

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Arthur Reis Bello

**Simulação e implementação de um sistema
físico de pêndulo invertido sobre duas rodas**

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Arthur Reis Bello

**Simulação e implementação de um sistema físico de
pêndulo invertido sobre duas rodas**

Trabalho de conclusão de curso apresentado
à Faculdade de Engenharia Mecânica da Uni-
versidade Federal de Uberlândia, Minas Ge-
rais, como requisito exigido parcial à obten-
ção do grau de Bacharel em Engenharia Me-
catrônica.

Orientador: Werley Rocherter Borges Ferreira

Universidade Federal de Uberlândia – UFU

Faculdade de Engenharia Mecânica

Bacharelado em Engenharia Mecatrônica

Uberlândia, Brasil

2024

Arthur Reis Bello

Simulação e implementação de um sistema físico de pêndulo invertido sobre duas rodas

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Engenharia Mecatrônica.

Trabalho aprovado. Uberlândia, Brasil, 29 de novembro de 2024:

Werley Rocherter Borges Ferreira
Orientador

Pedro Pio Rosa Nishida

Guênia Mara Vieira Ladeira

Uberlândia, Brasil
2024

Agradecimentos

Agradeço aos meus pais, Ricardo e Cida, pelo apoio. À Laura, que me acompanhou em toda jornada acadêmica. Aos professores que contribuíram para minha formação.

Resumo

Este trabalho apresenta o projeto e desenvolvimento de um robô autobalanceado sobre duas rodas. O sistema utiliza motores de passo NEMA 17 como atuadores, controlados por drivers A4988 e um microcontrolador Arduino Nano. Baseando-se em um modelo matemático da literatura, foi realizada a conversão para o espaço de estados, permitindo o desenvolvimento de estratégias avançadas de controle. O estudo inclui procedimentos experimentais para a estimativa dos parâmetros da planta, fundamentais para o projeto do controlador e do observador de estados. Por fim, obteve-se um sistema capaz de se equilibrar, tanto em simulação quanto no sistema físico.

Palavras-chave: Pêndulo invertido sobre duas rodas, motor de passo, espaço de estados.

Abstract

This work presents the design and development of a self-balancing two-wheeled robot. The system uses NEMA 17 stepper motors as actuators, controlled by A4988 drivers and an Arduino Nano microcontroller. Based on a mathematical model from the literature, the system was converted to state-space representation, enabling the development of advanced control strategies. The study includes experimental procedures for estimating the plant parameters, which are essential for the design of the controller and state observer. Finally, a system capable of balancing itself was achieved, both in simulation and on the physical prototype.

Palavras-chave: *Two wheeled self balancing robot, stepper motor, state space.*

Lista de figuras

Figura 1 – Modelo físico detalhado	12
Figura 2 – Diagrama de corpo livre do sistema com a indicação do raio R do círculo.	12
Figura 3 – Motor de Passo NEMA17 (Fonte: (ELETROGATE, 2024))	16
Figura 4 – Driver A4988 (Fonte: (ELETROGATE, 2024))	17
Figura 5 – Controle do motor de passo (Fonte: (KUNG, 2019))	17
Figura 6 – Arduino Nano (Fonte: (ELETROGATE, 2024))	17
Figura 7 – Orange Pi 3 lts (Fonte: (ELETROGATE, 2024))	18
Figura 8 – Unidade de medição inercial MPU6050 (Fonte: (ELETROGATE, 2024))	18
Figura 9 – Bateria de LiPo 11,1 V (Fonte: (ZEEE, 2024))	19
Figura 10 – Estrutura física	19
Figura 11 – Diagrama de corpo livre de um pêndulo simples	20
Figura 12 – Procedimento para estimar o centro de massa	21
Figura 13 – Estimativa do centro de massa	21
Figura 14 – Dados do procedimento para obter momento de inércia	22
Figura 15 – Transformada rápida de Fourier	23
Figura 16 – Visualização descrição do robô	24
Figura 17 – Controlador no espaço de estados (Fonte: (OGATA, 2010))	25
Figura 18 – Observador de estados (Fonte: (OGATA, 2010))	25
Figura 19 – Simulação da planta em malha aberta	27
Figura 20 – Simulação da planta em malha fechada	28
Figura 21 – Saída do modelo físico após implementação do controlador	28

Lista de tabelas

Tabela 1 – Parâmetros estimados da planta	23
---	----

Lista de abreviaturas e siglas

FFT	Fast Fourier Transform
IMU	Inertial Measurement Unit
ROS	Robot Operating System
SISO	Single Input Single Output
URDF	Universal Robot Description Format

Lista de símbolos

θ_b	Inclinação do Corpo em relação ao eixo y
J_b	Momento de Inércia do Corpo
m_b	Massa do Corpo
l	Distância do centro das rodas ao centro de massa do corpo
m_{wa}	Massa da montagem da roda
J_w	Momento de inércia da montagem das rodas
R	Raio das rodas
x	Deslocamento horizontal do centro das rodas
u	Entrada do sistema de controle
X	Vetor de estados
Y	Saída do sistema
A	Matriz de estados
B	Matriz de entrada
C	Matriz de saída
B	Matriz de transição direta
g	gravidade
m	massa do pêndulo
d	distância entre ponto de apoio e centro de massa
I	momento de inércia do pêndulo
ω	frequência de oscilação do pêndulo
K	ganho do controlador no espaço de estados
$z_1 e z_2$	polos desejados do sistema
K_e	ganho do estimador preditivo

Sumário

1	INTRODUÇÃO	11
2	FUNDAMENTAÇÃO TEÓRICA	12
2.1	Modelagem matemática	12
2.2	Modelo no espaço de estados	14
3	DESENVOLVIMENTO	16
3.1	Montagem do modelo físico	16
3.2	Obtenção dos parâmetros	19
3.3	Simulação no Gazebo	24
3.4	Projeto do controlador	24
4	RESULTADOS	27
4.1	Simulação	27
5	CONCLUSÕES E SUGESTÕES DE TRABALHOS FUTUROS	29
	REFERÊNCIAS	31
	APÊNDICES	32
	APÊNDICE A – DESCRIÇÃO DO ROBÔ	33
	APÊNDICE B – CÓDIGO DO ROBÔ	38

1 Introdução

A palavra "robô" carrega diferentes significados dependendo do contexto e da perspectiva de quem a utiliza (TSAI, 1999). De acordo com International Organization for Standardization (2021), um robô é definido como um mecanismo atuado programado com algum grau de autonomia, projetado para realizar tarefas como locomoção, manipulação ou posicionamento. Com os avanços tecnológicos, a robótica tornou-se cada vez mais presente em diversos cenários. Desde operações em ambientes industriais, como montagem e transporte de mercadorias, até aplicações no transporte de pessoas e execução de tarefas domésticas, a robótica demonstra sua versatilidade. Dentro desse campo, a robótica móvel se destaca, sendo um ramo da engenharia dedicado ao estudo de robôs capazes de se movimentar livremente pelo ambiente. Sua relevância cresce devido à ampla gama de aplicações, incluindo logística, exploração de terrenos e robótica de serviço.

A escolha do sistema de locomoção para um robô móvel depende diretamente da aplicação pretendida. Por exemplo, robôs com pernas articuladas são mais adequados para terrenos irregulares, pois conseguem superar obstáculos que sistemas baseados em rodas não conseguem. No entanto, esses robôs apresentam maior custo e complexidade de controle devido ao número elevado de graus de liberdade. Por outro lado, robôs sobre rodas oferecem maior eficiência energética, simplicidade de construção e facilidade de controle, sendo amplamente utilizados em ambientes com superfícies regulares (CHAN; STOL; HALKYARD, 2013).

Este documento apresenta o desenvolvimento de um robô autobalanceado de duas rodas, com foco na utilização de motores de passo como atuadores e um sensor inercial para medição. O trabalho inclui a formulação da planta matemática no modelo de espaço de estados, baseada em um modelo extraído de Kung (2019). Além disso, são descritos os experimentos realizados para a identificação dos parâmetros do modelo, seguidos pelo projeto do controlador no espaço de estados. Como alguns dos estados propostos não podem ser diretamente medidos, o trabalho também descreve o projeto de um observador de estados.

2 Fundamentação Teórica

2.1 Modelagem matemática

O modelo físico do robô é mostrado pela figura 1. Deseja-se um modelo matemático capaz de descrever a sua dinâmica.

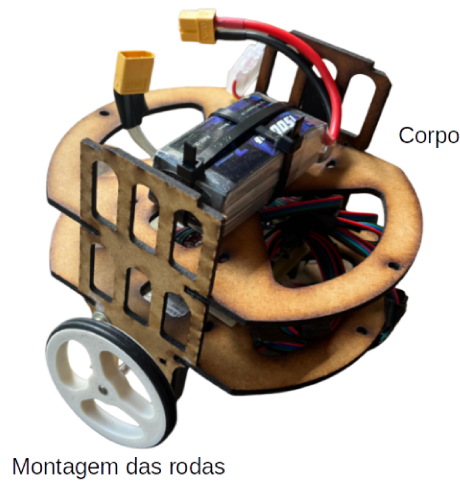


Figura 1 – Modelo físico detalhado

A representação matemática do modelo físico pode ser realizada por meio de uma haste conectada a duas rodas que podem ser submetidas a um torque advindo de um motor cada. O diagrama de corpo livre está representado pela figura 2.

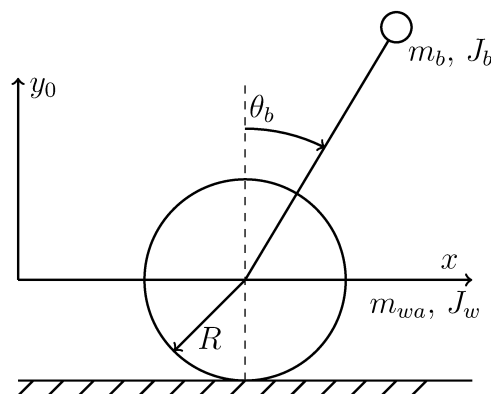


Figura 2 – Diagrama de corpo livre do sistema com a indicação do raio R do círculo.

O modelo matemático completo, retirado do artigo de Kung (2019), é mostrado pelas equações 2.1 e 2.2:

$$\ddot{x} = \left[m_b + 2 \left(m_{wa} + \frac{J_w}{R^2} \right) - \frac{(m_b l \cos(\theta_b))^2}{m_b l^2 + J_b} \right]^{-1} \cdot \left\{ (m_b l \sin(\theta_b)) \dot{\theta}_b^2 - \frac{(m_b l)^2 g \cos(\theta_b) \sin(\theta_b)}{m_b l^2 + J_b} + 2 \left(\frac{1}{R} + \frac{m_b l \cos(\theta_b)}{m_b l^2 + J_b} \right) T_w \right\} \quad (2.1)$$

$$\ddot{\theta}_b = \left[J_b + m_b l^2 - \frac{(m_b l \cos(\theta_b))^2}{m_b + 2(m_{wa} + \frac{J_w}{R^2})} \right]^{-1} \cdot \left\{ (m_b l \sin(\theta_b)) \left[g - \frac{m_b l \cos(\theta_b) (\dot{\theta}_b^2)}{m_b + 2(m_{wa} + \frac{J_w}{R^2})} \right] - 2 \left[R + \frac{m_b l \cos(\theta_b)}{m_b + 2(m_{wa} + \frac{J_w}{R^2})} \right] \frac{T_w}{R} \right\} \quad (2.2)$$

Onde:

- $\theta_b \rightarrow$ inclinação do corpo;
- $J_b \rightarrow$ momento de inércia do corpo;
- $m_b \rightarrow$ massa do corpo;
- $l \rightarrow$ distância do centro das rodas ao centro de massa do corpo;
- $m_{wa} \rightarrow$ massa da montagem da roda;
- $J_w \rightarrow$ momento de inércia da montagem das rodas;
- $R \rightarrow$ raio das rodas;
- $x \rightarrow$ deslocamento horizontal do centro das rodas;

Deseja-se controlar o sistema em torno do ponto de equilíbrio $\theta_b = \dot{\theta}_b = 0$, para isso, é conveniente ter-se um modelo linear. Assim, utilizando $\sin(\theta_b) \cong 0$, $\cos(\theta_b) \cong 1$ e $\dot{\theta}_b \cong 0$ tem-se:

$$\ddot{x} = -C_1 \theta_b + C_2 T_w \quad (2.3)$$

$$\ddot{\theta}_b = C_3 \theta_b - C_4 T_w \quad (2.4)$$

Onde:

$$C_1 = \frac{1}{M} \frac{(m_b l)^2 g}{m_b l^2 + J_b} \quad (2.5)$$

$$C_2 = \frac{2}{M} \left(\frac{1}{R} + \frac{m_b l}{m_b l^2 + J_b} \right) \quad (2.6)$$

$$C_3 = \frac{m_b g l}{J} \quad (2.7)$$

$$C_4 = \frac{2}{JR} \left[R + \frac{m_b l}{m_b + 2 \left(m_{wa} + \frac{J_w}{R^2} \right)} \right] \quad (2.8)$$

$$M = m_b + 2 \left(m_{wa} + \frac{J_w}{R^2} \right) - \frac{(m_b l)^2}{m_b l^2 + J_b} \quad (2.9)$$

$$J = J_b + m_b l^2 - \frac{m_b l^2}{m_b + 2 \left(m_{wa} + \frac{J_w}{R^2} \right)} \quad (2.10)$$

Combinando 2.3 e 2.4 e assumindo a entrada do sistema (u) como sendo a velocidade em x do eixo das rodas $u = \dot{x}$, $\dot{u} = \ddot{x}$, a saída $y = \theta_b$, tem-se:

$$\dot{u} = \left[\frac{C_2}{R} \left(\frac{C_3}{C_4} - \frac{C_1}{C_2} \right) \right] y - \left(\frac{C_2}{RC_4} \right) \ddot{y} \quad (2.11)$$

A equação apresentada é uma simplificação e linearização do modelo completo do sistema. Ela será utilizada para a conversão ao formato de espaço de estados e, posteriormente, para o projeto do controlador. Com isso, espera-se que, mesmo após as simplificações, a equação consiga representar adequadamente o comportamento do sistema na região de interesse.

2.2 Modelo no espaço de estados

O modelo de espaço de estados é conveniente para projeto, tanto do controlador quanto do observador de estados. Assim, deseja-se escrever o sistema na forma da equação matricial 2.12 e 2.13:

$$\dot{X}(t) = AX(t) + BU(t) \quad (2.12)$$

$$Y(t) = CX(t) + DU(t) \quad (2.13)$$

Para fins de simplicidade o modelo 2.11 será escrito como 2.14:

$$\dot{u} = ay - b\ddot{y} \quad (2.14)$$

Onde:

$$a = \left[\frac{C_2}{R} \left(\frac{C_3}{C_4} - \frac{C_1}{C_2} \right) \right] \quad (2.15)$$

$$b = \left(\frac{C_2}{RC_4} \right) \quad (2.16)$$

Uma possível abordagem para o problema é fazer:

$$x_1 = y \quad (2.17)$$

$$x_2 = u + b\dot{y} \quad (2.18)$$

Assim tem-se:

$$\dot{x}_1 = \dot{y} \quad (2.19)$$

$$\dot{x}_2 = \dot{u} + b\ddot{y} \quad (2.20)$$

Substituindo 2.19 em 2.18 tem-se:

$$\dot{x}_1 = \frac{x_2}{b} - \frac{u}{b} \quad (2.21)$$

Substituindo 2.14 em 2.20 tem-se:

$$\dot{x}_2 = ay = ax_1 \quad (2.22)$$

O sistema fica então:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1/b \\ a & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -1/b \\ 0 \end{bmatrix} u \quad (2.23)$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (2.24)$$

Com o modelo convertido, será possível projetar um controlador. Observa-se que o estado x_1 , correspondente à inclinação, pode ser medido diretamente por um sensor inercial. Contudo, o estado x_2 não possui um significado físico direto e não pode ser medido diretamente por nenhum sensor disponível. Assim, torna-se necessário o projeto de um observador de estados para estimar os valores de x_1 e x_2 .

3 Desenvolvimento

3.1 Montagem do modelo físico

O objetivo deste trabalho é controlar um pêndulo invertido montado sobre duas rodas, utilizando motores de passo para garantir o movimento preciso. Motores de passo são amplamente utilizados em sistemas que exigem controle preciso de posição, pois possuem a capacidade de converter pulsos elétricos em incrementos exatos de deslocamento. Essa característica possibilita também um controle preciso da velocidade. Para este projeto, foram escolhidos motores de passo NEMA 17 (Fig.3), com um motor para cada roda, que possui um torque de 4,2 kgf.cm.

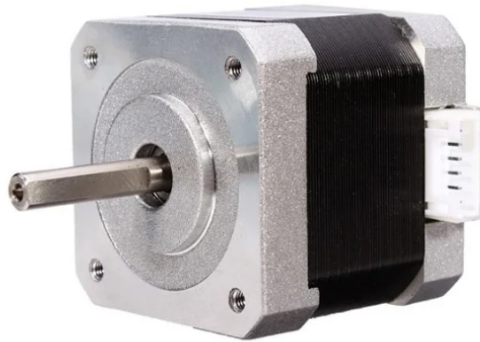


Figura 3 – Motor de Passo NEMA17 (Fonte: (ELETROGATE, 2024))

Para controlar o motor NEMA 17, é necessário utilizar um driver capaz de operar em modos de micropassos. Uma opção eficiente em termos de custo-benefício é o driver A4988 (Fig.4), que pode operar com resolução de até 1/8 de passo. Essa resolução é mais do que suficiente, dado que o ângulo de passo do NEMA 17 é de 1,8 ° por passo. O funcionamento deste componente se dá configurando o comprimento de cada incremento de ângulo conforme o nível lógico aplicado nos pinos MS, a direção no pino DIR e aplicando um pulso para que o movimento ocorra. Para a aplicação desse trabalho, deseja-se controlar sua velocidade, que será proporcional à frequência de pulsos aplicada na porta STEP, como mostra a figura 5.

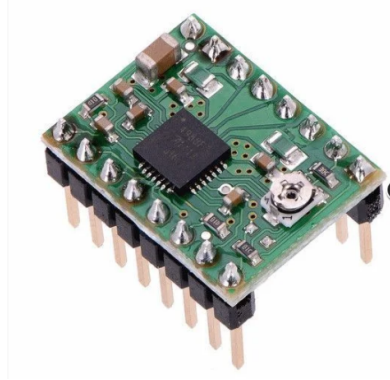


Figura 4 – Driver A4988 (Fonte: (ELETROGATE, 2024))

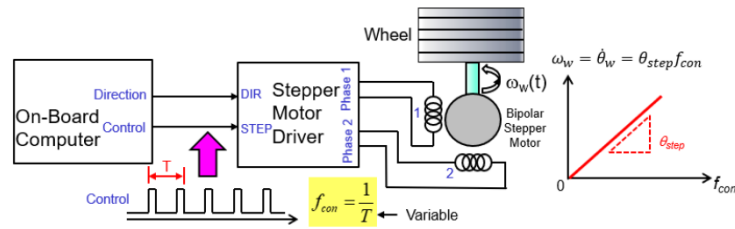


Figura 5 – Controle do motor de passo (Fonte: (KUNG, 2019))

O microcontrolador escolhido para o sistema é o Arduino Nano (Fig.6), que utiliza o processador ATmega328 rodando a 16 MHz, sendo adequado para o controle das operações do sistema. A inclinação do pêndulo em relação ao solo será medida por uma Unidade de Medição Inercial (IMU), garantindo o monitoramento preciso da posição.

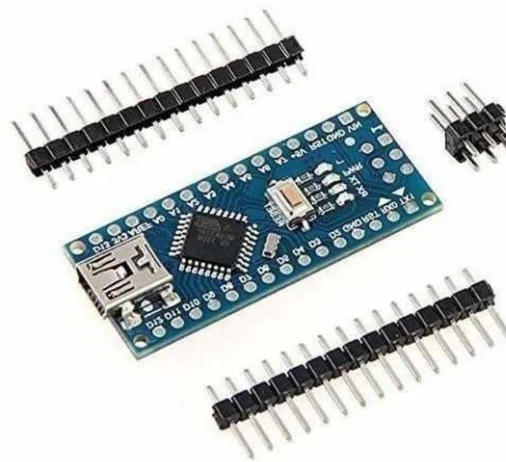


Figura 6 – Arduino Nano (Fonte: (ELETROGATE, 2024))

A comunicação e o controle geral do sistema serão realizados por uma Orange

Pi 3 (Fig.7), rodando o ROS (Robot Operating System), que comandará os motores e processará os dados de controle.

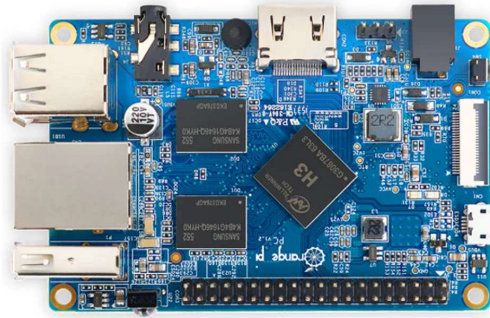


Figura 7 – Orange Pi 3 lts (Fonte: (ELETROGATE, 2024))

O sensor responsável por estimar a inclinação do robô escolhido foi o MPU6050 (Fig.8), devido a sua vasta documentação e utilização em projetos similares. Aliado ao mpu, foi utilizado a biblioteca de filtro de Kalman (LAUSZUS, 2012), capaz de reduzir consideravelmente o ruído.

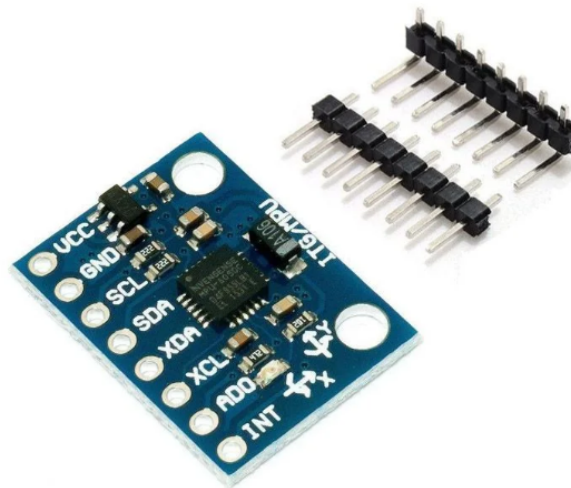


Figura 8 – Unidade de medição inercial MPU6050 (Fonte: (ELETROGATE, 2024))

A alimentação elétrica do mecanismo será feita por uma bateria LiPo de 11,1 V (Fig.9), fornecendo a energia necessária para o funcionamento dos componentes.



Figura 9 – Bateria de LiPo 11,1 V (Fonte: (ZEEE, 2024))

A estrutura física (Fig.10) que acomoda todos os componentes mencionados é fabricada por meio de corte a laser em MDF e impressão 3D em plástico ABS.

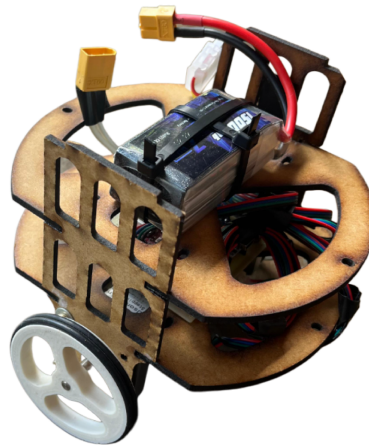


Figura 10 – Estrutura física

Com o modelo físico construído, deve-se então estimar o parâmetros para utilização do modelo matemático descrito nos capítulos anteriores.

3.2 Obtenção dos parâmetros

Para o desenvolvimento do controlador, é necessário obter os parâmetros do sistema, como a massa e a inércia do corpo, a massa e a inércia das rodas, e a posição do centro de massa do corpo. Para as rodas, considerando que o motor do sistema é o mesmo

utilizado no artigo de Kung (2019), adotou-se o mesmo momento de inércia mencionadas no estudo.

A medição da massa do robô foi realizada de maneira simples, utilizando uma balança digital. Esse valor é um parâmetro importante para o cálculo posterior do momento de inércia. A massa obtida foi de 702 g para o corpo e 96 g para cada roda.

Quando um corpo é suspenso por um único ponto de apoio que não exerce torque (Fig. 11), ele tende a oscilar em um movimento harmônico amortecido até alcançar uma posição de equilíbrio. O modelo matemático do sistema em regime permanente é descrito pela equação 3.1. Os únicos valores de α que satisfazem esta equação são da forma $0 + n\pi$, com $n \in \mathbb{Z}$. Isso implica que a linha que conecta o centro de massa do corpo ao ponto de apoio estará necessariamente alinhada com a direção da aceleração gravitacional. Este princípio será aplicado para estimar a posição do centro de massa.

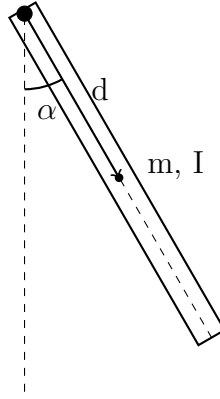


Figura 11 – Diagrama de corpo livre de um pêndulo simples

$$\tau_{res} = 0 = gmd \sin(\alpha) \quad (3.1)$$

Assim, utilizou-se de uma bancada (Fig. 12) que permite a suspensão do robô por um único ponto de apoio para então inferir sua inclinação. Conhecendo o ponto e a inclinação do corpo, define-se uma reta e, ao repetir esse procedimento outra vez com outro apoio, obtém-se uma segunda reta que se cruza com a primeira no centro de massa.

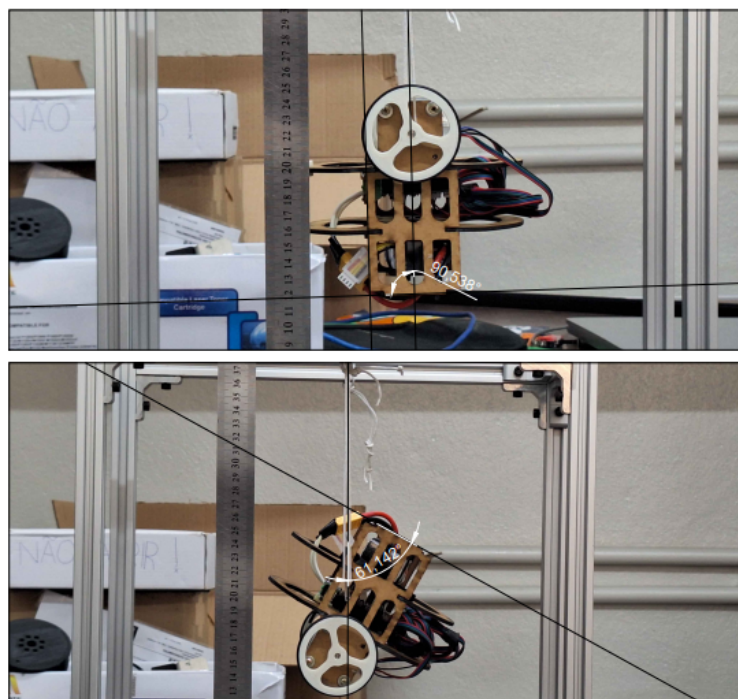


Figura 12 – Procedimento para estimar o centro de massa

O resultado do procedimento (Fig.13) indica que o centro de massa está localizado a 18,06 mm de altura em relação ao eixo das rodas, com um deslocamento horizontal de 0,17 mm. Como esse deslocamento horizontal é muito pequeno, ele será desconsiderado.

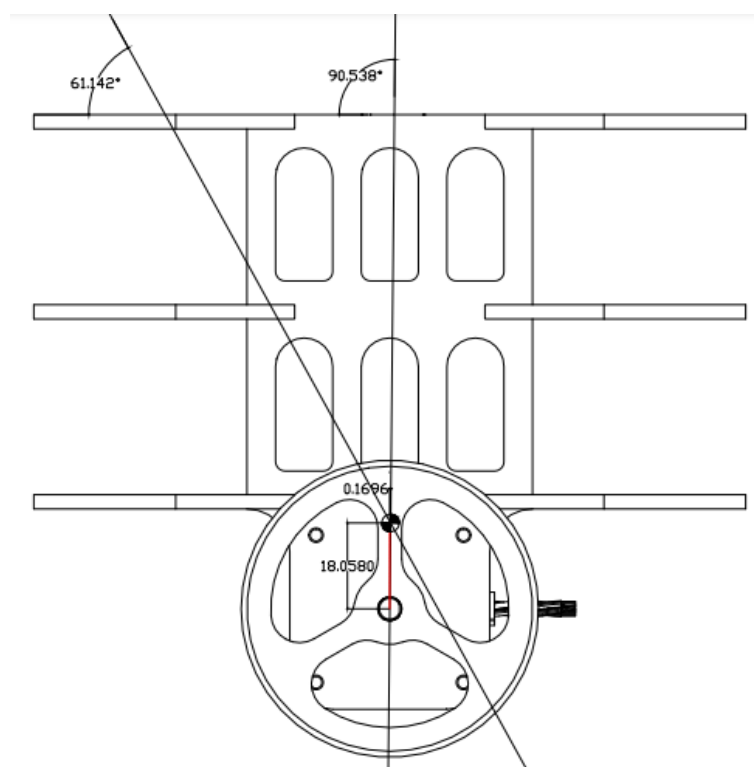


Figura 13 – Estimativa do centro de massa

Para estimar o momento de inércia, foi utilizada uma bancada de teste com um apoio fixo no eixo das rodas, configurando o sistema como um pêndulo simples. Ao soltar o robô de uma posição inclinada inicial, o sistema oscila em uma frequência específica, a partir da qual é possível calcular o momento de inércia. Utilizou-se o software *Tracker*, gratuito e de código aberto, para capturar dados por meio da análise de um vídeo do movimento. Nesse processo, marca-se o centro de um eixo de coordenadas e uma distância conhecida, usada para calibrar o valor em milímetros de cada pixel. Em seguida, escolhe-se um ponto de interesse cuja posição é registrada em cada quadro do vídeo, possibilitando a exportação dos dados para análise posterior. Para aumentar a taxa de amostragem de dados e reduzir o efeito de desfoque (blur), causado pelo movimento durante o intervalo de captura de cada quadro, optou-se por gravar o vídeo em câmera lenta e com boa iluminação. Assim, com uma taxa de amostragem de 4,16 ms, ou seja, 240 quadros por segundo, obteve-se os dados apresentados na Figura 14.

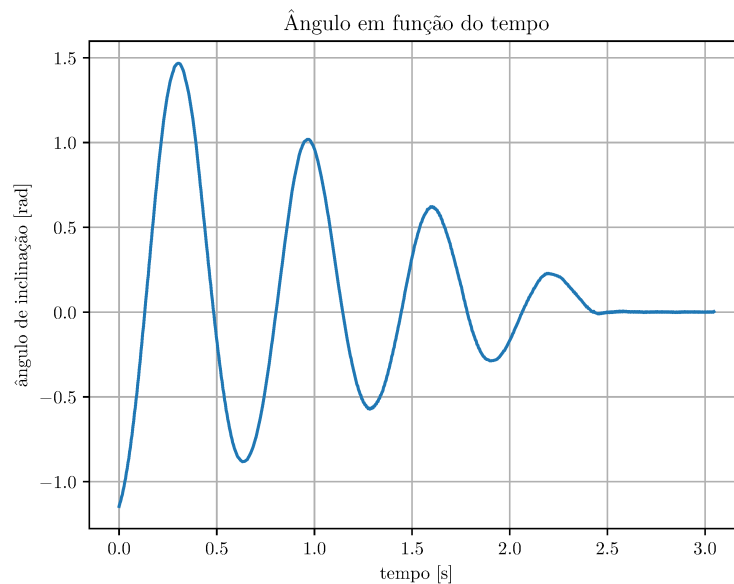


Figura 14 – Dados do procedimento para obter momento de inércia

Para determinar a frequência, foi aplicada a Transformada Rápida de Fourier (FFT), que permite identificar as principais frequências presentes em um sinal. Utilizando as bibliotecas *pandas* e *scipy* em Python, obteve-se o gráfico mostrado na Figura 15. A análise desse gráfico revela que a frequência predominante é de 1,64 Hz. Após 5 repetições deste procedimento, obteve-se a frequência média de 1,543 Hz.

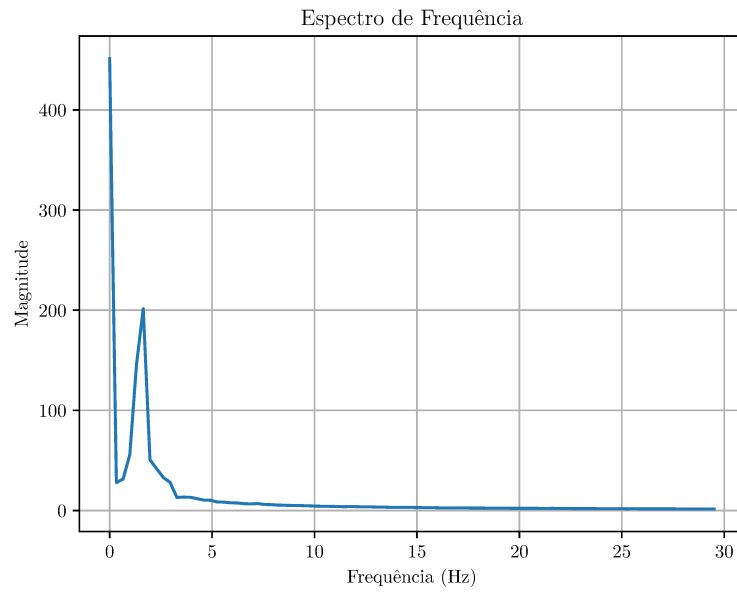


Figura 15 – Transformada rápida de Fourier

O diagrama de corpo livre de um pêndulo simples é mostrado na Figura 11. Seu modelo matemático, descrito pela equação 3.2, pode ser aproximado por um movimento harmônico simples, e sua frequência de oscilação é dada pela equação 3.3, conforme descrito por Coramik e İnanç (2023). Ao combinar essa equação com os valores encontrados para a massa e a posição do centro de massa, obtém-se um valor para o momento de inércia do robô de $66,42 \cdot 10^{-3} \text{ kg.m}^2$. Vale ressaltar que este momento de inércia é em relação ao eixo que passa pelas rodas e que engloba a tanto o corpo quanto a montagem das rodas. Após desconto do momento de inércia das rodas de $24,1 \cdot 10^{-6} \text{ kg.m}^2$ cada, tem-se o valor de $66,3725 \cdot 10^{-3} \text{ kg.m}^2$.

$$\frac{d^2\alpha}{dt^2} = \frac{-mgd}{I}\alpha \quad (3.2)$$

$$\omega = \sqrt{\frac{mgd}{I}} \quad (3.3)$$

Assim, os parâmetros da planta são:

Tabela 1 – Parâmetros estimados da planta

Parâmetro	Descrição	Valor	Unidade
R	Raio das rodas	$63 \cdot 10^{-3}$	m
m_{wa}	Massa da montagem das rodas	$96 \cdot 10^{-3}$	kg
J_w	Momento de inércia da montagem das rodas	$24,1 \cdot 10^{-6}$	m^2kg
L	Distância entre o eixo das rodas e o CM	$18,06 \cdot 10^{-3}$	m
m_b	Massa do corpo	$702 \cdot 10^{-3}$	kg
J_b	Momento de inércia do corpo	$66,37 \cdot 10^{-3}$	m^2kg
g	Aceleração gravitacional	9.81	m/s^2

A partir dos parâmetros estimados e do modelo matemático, é possível estabelecer um ambiente de simulação e realizar o projeto de um controlador para o sistema.

3.3 Simulação no Gazebo

Com o objetivo de simular o sistema, foi desenvolvido um ambiente de teste no software Gazebo, permitindo estimar o comportamento do sistema físico. O Gazebo é um simulador de robótica de código aberto que oferece diversas ferramentas de desenvolvimento, incluindo uma engine física robusta. Isso acelera o processo de criação de um modelo virtual que se aproxime ao máximo do modelo real.

Em conjunto com o Gazebo, utilizou-se o ROS (Robot Operating System), um conjunto de bibliotecas projetado para simplificar o desenvolvimento de sistemas robóticos. Sua aplicação visou a implementação de um algoritmo de controle que pudesse ser utilizado tanto na simulação quanto no modelo físico, reduzindo as discrepâncias entre os dois cenários.

A primeira etapa consistiu na criação da descrição do sistema utilizando o padrão URDF (Universal Robot Description Format). Esse formato permite especificar tanto características visuais, como cor e aparência, quanto propriedades físicas, como colisão, massa e momento de inércia. O código da descrição está apresentado no Apêndice A, enquanto a figura 16 ilustra o modelo do robô.

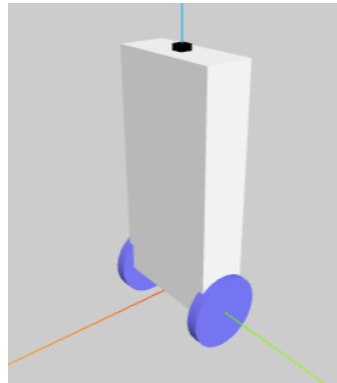


Figura 16 – Visualização descrição do robô

Com a descrição concluída, foi possível integrar o modelo em um ambiente do Gazebo e desenvolver um controlador no ROS. Esse controlador publica as posições desejadas das rodas em um tópico, garantindo que o robô se comporte conforme o esperado.

3.4 Projeto do controlador

Um controlador no espaço de estados possui a seguinte forma:

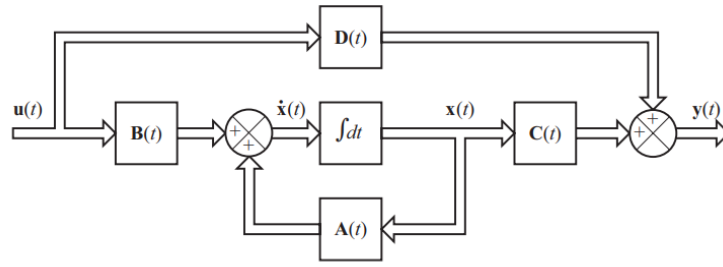


Figura 17 – Controlador no espaço de estados (Fonte: (OGATA, 2010))

Projetar um controlador no espaço de estados consiste no processo de encontrar a matriz de ganho K de tal forma que o sistema possua comportamento desejado. A matriz de ganhos K é um vetor linha mostrado abaixo:

$$K = [k_1 \quad k_2] \quad (3.4)$$

Como o controlador deverá rodar em um microcontrolador, o sistema projetado utiliza da planta discretizada e o projeto será feito no domínio da transformada Z . No caso de sistemas SISO (*Single Input Single Output*), os polos de malha fechada são os os autovalores da matriz $A - BK$. Posicionando os polos em $z_1 = 0,9$ e $z_2 = 0,5$, obtém-se uma resposta satisfatória. Resolvendo K , para que os polos sejam estes, resulta em:

$$K = [-3505,13 \quad -4,13] \quad (3.5)$$

A ação de controle u fica $u = -Kx$, entretanto, os sensores presentes no sistema não são capazes de medir todos os estados x . Faz-se necessária a introdução de um observador de estados que tem por objetivo, a partir a leitura de uma saída y , estimar os valores de x . Um observador de estados segue a seguinte estrutura da figura 18.

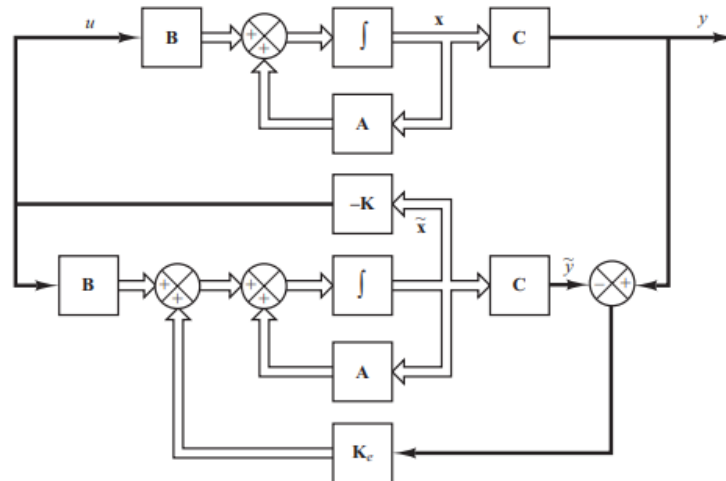


Figura 18 – Observador de estados (Fonte: (OGATA, 2010))

Projetar um observador de estados consiste no processo de encontrar a matriz de ganho K_e de tal forma que o observador possua dinâmica desejada, os polos do observador de estados são dados pelos autovalores da matriz $A - K_e C$. Tipicamente, escolhe-se polos do observador de 3 a 5 vezes mais rápidos do que os polos da planta. Assim, foram escolhidos os polos $z_1 = 0,1$ e $z_2 = 0,18$. O vetor K_e que satisfaz essas condições é dado por:

$$K = \begin{bmatrix} 1,74 \\ 4616,35 \end{bmatrix} \quad (3.6)$$

Feito o projeto do controlador, é necessária implementação do algoritmo de controle e análise dos resultados obtidos.

4 Resultados

4.1 Simulação

Para a simulação inicial do sistema, foi desenvolvido um código em *Python*. A biblioteca *NumPy* foi utilizada para definir vetores e matrizes, enquanto a *SciPy* (módulo *control*) permitiu a modelagem do sistema no espaço de estados e sua discretização com um período de amostragem específico.

A primeira etapa consiste na simulação do sistema em malha aberta. A Figura 19 apresenta a resposta da planta com uma entrada de 0 rad/s. Observa-se que o sistema divergiu, o que era esperado, já que se trata de um sistema instável.

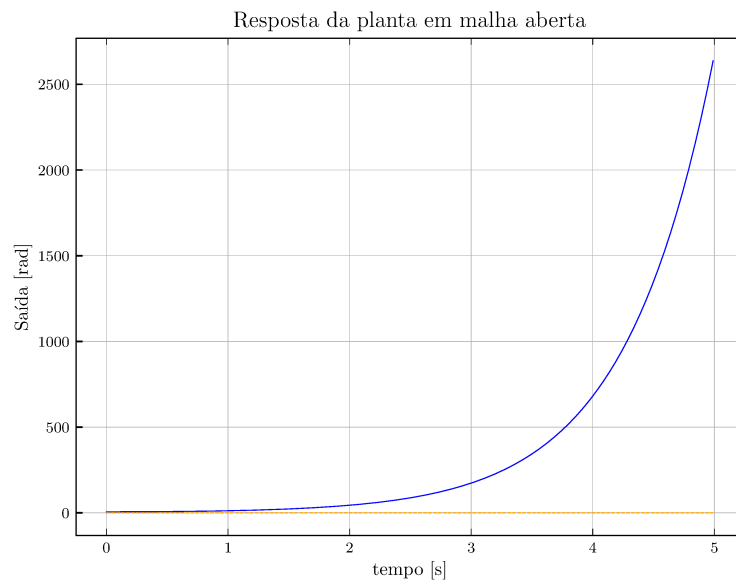


Figura 19 – Simulação da planta em malha aberta

Ao introduzir a simulação de ruído e o controlador no espaço de estados, cujo projeto está descrito na seção anterior, obtém-se a resposta mostrada na figura 20

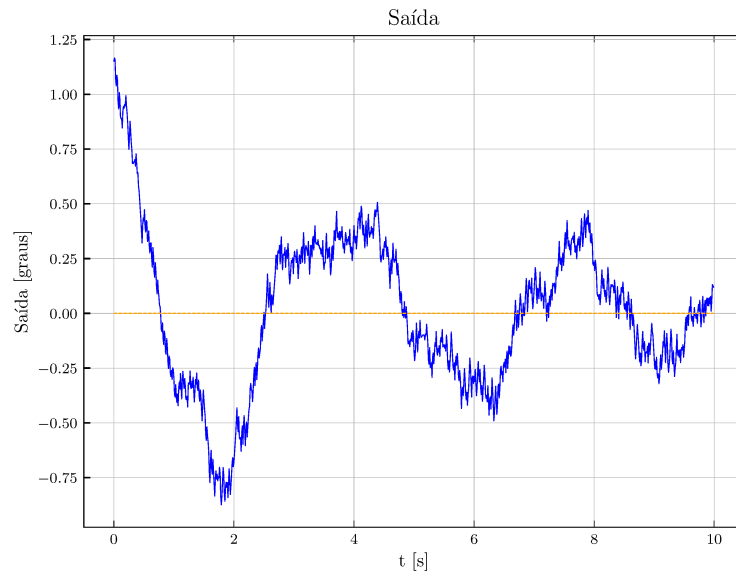


Figura 20 – Simulação da planta em malha fechada

Por fim, utilizando os ganhos do controlador projetado, foi implementado o algoritmo mostrado no apêndice B no sistema físico real. Os resultados são mostrados na figura 21.

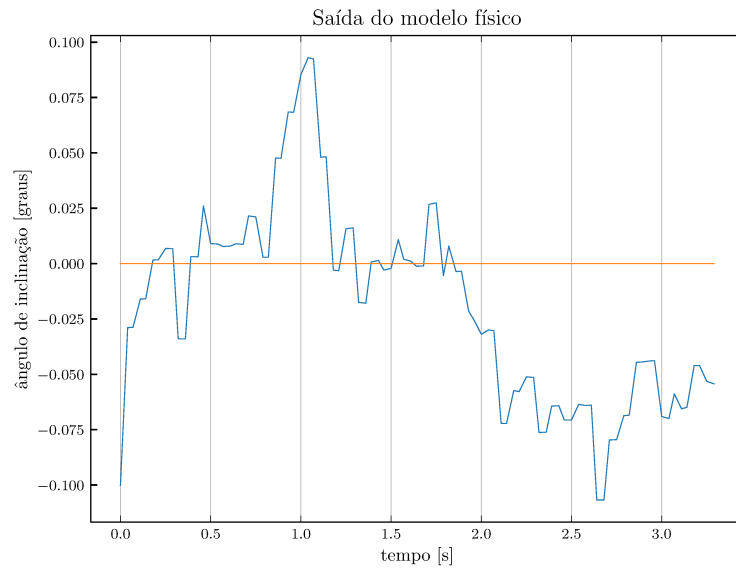


Figura 21 – Saída do modelo físico após implementação do controlador

Os dados mostram que o sistema, anteriormente instável, foi capaz de se estabilizar em malha fechada. Infere-se também que o sistema apresenta oscilação que pode ser atribuída fatores como descasamento de modelo, ruídos de medição e ruídos nos atuadores.

5 Conclusões e sugestões de trabalhos futuros

O problema de construir e controlar um pêndulo invertido sobre duas rodas demonstra-se uma tarefa complexa e desafiadora, principalmente devido à necessidade de conhecimentos em áreas diversas. O processo inicia-se com a construção de um modelo físico que exige o projeto de uma estrutura robusta, capaz de suportar testes variados e de ser modificada conforme os requisitos de controle. Além disso, é necessário projetar uma placa de circuito eletrônico que viabilize a comunicação entre os módulos componentes e forneça tensão e corrente adequadas. É fundamental que estas etapas iniciais sejam realizadas com rigor, pois, o sucesso nessa fase facilita o diagnóstico de falhas em etapas posteriores.

A obtenção dos parâmetros da planta construída é essencial para o projeto do controlador. Os métodos utilizados mostraram-se eficazes ao fornecer estimativas suficientemente próximas dos valores reais para permitir o comportamento desejado do sistema. Enquanto parâmetros como massa e dimensões podem ser medidos diretamente com ferramentas convencionais, outros exigem abordagens indiretas e mais complexas, como a posição do centro de massa e o momento de inércia do corpo e das rodas. A construção de uma bancada experimental revelou-se uma solução prática para determinar tanto o centro de massa quanto o momento de inércia do corpo. Além disso, foi indispensável o uso de técnicas de análise e processamento de sinais para a obtenção dos dados.

Com a planta física definida e os parâmetros determinados, a próxima etapa consistiu em transpor o modelo para o espaço de estados. Esse processo algébrico não foi trivial, já que, na literatura existente, os exemplos predominantes associam os estados às derivadas de posição. A abordagem adotada foi associar um dos estados à posição e o outro a uma grandeza sem significado físico direto. Como consequência, os estados não podiam ser medidos diretamente por sensores, o que tornou necessária a implementação de técnicas para estimativa em tempo real.

Com o sistema descrito na forma de espaço de estados, o projeto do controlador seguiu um caminho direto, envolvendo a escolha dos polos do sistema de acordo com os requisitos de desempenho e a aplicação de passos bem estabelecidos na literatura. Além do controlador, foi necessário projetar um observador de estados para estimar os valores da matriz de estados. Durante essa etapa, verificou-se que ajustes nos parâmetros físicos, como o aumento do raio da roda e a alteração na altura do centro de massa, facilitam a atuação do controlador, entretanto exigem uma reestimação dos parâmetros.

Após a construção do modelo físico e a implementação do controlador, o sistema,

inicialmente instável, foi capaz de se estabilizar em torno do ponto desejado. O algoritmo implementado obteve sucesso, embora com algumas limitações. A primeira delas refere-se à oscilação significativa do modelo, atribuída a fatores como descasamento de modelo, ruídos de medição e ruídos nos atuadores. Os motores de passo demonstraram-se adequados para a aplicação devido à sua alta precisão, mas apresentaram ruídos entre os passos. A segunda limitação é que, embora o sistema consiga se estabilizar em uma posição angular de equilíbrio, sua posição linear não é controlada.

Como sugestões para trabalhos futuros, destacam-se: análise de desempenho com outras técnicas de controle e diferentes tipos de atuadores; implementação de um controlador de posição para ampliar as possíveis aplicações do projeto; e utilização da arquitetura desenvolvida para o controle de múltiplos robôs em atividades coordenadas.

Referências

- CHAN, R. P. M.; STOL, K. A.; HALKYARD, C. R. Review of modelling and control of two-wheeled robots. *Annual Reviews in Control*, v. 37, n. 1, p. 89–103, 2013. ISSN 1367-5788. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S1367578813000060>>.
- CORAMIK, M.; İNANÇ, B. A physical pendulum experiment with lego, phyphox and tracker. *Physics Education*, IOP Publishing, v. 58, n. 5, p. 055014, jul 2023. Disponível em: <<https://dx.doi.org/10.1088/1361-6552/ace57d>>.
- ELETROGATE. *Catálogo Eletrogate*. 2024. <<https://www.eletrogate.com>> [Acessado: (Novembro de 2024)].
- INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Robotics – Vocabulary*. Geneva, Switzerland, 2021. Available at: <<https://www.iso.org/standard/73073.html>>.
- KUNG, F. A tutorial on modelling and control of two- wheeled self-balancing robot with stepper motor. *Applications of Modelling and Simulation*, v. 3, 07 2019.
- LAUSZUS, K. *Kalman Filter*. 2012. Acessado em 08/11/2024. Disponível em: <<https://github.com/TKJElectronics/KalmanFilter>>.
- OGATA, K. *Engenharia de Controle Moderno*. [S.l.]: Pearson Universidades, 2010. ISBN 9788576058106.
- TSAI, L.-W. *Robot Analysis and Design: The Mechanics of Serial and Parallel Manipulators*. 1st. ed. USA: John Wiley & Sons, Inc., 1999. ISBN 0471325937.
- ZEEE. *Catálogo Zeee*. 2024. <<https://zeebattery.com/>> [Acessado: (Novembro de 2024)].

Apêndices

APÊNDICE A – Descrição do robô

Listing A.1 – Descrição do robô

```

<?xml version="1.0" ?>
  <material name="white">
    <color rgba="1_1_1_1"/>
  </material>
  <material name="orange">
    <color rgba="1_0.3_0.1_1"/>
  </material>
  <material name="blue">
    <color rgba="0.2_0.2_1_1"/>
  </material>
  <material name="black">
    <color rgba="0_0_0_1"/>
  </material>
  <link name="base_link">

    </link>
  <!-- BASE LINK -->
  <joint name="chassis_joint" type="fixed">
    <parent link="base_link"/>
    <child link="chassis"/>
    <origin xyz="0_0_0"/>
  </joint>
  <link name="chassis">
    <visual>
      <origin xyz="0_0_0.1"/>
      <geometry>
        <box size="0.04_0.1_0.2"/>
      </geometry>
      <material name="white"/>
    </visual>
    <collision>
      <origin xyz="0_0_0.1"/>
      <geometry>

```

```

    <box size="0.04 0.1 0.2" />
  </geometry>
</collision>
<inertial>
  <origin rpy="0 0 0" xyz="0 0 0.04" />
  <mass value="0.66" />
  <inertia
    ixx="0.0001304"
    ixy="0.0"
    ixz="0.0"
    iyy="0.001304"
    iyz="0.0"
    izz="0.0001304" />
</inertial>
</link>
<gazebo reference="chassis">
  <material>Gazebo/White</material>
  <mu1 value="0.001" />
  <mu2 value="0.001" />
</gazebo>
<!-- LEFT WHEEL -->
<joint name="left_wheel_joint" type="continuous">
  <parent link="base_link" />
  <child link="left_wheel" />
  <origin rpy="-1.5707963267948966 0 0" xyz="0 0.05 0" />
  <axis xyz="0 0 1" />
</joint>
<link name="left_wheel">
  <visual>
    <geometry>
      <cylinder length="0.01" radius="0.0315" />
    </geometry>
    <material name="blue" />
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.01" radius="0.0315" />
    </geometry>
  </collision>

```

```

    <inertial>
      <origin rpy="0 0 0" xyz="0 0 0" />
      <mass value="0.104" />
      <inertia
        ixx="2.5852666666666664e-05"
        ixy="0.0"
        ixz="0.0"
        iyy="2.5852666666666664e-05"
        iyz="0.0"
        izz="4.9971999999999995e-05" />
    </inertial>
  </link>
  <gazebo reference="left_wheel">
    <material>Gazebo/Blue</material>
  </gazebo>
  <!-- RIGHT WHEEL -->
  <joint name="right_wheel_joint" type="continuous">
    <parent link="base_link" />
    <child link="right_wheel" />
    <origin rpy="1.5707963267948966 0 0" xyz="0 -0.05 0" />
    <axis xyz="0 0 1" />
  </joint>
  <link name="right_wheel">
    <visual>
      <geometry>
        <cylinder length="0.01" radius="0.0315" />
      </geometry>
      <material name="blue" />
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.01" radius="0.0315" />
      </geometry>
    </collision>
    <inertial>
      <origin rpy="0 0 0" xyz="0 0 0" />
      <mass value="0.104" />
      <inertia
        ixx="2.5852666666666664e-05"

```

```

        ixy=" 0.0 "
        ixz=" 0.0 "
        iyy=" 2.5852666666666664e-05 "
        iyz=" 0.0 "
        izz=" 4.9971999999999995e-05 "/>
    </inertial>
</link>
<gazebo reference="right_wheel">
    <material>Gazebo/Blue</material>
</gazebo>
<ros2_control name="GazeboSystem" type="system">
    <hardware>
        <plugin>gazebo_ros2_control/GazeboSystem</plugin>
    </hardware>
    <joint name="left_wheel_joint">
        <command_interface name="velocity">
            <param name="min">-10</param>
            <param name="max">10</param>
        </command_interface>
        <state_interface name="velocity"/>
        <state_interface name="position"/>
    </joint>
    <joint name="right_wheel_joint">
        <command_interface name="velocity">
            <param name="min">-10</param>
            <param name="max">10</param>
        </command_interface>
        <state_interface name="velocity"/>
        <state_interface name="position"/>
    </joint>
</ros2_control>
<gazebo>
    <plugin
        filename="libgazebo_ros2_control.so"
        name="gazebo_ros2_control">
        <parameters>
            path/to/parameters
        </parameters>
    </plugin>

```

```

</gazebo>
<joint name="imu_joint" type="fixed">
  <parent link="chassis" />
  <child link="imu_frame" />
  <origin rpy="0 0 0" xyz="0 0 0.2" />
</joint>
<link name="imu_frame">
  <visual>
    <origin xyz="0 0 0" />
    <geometry>
      <box size="0.01 0.01 0.01" />
    </geometry>
    <material name="black" />
  </visual>
</link>
<gazebo reference="imu_frame">
  <material>Gazebo/Black</material>
  <gravity>true</gravity>
  <sensor name="imu_sensor" type="imu">
    <always_on>true</always_on>
    <update_rate>100</update_rate>
    <visualize>true</visualize>
    <topic>__default_topic__</topic>
    <plugin
      filename="libgazebo_ros_imu_sensor.so"
      name="imu_plugin">
      <topicName>imu</topicName>
      <bodyName>imu_link</bodyName>
      <updateRateHZ>1.0</updateRateHZ>
      <gaussianNoise>0.0</gaussianNoise>
      <xyzOffset>0 0 0</xyzOffset>
      <rpyOffset>0 0 0</rpyOffset>
      <frameName>imu_link</frameName>
    </plugin>
    <pose>0 0 0 0 0 0</pose>
  </sensor>
</gazebo>
</robot>

```

APÊNDICE B – Código do robô

Listing B.1 – Código do robô

```
#include <Wire.h>

// Source: https://github.com/TKJElectronics/KalmanFilter
#include <Kalman.h>

#include <AccelStepper.h>

// Define a stepper and the pins it will use
#define STEP_LEFT 5
#define DIR_LEFT 4
#define STEP_RIGHT 3
#define DIR_RIGHT 2

#define RESTRICT_PITCH

AccelStepper stepper(AccelStepper::DRIVER, STEP_LEFT, DIR_LEFT);
AccelStepper stepper2(AccelStepper::DRIVER, STEP_RIGHT, DIR_RIGHT);

Kalman kalmanX; // Create the Kalman instances
Kalman kalmanY;

/* IMU Data */
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int16_t tempRaw;

double gyroXangle, gyroYangle;
double compAngleX, compAngleY;
double kalAngleX, kalAngleY;

uint32_t timer;

uint8_t i2cData[14]; // Buffer for I2C data

double offset = -4.75;
```

```
double Ta = 10;

double u = 0;
double maxu = 31.42;
double minu = -maxu;

double a11 = 1.00009359;
double a12 = 0.0000410839667;
double a21 = 4.55361788;
double a22 = 1.00009359;

double x_c1 = 0;
double x_c2 = 0;

double xp_c1 = 0;
double xp_c2 = 0;

double b1 = -0.0000410839667;
double b2 = -0.0000935359681;

double Kep1 = 1.72018707;
double Kep2 = 17971.6817;

double K1 = -13998.00414405;
double K2 = -268.27686159;

double y = 0;

double c1 = 1;
double c2 = 0;

void setup() {
    Serial.begin(115200);
    stepper.setMaxSpeed(1000);
    stepper2.setMaxSpeed(1000);
    Wire.begin();
    #if ARDUINO >= 157
        Wire.setClock(400000UL); // Set I2C frequency to 400kHz
```

```

#else

    TWBR = ((F_CPU / 400000UL) - 16) / 2; // Set I2C frequency to 400kHz
#endif

    i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
    // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz Gyro filtering, 8 KHz sampling
    i2cData[1] = 0x00;
    i2cData[2] = 0x00;
    i2cData[3] = 0x00;

    while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers at once
    // PLL with X axis gyroscope reference and disable sleep mode

    while (i2cWrite(0x6B, 0x01, true));

    while (i2cRead(0x75, i2cData, 1));
    if (i2cData[0] != 0x68) { // Read "WHO_AM_I" register
        Serial.print(F("Error reading sensor"));
        while (1);
    }

    delay(100); // Wait for sensor to stabilize

    /* Set kalman and gyro starting angle */
    while (i2cRead(0x3B, i2cData, 6));
    accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
    accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
    accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);

    // Source: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf
    //eq. 25 and eq. 26
    // It is then converted from radians to degrees
#ifndef RESTRICT_PITCH // Eq. 25 and 26
        double roll = atan2(accY, accZ) * RAD_TO_DEG;
        double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
        double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
        double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif

```

```
    kalmanX.setAngle(roll); // Set starting angle
    kalmanY.setAngle(pitch);

    gyroXangle = roll;
    gyroYangle = pitch;
    compAngleX = roll;
    compAngleY = pitch;

    timer = micros();
}

void applyU(double u){
    double radius = 0.063;
    double w = u / radius*0.000001014;
    double w_rpm = w*9.5493;

    double ratio = 31.957699531;
    double speed = ratio*w;
    Serial.print("\uSpeed:\u");
    Serial.print(speed);
    Serial.print("\u\uw_rpm:\u");
    Serial.print(w_rpm);

    stepper.setSpeed(-speed);
    stepper2.setSpeed(speed);
}

void runSteppers(){
    stepper.runSpeed();
    stepper2.runSpeed();
}

void loop() {

    unsigned long now = millis();

    applyU(u);
    runSteppers();
    /* Update all the values */
```

```

while (i2cRead(0x3B, i2cData, 14));
accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);
tempRaw = (int16_t)((i2cData[6] << 8) | i2cData[7]);
gyroX = (int16_t)((i2cData[8] << 8) | i2cData[9]);
gyroY = (int16_t)((i2cData[10] << 8) | i2cData[11]);
gyroZ = (int16_t)((i2cData[12] << 8) | i2cData[13]);

// Calculate delta time
double dt = (double)(micros() - timer) / 1000000;
timer = micros();

// It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
    double roll = atan2(accY, accZ) * RAD_TO_DEG;
    double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
    double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
    double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif

double gyroXrate = gyroX / 131.0; // Convert to deg/s
double gyroYrate = gyroY / 131.0; // Convert to deg/s

#ifdef RESTRICT_PITCH
    // This fixes the transition problem
    //when the accelerometer angle jumps
    //between -180 and 180 degrees
    if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {
        kalmanX.setAngle(roll);
        compAngleX = roll;
        kalAngleX = roll;
        gyroXangle = roll;
    } else
    // Calculate the angle using a Kalman filter
        kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt);

```

```

    if (abs(kalAngleX) > 90)
        // Invert rate, so it fits the restriced accelerometer reading
        gyroYrate = -gyroYrate;
        kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
#else
    // This fixes the transition problem
    //when the accelerometer angle jumps between -180 and 180 degrees
    if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {
        kalmanY.setAngle(pitch);
        compAngleY = pitch;
        kalAngleY = pitch;
        gyroYangle = pitch;
    } else
        // Calculate the angle using a Kalman filter
        kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);

    if (abs(kalAngleY) > 90)
        // Invert rate, so it fits the restriced accelerometer reading
        gyroXrate = -gyroXrate;
        // Calculate the angle using a Kalman filter
        kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt);
#endif

    gyroXangle += gyroXrate * dt; // Calculate gyro angle without any filter
    gyroYangle += gyroYrate * dt;
    // Calculate gyro angle using the unbiased rate
    //gyroXangle += kalmanX.getRate() * dt;
    //gyroYangle += kalmanY.getRate() * dt;

    // Calculate the angle using a Complimentary filter
    compAngleX = 0.93 * (compAngleX + gyroXrate * dt) + 0.07 * roll;
    compAngleY = 0.93 * (compAngleY + gyroYrate * dt) + 0.07 * pitch;

    // Reset the gyro angle when it has drifted too much
    if (gyroXangle < -180 || gyroXangle > 180)
        gyroXangle = kalAngleX;
    if (gyroYangle < -180 || gyroYangle > 180)
        gyroYangle = kalAngleY;

```

```

    /* Print Data */
#ifdef 0 // Set to 1 to activate
    Serial.print(accX); Serial.print("\t");
    Serial.print(accY); Serial.print("\t");
    Serial.print(accZ); Serial.print("\t");

    Serial.print(gyroX); Serial.print("\t");
    Serial.print(gyroY); Serial.print("\t");
    Serial.print(gyroZ); Serial.print("\t");

    Serial.print("\t");
#endif

    Serial.print(roll); Serial.print("\t");
    Serial.print(gyroXangle); Serial.print("\t");
    Serial.print(compAngleX); Serial.print("\t");
    Serial.print(kalAngleX); Serial.print("\t");

    Serial.print("\t");

    Serial.print(pitch); Serial.print("\t");
    Serial.print(gyroYangle); Serial.print("\t");
    Serial.print(compAngleY); Serial.print("\t");
    Serial.print(kalAngleY); Serial.print("\t");

#ifdef 0 // Set to 1 to print the temperature
    Serial.print("\t");

    double temperature = (double)tempRaw / 340.0 + 36.53;
    Serial.print(temperature); Serial.print("\t");
#endif

    y = (kalAngleY - offset)*PI;
    Serial.print("␣Y␣:␣");
    Serial.print(y);
    y = y/180.0;

```

```
xp_c1 = a11*x_c1 + a12*x_c2;
xp_c2 = a21*x_c1 + a22*x_c2;

xp_c1+= b1*u;
xp_c2+= b2*u;

xp_c1+= Kep1*(y - x_c1*c1 - x_c2*c2);
xp_c2+= Kep2*(y - x_c1*c1 - x_c2*c2);

u = -K1*xp_c1 - K2*xp_c2;

Serial.print("maxu:");
Serial.print(maxu);

Serial.print("\r\n");

while ( millis() - now < Ta) {
    runSteppers();
}
// delay(100);
}
```