Universidade Federal de Uberlândia – UFU
Faculdade de Engenharia Mecânica – FEMEC
Graduação em Engenharia Mecatrônica
Sistemas Digitais para Mecatrônica

Prof. Eder Alves Moura

RELATÓRIO SISTEMAS DIGITAIS

Jean Robert da Cunha Marquez, 11621EMT008

João Vitor Barbosa Rocha, 11711EMT007

Lucas Eduardo Marra de Lima, 11611EMT002

Paulo Vítor Arrais de Lima, 11421EMT024

Rafael Marques Borges Pacheco, 11721EMT019

Uberlândia, Julho de 2022

Sumário:

1. Introdução

Este trabalho consiste no desenvolvimento de uma simulação de um aeropêndulo utilizando o ambiente python e as modelagens desenvolvidas durante o semestre. O aeropêndulo deverá ter seu movimento de acordo com o ângulo solicitado pelo usuário. Para desenvolvimento do código foi utilizado as bibliotecas pygame, numpy e matplotlib.

2. Bibliotecas Utilizadas

Pygame:

A biblioteca pygame é uma biblioteca para Python que estende o SDL (biblioteca multimídia multiplataforma, escrita em C) com uma série de módulos para criação de jogos, com esses módulos é possível realizar a codificação. Essa biblioteca é multiplataforma que abstrai os componentes multimídia do computador como áudio e vídeo, permitindo o desenvolvimento com maior facilidade de programas que lidam com estes recursos como os jogos. Por ser uma biblioteca relativamente simples, e com códigos em C e assembly para funções básicas, é amplamente utilizada.

Figura 1: Logo Pygame

Numpy:

NumPy, que significa Numerical Python, é uma poderosa biblioteca da linguagem de programação Python, que consiste em objetos chamados de arrays (matrizes), que são multidimensionais. NumPy fornece um grande conjunto de funções e operações de biblioteca que ajudam os programadores a executar facilmente cálculos numéricos.

Matplotlib

Matplotlib é uma biblioteca para criação de gráficos e visualização de dados em geral e foi feita para a linguagem de programação Python. Podemos obter uma interface de programação orientada a objetos para incluir gráficos. O Matplotlib oferece um módulo chamado pyplot que oferece diversas ferramentas análogas às ofertadas pelo MATLAB.

3. Desenvolvimento

O sistema a ser controlado e simulado no ambiente pygame é um aeropêndulo. Sua modelagem e linearização são descritas a seguir:

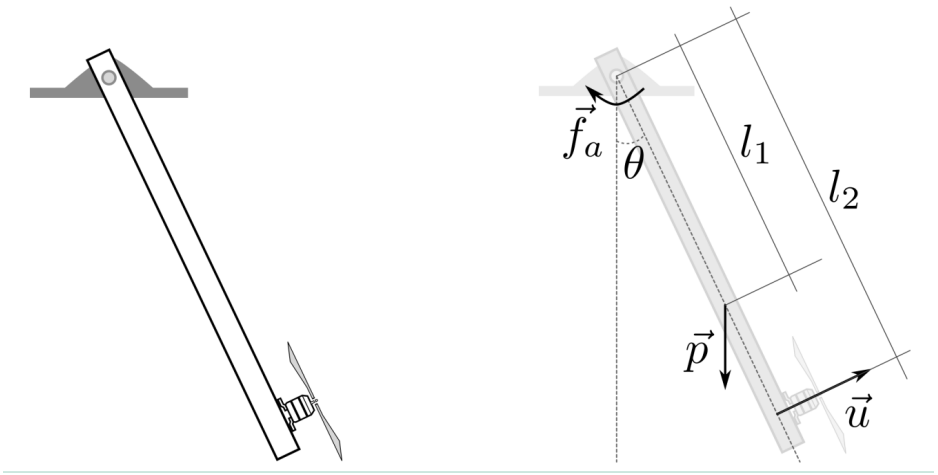

Figura 2: Aeropêndulo

A modelagem do sistema é dada por:

$$\sum T = J\ddot{\theta}$$
$$l_2 \cdot u - p \cdot l_1 \cdot \sin\theta - \mu_a \cdot \dot{\theta} = J\ddot{\theta}$$

Fazendo $x_1 = \theta$ e $x_2 = x'_1 = \theta' = \omega$:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{p \cdot l_1}{J} \cdot \sin x_1 - \frac{\mu_a}{J} \cdot x_2 + \frac{l_2}{J} \cdot u \end{bmatrix}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

assim para o sistema não linear:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{p \cdot l_1}{J} \cdot \sin x_1 - \frac{\mu_a}{J} \cdot x_2 + \frac{l_2}{J} \cdot u \end{bmatrix} = \begin{bmatrix} f_1(x,u) \\ f_2(x,u) \end{bmatrix}$$
$$= f(x,u)$$

podemos calcular as matrizes do sistema linear aproximado:

$$A = \frac{\partial f(x,u)}{\partial x} = J_x[f(x,u)] = \begin{bmatrix} 0 & 1 \\ -\frac{p \cdot l_1}{J}\cos x_1 & -\frac{\mu_a}{J} \end{bmatrix}$$

$$B = \frac{\partial f(x,u)}{\partial u} = J_u\,[f(x,u)] = \begin{bmatrix} 0 \\ \frac{l_2}{J} \end{bmatrix}$$

Assim para o sistema não linear:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = h(x, T)$$

podemos calcular as matrizes do sistema linear aproximado:

$$C = \frac{\partial h(x,T)}{\partial x} = J_x[h(x,u)] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$D = \frac{\partial h(x,T)}{\partial T} = J_T[h(x,u)] = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Por fim, o sistema linearizado fica:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{p \cdot l_1}{J} \cos x_1 & -\frac{\mu_a}{J} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{l_2}{J} \end{bmatrix} T$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} T$$

Alternativamente, a modelagem do sistema é dada por:

$$\sum T = J\ddot{\theta}$$

$$l_2 \cdot u(t) - p \cdot l_1 \cdot \sin\theta(t) - \mu_a \cdot \dot{\theta}(t) = J\ddot{\theta}(t)$$

Onde sin θ $t \approx \theta$:

$$J\ddot{\theta} + \mu_a \cdot \dot{\theta} + p \cdot l_1 \cdot \theta = l_2 \cdot u$$

aplicando a transformada de Laplace, obtemos:

$$Js^2\Theta(s) + \mu_a s\Theta(s) + p \cdot l_1\Theta(s) = l_2 \cdot U(s)$$

Logo, a função de transferência linearizada da função fica:

$$\frac{\Theta(s)}{U(s)} = \frac{l_2}{Js^2 + \mu_a s + p \cdot l_1}$$

$$\frac{\Theta(s)}{U(s)} = \frac{\frac{l_2}{J}}{s^2 + \frac{\mu_a}{J} s + \frac{mgl_1}{J}}$$

4. Lei de Controle

A partir do modelo foram realizadas simulações, com o objetivo de projetar um sistema de controle.

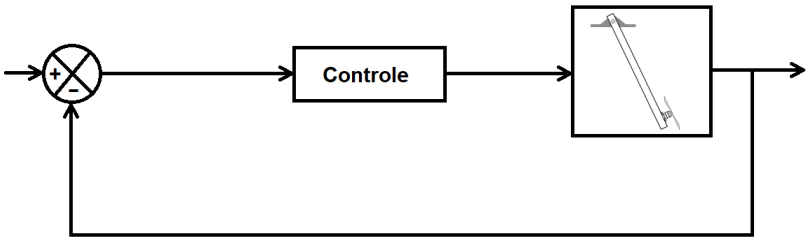

Figura 3: controle em malha fechada

Primeiramente optou-se por utilizar um controlador PID, mas devido a dificuldade encontrada de encontrar parâmetros de maneira empírica facilmente, optou-se por uma outra alternativa. Isso levou o grupo a utilizar um PIV, devido a sua facilidade de encontrar parâmetros de controle empiricamente (tuning) para o sistema. Uma planta em malha fechada com controlador PIV é descrita a seguir:

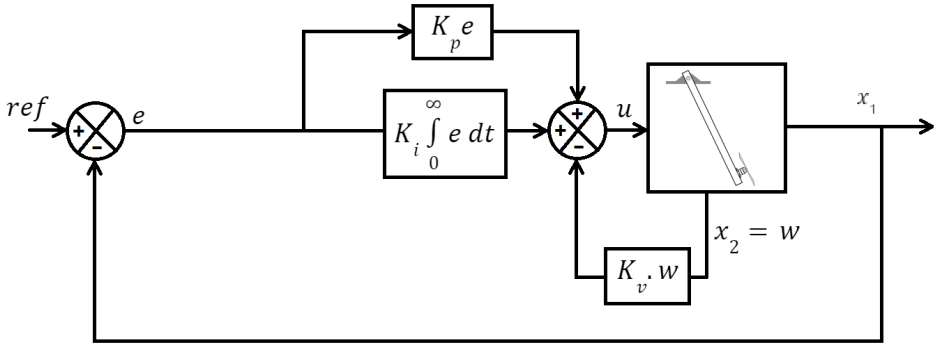

Figura 4: Aeropêndulo com PIV

Após optar pelo PIV foi implementado no código, a equação característica da lei de controle, dada por:

$$u(t) = u_p(t) + u_i(t) - u_v(t)$$

$$u_i(t) = Ki\int_0^t e(\tau)d\tau \Rightarrow L\rightarrow U_i(s) = \frac{K_i.E(s)}{s} \Rightarrow Z \rightarrow U_i(z) = \frac{K_i.E(z)T}{z-1} = \frac{K_i.E(z)Tz^{-1}.}{1-z^{-1}}$$

$$\Rightarrow U_i(z) - z^{-1}U_i(z) = K_iE(z)z^{-1}T \Rightarrow Z^{-1}\rightarrow u_i(k) - u_i(k-1) = K_ie(k-1)T$$

$$\Rightarrow u_i(k) = u_i(k-1) + e(k-1)TK_i$$

$$u_p(t) = K_pe(t) \Rightarrow L\rightarrow U_p(s) = K_pE(s) \Rightarrow Z \rightarrow U_p(z) = K_pE(z)$$

$$\Rightarrow Z^{-1}\rightarrow u_p(k) = K_pe(k)$$

$$u_v(t) = K_v w(t) \Rightarrow u_v(k) = K_v w(k) = K_v x_2(k)$$

$$u(t) = u_p(t) + u_i(t) - u_v(t) \Rightarrow u(k) = u_p(k) + u_i(k) - u_v(k)$$

Vale salientar, que na demonstração acima, o controlador digital foi obtido a partir da aproximação de retângulo de avanço em integração no tempo (forward Euler).

Implementada a ação de controle, iniciaram-se as tentativas para encontrar os parâmetros do PIV empiricamente. Para tal, utilizamos as propriedades de cada constante, e essas propriedades podem ser vistas na tabela 1 (sendo, tr: tempo de subida, MS:sobressinal, ts: tempo de estabelecimento e ess: erro de regime estacionário):

Tabela 1: Características da resposta

Aumento de	Características da resposta			
	t_r	MS	t_s	e_{ss}
K_p	Reduz	Aumenta	Pouco afeta	Reduz
K_i	Reduz	Aumenta	Aumenta	Elimina
K_v	Aumenta	Reduz	Reduz	Não afeta

E finalmente, foi possível obter excelentes resultados para as constantes, sendo elas:

$$K_p = 1; \quad K_i = 2,8; \quad K_v = 0,9$$

Para esses parâmetros, obtivemos ótimas convergências na simulação. Podemos ver a seguir o comportamento para um ângulo inicial de 0° quando aplicada uma referência de 95°:

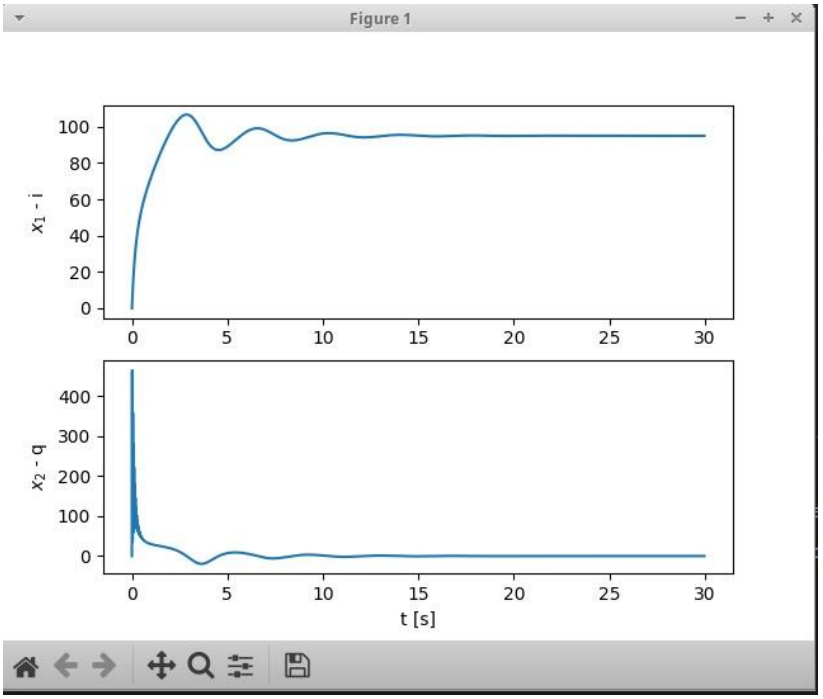

Figura 5: Ângulo inicial = 0°; referência = 95°

Também o comportamento para um ângulo inicial de 10° quando aplicada uma referência de 20°:

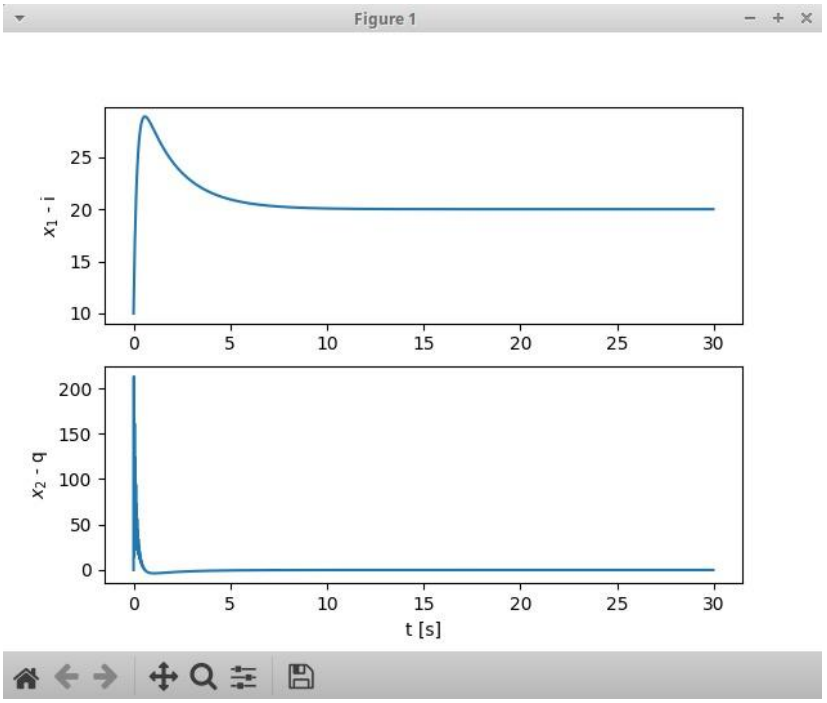

Figura 6: Ângulo inicial = 10°; referência = 20°

E por último, é possível ver o comportamento partindo de um ângulo inicial de 35° quando aplicada uma referência de 50°:

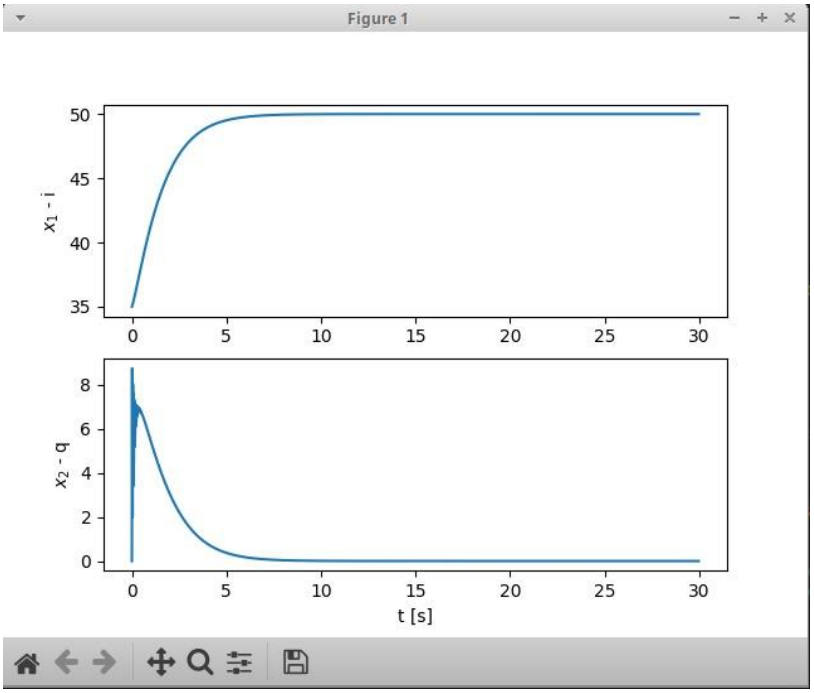

Figura 6: Ângulo inicial = 35°; referência = 50°

5. Simulação

Seguindo o objetivo do projeto, foi desenvolvido um programa simulando um aeropêndulo. Esse programa recebe a entrada do ângulo pelo usuário para movimento do aeropêndulo. Na Figura 7 podemos observar a simulação em execução com o ângulo desejado e o ângulo atual que é retornado do controlador.

Figura 7: Jogo desenvolvido

6. Integração com Pygame

Com o sistema controlado desenvolvido foi realizado a integração com o pygame. Na Figura 8 observa-se a importação das bibliotecas pygame e numpy e o sistema controlado. Em seguida foi desenvolvido uma class button com todas as configurações dos botões necessários para o jogo.

```python
import pygame
import numpy as np
import sistema_controlado

class Button:
    def __init__(self,text,width,height,pos,elevation):
        #Core attributes
        self.pressed = False
        self.elevation = elevation
        self.dynamic_elecation = elevation
        self.original_y_pos = pos[1]

        # top rectangle
        self.top_rect = pygame.Rect(pos,(width,height)) #criando rect de cima (position),(w,h)
        self.top_color = CINZA# '#475F77'
```

Figura 8: Bibliotecas importadas e class button

Foi desenvolvido uma paleta de cores e definido os frames por segundos e tempo de simulação, observado na Figura 9.

```python
#Paleta de Cores
BRANCO = (255,255,255)
PRETO = (0,0,0)
CINZA = (128,128,128)
CINZA_ESCURO = (90,90,90)
CINZA_ESCURO2 = (30,30,30)

FPS = 60 #FRAMES PER SECONDS
TEMPO_SIM = 90 #tempo padrão da simulação

AEROPENDULO_LARGURA, AEROPENDULO_ALTURA = 78,425 #Proporção padrão 1.95 por 11.26 (x40)
```

Figura 9: Paleta de cores e definição dos frames e tempo de simulação

Para desenvolver o movimento da hélice foi carregado 5 frames com posições diferentes para dar a impressão de giro representado na Figura 10.

```python
AEROPENDULO_LARGURA, AEROPENDULO_ALTURA = 78,425 #Proporção padrão 1.95 por 11.26 (x40)

#Carregando Frames do aeropêndulo que simulam a rotação da hélice
AEROPENDULO_IMAGE_0 = pygame.transform.scale(pygame.image.load('imagens/Frame1.png'),(AEROPENDULO_LARGURA, AER
AEROPENDULO_IMAGE_1 = pygame.transform.scale(pygame.image.load('imagens/Frame2.png'),(AEROPENDULO_LARGURA, AER
AEROPENDULO_IMAGE_2 = pygame.transform.scale(pygame.image.load('imagens/Frame3.png'),(AEROPENDULO_LARGURA, AER
AEROPENDULO_IMAGE_3 = pygame.transform.scale(pygame.image.load('imagens/Frame4.png'),(AEROPENDULO_LARGURA, AER
AEROPENDULO_IMAGE_4 = pygame.transform.scale(pygame.image.load('imagens/Frame5.png'),(AEROPENDULO_LARGURA, AER
INICIO_IMG = (150, 50) #distancia do furo para borda
VETOR = pygame.math.Vector2(18, 180.5) # x=dist_x_furo_propria_imagem, y= (altura_img/2) - dist_y_furo_propria
```

Figura 10: Frames

Na interface do jogo foi desenhado os botões iniciar e zerar com a função def draw_window para receber os textos com o ângulo desejado pelo usuário e o ângulo atual que é retornado pelo sistema controlado. A janela do jogo é atualizada com o pygame.display.update(), observado na Figura 11.

```python
def draw_window(imagem_aeropendulo, retangulo_aeropendulo, angulo_input_str, angulo_atual):
    WIN.fill(BRANCO)
    BotaoComecar.draw()
    BotaoZerar.draw()
    WIN.blit(imagem_aeropendulo, retangulo_aeropendulo) #expõe a imagem do aeropendulo

    entrada_texto = BOTOES_FONTE.render("Ângulo Desejado: " + angulo_input_str + "°", 1, PRETO)
    WIN.blit(entrada_texto, POS_ENTRADA)

    entrada_texto2 = BOTOES_FONTE.render("Ângulo Atual: " + angulo_atual + "°", 1, PRETO)
    WIN.blit(entrada_texto2, POS_ENTRADA2)

    pygame.display.update()
```

Figura 11: Ângulo desejado e Ângulo atual

Para realizar o movimento do aeropêndulo foi desenvolvida uma função def mover_pendulo que é constituída de 5 frames, onde o programa passa por cada ponto criando um vetor e um retângulo associado como representado na Figura 12. Após foi desenvolvido a função def main() que terá as condições iniciais do nosso jogo, observado na Figura 13.

```python
def mover_aeropendulo(angulo, imagem_indice):
    if imagem_indice == 0:
        imagem_rotacionada = pygame.transform.rotate(AEROPENDULO_IMAGE_0, angulo)
        vetor_rotacionado = VETOR.rotate(-angulo)
        retangulo = imagem_rotacionada.get_rect(center=INICIO_IMG+vetor_rotacionado)
    elif imagem_indice == 1:
        imagem_rotacionada = pygame.transform.rotate(AEROPENDULO_IMAGE_1, angulo)
        vetor_rotacionado = VETOR.rotate(-angulo)
        retangulo = imagem_rotacionada.get_rect(center=INICIO_IMG+vetor_rotacionado)
    elif imagem_indice == 2:
        imagem_rotacionada = pygame.transform.rotate(AEROPENDULO_IMAGE_2, angulo)
        vetor_rotacionado = VETOR.rotate(-angulo)
        retangulo = imagem_rotacionada.get_rect(center=INICIO_IMG+vetor_rotacionado)
    elif imagem_indice == 3:
        imagem_rotacionada = pygame.transform.rotate(AEROPENDULO_IMAGE_3, angulo)
        vetor_rotacionado = VETOR.rotate(-angulo)
```

Figura 12: Vetor e retângulo associado

```python
def main():
    clock = pygame.time.Clock()  # controla a velocidade do while loop
    run = True

    ta = 1/FPS # tempo de amostragem
    vetor_tempo = np.arange(0,TEMPO_SIM,ta) # vetor tempo
    tamanho_vetor = len(vetor_tempo) #tamanho vetor tempo
    output = 0*np.ones([tamanho_vetor],dtype='float64') #alocação vetor output
    k = 0
    angulo_input_str = '0'
    contador = 0
    angulo_inicial = 0
```

Figura 13: Condições iniciais do jogo

Após declarar as condições do jogo, foram mapeados os eventos e iniciado no

loop, foram mapeados tantos os cliques como as entradas do teclado para receber os comandos pelo usuário, como observado na Figura 14.

```
while run:
    clock.tick(FPS)

    for event in pygame.event.get(): #pega todos os eventos
        pos_mouse = pygame.mouse.get_pos()

        if event.type == pygame.QUIT: #se algum dos eventos for de saida
            run = False #sai do while com run falso
            pygame.quit()

        if event.type == pygame.KEYDOWN:
            #46 - ponto no unicode
            if (ord(event.unicode)>=48 and ord(event.unicode)<= 57) or ord(event.unicode)==46:
                if angulo_input_str == '0':
                    angulo_input_str = event.unicode
```

Figura 14: Mapeamento de eventos

Foi desenvolvido um contador que reinicia e altera os frames quando chega no valor igual a 5, observado na Figura 15.

```
if contador == 5: #reinicia contador que altera os frames
    contador = 0

try: # try/except - devido ao valor de k
    imagem_aeropendulo, retangulo_aeropendulo = mover_aeropendulo(output[k], contador)
    angulo_inicial = output[k]
    angulo_atual = round(output[k],3)
except:
    pass

#incrementa indice do output e o contador
k += 1
contador = contador + 1
```

Figura 15: Contador que altera os frames.

7. Conclusão

Neste trabalho foi encontrado diversos desafios, como entender o desenvolvimento de jogos através da plataforma do Pygame. A modelagem do nosso sistema na simulação, levando em conta a estrutura da linguagem de python, e o controle obtido mostraram excelentes resultados apesar destes desafios.

8. Referências Bibliográficas

[1] PYGAME ORG. Pygame.org, 2021. Disponível em: <http://pygame.org/>. Acesso em: 23 de julho de 2022.

[2] Python Books: Disponível em: <https://wiki.python.org/moin/PythonBooks>. Acesso em: 23 de julho de 2022