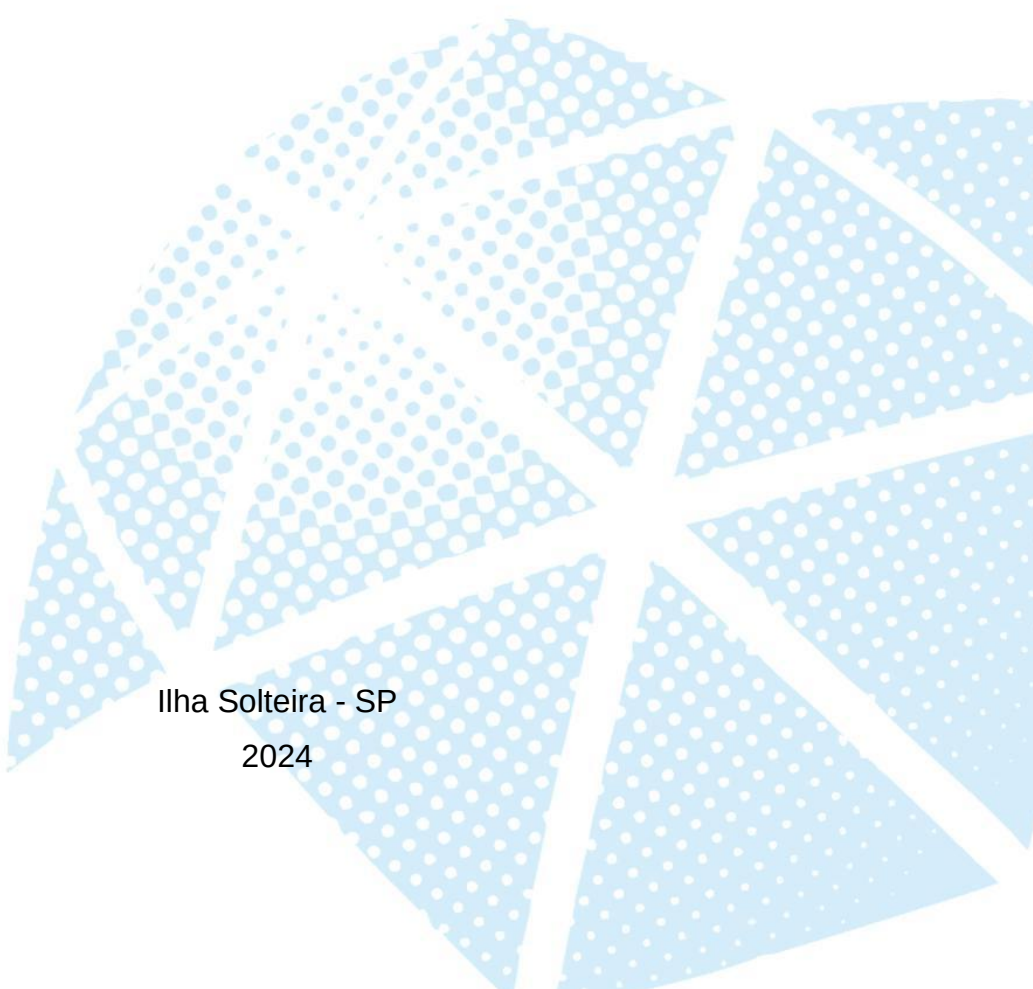


**UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”
CÂMPUS DE ILHA SOLTEIRA**

ARTHUR YUGO YAMANAKA

**CONSTRUÇÃO E CONTROLE PID DISCRETO APLICADO A UM SISTEMA ROBÔ
PÊNDULO INVERTIDO**

Ilha Solteira - SP
2024



ARTHUR YUGO YAMANAKA

**CONSTRUÇÃO E CONTROLE PID DISCRETO APLICADO A UM SISTEMA ROBÔ
PÊNDULO INVERTIDO**

Trabalho de Conclusão de Curso
apresentada à Universidade Estadual
Paulista (UNESP), Ilha Solteira, para
obtenção do título de Grau acadêmico de
Bacharel em Engenharia Elétrica.

Área de Concentração: Controle linear

Orientador(a): Prof. Dr. Marcelo Carvalho
Minhoto Teixeira

Coorientador(a): Eng. Hugo Fernando
Yamanaka

Ilha Solteira - SP

2024

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

Y19c Yamanaka, Arthur Yugo.
Construção e controle PID discreto aplicado a um sistema robô pêndulo invertido / Arthur Yugo Yamanaka. -- Ilha Solteira: [s.n.], 2024
75 f. : il.

Trabalho de conclusão de curso - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Engenharia Elétrica, 2024

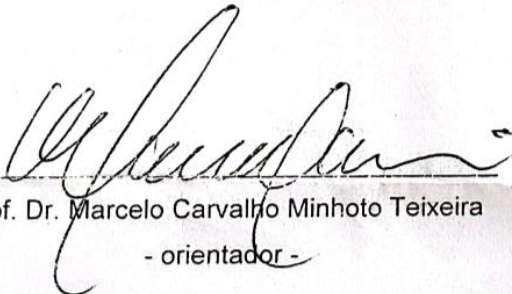
Orientador: Marcelo Carvalho Minhoto Teixeira
Co-orientador: Hugo Fernando Yamanaka
Inclui bibliografia

1. Baixo Custo. 2. Controle PID discreto. 3. Robô pêndulo invertido.

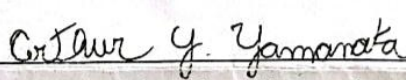

Amanda Sertori dos Santos
Biblioteca - CRB 3061
Setor Técnico de Referência, Atendimento ao
Usuário e Documentação
Departamento Técnico de Biblioteca e Documentação

ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO

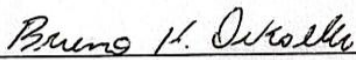
Aos dezessete dias do mês de dezembro do ano de dois mil e vinte e quatro, o discente **Arthur Yugo Yamanaka**, matriculado sob o nº 162053461, tendo como banca examinadora o seu orientador, o *Prof. Dr. Marcelo Carvalho Minhoto Teixeira*, o *Eng. Bruno Hiromoto Oikawa* e o *Mestrando Hugo Fernando Yamanaka* apresentou o Trabalho de Graduação intitulado "**Construção e Controle PID discreto aplicado a um sistema Robô Pêndulo Invertido**", obtendo a nota 10,0 (dez inteiros) e conceito APROVADO.


Prof. Dr. Marcelo Carvalho Minhoto Teixeira

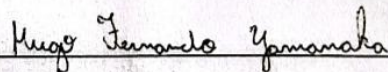
- orientador -


Arthur Yugo Yamanaka

- discente -


Eng. Bruno Hiromoto Oikawa

- Membro da Banca -


Mestrando Hugo Fernando Yamanaka

- Membro da Banca -

Dedico este trabalho aos meus pais que sempre serão minha inspiração.

AGRADECIMENTOS

Aos meus pais, que sempre fizeram de tudo para garantir minha educação e pelos valores éticos e morais que me foram passados. Sem isso, não seria nada.

Aos meus professores, sobretudo o professor Marcelo e ao sr. Hugo Yamanaka que me deram a oportunidade de realizar esse trabalho, permitindo que eu pudesse aprender e desenvolver meus conhecimentos na área de controle, a qual tenho enorme interesse.

Aos meus amigos que sempre me ajudaram e aconselharam durante todo o período da graduação.

À minha namorada, que sem ela não teria chegado até aqui, pois foi a melhor companheira e incentivadora que poderia ter.

“Não importa quanto a vida possa ser ruim, sempre existe algo que você pode fazer, e triunfar. Enquanto há vida, há esperança”

(Stephen Hawking).

RESUMO

O presente trabalho aborda a construção de um robô pêndulo invertido, cujo sistema é de grande interesse na área de controle devido à sua natureza intrinsecamente instável. O objetivo principal é a construção de um robô pêndulo invertido de baixo custo e, utilizando um modelo matemático desse sistema, projetar um controlador de tempo discreto que permita com que robô opere de forma estável. A metodologia utilizada foi a elaboração do robô que foi montado de acordo com os materiais listados neste trabalho, que no caso foram os motores, componentes eletrônicos e peças auxiliares. Além disso, foram utilizados recursos de *softwares* para o desenho do circuito elétrico, programação do robô e para a análise dos dados. Com o robô finalizado, foram realizados os cálculos dos parâmetros utilizados no modelo para encontrar o modelo da planta real em tempo discreto. Por fim, com auxílio do Matlab, foram encontrados os ganhos do controlador PID e implementados, cujo resultado obtido permitiu um funcionamento estável do sistema em malha fechada.

Palavras-chave: Controle PID discreto; Robô Pêndulo Invertido; baixo custo.

ABSTRACT

This work is focused on building a low-cost inverted pendulum robot. This system is of enormous interest on the control area due to its instable identity. The main goal is to build a low-cost inverted pendulum robot and, using a mathematical model of this system, to project a discrete-time controller that allows an adequate operation of the robot. The methodology used was the elaboration of the robot, based on the mathematical model, which montage was according to the materials listed in this work, which are the motors, the electronic components and auxiliary parts. Also, it were used some softwares for drawing the electrical circuit, to program the robot and to make the analysis of the robot data. With the robot finished, the values of the parameters of the model were found through calculations to find the discrete-time real plant. Then, the PID controller gains were found with the support of Matlab and these gains implemented, resulting in a stable functioning of the system.

Keywords: Discrete PID Control; Inverted Pendulum Robot; Low-cost.

LISTA DE FIGURAS

Figura 1- Robô Pêndulo Invertido.....	17
Figura 2 - Pêndulo Invertido em um carrinho acionado por motor.....	18
Figura 3 - Modelo elétrico do motor DC.....	20
Figura 4- Nomenclaturas e reações da roda.....	22
Figura 5- Corpo do robô pêndulo invertido.....	23
Figura 6 - Controle de malha fechada do robô.....	27
Figura 7 - Espaço de estado contínuo para função de transferência discreta.....	28
Figura 8 - Gráfico do sinal discretizado.....	29
Figura 9 - Sinais de tensão por largura de pulso.....	31
Figura 10 - Transferência de bits no barramento I ² C.....	32
Figura 11 - Imagem 3D frontal da placa	34
Figura 12 - Imagem 3D posterior da placa	35
Figura 13 - ESP32-DevKitC V4 com o módulo ESP32-WROOM-32.....	36
Figura 14 - Esquema de pinagem do ESP32-DevKitC V4.....	38
Figura 15 - Dimensões da placa ESP32-DevKitC V4.....	38
Figura 16 - Driver motor L298N	39
Figura 17 - Esquemático da ponte H L298N.....	40
Figura 18 - Sensor MPU6050.....	41
Figura 19 - Motor DC modelo JGA25-370.....	43
Figura 20 - Fios do encoder do motor JGA25-370.....	43
Figura 21 - Funcionamento do robô pêndulo invertido.....	44
Figura 22 - Interface do software Arduino IDE.....	46
Figura 23 - Tipos de controlador PID TUNER.....	47
Figura 24 - Opções do controlador no PID TUNER.....	48
Figura 25 - Ajustes parâmetros da resposta no PID TUNER.....	49
Figura 26 - Imagem frontal do robô.....	51
Figura 27 - Imagem posterior do robô.....	51
Figura 28 - Menu de opções do controlador.....	53
Figura 29 - Obtenção dos ganhos utilizando o PID TUNER.....	54
Figura 30 - Modelo implementado no Simulink.....	54
Figura 31 - Resposta a entrada degrau do modelo do Simulink.....	55
Figura 32 - Resultados para entrada degrau.....	55

Figura 33 - Aquisição dos parâmetros do modelo físico no MATLAB.....	56
Figura 34 - Entrada (Ângulo de Pitch) x Referência.....	57
Figura 35 - Tensão de saída (Resposta) x Amostras	57
Figura 36 - Entrada (ângulo pitch) x Resposta (Tensão de saída)	58
Figura 37 - Robô pêndulo invertido em funcionamento.....	58

LISTA DE TABELAS

Tabela 1- Parâmetros do modelo elétrico do motor DC.....	21
Tabela 2- Parâmetros relacionados às rodas	22
Tabela 3- Parâmetros do corpo do robô.....	23
Tabela 4- Descrição dos componentes principais do ESP32-DevKitC V4.....	37
Tabela 5 - Dados de especificação do L298N.....	39
Tabela 6- Especificações do MPU6050.....	41
Tabela 7- Especificações do motor JGA25-370.....	42
Tabela 8 - Lista de valores dos componentes e peças.....	50
Tabela 9 - Parâmetros medidos do robô.....	52

LISTA DE ABREVIATURAS E SIGLAS

CI	Circuito Integrado
DC	<i>Direct Current</i>
GPIO	<i>General Purpose Input/Output</i>
IDE	<i>Integrated Development Enviroment</i>
I ² C	<i>Inter-Integrated Circuit</i>
LED	<i>Light Emitter Diode</i>
LQR	<i>Linear Quadratic Regulator</i>
PCI	Placa de Circuito Impresso
PID	<i>Proporcional Integral Derivativo</i>
PPR	Pulsos Por Rotação
PWM	<i>Pulse Width Modulation</i>
SCL	<i>Serial Clock</i>
SDA	<i>Serial Data</i>
SPI	<i>Serial Peripheral Interface</i>
UART	<i>Universal Assincronous Receiver/Transmitter</i>

LISTA DE SÍMBOLOS

V	Tensão do motor
T_m	Torque do motor
R	Resistência do motor
K_t	Constante de torque
K_b	Constante de força eletromotriz
L_a	Indutância
i	Corrente no estator do motor
u_{brush}	Tensão da Escova
I_m	Momento de inércia do rotor do motor
ξ	Rotação do motor em relação ao estator
b_m	Fricção viscosa do rotor
F_p	Força do corpo do pêndulo em relação a roda
T_f	Torque de atrito proporcional à velocidade angular do motor
F	Força horizontal no eixo do motor
F_s	Força entre a roda e a superfície
m_w	Massa da roda
r_w	Raio da roda
θ	Ângulo da roda
I_w	Momento de inércia no centro de massa da roda
P	Centro de massa do pêndulo
α	Ângulo de inclinação
L	Distância do eixo da roda até o ponto P
m_p	Massa do pêndulo
I_p	Momento de inércia em P
x	Distância que a roda percorreu
C	Eixo da roda
K_i	Ganho Integral
u	Vetor de entrada
y	Vetor de saída
K_d	Ganho Derivativo
K_p	Ganho Proporcional
D	Duty Cycle

Ton Tempo do sinal em nível alto
Toff Tempo do sinal em nível baixo
e Erro entre valor de entrada e saída
IF(z) Fórmula integral discreta
DF(z) Fórmula derivativa discreta
g Aceleração da gravidade
Vpeak Tensão de pico
Snoload Velocidade de rotação do motor sem carga

SUMÁRIO

1	INTRODUÇÃO	17
1.1	Sistema pêndulo invertido	18
1.2	Objetivo	19
2	INTRODUÇÃO TEÓRICA.....	20
2.1	Modelagem do pêndulo invertido	20
2.2	Controle PID	26
2.3	Obtenção da função de transferência discreta	28
2.4	PWM.....	31
2.5	Barramento I ² C	32
3	CONSTRUÇÃO DO ROBÔ PÊNDULO INVERTIDO	33
3.1	ESP-32 DEVKITC V4.....	36
3.2	Ponte-H L298N.....	39
3.3	MPU6050.....	41
3.4	Motores DC.	42
4	PROGRAMAÇÃO DO ROBÔ.....	44
4.1	Arduino IDE	45
4.2	Matlab PID TUNER	47
5	RESULTADOS	50
6	CONSIDERAÇÕES FINAIS.....	59
	REFERÊNCIAS.....	60
	APÊNDICE A - Código para obtenção do valor PID.....	65
	APÊNDICE B - Código respostas entrada degrau.....	67
	APÊNDICE C - Código da programação do robô.....	69
	APÊNDICE D - Código de aquisição de dados no Matlab.....	74
	APÊNDICE E – Esquemático do circuito do robô pêndulo invertido.....	76

1 INTRODUÇÃO

O sistema pêndulo invertido é um sistema muito conhecido na área de controle e robótica que usa um controle por realimentação para estabilizar um sistema de malha aberta instável. De acordo com Kafetzis e Moysis (2017), esse sistema tem sido implementado em diversos campos da indústria, como por exemplo: veículos autônomos, drones, braços robóticos, e até em sistemas de armazenamento de energia.

Utilizando deste sistema, o robô equilibrista, vulgo robô pêndulo invertido, foi o foco de implementação deste sistema neste trabalho. A Figura 1 apresenta um exemplo de robô pêndulo invertido.

Figura 1 - Robô Pêndulo Invertido.



Fonte: Terasic (2019).

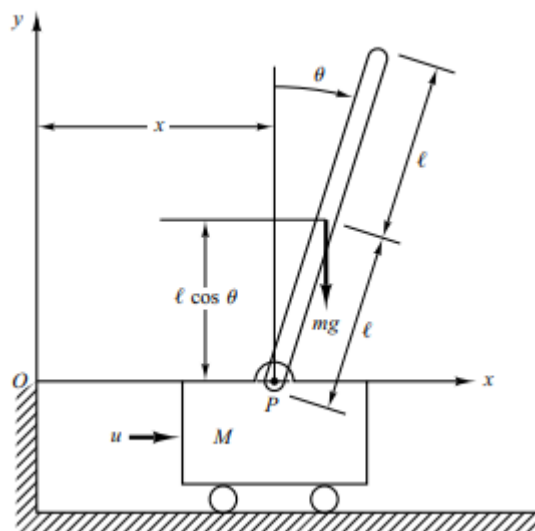
1.1 Sistema pêndulo invertido

Segundo Lundberg e Barton (2019), o sistema de pêndulo invertido é muito utilizado para fins de aprendizado sobre física, dinâmica e controle. Tal sistema é um sistema mecânico instável utilizado em diversos problemas de controle. O principal desafio, do ponto de vista da área de controle é devido à característica não linear e ao ponto de equilíbrio instável.

Alguns exemplos de aplicação desse sistema são: máquinas de transporte que exigem o equilíbrio de objetos, sistemas que auxiliam pacientes a andar, drones e até mesmo o uso desse sistema na construção de prédios. Para o controle desse tipo de sistema, são utilizados diversos métodos como controle PID, controle preditivo e controle *fuzzy* (Qi, 2022).

Um dos casos em que se utiliza esse sistema, similar ao caso aplicado a esse trabalho, considera a configuração clássica do pêndulo invertido montado em um carrinho acionado por motor. A Figura 2 mostra uma representação do modelo desse caso citado.

Figura 2 - Pêndulo invertido em um carrinho acionado por motor.



Fonte: Ogata (2010).

1.2 Objetivo

O objetivo desse trabalho é construir um robô pêndulo invertido de baixo custo utilizando materiais encontrados no mercado com facilidade e, a partir de um modelo matemático do sistema, projetar um controlador em tempo discreto que permita com que robô opere de forma estável em malha fechada.

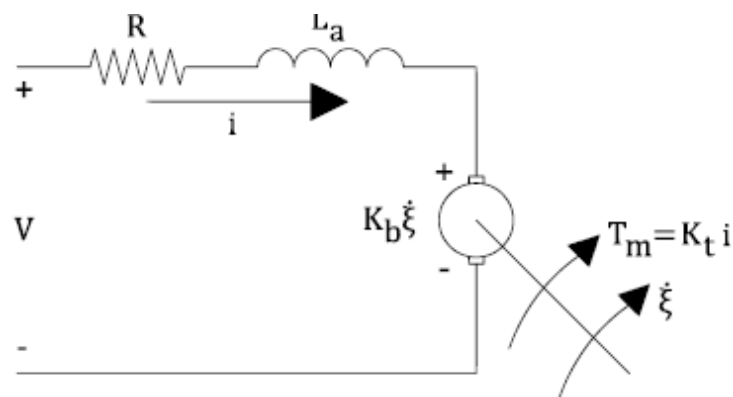
2 INTRODUÇÃO TEÓRICA

Para este trabalho, foram necessários os estudos sobre a modelagem do pêndulo invertido, com o propósito de projetar o controlador PID e obter os ganhos que serão implementados no sistema real, sobre o método de controle PID, o método PWM, a comunicação I²C, para comunicação entre o sensor acelerômetro/giroscópio, o microcontrolador ESP-DevKitC V4, e a transformação de espaço de estados em tempo contínuo para tempo discreto, que serão abordados a seguir. Essa transformação ocorre para conseguirmos projetar o controle em tempo discreto. No entanto, para simular o sistema, é necessário que ele esteja em tempo contínuo e o controlador em tempo discreto.

2.1 Modelagem do pêndulo invertido

O robô pêndulo invertido possui diversas representações na literatura com diferentes aplicações, como já citado. Como neste trabalho trata-se de um robô pêndulo invertido muito similar com o Segway, o modelo matemático adotado foi o modelo descrito por Howard e Bushnell (2015). Portanto, deve-se considerar, primeiramente, o motor elétrico responsável pelo movimento do robô, cujo diagrama está apresentado na Figura 3.

Figura 3- Modelo elétrico do motor DC.



Fonte: Howard e Bushnell (2015).

Baseado na Figura 3, a Tabela 1 apresenta os parâmetros que foram levados em conta para o desenvolvimento do modelo do motor.

Tabela 1- Parâmetros do modelo elétrico do motor DC.

Símbolo	Descrição
V	Tensão do motor
Tm	Torque do motor
R	Resistência do motor
Kt	Constante de torque
Kb	Constante de força eletromotriz
La	Indutância
i	Corrente no estator do motor
$vbrush$	Tensão da escova
Im	Momento de inércia do rotor do motor
ξ	Rotação do motor em relação ao estator
bm	Fricção viscosa do rotor

Fonte: Adaptado de Howard e Bushnell (2015).

Analizando o modelo da Figura 3, é possível relacionar a tensão aplicada ao torque desenvolvido através de uma equação da seguinte forma:

$$V = La * \frac{di}{dt} + R * i + Kb * \dot{\xi} + 2 * vbrush . \quad (1)$$

Assim como a tensão, a corrente também pode ser relacionada como:

$$Im * \ddot{\xi} = Kt * i - Tm - bm * \dot{\xi} . \quad (2)$$

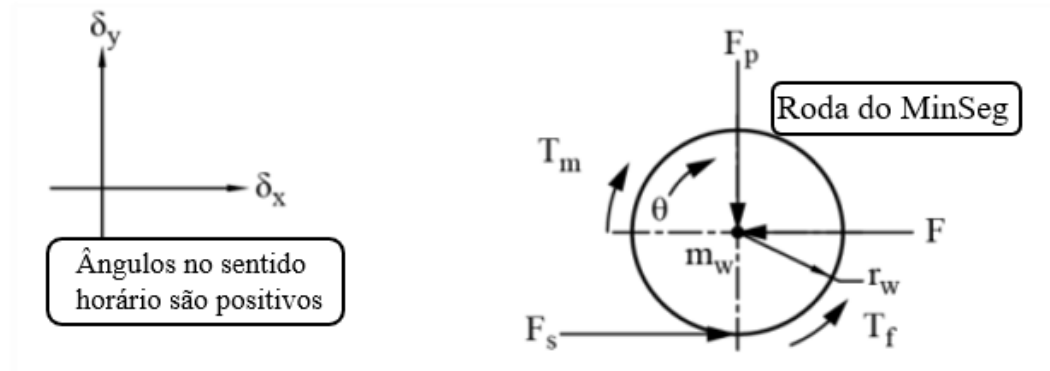
Ao assumirmos que a resposta do sistema elétrico é muito mais rápida que do sistema mecânico, a contribuição da indutância para o circuito pode ser desprezada. Assim como a indutância, também é possível considerar irrisórios os efeitos da tensão da escova e da fricção viscosa.

Ao combinar as duas equações, notando que o ângulo de rotação de armadura é a diferença entre o ângulo do corpo do robô, α , e o ângulo de rotação da roda, θ , o torque pode ser descrito como:

$$Tm = \frac{Kt * V - Kb * Kt * (-\dot{\alpha} + \dot{\theta})}{R} . \quad (3)$$

Os motores movem as rodas e as rodas conectam o robô ao solo. A Figura 4 mostra a nomenclatura e os parâmetros relacionados à roda, cujas descrições estão citadas na Tabela 2.

Figura 4- Nomenclaturas e reações da roda.



Fonte: Adaptado de Howard e Bushnell (2015).

Tabela 2- Parâmetros relacionados às rodas

Símbolo	Descrição
F_p	Força do corpo do pêndulo em relação a roda
T_m	Torque do motor
T_f	Torque de atrito proporcional à velocidade angular do motor
F	Força horizontal no eixo do motor
F_s	Força entre a roda e a superfície
m_w	Massa da roda
r_w	Raio da roda
θ	Ângulo da roda
I_w	Momento de inércia no centro de massa da roda

Fonte: Adaptado de Howard e Bushnell (2015).

As equações das rodas possuem componentes rotacionais e translacionais que podem ser escritas de forma separada. Somando os torques sobre o eixo da roda para derivar as equações de movimento rotacionais, a equação resultante é:

$$I_w * \ddot{\theta} = -F_s * r_w - T_f + T_m. \quad (4)$$

Essa equação pode ser combinada com a expressão para o torque do motor para eliminar uma variável:

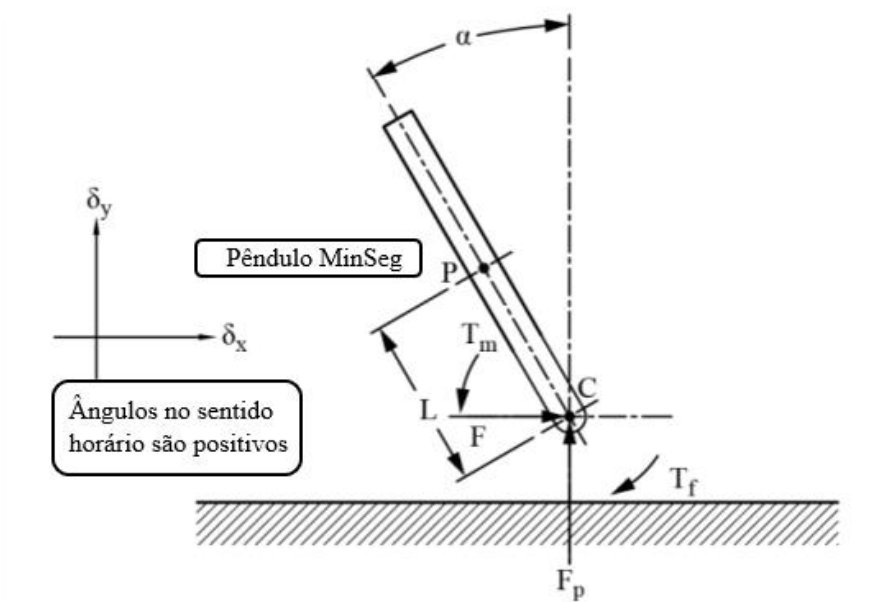
$$F_s = -\frac{T_f}{rw} + \frac{K_t}{R * rw} * V + \frac{K_b * K_t}{R * rw} * \dot{\alpha} - \frac{K_b * K_t}{R * rw} \dot{\theta} * -\frac{I_w}{rw} * \ddot{\theta} . \quad (5)$$

As equações de movimento de translação para a roda podem ser escritas como:

$$mw * \ddot{x} = -F + F_s \quad (6)$$

A seguir, as equações de movimento para o corpo do robô são desenvolvidas seguindo o esquema do corpo, apresentado na Figura 5, cuja nomenclatura e descrição dos parâmetros envolvidos para realizar a modelagem estão expostos na Tabela 3.

Figura 5- Corpo do robô pêndulo invertido.



Fonte: Adaptado de Howard e Bushnell (2015).

Tabela 3- Parâmetros do corpo do robô.

Símbolo	Descrição
P	Centro de massa do pêndulo
α	Ângulo de inclinação
L	Distância do eixo da roda até o ponto P
mp	Massa do pêndulo
Ip	Momento de inércia em P
x	Distância que a roda percorreu
C	Eixo da roda
Fp	Força do corpo do pêndulo em relação a roda
F	Força horizontal no eixo do motor
Tf	Torque de atrito proporcional à velocidade angular do motor
Tm	Torque do motor

Fonte: Adaptado de Howard e Bushnell (2015).

Assim, são derivadas para a direção horizontal, a rotação sobre o centro de massa e o movimento perpendicular ao pêndulo:

$$mp * \ddot{x} = F - mp * L * \ddot{\alpha} * \cos(\alpha), \quad (7)$$

$$Ip * \ddot{\alpha} = -L * \cos(\alpha) * F + L * \sin(\alpha) * Fp - Tm + Tf, \quad (8)$$

$$mp * \ddot{x} * \cos(\alpha) = F * \cos(\alpha) - mp * L * \ddot{\alpha} - Fp * \sin(\alpha) + mp * g * \sin(\alpha). \quad (9)$$

Assumindo o fato de que as rodas não escorregam, $x = \theta * r_w$, e equacionando as forças no eixo do motor, as equações (3), (8) e (9) podem ser combinadas a fim de eliminar a força horizontal no eixo do motor:

$$(Ip + L^2 * mp) * \ddot{\alpha} + L * \cos(\alpha) * mp * \ddot{x} = -\frac{Kt}{R} * V + g * L * \sin(\alpha) * mp + \frac{Kb * Kt}{R * r_w} * \dot{x} - \frac{Kb * Kt}{R} * \dot{\alpha}. \quad (10)$$

Ligando as rodas ao corpo do robô, as equações (6) e (7), juntas e combinadas com a expressão do torque do motor na equação (5), é possível chegar a uma segunda equação para o sistema:

$$L * \cos(\alpha) * mp * rw * \ddot{\alpha} + \left(\frac{Iw}{rw} + mp * rw + mw * rw \right) * \ddot{x} = \frac{Kt}{R} * V - \frac{Kb * Kt}{R * rw} * \dot{x} + \frac{Kb * Kt}{R} * \dot{\alpha} . \quad (11)$$

Isto resulta em duas equações com termos não lineares. Para linearizar o sistema, foi necessário assumir, com relação ao ponto de equilíbrio desejado (ângulo do pêndulo, α , igual a 0), que os ângulos em que o robô opera são pequenos a ponto de considerar $\sin(x) \approx x$ e $\cos(x) \approx 1$:

$$(Ip + L^2 * mp) * \ddot{\alpha} + L * mp * \ddot{x} = -\frac{Kt}{R} * V + g * L * mp * \alpha + \frac{Kb * Kt}{R * rw} * \dot{x} - \frac{Kb * Kt}{R} * \dot{\alpha} , \quad (12)$$

$$L * mp * \ddot{\alpha} + \left(\frac{Iw}{rw^2} + mp + mw \right) * \ddot{x} = \frac{Kt}{R * rw} * V - \frac{Kb * Kt}{R * rw^2} * \dot{x} + \frac{Kb * Kt}{R * rw} * \dot{\alpha} . \quad (13)$$

A seguir é necessário transformar as equações linearizadas para a forma de espaço de estados utilizando as seguintes equações:

$$\begin{aligned} \dot{x} &= A * x + B * u , \\ y &= C * x + D * u . \end{aligned} \quad (14)$$

Nas quais, para o modelo real do robô, a variável de estados X é definida como:

$X := [\alpha \ \dot{\alpha} \ x \ \dot{x}]^T$, o vetor de entrada u = Tensão, o vetor de saída $y = X := [\alpha \ \dot{\alpha} \ x \ \dot{x}]^T$ e $D = [0]$.

Utilizando as equações linearizadas deste modelo, a representação em espaços de estado do modelo pode ser escrita

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{g*q1*q4}{q1^2 + q2*q4} & \frac{q3(q1 - q4*rw)}{(q1^2 + q2*q4)*rw} & 0 & \frac{q3(-q1 + q4*rw)}{(q1^2 - q2*q4)*rw^2} \\ 0 & 0 & 0 & 1 \\ \frac{g*q1^2}{q1^2 + q2*q4} & -\frac{q3*(q2 + q1*rw)}{(q1^2 - q2*q4)*rw} & 0 & \frac{q3*(q2 + q1*rw)}{(q1^2 + q2*q4)*rw^2} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{kt(q1 - q4*rw)}{R(q1^2 - q2*q4)*rw} \\ 0 \\ \frac{kt(-q2 + q1*rw)}{R*(q1^2 - q2*q4)*rw} \end{bmatrix}, \quad (15)$$

$$C = [0 \ 0 \ 1 \ 0], \quad D = [0] . \quad (16)$$

Em que:

$$q1 = L * mp \quad q2 = Ip + L^2 * mp \quad q3 = \frac{Kb*Kt}{R} \quad q4 = -mp - mw - \frac{Iw}{rw^2}$$

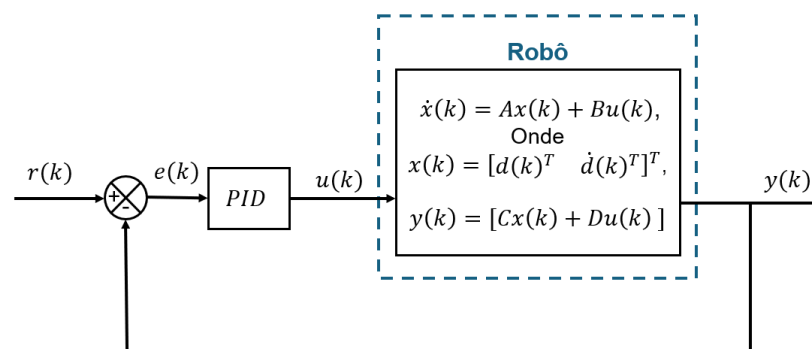
2.2 Controle PID

O controle PID (Proporcional Integral Derivativo) é um método de controle largamente utilizado na indústria para obter um resultado preciso de acordo com parâmetros de entrada e saída (Ogata, 2010).

Esse método de controle utiliza três tipos de ganhos, proporcional, integral e derivativo, que atuam de formas diferentes. De acordo com Paz (2001), o ganho proporcional (K_p) leva em conta os valores presentes do erro. Por exemplo, se o erro é grande e de sinal positivo, a saída do controlador também será grande e positiva. Já o ganho integral (K_i) leva em conta os valores dos erros passados. Por exemplo, se a corrente de saída não é suficientemente grande, a integral do erro se acumulará com o passar do tempo e o controlador responderá com uma ação mais forte para compensá-lo. Por fim, o ganho derivativo (K_d) leva em conta possíveis valores de erro futuros, baseada em sua atual taxa de mudanças. Continuando com o exemplo do ganho P, quando a saída de controle positiva é capaz de levar o erro perto de zero, este também coloca o processo a caminho de um erro grande e sinal negativo no futuro. Neste caso, o valor do ganho derivativo se torna negativo.

No caso deste projeto, a variável de entrada é o ângulo do robô. Através de um valor determinado para o *Setpoint*, que indica a posição de equilíbrio do robô, o controlador processa a informação da posição angular deste objeto e assim são acionados os motores DC que tentam mantê-lo estabilizado. Conforme a variação angular do robô, essa informação retorna para ser comparada com o valor do *setpoint* e assim os motores recebem maior ou menor potência, sendo aplicado a eles um sinal PWM. Na Figura 6 é possível observar o modelo em malha fechada do robô.

Figura 6 - Controle de malha fechada do robô.



Fonte: Produção do próprio autor.

A equação abaixo descreve a saída obtida do controlador PID de tempo discreto, dada por:

$$u(k) = Kp \left[e(k) + \frac{T_o}{T_i} \sum_{v=0}^k e(v) + \frac{T_d}{T_o} * (e(k) - e(k-1)) \right], \quad (17)$$

sendo

$$Ki = \frac{Kp}{T_i}, \quad Kd = Kp * T_d$$

Os parâmetros Ki , Kp e Kd são, respectivamente, os ganhos integral, proporcional e derivativo, $r(k)$ é o valor de referência, $u(k)$ é a saída do controlador PID, T_o é o período do sinal e $e(k)$ é o erro.

Com a equação do controlador em tempo discreto, devido ao fato de cada ganho interagir de uma forma com o erro definido, deve-se definir também como os valores dos ganhos afetam o funcionamento do sistema. Sendo assim, de acordo com Moreira *et al.*, (2021), o ganho Kp interfere no tempo de resposta a uma dada entrada em um sistema, quando comparado a um sistema de controle do tipo *on/off*. Esse ganho é caracterizado pela redução do erro em estado estacionário, mas não provoca sua total supressão; o ganho Ki realiza a total eliminação do erro em estado estacionário. Contudo, o tempo de resposta torna-se mais lento a determinadas entradas, além de provocar um aumento no *overshoot* da resposta em regime transiente a uma entrada degrau.

Por fim, o ganho Kd agiliza a conversão do sinal para o valor de referência, realizando uma pequena redução no tempo de resposta. Este ganho também causa um aumento no amortecimento do sistema, promovendo uma redução no *overshoot* na resposta em regime transitório. Todavia, em sistemas com sensores com ruído considerável, este ganho pode interferir causando oscilações indesejadas.

2.3 Obtenção da função de transferência discreta

Em um modelo de planta representado em espaço de estados, sua vantagem com relação à função de transferência é relacionar ao sistema diversas entradas e saída (Assunção, 2020). Porém, dependendo da aplicação, se demonstra necessária a transformação de uma representação para outra (Cabral, 2013).

Partindo de um modelo de espaço de estados contínuo, quando aplicado a um sistema discreto, é necessária a transformação deste espaço de estados para tempo discreto, pois o controlador será implementado em um microcontrolador, então, o controle será em tempo discreto. Essa sequência está apresentada na Figura 7.

Figura 7 – Espaço de estado contínuo para função de transferência discreta.



Fonte: Adaptado de Assunção (2020).

De acordo com Cabral (2013), para se transformar um espaço de estados contínuo em discreto, é necessário realizar uma discretização temporal. Assim, dado um sistema linear invariante no tempo de ordem N representado na forma de espaço de estados:

$$\begin{cases} \dot{x}(t) = A * x(t) + B * u(t) \\ y(t) = C * x(t) + D * u(t) \end{cases} \quad (18)$$

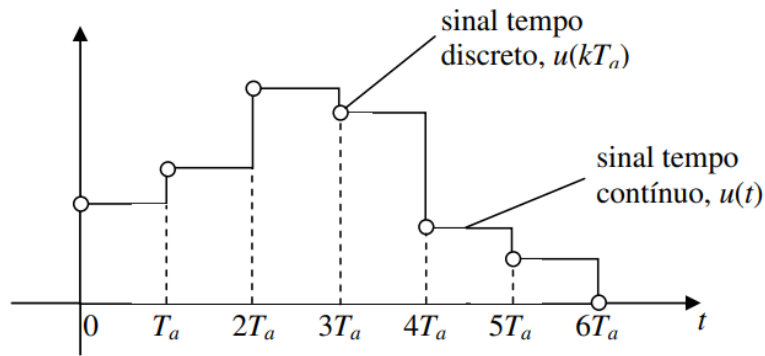
onde $x(k) \in R^N$, $u(k) \in R^M$ e $y(k) \in R^P$. A resposta Analógica desse sistema é dada por

$$x(t) = e^{A*t} * x(0) + \int_0^t e^{A*(t-\tau)} * B * u(\tau) d\tau \quad (19)$$

Em um conversor Digital/Analógico, durante um período de amostragem (T_a) as entradas do sistema permanecem constante conforme a Figura 8 e a equação abaixo:

$$u(t) = u(k * T_a), \text{ para } k * T_a < t \leq (k + 1) * T_a. \quad (20)$$

Figura 8 -Gráfico do sinal discretizado.



Fonte: Cabral (2013).

Realizando a integral da equação (19) para um período de amostragem ($kT_a \leq t \leq (k + 1)T_a$) e usando a restrição imposta pela eq. (20), obtêm-se:

$$x[(k + 1) * T_a] = e^{A * [(k+1) * T_a - k * T_a]} * x(k * T_a) + \left(\int_{k * T_a}^{(k+1) * T_a} e^{(A * ((k+1) * T_a - \tau))} * B * d\tau \right) * u(k * T_a). \quad (21)$$

Simplificando a notação (omitindo período de amostragem na escala de tempo) e realizando mudança de variável de t para $\sigma = \tau - k * T_a$ para simplificar o segundo termo do lado direito, é obtida a equação:

$$x(k + 1) = e^{A * T_a} * x(k) + \left(\int_0^{T_a} e^{(A * (\sigma))} * B * d\sigma \right) * u(k), \quad (22)$$

ou

$$x(k + 1) = \Phi * x(k) + \Gamma * u(k), \quad (23)$$

na qual

$$\Phi = e^{A^*Ta}, \quad \Gamma = \int_0^{Ta} e^{(A^*(\sigma))} * B * d\sigma. \quad (24)$$

Na medida em que a equação da saída é uma equação algébrica a sua versão em tempo discreto é simplesmente dada por:

$$y(k) = C * x(k) + D * u(k). \quad (25)$$

A função de transferência de um sistema em tempo discreto pode ser obtida da sua representação no espaço dos estados da mesma forma em para tempo contínuo, porém utilizando a Transformada Z no lugar da Transformada de Laplace. Assim, aplicando a Transformada Z na equação (23):

$$Z \{x(k + 1)\} = Z \{\Phi * x(k) + \Gamma * u(k)\}. \quad (26)$$

Assumindo condições iniciais nulas:

$$z * X(z) = \Phi * X(z) + \Gamma * U(z). \quad (27)$$

Rearranjando e isolando X(z),

$$X(z) = (z * I - \Phi)^{-1} * \Gamma * U(z). \quad (28)$$

Aplicando Transformada Z na equação (25) e substituindo X(z) da equação (28), podemos afirmar que:

$$Y(z) = [C(z * I - \Phi)^{-1} * \Gamma + D] * U(z). \quad (29)$$

Por fim, a matriz de funções de transferência é dada por:

$$G(z) = C * (z * I - \Phi)^{-1} * \Gamma + D. \quad (30)$$

2.4 PWM

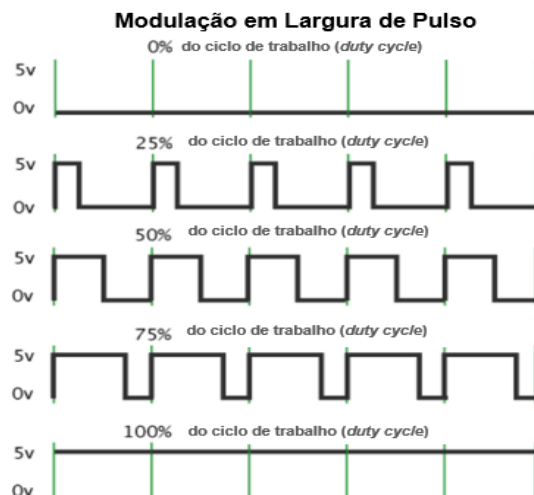
O PWM (*Pulse Width Modulation*), de acordo com Butterfield e Szymanski (2018) é todo e qualquer método de representar um sinal como onda retangular com um *duty cycle* variável, ou seja, uma técnica de controle de potência que ocorre através da variação da largura de pulso de um sinal ou tensão. A equação para se calcular o *duty cycle* é dada por:

$$D = 100\% * \frac{T_{on}}{(T_{off} + T_{on})} . \quad (31)$$

Para essa equação, tem-se que D é o *duty cycle*, Ton é o valor da largura de pulso e a soma de Ton e Toff é o valor do período do sinal. Assim, ao se definir, para um sinal de pulso de tensão de determinada frequência, uma largura de pulso de 50%, por exemplo, a tensão média recebida pela carga do circuito é de 50%, o que implica que a potência recebida pela carga é diretamente proporcional ao valor da largura de pulso, ou seja, o valor médio da tensão equivale a 50% da tensão nominal.

A modulação por largura de pulso é utilizada em sinais de tensão retangulares, principalmente aplicada em controle de motores DC, áudio e amplificadores e na área de telecomunicações. Assim, aplicando o método de modulação em largura de pulso, é possível controlar a velocidade dos motores DC sem que haja a necessidade variação do valor da tensão nominal ou corrente. Na Figura 9 estão apresentadas formas de onda retangulares de acordo com a porcentagem da largura de pulso.

Figura 9- Sinais de tensão por largura de pulso.



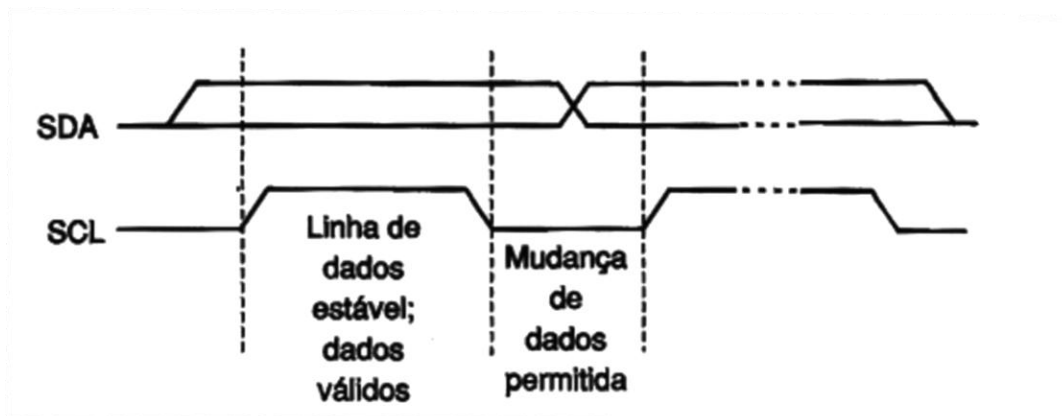
Fonte: Adaptado de Arduino (2024).

2.5 Barramento I²C

O barramento I²C, segundo Braga (2018), é um protocolo de comunicação serial que utiliza duas ligações (SDA- *Serial Data* e SCL- *Serial Clock*) inventado pela empresa Philips na década de 90. O uso do barramento I²C permite a fácil integração em componentes de interface, que possam trabalhar diretamente com seus sinais simplificando assim a comunicação com outros componentes da mesma família. Esse tipo de barramento aceita qualquer tipo de fabricação de CI's (Circuitos Integrados), permitindo a incorporação de qualquer dispositivo ao barramento sem que haja conflito entre os componentes.

Assim, o barramento I²C funciona com dois dados seriais (*serial data* e *serial clock*) sendo a velocidade de transferência de dados estando entre a faixa de 100kb/s até 400kb/s no modo Standard. Na Figura 10 está apresentado uma representação de como ocorre a transferência de dados no barramento I²C.

Figura 10 - Transferência de bits no barramento I²C.



Fonte: Braga (2019).

3 CONSTRUÇÃO DO ROBÔ PÊNDULO INVERTIDO

A construção deste robô baseia-se no modelo de robô pêndulo invertido Segway apresentado por Andrade *et al.*, (2019). No caso desse trabalho, será utilizada uma placa de desenvolvimento ESP32 DevKitC V4 com módulo ESP32-WROOM integrado para realizar o controle dos dispositivos periféricos que são o sensor Acelerômetro/Giroscópio modelo MPU6050 e a ponte H modelo L298N para o controle de sentido de giro e rotação dos dois motores DC (3-6V) modelo JGA25-370.

Para a estrutura, foi utilizada uma placa de circuito impresso, cujo projeto foi desenhado no software Altium Designer, 4 cantoneiras em L e dois suportes planos para fixar os motores. Além disso, como elementos secundários foram utilizados um kit de fios *jumper* para protoboard, duas barras de pino fêmea e duas barras de pino macho, uma chave alavanca de 2 posições, 6 baterias de 1,2V, suporte para bateria tipo AAA, dezoito parafusos M3x6. No APÊNDICE E está apresentado o esquemático do circuito elétrico do robô.

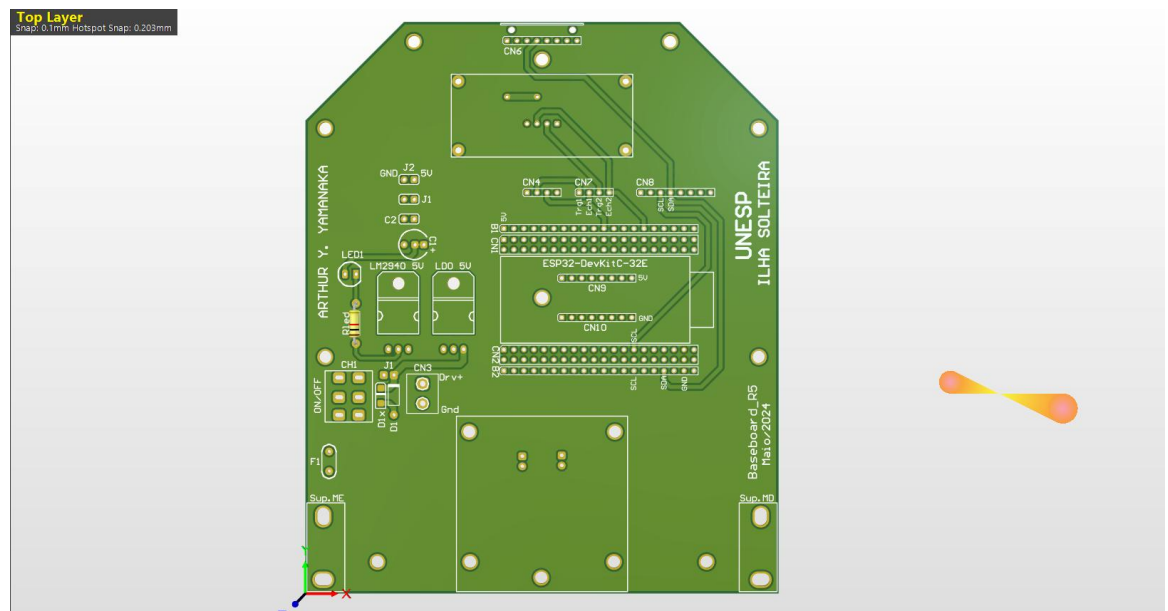
Primeiramente, para a construção do robô, foi necessário realizar o esquemático do circuito e as conexões necessárias. O primeiro passo foi o conjunto de baterias, que fornecem 7.2V, será a fonte de alimentação de todo o circuito, e, portanto, deve fornecer energia ao ESP32, ao L298N e ao sensor acelerômetro/giroscópio. Para facilitar a opção de ligar e desligar o robô, foi inserida uma chave alavanca de duas posições em série com as baterias. Além disso, foi inserido também um fusível resetável com propósito de proteção do circuito caso a corrente suba muito.

Como o ESP32 opera em uma tensão de 5V, foi utilizado um regulador de tensão, L2940, para que a tensão não ultrapasse 5V na alimentação do microcontrolador. Além disso foi adicionado um diodo (D1) modelo 1N4007, como pode-se observar pelo Apêndice E, com o propósito de que caso se queira alterar o código do ESP através do uso de um cabo micro USB, os motores do robô não fiquem acionados durante a instalação do código.

Prosseguindo com o circuito, os pinos 21 e 22 do ESP devem ser conectados ao sensor MPU6050 às portas SCL e SDA respectivamente para comunicação I²C. Devem ser selecionados 4 pinos I/O (pinos 16, 17, 18 e 19) do ESP32-DevKitC V4 que são conectados às portas INPUT do L298N para acionamento dos motores, que são ligados às saídas do L298N e 2 pinos I/O (pinos 4 e 15) para leitura do encoder do motor através dos cabos *jumper*.

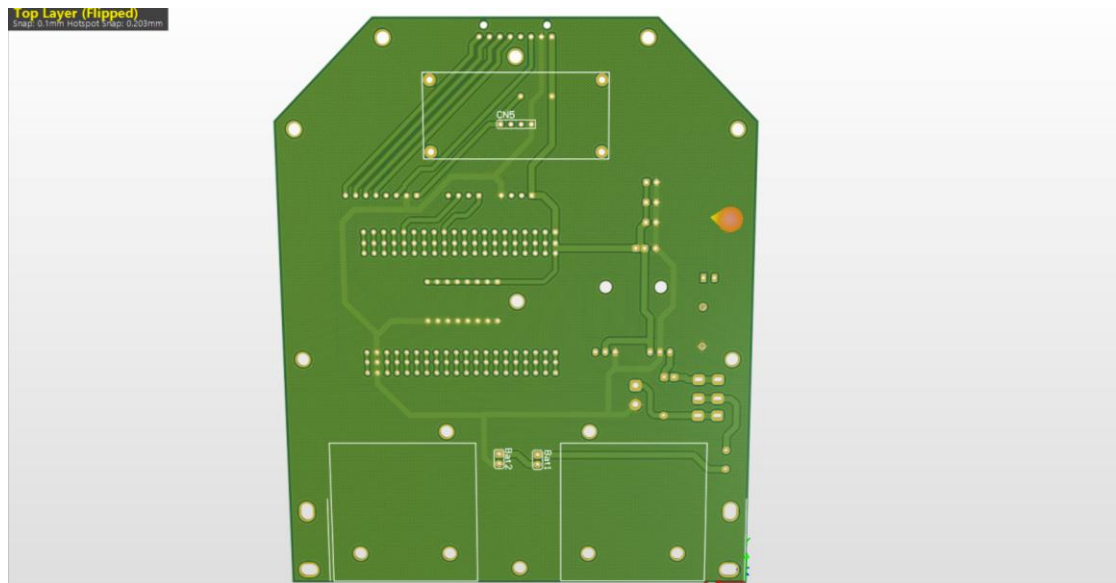
Após isso, foi realizado o *Gerber* do circuito impresso no software com as dimensões da placa de 120x145x1,5 mm (Largura, Comprimento e Espessura). As Figuras 11 e 12 apresentam a imagem frontal e traseira do modelo 3D da placa.

Figura 11 - Imagem 3D frontal da placa.



Fonte: Produção do próprio autor.

Figura 12 - Imagem 3D posterior da placa.



Fonte: Produção do próprio autor.

Em seguida a PCI (placa de circuito impresso) foi enviada para fabricação e após finalizada, com base nos furos da placa, houve a furação das cantoneiras para fixação dos motores. Com isso, foram fixadas à placa de circuito impresso, através de parafusos e porcas, a ponte H, o suporte para as baterias e os motores.

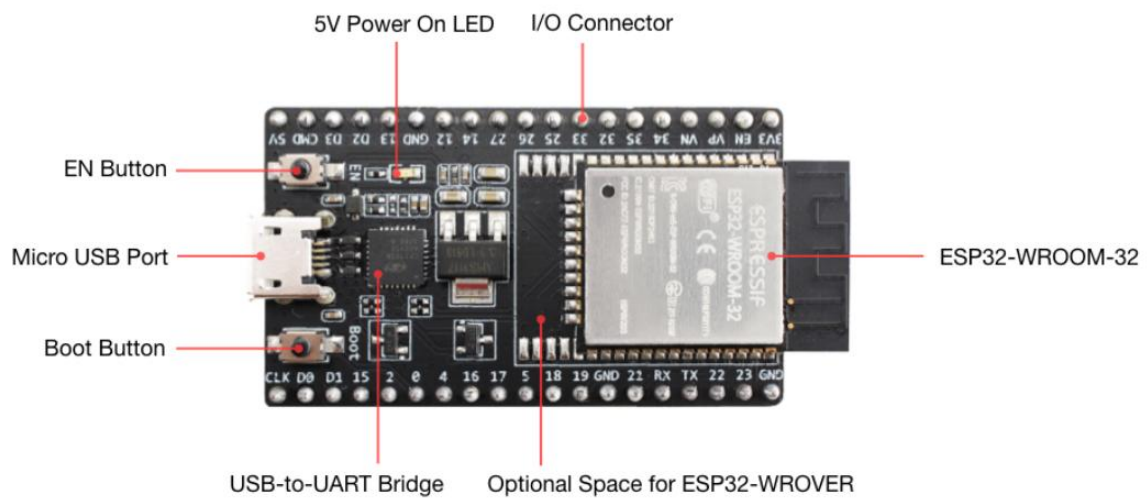
Foi realizada a soldagem do MPU6050 e das barras de pino tipo macho e tipo fêmea à placa para que seja possível a conexão do driver L298N à placa ESP32 através dos fios *jumper* e para conectar o próprio ESP32 na placa. Foram soldadas também a chave de duas posições e componentes complementares no projeto, como: capacitores (utilizados como filtros, eliminando flutuações indesejadas e picos de tensão), diodo, o led, o fusível resetável e o regulador de tensão.

Após todos os componentes estarem integrados, foram realizadas as conexões com os fios *jumper* no driver do motor, conforme o diagrama esquemático do APÊNDICE E, e por fim os ajustes e implementação do código realizado no software Arduino IDE à placa ESP32 DevKitC V4 para o funcionamento desejado do robô.

3.1 ESP-32 DevKitC V4

A placa de desenvolvimento ESP32 DevKitC V4, segundo a Espressif (2024), é uma placa de tamanho pequeno baseada no módulo ESP32-WROOM que apresenta um barramento de pinos macho. Na Figura 13 é possível observar a imagem da placa de desenvolvimento.

Figura 13 - ESP32-DevKitC V4 com o módulo ESP32-WROOM-32.



Fonte: Espressif (2024).

Tendo em vista os componentes apontados pela Figura 13, na Tabela 4 estão apresentadas as descrições de cada um.

Tabela 4- Descrição dos componentes principais do ESP32-DevKitC V4.

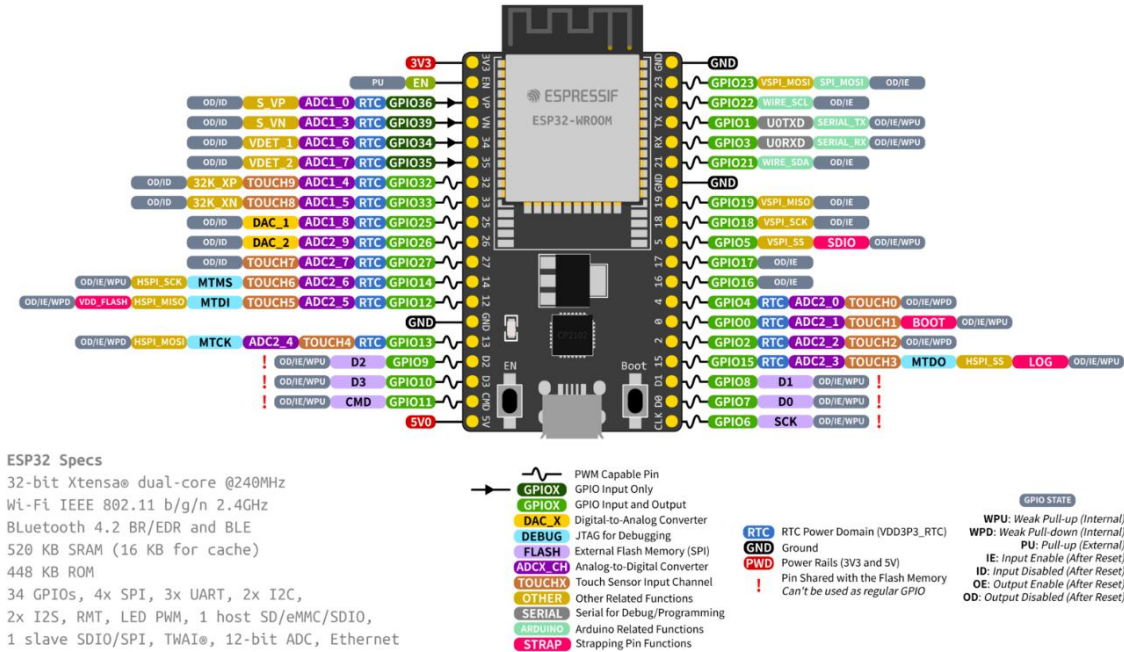
Componentes Principais	Descrição
ESP32-WROOM-32	Módulo com o ESP32 como núcleo da placa
EN	Botão de reset da placa
Boot	Botão de download da placa (Segurar o EN e pressionar o Boot aciona o modo de download do firmware através da porta serial)
USB-to-UART Bridge	Single USB-UART bridge chip provém taxas de transferência de dados de até 3 Mbps.
Porta Micro USB	Interface USB. Fornece energia para a placa e serve como interface de comunicação entre o computador e o módulo ESP32-WROOM-32.
5V Power On LED	LED que é acionado quando o USB ou uma fonte externa de 5V é conectada à placa
I/O	A placa fornece diversas portas que podem ser programadas para executar diversas funções como: PWM, ADC, DAC, I2C, I2S, SPI, etc.

Fonte: Espressif (2024).

Além desses aspectos descritos pela Tabela 4, o ESP32-DevKitC V4 utiliza um processador de 32 bits Xtensa dual-core que opera até 240MHz, possui conexão *WiFi* e *Bluetooth* incorporados, uma memória SRAM de 520KB e uma memória ROM de 448KB. Com relação às portas, a placa possui 34 GPIOs (*General Purpose Input/Output*), sendo que 4 dessas podem ser aplicadas à comunicação SPI (*Serial Peripheral Interface*), 3 à UART (*Universal Asynchronous Receiver/Transmitter*) e 2 portas para comunicação I²C (*Inter-Integrated Circuit*). A Figura 14 apresenta o esquema de pinagem da placa associada com as possíveis funções de cada pino.

Figura 14 - Esquema de pinagem do ESP32-DevKitC V4.

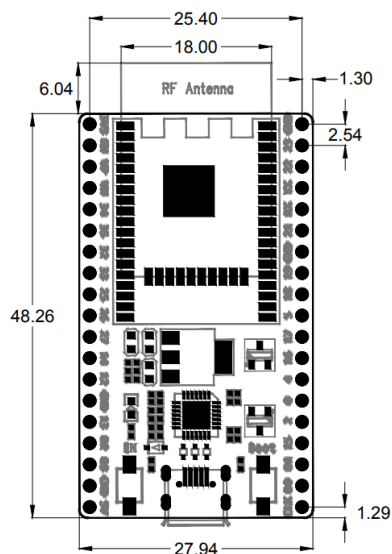
ESP32-DevKitC



Fonte: Espressif (2024).

Na Figura 15, é possível observar o desenho do ESP32-DevKitC V4 com os valores de suas dimensões.

Figura 15 - Dimensões da placa ESP32-DevKitC V4.



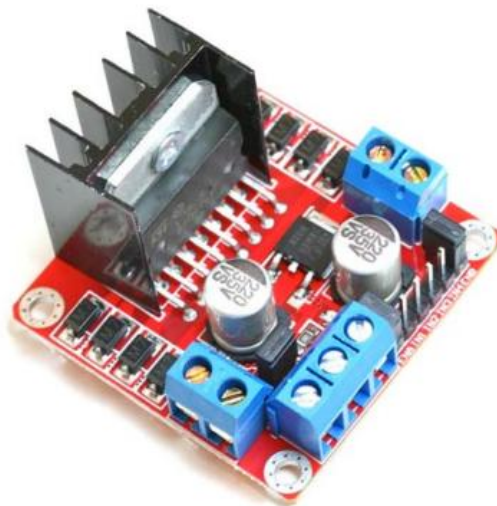
Fonte: Espressif (2024).

3.2 Ponte-H L298N

Conforme a descrição da Handson Technology (2017), o L298N é um *driver* com duas pontes H inspirado no componente integrado L298, permitindo o controle independente de dois motores simultaneamente e podendo variar a direção de rotação dos motores. Este é muito utilizado em aplicações na área de robótica e se adequa bem em que é necessária a conexão entre um microcontrolador e algumas linhas de controle por motor.

Este driver possui LEDs indicadores de energia, que acendem quando o *driver* é alimentado com uma fonte de tensão, um regulador de tensão de 5V e diodos de proteção. O L298N pode ser interconectado a chaves manuais, portas lógicas e relés. A Figura 16 apresenta a imagem do módulo L298N.

Figura 16 - Driver motor L298N.



Fonte: Handson Technology (2017).

A Tabela 5 apresenta os dados relacionados à operação do driver.

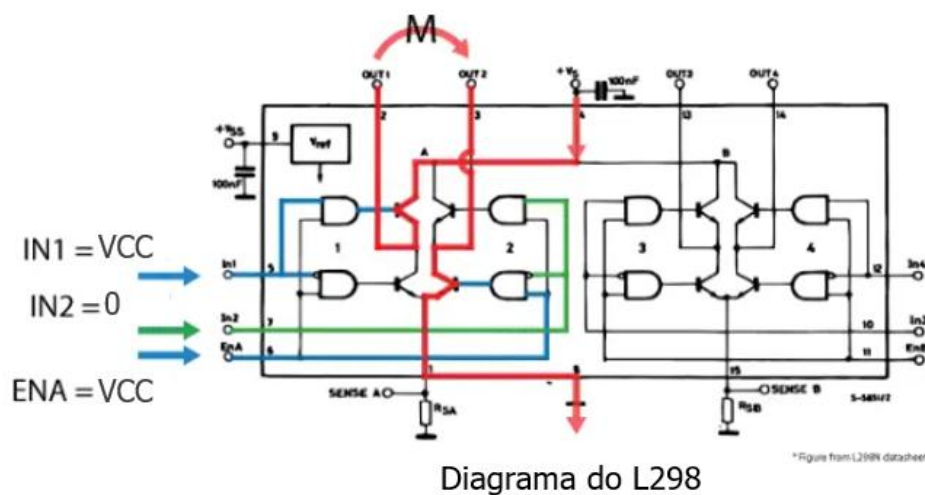
Tabela 5 - Dados de especificação do L298N.

Tensão lógica	5V
Tensão de operação	7 a 30 V
Corrente de operação	Até 2A por canal
Dimensões	43 x 43 x 27 mm
Peso	30 g

Fonte: Robocore (2024).

O driver também é capaz de controlar a velocidade do motor. Utiliza-se um sinal PWM para regular o nível de rotação do eixo. PWM (*Pulse Width Modulation*) é traduzido como Modulação por Largura de Pulso (MLP), e utiliza a razão do ciclo (*duty cycle*) para controlar o valor de alimentação por tensão. Como o sinal de PWM varia de 0% a 100% em relação ao *duty cycle*, as placas ESP convertem o sinal gerado pelo comando *analogWrite* proporcionalmente. Deste modo, o valor 255 do sinal digital da placa é convertido para um sinal analógico que produz um PWM de 100%, e o valor 0 é convertido para um sinal PWM de 0%. Portanto, para o valor aplicado como 0, o motor está em repouso e quando o valor é 255 o motor gira em máxima potência. A Figura 17 apresenta o esquemático elétrico do L298N.

Figura 17 - Esquemático da ponte H L298N.



Fonte: Adaptado de How to Mechatronics (2017).

Pelas cores utilizadas nas linhas é possível observar o caminho percorrido pelos sinais de entrada e saída através do circuito. As linhas azuis representam o sinal de tensão alto, as verdes representam sinal baixo de tensão e as vermelhas representam o sinal de saída que, determinados pelos sinais de entrada, determina o sentido de giro do motor. Caso IN1 e IN2 sejam invertidos, o sentido de giro é invertido. Outro ponto importante é que a entrada ENA deve estar sempre com valor de tensão elevado para que o circuito funcione.

3.4 Motores DC

Os motores DC utilizados no robô foram motores modelo JGA25-370 cujas especificações dos motores estão na Tabela 7

Tabela 7- Especificações do motor JGA25-370.

Tensão de alimentação	3~6 V
Corrente em aberto	100mA
Corrente com máxima carga	500mA
Corrente de <i>Stall</i>	2.6 A para 6V
Relação da caixa de redução:	35:1
Velocidade	170 rpm
Torque	600 gf.cm

Fonte: Aliexpress (2024).

Além disso, os motores JGA25-370, de acordo com a Precision Mini Drives (2024), possuem *encoders* acoplados à sua estrutura, cujos sensores funcionam por efeito Hall, que permitem a leitura de distância percorrida pelo robô. Tais *encoders* possuem 11 PPR (pulsos por rotação), ou seja, a cada rotação do disco, são lidas 11 posições. Neste caso, como a redução do motor é de 1:35, a cada 35 rotações do disco, a roda realiza uma rotação, fazendo com que sejam percorridos 385 pontos até que se complete 360 graus.

Portanto o encoder é capaz de captar uma variação mínima de 0,935 graus no ângulo da roda. A Figura 19 apresenta a imagem do motor onde é possível observar o *encoder* na parte inferior.

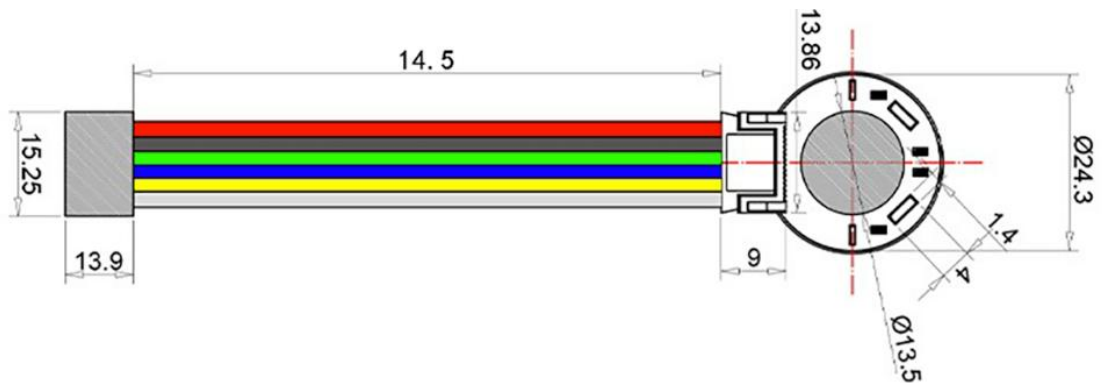
Figura 19 - Motor DC modelo JGA25-370



Fonte: Aliexpress (2024).

Na Figura 20 está esquematizado o *encoder* com suas dimensões juntamente com o conector JST de 6 pinos com a diferenciação dos fios por cor.

Figura 20 - Fios do *encoder* do motor JGA25-370.



Fonte: Precision Mini Drives (2024).

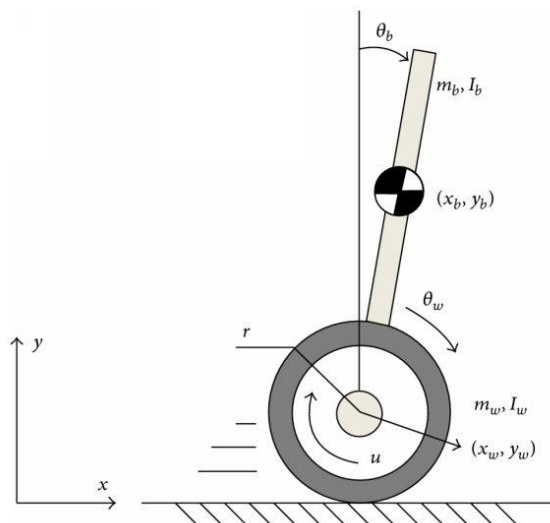
O fio vermelho é ligado ao polo positivo de tensão de alimentação do motor e o fio branco é ligado o polo negativo. Já os fios preto e azul, respectivamente, são os polos negativo e positivo de alimentação do encoder, sendo a tensão de operação do encoder na faixa entre 3.3V e 5V. por fim, os fios amarelo e verde enviam os sinais de *feedback* de leitura do *encoder*.

4 PROGRAMAÇÃO DO ROBÔ

Foi utilizado, para programação do robô, a plataforma ARDUINO IDE que é uma plataforma programável de código livre que permite a programação em código C/C++ do ESP32. O código aplicado ao robô está apresentado no APÊNDICE C deste trabalho.

Para o funcionamento correto do robô, é necessária a atribuição dos pinos aos terminais dos motores, pinos 16, 17, 18 e 19, do MPU6050, pinos 21 e 22 e do encoder do motor, pinos 4 e 15, como já citados anteriormente. Conforme o robô se inclina para um sentido (para frente ou para trás), os motores devem atuar girando no mesmo sentido da inclinação para que ele volte para a posição de equilíbrio, conforme a Figura 21.

Figura 21- Funcionamento do robô pêndulo invertido.



Fonte: Ri, Songhyok *et al.* (2014).

A partir das leituras do acelerômetro/giroscópio, com a utilização das bibliotecas disponíveis e abertas, através dos parâmetros de ganho PID o valor da largura de pulso PWM varia de acordo com a posição angular e sua variação. Caso o valor do ângulo lido pelo MPU6050 fosse positivo, o motor girava para frente e caso fosse negativo, girava em sentido contrário. Como o valor analógico é discretizado, a largura de pulso do sinal de tensão é dividida em 256 partes, já que a saída analógica é de 8 bits. Por fim, os *encoders* servem como medidores de distância para que haja uma melhor calibração do robô e um recurso a mais necessário para realizar a análise do modelo.

4.1 Arduino IDE

O Arduino IDE (*Integrated Development Enviroment*), de acordo com Fezari e Al Dahoud (2018), é um software de código aberto (*open source*) introduzido pela Arduino.cc que é muito utilizado para escrever, editar, compilar e instalar o código a um dispositivo Arduino ou dispositivos compatíveis. Esta plataforma suporta as linguagens de programação C e C++. A Figura 22 apresenta a imagem da interface do Arduino IDE, na qual é possível notar os seguintes ícones:

- Verificar e carregar: Compila e carrega o código para sua placa compatível.
- Selecionar placa e porta: Detecta a placa automaticamente, juntamente com o número da sua porta de comunicação.
- Biblioteca de Rascunhos: Opção para encontrar os sketches salvos no seu computador ou dispositivo de armazenamento de dados.
- Gerenciador de placas: Navega através dos pacotes do Arduino e de terceiros para selecionar o modelo da placa utilizado.
- Gerenciador de bibliotecas: Pesquisa através das bibliotecas de programação feitas tanto pela empresa Arduino quanto pela comunidade.
- Depurador: Realiza o teste e debug dos programas em tempo real.
- Buscar: Procura por palavras-chave no código
- Abrir Monitor Serial: Habilita a ferramenta de monitor serial que possibilita a visualização dos dados, sejam eles números ou letras.
- Abrir *Serial Plotter*: Habilita a ferramenta do *plotter* serial que possibilita a visualização gráfica dos dados.

Figura 22 - Interface do software Arduino IDE.



Fonte: Adaptado de Söderby e Hylén (2024).

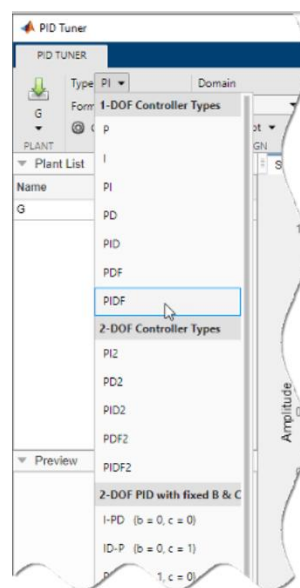
Segundo a Arduino (2024), o Arduino IDE é uma ferramenta versátil que possibilita ao usuário a instalação de bibliotecas com funções e aplicações diversas, sincronizar os *sketches* (rascunhos) com o *Arduino Cloud* e muito mais. Além disso, por ser de código aberto, o acesso, o desenvolvimento de códigos e a possibilidade de as pessoas poderem desenvolver um equipamento ou projeto tornam-se enormes, permitindo maior aprendizado prático sobre programação e a autonomia às pessoas poderem criar projetos de automação, por exemplo, com baixo custo.

4.2 Matlab PID TUNER

O PID TUNER, conforme Saputra *et al.* (2023), é um aplicativo do software MATLAB que auxilia no ajuste dos ganhos do controlador PID provendo os valores com melhor performance do sistema para os parâmetros especificados. O controlador é utilizado para plantas tipo SISO (*Single Input Single Output*), equilibrando os aspectos de robustez e performance do controlador.

Seguindo a documentação disponibilizada pelo site da MathWorks (2018), o PID TUNER permite a seleção dos tipos de controladores P, PI, PD, PID, PDF e PIDF, e a quantidade de graus de liberdade (1 ou 2) do controlador. A Figura 23 fornece a imagem da interface de configuração do tipo de controlador no PID TUNER.

Figura 23 - Tipos de controlador PID TUNER.

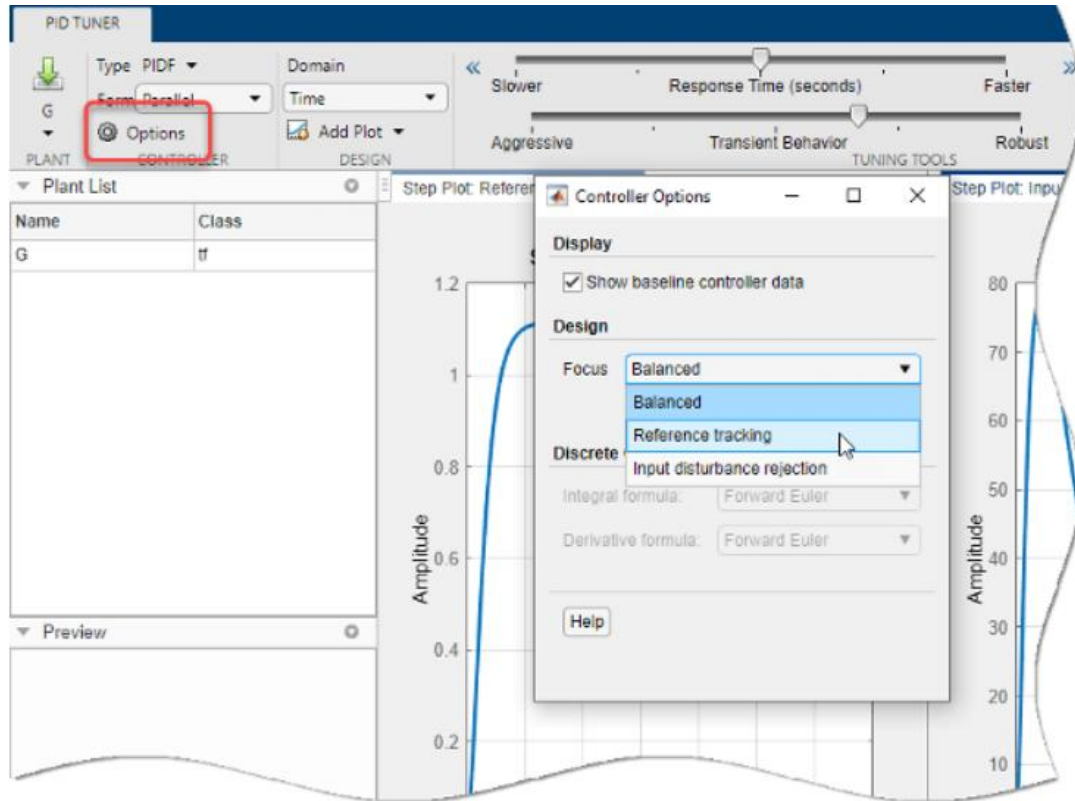


Fonte: MathWorks (2018).

Além dos tipos de controladores disponíveis nesse aplicativo, é possível alterar na opção “Focus” qual o foco do controlador, cujas opções são: *Balanced*, *Reference Tracking* e *Input Disturbance Rejection*.

Na opção “*Balanced*” os valores do controlador são calculados para um sistema mais equilibrado. Já na opção “*Reference Tracking*” o software irá focar em se aproximar o máximo possível de uma determinada resposta que foi apontada pelo usuário como referência. E a opção “*Input Disturbance Rejection*” se baseia em calcular os ganhos para maior rejeição de distúrbios. A Figura 24 apresenta a interface de opções do controlador.

Figura 24 - Opções do controlador no PID TUNER.



Fonte: MathWorks (2018).

Nas opções do controlador, mostradas da Figura 22, é possível também modificar as fórmulas integral e derivativa de sistemas discretos, cujas opções de fórmulas são: “*Forward Euler*”, “*Backward Euler*” e “Trapezoidal”. A equação do controlador PID discreto utilizado pelo PID TUNER é:

$$C = Kp + Ki * IF(z) + \frac{Kd}{Tf + DF(z)} . \quad (32)$$

Nesta equação, $IF(z)$ se refere à fórmula integral e $DF(z)$ se refere à fórmula derivativa. Das três opções de fórmulas citadas, a fórmula “*Forward Euler*” é mais utilizada para pequenas amostras de tempo, a qual é determinada como:

$$IF(z) \text{ ou } DF(z) = \frac{T_s}{z - 1} . \quad (33)$$

Já a equação “*Backward Euler*” possui a vantagem de que ao discretizar um sistema estável em tempo contínuo, o resultado fornecido é sempre estável. Tal equação é dada por:

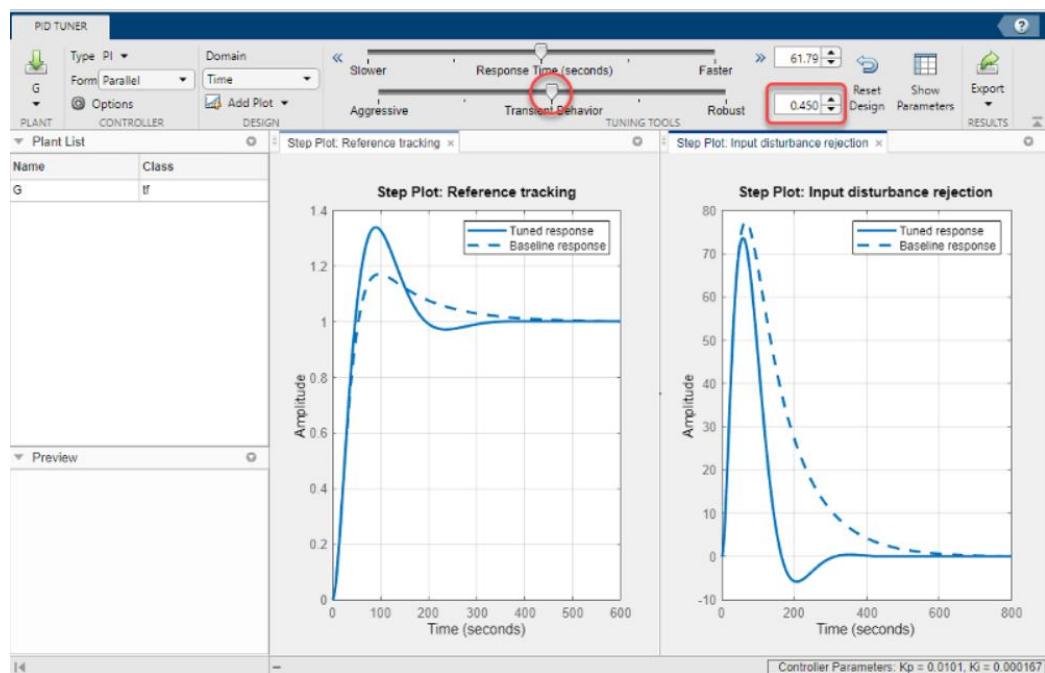
$$IF(z) \text{ ou } DF(z) = \frac{Ts * z}{z - 1}. \quad (34)$$

Por fim, a fórmula “Trapezoidal” fornece as propriedades no domínio da frequência mais próximas entre um sistema discretizado e seu sistema de tempo contínuo correspondente. Tal equação é formulada:

$$IF(z) \text{ ou } DF(z) = \frac{Ts}{2} * \frac{z + 1}{z - 1}. \quad (35)$$

Para realizar o ajuste da resposta para o comportamento desejado, são necessárias as regulagens do tempo de resposta do sistema para estabilizar mais rápida ou lentamente a saída, e o comportamento transiente, que interfere no overshoot e nas oscilações do sistema até que ele atinja o valor alvo. Na Figura 25 está apresentada a interface do PID TUNER para determinar o comportamento da resposta esperada do sistema.

Figura 25 - Ajustes parâmetros da resposta no PID TUNER.



Fonte: MathWorks (2018).

5 RESULTADOS

Primeiramente, a construção do robô foi realizada com base no esquemático do circuito, do desenho 3D e das peças e componentes listadas neste trabalho. Portanto, foi necessário inicialmente, realizar uma busca pelos itens necessários com custo-benefício otimizado. Na Tabela 8 está apresentada a lista de componentes e peças utilizados com seus respectivos valores de mercado.

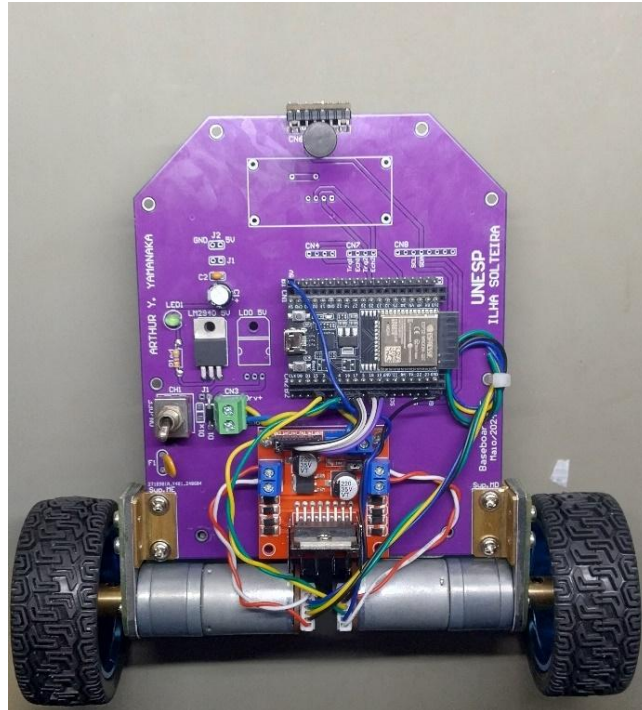
Tabela 8 – Lista de valores dos componentes e peças.

Componentes/ Peças	Valor de mercado
ESP32-DevKitC V4	R\$ 45,00
Motores DC(3-6V)	R\$ 90,00
MPU-6050	R\$ 15,00
L298N	R\$ 16,00
Chave alavanca 2 posições	R\$ 10,00
Capacitor eletrolítico (330uF)	R\$ 2,00
Diodo (N4007)	R\$ 0,10
LED verde	R\$ 0,20
Fusível resetável	R\$ 3,00
Baterias recarregáveis	R\$ 100,00
Regulador de tensão LM2940	R\$ 10,00
Suporte de baterias	R\$ 20,00
Placa de circuito impresso	R\$ 15,00
Valor Total	R\$ 316,30

Fonte: Produção do próprio autor.

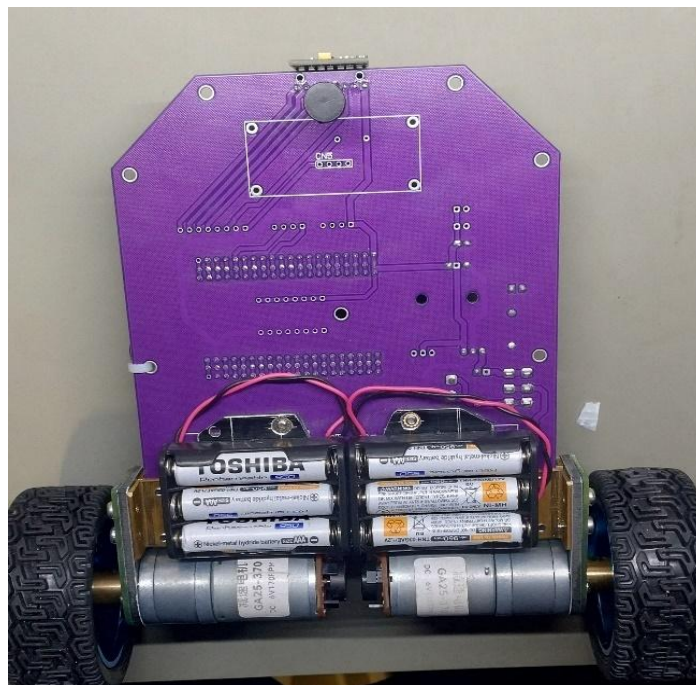
Assim, é possível observar que o valor gasto foi acessível para a realização deste trabalho, e partir desses itens, foi realizada a montagem do robô. Em comparação com o modelo MinSegShield M2V5, o valor de mercado, segundo a MinSeg (2024), é de 195 dólares. As Figuras 26 e 27 mostram as imagens frontal e posterior do robô montado.

Figura 26 – Imagem frontal do robô.



Fonte: Produção do próprio autor.

Figura 27 – Imagem posterior do robô.



Fonte: Produção do próprio autor.

Para encontrar os parâmetros necessários para obtenção da função de transferência do modelo, foram necessárias medições utilizando uma balança, um paquímetro e um multímetro. Apenas para os valores Kb e Kt foram necessários realizar os cálculos com as seguintes equações:

$$Kb = \frac{V_{peak}}{S_{noload}} , \quad (36)$$

$$Kt = \frac{Tm}{2\pi i \cdot \frac{1}{Kb}} . \quad (37)$$

Nas equações acima, S_{noload} corresponde à velocidade de rotação sem carga do motor e V_{peak} é a tensão de pico. Assim, foram encontrados os valores dos parâmetros descritos na Tabela 9.

Tabela 9- Parâmetros medidos do robô.

Símbolo	Descrição	Valor
g	Aceleração da gravidade	9.81 m/s ²
h	altura total do pêndulo	16,14 cm
l	largura total do pêndulo	12 cm
L	distância do eixo da roda até o ponto P	6.84cm
mp	massa do pêndulo	570g
Ip	momento de inércia em P	0,0039 Kg.m ²
mw	Massa da roda	34g
Iw	Momento de Inércia no centro de massa da roda	1.785e-5 Kg.m ²
rw	Raio da roda	32,4mm
R	Resistência do motor	10 ohms
Kb	Constante da Força Eletromotriz	0,337 V.s/rad
Kt	Constante de torque	0,3156 N.m/A

Fonte: Produção do próprio autor.

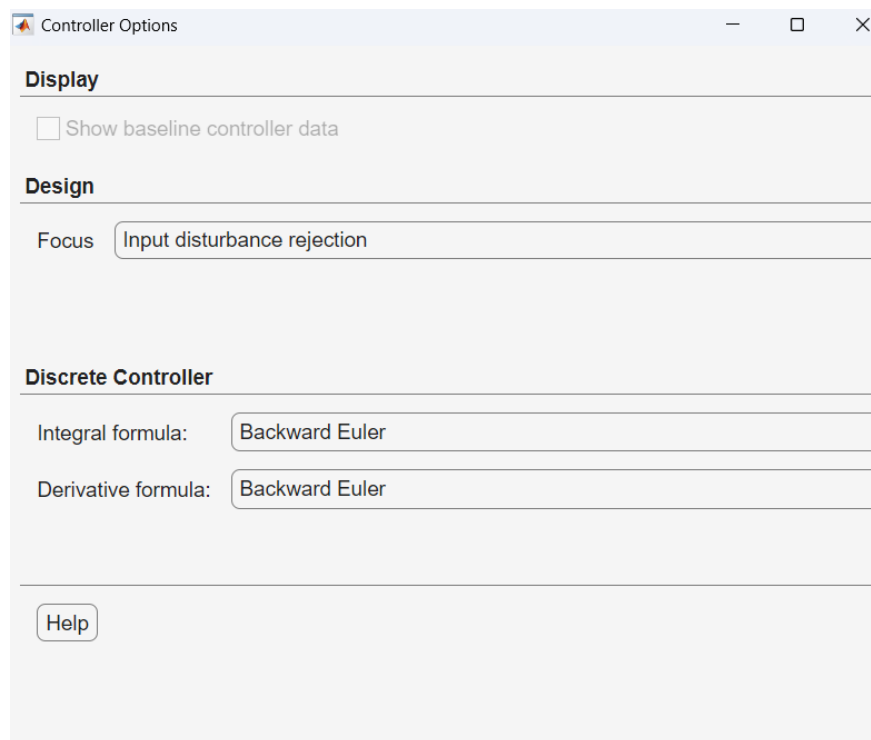
Após esses valores terem sido encontrados, utilizando o software MATLAB e utilizando as matrizes A, B, C e D, citadas na subseção 2.1. Modelagem do pêndulo invertido, e utilizando o código apresentado no APÊNDICE A, foi encontrada a função de transferência discreta da planta realizando a passagem dos espaços de estados contínuo para discreto e transformando em uma função de transferência de tempo discreto. Tal função é dada como:

$$Tf_{plant} = \frac{0.000517 z^2 + 0.0001759 z - 0.0003467}{z^3 - 2.674 z^2 + 2.35 z - 0.6798} . \quad (38)$$

Assim, é utilizado um aplicativo do MATLAB chamado PID TUNER que utiliza a planta como base e fornece o gráfico de resposta a uma entrada degrau e os valores dos ganhos conforme o usuário ajusta o tempo de resposta e o comportamento transitório desejado.

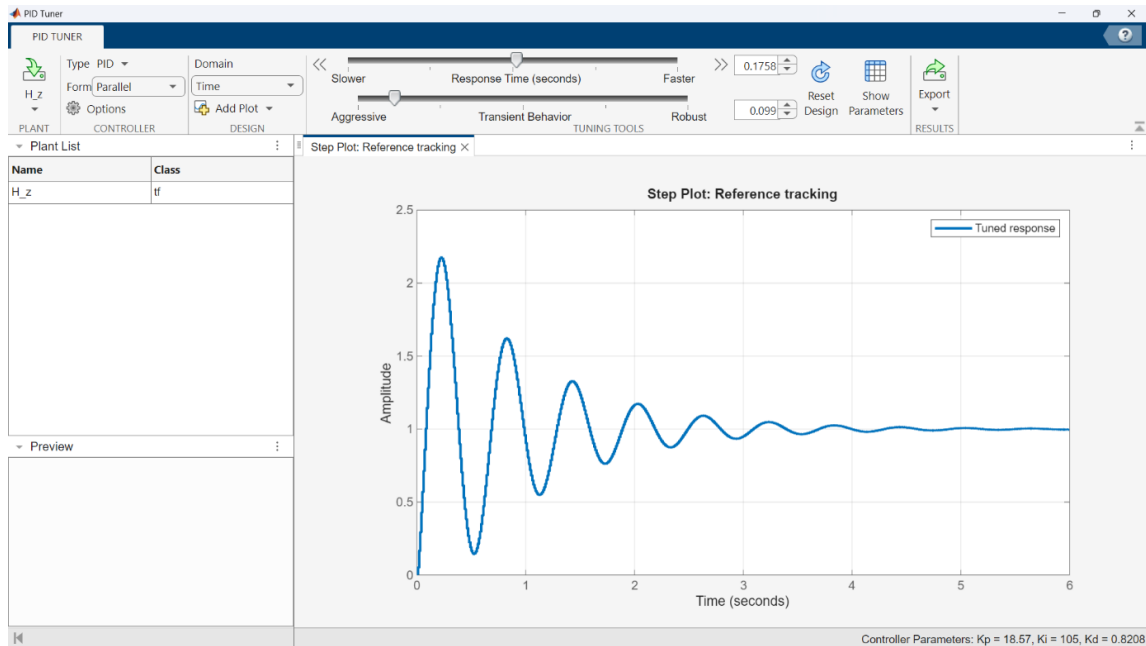
Para fins de ajuste de simulação, foram selecionadas as opções de “*focus*” para “*Input disturbance rejection*”, e para o controlador discreto foram selecionados em “*Integral formula*” e “*Derivative formula*” a opção “*Backward Euler*” em ambas. A Figura 28 apresenta a tela de opções do controlador e a Figura 29 mostra o aplicativo PID TUNER após obtenção dos valores PID.

Figura 28 - Menu de opções do controlador



Fonte: Produção do próprio autor.

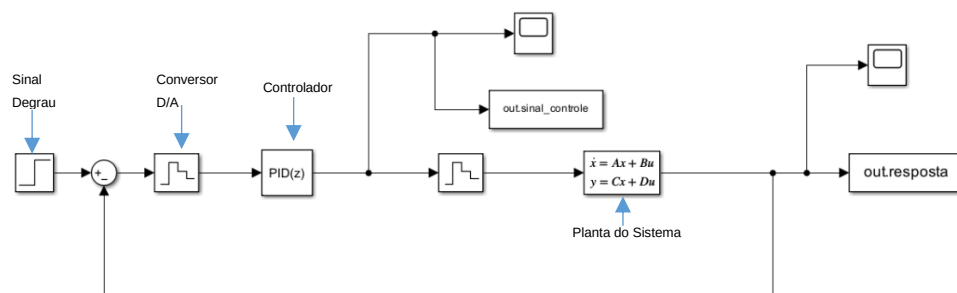
Figura 29 - Obtenção dos ganhos utilizando o PID TUNER.



Fonte: Produção do próprio autor.

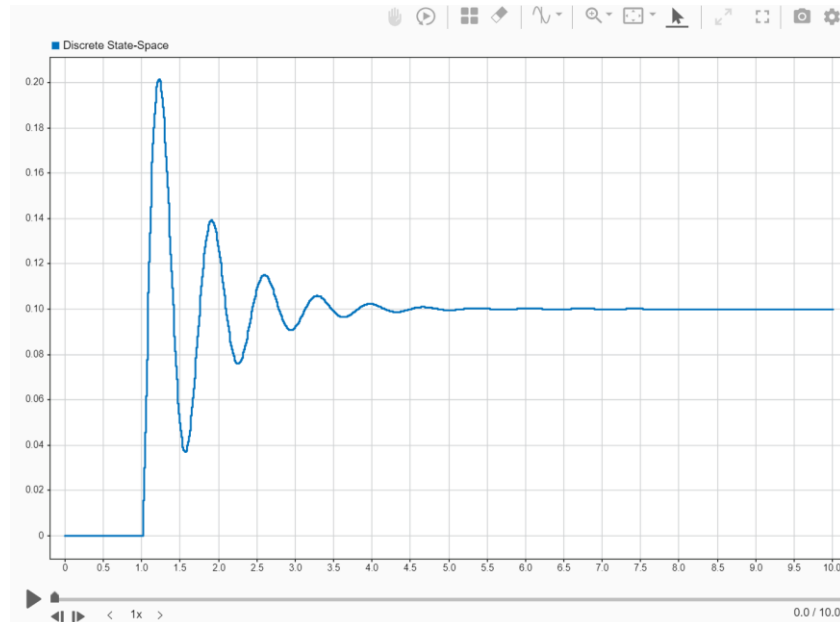
Com a aquisição dos valores dos ganhos e as matrizes da planta discreta, foi utilizado o aplicativo Simulink para simular o modelo obtido. Para isso, foram ajustados, previamente à simulação, uma entrada degrau de 0 para 0.1, os valores dos *zero-order hold* para um período amostral de 0.002 segundos, foram inseridos os ganhos no bloco do controlador PID discreto e alteradas, assim como nas opções do PID Tuner mostradas na Figura 26, as fórmulas para *Backward Euler*. Assim, é possível observar o modelo implementado no Simulink na Figura 30 e a resposta obtida com a simulação na Figura 31.

Figura 30 - Modelo implementado no Simulink.



Fonte: Produção do próprio autor.

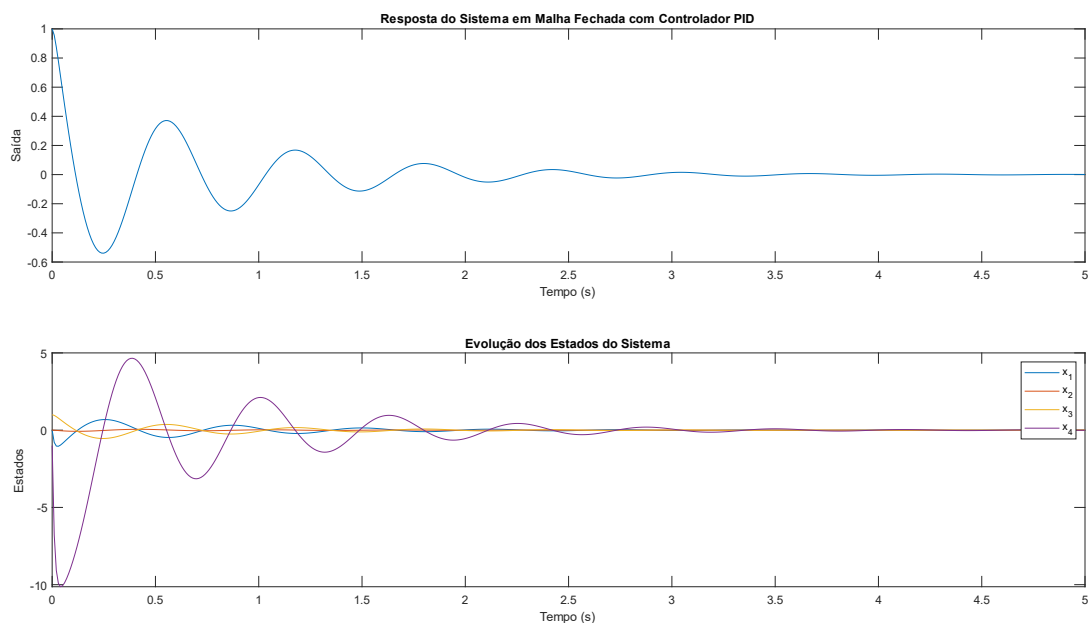
Figura 31 – Resposta a entrada degrau do modelo do Simulink.



Fonte: Produção do próprio autor.

Para analisar melhor o comportamento de evolução de estados e comprovar a estabilidade da resposta a um sinal degrau de entrada, foi utilizado o código do APÊNDICE B no MATLAB, cujos gráficos da resposta transitória à entrada degrau e a evolução dos estados do sistema gerados estão expostos na Figura 32.

Figura 32 - Resultados para entrada degrau.



Fonte: Produção do próprio autor.

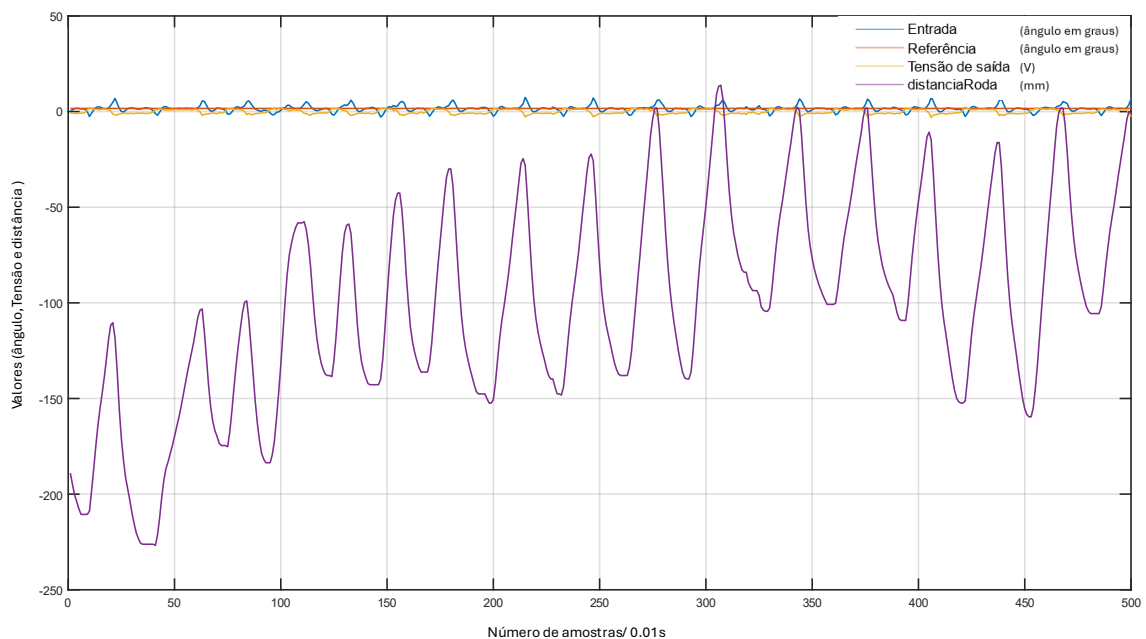
No APÊNDICE B, o código implementado realiza também a função de, a partir dos valores dos ganhos PID e da função de transferência discreta da planta, obter a função de transferência discreta do sistema em malha fechada. Tal função é dada por:

$$Tf_{cl} = \frac{0.04066 z^4 - 0.06942 z^3 + 0.004748 z^2 + 0.04145 z - 0.0172}{z^5 - 3.969 z^4 + 6.195 z^3 - 4.73 z^2 + 1.754 z - 0.2501} \quad (39)$$

Com os mesmos valores dos ganhos obtidos pelas simulações, tais valores foram implementados no sistema físico e foi observado o comportamento do robô, que caso apresentasse comportamento instável, ajustes empíricos seriam sendo feitos. Assim, após a gravação do código do APÊNDICE C e o sistema real ter sido implementado e testado o robô permaneceu em equilíbrio durante todo tempo de funcionamento.

Com auxílio do software MATLAB para gerar os gráficos, cujo código está presente no APÊNDICE D, a leitura dos parâmetros ângulos de *pitch*, *setpoint* (leituras dadas pelo sensor MPU 6050), tensão de saída (de 0 a 3.3V) e a posição do robô com relação à posição inicial (dada em milímetros), a qual se refere à posição em que o robô foi ligado (leitura do *encoder*), foram sendo enviados via *bluetooth* para o *software* que plotou os gráficos dos parâmetros do sistema real, cujos gráficos está exposta na Figura 33.

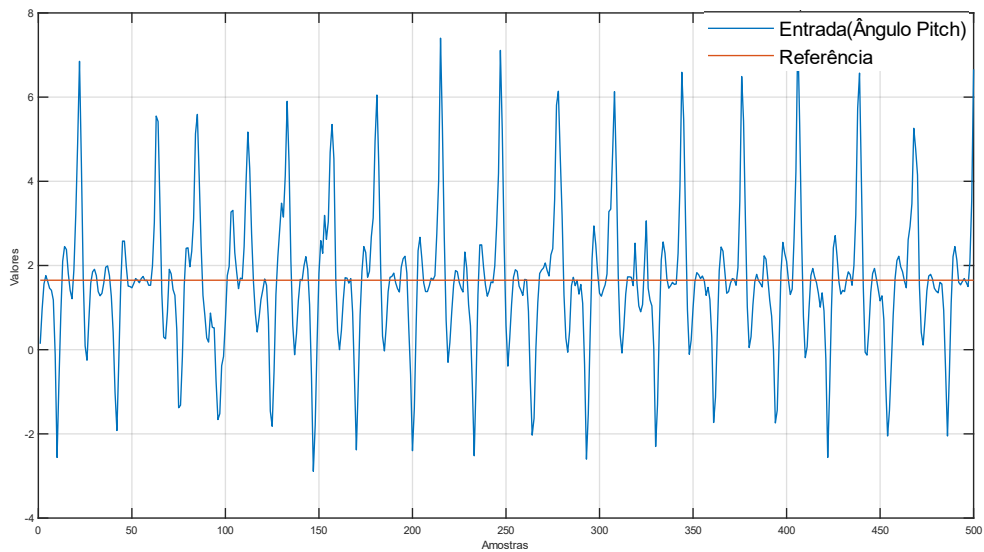
Figura 33 - Aquisição dos parâmetros do modelo físico no MATLAB.



Fonte: Produção do próprio autor.

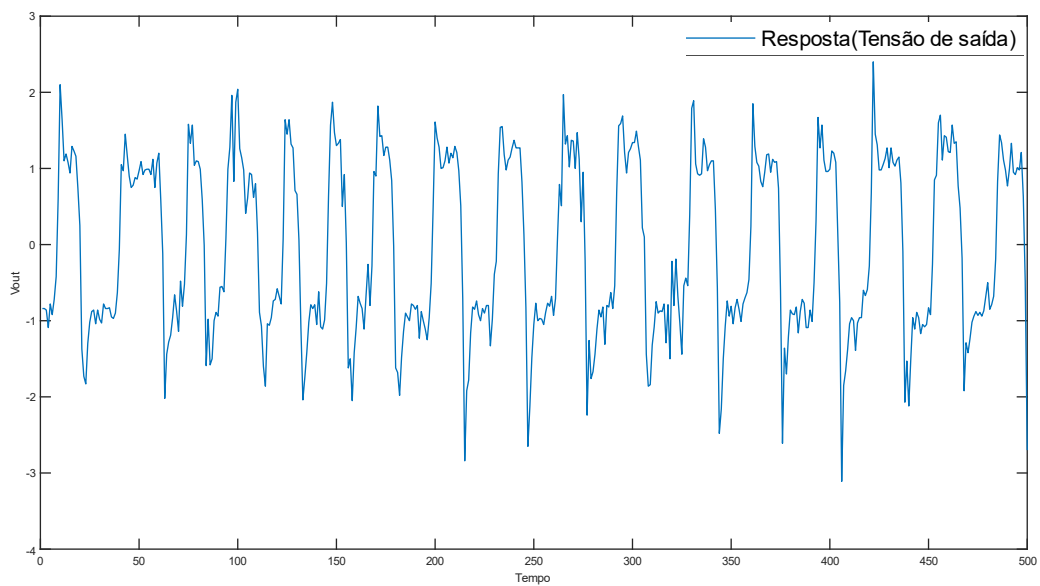
Para melhor visualização, as Figuras 34, 35 e 36 são respectivamente, gráficos dos parâmetros de ângulo de pitch pelo ângulo de referência (*setpoint*), tensão de saída (resposta do sistema), e ângulo de pitch pela tensão de saída.

Figura 34 - Entrada (Ângulo de *Pitch*) x Referência.



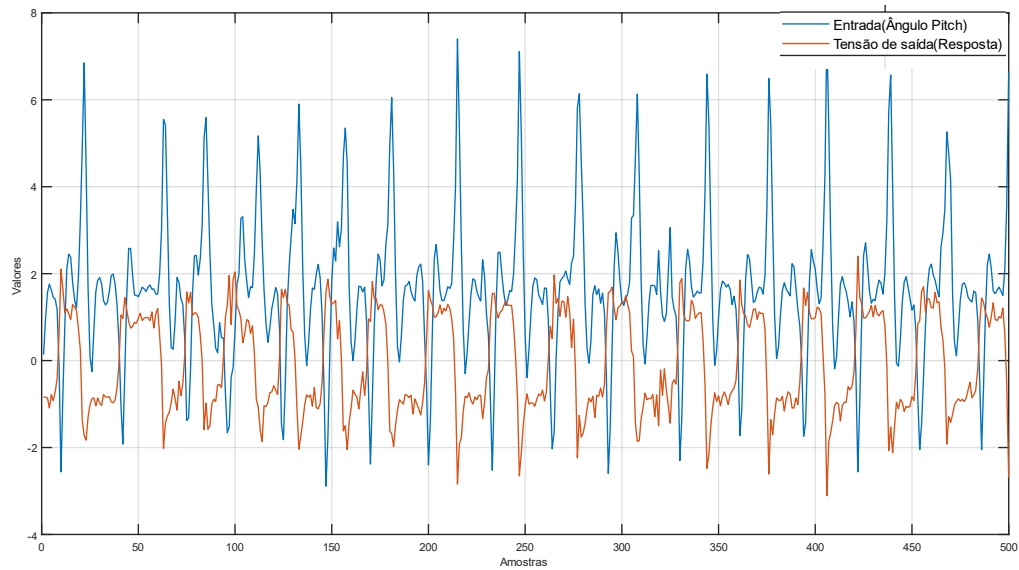
Fonte: Produção do próprio autor.

Figura 35 - Tensão de saída (Resposta) x Amostras.



Fonte: Produção do próprio autor.

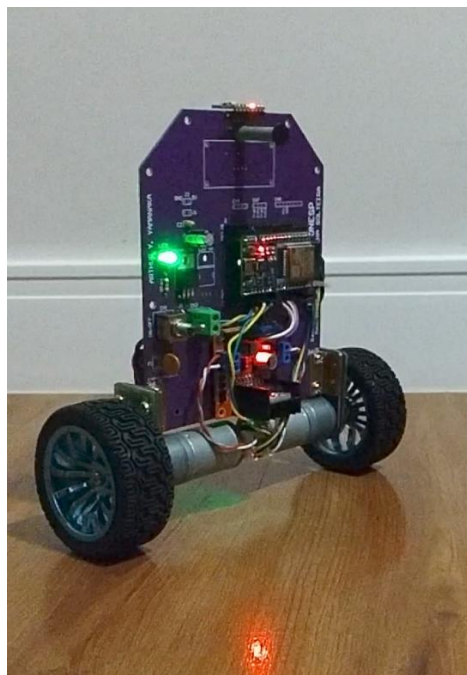
Figura 36 - Entrada (ângulo *pitch*) x Resposta (Tensão de saída).



Fonte: Produção do próprio autor.

Por fim, foi realizado o teste do robô em operação por 10 minutos para comprovar sua estabilidade e verificar se este permanece em equilíbrio durante um período considerável, o que comprovou o fato do robô pêndulo invertido ser funcional. A Figura 37 apresenta a imagem do robô pêndulo invertido durante seu funcionamento.

Figura 37 – Robô pêndulo invertido em funcionamento.



Fonte: Produção do próprio autor.

6 CONSIDERAÇÕES FINAIS

Conforme os resultados e testes realizados do controle implementado no sistema real, foi possível observar que o robô atingiu os objetivos esperados e funcionou de acordo com a teoria, mantendo-se em equilíbrio. Uma característica que foi observada foi que devido ao torque do motor e ao peso do robô, dependendo do valor do ângulo do pêndulo ou da força externa aplicada a ele, esse não é capaz de retornar à posição de equilíbrio devido às próprias limitações do sistema físico. Além disso, como o sistema é não linear, a faixa de ângulos em que o robô permanece equilibrado é limitada, o que exigiria métodos mais sofisticados para mantê-lo em equilíbrio sob condições mais desfavoráveis. Além disso, apesar de não impedir do robô de se manter sob a condição de equilíbrio, a transmissão de dados por bluetooth interferiu levemente no funcionamento do robô fazendo com que houvesse pequenas oscilações.

Com relação ao custo do robô, contando os valores de cada item, o valor total foi de 316,30 reais, descontando o valor de parafusos e porcas para fixação dos componentes. Apesar disso, é possível se utilizar materiais mais baratos, como por exemplo as baterias e os motores, que aumentaram o custo devido ao seu torque e a presença do encoder.

Como sugestões para melhorias e futuros trabalhos, pode-se implementar por exemplo: Implementação de sensores ultrassônicos para evitar a colisão do robô com outros objetos e o estudo de como a adição desses sensores interfere no sistema em malha fechada, utilizar algum método para suavizar ainda mais o movimento do robô para que este utilize movimentos mínimos para o equilíbrio, utilizar outro método de obtenção dos valores PID e utilizar outros métodos de controle no robô, como o LQR (*Linear Quadratic Regulator*) e o *fuzzy*.

Tomando em consideração os resultados e os objetivos do trabalho, foi possível obter, através do modelo matemático e da medição dos parâmetros do robô, um sistema estável e otimizado para que o robô se mantivesse em equilíbrio e comprovar que o modelo matemático foi condizente com o modelo físico. Além disso, foram utilizados materiais que são de baixo custo, mas que foram capazes de suprir as necessidades do projeto.

REFERÊNCIAS

ALIEXPRESS. **JGA25-370 Geared Motor Encoder, Speed Measuring Code Disc, alta potência, grande torque, Balance Trolley Motor, DC 6V, 12V, 24V**, [2024]. Disponível em: <<https://pt.aliexpress.com/item/1005007403396408.html>>. Acesso em: 14 ago. 2024.

ANDRADE, W. F.; SANTOS, P. H.; FILHO, A. M. S. T.; NETO, N. C. P. S. T.; GARCIA, G. C., Open-dulo: pêndulo invertido como plataforma educacional, **Revista Brasileira de Automática Industrial**, Ouro Preto, v.1, p. 2040-2045, 2019.

ARDUINO, **Basics of PWM (Pulse Width Modulation)**, [2024]. Disponível em: <https://docs.arduino.cc/learn/microcontrollers/analog-output/>. Acesso em: 28 ago. 2024.

ARDUINO. **Getting Started with Arduino IDE 2.0 | Arduino Documentation**, [2024] Disponível em: <https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2>. Acesso em: 28 ago. 2024.

ASSUNÇÃO, E. **CONTROLE LINEAR CONTROLE DISCRETO Sistemas Contínuos e Discretos no Tempo**. Vol.3. Ilha Solteira, 2020.

BUTTERFIELD, A. J.; SZYMANSKI, J. **A Dictionary of Electronics and Electrical Engineering**. 5. ed. OUP Oxford: Inglaterra, 2018.

ESPRESSIF. **ESP32-DevKitC V4.**, [2024]. Disponível em: https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/user_guide.html#get-started-esp32-devkitc-board-front. Acesso em: 28 ago. 2024.

ESPRESSIF. **ESP32-DevKitC V4.**, [2024]. Disponível em: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/_images/esp32-devkitC-v4-pinout.png. Acesso em: 28 ago. 2024.

HANDSON TECHNOLOGY. **L298N Dual H-Bridge Motor Driver**, [2017]. Disponível em: https://components101.com/sites/default/files/component_datasheet/L298N-Motor-Driver-Datasheet.pdf. Acesso em: 12 jun. 2024.

HOW TO MECHATRONICS, **L298N Motor Driver – Arduino Interface, How It Works, Codes, Schematics**, [2017]. Disponível em: <https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/>. Acesso em: 08 ago. 2024.

HOWARD, B.; BUSHNELL, L. Enhancing Linear System Theory Curriculum with an Inverted Pendulum Robot, *American Control Conference (ACC)*, Chicago, IL, USA, p. 2185-2192, 2015.

INSTITUTO NEWTON BRAGA. **Conheça o Barramento I2C (MIC098)**. Disponível em: <https://www.newtonbraga.com.br/microcontroladores/143-tecnologia/12085-conheca-o-barramento-i2c-mic098.html>, 2018. Acesso em: 28 ago. 2024.

INVENSENSE, **MPU-6000 and MPU-6050 Product Specification Revision 3.4**, 08/19/2013. Disponível em: <https://d229kd5ey79jzj.cloudfront.net/974/MPU-6000-Datasheet1.pdf>. Acesso em: 12 mai. 2024

KAFETZIS, I.; MOYSIS, L. Inverted Pendulum: A system with innumerable applications, **9th International Week Dedicated to Maths**, 2017.

LUNDBERG, K.H.; BARTON, T.W. History of Inverted-Pendulum Systems. **IFAC Proceedings Volumes**, v. 42, n. 24, p. 131–135, 2010.

MATHWORKS. **Tune PID Controller for LTI plant in the Live Editor – MATLAB**, [2019]. Disponível em: <https://www.mathworks.com/help/control/ref/tunepidcontroller.html>. Acesso em: 29 ago. 2024.

MATHWORKS. **Tune PID Controller to Favor Reference Tracking or Disturbance Rejection**, [2018]. Disponível em: <https://www.mathworks.com/help/slcontrol/ug/tune-pid-controller-to-balance-tracking-and-disturbance-rejection-performance.html>. Acesso em: 26 ago. 2024.

MINSEG. **MinSegShield M2V5 Dual Axis MinSeg Kit (New!)**, [2024]. Disponível em: <https://minseg.com/products/minsegshield-m2v5-dual-axis-balance-minseg-kit-new>. Acesso em: 20 dez. 2024

MOHAMED, F.; AL DAHOUD, A. **Integrated Development Environment "IDE" For Arduino**, Al Zaytoonah University, Amman, Jordan, 2018.

MOREIRA, M. M. A. C.; MOSCONI, D.; SILVEIRA, S. R.; SOUSA, F. S. I.; NORDI, T. M.; SOARES, I. N.; SIQUEIRA, A. A. G. CONTROLADOR PID APLICADO AO CONTROLE DE UM PÊNDULO INVERTIDO. **SIPGEM**, São Carlos, SP: EESC-USP, 2021.

OGATA, K., **Engenharia de Controle Moderno**. 5.ed. São Paulo (SP): Pearson Prentice Hall, 2011.

PAZ, R. A., **The Design of the PID Controller**, 2001. Disponível em: https://www.researchgate.net/publication/237528809_The_Design_of_the_PID_Controller. Acesso em: 17 jul. 2024.

PRECISION MINI DRIVES, **25mm Gear Motor With Encoder – 48mm Type Model NFP-JGA25-370-EM**, [2024]. Disponível em: <https://precisionminidrives.com/product/25mm-gear-motor-with-encoder-%E2%88%92-48mm-type-model-nfp-jga25-370-en>. Acesso em: 07 ago. 2024.

QI, H. Application of Fuzzy Control Algorithm in First-order Inverted Pendulum System. **Journal of Physics: Conference Series**, v. 2417, n. 1, p. 012038, 2022.

RI, S.; HUANG, J.; WANG, Y.; KIM, M; AN, S. Terminal Sliding Mode Control of Mobile Wheeled Inverted Pendulum System with Nonlinear Disturbance Observer. **Mathematical Problems in Engineering**, v. 2014, p. 1–8, 2014.

ROBOCORE, **Acelerômetro e Giroscópio MPU-6050**, [2024]. Disponível em: <https://www.robocore.net/sensor-robo/acelerometro-e-giroscopio-mpu6050>. Acesso em: 20 jun. 2024.

ROBOCORE, **Controlando Motores com o Módulo L298N- Tutoriais**, [2024]. Disponível em: <https://www.robocore.net/tutoriais/motor-dc-arduino-ponte-h-l298n>. Acesso em: 21 jun. 2024.

SAPUTRA, D., MA'ARIF, A.; MAGHFIROH, H.; CHOTIKUNNAN, P.; RAHMADHIA, S. N. Design and Application of PLC-based Speed Control for DC Motor Using PID with Identification System and MATLAB Tuner. **International Journal of Robotics and Control Systems**, v. 3, n. 2, p. 233–244, 7 abr. 2023.

TECHNOLOGIES, T. **Terasic - Robotic Kits - Self-Balancing Robot**, [2024]. Disponível em: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=1096>. Acesso em: 17 abr. 2024.

USP. **Controlador PID discreto 1.1 Objetivo**. [s.l.: s.n.]. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/1166397/mod_resource/content/1/Pratica1_SCII_2016.pdf>. Acesso em: 20 dez. 2024.

USP. **TRANSFORMAÇÃO ENTRE AS FORMAS ESPAÇO DOS ESTADOS E FUNÇÃO DE TRANSFERÊNCIA**, [2013]. Disponível em: <http://sites.poli.usp.br/d/pmr2400/11-%20Discretiza%C3%A7%C3%A3o%20temporal%20na%20forma%20SS.pdf>. Acesso em: 09 set. 2024.

APÊNDICE A - Código para obtenção do valor PID.

```

clear
clc

g=9.81;
h=0.1614;
l=0.12;
L=0.0684;
mp=0.570;
Ip=0.0039;
mw=0.034;
Iw=1.785e-5;
rw=0.0324;
R=10;
Kb=0.337;
Kt=0.3156;

q1=L*mp;
q2=Ip+L^2*mp;
q3=Kb*Kt/R;
q4=-mp-mw-Iw/rw^2;

A=[0 1 0 0;
    ((g*q1*q4)/(q1^2+q2*q4)) (q3*(q1-q4*rw)/((q1^2+q2*q4)*rw)) 0
    (q3*(-q1+q4*rw)/((q1^2-q2*q4)*rw^2));
    0 0 0 1;
    ((g*q1^2)/(q1^2+q2*q4)) -(q3*(q2+q1*rw)/((q1^2-q2*q4)*rw)) 0
    (q3*(q2+q1*rw)/((q1^2+q2*q4)*rw^2))];

B=[0;
    ((Kt*(q1-q4*rw))/(R*(q1^2-q2*q4)*rw));
    0;
    ((Kt*(-q2+q1*rw))/(R*(q1^2-q2*q4)*rw))];

C=[1 0 0 0];
D=0;

% Criar o sistema em espaço de estado contínuo
sis_cont = ss(A, B, C, D);

% Tempo de amostragem (escolha um valor, por exemplo T = 0.1)
T = 0.01;

% Converter para o sistema discreto
sis_disc = c2d(sis_cont, T);
tf_plant=tf(sis_disc);

% Extrair as matrizes do sistema discreto
[A_discr, B_discr, C_discr, D_discr] = ssdata(sis_disc);

```

```
% Mostrar as matrizes discretas
disp('Matriz A (discreta):');
disp(A_discr);
disp('Matriz B (discreta):');
disp(B_discr);
disp('Matriz C (discreta):');
disp(C_discr);
disp('Matriz D (discreta):');
disp(D_discr);

disp('Função de Transferência em Tempo Discreto:');
disp(tf_plant);

pidTuner(tf(sis_disc));
```

APÊNDICE B - Código respostas entrada degrau.

```

Kp = 18.57; % Ganho proporcional
Ki = 105; % Ganho integral
Kd = 0.82; % Ganho derivativo

% Constante de filtro para o termo derivativo
N = 100; % Constante de filtro alta para limitar o ganho em alta
frequência
ts=0.01;

% Criar o controlador PID com filtro de alta frequência
z=tf('z',ts);

PID1=Kp + Ki/z + Kd*N/(1 + N/z);

% Converter o controlador PID para espaço de estados

PID1_ss= ss(PID1);

% Extrair representações explícitas para o sistema combinado
[A1, B1, C1, D1] = ssdata(PID1_ss);
[A2, B2, C2, D2] = ssdata(sis_cont);

% Ajustar as dimensões das matrizes para combinar os sistemas
n1 = size(A1, 1); % Número de estados do controlador PID
n2 = size(A2, 1); % Número de estados do sistema original

A_combined = [A1, zeros(n1, n2); B2*C1, A2];
B_combined = [B1; B2*D1];
C_combined = [D2*C1, C2];
D_combined = D2*D1;

% Criar o sistema em espaço de estados combinado
sys_combined = ss(A_combined, B_combined, C_combined, D_combined);

% Criar o sistema em malha fechada
sys_cl = feedback(sys_combined, 1);

% Gera a função de transferência discreta do sistema em malha
fechada
tf_cl=tf(c2d(sys_cl,ts));

% Definir as condições iniciais
x0 = [1; -1; 0.5; -0.5]; % Condições iniciais não nulas

% Tempo de simulação

```

```

t = 0:0.01:5; % 10m segundos

% Definir condições iniciais para o sistema combinado
x0_combined = [zeros(n1, 1); x0];

% Simular o sistema em malha fechada
[y, t, x] = lsim(sys_cl, zeros(size(t)), t, x0_combined);

% Plot dos resultados
figure;
subplot(2,1,1);
plot(t, y);
title('Resposta do Sistema em Malha Fechada com Controlador PID');
xlabel('Tempo (s)');
ylabel('Saída');

subplot(2,1,2);
plot(t, x(:,3), t, x(:,4), t, x(:,1), t, x(:,2));
title('Evolução dos Estados do Sistema');
xlabel('Tempo (s)');
ylabel('Estados');
legend('x_1', 'x_2', 'x_3', 'x_4');

disp(tf_cl);

```

APÊNDICE C - Código da programação do robô.

```

#include <PID_v1.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include <BluetoothSerial.h>

BluetoothSerial SerialBT;

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

MPU6050 mpu;

#define ENCODERINPUT1 4
#define ENCODERINPUT2 15
volatile signed int position=0 ;
float d;

bool dmpReady = false;
uint16_t packetSize;
uint8_t fifoBuffer[64];

Quaternion q;
VectorFloat gravity;
float ypr[3];
VectorInt16 gy;

#define PID_MIN_LIMIT -255
#define PID_MAX_LIMIT 255
#define PID_SAMPLE_TIME_IN_MILLI 10
#define SETPOINT_PITCH_ANGLE_OFFSET 1.65
#define MIN_DISTANCE 20

#define MIN_ABSOLUTE_SPEED 0
int a=0;

double setpointPitchAngle = SETPOINT_PITCH_ANGLE_OFFSET;
double pitchGyroAngle = 0;
double pitchPIDOutput = 0;

double setpointYawRate = 0;
double yawGyroRate = 0;
double yawPIDOutput = 0;

double PWMpercent;

```

```

#define PID_PITCH_KP 18.57
#define PID_PITCH_KI 105
#define PID_PITCH_KD 0.82

#define PID_YAW_KP 0
#define PID_YAW_KI 0
#define PID_YAW_KD 0

PID pitchPID(&pitchGyroAngle, &pitchPIDOutput, &setpointPitchAngle,
PID_PITCH_KP, PID_PITCH_KI, PID_PITCH_KD, DIRECT);
PID yawPID(&yawGyroRate, &yawPIDOutput, &setpointYawRate,
PID_YAW_KP, PID_YAW_KI, PID_YAW_KD, DIRECT);

int motor1Pin1 = 17;
int motor1Pin2 = 16;
int motor2Pin1 = 18;
int motor2Pin2 = 19;

int b;

void setupPID()
{
    pitchPID.SetOutputLimits(PID_MIN_LIMIT, PID_MAX_LIMIT);
    pitchPID.SetMode(AUTOMATIC);
    pitchPID.SetSampleTime(PID_SAMPLE_TIME_IN_MILLI);

    yawPID.SetOutputLimits(PID_MIN_LIMIT, PID_MAX_LIMIT);
    yawPID.SetMode(AUTOMATIC);
    yawPID.SetSampleTime(PID_SAMPLE_TIME_IN_MILLI);
}

void setupMotors()
{
    pinMode(motor1Pin1, OUTPUT);
    pinMode(motor1Pin2, OUTPUT);
    pinMode(motor2Pin1, OUTPUT);
    pinMode(motor2Pin2, OUTPUT);
    rotateMotor(0, 0);
}

void setupMPU()
{
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin(22, 21);
    Wire.setClock(400000);
    #endif

```

```

mpu.initialize();
pinMode(INTERRUPT_PIN, INPUT);
if (mpu.dmpInitialize() == 0)
{
    mpu.setXAccelOffset(-338);
    mpu.setYAccelOffset(1989);
    mpu.setZAccelOffset(1902);
    mpu.setXGyroOffset(18);
    mpu.setYGyroOffset(-39);
    mpu.setZGyroOffset(45);
    mpu.setDMPEnabled(true);
    dmpReady = true;
    packetSize = mpu.dmpGetFIFOPacketSize();
}
}

void pos(){
    int e2=digitalRead(ENCODERINPUT2);
    if (e2>0){
        d=0.6*(position++);
    }
    else{
        d=0.6*(position--);
    }
}

void setup()

{
    pinMode(ENCODERINPUT1, INPUT);
    pinMode(ENCODERINPUT2, INPUT);
    attachInterrupt(ENCODERINPUT1,pos,RISING);

    b=0;

    Serial.begin(115200);
    SerialBT.begin("ESP32_BT");
    setupMotors();
    setupMPU();
    setupPID();

    SerialBT.println("pitchGyroAngle,setpointPitchAngle,PWMpercent,distance ");
}

void loop()

```

```

{
    static unsigned long lastSerialTime = 0;
    unsigned long currentMillis = millis();
    const unsigned long serialInterval = 100;
    b=b+1;

    if (!dmpReady) return;

    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer))
    {
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
        mpu.dmpGetGyro(&gy, fifoBuffer);

        yawGyroRate = gy.z;
        pitchGyroAngle = ypr[1] * 180/M_PI;

        pitchPID.Compute();
        yawPID.Compute();

        rotateMotor(pitchPIDOutput + yawPIDOutput, pitchPIDOutput -
yawPIDOutput);

        PWMpercent=pitchPIDOutput/2.55;

        if (currentMillis - lastSerialTime >= serialInterval) {
            lastSerialTime = currentMillis;

            // Envie os dados pela porta serial

            SerialBT.print(pitchGyroAngle);
            SerialBT.print(",");
            SerialBT.print(setpointPitchAngle);
            SerialBT.print(",");
            SerialBT.print(PWMpercent*0.05);
            SerialBT.print(",");
            SerialBT.println(d);
        }
    }
}

void rotateMotor(int speed1, int speed2)
{
    //rotaciona os motores individualmente(speed 1 e speed 2)

```

```
    speed1 = speed1 + MIN_ABSOLUTE_SPEED;
    speed2 = speed2 + MIN_ABSOLUTE_SPEED;

    if (speed1 < 0)
    {
        analogWrite(motor1Pin1, 0);
        analogWrite(motor1Pin2, -speed1);
    }
    else if (speed1 >= 0)
    {
        analogWrite(motor1Pin1, speed1);
        analogWrite(motor1Pin2, 0);
    }

    if (speed2 < 0)
    {
        analogWrite(motor2Pin1, 0);
        analogWrite(motor2Pin2, -speed2);
    }
    else if (speed2 >= 0)
    {
        analogWrite(motor2Pin1, speed2);
        analogWrite(motor2Pin2, 0);
    }
}
```

APÊNDICE D - Código de aquisição de dados no Matlab.

```

serialportlist
serialPort = 'COM3'; % Porta serial do ESP
baudRate = 115200; % Frequência estabelecida no ARDUINO IDE

% Abrir conexão Serial
s = serialport(serialPort, baudRate);
configureTerminator(s, 'LF');

% Variáveis de início
numSamples = 500; % Número de dados colhidos de cada informação
data = zeros(numSamples, 4); % Assumindo 4 valores enviados pelo
ESP32 (pitchAngle, Setpoint, VOUT, distancia)

% Configure o plot
figure;
h = plot(data);

xlabel('Sample');
ylabel('Value');
legend('Pitch Angle', 'Setpoint', 'V_OUT', 'distanciaRoda' );
grid on;

% Loop principal
for i = 1:numSamples

    line = readline(s);

    values = str2double(strsplit(line, ','));

    % Checagem do número de informações
    if numel(values) == 4
        % Atualiza os dados lidos
        data(i, :) = values;
    else
        % Mostra um sinal de que o número de informações não
        corresponde a 4
        warning('Unexpected number of values received from
        Arduino. ');
        % Continua a operação ou finaliza o programa
        continue;
    end

    % Atualiza o plot
    set(h(1), 'YData', data(:, 1)); % PitchAngle
    set(h(2), 'YData', data(:, 2)); % SetPoint
    set(h(3), 'YData', data(:, 3)); % V_OUT
    set(h(4), 'YData', data(:, 4)); % distanciaRoda
    drawnow;
end

```

```
end
Vout=data(:,3);

pitch=data(:,1);

PitchAngle= fillmissing(pitch,'constant',0);
V0 = fillmissing(Vout,'constant',0);
figure;
plot()
% Finaliza a conexão serial
clear s;

% Salva o arquivo
save('data.mat', 'data');
```

APÊNDICE E – Esquemático do circuito do robô pêndulo invertido.

