http://journal.unoeste.br/index.php/ce/index
DOI: 10.5747/ce.2025.v17.e435
ISSN on-line 2178-8332

Colloquium Exactarum

Submetido: 05/12/2024 Revisado: 26/07/2025 Aceito: 25/08/2025

DESENVOLVIMENTO DE UM CONTROLADOR PID PARA MELHORAR A ODOMETRIA NO ROBÔ LEGO MINDSTORM EV3

Development of a PID Controller to Improve Odometry on the LEGO Mindstorm EV3 Robot

Brendon Wilyen de Queiroz Souza, Martha Ximena Torres Delgado, Emmanuel Norberto Ribeiro dos Santos

[1]**Universidade Estadual de Santa Cruz, Departamento de ciências exatas e computação – DEC.**
e-mail: bwqsouza@gmail.com; mxtd@uesc.br

RESUMO – Na área da robótica móvel, a precisão na navegação e localização é crucial para o desempenho eficaz dos robôs e, com isso, o desenvolvimento de algoritmos de controle se torna fundamental. O controlador proporcional integral e derivativo (PID) é o algoritmo de controle mais utilizado na indústria devido sua eficácia, simplicidade e robustez. O presente trabalho explora a implementação de um controlador PID em um robô construído na plataforma LEGO Mindstorm EV3, com o objetivo de melhorar a precisão de sua navegação e, por consequência, aprimorar os resultados de um algoritmo de localização baseado em odometria. Para isso, foi desenvolvido o controlador PID e feita a união com o algoritmo baseado em odometria. Além disso, foram realizados testes práticos para validar a eficácia da plataforma robótica. Esses testes foram executados em diferentes trajetórias permitindo calcular os erros médios e o desvio padrão entre a posição estimada e a posição real. Os resultados mostraram que, com a inclusão do controlador PID, os desvios médios foram reduzidos para valores entre 12 mm no eixo X e 52 mm no eixo Y. Esses resultados evidenciam que a integração do PID reduziu significativamente as distorções acumuladas na odometria, aumentando a precisão da navegação. Este estudo avança as pesquisas relacionadas a área de robótica móvel no curso de Ciência da Computação da Universidade Estadual de Santa Cruz, servindo como incentivo para futuras pesquisas no campo da robótica.
Palavras-chave: PID; Odometria; robótica.

ABSTRACT – In the area of mobile robotics, precision in navigation and localization is crucial for the effective performance of robots and, therefore, the development of control algorithms becomes fundamental. The proportional integral and derivative controller (PID) is the most used control algorithm in the industry due to its effectiveness, simplicity and robustness. The present work explores the implementation of a PID controller in a robot built on the LEGO Mindstorm EV3 platform, with the aim of improving the accuracy of its navigation and, consequently, improving the results of a localization algorithm based on odometry. For this, the PID controller was developed and combined with the odometry-based algorithm. Furthermore, practical tests were carried out to validate the effectiveness of the robotic platform. These tests were performed on different trajectories, allowing the calculation of the mean error and standard deviation between the estimated and actual positions. The results showed that, with the inclusion of the PID controller, the average deviations were reduced to values between 12 mm on the X-axis and 52 mm on the Y-axis. These results demonstrate that the integration of the PID significantly reduced accumulated odometry distortions, increasing navigation accuracy. This study advances research related to the

area of mobile robotics in the Computer Science course at the Universidade Estadual de Santa Cruz, serving as an incentive for future research in the field of robotics.
Keywords: PID; Odometry; robotic.

1. INTRODUÇÃO

O presente artigo tem como objetivo apresentar o desenvolvimento de um controlador PID (Proporcional, Integral e Derivativo) e implementar no robô EV3, com o propósito de melhorar a trajetória que o algoritmo de localização baseado em odometria proporciona. O trabalho foi desenvolvido entre agosto e dezembro de 2023 como uma continuidade da pesquisa desenvolvida por Santos (2023) que resultou na construção de um algoritmo de localização baseado em odometria.

Ao longo do tempo, a definição da palavra "robô" passou por diversas alterações. Em seu significado original, "robô" é uma derivação da palavra tcheca *robotnik* que significa "servo", combinada com a outra palavra tcheca *rabota* que significa trabalho obrigatório (Capek, 2021). Em sua obra ficcional, Capek descreve um robô como um ser vivo artificial e inteligente capaz de realizar trabalhos que os humanos não gostariam de fazer (Fernandes, 2003).

Atualmente, a ISO 8373:2021 define os robôs como "mecanismo acionado programado com grau de autonomia para realizar locomoção, manipulação ou posicionamento" (ISO, 2021). De acordo com o dicionário Michaellis (2025), robô é definido como "Aparelho automático, com aspecto humanoide, capaz de se movimentar e executar diferentes tarefas, inclusive algumas geralmente feitas pelo homem". Como a definição tornou-se muito ampla, foi necessário classificá-los levando em consideração diversos critérios, como: locomoção, autonomia, tipo dos sensores, sistemas de controle, tipo de deslocamento e até por seu comportamento (Marcomini, 2023). O robô móvel é uma dessas classes de robôs. Eles são chamados dessa forma pois possuem a capacidade de se movimentar. Geralmente são desenvolvidos sob uma base sobre rodas e podem ser autônomos (capaz de se movimentar sozinho e se localizar no ambiente) ou não.

Um dos desafios da robótica móvel é navegar e mapear o ambiente no qual o robô está inserido, e a localização do dispositivo, ou seja, a determinação da posição do robô no ambiente em um instante de tempo, é fundamental para garantir uma navegação mais precisa e a autonomia do sistema. A odometria desempenha um papel importante nesse processo. O método visa a determinação do robô através da medida dos deslocamentos incrementais das rodas do robô em relação a um eixo fixo, permitindo assim que possamos localizar o robô em tempo real (Bezerra; Alsina; Medeiros, 2003).

A indústria enxergou na robótica uma possível solução para a redução no tempo de produção e diminuição nos custos de mão de obra, ao perceber que a automação robótica poderia aumentar a eficiência da esteira de produção, tornando esse processo mais rápido e econômico. Hoje, a robótica móvel ganhou destaque após o surgimento de robôs aspiradores, carros autônomos e outros AGVs (Automated Guided Vehicle - Veículo Guiado Automaticamente) (Bastos, 2019).

Com o avanço da robótica móvel ao longo dos anos e suas diversas aplicações no setor industrial, estudos e desenvolvimento de algoritmos de controle mais precisos tiveram sua relevância elevada, especialmente devido à sua importância na precisão da localização desses dispositivos. A utilização de controladores PID (proporcional, integral e derivativo) destaca-se por possuir a capacidade de fornecer feedbacks, eliminar erros estacionários e antecipar ações futuras (Bastos, 2019).

As demais seções deste trabalho estão configuradas da seguinte maneira: A seção 2 introduz os conceitos de controladores e suas classificações; Na seção 3 é apresentado a técnica de localização de robôs móveis utilizada neste trabalho, a odometria; Na seção 4 são apresentados os materiais e métodos utilizados; O desenvolvimento prático do trabalho é exposto na Seção 5, seguido pela análise dos resultados na Seção 6; Por fim a seção 7 consolida as conclusões do estudo.

2. CONTROLADORES

Controladores são dispositivos projetados para receber informações de entrada, compará-las com um valor de referência, chamado *setpoint*, e, com base nessa comparação, gerar um sinal de saída que visa influenciar ou corrigir o comportamento de um sistema. Eles são componentes essenciais em sistemas automatizados, onde sua capacidade de monitorar e ajustar variáveis permite manter um sistema funcionando de maneira eficiente e precisa. Os controladores são a ponte entre o mundo real e os sistemas de controle, garantindo que as saídas estejam alinhadas com os objetivos estabelecidos.

Os controladores podem ser classificados em dois grupos, dependendo da natureza da relação entre as entradas e as saídas. O primeiro grupo, é conhecido como controlador de malha fechada, no qual há uma realimentação do sistema de controle a partir do valor de saída obtido anteriormente. Essa característica confere ao controlador maior precisão, mas pode torná-lo suscetível a instabilidades e oscilações. Por outro lado, o segundo grupo, corresponde aos controladores de malha aberta, nos quais não há realimentação do sistema. Um exemplo básico de sistema de controle em malha aberta é um automóvel, que acelera somente quando o pedal é pressionado (Marcomini, 2023; Mathworks, 2023). Embora esses controladores sejam mais simples, eles podem ser menos precisos em situações que demandam um controle mais refinado (Distefano; Stubberud; Williams, 2014). Na seção 2.1 são apresentados os componentes do controlador PID.

2.1. CONTROLADOR PROPORCIONAL, INTEGRAL E DERIVATIVO (PID)

O controlador PID é um sistema de controle que opera em malha fechada, simples e amplamente utilizado na indústria devido sua fácil implementação, robustez e baixo custo (). é utilizado desde o período entre as guerras mundiais e, desde então, é a classe de controladores mais utilizada nos sistemas industriais (Ferraz, 2001). Entretanto, é importante ressaltar que, embora o PID seja uma escolha popular, não é um controlador ótimo, mas que com o ajuste fino de suas variáveis consegue desempenhar satisfatoriamente (Gentilho, 2014).

A estrutura do controlador PID, descrita na Equação 1, é composta pela ação de três tipos de controladores, que são classificados de acordo com a ação de controle, sendo eles: Proporcional, Integral e Derivativo. Essa estrutura permite que qualquer controlador dessa família seja criado desativando alguma dessas estruturas, por exemplo: se o sistema precisa de um controlador proporcional, basta zerar os coeficientes K_i e K_d. Os controladores que compõem o PID são tratados nas seções 2.1.1, 2.1.2 e 2.1.3.

$$u(t) = K_p * e(t) + K_i \int_0^1 e(t)dt + K_d * \frac{d}{dt} * e(t) \qquad (1)$$

O controlador PID também pode ser descrito matematicamente por meio da Equação 2 U_{out} é a saída do controlador, ou seja, é o sinal de ajuste que o controlador envia para influenciar no comportamento ou estado do sitema. $1/T_iS$ representa a ação integral, e $1/T_dS$ representa a ação derivativa. “S” é a variável de Laplace.

$$U_{out} = K_p(1 + \frac{1}{T_iS} + T_dS) \qquad (2)$$

2.1.1. CONTROLADOR PROPORCIONAL

O controlador proporcional, age proporcionalmente ao erro atual entre o valor desejado (*setpoint*) e a saída real do sistema. Quanto maior o erro, maior será a ação corretiva do termo proporcional. Esta ação é responsável por determinar a resposta imediata do sistema a variações no *setpoint*.

$$u(t) = K_p * e(t) \qquad (3)$$

Um valor elevado do coeficiente proporcional ou ganho (K_p) resultará em uma correção mais rápida, reduzindo o tempo de resposta do sistema, mas pode aumentar o risco de *overshoot* (quando a saída ultrapassa o *setpoint* antes de se estabilizar). Um valor muito baixo de K_p} pode levar a uma resposta lenta e insuficiente à variação do *setpoint*, visto que este controlador não é capaz de eliminar o erro em estado estacionário (Murrill, 1981).

Quando K_p assume valores positivos, o controlador reduz a ação u(t) quando há um aumento na variável controlada. Por exemplo, em um sistema de controle de velocidade cujo erro calculado foi de -5, ou seja, a velocidade está a 5 u.m do *setpoint* e o ganho proporcional definido como 3, será produzido um valor proporcional de -15, o que deve reduzir o acionamento do motor. Esta ação é chamada de reversa.

2.1.2. CONTROLADOR INTEGRAL

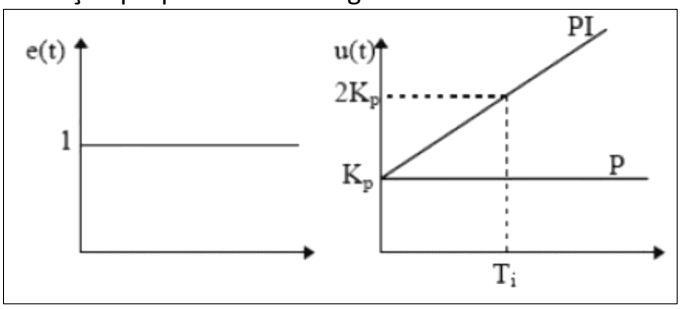
A ação de controle integral, também chamada de controle de reset, é responsável por eliminar ou reduzir o erro estacionário. Este controlador geralmente é utilizado em conjunto com o controlador

proporcional (Murrill, 1981), e quando atuam juntos, a ação proporcional-integral corrige os erros acumulados ao longo do tempo (Gentilho, 2014). A Equação 4 apresenta o controlador como função do erro e da integral do erro, resultando assim em um controlador proporcional - integral.

$$u(t) = K_i \int_0^t e(t)dt \qquad (4)$$

O tempo integral (T_i) ou tempo de reset, tem dimensão de tempo geralmente expresso em minutos e torna os controladores PI mais conservadores em suas oscilações (Tannuri, 2023). De um lado, a ação integral reduz o erro estacionário, de outro, aumenta o tempo necessário para a estabilidade. A Figura 1 apresenta a comparação entre a ação proporcional e a proporcional integral.

Figura 1. Comparação entre a ação proporcional e integral

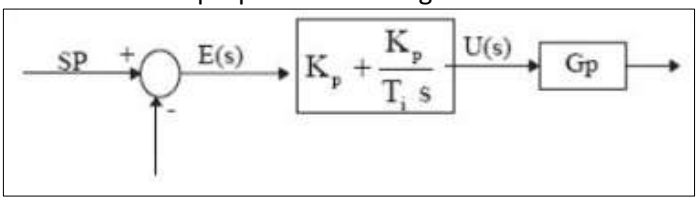


Fonte: (Ziegler; Nichols, 1942).

Devido à propensão deste modo de controle de diminuir a estabilidade do sistema, é necessário ajustar e reduzir o ganho associado à ação proporcional. A ação integral é utilizada sempre que o controlador proporcional não seja capaz de reduzir o erro estacionário a um nível tolerável. Na Figura 2 é apresentado o diagrama de blocos que ilustra a ação proporcional-integral.

Um valor muito alto do coeficiente Integral (K_i), pode resultar em instabilidades e oscilações nas respostas do sistema. Esse efeito pode ser reduzido por meio da sintonia do controlador.

Figura 2. Diagrama de blocos do controlador proporcional - integral



Fonte: (Ziegler; Nichols, 1942).

2.1.3. CONTROLADOR DERIVATIVO

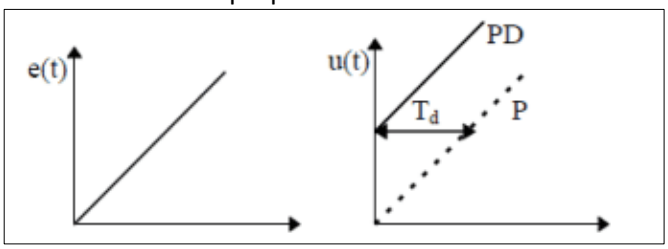
A ação derivativa tem um caráter antecipatório e é responsável pela prevenção das oscilações no sistema. Essa ação, responde ao sistema com um sinal de controle proporcional à derivada do sinal de erro, ou seja, aplica uma ação baseada na tendência evolutiva do erro (Gentilho, 2014).

Como pode-se observar na Equação 5, dado que o sinal de controle é proporcional à taxa de variação do erro, a ação derivativa não é viável como um modo de controle isolado. Isso se deve ao fato de que essa ação reage apenas a mudanças transitórias. Caso fosse usada sozinha, em situações em que o erro é constante, o valor do controle resultante seria nulo.

$$u(t) = K_d * \frac{d}{dt} * e(t) \qquad (5)$$

A adição do controle derivativo ao proporcional cria um controlador muito mais sensível. Isso ocorre porque o modo derivativo, ao reagir à taxa de variação do erro, permite que as correções ocorram antes que o erro aumente significativamente. A Figura 3 ilustra a ação proporcional-derivativa.

Figura 3. Diagrama de blocos do controlador proporcional - derivativo



Fonte: (Ziegler; Nichols, 1942).

3. NAVEGAÇÃO E LOCALIZAÇÃO DE ROBÔS MÓVEIS

A navegação em robótica móvel, é o processo que permite ao robô se mover no seu ambiente de trabalho, de uma posição inicial até uma posição final (Bezerra; Alsina; Medeiros, 2003). A partir das informações coletadas por sensores, o robô deve ser capaz de identificar sua localização ao longo do percurso.

A localização é um item fundamental para o sistema de navegação, já que para navegar de forma segura e com precisão o robô precisa saber a sua posição em relação ao seu espaço de trabalho. Localizar um robô móvel é determinar sua pose, ou seja, sua posição e orientação no espaço em um determinado instante de tempo (Nascimento, 2014). De acordo com Bezerra; Alsina; Medeiros (2003), os métodos de localização podem ser classificados em duas categorias: métodos de localização relativa e métodos de localização absoluta.

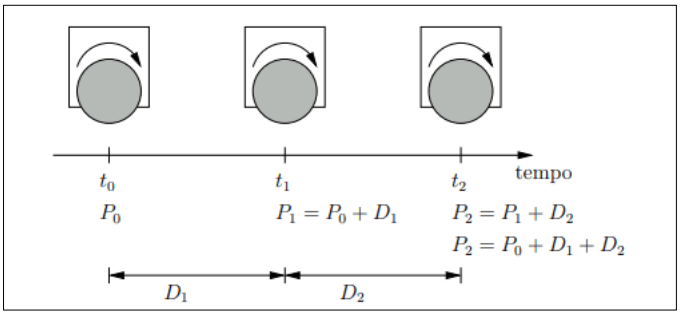
Os métodos de localização relativa, consistem em métodos que determinam a posição atual da plataforma móvel em relação a um ponto fixo. Para isso, tais métodos utilizam dados de sensores, como por exemplo, um odômetro, a fim de encontrar a diferença entre uma nova leitura e a leitura anterior. Dessa maneira, a posição e orientação do robô é obtida a partir de uma estimativa baseada no deslocamento do robô em relação ao ponto em que os atuadores foram ativados. São exemplos a odometria e a navegação inercial (Fernandes, 2003).

Já os métodos de localização absoluta, não determina a pose atual da plataforma móvel a partir de uma relação entre a localização atual e a inicial, ou seja, a pose atual não depende mais de uma leitura anterior dos sensores, apenas das informações atuais dos sensores para determinar sua localização em relação a um referencial fixo que não varia com o tempo (Nascimento, 2014). Exemplos destes métodos incluem o balizamento ativo, posicionamento baseado em mapas e o reconhecimento de "*landmarks*".

3.1. ODOMETRIA

A odometria é um método de localização relativa que consiste em determinar a posição e orientação do robô através da integração dos deslocamentos incrementais das rodas em relação a um referencial fixo (Pinto, 2016). Um exemplo desse método foi apresentado na Figura 4, considerando um robô que se move em uma trajetória em linha reta.

Figura 4. Deslocamento de um robô utilizando odometria



Fonte: (Bezerra, 2004).

No instante t_0 o robô se encontra na posição P_0. No instante t_1 o robô se encontra na posição P_1, que é calculada a partir da posição p0 somada ao deslocamento D_1}. De forma análoga, ocorre em P2. Assim, a posição atual do robô é calculada baseado nos deslocamentos efetuados por ele em relação à posição inicial (Pinto, 2016).

As informações do deslocamento do robô são coletadas por um sensor que consegue medir a rotação das rodas, o *encoder*. Os encoders são capazes de codificar as rotações em pulsos, o que possibilita determinar a posição angular, a velocidade e direção do movimento (Camilo,2025; Rezende et al.,2024). Existem alguns tipos de *encoders*, os mais conhecidos são os *encoders* óptico e o magnético, mas o funcionamento é semelhante em ambos os tipos (Bastos, 2019). O *encoder* óptico é composto por um fotoemissor e um fotorreceptor, em que o funcionamento se dá a partir da transmissão da luz através de um disco perfurado acoplado ao eixo do motor que bloqueia periodicamente a luz emitida pelo fotoemissor para o fotorreceptor. A partir disso, é gerado um sinal retangular no qual a frequência é diretamente proporcional à velocidade de giro do motor (Bastos, 2019).

4. MATERIAIS E MÉTODOS

A metodologia deste trabalho envolveu a implementação de um controlador PID para um robô EV3, com o objetivo de melhorar sua navegação autônoma. O processo foi dividido em cinco fases principais: planejamento, revisão bibliográfica, implementação de multitarefas, desenvolvimento e ajuste do controlador PID, e finalmente, a coleta e análise de resultados. Na fase de implementação, utilizou-se a linguagem Micropython para programar o Mindstorm EV3, adaptando algoritmos existentes para suportar multitarefas e calcular a odometria de forma eficiente.

No referido trabalho, foram utilizados os componentes do kit LEGO Mindstorm EV3, especificamente o computador central, chamado de brick, dois motores grandes, rodas e as peças LEGO Technic utilizadas na estrutura da plataforma móvel.

O robô foi montado na estrutura uniciclo, ou seja, com duas rodas fixas sobre um mesmo eixo e uma roda louca, utilizando o modelo diferencial. Foi escolhida essa estrutura devido a cinemática simples amplamente utilizada nas pesquisas. O robô utiliza os sensores *encoders* dos motores e um giroscópio.

5. DESENVOLVIMENTO

Os experimentos foram realizados utilizando o robô EV3 montado em uma estrutura de unicíclo, equipados com sensores de encoders e giroscópio para monitorar a posição e a orientação. A coleta de dados foi realizada em uma superfície quadriculada de 1,20 m², permitindo medições precisas dos desvios entre o ponto estimado e o ponto real da trajetória do robô. As trajetórias testadas incluíram formas em linha reta, L, U e quadrado fechado, com múltiplas execuções para obter uma média dos erros com uma maior acurácia. Esses dados foram então analisados para avaliar a precisão da movimentação do robô e identificar áreas para melhorias futuras. A seguir, será detalhado como foi feito o cálculo da odometria e o desenvolvimento do controlador PID.

5.1. CÁLCULO DA ODOMETRIA

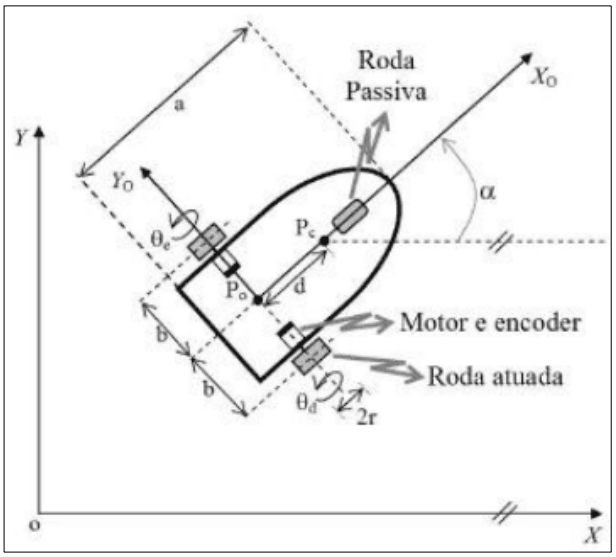
Neste estudo, foi aplicado o algoritmo proposto por Santos (2023) para realizar o cálculo da odometria. Para a inclusão desse algoritmo, foi necessário separar a tarefa de localização da ação dos atuadores, ou seja, foi necessário criar um *thread* responsável somente pelo cálculo da odometria. Esse *thread* é iniciado antes do acionamento dos motores. O controle dos motores é realizado pelo programa principal.

Dado que o processador do Brick do Ev3 possui um único núcleo, a multitarefa é na verdade uma alternância rápida entre a execução do *thread* e do programa. O uso de *threads* mostrou-se essencial devido ao caráter bloqueador das funções de acionamento dos motores do EV3, ou seja, funções que impedem a continuação da execução do programa até que a sua operação seja concluída. Essa estratégia foi adotada com o objetivo de reduzir o tempo que o robô leva para concluir sua trajetória e proporcionar uma coleta de dados em tempo real pelos *encoders*, o que aumenta a taxa de amostragem e, consequentemente, pode elevar a precisão dos cálculos. Em contrapartida pode aumentar o acúmulo de erros sistemáticos.

Para determinar a localização do robô, é necessário conhecer o seu modelo cinemático e, portanto, a disposição das rodas utilizadas na plataforma robótica. Neste trabalho, foram utilizados o modelo diferencial e a disposição uniciclo das rodas. Os robôs desenvolvidos nessa plataforma permitem raios de curvaturas equivalentes à metade do comprimento do entre eixo (Santos, 2023). As curvas nesse modelo, se faz diminuindo a atuação dos motores. Por exemplo, se o robô precisa fazer uma curva à direita, basta

diminuir a ação do motor da direita e manter a do motor esquerdo. A Figura 5 exemplifica a configuração diferencial.

Figura 5. Configuração diferencial



Fonte: (Pinto, 2016).

Em Bezerra (2004), são apresentadas as equações posteriormente adotadas por Santos (2023) para a construção do algoritmo de odometria que utilizamos neste trabalho e são apresentadas a seguir somente as equações finais para determinação da localização do robô.

$$x(t+\Delta t) = x(t) + (N_d * r_d + N_e * r_e) * \frac{\pi}{N_{res}} * \cos\theta\,(t) \qquad (6)$$

$$y(t+\Delta t) = y(t) + (N_d * r_d + N_e * r_e) * \frac{\pi}{N_{res}} * \operatorname{sen}\theta\,(t) \qquad (7)$$

$$\theta(t+\Delta t) = \theta(t) + (N_d * r_d + N_e * r_e) * \frac{2\pi}{b * N_{res}} * \cos\theta\,(t) \qquad (8)$$

As Equações 6, 7 e 8 determinam a localização do robô em relação a um referencial inicial, ao acumular as distâncias percorridas e as variações de orientação em cada instante de tempo (Bezerra; Alsina; Medeiros, 2003). N_d e N_e são os números de pulsos lidos nos *encoders* direito e esquerdo, respectivamente. N_{res} é a resolução do *enconder*, ou seja, indica o número de pulsos em um giro da roda. As variáveis r_d e r_e são os raios das rodas direita e esquerda.

A função responsável pelos cálculos da odometria, foi desenvolvida com base nas Equações 6, 7 e 8. Para capturar os pulsos das rodas esquerda e direita, utilizou-se o método “angle()”, responsável pela coleta dessas informações a partir dos *encoders* acoplados aos motores. Entretanto, há um tratamento interno da biblioteca Pybricks e esse método retorna à rotação total de todos os deslocamentos, o que torna necessário utilizar duas variáveis auxiliares (N_e e N_d), uma para cada motor, que registra a rotação da iteração atual, conforme descrito em Santos (2023).

Os dados resultantes desses cálculos são armazenados em listas. Como a odometria se baseia no princípio da localização relativa, é necessário utilizar as posições anteriores para determinar a posição atual (Santos, 2023).

O *thread* é iniciado dentro da função “move(alvo,eixo)” responsável pelo acionamento dos motores. Para isso, utilizamos a função “start_new_thread()” da biblioteca *threads*.

5.2 CONTROLADOR PID DIGITAL

A implementação de um controlador PID foi necessária para minimizar a interferência da diferença de velocidades entre os motores na trajetória autônoma do robô, pois essa diferença de velocidades impede que o robô siga uma linha reta, mesmo quando a mesma potência é aplicada em ambos os motores. O controlador foi implementado de acordo com o pseudocódigo apresentado na Figura 6.

Para desenvolver esse controlador, foi utilizada uma implementação da linguagem de programação Python, otimizada para uso em microcontroladores, a MicroPython. Essa linguagem foi escolhida diante da

facilidade em dar continuidade ao trabalho desenvolvido por Santos (2023), e por permitir de maneira fácil a implementação de multitarefas utilizando os *threads*.

Após uma análise da documentação da biblioteca Ev3dev e Pybricks, foi identificada a presença de um controlador PID de velocidade embutido na função de ativação dos motores devidamente sintonizado. É importante ressaltar que este controlador não pode ser desativado, uma vez que que sua desativação resultaria no não acionamento dos motores. Além disso, o firmware padrão do EV3, desenvolvido pela LEGO, também implementa por padrão um controlador PID de velocidade. O que facilita a tarefa de se mover em linha reta, desde que os motores estejam em perfeito estado de funcionamento.

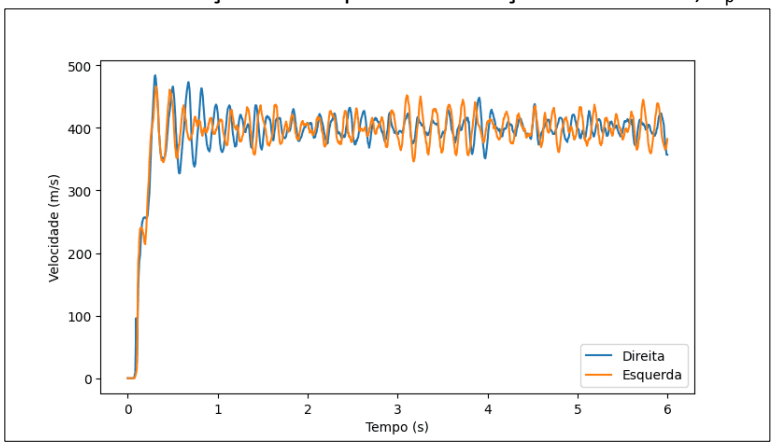
Figura 6. Pseudo código de um controlador PID digital

Algorithm:
Loop start:
To Calculate error = Midpoint value – actual value (read from the colour sensor)
To Calculate Integral = Error + Integral
To Derivative = Error – Last Error
*calculation for motor direction input = (Kp * Error) + (Ki * Integral) + (Kd * Derivative)*
Last error = error
Loop continues

Fonte: (Stephan *et al.*, 2023).

Inicialmente, foi utilizado o robô móvel construído por Santos (2023), no qual foram realizadas algumas tentativas de calibração dos controladores PID de velocidades da biblioteca pybrics. As tentativas falharam, devido a não obtenção de um K_p crítico em que a oscilação da velocidade se mantém por um período, como visto na Figura 7 Os testes realizados adotaram valores de 100 a 3000 para K_p e os motores foram acionados em um intervalo de 6 a 7 segundos. A falta de informações de como esse controlador foi implementado impactou bastante nessa etapa.

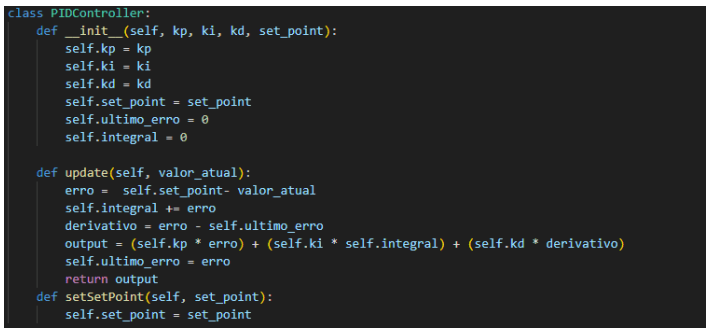
Figura 7. Gráfico da velocidade em relação ao tempo com oscilação inconstante, K_p = 800



Diante das dificuldades encontradas com a abordagem inicial, a busca por uma outra solução levou à implementação de um novo controlador PID, visto na Figura 8 A classe “PIDController”, implementa um controlador PID. O construtor dessa classe inicia os parâmetros com os valores desejados, sendo eles:

- K_p: O ganho proporcional;
- K_i: O ganho integral, que determina a reação do controlador à soma acumulada dos erros passados, ajudando a eliminar o erro estático;
- k_d: O ganho derivativo, que determina a reação do controlador à taxa de variação do erro, como visto na seção 2.1.3, a ação derivativa prevê e minimiza possíveis oscilações futuras;
- set_point: O valor objetivo que o sistema deve manter;
- Quatro portas de saída para os motores e quatro portas de entrada para os sensores;
- As variáveis ultimo_erro e integral, são iniciadas com 0;

Figura 8. Classe que implementa o controlador PID



```
class PIDController:
    def __init__(self, kp, ki, kd, set_point):
        self.kp = kp
        self.ki = ki
        self.kd = kd
        self.set_point = set_point
        self.ultimo_erro = 0
        self.integral = 0

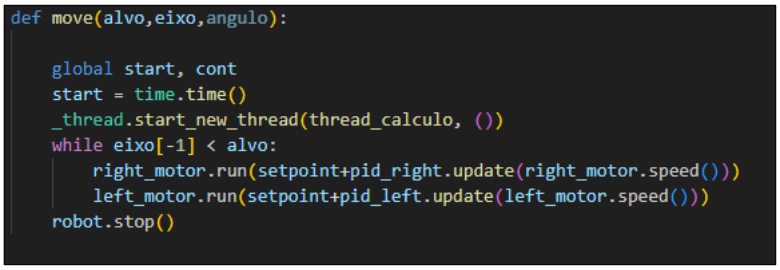
    def update(self, valor_atual):
        erro =  self.set_point- valor_atual
        self.integral += erro
        derivativo = erro - self.ultimo_erro
        output = (self.kp * erro) + (self.ki * self.integral) + (self.kd * derivativo)
        self.ultimo_erro = erro
        return output
    def setSetPoint(self, set_point):
        self.set_point = set_point
```

O controle em si é feito no método “update()”, que é chamado a cada iteração do programa principal para calcular a correção com base no valor atual da variável controlada, nesse caso a velocidade dos motores. Inicialmente, o método calcula o erro como a diferença entre o *setpoint* e o valor atual de controle. Em seguida, atualiza a variável ``integral', adicionando o erro atual ao acumulado anteriormente O método então, calcula um termo derivativo definido como a variação do erro entre as iterações.

A partir disso, a saída do controlador é computada como a soma dos termos proporcional, baseado no erro atual, integral e derivativo, ajustados pelos ganhos K_p, K_i e K_d, respectivamente. Ao concluir cada atualização, o valor atual do erro substitui o último erro registrado e será utilizado na próxima iteração, mantendo o *feedback* de controle.

O valor obtido pela saída foi utilizado para ajustar a velocidade dos motores, na Figura 9 pode-se notar o trecho do código responsável pelo acionamento dos motores. O método “run()” da biblioteca Pybricks espera um único parâmetro, sendo esse a velocidade em mm/s do motor. Infelizmente, não é possível fazer o controle por tensão, como feita nos trabalhos de Bastos (2019) utilizando a placa multiplataforma Raspberry Pi.

Figura 9. Ajuste da velocidade dos motores



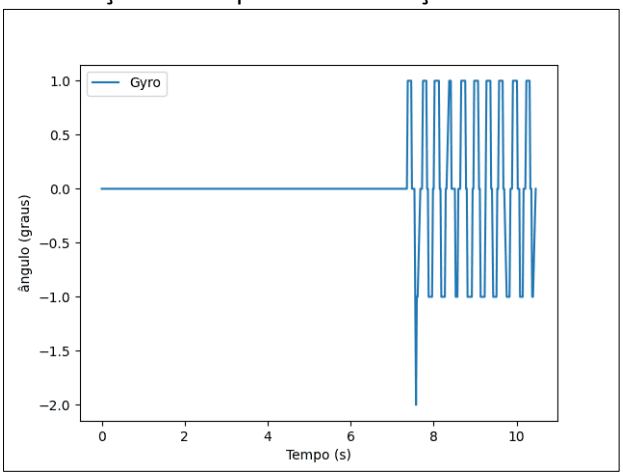
```
def move(alvo,eixo,angulo):

    global start, cont
    start = time.time()
    _thread.start_new_thread(thread_calculo, ())
    while eixo[-1] < alvo:
        right_motor.run(setpoint+pid_right.update(right_motor.speed()))
        left_motor.run(setpoint+pid_left.update(left_motor.speed()))
    robot.stop()
```

O controlador PID foi afinado para minimizar erros na trajetória do robô, garantindo que ele seguisse um caminho pré-determinado com precisão. Contudo, na primeira sintonia, foi percebido que um dos motores apresentava um problema na inicialização, que fazia com que o motor acelerasse de forma mais lenta ou menos agressiva. Diante desses problemas, foi realizada a substituição de toda estrutura do robô, com exceção do *brick*.

A sintonia do controlador PID foi feita utilizando o método da oscilação contínua. A variável de controle, neste caso, foi o ângulo lido pelo giroscópio. O *setpoint* para sintonia foi o ângulo 0, o que corresponde a uma trajetória em linha reta. A Figura 10 apresenta o gráfico da leitura do giroscópio para um $K_p = 16$ e um tempo de oscilação de 0.4s, ou seja, $k_{crítico}$ é 16 e o $T_{crítico}$ é 0,4s.

Figura 10. Gráfico do ângulo em relação ao tempo com a oscilação mantida



Utilizando as fórmulas para a sintonia por oscilação mantida propostas por Ziegler e Nichols e adaptada por Bega *et al.* (2006), obtem-se as seguintes constantes:

$$\begin{cases} K_p = 9{,}6 \\ T_i = 0{,}8 \\ T_d = 0{,}2 \end{cases} \qquad (9)$$

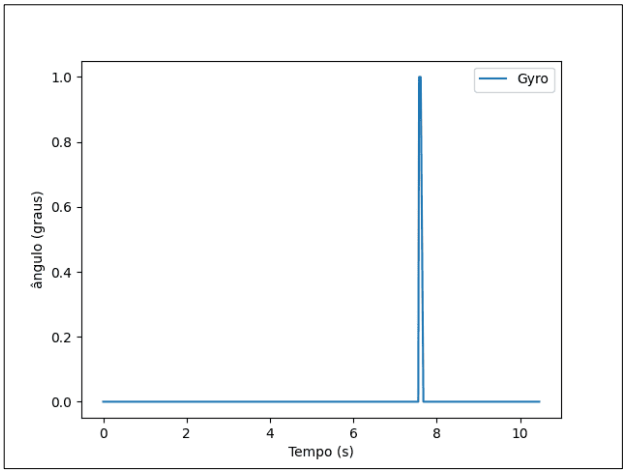
Para obter as constantes K_i e K_p é necessário realizar a transformação de T_i para K_i e de T_d para K_d, resultando nos seguintes valores:

$$\begin{cases} K_p = 9{,}6 \\ K_i = 5 \\ K_d = 0{,}05 \end{cases} \qquad (10)$$

Para realizar as correções na trajetória foi alterada a função de acionamento dos motores para a "robot.drive()", que recebe como parâmetros a velocidade do robô em mm/s e a taxa de rotação.

Pode-se notar então na Figura 11 uma estabilidade relativa do robô, corrigindo o ângulo sempre que necessário.

Figura 11. Ajuste da velocidade dos motores



5.3 EXPERIMENTO E COLETA DE DADOS

Para a coleta dos dados, foi necessária a construção de uma escala quadriculada, como sugerido por Santos (2023) para facilitar as medições e prover maior precisão e confiabilidade nos dados. Neste trabalho, foi adotada uma escala em que cada quadrado tem 1cm² e foi impressa em uma lona de 1,20m², mantendo a incerteza de +-3mm.

Foram sugeridas por Santos (2023) quatro trajetórias, sendo elas: Linha reta, em L, em U e um quadrado fechado. Para verificar o desvio do robô, mede-se a distância entre o ponto final estimado e o ponto final real. Esse desvio foi medido com uma fita métrica de 120 cm e anotado em papel. A cada duas execuções, reinicia-se o *brick* a fim de calibrar o giroscópio que acumula erros com o passar do tempo. Quanto maior o tempo de trabalho do sensor, maior o erro. Foram executados dez testes para cada trajetória, cinco em sentido horário e cinco em sentido anti-horário. Os dados da odometria foram salvos no final de cada trajetória em um arquivo. A partir desses dados coletados, foram preenchidas as tabelas que apontam o erro real de cada um desses percursos. Foi utilizada a biblioteca Numpy para calcular a média e o desvio padrão do erro obtido.

6. ANÁLISE DOS RESULTADOS

Foram realizados dez testes para cada trajetória proposta por Santos (2023), sendo elas: Em linha reta, em formato de L, em formato de U e por fim a trajetória fechada no formato de um quadrado de ($1m^2$). O experimento foi realizado em um plano cartesiano impresso em uma lona de ($120cm^2$), o robô percorre as trajetórias cinco vezes no sentido horário e cinco vezes no sentido anti-horário. O robô foi programado para percorrer 1m em cada direção, ou seja, se a trajetória é em L, a trajetória é dividida em duas partes, na primeira parte ele percorre 1m, faz uma curva de 90 graus e percorre a segunda parte, ou seja, mais um metro em linha reta. Dessa forma, calcula-se o erro real entre a localização real e a localização estimada.

Os testes foram realizados com o robô partindo do mesmo ponto, mudando apenas a orientação de acordo com o sentido desejado para percorrer a trajetória. O desvio do carro autônomo foi calculado adotando como referência a lateral direita do robô em relação ao ponto final estimado de cada trajetória.

Os resultados obtidos são apresentados nas tabelas de 1 a 7 Para melhor organização e compreensão as tabelas são divididas de acordo com a trajetória percorrida, estruturada da seguinte forma:

- Y est. e X est. referem-se à localização teórica do robô, ou seja, a localização esperada do robô no ponto final da trajetória;
- Y real e X real referem-se à localização real do robô ao fim do experimento;
- Erro (x) e erro (Y) referem-se a diferença entre a posição estimada e a posição real do robô ao fim do experimento;

Tabela 1. Trajetória em linha reta.

Teste	X real	Y real	X est.	Y est.	Erro X	Erro Y
1	1051 mm	-3 mm	1000 mm	0 mm	51 mm	-3 mm
2	1043 mm	-25 mm	1000 mm	0 mm	43 mm	-25 mm
3	1048 mm	-16 mm	1000 mm	0 mm	48 mm	-16 mm
4	1009 mm	18 mm	1000 mm	0 mm	9 mm	18 mm
5	1015 mm	33 mm	1000 mm	0 mm	15 mm	33 mm

Tabela 2. Trajetória em L no sentido anti-horário

Teste	X real	Y real	X est.	Y est.	Erro X	Erro Y
1	1015 mm	1020 mm	1000 mm	1000 mm	15 mm	20 mm
2	1008 mm	1040 mm	1000 mm	1000 mm	8 mm	40 mm
3	1040 mm	1003 mm	1000 mm	1000 mm	40 mm	3 mm
4	1001 mm	1001 mm	1000 mm	1000 mm	1 mm	1 mm
5	1002 mm	999 mm	1000 mm	1000 mm	2 mm	-1 mm

Tabela 3. Trajetória em U no sentido anti-horário

0	X real	Y real	X est.	Y est.	Erro X	Erro Y
1	-8 mm	1090 mm	0 mm	1000 mm	-8 mm	90 mm
2	-2 mm	1004 mm	0 mm	1000 mm	-2 mm	4 mm
3	-1 mm	1040 mm	0 mm	1000 mm	-1 mm	40 mm
4	2 mm	998 mm	0 mm	1000 mm	2 mm	-2 mm
5	53 mm	1131 mm	0 mm	1000 mm	53 mm	131 mm

Tabela 4. Trajetória fechada no sentido anti-horário

Teste	X real	Y real	X est.	Y est.	Erro X	Erro Y
1	67 mm	-62 mm	0 mm	0 mm	67 mm	-62 mm
2	100 mm	-10 mm	0 mm	0 mm	100 mm	-10 mm
3	-78 mm	-18 mm	0 mm	0 mm	-78 mm	-18 mm
4	-13 mm	-30 mm	0 mm	0 mm	-13 mm	-30 mm
5	-15 mm	-10 mm	0 mm	0 mm	-15 mm	-10 mm

Tabela 5. Trajetória em L no sentido horário

Teste	X real	Y real	X est.	Y est.	Erro X	Erro Y
1	1075 mm	-1072 mm	1000 mm	-1000 mm	75 mm	72 mm
2	1038 mm	-1055mm	1000 mm	-1000 mm	38 mm	55 mm
3	1003 mm	-1011 mm	1000 mm	-1000 mm	3 mm	11 mm
4	997 mm	-1020 mm	1000 mm	-1000 mm	-3 mm	20 mm
5	996 mm	-1003 mm	1000 mm	-1000 mm	-4 mm	3 mm

Tabela 6. Trajetória em U no sentido horário

Teste	X real	Y real	X est.	Y est.	Erro X	Erro Y
1	-79 mm	-1037 mm	0 mm	-1000 mm	-79 mm	37 mm
2	-50 mm	-953 mm	0 mm	-1000 mm	-50 mm	47 mm
3	-74 mm	-1010 mm	0 mm	-1000 mm	-74 mm	10 mm
4	43 mm	-1007 mm	0 mm	-1000 mm	43 mm	7 mm
5	2 mm	-986 mm	0 mm	-1000 mm	2 mm	14 mm

Tabela 7. Trajetória Fechada no sentido horário.

Teste	X real	Y real	X est.	Y est.	Erro X	Erro Y
1	67 mm	-118 mm	0 mm	0 mm	67 mm	-118 mm
2	142 mm	98 mm	0 mm	0 mm	142 mm	98 mm
3	126 mm	150 mm	0 mm	0 mm	126 mm	150 mm
4	-77 mm	30 mm	0 mm	0 mm	-77 mm	30 mm
5	-90 mm	70 mm	0 mm	0 mm	-90 mm	70 mm

Tabela 8. Média e desvio padrão do erro de cada trajetória.

Trajetória	Sentido	Média X	Média Y	Des. Padrão X	Des. Padrão Y
Linha Reta	–	33.2 mm	1.4 mm	19.6 mm	23.9 mm
Em L	Horário	21.8 mm	32.2 mm	19.6 mm	23.9 mm
Em L	Anti-horário	12.6 mm	30 mm	15.9 mm	17.4 mm
Em U	Horário	-31.6 mm	23.0 mm	52.6 mm	17.8 mm
Em U	Anti-horário	8.8 mm	52.6 mm	24.9 mm	57.11 mm
Fechada	Horário	33.6 mm	46.0 mm	110.5 mm	101.5 mm
Fechada	Anti-horário	12.2 mm	-26.0 mm	71.1 mm	21.7 mm

Tabela 9. Tabela com os dados obtidos por Santos (2023)

Trajetória	X Real	Y Real	Erro X	Erro Y
Linha Reta	983 mm	14mm	17 mm	14 mm
Em L	933 mm	1020 mm	-67 mm	20 mm
Em U	971 mm	71 mm	-29 mm	71 mm
Fechada	8 mm	183 mm	8 mm	183 mm

O resultado apresentado nas tabelas 1 a 7 evidenciam distorções na trajetória principalmente pelo acúmulo de erros do giroscópio ao realizar uma curva de 90 graus. Outras possíveis causas podem ser a perda de tração das rodas em algum trecho da trajetória e a falta de sensibilidade da função “gyro.angle()”da biblioteca “pybricks” a mudanças leves na angulação. Ainda assim, o robô é capaz de executar sua tarefa de maneira correta. A diferença significativa entre a localização estimada e a posição real, pode evidenciar um problema de sincronia entre o *thread* responsável pelo cálculo da odometria e o programa principal. A tabela 8 resume as informações apresentada nas tabelas 1 a 7 por meio da média e do desvio padrão. Nota-se que as trajetórias fechadas apresentaram maiores variabilidade, com desvio padrão superior a 100 mm em alguns casos, o que colabora com a hipótese de que o erro é acumulado proporcionalmente ao tempo de execução e quantidade de curvas do circuito.

Por fim, observando os dados de Santos (2023), apresentados na tabela 9, embora os resultados anteriores não permitam comparação direta, os erros médios obtidos neste trabalho foram significativamente menores em duas dimensões. Enquanto Santos registrou erro de 183mm no eixo X na trajetória fechada, neste estudo o maior erro médio na mesma trajetória foi de 46 mm. Esse resultado mostra que a integração do controlador PID reduziu as distorções na trajetória, confirmando a hipótese central do trabalho.

7. CONSIDERAÇÕES FINAIS

Conclui-se que a união do controlador PID desenvolvido neste trabalho com o algoritmo de localização baseado em odometria desenvolvido por Santos (2023) resulta em um robô móvel mais preciso para a maioria dos casos de teste. Apesar das limitações combinadas do sensor giroscópio e da biblioteca escolhida, como a falta de sensibilidade a pequenas mudanças angulares e o acúmulo de erros ao realizar curvas de 90 graus, o robô foi capaz de seguir trajetórias planejadas com um nível de precisão que atende às expectativas do projeto. A integração do controlador com o algoritmo de localização por odometria não só melhorou a navegação autônoma do robô, mas também reduziu significativamente os desvios acumulados durante a operação contínua.

Como passos futuros, propõe-se a inclusão de outros tipos de sensores ao EV3, como o sensor ultrassônico e câmera, para possibilitar a adição de um módulo de visão computacional. Além disso, sugere-se a fusão dos dados do giroscópio e acelerômetro para uma melhor precisão nos ângulos lidos, utilizando o filtro de Kalman. É interessante, também, explorar o uso de uma linguagem de programação que permita a utilização do EV3 com mais liberdade, como a Simulink, e aumentar a amostragem de teste para uma análise estatística mais precisa.

AGRADECIMENTOS

Os autores expressam sua gratidão à Universidade Estadual de Santa Cruz (UESC) pelo apoio e pela infraestrutura disponibilizada, que foram fundamentais para o desenvolvimento deste trabalho.

REFERÊNCIAS

BASTOS, F. F. **Estudo e implementação de controladores Fuzzy e PID para controle de direção e velocidade de um AGV com visão computacional.** Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal Rural do Semi-Árido, Mossoró, 2019.

BEGA, E. A. *et al.* **Instrumentação Industrial**. 2. ed. Curitiba: Interciência, 2006.

BEZERRA, C. G; ALSINA, P. J.; MEDEIROS, A A D. Um sistema de localização para um robô móvel baseado em odometria e marcos naturais. *In*: SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 6. 2003. **Anais** [...]. Bauru, SP: Unesp, 2003. Disponível em: https://www.dca.ufrn.br/~adelardo/artigos/SBAI03e.pdf. Acesso em: 12 set. 2025.

BEZERRA, C. G. **Localização de um robô móvel usando odometria e marcos naturais.** 2004. Dissertação (Mestrado em Ciências) - Universidade Federal do Rio Grande do Norte, Natal, RN, 2004.

CAMILO, J. I. S. **Localização baseada em odometria visual para robôs móveis terrestres**. 2025. Trabalho de conclusão de curso (Graduação em Engenharia de Controle e Automação) — Universidade Federal do Rio Grande do Norte, Natal, RN, 2025.

ČAPEK, K. **RUR:** Robôs Universais de Rossum. Edição: Rogério Pietro. [*S.l.*]: Madrepérola, 2021.

DISTEFANO, J. J.; STUBBERUD, A. R; WILLIAMS, I. J. **Sistemas de controle.** Porto Alegre: Bookman, 2014.

FERNANDES, L. C. **Análise e implementação de algoritmos para localização de robôs móveis.** 2003. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - USP, São Carlos, 2003.

FERRAZ, F. H. C. **Um método de sintonia de controladores PID multivariáveis baseado em otimização para sistemas lineares sob restrições**. 2022. Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica e de Computação. Natal, RN, 2022. DOI: https://doi.org/10.26678/ABCM.CONEM2022.CON22-0111

GENTILHO, E. **Instrumentação e controle PID embarcado para cadeiras de rodas elétricas Acionada por Sopro e Sucção**, 2014.

ISO**. ISO 8362:2021**. [*S. l.*: *s. n.*], 2021. Disponível em: https://www.iso.org/obp/ui/en/#iso:std:iso:8373:ed-3:v1:en. Acesso: 03 de Dezembro de 2023.

TEAM, MathWorks Student Competitions. Student Competition: Mobile Robotics Training. **MathWorks**, 2025. Disponível em: https://www.mathworks.com/matlabcentral/fileexchange/62966-student-competition-mobile-robotics-training. Acesso em: 15 ago. 2025.

MARCOMINI, M. H. C. **Análise da aplicação de diferentes controladores em robô seguidor de linha para LEGO® MINDSTORMS® EV**. 2023. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação) - Faculdade de Engenharia de Controle e Automação, Escola Politécnica, Pontifícia Universidade Católica de Campinas, Campinas, 2023.

MICHAELLIS DICIONÁRIO BRASILEIRO DE LÍNGUA PORTUGUESA. **Robô.** Michaellis: Dicionário Brasileiro de Língua Portuguesa. Disponível em: https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/robo Acesso em: 15 ago. 2025

MURRILL, P. W. **Fundamentals of process control theory**. Research Triangle Park: ISA, 1981.

NASCIMENTO, R. C. A. **Localização de robôs móveis em ambientes fechados utilizando câmeras montadas no teto.** 2014. Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte, Natal, 2014.

PINTO, N. J. F. **Implementação e teste de soluções de navegação de robôs móveis com base no sistema NXT/EV3 da LEGO®,** 2016.

REZENDE, G. L.; SILVA, F. O.; MARQUES, L. S. **Comparison of Wheel Odometry and Visual Odometry for low-cost vehicle navigation**. *In*: BRAZILIAN SYMPOSIUM ON ROBOTICS (SBR), WORKSHOP ON ROBOTICS IN EDUCATION (WRE). [*S. l.: s. n*.], 2024. p. 91–96. DOI: https://doi.org/10.1109/SBR/WRE63066.2024.10838039

SANTOS, E. N. R. **Um algoritmo de localização baseado em Odometria usando a plataforma LEGO MINDSTORMS EV3.** 2023. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade Estadual de Santa Cruz, Ilhéus, 2023.

STEPHAN, A. *et al.* P**erformance Analysis of LEGO Mindstorm EV3 Robot with PID Controller for Trajectory Tracking Application.** Institute of Electrical e Electronics Engineers (IEEE), p. 1782–1787, jun. 2023. DOI: https://doi.org/10.1109/ICICCS56967.2023.10142757.

TANNURI, E. **Apostila sobre PID e métodos de sintonia** [apostila]. São Paulo: USP, 2023.

ZIEGLER, J. G.; NICHOLS, N. B. Optimum settings for automatic controllers. Transactions of the American society of mechanical engineers. **American Society of Mechanical Engineers**, v. 64, n. 8, p. 759–765, 1942. DOI: https://doi.org/10.1115/1.4019264