UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

RYAN GUILHERME MORAIS NASCIMENTO

PLATAFORMA ROBÓTICA MULTISSENSORIAL DE BAIXO CUSTO PARA USO EM ROBÓTICA EDUCACIONAL

QUIXADÁ

2026

RYAN GUILHERME MORAIS NASCIMENTO

PLATAFORMA ROBÓTICA MULTISSENSORIAL DE BAIXO CUSTO PARA USO EM ROBÓTICA EDUCACIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Thiago Werlley Bandeira da Silva.

QUIXADÁ

2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

N198p Nascimento, Ryan Guilherme Morais.
Plataforma Robótica Multissensorial de Baixo Custo para uso em Robótica Educacional / Ryan Guilherme Morais Nascimento. – 2026.
85 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Computação, Quixadá, 2026.
Orientação: Prof. Dr. Thiago Werlley Bandeira da Silva.

1. Robótica Educacional. 2. ROS. 3. Sistemas Embarcados. 4. Robótica Acessível. 5. Programação Visual. I. Título.

CDD 621.39

RYAN GUILHERME MORAIS NASCIMENTO

PLATAFORMA ROBÓTICA MULTISSENSORIAL DE BAIXO CUSTO PARA USO EM ROBÓTICA EDUCACIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Aprovada em: 23/01/2026.

BANCA EXAMINADORA

Prof. Dr. Thiago Werlley Bandeira da Silva (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Antonio Joel Ramiro de Castro
Universidade Federal do Ceará (UFC)

Prof. Ms. Francisco Evando Nascimento dos Santos
Secretaria da Educação (SEDUC)

À minha família, por sua capacidade de acreditar em mim e investir em mim.

À minha madrinha e tia Maria Benoli (in memoriam), cujo carinho, apoio incondicional e exemplo permanecem vivos em minha memória e foram fundamentais na minha formação pessoal e humana.

AGRADECIMENTOS

À Instituição UFC Campus Quixadá, pelo apoio financeiro com a manutenção da bolsa de auxílio.

Ao Prof. Dr. Thiago Werlley, pela excelente orientação.

Aos professores participantes da banca examinadora Joel Ramiro e Evando Santos pelo tempo, pelas valiosas colaborações e sugestões.

Aos colegas da turma de graduação, pelas reflexões, críticas e sugestões recebidas, em especial ao colega Guilherme Floriano, pelo apoio técnico durante o desenvolvimento da implementação e pelos testes realizados ao longo do projeto.

Aos meus familiares pelo apoio incondicional.

"Educational robotics offers students a unique opportunity to learn through hands-on experience and critical thinking. By designing and building robots, students not only learn about science and technology, but also develop collaboration and leadership skills that will be valuable in any career they pursue."
(PANETTA, 2023.)

RESUMO

Este trabalho propõe o desenvolvimento de um robô multissensorial de baixo custo inspirado no TurtleBot fabricado pela OpenRobotics, voltado para aplicações educacionais no ensino básico e superior. A plataforma robótica integra sistemas embarcados de baixo consumo energético como Raspberry Pi Pico W, diversos sensores (LIDAR, ultrassônicos, encoders, IMU) e utiliza o Robot Operating System 2 (ROS2) como middleware para comunicação entre os componentes. O diferencial da proposta está na combinação de baixo custo, código aberto e versatilidade pedagógica, permitindo sua utilização em diferentes níveis educacionais. A metodologia contempla o projeto e implementação da plataforma física, o desenvolvimento de práticas pedagógicas fundamentadas em aprendizagem ativa e construcionismo inclusas em um website para desenvolvido neste trabalho, a integração com plataformas de programação visual No-Code como o BIPES, e a elaboração de atividades educacionais alinhadas às diretrizes curriculares nacionais. A análise de trabalhos relacionados evidencia a originalidade da proposta ao combinar acessibilidade financeira, robustez técnica e flexibilidade pedagógica. Os resultados incluem uma plataforma robótica funcional, documentação completa para reprodução e adaptação, e um conjunto de atividades educacionais estruturadas que promovam o desenvolvimento do pensamento computacional, criatividade e trabalho colaborativo. A plataforma também foi submetida à uma cadeia de testes em uma turma de Robótica de ensino superior para ter seu desempenho avaliado por meio de uma pesquisa de satisfação, onde apresentou resultados satisfatórios, principalmente no âmbito educacional. O projeto visa contribuir para a democratização do ensino de robótica no Brasil, oferecendo uma alternativa viável e adaptada à realidade educacional nacional.

Palavras-chave: Robótica Educacional; ROS; Sistemas Embarcados; Robótica Acessível; Programação Visual.

ABSTRACT

This work proposes the development of a low-cost multisensory robot inspired by the TurtleBot manufactured by OpenRobotics, aimed at educational applications in basic and higher education. The robotic platform integrates low-power embedded systems such as the Raspberry Pi Pico W, multiple sensors (LiDAR, ultrasonic sensors, encoders, IMU), and uses the Robot Operating System 2 (ROS2) as middleware for communication among components. The distinguishing feature of the proposal lies in the combination of low cost, open-source design, and pedagogical versatility, enabling its use across different educational levels. The methodology encompasses the design and implementation of the physical platform, the development of pedagogical practices grounded in active learning and constructionism included in a website developed in this work, integration with No-Code visual programming platforms such as BIPES, and the design of educational activities aligned with national curricular guidelines. The analysis of related work highlights the originality of the proposal by combining financial accessibility, technical robustness, and pedagogical flexibility. The results include a functional robotic platform, comprehensive documentation for reproduction and adaptation, and a structured set of educational activities that foster the development of computational thinking, creativity, and collaborative work. The platform was also tested in a higher education Robotics class, where its performance was evaluated through a satisfaction survey, where it showed satisfactory results, especially in the educational area. The project aims to contribute to the democratization of robotics education in Brazil by offering a viable alternative tailored to the national educational context.

Keywords: Educational Robotics; ROS; Embedded Systems; Accessible Robotics; Visual Programming

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ABS	*Acrylonitrile Butadiene Styrene*
API	*Application Programming Interface*
BIPES	*Block Based Integrated Platform for Embedded Systems*
IDC	*Insulation Displacement Connector*
LIDAR	*Light Detection and Ranging*
MQTT	*Message Queuing Telemetry Transport*
PCB	*Printed Circuit Board*
PWM	*Pulse Width Modulation*
ROS	*Robot Operating System*
SLAM	*Simultaneous Location and Mapping*
UART	*Universal Asynchronous Receiver/Transmitter*

SUMÁRIO

1 INTRODUÇÃO

A robótica tem se consolidado como uma das áreas mais promissoras da tecnologia, com aplicações que se expandem dos ambientes industriais para o cotidiano doméstico, educacional e científico. O avanço acelerado da inteligência artificial, sensores embarcados e plataformas computacionais permitiu o desenvolvimento de robôs cada vez mais autônomos, colaborativos e acessíveis. De acordo com a Federação Internacional de Robótica (International Federation of Robotics, 2022), a adoção de robôs em diversos setores cresceu mais de 30% nos últimos anos, refletindo não apenas sua relevância industrial, mas também o interesse acadêmico e educacional.

Nesse contexto, o Robot Operating System (ROS) surge como um middleware padrão para o desenvolvimento de sistemas robóticos modulares, escaláveis e reconfiguráveis. O ROS oferece uma infraestrutura robusta para o controle de sensores, atuadores e algoritmos de navegação, sendo amplamente utilizado tanto em pesquisa quanto em projetos comerciais (Open Robotics, s.d.). Um dos principais exemplos de aplicação educacional baseada em ROS é o TurtleBot, uma plataforma robótica compacta idealizada pela OpenRobotics, equipada com sensores de mapeamento e navegação, voltada para experimentação e aprendizado (Robotics, 2024c). Apesar de sua funcionalidade e versatilidade, o TurtleBot presente no mercado apresenta um alto custo, especialmente para instituições de ensino em países em desenvolvimento, como o Brasil (Robotics, 2024c). Além disso, sua aquisição depende de importações e infraestrutura compatível, o que compromete sua viabilidade para implementação em larga escala no ensino médio e técnico. Diante disso, torna-se necessária a criação de uma solução mais acessível, aberta e adaptável à realidade nacional.

A estrutura básica de um robô móvel educacional envolve componentes fundamentais como microcontroladores, atuadores, sensores de distância e posicionamento, além de unidades de processamento e comunicação (Siciliano *et al.*, 2010). Para permitir aplicações em diferentes níveis de ensino, desde o básico até o superior, é necessário que o sistema seja modular e compatível com plataformas como o ROS e interfaces visuais de programação, como o BIPES, que permite a criação de algoritmos por meio de blocos gráficos sem exigir conhecimentos prévios de codificação textual (Almeida *et al.*, 2021) tornando a iniciativa aplicável em unidades de ensino básico. Nesse sentido, a arquitetura do sistema proposto deve contemplar a redundância de sensores, a integração eficiente entre hardware e software, e a possibilidade de simulação e expansão dos módulos. A proposta deste trabalho consiste então no desenvolvimento de um

robô educacional inspirado no TurtleBot, porém com foco em baixo custo, código aberto e alta adaptabilidade.

A principal motivação deste trabalho está na falta de plataformas robóticas acessíveis e completas para fins educacionais em muitos contextos escolares e universitários. Embora iniciativas como o Arduino e o Raspberry Pi tenham popularizado o ensino de eletrônica e programação (Blikstein, 2013), ainda há uma lacuna quando se trata de robôs móveis capazes de operar com navegação autônoma, sensores LIDAR, mapeamento SLAM e outros recursos avançados de forma integrada e didática. Diversos estudos já destacam a importância de robôs educacionais na formação de habilidades do século XXI, como pensamento computacional, resolução de problemas e colaboração (Papert, 1980; Mubin *et al.*, 2013). No entanto, sem acesso a equipamentos adequados, essas habilidades permanecem distantes de grande parte dos estudantes brasileiros.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma plataforma robótica educacional inspirada no TurtleBot, baseado em ROS e compatível com interfaces de programação visual como o BIPES, capaz de atender tanto ao ensino básico quanto ao superior com o objetivo de democratizar o ensino de robótica avançada no Brasil, preenchendo lacunas no atual cenário.

1.1.2 Objetivos Específicos

- Desenvolver uma plataforma robótica, incluindo chassi, placas eletrônicas e microcontroladores
- Desenvolver um software modular de código aberto baseado em Robot Operating System (ROS)
- Integrar a plataforma robótica com um sistema de programação visual baseada em blocos para robótica (BIPES) para viabilizar o uso da plataforma no ensino básico
- Avaliar o desempenho da plataforma em uma disciplina de Robótica de Ensino Superior

1.2 Organização do Trabalho

Este trabalho está organizado da seguinte forma:

1. Capítulo 2: trata a fundamentação teórica, com os conceitos fundamentais para a compreensão da proposta descrita.
2. Capítulo 3: aborda os trabalhos relacionados, comparando os aspectos comuns ou divergentes entre eles e o trabalho aqui proposto.
3. Capítulo 4: descreve a metodologia que a ser abordada para o desenvolvimento deste trabalho.
4. Capítulo 5: apresenta os resultados obtidos.
5. Capítulo 6: descreve a conclusão deste trabalho e idealiza os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica que embasa o desenvolvimento de um robô multissensorial de baixo custo para aplicação em ensino básico e superior, inspirado no TurtleBot. Serão abordados os conceitos essenciais relacionados a robôs multissensoriais, os sistemas embarcados comumente utilizados, o Robot Operating System como framework de desenvolvimento, a plataforma No-Code como alternativa de programação viável para ensino básico e os princípios fundamentais da robótica móvel, além de conceitos no campo interdisciplinar da robótica educacional, que integra diversas teorias pedagógicas que fundamentam sua eficácia no processo de ensino-aprendizagem.

2.1 Software

2.1.1 ROS (Robot Operating System)

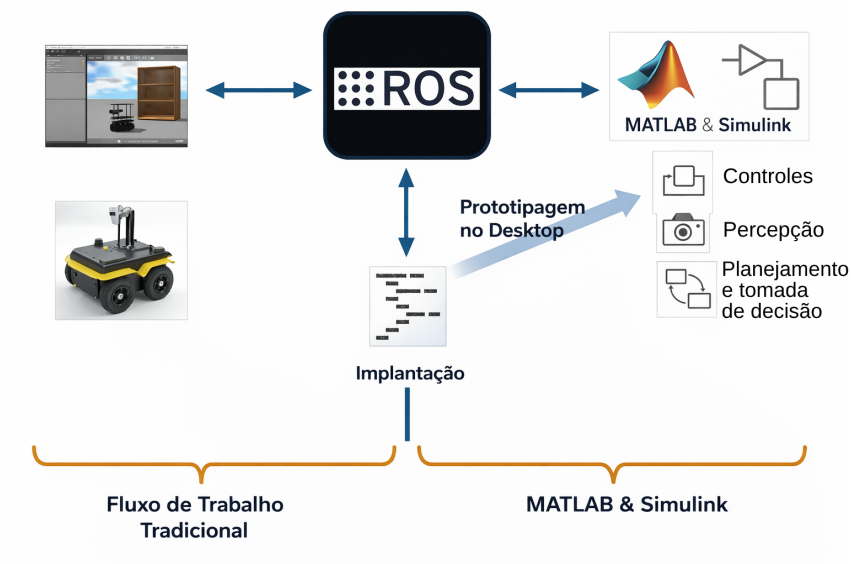
O ROS, apesar do nome, não se configura como um sistema operacional no sentido tradicional, mas sim como um meta-sistema operacional ou um framework de middleware flexível e de código aberto, projetado especificamente para o desenvolvimento de software para robôs (Open Robotics, s.d.). Sua concepção, iniciada na Universidade de Stanford e consolidada pela Willow Garage, visava superar a fragmentação e a complexidade inerentes ao desenvolvimento robótico, onde frequentemente equipes multidisciplinares necessitam integrar hardware e software de naturezas distintas (Quigley *et al.*, 2009). A filosofia central do ROS é fornecer um conjunto padronizado de ferramentas, bibliotecas e convenções que simplificam a criação de aplicações robóticas complexas e robustas, desde a abstração de hardware de baixo nível até algoritmos de percepção, planejamento e controle de alto nível.

A arquitetura do ROS é baseada em um grafo computacional distribuído, onde os processos, denominados nós (nodes), comunicam-se entre si através de um sistema de publicação/assinatura de mensagens (topics) e requisição/resposta de serviços (services). Essa estrutura permite uma modularidade significativa, onde diferentes funcionalidades (como controle de motores, processamento de sensores, navegação, etc.) podem ser desenvolvidas, testadas e executadas independentemente como nós separados (Open Robotics, s.d.). As mensagens trocadas possuem tipos de dados padronizados, facilitando a interoperabilidade entre nós desenvolvidos por diferentes equipes ou até mesmo em diferentes linguagens de programação, como C++ e

Python, que são as mais proeminentes no ecossistema ROS. Além dos tópicos e serviços, o ROS implementa um Parameter Server para configuração dinâmica dos nós e ferramentas como o rosbag para gravação e reprodução de dados de mensagens, essencial para depuração e análise.

A Figura 1, mostra um fluxograma do funcionamento de um sistema que utiliza ROS para interação entre as camadas.

Figura 1 – Fluxograma de interação de um sistema ROS



Fonte: Adaptado do blog MathWorks

A natureza open-source do ROS, majoritariamente sob a licença BSD, é um dos seus pilares fundamentais e fator chave para sua ampla adoção. Isso não apenas reduz custos de desenvolvimento, eliminando taxas de licenciamento, mas principalmente fomenta uma vasta comunidade global de desenvolvedores, pesquisadores e entusiastas que contribuem ativamente com novos pacotes, ferramentas, correções e suporte (Open Robotics, s.d.). Esse ecossistema colaborativo resultou em um repositório extenso de pacotes prontos para uso, cobrindo desde drivers para uma ampla gama de sensores e atuadores até implementações de algoritmos de ponta em áreas como visão computacional (integração com OpenCV), localização e mapeamento simultâneos (SLAM), planejamento de movimento (MoveIt!), simulação (Gazebo, RVIZ) e muito mais. A disponibilidade dessas ferramentas e algoritmos acelera drasticamente o ciclo de desenvolvimento, permitindo que os desenvolvedores se concentrem nos aspectos inovadores de suas aplicações robóticas.

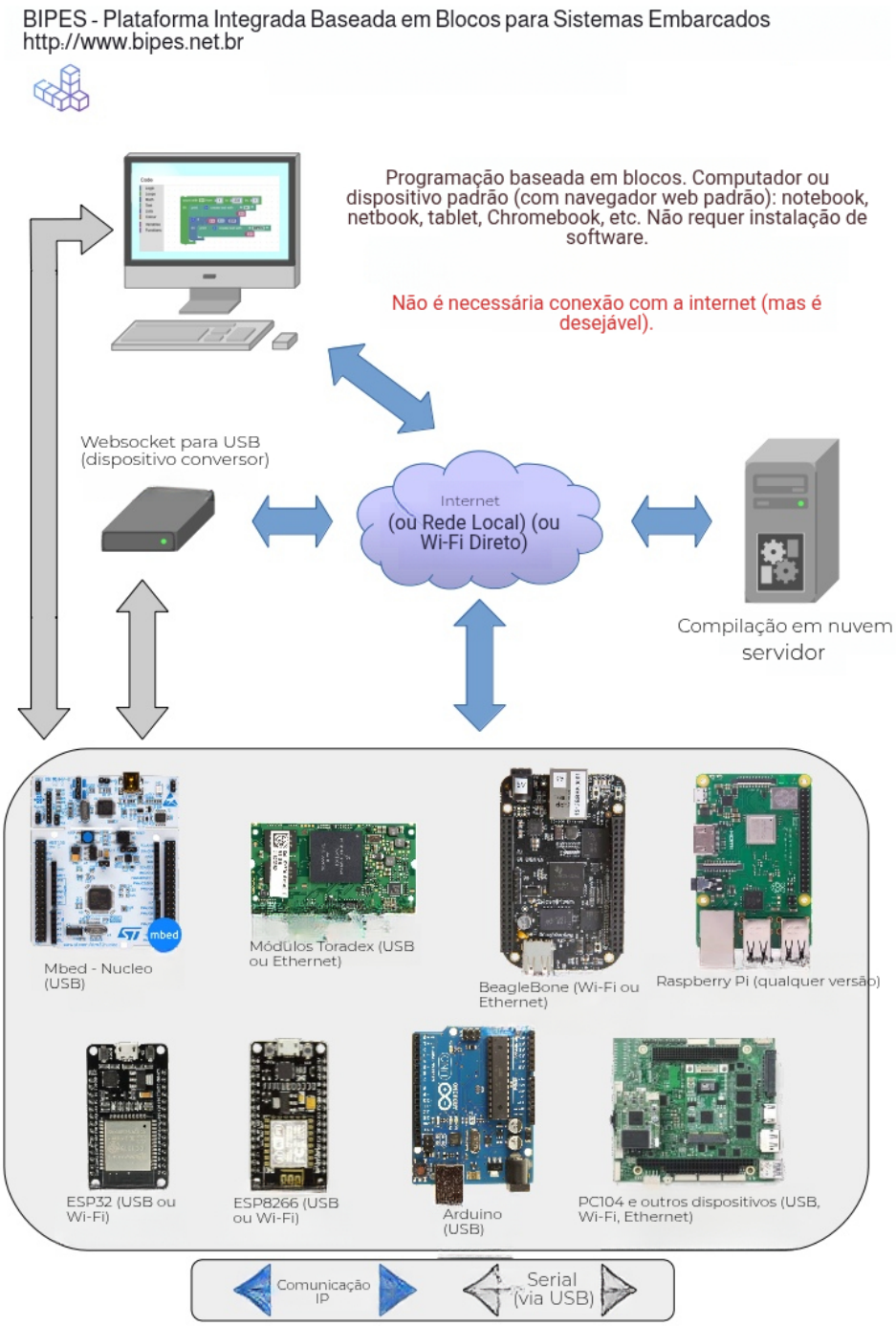
2.1.2 Plataforma BIPES

A plataforma *Block Based Integrated Platform for Embedded Systems* (BIPES) é uma solução de código aberto desenvolvida originalmente no Brasil para tornar a programação de sistemas embarcados mais acessível, especialmente no contexto educacional. Inspirada em projetos de código aberto como Google Blockly e MicroPython, a BIPES permite que os usuários criem programas arrastando blocos gráficos, os quais são automaticamente convertidos em código Python executável em placas como ESP32, ESP8266 e outras (Almeida *et al.*, 2021).

A proposta da plataforma BIPES é reduzir a barreira de entrada ao desenvolvimento embarcado, permitindo que estudantes explorem conceitos de programação, eletrônica e IoT de forma intuitiva e prática. Sua arquitetura permite a programação remota dos dispositivos, a execução em tempo real e a visualização de dados por meio de dashboards integrados, tornando-se uma ferramenta eficaz para o ensino de robótica e automação (Almeida *et al.*, 2021), e no contexto deste trabalho será utilizada principalmente para facilitar o ensino em níveis básicos e médios, onde os estudantes não tem um conhecimento tão aprofundado em programação ainda.

A Figura 2 apresenta um diagrama que resume o funcionamento da plataforma, desde a criação visual dos blocos até a execução embarcada do código gerado.

Figura 2 – Funcionamento da plataforma BIPES na programação de sistemas embarcados.



Fonte: Adaptado de BIPES Project (2024).

2.1.3 Robótica Móvel

A robótica móvel é um campo da robótica focado no estudo, projeto e construção de robôs capazes de se mover em seus ambientes, em vez de estarem fixos em um local como os

robôs industriais tradicionais (Siegwart; Nourbakhsh, 2004). Essa capacidade de locomoção abre um vasto leque de aplicações, desde exploração espacial e submarina até logística, assistência médica, segurança e, crucialmente para este trabalho, educação. Um robô móvel autônomo deve ser capaz de responder a três questões fundamentais para navegar com sucesso:

1. Onde estou? (Localização): Determinar sua posição e orientação (pose) dentro do ambiente, seja em relação a um mapa conhecido ou a um ponto de partida.
2. Aonde vou? (Objetivo/Planejamento de Missão): Definir ou receber um destino ou uma sequência de tarefas a serem cumpridas.
3. Como vou? (Planejamento de Caminho e Controle): Calcular uma trajetória segura e eficiente do ponto atual até o objetivo, evitando obstáculos, e gerar os comandos necessários para os atuadores (motores) seguirem essa trajetória.

Para responder a essas questões, a robótica móvel integra conceitos de diversas áreas, incluindo mecânica, eletrônica, ciência da computação (sensores, percepção, inteligência artificial, controle) e matemática.

2.1.4 Arquitetura distribuída em sistemas robóticos

Sistemas robóticos modernos tendem a adotar arquiteturas distribuídas, nas quais diferentes componentes de software executam de forma desacoplada e cooperativa, comunicando-se por meio de mecanismos padronizados de troca de mensagens. Esse modelo favorece modularidade, escalabilidade e reutilização de código, permitindo que sensores, atuadores, algoritmos de controle e interfaces de usuário sejam desenvolvidos e testados de forma independente. Em robótica, essa abordagem é especialmente relevante devido à heterogeneidade dos dispositivos envolvidos e à necessidade de integração entre diferentes níveis de abstração. (Quigley *et al.*, 2009)

O ROS consolidou-se como um dos principais frameworks a empregar arquitetura distribuída em robótica, organizando aplicações em nós interconectados que se comunicam por meio de tópicos, serviços e ações. Essa estrutura possibilita a separação clara entre aquisição de dados sensoriais, tomada de decisão e controle de atuadores, além de facilitar a integração com simuladores, ferramentas de visualização e bibliotecas de algoritmos. Dessa forma, o ROS tornou-se amplamente utilizado tanto em pesquisa quanto em ensino, servindo como base para diversas plataformas robóticas educacionais. (Quigley *et al.*, 2009)

Com a evolução do ROS para cenários embarcados e de recursos limitados, surgiram

extensões como o *micro-ROS*, que permitem aplicar os mesmos princípios de arquitetura distribuída em microcontroladores. Essa abordagem viabiliza a integração de dispositivos de baixo custo ao ecossistema ROS, mantendo compatibilidade conceitual e tecnológica com sistemas robóticos mais complexos. Em plataformas educacionais, essa característica é particularmente relevante, pois possibilita introduzir conceitos avançados de arquitetura de software robótico mesmo em ambientes com restrições de hardware, como microcontroladores utilizados para controle em tempo real. (Koubaa *et al.*, 2020)

2.2 Hardware

O "cérebro"de um robô móvel autônomo reside em seu sistema embarcado, um componente essencial de seu Hardware responsável por processar dados dos sensores, executar algoritmos de controle e inteligência artificial, e comandar os atuadores (Batista, 2013). A escolha do sistema embarcado é crucial, pois define as capacidades computacionais, o consumo de energia, o custo e a complexidade de desenvolvimento do robô. Para robôs de baixo custo com capacidades multissensoriais e potencial para IA, duas plataformas se destacam: NVIDIA Jetson e Raspberry Pi.

2.2.1 Robôs Multissensoriais

Robôs multissensoriais são sistemas robóticos equipados com uma variedade de sensores que lhes permitem perceber e interpretar o ambiente ao seu redor de forma mais completa e robusta (Romero, 2008). A integração de múltiplos sensores permite que o robô obtenha informações complementares sobre o ambiente, aumentando sua capacidade de percepção e tomada de decisão.

Sensores em robótica são essencialmente transdutores: dispositivos que convertem uma forma de energia ou propriedade física do ambiente em um sinal elétrico que pode ser processado pelo sistema de controle do robô (Oliveira *et al.*, 2017). Eles funcionam como os "sentidos"do robô, fornecendo dados brutos sobre variáveis como distância, presença, temperatura, umidade, luminosidade, entre outras.

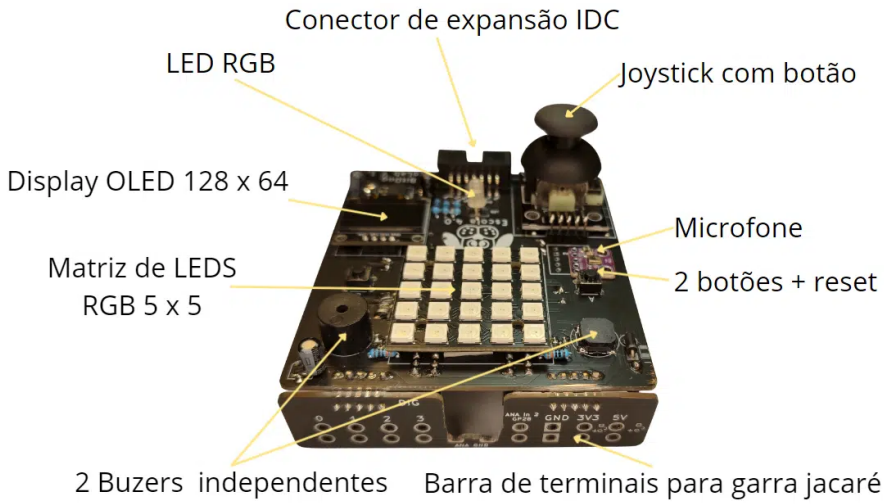
A redundância sensorial é um aspecto fundamental em robôs multissensoriais, pois permite a validação cruzada de informações e aumenta a confiabilidade do sistema. Por exemplo, um robô equipado com sensores ultrassônicos, infravermelhos e LIDAR pode obter medidas

de distância por diferentes princípios físicos, minimizando erros e aumentando a precisão da percepção espacial (Nagla *et al.*, 2014).

2.2.2 Placa BitDogLab como plataforma educacional

A BitDogLab é uma placa educacional desenvolvida no contexto brasileiro com o objetivo de facilitar o ensino de sistemas embarcados, eletrônica básica e programação, especialmente em ambientes de educação técnica e superior. Seu projeto prioriza acessibilidade, integração de periféricos e compatibilidade com microcontroladores amplamente utilizados no meio acadêmico, permitindo que estudantes tenham contato prático com conceitos fundamentais sem a necessidade de montagem complexa de circuitos externos (BitDogLab, 2023). A Figura 3 mostra a placa BitDogLab e sinaliza seus componentes.

Figura 3 – Placa educacional BitDogLab



Fonte: (BitDogLab, 2023)

Do ponto de vista didático, a BitDogLab incorpora diversos recursos integrados, como LEDs, botões, buzzer e interfaces de comunicação, possibilitando a exploração progressiva de conceitos de entrada e saída digital, temporização, interrupções e comunicação serial. Essa abordagem reduz a carga cognitiva inicial associada ao uso de protoboards e componentes discretos, favorecendo a aprendizagem ativa e a experimentação controlada, aspectos amplamente discutidos na literatura sobre o ensino de sistemas embarcados e computação física. (Gomes; Mendes, 2019)

Além disso, plataformas educacionais integradas como a BitDogLab alinham-se a abordagens pedagógicas baseadas no construcionismo e na aprendizagem baseada em projetos,

ao permitir que os estudantes desenvolvam artefatos funcionais desde as etapas iniciais do aprendizado. A possibilidade de utilizar linguagens de alto nível, ambientes visuais ou ferramentas de programação simplificadas contribui para a inclusão de alunos com diferentes níveis de experiência prévia, ampliando o alcance da robótica e da computação embarcada no contexto educacional. (Papert, 1980)

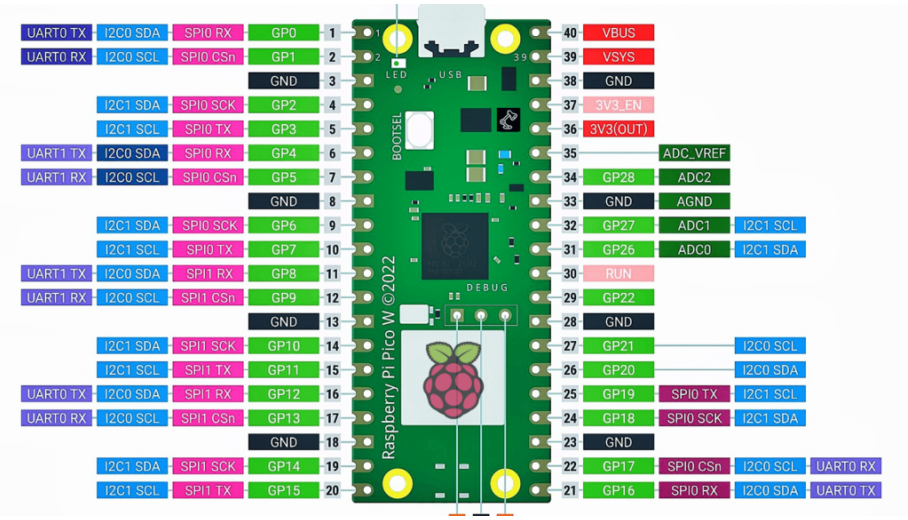
2.2.3 Raspberry Pi Pico W

O Raspberry Pi Pico W é uma placa de microcontrolador de baixo custo baseada no chip RP2040, desenvolvida pela Raspberry Pi Foundation para aplicações embarcadas e educacionais. Diferentemente dos computadores de placa única tradicionais, como o Raspberry Pi 4, o Pico W é projetado especificamente como uma unidade de controle compacta e eficiente, oferecendo conectividade sem fio integrada que amplia sua aplicabilidade em projetos de robótica e Internet das Coisas (IoT) (Raspberry Pi Ltd, 2023).

Equipado com um processador dual-core ARM Cortex-M0+ de até 133MHz, 2MB de memória flash e 264KB de SRAM, o Raspberry Pi Pico W combina desempenho suficiente para controle em tempo real e tarefas embarcadas com um consumo de energia reduzido (Raspberry Pi Ltd, 2023) . Além disso, a placa oferece conectividade Wi-Fi 2,4GHz compatível com o padrão IEEE802.11b/g/n, o que possibilita comunicação sem fio direta com redes e outros dispositivos (Raspberry Pi Ltd, 2023).

Uma das principais vantagens do Pico W para aplicações robóticas educacionais é sua interface de entrada e saída multifuncional (GPIO) com diversos protocolos de comunicação, como SPI, I²C, UART e ADC, permitindo integração direta com sensores, atuadores e módulos adicionais (Raspberry Pi Ltd, 2023). A Figura 4 mostra a placa Raspberry Pi Pico W, utilizada como unidade de controle auxiliar no robô deste trabalho, destacando suas interfaces e conectividade sem fio.

Figura 4 – Raspberry Pi Pico W com identificação dos conectores e interfaces embarcadas



Fonte: (Raspberry Pi Foundation, s.d.)

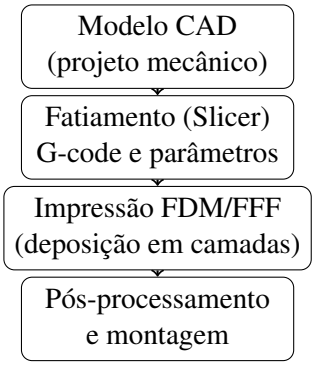
2.2.4 Impressão 3D para fabricação de robôs

A impressão 3D, no contexto da manufatura aditiva, refere-se a processos de fabricação nos quais objetos tridimensionais são produzidos por deposição sucessiva de material em camadas, a partir de um modelo digital. Diferentemente de métodos subtrativos ou formativos, a AM permite maior liberdade geométrica e rápida iteração de projetos, sendo amplamente utilizada em contextos educacionais e de pesquisa. A terminologia, os princípios fundamentais e a classificação desses processos são formalizados por normas técnicas internacionais, que fornecem uma base conceitual padronizada para o uso da impressão 3D em engenharia. (ISO/ASTM, 2021)

Entre as tecnologias de manufatura aditiva, a extrusão de material (conhecida como FDM ou FFF) destaca-se por sua ampla disseminação em impressoras de baixo custo e fácil operação. Nesse processo, um filamento termoplástico é aquecido e extrudado por um bico, sendo depositado camada a camada até a formação da peça final. Essa tecnologia tornou-se especialmente relevante para o desenvolvimento de plataformas robóticas educacionais, pois permite a fabricação de suportes mecânicos, carcaças, estruturas modulares e encaixes personalizados com custo reduzido e curto tempo de fabricação, favorecendo ciclos rápidos de prototipação e testes (Attaran, 2017). A Figura 5 ilustra de forma resumida o fluxo de fabricação de peças por impressão 3D.

Do ponto de vista estrutural, peças produzidas por extrusão de material apresentam características mecânicas anisotrópicas, ou seja, sua resistência depende da orientação das

Figura 5 – Fluxo de fabricação de peças por impressão 3D.



Fonte: Elaborado pelo Autor

camadas e dos parâmetros de impressão adotados. Fatores como altura de camada, orientação da peça, temperatura de extrusão e densidade de preenchimento influenciam diretamente a rigidez e a durabilidade dos componentes. Em robôs móveis educacionais, essas limitações devem ser consideradas no projeto de partes sujeitas a esforços mecânicos, como suportes de rodas, bases estruturais e torres de sensores, garantindo confiabilidade sem comprometer o baixo custo da plataforma. (Design. . . , 2015)

2.3 Robótica Educacional

A robótica educacional utiliza robôs como ferramentas de apoio ao ensino, promovendo a aprendizagem ativa, interdisciplinar e prática. Ao montar e programar robôs, os alunos desenvolvem habilidades em ciência, tecnologia, engenharia e matemática (STEM), além de estimular o raciocínio lógico, a criatividade e a resolução de problemas. Neste trabalho serão explorados conceitos como aprendizagem ativa, interdisciplinaridade e aprendizagem baseada em projetos no contexto da robótica educacional.

2.3.1 Plataformas robóticas educacionais de baixo custo

Plataformas robóticas educacionais têm sido amplamente utilizadas como ferramentas de apoio ao ensino de programação, eletrônica e sistemas embarcados, por permitirem a aplicação prática de conceitos teóricos em atividades experimentais. No entanto, muitas soluções comerciais apresentam custo elevado ou arquitetura fechada, o que limita sua adoção em instituições públicas e restringe possibilidades de adaptação e expansão por parte de professores e estudantes. Nesse contexto, iniciativas baseadas em hardware e software abertos têm ganhado destaque por promoverem maior acessibilidade e flexibilidade pedagógica. (Benitti, 2012)

Robôs educacionais de baixo custo geralmente priorizam modularidade, simplicidade construtiva e facilidade de manutenção, utilizando componentes amplamente disponíveis no mercado e técnicas de fabricação acessíveis, como a impressão 3D. Essas características permitem que a plataforma seja reproduzida, modificada e reparada localmente, além de incentivar a experimentação e o aprendizado ativo. Estudos indicam que soluções abertas e de baixo custo favorecem maior engajamento dos alunos, uma vez que reduzem barreiras técnicas e financeiras ao uso contínuo da robótica no ambiente educacional. (Benitti, 2012)

Entre as plataformas robóticas educacionais amplamente difundidas, destaca-se o TurtleBot, que se tornou uma referência no ensino de robótica móvel integrada ao ROS. Apesar de seu valor pedagógico, o custo relativamente elevado e a dependência de componentes específicos podem dificultar sua adoção em larga escala. Assim, propostas inspiradas nesse modelo, mas baseadas em hardware mais acessível e arquiteturas abertas, buscam preservar os benefícios educacionais da robótica móvel enquanto ampliam o alcance para diferentes níveis de ensino e contextos institucionais. (Open Robotics, 2023)

2.3.2 Aprendizagem Ativa com Robótica Educacional

A robótica educacional promove um ambiente de aprendizagem ativa, onde os estudantes deixam de ser receptores passivos de conhecimento para se tornarem protagonistas do próprio processo educativo. A construção e programação de robôs estimula a experimentação, a resolução de problemas e o pensamento crítico. Essa abordagem está fortemente alinhada com as ideias de Piaget e Papert (Piaget, 1976; Papert, 1980), especialmente a Teoria do Construcionismo, segundo a qual o conhecimento é melhor construído quando o aprendiz está envolvido na criação de artefatos concretos e significativos para si mesmo e para os outros.

Além disso, trabalhar com robótica desperta o interesse e o engajamento dos alunos, pois integra diferentes áreas do conhecimento como matemática, física, computação e engenharia de forma prática e contextualizada. Segundo (Almeida; Valente, 2012), o uso da robótica permite que o aluno se envolva cognitivamente com o conteúdo, estabelecendo relações entre teoria e prática, o que favorece aprendizagens mais duradouras e significativas.

3 TRABALHOS RELACIONADOS

Esta seção apresenta uma análise de trabalhos relacionados ao desenvolvimento de robôs multissensoriais de baixo custo para aplicações educacionais, destacando suas principais características, abordagens e contribuições para a área.

3.1 Minicurso de Introdução à Robótica Educacional (Azevedo *et al.*, 2010)

O trabalho de Azevedo, Aglaé e Pitta apresenta uma abordagem pedagógica para o ensino de robótica utilizando kits educacionais como o LEGO Mindstorms. Os autores introduzem conceitos fundamentais de robótica de forma abstrata e prática, abordando componentes essenciais como controladores, servo motores, atuadores e sensores, além da programação necessária para solucionar problemas específicos (Azevedo *et al.*, 2010). A metodologia proposta foca na montagem de robôs seguida pelo controle e programação, permitindo que estudantes compreendam gradualmente os princípios da robótica sem se preocupar com detalhes mecânicos complexos ou algoritmos avançados. Este trabalho se destaca pela abordagem didática e pela utilização de kits comerciais que, embora eficazes para o ensino, apresentam custo elevado quando comparados a soluções abertas como a proposta no presente trabalho.

O trabalho contou com o desenvolvimento de um software para agrupamento das atividades de ensino e controle dos componentes, mostrado na Figura 6.

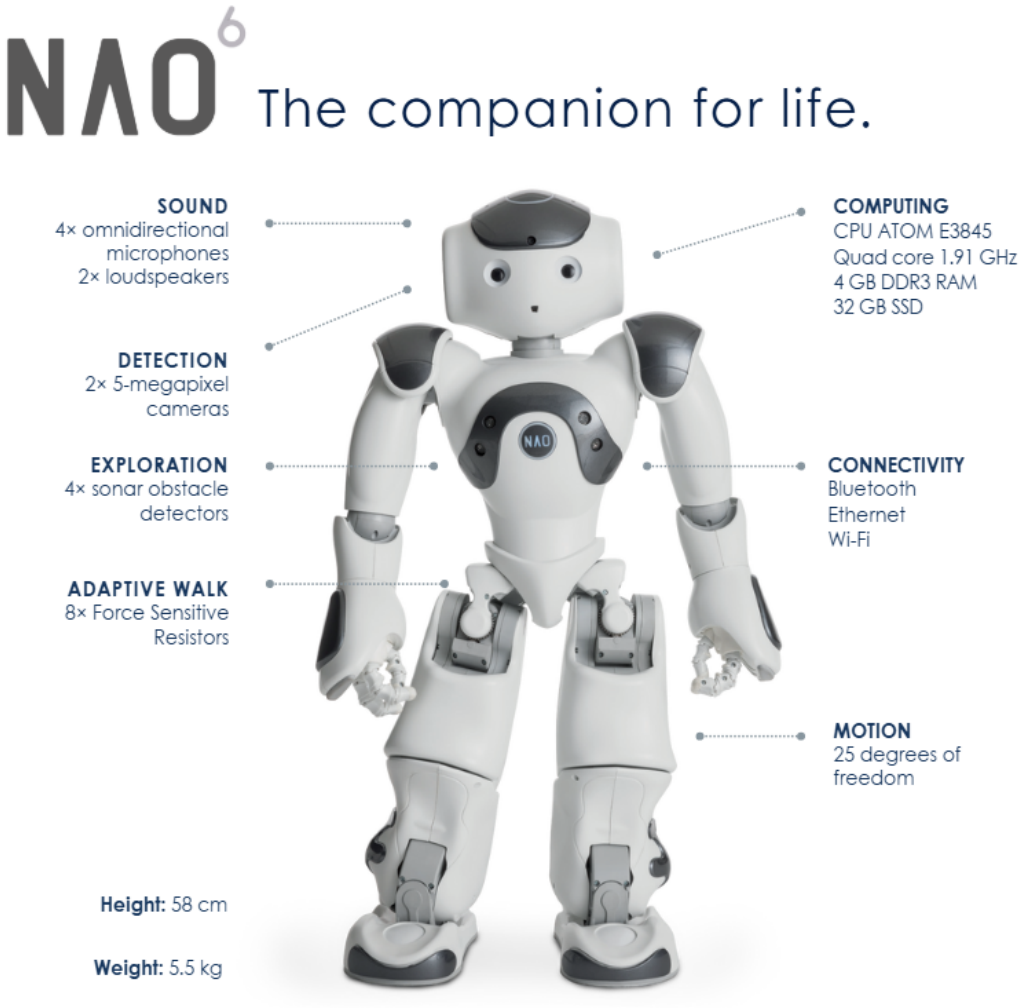
Figura 6 – Software Roboeduc.

Fonte: Azevedo *et al.* (2010)

3.2 Robô NAO6 (Robotics, 2024a)

O robô NAO6, desenvolvido pela Aldebaran Robotics, é um robô de característica humanoide como mostrado na Figura 7, amplamente reconhecido e utilizado em ambientes educacionais e de pesquisa em todo o mundo (Robotics, 2024a). Ele se destaca por sua plataforma aberta e totalmente programável, com uma interface programada para simular uma interação natural com pessoas, incluindo funcionalidades como reconhecimento de fala e diálogo em diversas línguas, e uma gama de sensores como microfones direcionais, câmeras, sensores de toque, força e também autuadores do tipo servo motor. O NAO6 é frequentemente empregado para ensinar programação, robótica e até mesmo para auxiliar em terapias e educação especial, devido à sua capacidade de engajar e motivar alunos, se portando como um dispositivo de categoria STEM (Ciência, Tecnologia, Engenharia e Matemática).

Figura 7 – Robô Aldebaran NAO6.



Fonte: Maxtronics (2024)

O robô NAO6 também possui possibilidade de programação baseada em blocos, uma funcionalidade proposta neste trabalho para inclusão do ensino básico. O projeto do robô proposto neste trabalho busca democratizar o acesso à robótica educacional, utilizando componentes de hardware mais acessíveis, e integrando-se a plataformas No-Code como o BIPES. Enquanto o NAO6 oferece uma solução "pronta para uso"com um alto nível de integração, o presente trabalho foca na construção de uma plataforma flexível e de baixo custo, que permite aos estudantes uma compreensão mais aprofundada dos componentes e da arquitetura de um robô multissensorial, alinhando-se com a proposta de um robô educacional acessível e replicável, além disso, o robô NAO6 possui um preço muito elevado e fora da realidade brasileira, ultrapassando valores de até 10,000 dólares americanos.

3.3 Robótica Educacional como recurso pedagógico para ensino de matemática e computação na educação básica (Araujo, 2023)

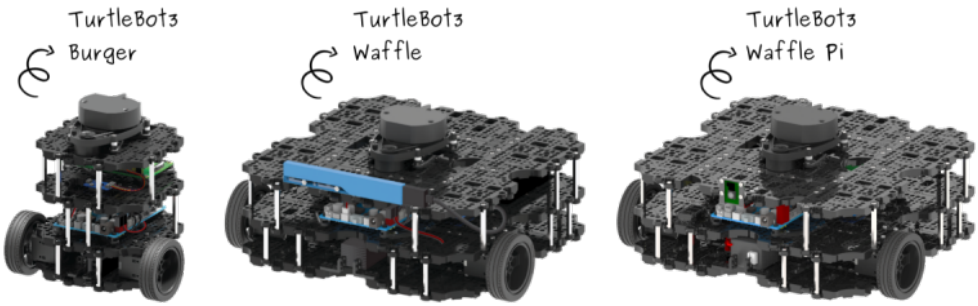
O trabalho de (Araujo, 2023) aborda dificuldades no ensino de conceitos como matemática e computação na educação básica, e baseando-se nisso propõe o uso de robótica como um recurso pedagógico para tornar o aprendizado mais envolvente aos alunos. O trabalho de Araújo tem como principal objetivo desenvolver uma metodologia de ensino utilizando kits de robótica acessíveis baseados em Arduino, assim como a criação de uma capacitação para professores da rede pública visando o uso destes métodos no plano de ensino. Assim como este trabalho, Araújo buscou focar em componentes acessíveis e usou robótica como ferramenta educacional, no entanto, abordou apenas ensino básico e não foca necessariamente na construção de um robô como é o caso deste trabalho, assim como usa hardwares muito mais simples e sem o foco na construção de um sistema central como o ROS.

3.4 OpenRobotics Turtlebot (Robotics, 2024c)

O Turtlebot é a alternativa mais popular atualmente no mercado (Robotics, 2024b), sendo uma alternativa Open Source para pesquisa, ensino e desenvolvimento, o Turtlebot é uma plataforma robótica móvel equipada com múltiplos sensores comuns na robótica, como IMU, LiDAR 2D, camera stereo, infravermelho, dentre outros. Seu sistema roda em cima do sistema operacional ROS 2, assim como este trabalho, possuindo suas funcionalidades instaláveis e configuráveis por meio de pacotes, além de possuir barramentos expansíveis para utilização de sensores externos. Apesar de se considerar uma alternativa de baixo custo (Robotics, 2024c), o Turtlebot 4 ainda está fora da realidade brasileira. Custando mais de 10 mil reais, acaba se tornando uma alternativa no Brasil levando em consideração o que oferece, diferente deste trabalho, que utiliza impressão 3D e componentes modulares acessíveis na construção do robô.

Como mostrado na Figura 8, o Turtlebot possui uma estrutura modular e versátil, permitindo a fácil manutenção e adição de componentes, um dos motivos de ser considerado um dos melhores modelos tanto para pesquisa e desenvolvimento quanto para ensino de robótica.

Figura 8 – Turtlebot e suas variações.



Fonte: Robotics (2024c)

3.5 Análise Comparativa

O trabalho de Azevedo *et al.* (2010) foca em lecionar aulas de Robótica utilizando o kit comercial LEGO Mindstorms, assimilando-se com o trabalho aqui proposto no quesito de lecionar utilizando robôs montados, no entanto, baseia-se em um kit com pouca versatilidade, alto custo e que carece de abordagens mais avançadas da robótica, como o desenvolvimento do controle PID de atuadores.

Em Robotics (2024a) é comercializado um robô proprietário com características humanóides com o foco em tornar o ensino de robótica mais dinâmico e amigável à crianças, no entanto, peca no elevado custo e carece de versatilidade em sua programação por utilizar de softwares proprietários e não ser modular para o uso de diferentes sensores e atuadores.

O trabalho Araujo (2023) não aborda diretamente a construção de uma plataforma robótica para o ensino, mas propõe práticas pedagógicas para o uso da robótica educacional como auxílio para o ensino de disciplinas como Física e Matemática, apresentando valiosas abordagens no uso de Robótica para fins educacionais.

Já Robotics (2024c) comercializa uma versão final do TurtleBot similar ao trabalho aqui proposto, porém, com custos de produção e venda mais elevados e sem disponibilização no Brasil.

O Quadro 1 destaca os principais aspectos dos trabalhos relacionados (foco principal, hardware, software, sensores, custo, aplicação, mobilidade e programação) no contexto do trabalho proposto e os comparam.

Quadro 1 – Comparação entre o trabalho proposto e trabalhos relacionados

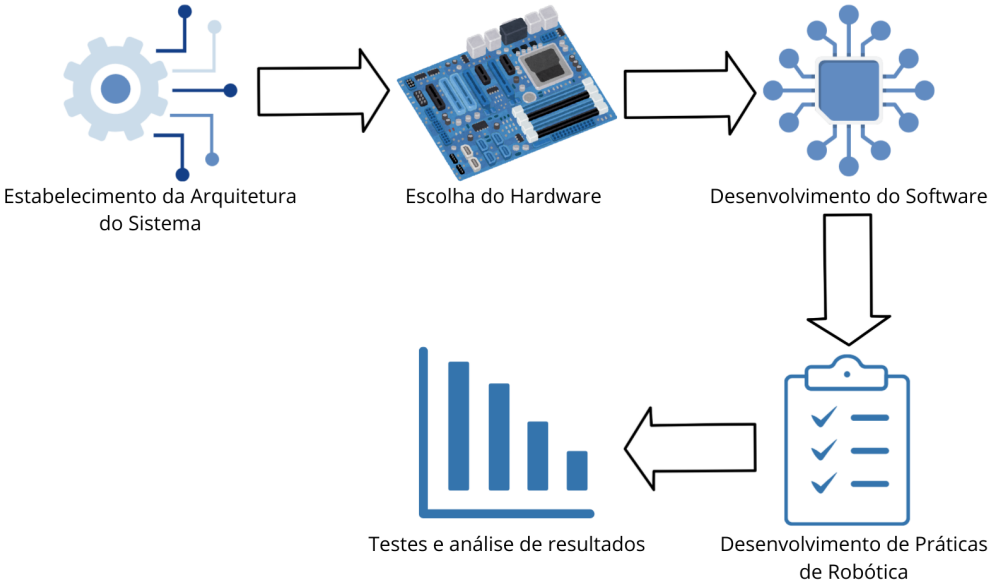
Características	**Trabalho Proposto**	**(Azevedo *et al.*, 2010)**	**(Robotics, 2024a)**	**(Araujo, 2023)**	**(Robotics, 2024c)**
Foco Principal	Robô multissensorial de baixo custo para educação	Robótica educacional com kits comerciais	Robô humanóide com foco em robótica educacional e interação humana	Uso de robótica no ensino básico	Plataforma robótica multissensorial móvel de código aberto
Hardware	Raspberry Pi Pico W	LEGO Mindstorms	Intel ATOM E3845	Genéricos	Raspberry Pi 4
Software	ROS 2, Bipes (No-Code)	Ambiente proprietário LEGO	ROS, proprietário	BareMetal	ROS 2
Sensores	Todos, modular	Básicos (LEGO)	Câmera, atuadores	Variados	Todos, modular
Custo	Baixo	Alto	Muito Alto	N/A	Médio / Alto
Aplicação	Pesquisa, ensino e desenvolvimento	Ensino básico	Ensino básico	Ensino básico e pesquisa	Pesquisa, ensino e desenvolvimento
Mobilidade	Robótica móvel	Robótica móvel	Robótica móvel	Estático	Robótica móvel
Programação	Flexível	Proprietária	No-Code baseada em blocos	Variados	Flexível

Fonte: Elaborado pelo Autor

4 METODOLOGIA

Neste capítulo, será detalhado o desenvolvimento da plataforma, desde a escolha e aplicação do Hardware até o desenvolvimento do Software e Firmware. A Figura 9 apresenta uma visão geral das etapas metodológicas adotadas neste trabalho, desde a concepção do sistema até sua implementação e validação. Essas etapas fazem-se necessárias para alcançar os objetivos propostos neste trabalho.

Figura 9 – Fluxograma dos procedimentos metodológicos.



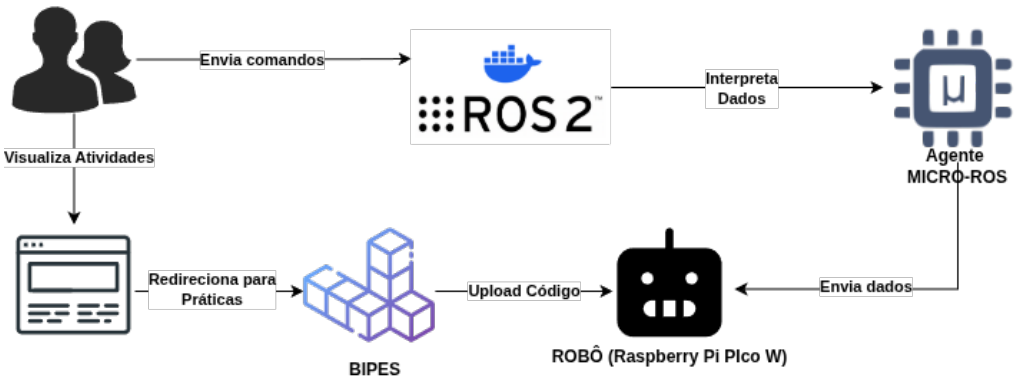
Fonte: Elaborado pelo Autor

4.1 Modelagem do sistema

Na Figura 10 está ilustrado o modelo simplificado da proposta deste trabalho. O fluxo do sistema inicia quando o usuário interaje com o website desenvolvido. Este website exibe diversas atividades pedagógicas de diferentes áreas da robótica, mostrando o passo a passo para o seu desenvolvimento, ao escolher sua atividade, o usuário receberá uma série de instruções para a realização da mesma. Em determinada etapa da atividade, o usuário será instruído a construir a modelagem do sistema utilizando a plataforma de programação baseada em blocos BIPES, e será redirecionada para a mesma em um modelo com blocos já desenvolvidos para diferentes componentes, como atuadores e sensores. Ao arranjar os blocos da maneira que o

usuário considerar apropriada, o usuário irá fazer o *upload* do programa gerado diretamente pelo BIPES para a placa do robô, ao iniciar o programa, o robô irá exibir seu endereço IP em um terminal web que estará em exibição no BIPES e por meio deste IP, o usuário será capaz de fazer a conexão com o MicroROS que está em execução no microcontrolador do robô por meio do ROS 2 em sua máquina, e assim enviar comandos e ler dados do robô em tempo-real.

Figura 10 – Modelo de sistema proposto.



Fonte: Elaborado pelo Autor

Portanto, essa foi a arquitetura de sistema escolhida para o trabalho proposto, visando um uso eficiente tanto do software como do hardware.

4.2 Escolha do Hardware

O Quadro 2 apresenta os principais componentes de hardware utilizados na plataforma robótica desenvolvida neste trabalho, organizados de acordo com sua função no sistema. A seleção desses componentes levou em consideração critérios como baixo custo, disponibilidade no mercado e compatibilidade com arquiteturas de robótica móvel baseadas em ROS e micro-ROS.

Quadro 2 – Componentes de hardware da plataforma robótica

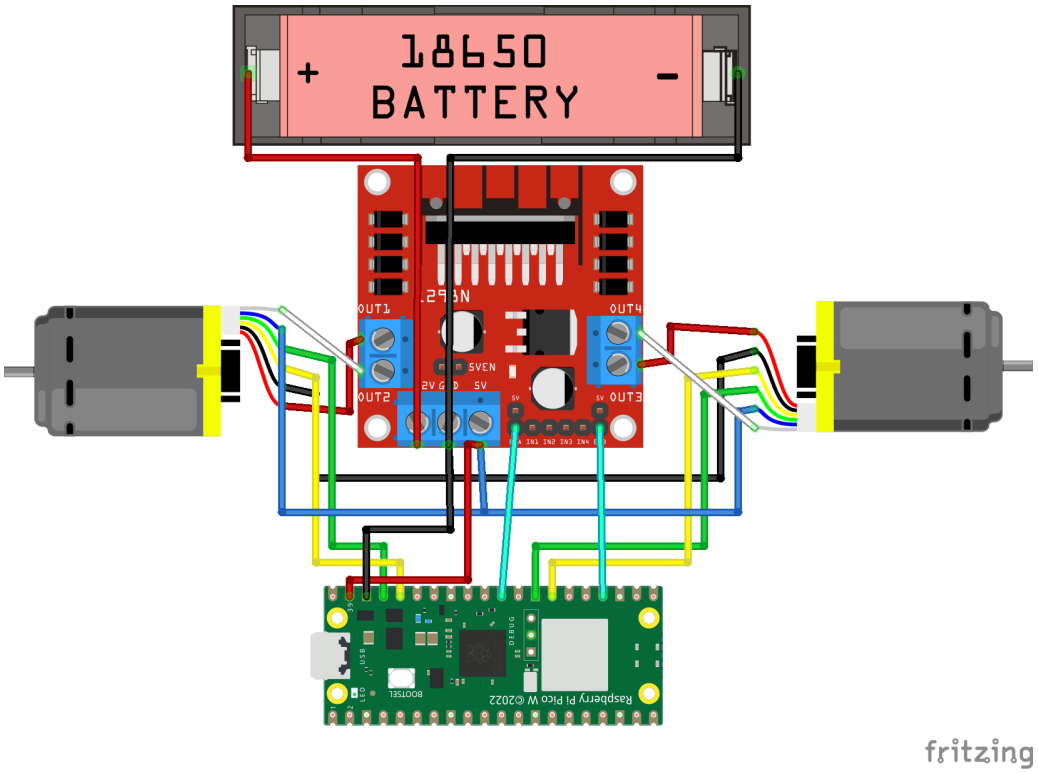
Categoria	Componente	Justificativa de uso
Unidade de processamento	Raspberry Pi Pico W	Microcontrolador de baixo custo com Wi-Fi integrado compatível com micro-ROS, permitindo comunicação com sistemas baseados em ROS.
Plataforma embarcada educacional	BitDogLab	Placa educacional que integra periféricos essenciais para ensino de sistemas embarcados, reduzindo a complexidade de montagem e facilitando o uso em ambientes didáticos.
Sensoriamento	LIDAR LDS02RR	Sensor de varredura a laser, permitindo mapeamento do ambiente, detecção de obstáculos e experimentação com algoritmos de navegação.
Atuação	Motores DC JGA25-370 com encoder e pneus (2 unidades)	Motores com encoders integrados, possibilitando controle de velocidade e estimativa de deslocamento.
Placa de controle elétrico	Driver L298N	Controlador elétrico de fácil integração e ampla utilização em projetos educacionais, adequado para acionamento de motores DC controle elétrico geral.
Estrutura do chassi	Carcaça em ABS impressa em 3D	Estrutura mecânica personalizada, permitindo baixo custo de fabricação, fácil replicação e adaptação do projeto para diferentes contextos.
Alimentação	Bateria de lítio 12V	Bateria recarregável de 12V.

Fonte: Elaborado pelo Autor

A estrutura geral do funcionamento elétrico da plataforma é ilustrada na Figura 11, onde é mostrada de forma resumida a conexão elétrica do sistema, onde uma bateria alimenta o

módulo L298N, onde o módulo além de alimentar a Raspberry Pi Pico W, também alimenta os motores, encoders e recebe os sinais para o controle dos atuadores da plataforma.

Figura 11 – Esquemático do Sistema.



Fonte: Elaborado pelo Autor

4.2.1 Unidade de processamento

A Raspberry Pi Pico W foi escolhida como unidade de processamento devido a sua vasta documentação com comunidade ativa, baixo-custo e eficiência energética (Raspberry Pi Foundation, s.d.). Além disso, o microcontrolador RP2040, presente na Raspberry Pi Pico W, mostrou-se adequado para o desenvolvimento da plataforma robótica proposta neste trabalho por combinar capacidade de processamento, flexibilidade de interfaces e baixo custo, características relevantes para satisfazer o sistema aqui proposto. O chip possui arquitetura dual-core ARM Cortex-M0+, permitindo a distribuição de tarefas como leitura de sensores e a transmissão de dados sem fio, o que contribui para maior previsibilidade temporal e organização do software embarcado (Raspberry Pi Ltd, 2023).

4.2.2 Plataforma embarcada

A placa educacional BitDogLab incorpora uma ampla variedade de periféricos, como botões, buzzers, LED RGB, joystick, display OLED, matriz de LEDS, microfone e sensor de toque (BitDogLab, 2023). Essa ampla disponibilidade de sensores, com suas conexões físicas já estabelecidas com o microcontrolador proporciona uma experiência mais eficiente no uso de todos os sensores mencionados anteriormente em práticas pedagógicas, pois deste modo os usuários podem se preocupar menos com arranjo de fios e direcionar o foco na resolução do problema por meio de programação e arquitetura de software, além de prevenir conexões instáveis.

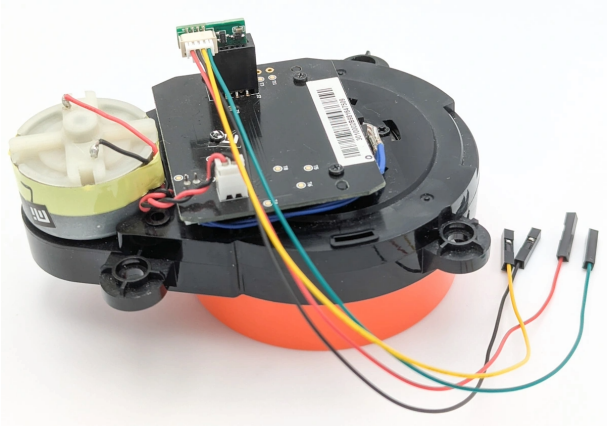
A BitDogLab também proporciona uma experimentação ampla e rápida, pois conta com expansores que permitem o uso de conectores jacaré, *Insulation Displacement Connector* (IDC) e jumpers, adaptando-se a necessidades de prototipagem rápida (BitDogLab, 2023).

4.2.3 Sensoriamento

O único sensor que é incluso na plataforma aqui proposta de forma modular é o *Light Detection and Ranging* (LIDAR) LDS02RR. Esse LIDAR é reaproveitado de um robô aspirador da marca Xiaomi, pois é comercializado de forma individual e possui a capacidade semelhante à sensores de alto custo do mercado (Reps, 2018), mostrando-se uma alternativa acessível para aplicações de robótica envolvendo mapeamento *Simultaneous Location and Mapping* (SLAM), tornando possível a implementação de práticas de nível mais avançado para ensino superior.

Em Maker's Pet (2024) é apresentada uma adaptação open source para o uso desse LIDAR com microcontroladores comuns, utilizando uma *Printed Circuit Board* (PCB) personalizada considerada para o trabalho aqui proposto mostrada na Figura 12.

Figura 12 – Adaptador para LIDAR LDS02RR.



Fonte: (Maker's Pet, 2024)

4.2.4 Atuação

Como atuadores principais do robô, foram escolhidas duas unidades do kit de motor DC JGA25-370 com encoders e pneus, uma unidade para cada lado do robô. Esse kit de motor é genérico e oferece um encoder capaz de detectar e definir a velocidade de rotação, além de mapear e salvar o estado e posição do motor, tornando possível o desenvolvimento de um sistema de auto-navegação ou navegação para locais pré-definidos, possibilitando uma vasta gama de atividades pedagógicas envolvendo atuadores e mapeamento.

Como mostrado na Figura 13, o kit fornece um Motor DC de 12V, um Encoder já soldado ao motor, um pneu em miniatura e um cabo com conector.

Figura 13 – Kit Motor DC JGA25-370 com encoder.

Fonte: Adaptado de Aliexpress

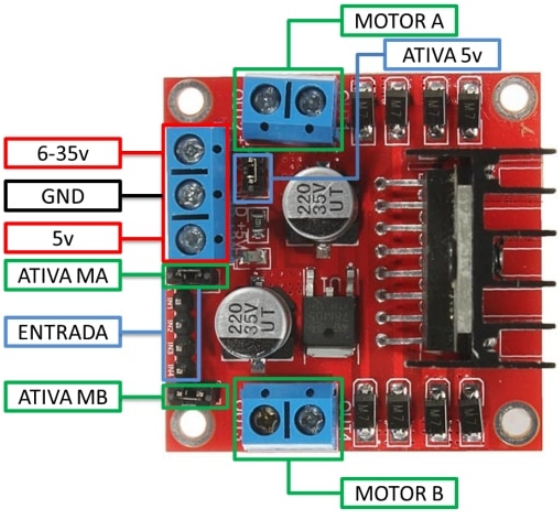
4.2.5 Placa de controle elétrico

O módulo L298N foi escolhido como placa de controle elétrico principal para o sistema, pois apresenta uma ampla faixa de tensão (5 a 35V), controle de saída de 2A por canal e pode controlar até 2 motores DC (MakerHero, 2024), satisfazendo a necessidade do projeto.

Esse módulo é baseado no chip L298N, feito para controlar cargas indutivas como relés, solenoides, motores DC e motores de passo. Ele possui terminais parafusáveis para fácil instalação, assim como buracos para fixação por meio de parafusos, mostrando-se uma alternativa completa por um custo significativamente inferior ao de alternativas comerciais equivalentes (MakerHero, 2024). Como mostrado na Figura 14, o módulo possui terminais para 2 motores, entradas para controle de direção da rotação dos motores, uma saída 5V, controle *Pulse Width Modulation* (PWM) para ambos os motores e uma entrada de alimentação que suporta até 35V.

No contexto deste trabalho, o módulo L298N se mostrou uma alternativa completa capaz de satisfazer tanto o controle dos motores quando a alimentação de todo o circuito em si, podendo receber a alimentação de 12V da bateria e redistribuir em 5V estáveis para os demais componentes, como o LIDAR e a BitDogLab.

Figura 14 – Módulo L298N.



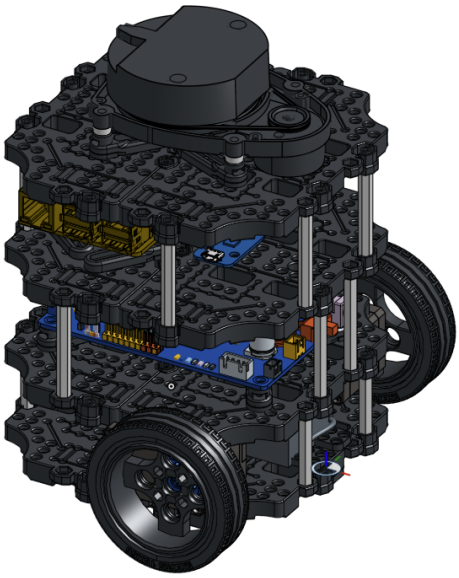
Fonte: (MakerHero, 2024)

4.2.6 Estrutura do chassi

O material *Acrylonitrile Butadiene Styrene* (ABS) foi selecionado para a fabricação do chassi da plataforma robótica por apresentar um equilíbrio adequado entre resistência mecânica, durabilidade e viabilidade de fabricação por impressão 3D (Gibson *et al.*, 2015). Em comparação a outros materiais utilizados nesse processo, o ABS oferece mais resistência a impactos e melhor estabilidade térmica, características relevantes para aplicações em robótica móvel, nas quais o chassi está sujeito a vibrações, esforços mecânicos e variações de temperatura. Além disso, o ABS possibilita acabamento mais robusto e suporta processos de pós-tratamento, como colagem e ajustes mecânicos, facilitando a manutenção e a adaptação do projeto.

Para os processos de impressão 3D das partes, chassi da plataforma foi desenvolvido a partir de modelos 3D disponibilizados gratuitamente na plataforma CAD Onshape. Esses modelos serviram como base para a fabricação da estrutura em impressão 3D, sendo utilizados e, quando necessário, adaptados às dimensões e aos componentes eletrônicos do sistema proposto. Os modelos foram obtidos a partir de repositório público na plataforma Onshape (Onshape Community, 2024), a Figura 15 mostra uma captura da página do repositório público, em uma pré-visualização dos componentes montados e utilizados nesse trabalho.

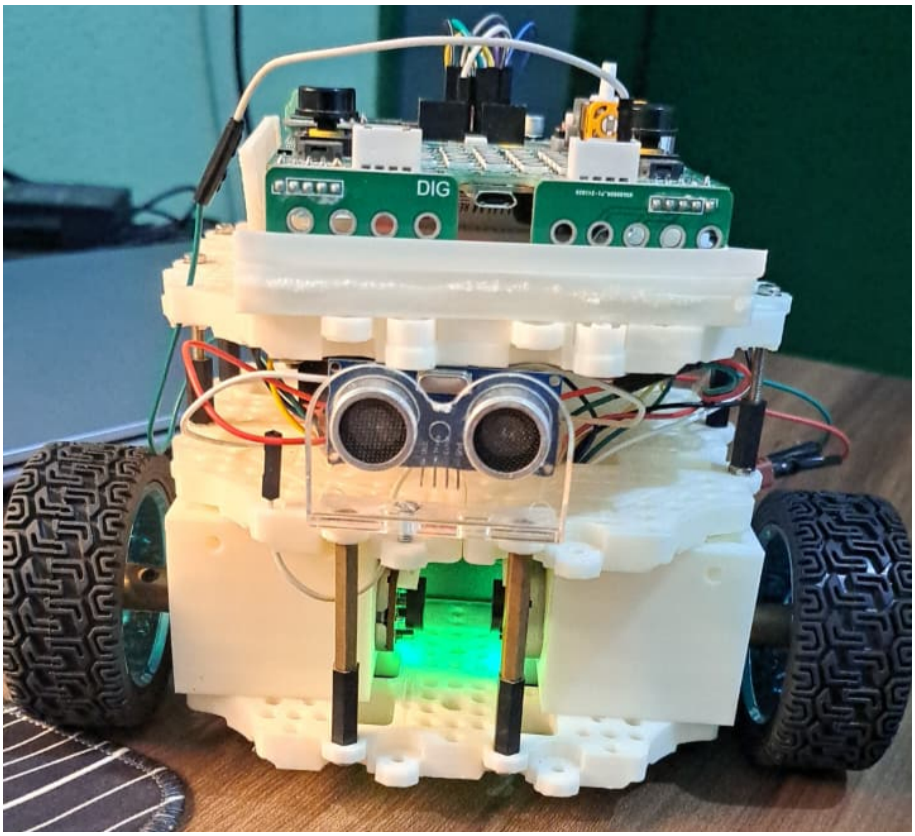
Figura 15 – Modelo 3D Turtlebot gratuito no CAD Onshape.



Fonte: (Onshape Community, 2024)

Já a Figura 16 apresenta o protótipo desenvolvido neste trabalho após modificações e adaptações dos modelos utilizados da plataforma Onshape.

Figura 16 – Protótipo físico da plataforma robótica.



Fonte: Elaborado pelo Autor

4.2.7 Alimentação

Para a alimentação da plataforma, foi utilizada uma bateria de Lítio de 12V, 1000mAh com corrente máxima de descarga de até 10A. Características de baterias comumente utilizadas em drones, mostrando-se uma alternativa eficaz para uso em múltiplos atuadores e suportando múltiplas recargas.

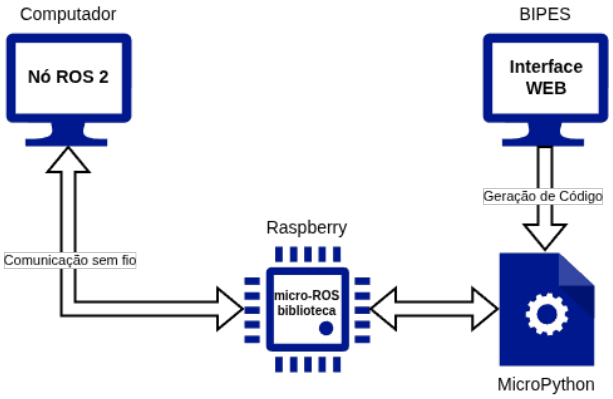
4.3 Desenvolvimento do Software

Como no atual estado da arte não há alternativas otimizadas para a adaptação de um código em micro-ROS para MicroPython, para o desenvolvimento deste trabalho é necessária a criação de uma biblioteca exclusiva para satisfazer essa necessidade, pois a plataforma BIPES gera códigos somente em MicroPython. Ademais, também é criado um website para agrupamento das atividades pedagógicas com instruções e redirecionamento para o BIPES.

4.3.1 Biblioteca micro-ROS para MicroPython

Para o uso do BIPES e do ROS 2 no mesmo projeto, deve ser feita uma integração entre micro-ROS e MicroPython para a Raspberry Pi Pico W. Nesse sentido, é necessária a criação de uma biblioteca visando criar uma ponte direta entre ROS 2 e BIPES usando micro-ROS, eliminando a necessidade de pontes *Message Queuing Telemetry Transport* (MQTT) e permitindo o uso do BIPES para programação visual com o MicroPython. A Figura 17 ilustra de forma resumida a arquitetura da biblioteca a ser criada, a biblioteca funciona como qualquer outra, é escrita em C e define as principais funções para um fluxo de comunicação comum micro-ROS, como a criação de nós, atuando como uma camada de abstração expondo uma *Application Programming Interface* (API) de alto nível permitindo o uso de micro-ROS com MicroPython para sistemas embarcados.

Figura 17 – Arquitetura da Biblioteca Micropython.



Fonte: Elaborado pelo Autor

A biblioteca desenvolvida encontra-se apresentada no Apêndice A, sendo seus principais trechos mostrados na Listagem 4. A função `microros.init()` é responsável pela inicialização da infraestrutura micro-ROS no dispositivo, configurando o tipo de transporte de comunicação e preparando a camada responsável pela troca de mensagens com o *micro-ROS Agent*. Por padrão, o transporte *Universal Asynchronous Receiver/Transmitter* (UART) é utilizado, por apresentar simplicidade de configuração e ampla compatibilidade com microcontroladores. Essa função abstrai detalhes de baixo nível da inicialização do micro-ROS, permitindo que o usuário realize a configuração do sistema por meio de uma única chamada em MicroPython, o detalhamento da função é mostrado na Listagem 1.

Já a criação de nós ROS é realizada por meio da função `microros.Node()`, que

permite instanciar um nó a partir do ambiente MicroPython. Essa abordagem mantém a compatibilidade conceitual com o ROS 2, preservando o modelo baseado em nós, tópicos, publicadores e assinantes. Dessa forma, estudantes e desenvolvedores podem utilizar os princípios do ROS em sistemas embarcados sem a necessidade de programação direta em linguagens de baixo nível, como C ou C++, o detalhamento da função é mostrado na Listagem 1.

Código-fonte 1 – Funções principais da biblioteca micro-ROS

```
static mp_obj_t microros_init(size_t n_args, const mp_obj_t *args) {
    const char *transport = "uart";  // default to UART
    if (n_args > 0) {
        transport = mp_obj_str_get_str(args[0]);
    }
    // Set transport type
    extern int microros_set_transport_impl(const char *transport);
    if (microros_set_transport_impl(transport) != 0) {
        mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to set transport type"));
    }
    // Initialize transport layer
    if (microros_init_transport_layer() != 0) {
        mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to initialize micro-ROS transport"));
    }

    mp_printf(&mp_plat_print, "[micro-ROS] Transport '%s' initialized\n", transport);
    return mp_const_none;
}

// microros.Node(name)
```

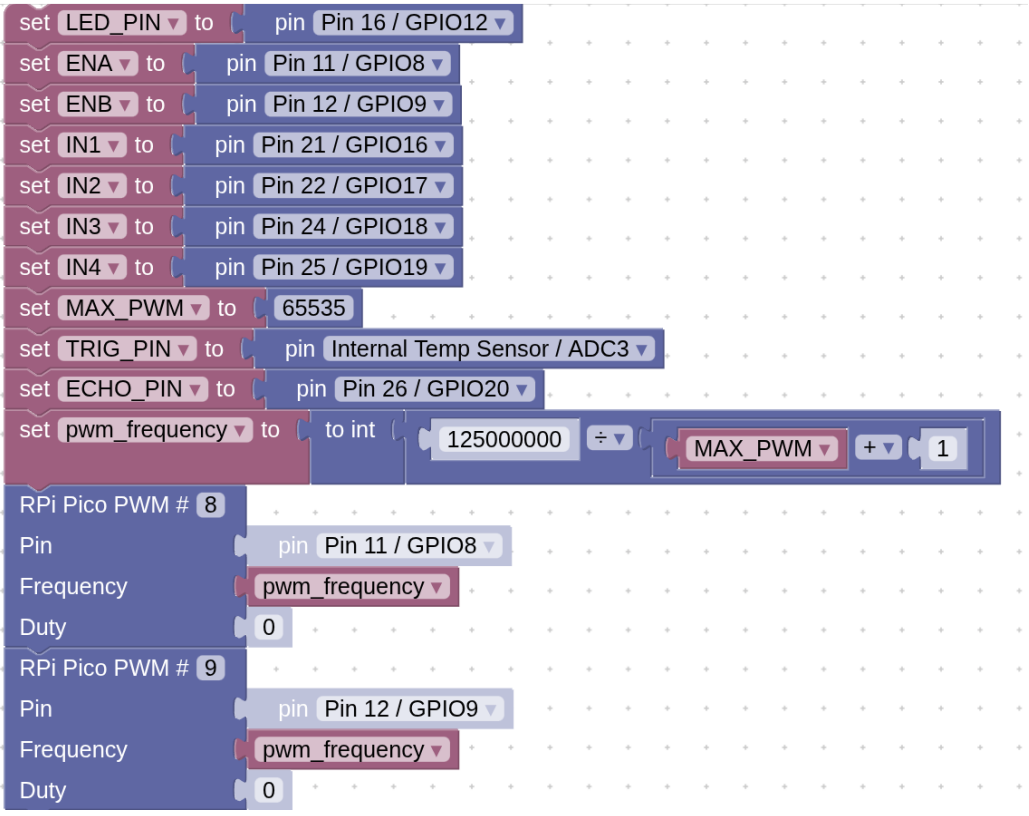
```
21 static mp_obj_t microros_create_node(mp_obj_t name_obj) {
22     mp_obj_t args[1] = { name_obj };
23     return microros_node_make_new(&microros_node_type, 1,
           0, args);
24 }
```

Fonte: Elaborado pelo autor

4.3.2 Criação de Blocos BIPES

Para fornecer uma experiência completa de uma plataforma de aprendizagem, sem necessidade de etapas adicionais de configuração de ambiente, é necessária a criação dos blocos visuais de programação para o BIPES, para isso são necessários 2 arquivos JavaScript, um para a criação da estrutura visual do bloco e outro para a identificação e conversão do bloco para um código equivalente em MicroPython. A Figura 18 mostra um exemplo de um conjunto de blocos agrupados, este bloco mostra a configuração de um conjunto de pinos da Raspberry, além da configuração de um canal PWM.

Figura 18: Exemplo de blocos BIPES.



Fonte: Elaborado pelo Autor

A Listagem 2 mostra um exemplo de criação da estrutura de um bloco no BIPES, este bloco cria a função `microros_init` responsável por inicializar o subsistema micro-ROS na Raspberry. O bloco foi implementado no ambiente *Blockly* (utilizado pelo BIPES) como um bloco de configuração responsável por inicializar o subsistema *micro-ROS* no microcontrolador. Sua definição é composta por:

- (i) um rótulo textual com ícone (`Inicializar micro-ROS`) para facilitar a identificação visual do propósito do bloco;
- (ii) um campo do tipo *dropdown* (`Blockly.FieldDropdown`) que permite ao usuário selecionar o meio de transporte de comunicação (`TRANSPORT`), oferecendo as opções UART (padrão), WiFi e USB; e
- (iii) a configuração de encadeamento do fluxo do programa, definida por `setNextStatement(true, null)`, permitindo que o bloco seja utilizado como uma etapa inicial (*setup*) conectada a outros blocos subsequentes.

Do ponto de vista de usabilidade, o bloco inclui um *tooltip* explicativo (`setTooltip`), detalhando o objetivo da inicialização e resumindo as diferenças entre os transportes disponíveis, reduzindo a necessidade de consulta externa durante a montagem do programa. Adicionalmente, o método `setHelpUrl` aponta para a documentação oficial do *micro-ROS*, oferecendo um caminho de referência para aprofundamento. Por fim, a cor do bloco (`setColour`) segue um padrão visual da categoria *micro-ROS*, facilitando a organização e a leitura do programa em projetos com múltiplos módulos. Os trechos de código referentes à criação de blocos foram adicionados no próprio código-fonte do BIPES, sendo apenas um incremento na documentação oficial do software.

Código-fonte 2: Criação de bloco `microros_init` no BIPES

```
1 Blockly.Blocks['microros_init'] = {
2   init: function() {
3     this.appendDummyInput()
4         .appendField("Inicializar micro-ROS")
5         .appendField(new Blockly.FieldDropdown([
6           ["UART (padr o)", "uart"],
7           ["WiFi", "wifi"],
8           ["USB", "usb"]
```

```
9           ]), "TRANSPORT");
10      this.setNextStatement(true, null);
11      this.setColour(MICROROS_COLOUR_INIT);
12      this.setTooltip("Inicializa o sistema micro-ROS para
            comunica  o com ROS 2.\n\n" +
13                      "UART: Comunica  o serial (padr o)\n"
                            +
14                      "WiFi: Comunica  o via rede sem fio\n"
                            +
15                      "USB: Comunica  o via cabo USB");
16      this.setHelpUrl("https://micro.ros.org/docs/tutorials/
            core/first_application_rtos/");
17    }
18  };
```

Fonte: Elaborado pelo autor

Já a conversão do bloco `microros_init` para código executável é detalhada na Listagem 3. A conversão do bloco é realizada por meio de um gerador definido na linguagem JavaScript, responsável por produzir instruções em MicroPython. O gerador associado ao bloco `microros_init` recupera o valor selecionado no campo `TRANSPORT` e, com base nessa escolha, gera a chamada apropriada da função `microros.init()`.

Além disso, o gerador insere automaticamente a instrução de importação do módulo micro-ROS (`import microros`) utilizando o mecanismo `Blockly.Python.definitions_`, garantindo que a biblioteca necessária esteja disponível no código final, independentemente da ordem ou quantidade de blocos utilizados. Esse mecanismo evita duplicações e assegura a correta inicialização do ambiente de execução.

Código-fonte 3: Gerador Blockly para conversão do bloco de inicialização do micro-ROS em código MicroPython

```
1  Blockly.Python['microros_init'] = function(block) {
2    var transport = block.getFieldValue('TRANSPORT');
3
4    // Adicionar import micro-ROS
```

```
5   Blockly.Python.definitions_['import_microros'] = 'import
       microros';
6
7   var code = '';
8   if (transport === 'uart') {
9     code = 'microros.init()  # Inicializar micro-ROS via
         UART\n';
10  } else {
11    code = 'microros.init("' + transport + '")  #
         Inicializar micro-ROS via ' + transport.toUpperCase
         () + '\n';
12  }
13
14  return code;
15 };
```

Fonte: Elaborado pelo autor.

O código gerado considera a UART como transporte padrão, omitindo o parâmetro quando essa opção é selecionada, enquanto, para os demais transportes (WiFi ou USB), o parâmetro correspondente é explicitamente passado à função de inicialização. Dessa forma, o bloco visual abstrai detalhes de implementação, permitindo que usuários iniciantes configurem a comunicação com o micro-ROS de maneira intuitiva, sem a necessidade de conhecimento prévio da sintaxe da linguagem ou das particularidades da API.

O desenvolvimento da implementação contou com apoio técnico de um colaborador, especialmente nas atividades de codificação e testes do sistema. As decisões de projeto, definição da arquitetura, integração dos módulos e a documentação científica apresentada neste trabalho são de responsabilidade do autor.

Devido ao extenso volume de funções e linhas de código para a implementação dos blocos BIPES e suas conversões para código executável, a totalidade da implementação não está inclusa neste documento. O código-fonte completo encontra-se disponível em um repositório público que pode ser acessado em: https://github.com/ryanguilherme/microros-micropython.

4.3.3 Website para visualização das atividades educacionais

Como forma de complementar a plataforma desenvolvida neste trabalho, foi construído um website com o objetivo de centralizar informações sobre o projeto, disponibilizar documentação técnica e apoiar o uso da plataforma em contextos educacionais. O website atua como um ponto de acesso unificado para os usuários, facilitando o acesso ao conteúdo associado à plataforma.

No contexto da robótica educacional, o website tem como finalidade auxiliar o processo de aprendizagem, oferecendo materiais explicativos, exemplos de uso, tutoriais de configuração e orientações para replicação da plataforma. Dessa forma, o ambiente web contribui para a autonomia dos usuários e para a reutilização do projeto em atividades didáticas no ensino básico e superior.

A construção do website foi realizada utilizando tecnologias web amplamente difundidas, priorizando simplicidade, acessibilidade e facilidade de manutenção. A interface inicial do website é apresentada na Figura 19.

Figura 19: Página inicial do website

Fonte: Elaborado pelo autor

As atividades iniciais foram divididas em módulos, sendo eles módulo iniciante, intermediário e avançado, agregando desde atividades simples como piscar um LED, até o controle PID e navegação autônoma da plataforma.

O módulo básico é voltado a usuários iniciantes e tem como objetivo a introdução aos conceitos fundamentais de controle e interação com a plataforma robótica. As atividades propostas incluem:

- Controle do robô por meio de joystick, com mapeamento de sinais analógicos para acionamento dos motores;
- Implementação de um servidor web embarcado na placa, permitindo o controle remoto do robô por meio de uma interface HTTP;
- Aplicação de mecanismos básicos de segurança utilizando sensores ultrassônicos para detecção de obstáculos e parada automática do sistema.

O módulo intermediário aprofunda os conceitos de comunicação e integração de sistemas, introduzindo o uso de middleware e ferramentas distribuídas. As atividades previstas nesse módulo são:

- Utilização do protocolo MQTT como middleware para comunicação entre o ROS 2, o microcontrolador e sistemas externos;
- Implementação de teleoperação do robô por meio do micro-ROS, permitindo o envio de comandos de movimentação via tópicos ROS utilizando o terminal do dispositivo do usuário.

Já o módulo avançado é destinado a usuários com maior familiaridade com robótica e sistemas de controle, abordando técnicas mais complexas de sensoriamento e controle. As atividades propostas incluem:

- Leitura de encoders e implementação de odometria para estimativa precisa do deslocamento do robô;
- Aplicação de controle PID para ajuste fino da velocidade e do movimento dos motores.

4.4 Pesquisa de satisfação

Para análise do desempenho da plataforma e satisfação dos usuários, foi realizada uma cadeia de testes em uma disciplina de introdução a Robótica (Robótica I) da Universidade Federal do Ceará, com 16 alunos, visando saber a opinião de estudantes com níveis variados de conhecimento em robótica no geral. Para a cadeia de testes, os alunos receberam o protótipo da

plataforma robótica desenvolvida neste trabalho e foram instruídos a apenas acessar o website desenvolvido e realizar duas tarefas, uma básica e uma intermediária, sem auxílio externo, apenas utilizando o ambiente de ensino desenvolvido. As perguntas da pesquisa de satisfação foram as seguintes:

- Qual seu nível de conhecimento prévio sobre ROS (Robot Operating System)? (Opções: Nenhum, Básico, Intermediário, Avançado)
- Como você avalia a facilidade de uso da interface BIPES (programação em blocos) na programação do robô? (Opções: escala de 1, muito difícil até 5, muito fácil)
- A conexão com o robô (Web Portal -> Micro-ROS Agent -> Robô) foi estável durante a prática? (Opções: escala de 1, instável até 5, muito estável)
- A utilização de blocos (No-Code) facilitou o entendimento da lógica de controle (navegação/desvio de obstáculos) em comparação com a codificação tradicional? (Opções: escala de 1, não ajudou até 5, ajudou muito)
- Comparado a outros métodos de ensino de robótica (simuladores ou teoria pura), você sentiu que a interação física com este robô acelerou seu aprendizado? (Opções: escala de 1, irrelevante até 5, acelerou significativamente)
- Você acredita que esta plataforma seria viável para ensino de robótica em escolas com menos recursos? (Opções: Sim, Talvez, Não)

As respostas auxiliaram na validação da viabilidade do uso da plataforma para o ensino de robótica no geral.

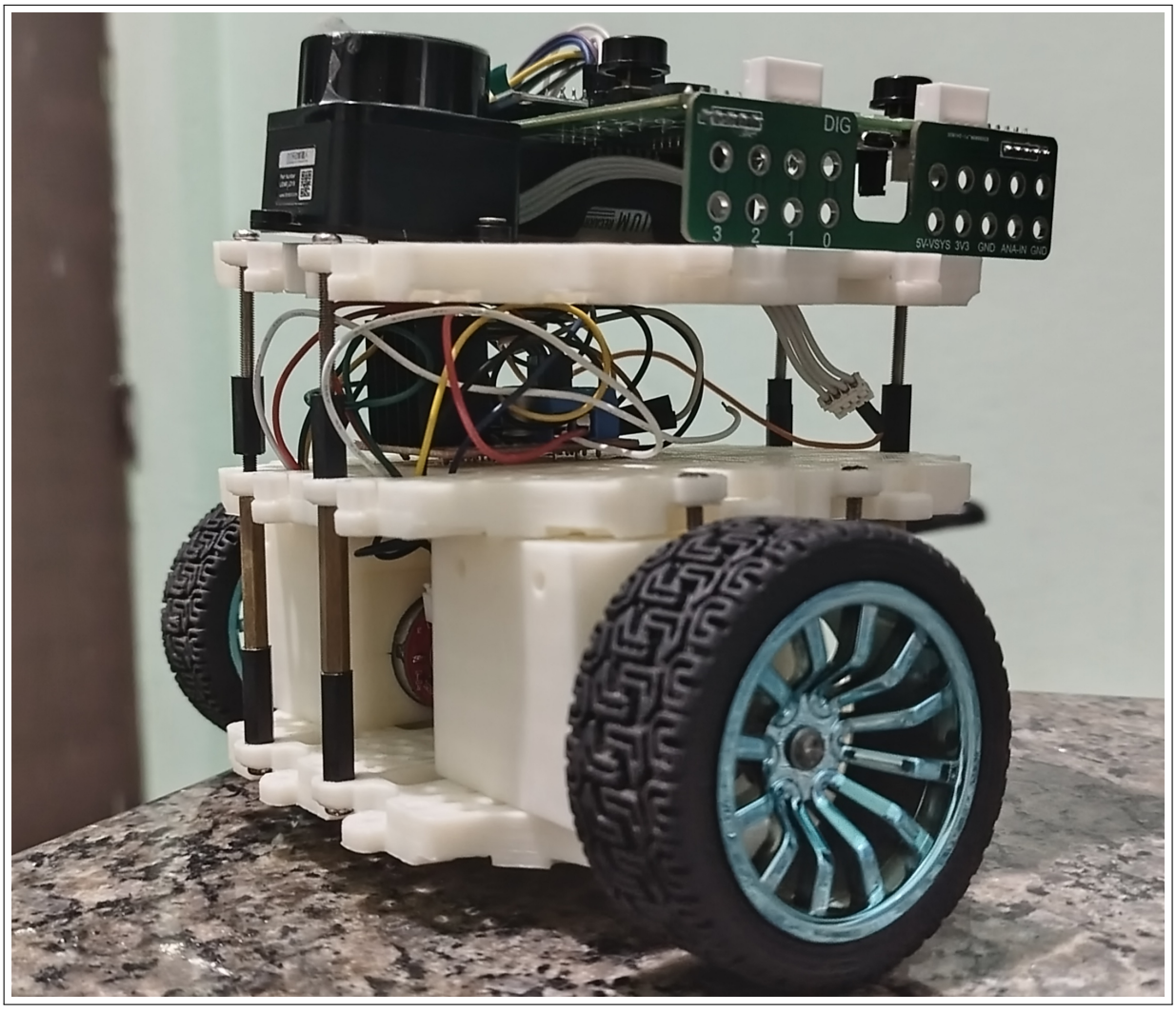
5 RESULTADOS

Neste capítulo são apresentados os resultados obtidos na plataforma física, software, firmware e na pesquisa de satisfação realizada.

5.1 Resultado físico: protótipo construído

Ao final do processo de desenvolvimento, foi obtido um protótipo da plataforma robótica funcional, composta por chassi impresso em 3D, sistema de locomoção e sensores integrados, conforme ilustrado na Figura 20.

Figura 20: Protótipo físico



Fonte: Elaborado pelo autor.

O robô apresenta uma estrutura modular e compacta, característica da versão Burguer do Turtlebot, possuindo compartimentos para diferentes áreas do circuito, sendo um comparti-

mento para os atuadores e bateria, um para a placa de controle elétrico e fios e um para a placa de processamento (BitDogLab com Raspberry Pi Pico W). Com todos os componentes montados e sua versão básica atingida, o protótipo atingiu um custo de fabricação consideravelmente menor que o de outras alternativas de mercado, atingindo um valor médio de 220,00 R$ baseado nos preços de mercado em janeiro de 2026. O Quadro 3 mostra um comparativo entre este trabalho e outras alternativas do mercado. O custo de mercado da plataforma proposta foi inflacionado em 100% para considerar um preço de venda de forma especulativa, pois a tabela considera o preço de venda das outras alternativas, e não o de fabricação pois geralmente não há dados públicos referentes.

Quadro 3: Quadro comparativo com alternativas do mercado

Platform	**Preço (R$)**	**ROS 2**	**No-Code**
TurtleBot 4	7.000 R$	Sim	Não
LEGO Mindstorms	1.200 R$	Não	Sim
Kit de Robô Arduino	450,00 R$	Não	Limitado
Este Trabalho	**440,00 R$**	**Sim**	**Sim**

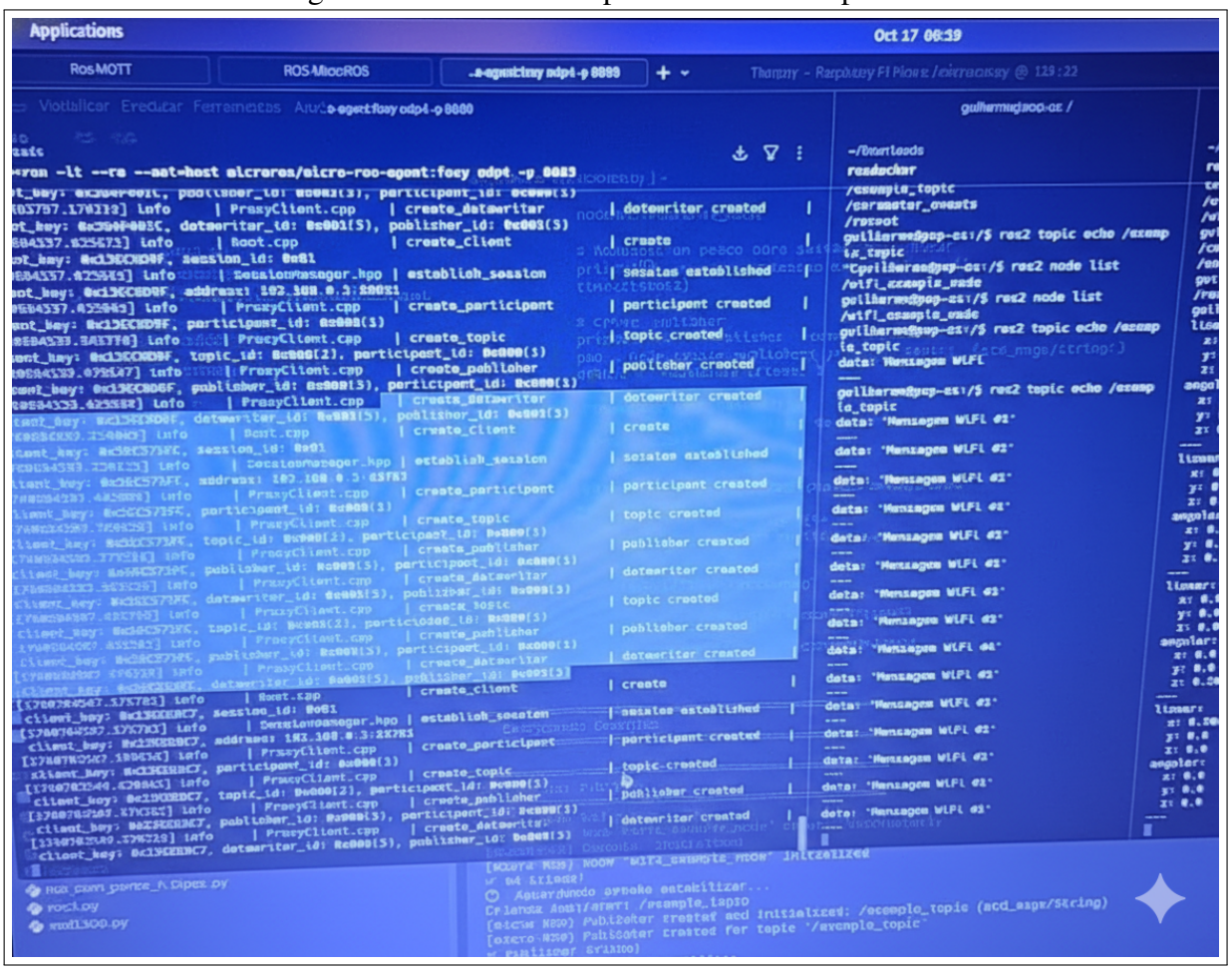
Fonte: Elaborado pelo Autor

5.2 Resultado de integração: hardware + firmware + software

Os testes realizados demonstraram a comunicação bem-sucedida entre o microcontrolador e o ambiente ROS 2 por meio do micro-ROS, permitindo a publicação e subscrição de tópicos em tempo real. A plataforma apresentou comportamento estável durante a teleoperação, com resposta adequada aos comandos enviados via interface web e via tópicos ROS. Já os módulos educacionais desenvolvidos foram organizados em níveis de complexidade crescente, permitindo a introdução gradual de conceitos de robótica, sistemas embarcados e comunicação distribuída.

A Figura 21 mostra os tópicos ROS sendo publicados e lidos em tempo-real, além dos dados dos *encoders* dos atuadores do robô sendo exibidos e alterados por teleoperação.

Figura 21: Processos e tópicos ROS em tempo-real



Fonte: elaborada pelo autor.

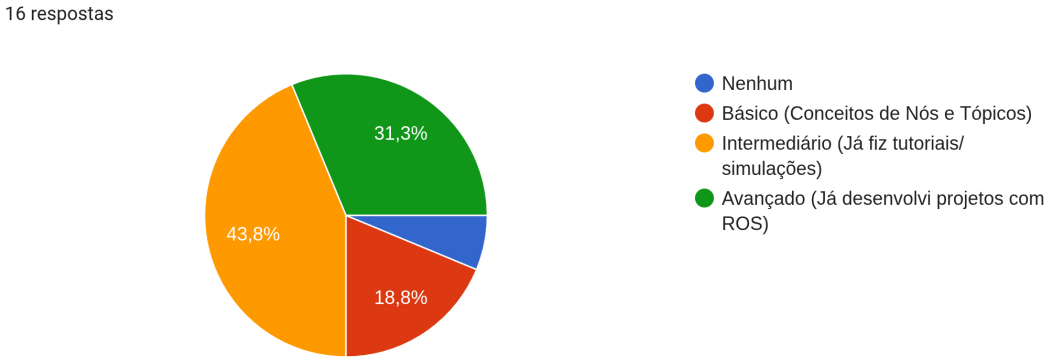
5.3 Resultados da pesquisa de satisfação

Após o uso da plataforma para a realização das práticas de dois módulos, como mencionado em 4.4, os alunos de uma turma de Robótica de ensino superior com 16 estudantes foram submetidos a um formulário de satisfação, onde responderam a um total de 7 perguntas sobre a experiência com a plataforma no contexto de aprendizagem.

Para a primeira pergunta, os resultados foram variados, com a maior parte dos usuários marcando Intermediário ou Avançado como o nível de conhecimento sobre ROS, sendo 6,3% com nenhum conhecimento, 18,8% com um nível básico de conhecimento, 31,3% em nível avançado de conhecimento e 43,8% em nível intermediário. Essa estatística demonstra a variedade do nível de conhecimento técnico sobre ROS dos usuários que foram submetidos a cadeia de testes. Resultados mostrados na Figura 22.

Para a segunda pergunta, foi avaliada a facilidade de uso da interface da plataforma BIPES, um dos principais diferenciais do projeto e seu uso na programação do robô. Cerca de

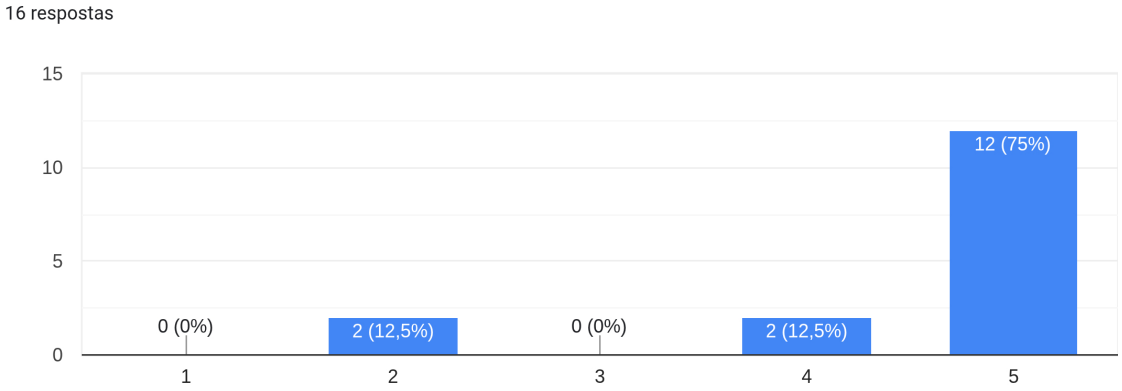
Figura 22: Resultados da pergunta 1



Fonte: Elaborado pelo autor

75% dos usuários marcaram 5, em uma escala de 1-5 do quão fácil foi o uso da plataforma no processo de desenvolvimento. 7,5% marcaram 4 e 2,5% marcaram 1. Resultados mostrados na Figura 23

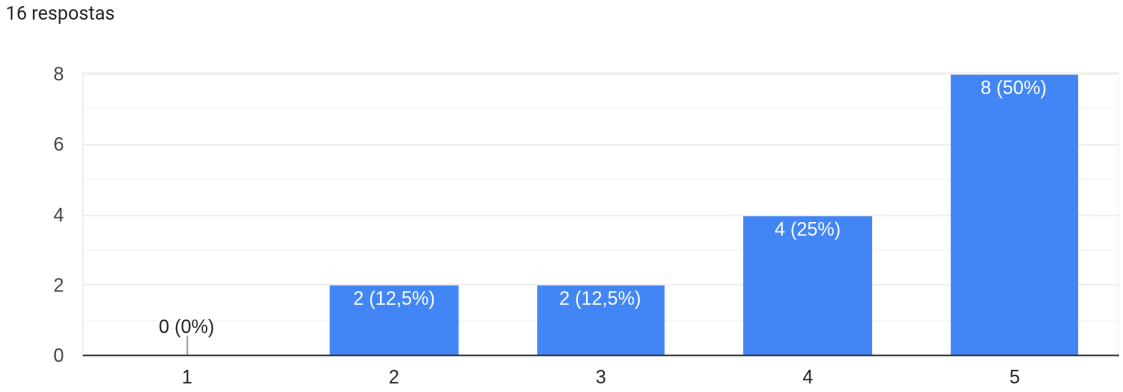
Figura 23: Resultados da pergunta 2



Fonte: Elaborado pelo autor

Para a terceira pergunta, foi avaliada a experiência do usuário perante a estabilidade de conexão dentre todo o fluxo de desenvolvimento da plataforma, desde o uso do portal web até a criação do agente micro-ROS e o upload do firmware no robô. Cerca de 50% dos usuários marcaram 5, em uma escala de 1-5 do quão estável foi a conexão durante o fluxo de desenvolvimento. Cerca de 25% marcaram 4 enquanto os demais 25% marcaram 3 ou menos. Resultados mostrados na Figura 24.

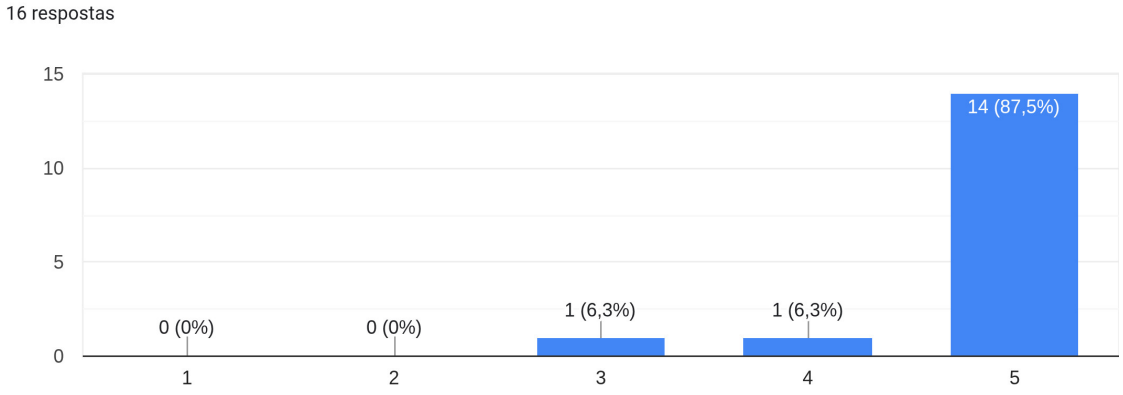
Figura 24: Resultados da pergunta 3



Fonte: Elaborado pelo autor

Já a quarta pergunta buscou medir o quão facilitada foi o entendimento da lógica de controle para a tomada de decisões do robô utilizando os blocos BIPES. Cerca de 87,5% dos usuários marcaram 5, em uma escala de 1-5 do quão facilitado foi a modelagem do sistema utilizando os blocos, 6,3% marcaram 4 e 6,3% marcaram 3. Essa estatística mostra o quão eficiente é o uso da programação visual baseada em blocos para a modelagem de sistemas baseados em tomada de decisão, como um sistema de navegação autônoma do robô ou teleoperação, presentes em atividades do módulo 2 da plataforma utilizados na cadeia de testes. Resultados mostrados na Figura 25.

Figura 25: Resultados da pergunta 4

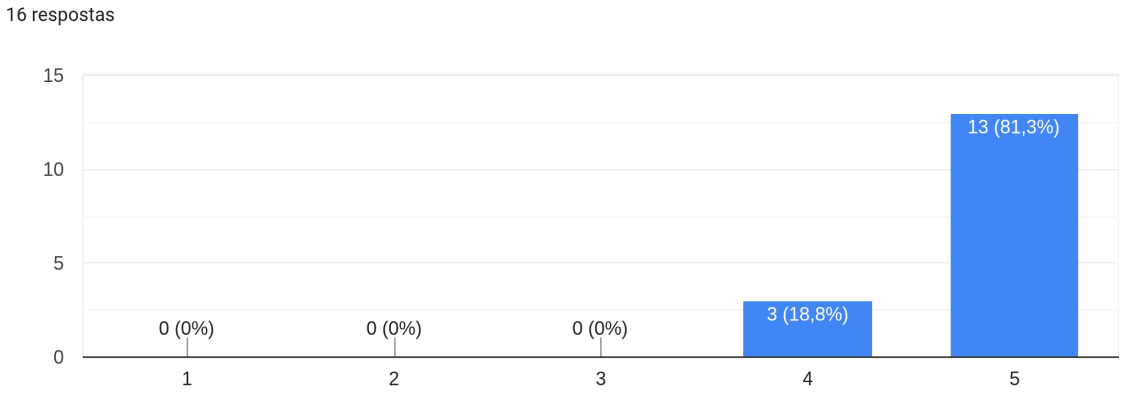


Fonte: Elaborado pelo autor

Na quinta pergunta, foi medido o nível de satisfação dos usuários com o uso da plataforma desenvolvida neste trabalho em comparação com outras alternativas como simuladores ou somente teoria. Cerca de 81,3% dos participantes marcaram 5, em uma escala de 1-5 do quão a plataforma acelerou o aprendizado, os demais 18,8% marcaram 4. Resultados mostrados na

Figura 26

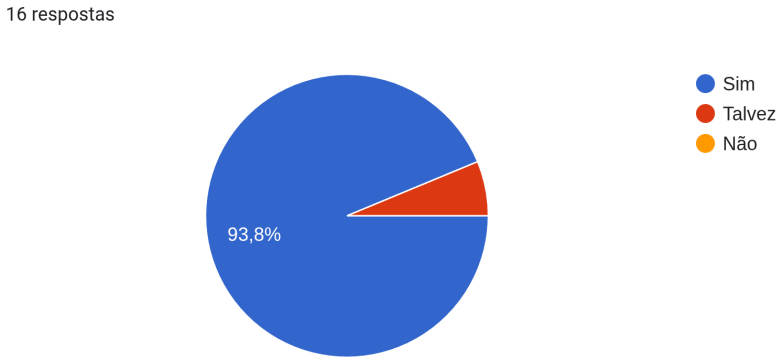
Figura 26: Resultados da pergunta 5



Fonte: Elaborado pelo autor

Já a sexta e última pergunta teve como objetivo capturar um feedback geral dos participantes em relação ao uso da plataforma no contexto de robótica educacional, capturando as opiniões sobre se consideram a plataforma viável, inviável ou talvez viável no uso para o ensino de robótica em escolas com poucos recursos. Cerca de 93,8% marcaram Sim, enquanto os demais 6,3% marcaram Talvez. Resultados mostrados na Figura 27.

Figura 27: Resultados da pergunta 6

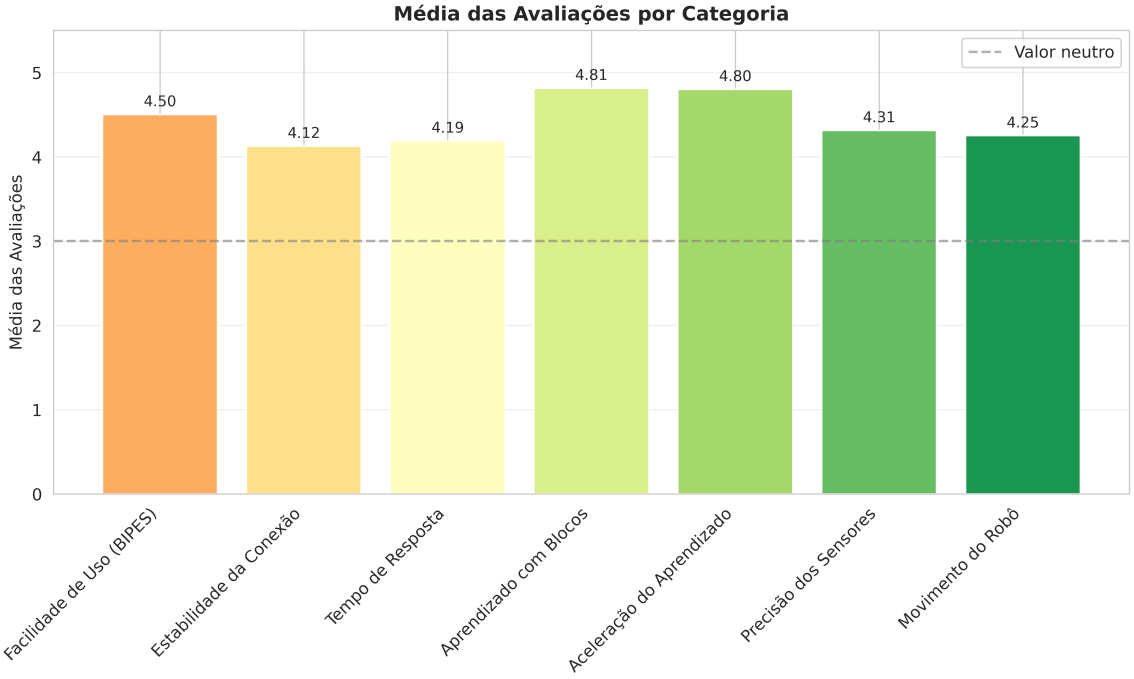


Fonte: Elaborado pelo autor

Com o objetivo de realizar uma análise categorizada das avaliações dos usuários, as questões do questionário foram organizadas em diferentes categorias, a saber: facilidade de uso, estabilidade da conexão, tempo de resposta, aprendizado com blocos, aceleração do aprendizado, precisão dos sensores e movimento do robô. As médias obtidas para cada categoria foram, respectivamente, 4,50, 4,12, 4,19, 4,81, 4,80, 4,31 e 4,25, considerando uma escala de 1 a 5, na qual o valor 5 representa a melhor avaliação.

Destacam-se as categorias *Aprendizado com Blocos* e *Aceleração do Aprendizado*, que apresentaram as maiores pontuações médias. Esses resultados são coerentes com o contexto educacional da plataforma e indicam que os objetivos propostos pelo trabalho, especialmente no que se refere ao apoio ao processo de ensino e aprendizagem, foram satisfatoriamente alcançados. A Figura 28 mostra a distribuição das avaliações por categoria.

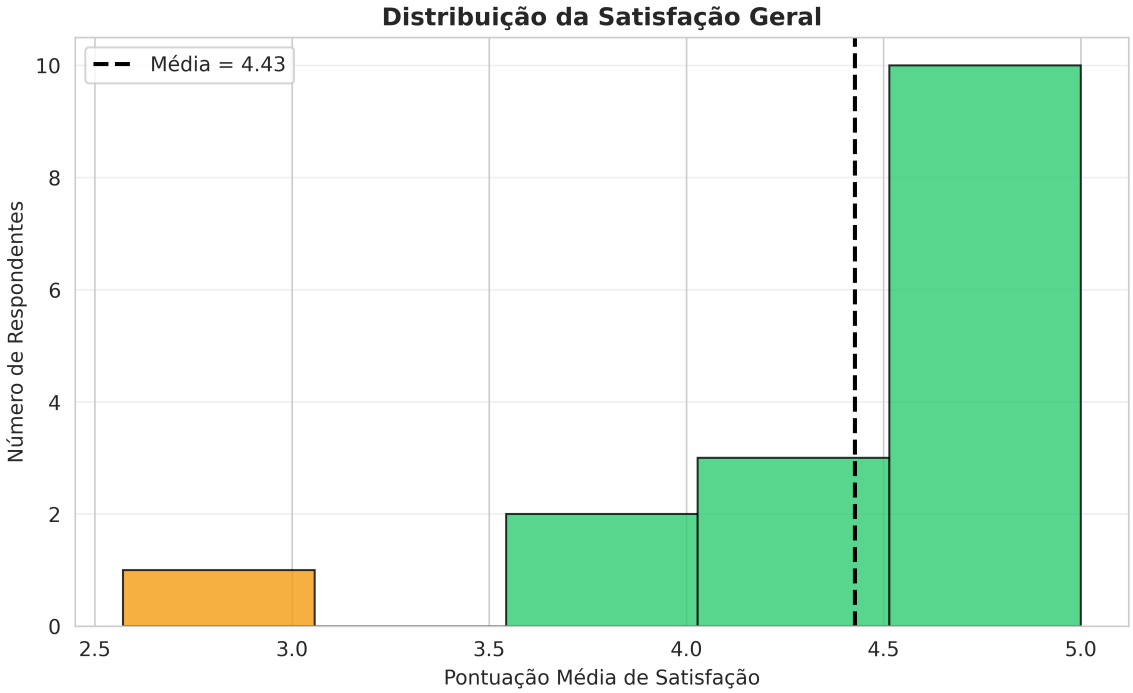
Figura 28: Média categorizada das avaliações



Fonte: Elaborado pelo autor

Por fim, a Figura 29 ilustra a distribuição do nível de satisfação dos usuários em relação à plataforma, considerando uma escala de 1 a 5, na qual 5 representa o maior grau de satisfação. Observa-se que a plataforma alcançou uma média de satisfação de 4,43, indicando uma avaliação predominantemente positiva por parte dos participantes e sugerindo boa aceitação da proposta desenvolvida.

Figura 29: Satisfação geral com a plataforma



Fonte: Elaborado pelo autor

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou o desenvolvimento de uma plataforma robótica multissensorial de baixo custo, voltada ao apoio do ensino de robótica no contexto da educação básica e superior. A proposta teve como principal objetivo aliar acessibilidade, modularidade e integração com ferramentas modernas de robótica, de modo a reduzir barreiras técnicas e financeiras frequentemente associadas ao uso de plataformas robóticas educacionais.

Ao longo do desenvolvimento, foi possível projetar e construir uma plataforma física funcional, composta por chassi impresso em 3D, sistema de locomoção, sensores e atuadores, bem como uma arquitetura de software baseada em ROS 2 e micro-ROS. A utilização da Raspberry Pi Pico W como microcontrolador principal mostrou-se adequada ao contexto do projeto, permitindo a execução de aplicações embarcadas com comunicação distribuída, mantendo baixo consumo energético e custo reduzido.

Já em relação ao software, a integração entre micro-ROS e a plataforma BIPES é um diferencial, possibilitando a programação do robô por meio de blocos visuais. Essa abordagem contribui para a democratização do acesso à robótica, permitindo que estudantes com diferentes níveis de conhecimento em programação possam interagir com conceitos avançados, como comunicação entre nós, publicação e subscrição de tópicos e controle de dispositivos robóticos de forma intuitiva.

Os resultados obtidos a partir da avaliação com usuários indicaram uma boa aceitação da plataforma, especialmente nos aspectos relacionados ao aprendizado com blocos e à aceleração do processo de aprendizagem. Esses dados reforçam o potencial da plataforma como ferramenta de apoio pedagógico, alinhada a metodologias ativas e à aprendizagem baseada em projetos, favorecendo o desenvolvimento de competências técnicas e cognitivas relevantes.

Dessa forma, conclui-se que os objetivos propostos neste trabalho foram alcançados, resultando em uma plataforma robótica educacional funcional, acessível e extensível, capaz de servir como base para atividades didáticas em diferentes níveis de ensino e para futuras pesquisas na área de robótica educacional.

6.1 Trabalhos Futuros

Apesar dos resultados alcançados, este trabalho pode ser estendido ainda mais. Uma possibilidade consiste na evolução dos módulos educacionais, com a inclusão de novos conteúdos

voltados a temas como inteligência artificial aplicada à robótica. A realização de estudos de caso em ambientes educacionais reais, com acompanhamento longitudinal do impacto da plataforma no processo de aprendizagem, também se apresenta como uma extensão relevante do trabalho.

Adicionalmente, a otimização da biblioteca desenvolvida para integração entre micro-ROS e MicroPython pode contribuir para melhorar o desempenho, a estabilidade da comunicação e a portabilidade para outros microcontroladores compatíveis. A disponibilização da plataforma como um projeto aberto, com documentação ampliada e materiais didáticos complementares, favorece sua adoção por instituições de ensino e pela comunidade acadêmica.

Por fim, a integração da plataforma com ambientes de simulação e ferramentas de avaliação automática de atividades educacionais representa uma oportunidade para tornar o sistema ainda mais robusto e alinhado às demandas contemporâneas do ensino de robótica.

REFERÊNCIAS

ALMEIDA, M. E. B. de; VALENTE, J. A. **Tecnologia e Currículo**: Trajetórias convergentes ou divergentes? Campinas: Papirus Editora, 2012. ISBN 9788530808634.

ALMEIDA, R.; GONçALVES, J.; FONSECA, H.; MAGALHãES, A.; SOUZA, S. Bipes: Block based integrated platform for embedded systems. **IEEE Latin America Transactions**, IEEE, v. 19, n. 6, p. 1003–1010, 2021.

ARAUJO, D. R. **Robótica Educacional como Recurso Pedagógico para Ensino de Matemática e Computação na Educação Básica**. Dissertação (Trabalho de Conclusão de Curso (Graduação)) – Instituto Federal de Educação, Ciência e Tecnologia Sul-rio-grandense, Pelotas, 2023.

ATTARAN, M. The rise of 3-d printing: The advantages of additive manufacturing over traditional manufacturing. **Business Horizons**, v. 60, n. 5, p. 677–688, 2017.

AZEVEDO, S.; AGLAé, A.; PITTA, R. Introdução à robótica educacional. In: **62ª Reunião Anual da SBPC**. Natal: Sociedade Brasileira para o Progresso da Ciência, 2010.

BATISTA, L. F. W. **Implementação, Controle e Monitoração de uma Plataforma Móvel Utilizando Linux Embarcado**. Dissertação (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2013.

BENITTI, F. B. V. Exploring the educational potential of robotics in schools: A systematic review. **Computers & Education**, v. 58, n. 3, p. 978–988, 2012.

BIPES Project. **BIPES**: Block-based integrated platform for embedded systems. 2024. Disponível em: http://www.bipes.net.br/pt/. Acesso em: 17 jan. 2026.

BitDogLab. **BitDogLab**: Uma jornada educativa com eletrônica, embarcados e ia. 2023. Disponível em: https://embarcados.com.br/bitdoglab-uma-jornada-educativa-com-eletronica-embarcados-e-ia/. Acesso em: 17 jan. 2026.

BLIKSTEIN, P. **Digital Fabrication and 'Making' in Education**: The democratization of invention. Bielefeld: Transcript Publishers, 2013.

DESIGN for Fused Filament Fabrication Additive Manufacturing. 2015. Disponível em: https://www.researchgate.net/. Acesso em: 17 jan. 2026.

GIBSON, I.; ROSEN, D.; STUCKER, B. **Additive Manufacturing Technologies**. [*S. l.*]: Springer, 2015.

GOMES, T.; MENDES, C. Teaching embedded systems using low-cost educational platforms. **IEEE Revista Iberoamericana de Tecnologias del Aprendizaje**, v. 14, n. 3, p. 120–127, 2019.

International Federation of Robotics. World robotics 2022 report. **IFR Statistical Department**, International Federation of Robotics, 2022.

ISO/ASTM. **ISO/ASTM 52900:2021 — Additive manufacturing**: General principles: Fundamentals and vocabulary. 2021. Norma técnica internacional.

KOUBAA, A. *et al.* Micro-ros: Enabling ros 2 for microcontrollers. In: **IEEE International Conference on Robotics and Automation (ICRA)**. [*S. l.*: *s. n.*], 2020.

MakerHero. **Driver Motor Ponte H L298N**. 2024. Disponível em: https://www.makerhero.com/produto/driver-motor-ponte-h-l298n/. Acesso em: 17 jan. 2026.

Maker's Pet. **How-to**: Connect xiaomi $15 lds02rr lidar to esp32, arduino. 2024. Disponível em: https://makerspet.com/blog/how-to-connect-xiaomi-lds02rr-lidar-to-esp32/. Acesso em: 17 jan. 2026.

Maxtronics. **NAO6 Humanoid Robot**. 2024. Disponível em: https://maxtronics.com/en/nao6/. Acesso em: 17 jan. 2026.

MUBIN, O.; STEVENS, C. J.; SHAHID, S.; MAHMUD, A. A.; DONG, J.-J. A review of the applicability of robots in education. **Technology for Education and Learning**, v. 1, n. 1, p. 1–7, 2013.

NAGLA, K.; UDDIN, M.; SINGH, D. Multisensor data fusion and integration for mobile robots: A review. **International Journal of Robotics and Automation**, Institute of Advanced Engineering and Science, v. 3, n. 2, p. 131–138, 2014.

OLIVEIRA, B. Q. d. *et al.* Tipos e aplicações de sensores na robótica. **Cadernos de Graduação - Ciências Exatas e Tecnológicas (UNIT)**, Aracaju, v. 4, n. 2, p. 151–160, 2017.

Onshape Community. **Modelos 3D de chassi robótico para impressão 3D**. 2024. Disponível em: https://cad.onshape.com/documents/2586c4659ef3e7078e91168b/w/14abf4cb615429a14a2732cc/e/8767d6ebf200d3a76df782bc. Acesso em: 15 dez. 2025.

Open Robotics. **TurtleBot**: An open platform for robotics education. 2023. Disponível em: https://www.turtlebot.com. Acesso em: 17 jan. 2026.

Open Robotics. **ROS.org**: Powering the world's robots. s.d. Disponível em: http://www.ros.org. Acesso em: 10 jun. 2025.

PAPERT, S. **Mindstorms**: Children, computers, and powerful ideas. New York: Basic Books, 1980.

PIAGET, J. **A epistemologia genética**. Petrópolis: Vozes, 1976.

QUIGLEY, M. *et al.* Ros: an open-source robot operating system. **ICRA Workshop on Open Source Software**, 2009.

Raspberry Pi Foundation. **About Us**. s.d. Disponível em: https://www.raspberrypi.org/about/. Acesso em: 10 jun. 2025.

Raspberry Pi Ltd. **Raspberry Pi Pico W Datasheet**. 2023. Disponível em: https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf. Acesso em: 17 jan. 2026.

REPS, M. **All about the Xiaomi LiDAR Scanner and the SunFounder RasPad**. 2018. Canal do YouTube. Disponível em: https://www.youtube.com/watch?v=4sQCz75BfrM. Acesso em: 17 jan. 2026.

ROBOTICS, A. **NAO6**. Paris, França, 2024. Disponível em: https://aldebaran.com/en/nao6/. Acesso em: 9 jul. 2025.

ROBOTICS, C. P. **Turtlebot 4**. Kitchener, Canadá, 2024. Disponível em: https://clearpathrobotics.com/turtlebot-4/. Acesso em: 23 jul. 2025.

ROBOTICS, O. **Turtlebot**. 2024. Disponível em: https://www.turtlebot.com/. Acesso em: 23 jul. 2025.

ROMERO, R. A. F. **Robótica Móvel**. São Paulo: LTC, 2008.

SICILIANO, B.; SCIAVICCO, L.; VILLANI, L.; ORIOLO, G. **Robotics**: Modelling, planning and control. London: Springer, 2010.

SIEGWART, R.; NOURBAKHSH, I. R. **Introduction to Autonomous Mobile Robots**. Cambridge, MA: MIT Press, 2004.

APÊNDICE A – CÓDIGO-FONTE DA BIBLIOTECA DE MICRO-ROS PARA MICROPYTHON

Código-fonte 4: Biblioteca micro-ROS

```
1  // MicroPython micro-ROS module
2  // Provides Python interface to micro-ROS functionality
3
4  #include "microros.h"
5  #include "py/runtime.h"
6  #include "py/mperrno.h"
7  #include "py/mphal.h"
8  #include <string.h>
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 // Node Class Implementation
13
14 // Node constructor
15 static mp_obj_t microros_node_make_new(const mp_obj_type_t *type,
       size_t n_args,
16                                        size_t n_kw, const mp_obj_t *
       args) {
17     mp_arg_check_num(n_args, n_kw, 1, 1, false);
18
19     // Create new node object
20     microros_node_obj_t *self = mp_obj_malloc(microros_node_obj_t, type
       );
21
22     // Store node name
23     self->name = args[0];
24
25     // Initialize micro-ROS node
```

```
26      const char *name = mp_obj_str_get_str(args[0]);
27      if (microros_create_node_impl(name) == 0) {
28          mp_printf(&mp_plat_print, "[micro-ROS] Node '%s' initialized\n"
    , name);
29      } else {
30          mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to
    initialize node"));
31      }
32
33      return MP_OBJ_FROM_PTR(self);
34  }
35
36  // Node.create_publisher(topic, msg_type)
37  static mp_obj_t microros_node_create_publisher(mp_obj_t self_in,
    mp_obj_t topic_obj, mp_obj_t msg_type_obj) {
38      // Create publisher object
39      microros_publisher_obj_t *pub = mp_obj_malloc(
    microros_publisher_obj_t, &microros_publisher_type);
40
41      const char *topic = mp_obj_str_get_str(topic_obj);
42      const char *msg_type = mp_obj_str_get_str(msg_type_obj);
43
44      // Store topic and message type
45      pub->topic = topic;
46      pub->msg_type = msg_type;
47
48      // Initialize publisher with micro-ROS
49      if (microros_create_publisher_impl(&pub->handle, topic, msg_type)
    == 0) {
50          mp_printf(&mp_plat_print, "[micro-ROS] Publisher created for
    topic '%s'\n", topic);
51      } else {
```

```
52          mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to
    create publisher"));
53      }
54
55      return MP_OBJ_FROM_PTR(pub);
56  }
57
58  // Node.create_subscription(topic, msg_type, callback)
59  static mp_obj_t microros_node_create_subscription(size_t n_args, const
    mp_obj_t *args) {
60      if (n_args != 4) {
61          mp_raise_TypeError(MP_ERROR_TEXT("create_subscription requires
    3 arguments"));
62      }
63
64      // Create subscription object
65      microros_subscription_obj_t *sub = mp_obj_malloc(
    microros_subscription_obj_t, &microros_subscription_type);
66
67      const char *topic = mp_obj_str_get_str(args[1]);
68      const char *msg_type = mp_obj_str_get_str(args[2]);
69
70      // Store topic, message type and callback
71      sub->topic = topic;
72      sub->msg_type = msg_type;
73      sub->callback = args[3];
74
75      // Initialize subscription with micro-ROS
76      if (microros_create_subscription_impl(&sub->handle, topic, msg_type
    ) == 0) {
77          // Register the subscription with callback
```

```
78          if (microros_register_subscription(topic, msg_type, sub->
    callback, sub->handle) == 0) {
79              mp_printf(&mp_plat_print, "[micro-ROS] Subscription created
     for topic '%s'\n", topic);
80          } else {
81              mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed
    to register subscription callback"));
82          }
83      } else {
84          mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to
    create subscription"));
85      }
86
87      return MP_OBJ_FROM_PTR(sub);
88  }
89
90  // Node.spin_once()
91  static mp_obj_t microros_node_spin_once(mp_obj_t self_in) {
92      microros_spin_once_impl();
93      return mp_const_none;
94  }
95
96  // Node.spin()
97  static mp_obj_t microros_node_spin(mp_obj_t self_in) {
98      microros_spin_impl();
99      return mp_const_none;
100 }
101
102 // Node function objects
103 static MP_DEFINE_CONST_FUN_OBJ_3(microros_node_create_publisher_obj,
    microros_node_create_publisher);
```

```
104 static MP_DEFINE_CONST_FUN_OBJ_VAR_BETWEEN(
        microros_node_create_subscription_obj, 4, 4,
        microros_node_create_subscription);
105 static MP_DEFINE_CONST_FUN_OBJ_1(microros_node_spin_once_obj,
        microros_node_spin_once);
106 static MP_DEFINE_CONST_FUN_OBJ_1(microros_node_spin_obj,
        microros_node_spin);
107
108 // Node method table
109 static const mp_rom_map_elem_t microros_node_locals_table[] = {
110     { MP_ROM_QSTR(MP_QSTR_create_publisher), MP_ROM_PTR(&
        microros_node_create_publisher_obj) },
111     { MP_ROM_QSTR(MP_QSTR_create_subscription), MP_ROM_PTR(&
        microros_node_create_subscription_obj) },
112     { MP_ROM_QSTR(MP_QSTR_spin_once), MP_ROM_PTR(&
        microros_node_spin_once_obj) },
113     { MP_ROM_QSTR(MP_QSTR_spin), MP_ROM_PTR(&microros_node_spin_obj) },
114 };
115 static MP_DEFINE_CONST_DICT(microros_node_locals_dict,
        microros_node_locals_table);
116
117 // Node type definition
118 MP_DEFINE_CONST_OBJ_TYPE(
119     microros_node_type,
120     MP_QSTR_Node,
121     MP_TYPE_FLAG_NONE,
122     make_new, microros_node_make_new,
123     locals_dict, &microros_node_locals_dict
124 );
125
126 // Publisher Class Implementation
127
```

```
128 // Publisher.publish(message, msg_type=None)
129 static mp_obj_t microros_publisher_publish(size_t n_args, const
    mp_obj_t *args) {
130     microros_publisher_obj_t *self = MP_OBJ_TO_PTR(args[0]);
131     mp_obj_t message = args[1];
132 
133     // Use provided msg_type or default to publisher's msg_type
134     const char *msg_type = self->msg_type;
135     if (n_args > 2) {
136         msg_type = mp_obj_str_get_str(args[2]);
137     }
138 
139     if (strcmp(msg_type, "std_msgs/String") == 0) {
140         const char *str_data = mp_obj_str_get_str(message);
141         if (microros_publish_string(self->handle, str_data) != 0) {
142             mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed
    to publish String message"));
143         }
144     }
145     else if (strcmp(msg_type, "std_msgs/Int32") == 0) {
146         int32_t int_data = mp_obj_get_int(message);
147         if (microros_publish_int32(self->handle, int_data) != 0) {
148             mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed
    to publish Int32 message"));
149         }
150     }
151     else if (strcmp(msg_type, "std_msgs/Float64") == 0) {
152         double float_data = mp_obj_get_float(message);
153         if (microros_publish_float64(self->handle, float_data) != 0) {
154             mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed
    to publish Float64 message"));
155         }
```

```
156     }
157     else if (strcmp(msg_type, "geometry_msgs/Twist") == 0) {
158         // Expect a dict with 'linear' and 'angular' keys
159         if (!mp_obj_is_type(message, &mp_type_dict)) {
160             mp_raise_TypeError(MP_ERROR_TEXT("Twist message must be a
    dict with 'linear' and 'angular' keys"));
161         }
162 
163         mp_obj_dict_t *msg_dict = MP_OBJ_TO_PTR(message);
164         mp_obj_t linear_obj = mp_obj_dict_get(message, mp_obj_new_str("
    linear", 6));
165         mp_obj_t angular_obj = mp_obj_dict_get(message, mp_obj_new_str(
    "angular", 7));
166 
167         if (linear_obj == MP_OBJ_NULL || angular_obj == MP_OBJ_NULL) {
168             mp_raise_ValueError(MP_ERROR_TEXT("Twist message must have
    'linear' and 'angular' keys"));
169         }
170 
171         // Extract linear values (expect dict with x, y, z)
172         mp_obj_t linear_x = mp_obj_dict_get(linear_obj, mp_obj_new_str(
    "x", 1));
173         mp_obj_t linear_y = mp_obj_dict_get(linear_obj, mp_obj_new_str(
    "y", 1));
174         mp_obj_t linear_z = mp_obj_dict_get(linear_obj, mp_obj_new_str(
    "z", 1));
175 
176         // Extract angular values
177         mp_obj_t angular_x = mp_obj_dict_get(angular_obj,
    mp_obj_new_str("x", 1));
178         mp_obj_t angular_y = mp_obj_dict_get(angular_obj,
    mp_obj_new_str("y", 1));
```

```
179         mp_obj_t angular_z = mp_obj_dict_get(angular_obj,
    mp_obj_new_str("z", 1));
180
181         if (linear_x == MP_OBJ_NULL || linear_y == MP_OBJ_NULL ||
    linear_z == MP_OBJ_NULL ||
182             angular_x == MP_OBJ_NULL || angular_y == MP_OBJ_NULL ||
    angular_z == MP_OBJ_NULL) {
183             mp_raise_ValueError(MP_ERROR_TEXT("Linear and angular must
    have x, y, z components"));
184         }
185
186         double lx = mp_obj_get_float(linear_x);
187         double ly = mp_obj_get_float(linear_y);
188         double lz = mp_obj_get_float(linear_z);
189         double ax = mp_obj_get_float(angular_x);
190         double ay = mp_obj_get_float(angular_y);
191         double az = mp_obj_get_float(angular_z);
192
193         if (microros_publish_twist(self->handle, lx, ly, lz, ax, ay, az
    ) != 0) {
194             mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed
    to publish Twist message"));
195         }
196     }
197     else if (strcmp(msg_type, "geometry_msgs/Point") == 0) {
198         // Expect a dict with 'x', 'y', 'z' keys
199         if (!mp_obj_is_type(message, &mp_type_dict)) {
200             mp_raise_TypeError(MP_ERROR_TEXT("Point message must be a
    dict with x, y, z keys"));
201         }
202
```

```
203         mp_obj_t x_obj = mp_obj_dict_get(message, mp_obj_new_str("x",
    1));
204         mp_obj_t y_obj = mp_obj_dict_get(message, mp_obj_new_str("y",
    1));
205         mp_obj_t z_obj = mp_obj_dict_get(message, mp_obj_new_str("z",
    1));
206
207         if (x_obj == MP_OBJ_NULL || y_obj == MP_OBJ_NULL || z_obj ==
    MP_OBJ_NULL) {
208             mp_raise_ValueError(MP_ERROR_TEXT("Point message must have
    x, y, z keys"));
209         }
210
211         double x = mp_obj_get_float(x_obj);
212         double y = mp_obj_get_float(y_obj);
213         double z = mp_obj_get_float(z_obj);
214
215         if (microros_publish_point(self->handle, x, y, z) != 0) {
216             mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed
    to publish Point message"));
217         }
218     }
219     else if (strcmp(msg_type, "geometry_msgs/Pose") == 0) {
220         // Expect a dict with 'position' and 'orientation' keys
221         if (!mp_obj_is_type(message, &mp_type_dict)) {
222             mp_raise_TypeError(MP_ERROR_TEXT("Pose message must be a
    dict with 'position' and 'orientation' keys"));
223         }
224
225         mp_obj_t position_obj = mp_obj_dict_get(message, mp_obj_new_str
    ("position", 8));
```

```
226         mp_obj_t orientation_obj = mp_obj_dict_get(message,
    mp_obj_new_str("orientation", 11));
227
228         if (position_obj == MP_OBJ_NULL || orientation_obj ==
    MP_OBJ_NULL) {
229             mp_raise_ValueError(MP_ERROR_TEXT("Pose message must have '
    position' and 'orientation' keys"));
230         }
231
232         // Extract position values (expect dict with x, y, z)
233         mp_obj_t pos_x = mp_obj_dict_get(position_obj, mp_obj_new_str("
    x", 1));
234         mp_obj_t pos_y = mp_obj_dict_get(position_obj, mp_obj_new_str("
    y", 1));
235         mp_obj_t pos_z = mp_obj_dict_get(position_obj, mp_obj_new_str("
    z", 1));
236
237         // Extract orientation values (expect dict with x, y, z, w)
238         mp_obj_t orient_x = mp_obj_dict_get(orientation_obj,
    mp_obj_new_str("x", 1));
239         mp_obj_t orient_y = mp_obj_dict_get(orientation_obj,
    mp_obj_new_str("y", 1));
240         mp_obj_t orient_z = mp_obj_dict_get(orientation_obj,
    mp_obj_new_str("z", 1));
241         mp_obj_t orient_w = mp_obj_dict_get(orientation_obj,
    mp_obj_new_str("w", 1));
242
243         if (pos_x == MP_OBJ_NULL || pos_y == MP_OBJ_NULL || pos_z ==
    MP_OBJ_NULL ||
244             orient_x == MP_OBJ_NULL || orient_y == MP_OBJ_NULL ||
    orient_z == MP_OBJ_NULL || orient_w == MP_OBJ_NULL) {
```

```
245             mp_raise_ValueError(MP_ERROR_TEXT("Position must have x,y,z and orientation must have x,y,z,w components"));
246         }
247 
248         double px = mp_obj_get_float(pos_x);
249         double py = mp_obj_get_float(pos_y);
250         double pz = mp_obj_get_float(pos_z);
251         double ox = mp_obj_get_float(orient_x);
252         double oy = mp_obj_get_float(orient_y);
253         double oz = mp_obj_get_float(orient_z);
254         double ow = mp_obj_get_float(orient_w);
255 
256         if (microros_publish_pose(self->handle, px, py, pz, ox, oy, oz, ow) != 0) {
257             mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to publish Pose message"));
258         }
259     }
260     else {
261         // Fallback para tipos desconhecidos - tenta converter para string
262         const char *str_data = mp_obj_str_get_str(message);
263         if (microros_publish_message(self->topic, str_data) != 0) {
264             mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to publish message"));
265         }
266     }
267 
268     return mp_const_true;  // Return True on success
269 }
270 
271 // Publisher function objects
```

```
272 static MP_DEFINE_CONST_FUN_OBJ_VAR_BETWEEN(
        microros_publisher_publish_obj, 2, 3, microros_publisher_publish);
273
274 // Publisher method table
275 static const mp_rom_map_elem_t microros_publisher_locals_table[] = {
276     { MP_ROM_QSTR(MP_QSTR_publish), MP_ROM_PTR(&
        microros_publisher_publish_obj) },
277 };
278 static MP_DEFINE_CONST_DICT(microros_publisher_locals_dict,
        microros_publisher_locals_table);
279
280 // Publisher type definition
281 MP_DEFINE_CONST_OBJ_TYPE(
282     microros_publisher_type,
283     MP_QSTR_Publisher,
284     MP_TYPE_FLAG_NONE,
285     locals_dict, &microros_publisher_locals_dict
286 );
287
288 // Subscription Class Implementation
289
290 // Subscription type definition (simple - no additional methods for now
        )
291 MP_DEFINE_CONST_OBJ_TYPE(
292     microros_subscription_type,
293     MP_QSTR_Subscription,
294     MP_TYPE_FLAG_NONE
295 );
296
297 // Module Functions
298
299 // microros.set_transport(transport_type)
```

```
300 static mp_obj_t microros_set_transport(mp_obj_t transport_obj) {
301     const char *transport = mp_obj_str_get_str(transport_obj);
302 
303     extern int microros_set_transport_impl(const char *transport);
304     if (microros_set_transport_impl(transport) != 0) {
305         mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to
    set transport type"));
306     }
307 
308     mp_printf(&mp_plat_print, "[micro-ROS] Transport set to: %s\n",
    transport);
309     return mp_const_none;
310 }
311 
312 // microros.set_agent_ip(ip_address)
313 static mp_obj_t microros_set_agent_ip(mp_obj_t ip_obj) {
314     const char *ip = mp_obj_str_get_str(ip_obj);
315 
316     extern int microros_set_agent_ip_impl(const char *ip);
317     if (microros_set_agent_ip_impl(ip) != 0) {
318         mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to
    set agent IP"));
319     }
320 
321     mp_printf(&mp_plat_print, "[micro-ROS] Agent IP set to: %s\n", ip);
322     return mp_const_none;
323 }
324 
325 // microros.set_agent_port(port)
326 static mp_obj_t microros_set_agent_port(mp_obj_t port_obj) {
327     uint16_t port = mp_obj_get_int(port_obj);
328 
```

```
329     extern int microros_set_agent_port_impl(uint16_t port);
330     if (microros_set_agent_port_impl(port) != 0) {
331         mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to
    set agent port"));
332     }
333 
334     mp_printf(&mp_plat_print, "[micro-ROS] Agent port set to: %d\n",
    port);
335     return mp_const_none;
336 }
337 
338 // microros.ping_agent(timeout_ms, attempts)
339 static mp_obj_t microros_ping_agent(size_t n_args, const mp_obj_t *args
    ) {
340     int timeout_ms = 1000;  // default 1 second
341     int attempts = 3;        // default 3 attempts
342 
343     if (n_args > 0) {
344         timeout_ms = mp_obj_get_int(args[0]);
345     }
346     if (n_args > 1) {
347         attempts = mp_obj_get_int(args[1]);
348     }
349 
350     extern int microros_ping_agent_impl(int timeout_ms, int attempts);
351     int result = microros_ping_agent_impl(timeout_ms, attempts);
352 
353     return mp_obj_new_bool(result == 0);
354 }
355 
356 // microros.init(transport) - transport is optional
357 static mp_obj_t microros_init(size_t n_args, const mp_obj_t *args) {
```

```
358     const char *transport = "uart";  // default to UART
359     if (n_args > 0) {
360         transport = mp_obj_str_get_str(args[0]);
361     }
362 
363     // Set transport type
364     extern int microros_set_transport_impl(const char *transport);
365     if (microros_set_transport_impl(transport) != 0) {
366         mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to
    set transport type"));
367     }
368 
369     // Initialize transport layer
370     if (microros_init_transport_layer() != 0) {
371         mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("Failed to
    initialize micro-ROS transport"));
372     }
373 
374     mp_printf(&mp_plat_print, "[micro-ROS] Transport '%s' initialized\n
    ", transport);
375     return mp_const_none;
376 }
377 
378 // microros.Node(name)
379 static mp_obj_t microros_create_node(mp_obj_t name_obj) {
380     mp_obj_t args[1] = { name_obj };
381     return microros_node_make_new(&microros_node_type, 1, 0, args);
382 }
383 
384 // Message Creation Helper Functions
385 
```

```
386 // microros.create_twist(linear_x, linear_y, linear_z, angular_x,
    angular_y, angular_z)
387 static mp_obj_t microros_create_twist(size_t n_args, const mp_obj_t *
    args) {
388     if (n_args != 6) {
389         mp_raise_TypeError(MP_ERROR_TEXT("create_twist requires 6
    arguments: linear_x, linear_y, linear_z, angular_x, angular_y,
    angular_z"));
390     }
391
392     // Create nested dictionary structure
393     mp_obj_t twist_dict = mp_obj_new_dict(2);
394     mp_obj_t linear_dict = mp_obj_new_dict(3);
395     mp_obj_t angular_dict = mp_obj_new_dict(3);
396
397     // Set linear values
398     mp_obj_dict_store(linear_dict, mp_obj_new_str("x", 1), args[0]);
399     mp_obj_dict_store(linear_dict, mp_obj_new_str("y", 1), args[1]);
400     mp_obj_dict_store(linear_dict, mp_obj_new_str("z", 1), args[2]);
401
402     // Set angular values
403     mp_obj_dict_store(angular_dict, mp_obj_new_str("x", 1), args[3]);
404     mp_obj_dict_store(angular_dict, mp_obj_new_str("y", 1), args[4]);
405     mp_obj_dict_store(angular_dict, mp_obj_new_str("z", 1), args[5]);
406
407     // Set top-level keys
408     mp_obj_dict_store(twist_dict, mp_obj_new_str("linear", 6),
    linear_dict);
409     mp_obj_dict_store(twist_dict, mp_obj_new_str("angular", 7),
    angular_dict);
410
411     return twist_dict;
```

```
412 }
413
414 // microros.create_point(x, y, z)
415 static mp_obj_t microros_create_point(mp_obj_t x_obj, mp_obj_t y_obj,
        mp_obj_t z_obj) {
416     mp_obj_t point_dict = mp_obj_new_dict(3);
417
418     mp_obj_dict_store(point_dict, mp_obj_new_str("x", 1), x_obj);
419     mp_obj_dict_store(point_dict, mp_obj_new_str("y", 1), y_obj);
420     mp_obj_dict_store(point_dict, mp_obj_new_str("z", 1), z_obj);
421
422     return point_dict;
423 }
424
425 // microros.create_pose(x, y, z, qx, qy, qz, qw)
426 static mp_obj_t microros_create_pose(size_t n_args, const mp_obj_t *
        args) {
427     if (n_args != 7) {
428         mp_raise_TypeError(MP_ERROR_TEXT("create_pose requires 7
        arguments: x, y, z, qx, qy, qz, qw"));
429     }
430
431     mp_obj_t pose_dict = mp_obj_new_dict(2);
432     mp_obj_t position_dict = mp_obj_new_dict(3);
433     mp_obj_t orientation_dict = mp_obj_new_dict(4);
434
435     // Set position
436     mp_obj_dict_store(position_dict, mp_obj_new_str("x", 1), args[0]);
437     mp_obj_dict_store(position_dict, mp_obj_new_str("y", 1), args[1]);
438     mp_obj_dict_store(position_dict, mp_obj_new_str("z", 1), args[2]);
439
440     // Set orientation (quaternion)
```

```
441     mp_obj_dict_store(orientation_dict, mp_obj_new_str("x", 1), args
    [3]);
442     mp_obj_dict_store(orientation_dict, mp_obj_new_str("y", 1), args
    [4]);
443     mp_obj_dict_store(orientation_dict, mp_obj_new_str("z", 1), args
    [5]);
444     mp_obj_dict_store(orientation_dict, mp_obj_new_str("w", 1), args
    [6]);
445 
446     // Set top-level keys
447     mp_obj_dict_store(pose_dict, mp_obj_new_str("position", 8),
    position_dict);
448     mp_obj_dict_store(pose_dict, mp_obj_new_str("orientation", 11),
    orientation_dict);
449 
450     return pose_dict;
451 }
452 
453 // microros.simulate_message(topic, data) - Para testar subscriptions
454 static mp_obj_t microros_simulate_message_func(mp_obj_t topic_obj,
    mp_obj_t data_obj) {
455     const char *topic = mp_obj_str_get_str(topic_obj);
456     const char *data = mp_obj_str_get_str(data_obj);
457 
458     if (microros_simulate_message(topic, data) != 0) {
459         mp_raise_msg(&mp_type_RuntimeError, MP_ERROR_TEXT("No
    subscription found for topic"));
460     }
461 
462     return mp_const_none;
463 }
464 
```

```
465 // Module function objects
466 static MP_DEFINE_CONST_FUN_OBJ_VAR_BETWEEN(microros_init_obj, 0, 1,
       microros_init);
467 static MP_DEFINE_CONST_FUN_OBJ_1(microros_create_node_obj,
       microros_create_node);
468 static MP_DEFINE_CONST_FUN_OBJ_1(microros_set_transport_obj,
       microros_set_transport);
469 static MP_DEFINE_CONST_FUN_OBJ_1(microros_set_agent_ip_obj,
       microros_set_agent_ip);
470 static MP_DEFINE_CONST_FUN_OBJ_1(microros_set_agent_port_obj,
       microros_set_agent_port);
471 static MP_DEFINE_CONST_FUN_OBJ_VAR_BETWEEN(microros_ping_agent_obj, 0,
       2, microros_ping_agent);
472 static MP_DEFINE_CONST_FUN_OBJ_VAR_BETWEEN(microros_create_twist_obj,
       6, 6, microros_create_twist);
473 static MP_DEFINE_CONST_FUN_OBJ_3(microros_create_point_obj,
       microros_create_point);
474 static MP_DEFINE_CONST_FUN_OBJ_VAR_BETWEEN(microros_create_pose_obj, 7,
        7, microros_create_pose);
475 static MP_DEFINE_CONST_FUN_OBJ_2(microros_simulate_message_obj,
       microros_simulate_message_func);
476
477 // Module globals table
478 static const mp_rom_map_elem_t microros_module_globals_table[] = {
479     { MP_ROM_QSTR(MP_QSTR___name__), MP_ROM_QSTR(MP_QSTR_microros) },
480     { MP_ROM_QSTR(MP_QSTR_init), MP_ROM_PTR(&microros_init_obj) },
481     { MP_ROM_QSTR(MP_QSTR_Node), MP_ROM_PTR(&microros_create_node_obj)
       },
482     { MP_ROM_QSTR(MP_QSTR_set_transport), MP_ROM_PTR(&
       microros_set_transport_obj) },
483     { MP_ROM_QSTR(MP_QSTR_set_agent_ip), MP_ROM_PTR(&
       microros_set_agent_ip_obj) },
```

```
484     { MP_ROM_QSTR(MP_QSTR_set_agent_port), MP_ROM_PTR(&
        microros_set_agent_port_obj) },
485     { MP_ROM_QSTR(MP_QSTR_ping_agent), MP_ROM_PTR(&
        microros_ping_agent_obj) },
486     { MP_ROM_QSTR(MP_QSTR_create_twist), MP_ROM_PTR(&
        microros_create_twist_obj) },
487     { MP_ROM_QSTR(MP_QSTR_create_point), MP_ROM_PTR(&
        microros_create_point_obj) },
488     { MP_ROM_QSTR(MP_QSTR_create_pose), MP_ROM_PTR(&
        microros_create_pose_obj) },
489     { MP_ROM_QSTR(MP_QSTR_simulate_message), MP_ROM_PTR(&
        microros_simulate_message_obj) },
490 };
491 static MP_DEFINE_CONST_DICT(microros_module_globals,
        microros_module_globals_table);
492
493 // Module definition
494 const mp_obj_module_t microros_module = {
495     .base = { &mp_type_module },
496     .globals = (mp_obj_dict_t *)&microros_module_globals,
497 };
498
499 // Register module
500 MP_REGISTER_MODULE(MP_QSTR_microros, microros_module);
```

Código presente em: https://github.com/ryanguilherme/microros-micropython