UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
ENGENHARIA MECATRÔNICA
SISTEMAS DIGITAIS PARA MECATRÔNICA (FEELT 49081)

TRABALHO FINAL 1 DA DISCIPLINA SISTEMAS DIGITAIS PARA MECATRÔNICA (FEELT 49081):

PÊNDULO INVERTIDO POR RODA DE REAÇÃO

ARTHUR REIS BELLO	11811EMT009
DOUGLAS DA SILVA CARVALHO	11811EMT025
JOÃO PEDRO ÁVILA DE ALCANTRA	11621EMT009
ISLAM ELOIRRANO RODRIGUES CARVALHO SOUZA	11121EMT004

UBERLÂNDIA
2023

Sumário

UBERLÂNDIA
2023

1. Introdução

A ideia da simulação de um processo é analisar o comportamento de um sistema mediante um cenário específico, bem como as características de um sistema influenciam no todo. Possui uma importância significativa para o entendimento e comportamento prático de um sistema, sendo assim um recurso que foi utilizado para a simulação de um controle de posição de um pêndulo invertido por roda de reação.

Inicialmente, é preciso entender o conceito do sistema utilizado. Ele consiste em uma barra linear pivotada em uma de suas extremidades, com possibilidade de girar em torno deste ponto, enquanto outra extremidade é mantida em posição vertical de alguma forma.

O sistema produz movimentos angulares que modificam a posição do ângulo estabelecido, e após isso se estabiliza a haste em tal posição.

Apesar de um projeto simples, o sistema de um pêndulo invertido por roda de reação tem amplo uso para a realização de teste de diversas técnicas avançadas. Também é um sistema muito útil para análise e demonstração de um controle de sistema. A figura 1 a seguir apresenta uma figura ilustrativa.

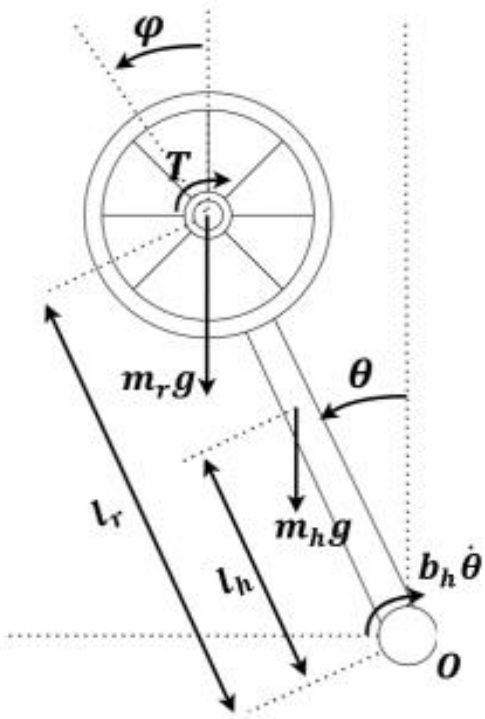

Figura 1: Sistema conceitual de um pêndulo invertido por roda de reação.

2. Objetivos

O primeiro trabalho da disciplina de Sistema Digitais tem por objetivo a implementação da simulação de um sistema de controle que, no caso, tem como exemplo um pêndulo invertido com roda de reação. Ainda, como parte do objetivo, tal implementação foi feita utilizando a linguagem Python e a biblioteca Pygame.

Como parte do objetivo da implementação, foi necessário para o projeto realizar a simulação das leis da física, modelagem e linearização do sistema e, como não poderia faltar, foi necessário também aplicar técnicas de controle para análise da simulação do processo.

3. Biblioteca Utilizadas

As bibliotecas utilizadas foram as seguintes:

- import numpy as np
- import matplotlib.pyplot as plt
- import pygame
- import sys

A primeira biblioteca é utilizada para se trabalhar com computação numérica em linguagem Python, ideal para os cálculos necessários. Já a segunda biblioteca é utilizada para criar e personalizar construções gráficas em linguagem Python, ideal para as análises de resultados necessárias para o processo.

Já o comando 'import pygame' serve para podermos utilizar a biblioteca Pygame, voltada para o desenvolvimento de games e interfaces gráficas escritas em linguagem Python. Por fim, o módulo sys fornece funções e variáveis usadas para manipular diferentes partes do ambiente de tempo de execução do Python.

4. Implementação e Resultados

As simulações dinâmicas e do sistema de controle foram implementadas com a linguagem Python. A representação visual do mesmo é feita com o auxílio da biblioteca do Pygame. A seguir tem a apresentação do código utilizado e dos resultados gráfico obtidos.

O código apresenta os dados a serem inseridos e controlados na simulação. Também está inserido a parte do código do Pygame para a apresentação gráfica da simulação.

Código em Python:

```
import numpy as np
import matplotlib.pyplot as plt
import pygame
import sys
```

```
    def draw_system(phi, theta):
        window.fill((255, 255, 255))  # Clear the window

        # Draw the cart
        cart_x = int(WINDOW_WIDTH / 2)
        cart_y = int(WINDOW_HEIGHT / 2)
        pygame.draw.rect(window, (0, 0, 0), (cart_x - CART_WIDTH/2, cart_y - CART_HEIGHT/2, CART_WIDTH, CART_HEIGHT))

        # Calculate the pendulum tip position
        pendulum_tip_x = int(cart_x + PENDULUM_LENGTH * np.sin(phi))
        pendulum_tip_y = int(cart_y - PENDULUM_LENGTH * np.cos(phi))  # Flip the y-coordinate

        # Calculate the reaction wheel position
        wheel_length = 20
        wheel_x = int(pendulum_tip_x + wheel_length * np.sin(theta))
        wheel_y = int(pendulum_tip_y - wheel_length * np.cos(theta))  # Flip the y-coordinate

        # Draw the pendulum
        pygame.draw.line(window, (0, 0, 0), (cart_x, cart_y), (pendulum_tip_x, pendulum_tip_y), PENDULUM_WIDTH)

        # Draw the reaction wheel
        pygame.draw.circle(window, (255, 255, 0), (int(pendulum_tip_x), int(pendulum_tip_y)), 20)
        pygame.draw.circle(window, (0, 0, 255), (wheel_x, wheel_y), 10)

        # Draw the target
        target_x = int(cart_x + PENDULUM_LENGTH * np.sin(target_phi))
        target_y = int(cart_y - PENDULUM_LENGTH * np.cos(target_phi))  # Flip the y-coordinate
        pygame.draw.circle(window, (255, 0, 0), (target_x, target_y), 10)

        # Draw the angles
        font = pygame.font.SysFont(None, 30)
        phi_text = font.render(f"Phi: {np.degrees(phi):.2f}", True, (0, 0, 0))
        theta_text = font.render(f"Theta: {np.degrees(theta):.2f}", True, (0, 0, 0))
        window.blit(phi_text, (10, 10))
        window.blit(theta_text, (10, 40))

        # Update the display
```

```
        pygame.display.update()



    def complementary_filter(data, filtered_data, alpha):
        return alpha * filtered_data + (1 - alpha) * data

    def simulate_system(a, b, c, phi0, phi_dot0, theta0, theta_dot0, dt, T, control_strategy, target_phi, target_theta, sensor_noise_std, filter_alpha):
        # Initialize time and arrays to store results
        t = np.arange(0, T, dt)
        phi = np.zeros_like(t)
        phi_dot = np.zeros_like(t)
        theta = np.zeros_like(t)
        theta_dot = np.zeros_like(t)

        # Set initial conditions
        phi[0] = phi0
        phi_dot[0] = phi_dot0
        theta[0] = theta0
        theta_dot[0] = theta_dot0

        # PID controller variables
        integral_error_phi = 0.0
        integral_error_theta = 0.0
        prev_error_phi = 0.0
        prev_error_theta = 0.0

        # Filter variables
        filtered_phi = phi[0]
        filtered_theta = theta[0]

        # Simulation loop
        for i in range(1, len(t)):
            # Calculate errors
            error_phi = target_phi - phi[i-1]
            error_theta = target_theta - theta[i-1]

            # Calculate control input using PID control
            u = control_strategy(error_phi, error_theta, dt, integral_error_phi, integral_error_theta, prev_error_phi, prev_error_theta)

            # Calculate the second derivatives
```

```
            phi_double_dot = a * np.sin(phi[i-1]) - b * u
            theta_double_dot = c * u

            # Update phi, phi_dot, theta, theta_dot using Euler's method
            phi[i] = phi[i-1] + dt * phi_dot[i-1]
            phi_dot[i] = phi_dot[i-1] + dt * phi_double_dot
            theta[i] = theta[i-1] + dt * theta_dot[i-1]
            theta_dot[i] = theta_dot[i-1] + dt * theta_double_dot

            # Add sensor noise to measurements
            noisy_phi = phi[i] + np.random.normal(0, sensor_noise_std)
            noisy_theta = theta[i] + np.random.normal(0, sensor_noise_std)

            # Apply complementary filter
            filtered_phi = complementary_filter(noisy_phi, filtered_phi, filter_alpha)
            filtered_theta = complementary_filter(noisy_theta, filtered_theta, filter_alpha)

            # Update measurements with filtered values
            phi[i] = filtered_phi
            theta[i] = filtered_theta

            # Update integral and previous errors for PID controller
            integral_error_phi += error_phi * dt
            integral_error_theta += error_theta * dt
            prev_error_phi = error_phi
            prev_error_theta = error_theta

            # Draw the system
            draw_system(phi[i], theta[i])
            clock.tick(100)

        return t, phi, phi_dot, theta, theta_dot

    # Control strategy (PID controller)
    def pid_controller(error_phi, error_theta, dt, integral_error_phi, integral_error_theta,
prev_error_phi, prev_error_theta):
        # Constants for the controller
        Kp_phi = 15  # Proportional gain for phi control
        Ki_phi = 8  # Integral gain for phi control
        Kd_phi = 10  # Derivative gain for phi control
        Kp_theta =0.5  # Proportional gain for theta control
        Ki_theta = 0.2  # Integral gain for theta control
        Kd_theta = 0.1  # Derivative gain for theta control
```

```
        # Control input calculation
        proportional_term_phi = Kp_phi * error_phi
        integral_term_phi = Ki_phi * integral_error_phi
        derivative_term_phi = Kd_phi * (error_phi - prev_error_phi) / dt

        proportional_term_theta = Kp_theta * error_theta
        integral_term_theta = Ki_theta * integral_error_theta
        derivative_term_theta = Kd_theta * (error_theta - prev_error_theta) / dt

        u = -(proportional_term_phi + integral_term_phi + derivative_term_phi) - (proportional_term_theta + integral_term_theta + derivative_term_theta)

        return u

    # Parameters
    a = 1.0
    b = 0.5
    c = 2.0
    phi0 = 0.7
    phi_dot0 = 0.0
    theta0 = 0
    theta_dot0 = 0.0
    dt = 0.01
    T = 100.0
    target_phi = 0
    target_theta = 0.3
    sensor_noise_std = 0.01  # Standard deviation of sensor noise
    filter_alpha = 0.8  # Alpha value for the complementary filter

    # Pygame parameters
    WINDOW_WIDTH = 800
    WINDOW_HEIGHT = 600
    PENDULUM_LENGTH = 150
    PENDULUM_WIDTH = 5
    CART_WIDTH = 80
    CART_HEIGHT = 40

    pygame.init()
    clock = pygame.time.Clock()
    window = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
    pygame.display.set_caption("Inverted Pendulum Simulation")
```

```
    # Simulate the system
    t, phi, phi_dot, theta, theta_dot = simulate_system(a, b, c, phi0, phi_dot0, theta0, theta_dot0, dt, T, pid_controller, target_phi, target_theta, sensor_noise_std, filter_alpha)


    pygame.quit()
    sys.exit()

    # Plot the results
    plt.figure()
    plt.subplot(2, 1, 1)
    plt.plot(t, phi, label='Phi')
    plt.plot(t, phi_dot, label='Phi_dot')
    plt.axhline(y=target_phi, color='r', linestyle='--', label='Target Phi')
    plt.xlabel('Time')
    plt.ylabel('Phi / Phi_dot')
    plt.legend()

    plt.subplot(2, 1, 2)
    plt.plot(t, theta, label='Theta')
    plt.plot(t, theta_dot, label='Theta_dot')
    plt.axhline(y=target_theta, color='r', linestyle='--', label='Target Theta')
    plt.xlabel('Time')
    plt.ylabel('Theta / Theta_dot')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

Para uma análise bem realizada, foi feita a simulação da planta ideal, a simulação com o acionamento de ruído e, por fim, a simulação com um filtro complementar. Por meio de tais simulações, foi possível analisar um sistema real, um outro com possíveis interferências do meio e um sistema com filtro para as possíveis interferências.

As figuras 2, 3 e 4 a seguir ilustram o realizado anteriormente.

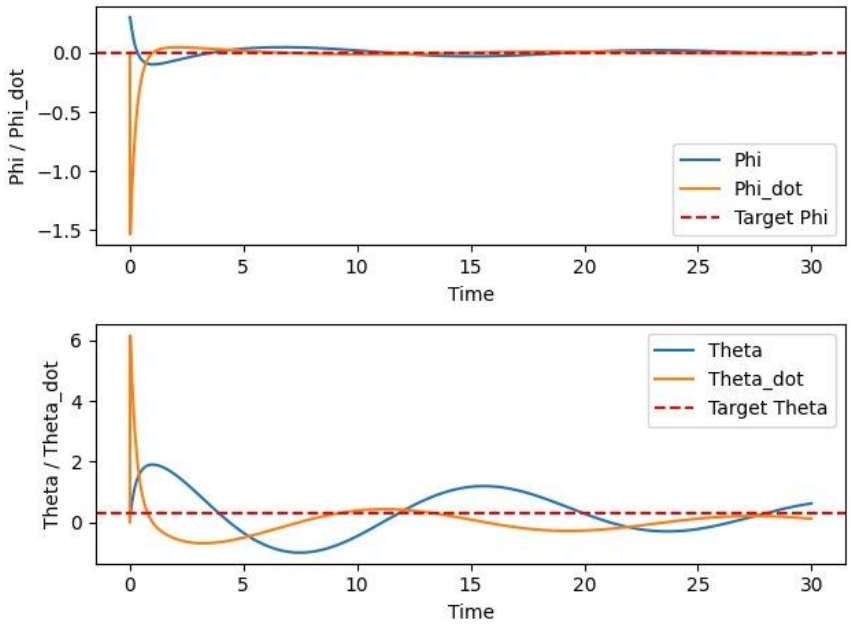

Figura 2: Simulação da planta.

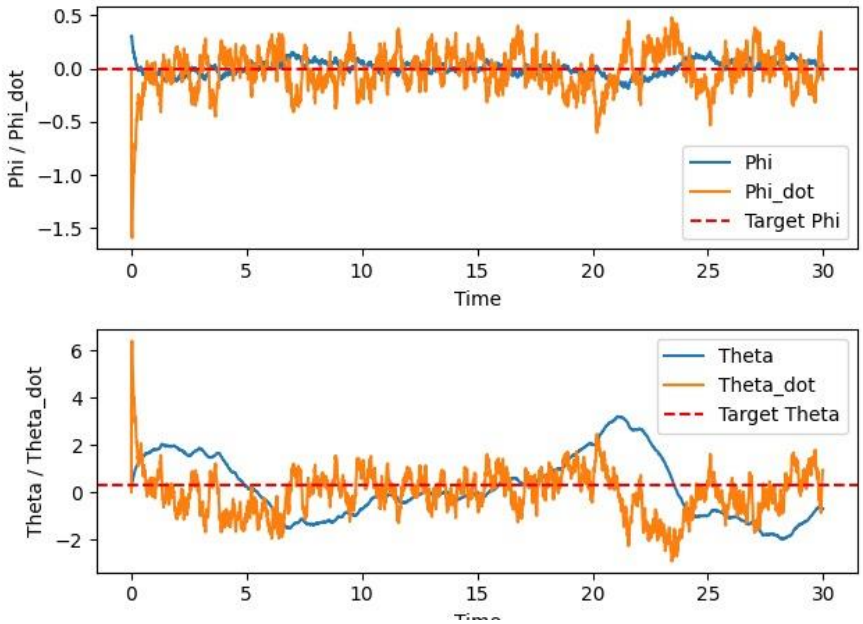

Figura 3: Simulação da planta com ruído acionado.

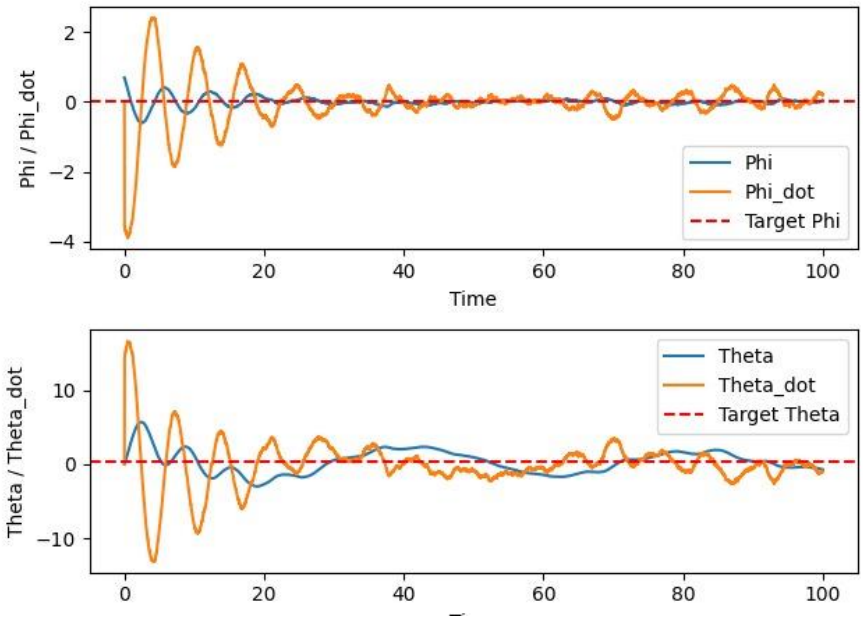

Figura 4: Simulação da planta com filtro complementar.

A seguir também será apresentada a figura 5 da representação visual do sistema, obtido da plataforma Pygame.

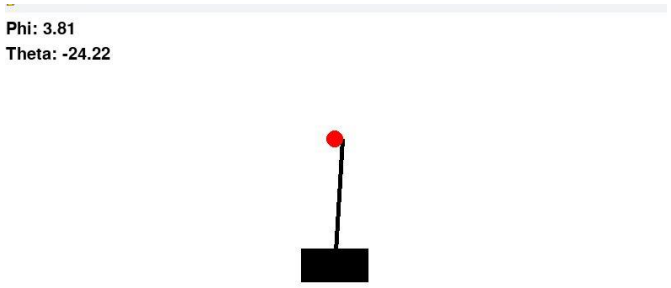

Figura 5: Simulação visual do sistema.

Já para a parte física do projeto, utilizamos o seguinte código para a execução e manipulação do sistema:

```
#include <Arduino.h>

#define LED 2

#define POT 32

#define motorPWM 26
#define motorIN1 33
#define motorIN2 25
#define Kp 200
#define Ki 0.001
#define Kd 90
#define Ta 1

#define target 8

const int numSamples = 50; // Number of samples to filteredInclination
double samples[numSamples];  // Array to store the samples
int currentIndex = 0;     // Current index in the array

double sensorMax = 2665.5;
double sensorMin = 1665.5;
double sensor = 0;
double angle = 0;
double speed = 0;

double error = 0;
double lasterror = 0;

double P = 0;
double Integral = 0;
double D = 0;

double readFilteredSensor(){
   // Read the sensor value or any input data
 double sensorValue = analogRead(POT);

 // Update the samples array
 samples[currentIndex] = sensorValue;
 currentIndex = (currentIndex + 1) % numSamples; // Circular buffer
```

```
  // Calculate the filteredInclination
  double filteredInclination = 0;
  for (int i = 0; i < numSamples; i++) {
    filteredInclination += samples[i];
  }
  filteredInclination /= numSamples;
  return filteredInclination;

}

void setupFilter(){
  for(int i = 0; i < numSamples; i++){
    samples[i] = 0.0;
  }
}



void printSensor(){
  Serial.print("angleVal: ");
  Serial.print(angle);
  Serial.print(" sensorMin: ");
  Serial.print(sensorMin);
  Serial.print(" sensorMax: ");
  Serial.print(sensorMax);
  Serial.print(" sensorVal: ");
  Serial.print(sensor);
  Serial.print(" speed: ");
  Serial.print(speed);
  Serial.print(" error: ");
  Serial.print(error);
  Serial.print(" I: ");
  Serial.print(Integral);
  Serial.println();
  Serial.println();
}

void moveMotor(int speed){
  if(speed>=0){
    digitalWrite(motorIN1, HIGH);
    digitalWrite(motorIN2, LOW);
    analogWrite(motorPWM, speed);
```

```
  }
  else if(speed<0){
    digitalWrite(motorIN1, LOW);
    digitalWrite(motorIN2, HIGH);
    analogWrite(motorPWM, -speed);
  }

}

float mapf(float x, float in_min, float in_max, float out_min, float out_max) {
    float result;
    result = (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
    return result;
}

void setup(){
  Serial.begin(115200);

  pinMode(POT, INPUT);
  pinMode(motorIN1,OUTPUT);
  pinMode(motorIN2,OUTPUT);
  pinMode(motorPWM,OUTPUT);


  Serial.println("---Reading initial values, please wait.---");
  setupFilter();
  for (int i = 0; i<100; i++){
    readFilteredSensor();
  }

}


void loop(){

  unsigned long now = millis();
  sensor = readFilteredSensor();
  // if(sensor>sensorMax) sensorMax = sensor;
  // if(sensor<sensorMin) sensorMin = sensor;


  // Output the sensor
```

```
    angle = mapf(sensor, sensorMin, sensorMax, -45, 45);
    error = target - angle;
    printSensor();

    P = Kp*error;
    Integral = Integral + Ki*(error);
    D = Kd*(error-lasterror);

    lasterror = error;

    speed = P+Integral+D;
    speed = constrain(speed, -255, 255);
    moveMotor(speed);
    //Serial.println(-speed);

    while(millis() - now < Ta){

    }
  }
```

A seguir também se tem uma figura para ilustrarmos o sistema físico montado. Foi realizada a modelagem e impressão 3D da haste do pêndulo invertido e do pêndulo invertido em si.

Figura 6: Sistema físico montado.

5. Conclusão

O trabalho foi muito útil e eficaz na conclusão dos objetivos propostos. Foi possível ter uma ampla experiência e aprendizagem com a linguagem Python, bem como a plataforma Pygame.

Em relação ao sistema proposto a ser analisado, foi possível analisar os diversos fatores que influenciam o sistema de um aeropêndulo. Foi possível entender a influência da velocidade do motor e da corrente de ar criada que equilibra o sistema em uma posição específica.

Por fim, realizar a simulação de um modo geral conseguiu elucidar de forma considerável a importância da simulação de um sistema, a importância de entender as influências em um sistema e, por fim, e como é interessante observarmos algo na simulação antes de colocarmos em prática. Em certos casos (principalmente operações de riscos) a simulação é essencial.

6. Referências

[1] SOUZA, Thiago Francisco Ferreira de – Projeto de Sistema de Controle de um Pêndulo Invertido com Roda de Reação. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/48892/1/THIAGO%20FRANCISCO%20FERREIRA%20DE%20SOUZA.pdf

[2] SILVA, Yago Luiz Monteiro – Projeto, construção e controle de um aeropêndulo. Disponível em: http://dspace.sti.ufcg.edu.br:8080/jspui/bitstream/riufcg/18954/1/YAGO%20LUIZ%20MONTEIRO%20SILVA%20-%20TCC%20ENG.%20EL%C3%89TRICA%202018.pdf

[3] Discrete-time inverse optimal control for a reaction wheel pendulum: a passivity-based control approach. Disponível em: https://www.redalyc.org/journal/5537/553768213012/html/

[4] Python. Disponível em: https://www.python.org/

[5] Pygame. Disponível em: https://www.pygame.org/news

[6] Introdução ao Pygame. Disponível em: http://www.linhadecodigo.com.br/artigo/503/introducao-ao-pygame.aspx