Universidade Federal de Uberlândia

FEELT – Faculdade de Engenharia Elétrica

**DESENVOLVIMENTO E SIMULAÇÃO DE UM MODELO DE AERO PENDULO**

Trabalho I da disciplina de Sistemas Embarcados II

André de Oliveira Águila Favotto – 11811EAU013

Alisson Carvalho Vasconcelos - 11511EMT016

Gabriel Medeiros Sollero – 11811EAU006

João Victor Luiz de Andrade - 11811EAU003

Rodrigo Santa Soares – 11821EAU013

Thiago Fernando Cuevas Mestanza – 11821EAU002

Orientador: Professor Éder Alves de Moura

Uberlândia

2022

Universidade Federal de Uberlândia

FEELT – Faculdade de Engenharia Elétrica

**DESENVOLVIMENTO E SIMULAÇÃO DE UM MODELO DE AERO PENDULO**

Relatório apresentado ao curso de Engenharia de Controle e Automação por meio da disciplina de Sistemas Embarcados II como requisito para a obtenção de conhecimentos na área de desenvolvimento em sistemas de controle e embarcados.

Orientador: Professor Éder Alves de Moura

Uberlândia

2022

**SUMÁRIO**

1. INTRODUÇÃO.....................................................................................................3
2. OBJETIVOS..........................................................................................................4
3. DESENVOLVIMENTO........................................................................................4
   1. MODELAGEM MATEMÁTICA........................................................................4
   2. DEFINIÇÃO DO CONTROLADOR...................................................................5
4. SIMULAÇÃO.........................................................................................................8
5. CONCLUSÃO.......................................................................................................12
6. CÓDIGOS.............................................................................................................12
7. REFERENCIAS....................................................................................................23

# **INTRODUÇÃO**

Sistemas de controle propiciam hoje, a otimização e a garantia de eficácia de processos industriais, mas além disso, de maneira geral, o controle, através de modelagens e simulações, permitem estudar, calcular e ter domínio do comportamento de sistemas de qualquer natureza. Um exemplo de sistema que usa desse recurso de controle é o aero pêndulo.

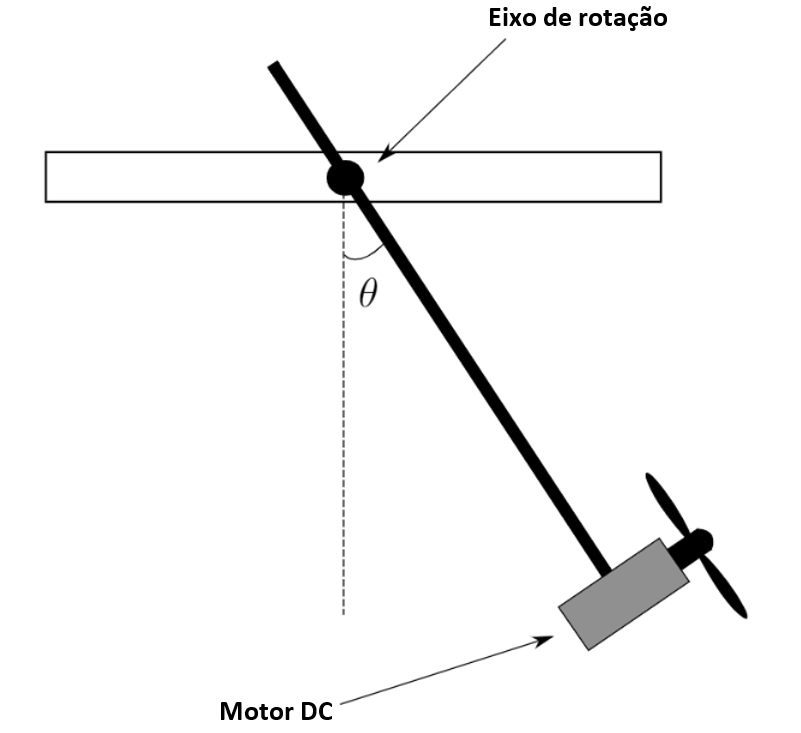
O aero pêndulo é bastante similar ao pêndulo simples, em que uma massa acoplada a um fio inextensível e fixo em uma extremidade a uma superfície. A diferença está no atuador, que no aero pêndulo é formado por um conjunto motor CC e hélice, que faz com que o eixo se movimente, produzindo uma força que atua de forma contrária à gravidade, dependendo da posição do motor. Aplicando um sistema de controle ao aero pêndulo, pode-se então determinar qual a intensidade da força produzida pelo motor para que o eixo assuma uma posição desejada. A figura 1 mostra um esquema de funcionamento de um aero pendulo:

Figura 1 - Modelo genérico de um aero pêndulo.

A construção e implementação de um sistema aero pendulo envolve diversas áreas das ciências exatas, tais como mecânica (acionamentos e estrutura do sistema), eletrônica (sensores, circuitos de controle dos motores de corrente contínua), programação (algoritmo para funcionamento em um microcontrolador) e controle de processos (ajuste dos parâmetros de controle do sistema). Nesse sentido, por se tratar de um tema multidisciplinar, o sistema de um aero pêndulo é uma oportunidade de demonstração física ou não de um controle de um sistema instável.

# **OBJETIVOS**

Este trabalho tem como objetivo o desenvolvimento de um modelo simulado de um sistema de um aero pêndulo usando a linguagem de programação Python. O programa computacional deve possuir uma interface gráfica que permita o ajuste da posição do pêndulo de acordo com a referência definida pelo usuário.

1. **DESENVOLVIMENTO**

Nesta seção, toda a construção e organização dos requisitos para a implementação do aero pendulo simulado será detalhada.

**3.1 MODELAGEM MATEMÁTICA**

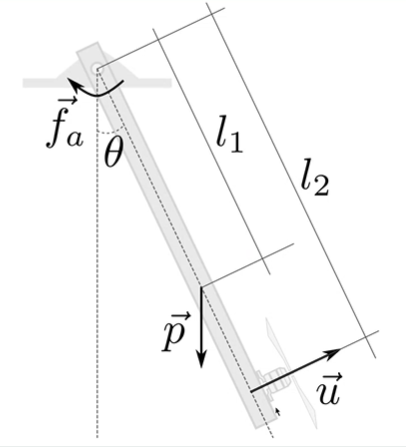
Para implementar a simulação do sistema em questão, é necessário modelar matematicamente seu comportamento, para posteriormente implementá-lo no algoritmo. Nota-se que pelo comportamento do aero pendulo caracterizar um sistema não linear, faz-se necessária a definição das condições de operação do mesmo e lineariza-lo nessa faixa de operação. Para isso, é importante conhecer o diagrama de forças envolvidas no sistema. A figura 2 representa o diagrama.

Figura 2 - Diagrama de forças do sistema

Onde:

é a força motriz de controle

é a força peso

é o comprimento total do braço

é a distância do eixo de rotação ao centro de massa do sistema.

Do diagrama de forças, aplica-se uma análise matemática no sistema. Da segunda lei de Newton, podemos escrever:

Onde representa o torque, é o momento de inércia e é a aceleração angular.

Assim, assumindo :

Aplicando a Transformada de Laplace, obtém-se:

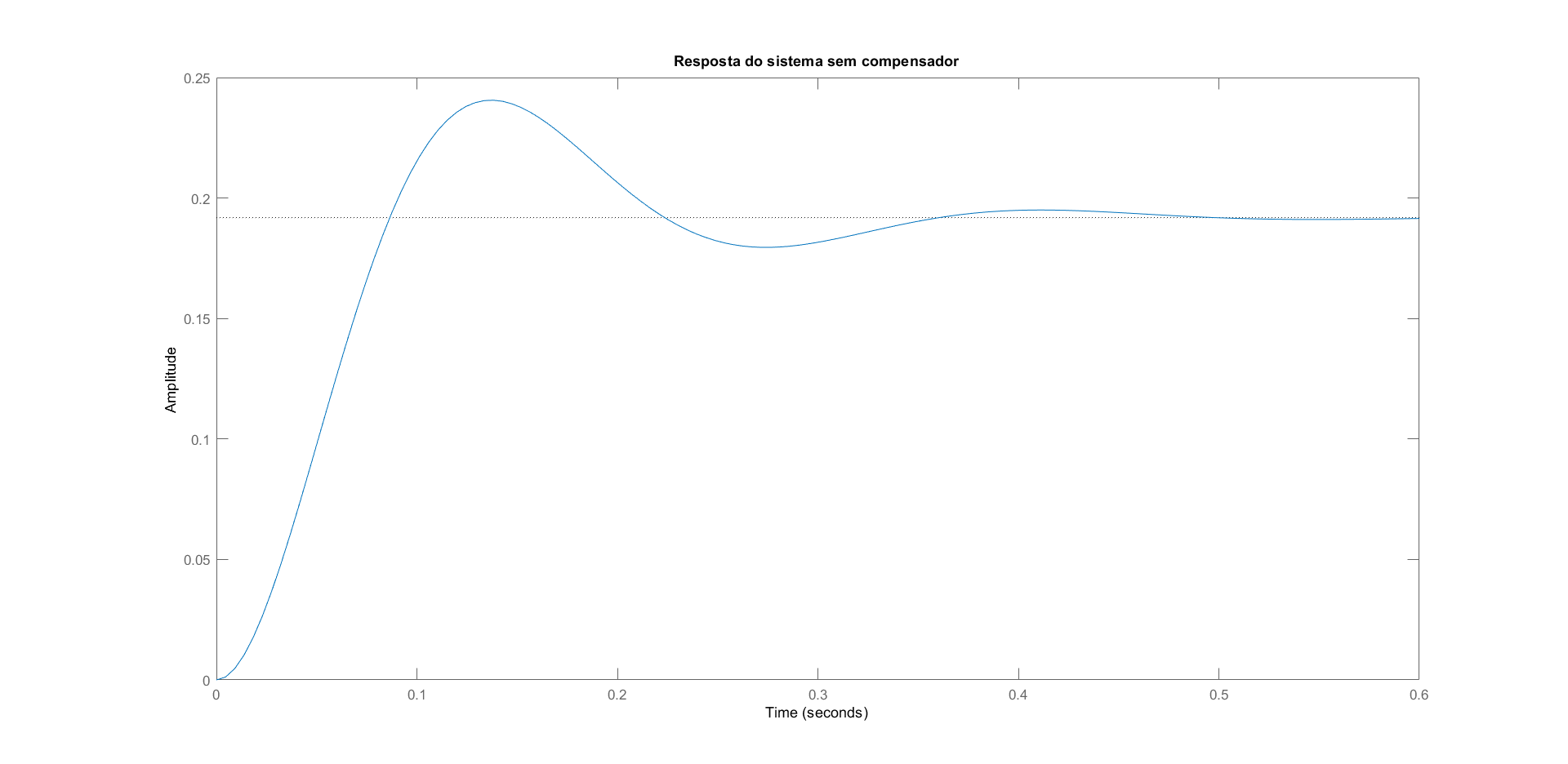
Assim, isolando a relação saída/entrada, obtém-se a função de transferência:

E, isolando o maior expoente do denominador e expandindo a força peso, temos:

Obtemos, assim, a função de transferência que representa o sistema do aero pêndulo.

**3.2 DEFINIÇÃO DO CONTROLADOR**

A partir da função de transferência obtida, torna-se possível simular seu comportamento. Nessa perspectiva, simulou-se o sistema com o auxílio da ferramenta MATLAB.

Figura 3-Resposta ao degrau unitário do sistema sem compensador.

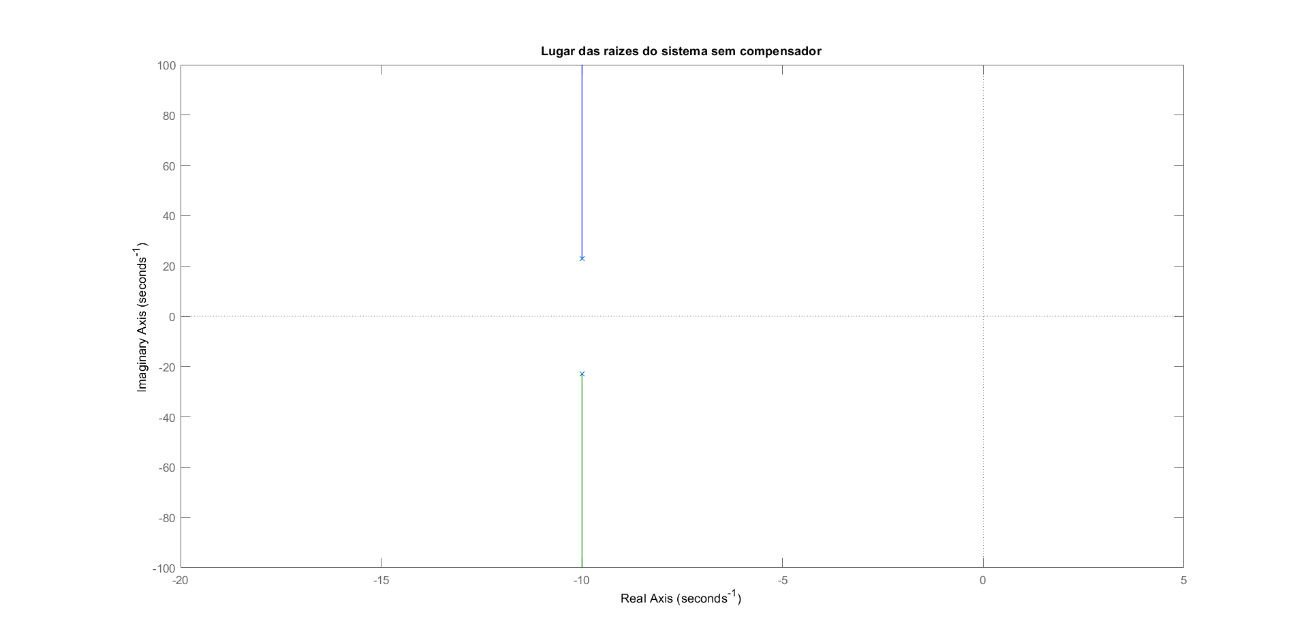
Nota-se que para uma entrada unitária, o sistema transfere aproximadamente apenas 20% do valor de referência, bem como apresenta um *overshoot* considerável de aproximadamente 25% do valor em regime permanente. A fim de compreender melhor o sistema e propor uma técnica de projeto de controlador, esboçou-se o lugar das raízes do mesmo.

Figura 4-Lugar das raízes do sistema sem compensador.

A partir da resposta do sistema e de seu lugar das raízes, conclui-se que o sistema é estável e possui polos complexos conjugados .

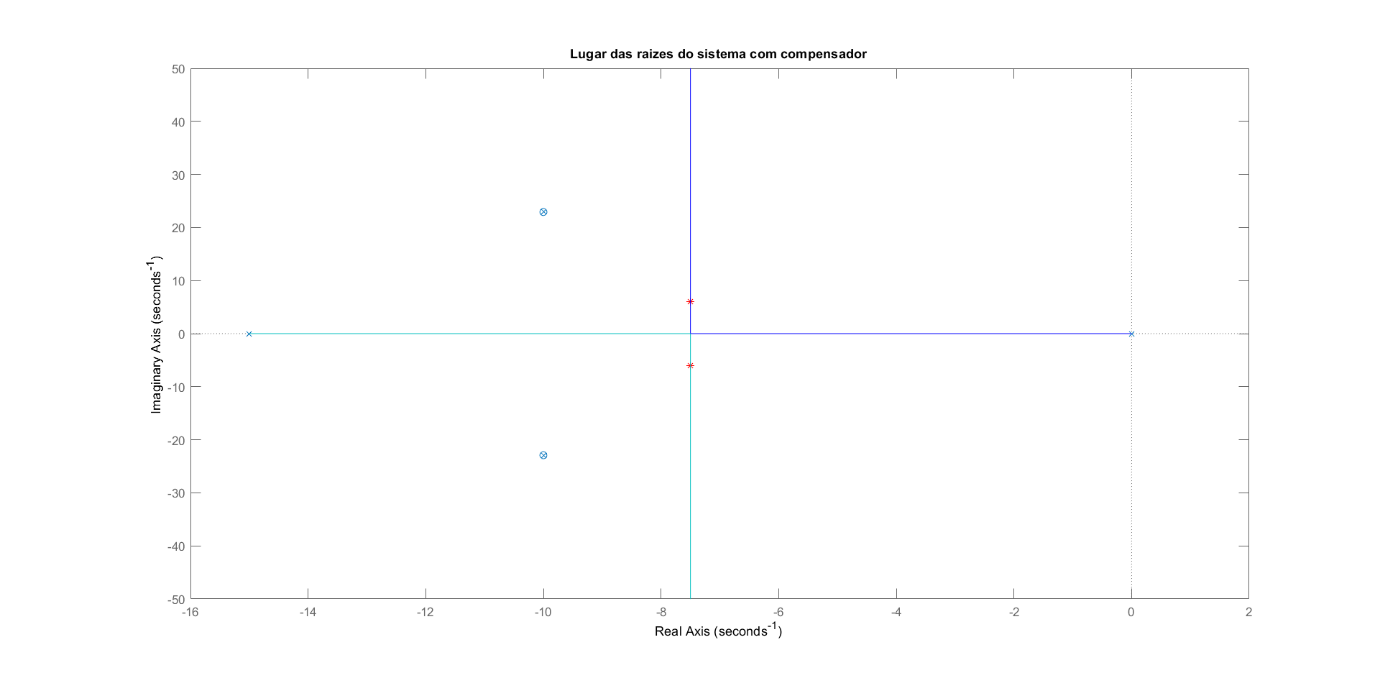
Nesse sentido, com base nas informações obtidas, será utilizada a técnica de cancelamento de polos e zeros para o projeto de um compensador que proporcione um overshoot percentual de 2% e um tempo de acomodação de cinco porcento máximo de 0,4 segundos. Buscando atingir tais requisitos, obteve-se o seguinte lugar das raízes.

Figura 5-Lugar das raízes do sistema com compensador.

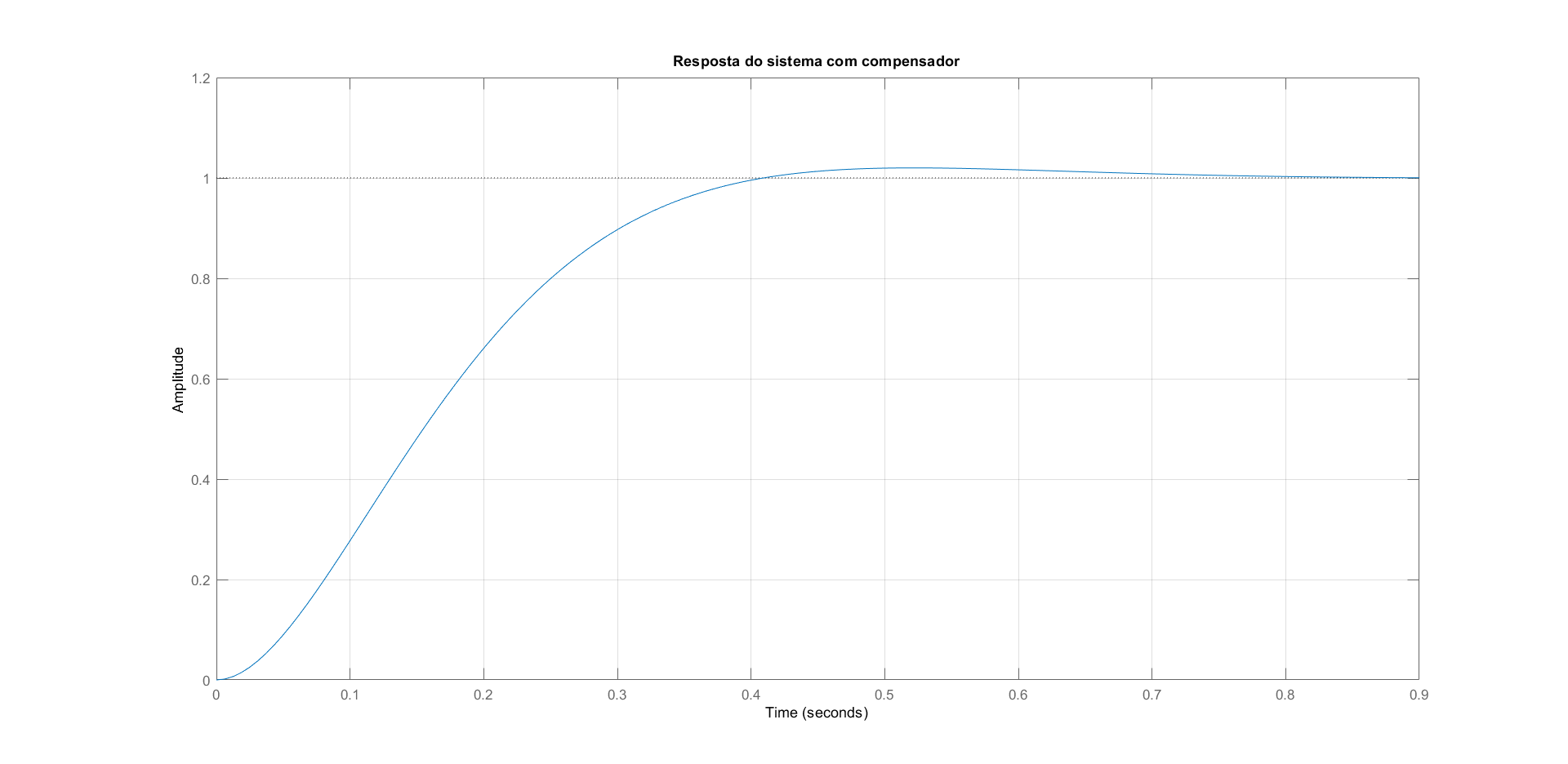
Finalmente, obteve-se a seguinte resposta para o sistema compensado:

Figura 6-Resposta ao degrau unitário do sistema com compensador.

Dessa forma, a equação do controlador obtido, no domínio s, é dada por:

Em seguida, realiza-se a discretização do compensador através do método de Tustin. O tempo de amostragem escolhido foi

A partir da equação obtida pode se obter a equação de diferenças para implementa-la em um sistema embarcado. Nesse viés, temos que:

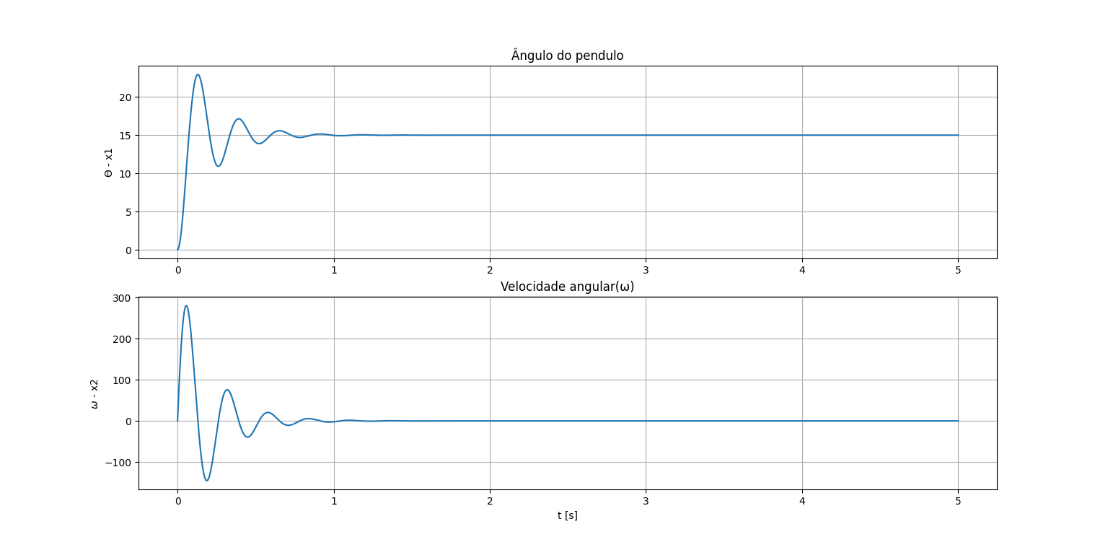
Sabe-se que a equação do erro é dada por:

Logo:

Essa equação não é causal e por consequência não pode ser implementada. Portanto, torna-se necessário realizar um deslocamento no tempo de 2 amostras. Para isso, basta multiplicar a equação por .

Aplicando a transformada Z inversa obtemos a seguinte equação no domínio do tempo:

1. **SIMULAÇÃO**

Conforme a figura 6, com auxílio da equação obtida no tópico anterior e da linguagem Python, realizou-se novamente a simulação do sistema original.

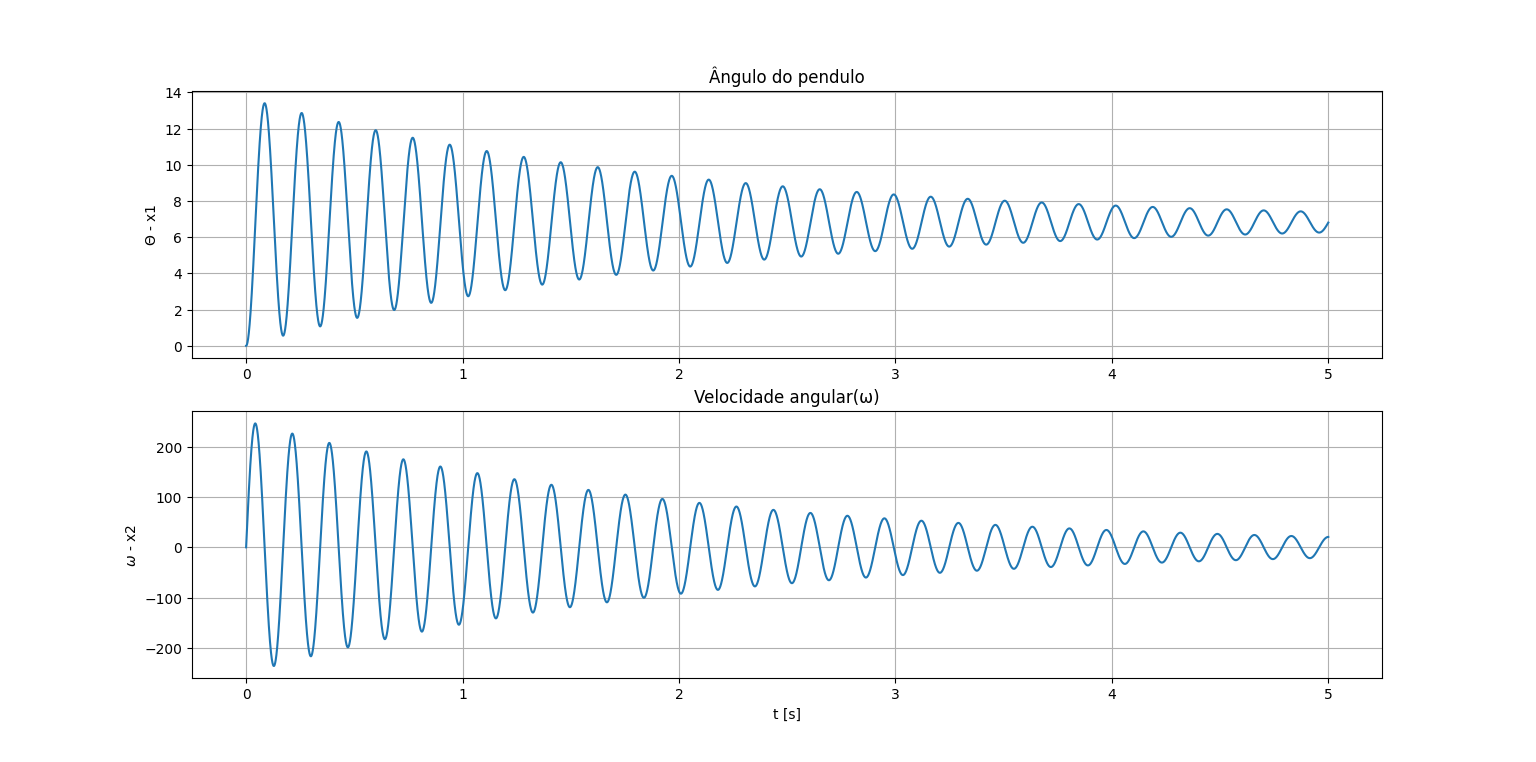
Figura 6: Resposta genérica do sistema sem compensador para condições próximas ao ponto de equilíbrio.

Figura 7: Resposta do sistema para μa = 0.01 e m=1.85Kg.

Com base na figura 6, nota-se que o sistema se comporta bem para pequenas variações no angulo de referência, desde que as condições de equilíbrio sejam mantidas. Por outro lado, a figura 7 nos mostra que para mudanças nos parâmetros do sistema a sua resposta se altera bruscamente, de modo que o coeficiente de atrito impacta diretamente no fator oscilatório do mesmo e a massa influência no angulo de equilíbrio.

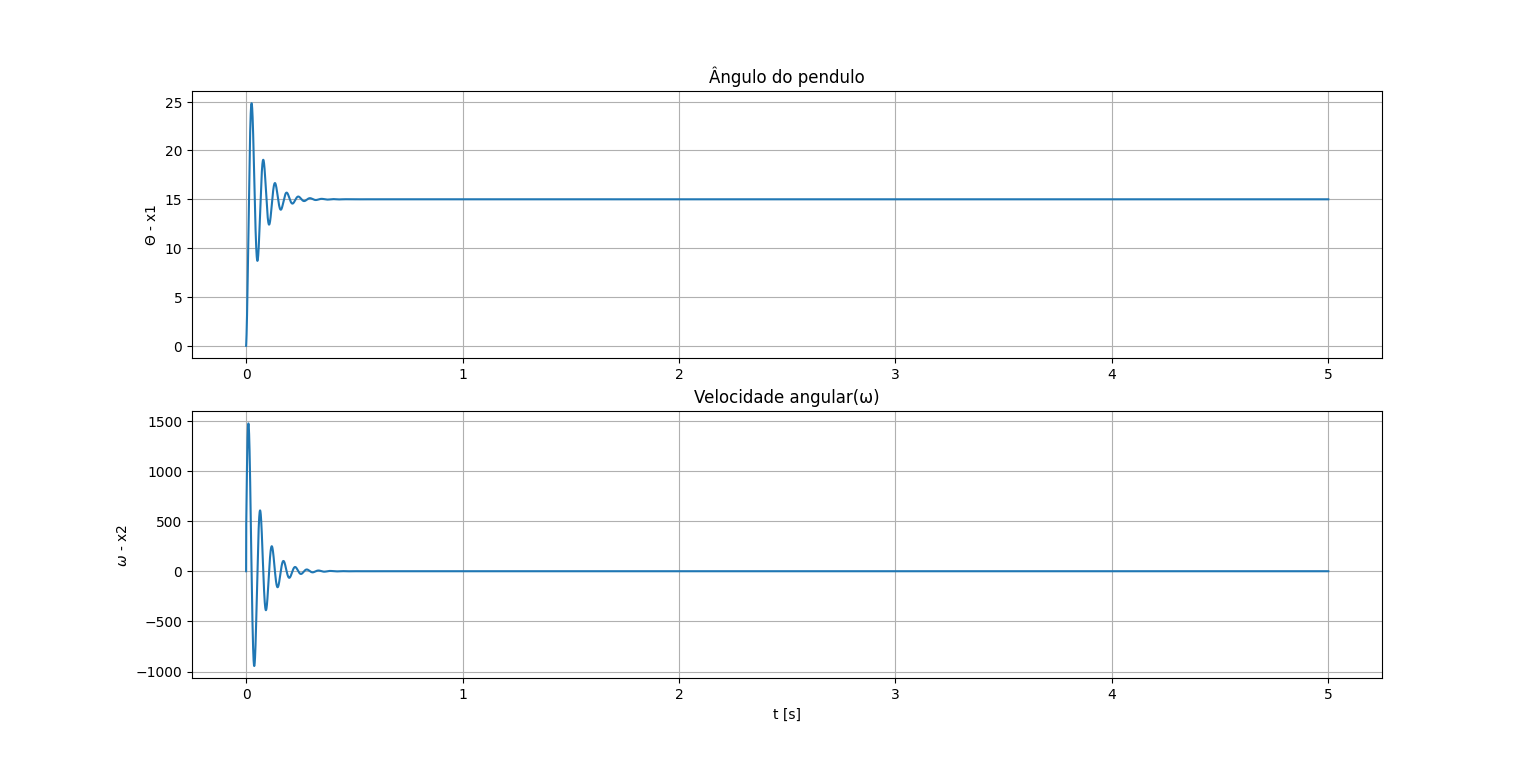
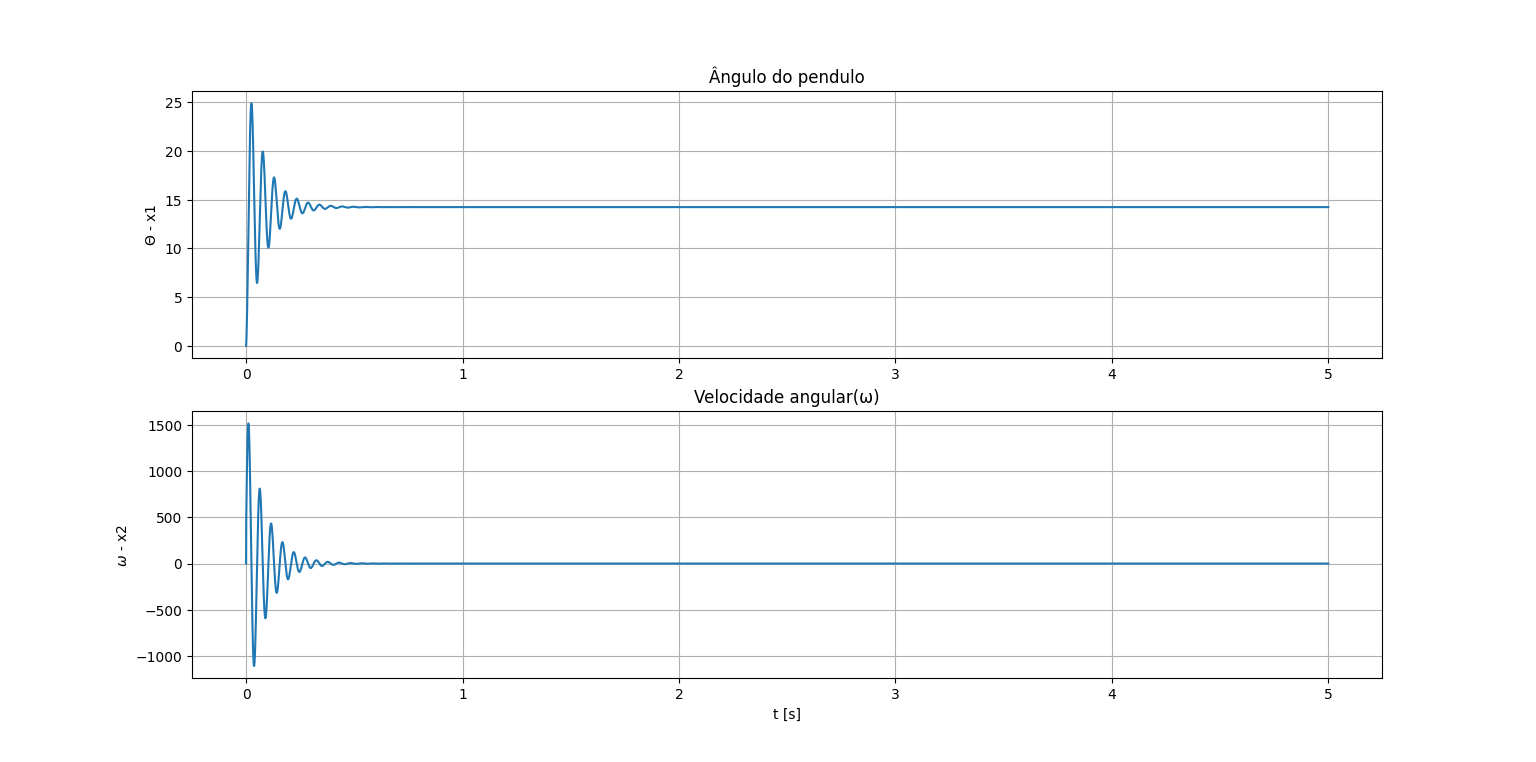
A fim de observar as mudanças no sistema com o controlador, simulou-se as mesmas condições para o sistema compensado.

Figura 8-Resposta do sistema compensado para Θref = 15° para condições próximas ao ponto de equilíbrio.

Figura 9-Resposta do sistema compensado para Θref = 15° μa = 0.01 e m=1.85Kg.

A partir da figura 8, percebe-se que o compensador atua conforme o esperado, visto que o sistema segue a referência de entrada. Ademais, conforme mostrado na figura 9, o sistema conserva a sua saída próxima dos valores desejados mesmo com mudanças nos parâmetros do sistema, porém nota-se que a medida que as condições iniciais divergem, o erro em regime permanente tende a aumentar.

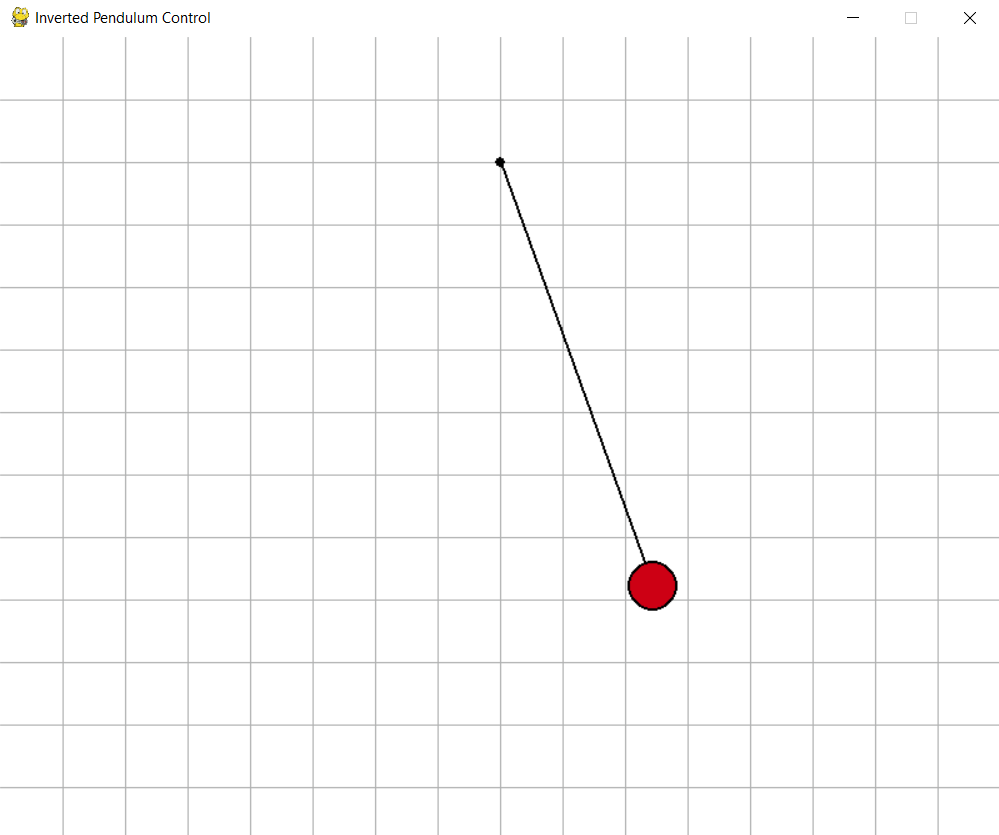
A fim de uma melhor compreensão do sistema, utilizou-se o módulo *pygame* da linguagem Python para a criação de uma representação 2D interativa. Ressalta-se que nessa representação o motor é representado por uma esfera simples.

Figura 10-Pendulo no *pygame.*

# **CONCLUSÃO**

Com a realização deste trabalho foi possível observarmos na prática a implementação da simulação de um sistema físico de forma consideravelmente fiel à realidade. Para isso, exploramos algumas funcionalidades da linguagem Python, bastante estudada ao longo do semestre, aplicando recursos de diversos módulos e bibliotecas vistos - como a *numpy, matplotlib e a pygame* - na solução de um problema prático. Ressalta-se que a biblioteca pygame possibilitou a implementação de uma interface gráfica interativa, que permite a visualização da dinâmica do sistema de forma gráfica para diversas entradas definidas pelo próprio usuário, trazendo um resultado bastante satisfatório para o sistema como um todo.

Observou-se que após a simulação desenvolvida no algoritmo, bem como as simulações feitas no MATLAB, foi possível validar a eficácia tanto do modelo matemático proposto.

# **CÓDIGOS**

* ***MATLAB***

clc;

clear;

close all;

s = tf('s');

% Equação do sistema sem compensador

l1 = .75;

l2 = 1.2;

m1 = .85;

g = 9.81;

u0 = 0.2;

J = 1e-2;

Gs = (l2/J)/(s^2 + (u0/J)\*s + m1\*g\*l1/J);

Gs = zpk(Gs)

figure(1)

step(Gs);

title("Resposta do sistema sem compensador")

figure(2)

rlocus(Gs)

title("Lugar das raizes do sistema sem compensador")

SG = stepinfo(Gs,'SettilingTimeThreshold',0.05)

%Projeto do compensador

% Requisitos de Entrada

Mp = 2; % Criterio Mp = 2%

T5 = 0.4; % Criterio T5% = 0.4s

% Zeta em função de Mp

zetamf = abs(log(Mp/100))/((pi^2)+(log(Mp/100))^2)^(1/2);

% T5%

wnmf = 3/(T5\*zetamf);

% Polos malha fechada

wdmf = (wnmf\*sqrt(1-zetamf^2));

sigmamf = zetamf\*wnmf;

smf = -sigmamf + wdmf\*1j; %Raizes de Malha Fechada

figure(3)

plot([-sigmamf -sigmamf],[wdmf -wdmf], '\*r');

hold on

%Determinando os polos do compensador

pc = [];

C1 =1;

for i=1:size(Gs.z{1,1},1)

pc = [pc Gs.z{1,1}(i)];

C1 = C1\*(s-pc(i));

end

C1=1/C1;

%Determinando os zeros do compensador

zc = [];

for i=1:size(Gs.p{1,1},1)

zc = [zc Gs.p{1,1}(i)];

C1 = C1\*(s-zc(i));

end

%Determinando Kc

ppid = -2\*sigmamf;

kc = -(smf\*(smf-ppid))/Gs.k;

% Compensador

C = (kc/(s\*(s-ppid)))\*C1

rlocus(C\*Gs);

title('Lugar das raizes do sistema com compensador')

figure(4)

sys = minreal(feedback(C\*Gs,1));

step(sys);

title('Resposta do sistema com compensador')

SI = stepinfo(sys,'SettilingTimeThreshold',0.05);

% Discretização

ts = 1e-4

Cz = c2d(C,ts,'tustin')

* ***Python***
  + *main.py*

import pygame, sys            #Importação dos comandos dentro das bibliotecas pygame e sys (Sistema)

import simulacao\_pendulo as simup                                               #Importação dos comandos dentro da biblioteca simulacao\_pendulo como simup

from settings import \*            #Importaçao de todos os dados da biblioteca settings (configurações)

from button import Button

from info import Info

from draw import Draw           #Importação da classe Draw da biblioteca draw

class Pendulo:              #Definição de classe para chamadas

  def \_\_init\_\_(self):           #Definição de uma função de chamada incial

    #Configurações Gerais

    pygame.init()                                               #Inicialização da biblioteca pygame

    self.screen = pygame.display.set\_mode((WIDTH,HEIGHT)) #Criação de uma superfície de exibição de acordo com as configurações de comprimento e tamanho

    pygame.display.set\_caption('Inverted Pendulum Control')     #Alteração do título da janela

    self.clock = pygame.time.Clock()                            #Criação de um relógio

    #São configurações básicas necessarias para rodar a biblioteca pygame

    self.bg = pygame.image.load("Image/fundo.png")    #

    self.play = Button(pygame.image.load("Image/quad1.png"), \

    (WIDTH/2, 250),"PLAY",get\_font(75),"#d7fcd4","#0BCAFE") #

    self.info = Button(pygame.image.load("Image/quad1.png"), \

    (WIDTH/2, 400),"INFO",get\_font(75),"#d7fcd4","#0BCAFE") #

    self.quit = Button(pygame.image.load("Image/quad1.png"), \

    (WIDTH/2, 550),"QUIT",get\_font(75),"#d7fcd4","#0BCAFE") #

    self.back = Button(None,(WIDTH\*0.9, HEIGHT\*0.8),"BACK", \

    get\_font(40),"black","#0BCAFE")       #

    self.Info = Info()            #

    self.draw = Draw()            #Criação da chamada da classe Level

    self.aux = True           #

  def run(self):            #Definição de uma função de execução

    while True:             #Criação de um loop infinito de verrificação

      self.screen.blit(self.bg,(0,0))       #

      m\_mouse\_pos = pygame.mouse.get\_pos()      #

      m\_text = get\_font(90).render('AEROPÊNDULO',True,'#041B04')  #

      m\_rect = m\_text.get\_rect(center = (WIDTH/2,100))    #

      self.screen.blit(m\_text,m\_rect)       #

      for button in [self.play,self.info,self.quit]:    #

        button.changeColor(m\_mouse\_pos)       #

        button.update(self.screen)        #

      for event in pygame.event.get():        #Criação de uma verificação de eventos na tela pygame

        if event.type == pygame.QUIT:       #Verificando se o evento solicitado na tela pygame é de fechamento

          pygame.quit()           #Fechamento da tela pygame

          sys.exit()            #Fechamento da execução

        if event.type == pygame.MOUSEBUTTONDOWN:

          if self.play.checkForInput(m\_mouse\_pos):    #

            i = 0

            aux1 = 0

            while self.aux:         #

              i\_mouse\_pos = pygame.mouse.get\_pos()    #

              self.screen.fill('white')       #Preenchimento da tela com a cor branca

              i = self.draw.run(i)        #Chamada da função run (executar) da classe Level

              self.back.changeColor(i\_mouse\_pos)    #

              self.back.update(self.screen)     #

              for event1 in pygame.event.get():     #

                if event1.type == pygame.QUIT:      #

                  pygame.quit()         #

                  sys.exit()          #

                if event1.type == pygame.MOUSEBUTTONDOWN: #

                  if self.back.checkForInput(i\_mouse\_pos):  #

                    self.aux = False        #

              pygame.display.update()       #Atualização da tela

              if aux1 == 0:

                aux1 = 1

                pygame.time.delay(300)

              self.clock.tick(0.1\*FPS)        #Controle da taxa de quadros

            self.aux = True         #

          if self.info.checkForInput(m\_mouse\_pos):    #

            while self.aux:         #

              i\_mouse\_pos = pygame.mouse.get\_pos()    #

              self.Info.info()          #

              self.back.changeColor(i\_mouse\_pos)    #

              self.back.update(self.screen)     #

              for event1 in pygame.event.get():     #

                if event1.type == pygame.QUIT:      #

                  pygame.quit()         #

                  sys.exit()          #

                if event1.type == pygame.MOUSEBUTTONDOWN: #

                  if self.back.checkForInput(i\_mouse\_pos):  #

                    self.aux = False        #

              pygame.display.update()       #

            self.aux = True         #

          if self.quit.checkForInput(m\_mouse\_pos):    #

            pygame.quit()         #Fechamento da tela pygame

            sys.exit()            #Fechamento da execução

      pygame.display.update()         #Atualização da tela

def get\_font(size): # Returns Press-Start-2P in the desired size

  return pygame.font.Font("Font/font.ttf", size)

if \_\_name\_\_ == '\_\_main\_\_':                    #Verificação se a biblioteca atual é a principal

  pendulo = Pendulo()                         #Criação de uma instância de classe Pendulo

  pendulo.run()                               #Chamada da função run (executar) dentro da classe Pendulo

* + *simulação\_pendulo.py*

#!/usr/bin/python3

import numpy as np

import matplotlib.pyplot as plt

from settings import \*

def f(t, x, u):

    # Vetor de Estado

    x1 = x[0]                               #

    x2 = x[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 órdem

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

    k1 = f(tk       , xk        , uk)               #Chama da função f para obtenção do valor de

    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 simulacao():                            #

    ek\_1 = 0

    # PARÂMETROS DE SIMULAÇÃO

    t = np.arange(0,5,h)                        #Declaração do vetor tempo

    tam = len(t)                            #Números de variáveis do vetor tempo criado

    # Vetor de estados

    x = np.zeros([2, tam],dtype='float64')              #Criação de uma matrix com duas colunas e números de linhas iguais ao número de variáveis do vetor tempo

    # Determinar um valor para a força de controle de equilíbrio

    u\_ref = np.sin(theta\*np.pi/180)\*p\*l1/l2             #

    #u = u\_ref\*np.ones([tam],dtype='float64')

    # Entrada do Sistema compensado

    the\_ref = theta\*np.pi/180.0

    u = the\_ref\*np.zeros([tam],dtype='float64')

    e = np.zeros([tam],dtype='float64')

    #Ganhos euler e tustin

    kp = 0.0010\*FPS

    ki = 0.0001\*FPS

    kd = 0.15\*FPS

    # Execução da simulação

    for k in range(tam-1):                      #

        e[k] = the\_ref - x[0][k]

        u[k] = kp\*e[k] + ki\*(e[k] + ek\_1) + kd\*(e[k] - ek\_1) + u\_ref

        ek\_1 = e[k]

        # Atualização do estado

        x[:,k+1] = rk4(t[k], h, x[:,k], u[k])               #

    return t,x

if \_\_name\_\_ == '\_\_main\_\_':

  t,x = simulacao()

  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()

* + *button.py*

class Button():

  def \_\_init\_\_(self, image, pos, text\_input, font, base\_color, hovering\_color):

    self.image = image

    self.x\_pos = pos[0]

    self.y\_pos = pos[1]

    self.font = font

    self.base\_color, self.hovering\_color = base\_color, hovering\_color

    self.text\_input = text\_input

    self.text = self.font.render(self.text\_input, True, self.base\_color)

    if self.image is None:

      self.image = self.text

    self.rect = self.image.get\_rect(center=(self.x\_pos, self.y\_pos))

    self.text\_rect = self.text.get\_rect(center=(self.x\_pos, self.y\_pos))

  def update(self, screen):

    if self.image is not None:

      screen.blit(self.image, self.rect)

    screen.blit(self.text, self.text\_rect)

  def checkForInput(self, position):

    if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):

      return True

    return False

  def changeColor(self, position):

    if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):

      self.text = self.font.render(self.text\_input, True, self.hovering\_color)

    else:

      self.text = self.font.render(self.text\_input, True, self.base\_color)

* + *draw.py*

#!/usr/bin/python3

import pygame, math               #Importação dos comandos dentro das bibliotecas pygame e math (matemática)

import simulacao\_pendulo as simup           #Importação dos comandos dentro da biblioteca simulacao\_pendulo como simup

from settings import \*                #Importação de todos os dados da biblioteca settings (configurações)

from button import Button

class Draw:                 #Definição de classe para chamadas

  def \_\_init\_\_(self):               #Definição de uma função da chamada inicial

    #Pega as informações da superficie

    self.display\_surface = pygame.display.get\_surface()       #Capturando a superfície da tela pygame aberta

    #Definições para desenho

    self.center = (WIDTH/2,HEIGHT/3)            #Definição do ponto onde o pendulo é fixo

    self.mat, self.x = simup.simulacao()          #Chamada das variáveis de tamanho da matrix e do valor do vetor x na biblioteca simup

  def run(self, i):             #Definição da função de execução da biblioteca level

    for j in range(25,WIDTH,25):            #Loop para desenho das grids verticais do desenho

      pygame.draw.lines(self.display\_surface,'gray',False,[(j,0),(j,HEIGHT)]) #Chamada para desenho de linhas verticais

    for k in range(25,HEIGHT,25):           #Loop para desenho da grids horizontais do desenho

      pygame.draw.lines(self.display\_surface,'gray',False,[(0,k),(WIDTH,k)])  #Chamada para desenho de linhas horizontais

    pos = (round(self.center[0] + l2\*300\*math.sin(self.x[0,i])), \

    round(self.center[1] + l2\*300\*math.cos(self.x[0,i])))     #Definição da posição da extremidade movél do pendulo

    aux = str(round(self.x[0,i]\*180/math.pi)) + '°'       #Variável auxiliar para obtenção do ângulo em graus

    self.theta = Button(pygame.transform.scale(pygame.image.load("Image/quad1.png"), \

    (300,80)),(WIDTH/2,HEIGHT/5),aux,get\_font(75),"black","black")    #Variável para desenho de uma caixa com os valores de ângulo do pendulo

    if i < len(self.mat) - 1:             #Verificação da posição i dentro do vetor x para debug

      i += 1                  #Incremento de uma unidade na variável i

    pygame.draw.circle(self.display\_surface,'black',self.center,4)    #Chamada para desenho do ponto onde o pendulo é fixo

    pygame.draw.line(self.display\_surface,'black',self.center,pos,2)    #Chamada para desenho da haste do pendulo

    pygame.draw.circle(self.display\_surface,'black',pos,20)     #Chamada para desenho do contorno do ponto móvel do pendulo

    pygame.draw.circle(self.display\_surface,'red',pos,18)     #chamada para prenchimento do interior do ponto móvel do pendulo

    self.theta.update(self.display\_surface)         #Chamada da função update na classe Button

    return i

def get\_font(size): # Returns Press-Start-2P in the desired size

  return pygame.font.Font("Font/font.ttf", size)

* + *info.py*

#!/usr/bin/python3

import pygame, sys

from settings import \*

class Info():

  def \_\_init\_\_(self):

    #Pega as informações da superficie

    self.screen = pygame.display.get\_surface()       #Criação de uma superfície de exibição de acordo com as configurações de comprimento e tamanho

  #Entrada de dados do tipo de fonte e da posição onde será colocado o texto

  def info(self):

      self.screen.fill("white")

      i\_text = get\_font(18).render("Sistemas Embarcados II - Trabalho Prático 01: Sistema de Controle", True, "Black")

      i\_rect = i\_text.get\_rect(center=(WIDTH/2, 70))

      self.screen.blit(i\_text, i\_rect)

      i\_text = get\_font(17).render("PROFESSOR: Éder Alves de Moura", True, "Black")

      i\_rect = i\_text.get\_rect(center=(WIDTH/2, 120))

      self.screen.blit(i\_text, i\_rect)

      i\_text = get\_font(17).render("GRUPO: ", True, "Black")

      i\_rect = i\_text.get\_rect(center=(WIDTH/2, 170))

      self.screen.blit(i\_text, i\_rect)

      i\_text = get\_font(17).render("Alisson Carvalho Vasconcelos     -11511EMT016", True, "Black")

      i\_rect = i\_text.get\_rect(center=(WIDTH/2, 210))

      self.screen.blit(i\_text, i\_rect)

      i\_text = get\_font(17).render("André de Oliveira Águila Favotto - 11811EAU013", True, "Black")

      i\_rect = i\_text.get\_rect(center=(WIDTH/2, 250))

      self.screen.blit(i\_text, i\_rect)

      i\_text = get\_font(17).render("Gabriel Medeiros Sollero         - 11811EAU000", True, "Black")

      i\_rect = i\_text.get\_rect(center=(WIDTH/2, 290))

      self.screen.blit(i\_text, i\_rect)

      i\_text = get\_font(17).render("João Victor Luiz de Andrade      - 11811EAU003", True, "Black")

      i\_rect = i\_text.get\_rect(center=(WIDTH/2, 330))

      self.screen.blit(i\_text, i\_rect)

      i\_text = get\_font(17).render("Rodrigo Santana Soares           - 11821EAU013", True, "Black")

      i\_rect = i\_text.get\_rect(center=(WIDTH/2, 370))

      self.screen.blit(i\_text, i\_rect)

      i\_text = get\_font(17).render("Thiago Fernando Cuevas Mestanza  - 11821EAU002", True, "Black")

      i\_rect = i\_text.get\_rect(center=(WIDTH/2, 410))

      self.screen.blit(i\_text, i\_rect)

def get\_font(size): # Returns Press-Start-2P in the desired size

  return pygame.font.Font("Font/font.ttf", size)

* + *settings.py*

#!/usr/bin/python3

#Configurações Iniciais

WIDTH   = 1280          #Comprimento da Tela

HEIGHT  = 720           #Altura da Tela

FPS = 1e+4          #Numero de Telas por Segundo

#TILESIZE = 64 #Tamnho do Ladrilho (Malha)

#Dados de Entrada

h   = 2/FPS         #Declaração da variável de divisão do tempo

theta   = 60            #Declaração do angulo

l1  = 0.75          #Declaração da posição do centro de massa do pendulo

l2  = 1.2           #Declaração do comprimento do pendulo

J   = 1e-2          #Declaração do momento angular do pendulo

mp  = 0.85          #Declaração da massa do conjunto

g   = 9.81          #Aceleração da gravidade padrão

p   = mp \* g        #Calculo da força peso aplicada sobre o pendulo

ua  = 0.1           #Declaração do coeficiente de atrito

# **REFERÊNCIAS**

[1] **Sistemas de Controle Modernos**. Richard C. Dorf e Robert H. Bishop, Editora *LTC*, 2018, 13a. edição, ISBN: 9788521635123.

[2] **Engenharia de Sistemas de Controle** Nise, N. S., 3a Edição, LTC, 2002.