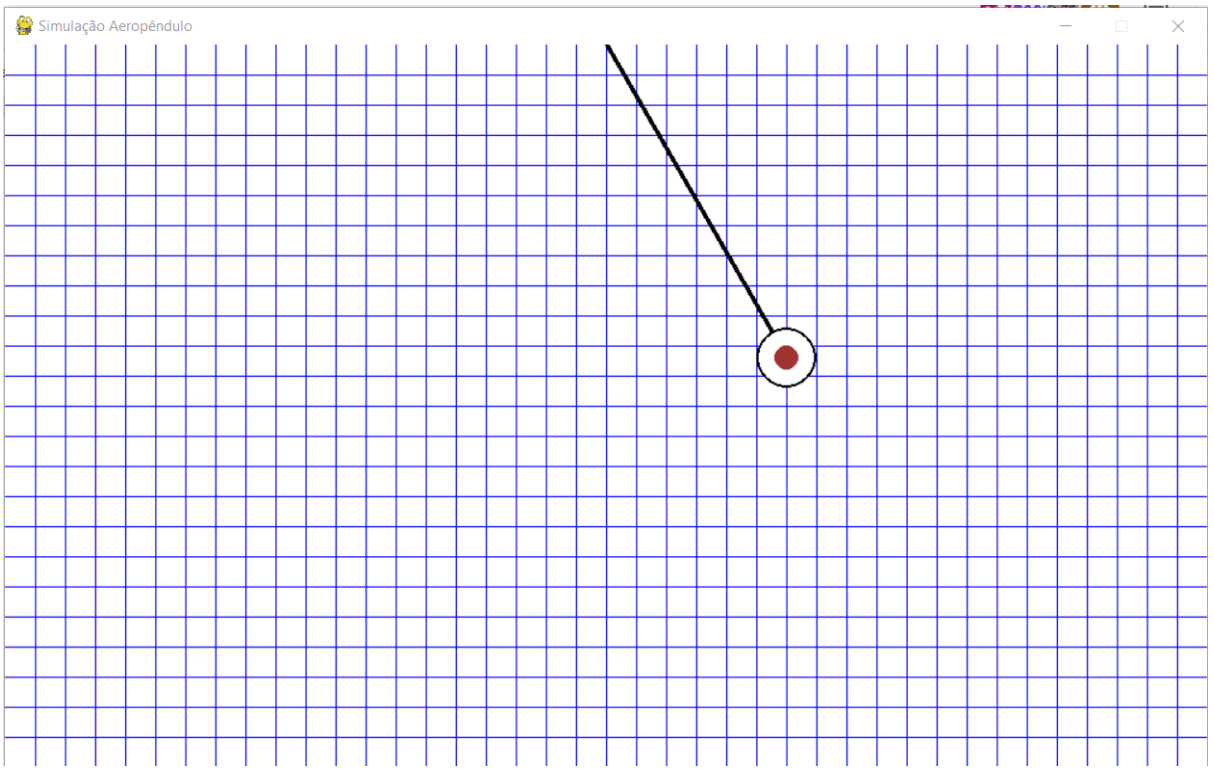

Figura 5: Simulação