UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA
BACHARELADO EM ENGENHARIA MECÂNICA

EMERSON GUILHERME LOURENÇO DE MELO

DESENVOLVIMENTO DE UMA PLATAFORMA EDUCACIONAL DE ROBÓTICA MÓVEL

Recife
2023

EMERSON GUILHERME LOURENÇO DE MELO

DESENVOLVIMENTO DE UMA PLATAFORMA EDUCACIONAL DE ROBÓTICA MÓVEL

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Mecânica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Engenharia Mecânica

Orientador: Pr. Dr. Adrien Durand-Petiteville

Recife

2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Melo, Emerson Guilherme Lourenço de .
Desenvolvimento de uma plataforma educacional de robótica móvel / Emerson Guilherme Lourenço de Melo. - Recife, 2023.
136p : il., tab.

Orientador(a): Adrien Durand Petiteville
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Mecânica - Bacharelado, 2023.
Inclui referências, apêndices.

1. ROS. 2. robótica. 3. robôs móveis. 4. impressão 3D. 5. CAD. I. Petiteville, Adrien Durand. (Orientação). II. Título.

620 CDD (22.ed.)

EMERSON GUILHERME LOURENÇO DE MELO

DESENVOLVIMENTO DE UMA PLATAFORMA EDUCACIONAL DE ROBÓTICA MÓVEL

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Mecânica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Engenharia Mecânica

Aprovado em: 22/12/2023

BANCA EXAMINADORA

_____

Pr. Dr. Adrien Durand-Petiteville

Universidade Federal de Pernambuco

_____

Prof. Dr. Justo Emilio Alvarez Jacobo

Universidade Federal de Pernambuco

_____

Prof. Dr. João Paulo Cerquinho Cajueiro

Universidade Federal de Pernambuco

RESUMO

O presente trabalho propôs o desenvolvimento de uma plataforma educacional de robótica móvel, denominada Obelix, com o intuito de proporcionar uma alternativa acessível e eficaz para o ensino de robótica. A motivação para este projeto surge da constatação de que, mesmo diante do avanço tecnológico, a barreira de custo ainda impede o pleno acesso a plataformas educacionais de qualidade. Com base nesse cenário, os objetivos foram delineados para criar uma plataforma modular, robusta e de fácil montagem, utilizando uma abordagem de baixo custo.

O trabalho teve início com o desenvolvimento de um modelo 3D da plataforma no software educacional, servindo como estrutura base para o robô. A implementação dos *drivers* em *Python*, utilizando o *framework* ROS, possibilitou o acesso aos *hardwares* da plataforma. Um software adicional em *Python*, também utilizando o ROS, foi desenvolvido para enfrentar um desafio de movimentação autônoma, tornando a plataforma uma ferramenta prática e desafiadora para aprendizado e experimentação.

Como resultado foram construídos quatro modelos da plataforma, os quais foram incorporados à disciplina de programação robótica na Universidade Federal de Pernambuco (UFPE). A versão final da plataforma atendeu aos requisitos iniciais do projeto, destacando-se por sua modularidade, robustez, facilidade de montagem e acesso aos componentes. Modelos 3D e códigos desenvolvidos foram disponibilizados no GitHub para replicação e aprimoramento do projeto.

Palavras-chave: ROS; robótica; robôs móveis; impressão 3D; CAD.

ABSTRACT

The present work proposed the development of an educational mobile robotics platform, named Obelix, aiming to provide an affordable and effective alternative for robotics education. The motivation for this project arises from the realization that, even in the face of technological advancements, cost barriers still hinder full access to quality educational platforms. Based on this scenario, objectives were outlined to create a modular, robust, and easily assembled platform, using a low-cost approach.

The work began with the development of a 3D model of the platform in educational software, serving as the base structure for the robot. The implementation of drivers in Python, using the ROS framework, enabled access to the platform's hardware. Additional software in Python, also utilizing ROS, was developed to address a challenge of autonomous movement, making the platform a practical and challenging tool for learning and experimentation.

As a result, four models of the platform were built and incorporated into the robotics programming course at the Federal University of Pernambuco (UFPE). The final version of the platform met the initial project requirements, standing out for its modularity, robustness, easy assembly, and access to components. 3D models and developed codes were made available on GitHub for replication and improvement of the project.

Keywords: ROS; Robotics; Mobile Robots; 3D Printing; CAD

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Robôs são máquinas que possuem capacidade de executar automaticamente uma série de ações, interpretando e raciocinando sobre uma tarefa e sua execução (SICILIANO; KHATIB, 2016). Um robô deve ser capaz de decidir as suas ações, sem o controle direto de humanos. Para ser capaz de realizar suas atividades, ele precisa conhecer o ambiente no seu entorno, através de sensores e interagir com ele, através de atuadores. Os robôs móveis são uma categoria de robôs que possuem a capacidade de se locomover, podendo ser aéreos, aquáticos, subaquáticos ou terrestres (JAHN et al. 2020). A utilização de robôs móveis permite o aumento do grau de autonomia e flexibilidade de sistemas automatizados, facilitando a integração e o controle.

A robótica encontra-se cada vez mais presente na sociedade. O que antes era usado apenas em indústrias e centros altamente tecnológicos hoje pode ser encontrado em shoppings, parques e residências mais simples. Essa popularidade ocasiona em um aumento do incentivo em pesquisas na área da robótica com pesquisadores e empresas desenvolvendo formas de facilitar a vida das pessoas em suas atividades do dia a dia e para substituir o ser humano em trabalhos repetitivos ou de alto grau de periculosidade. Segundo o banco de dados do *Institute of Electrical and Electronic Engineers* IEEE (2020*)* houve um aumento na quantidade de resultados de pesquisa pelo termo *robotics* e *mobile robotics*. Em 2009 o volume era de cerca de 12500 resultados e em 2019 chegaram perto de 26000. Esse dado evidencia que o interesse nessa área vem crescendo consideravelmente e manterá essa tendência.

O avanço e a popularização da robótica evoluíram muitos setores da sociedade que agora se beneficiam das novas tecnologias. Apesar disso, o campo educacional enfrenta dificuldades em se adaptar a esse cenário, criando uma desconexão entre as experiências vivenciadas no ambiente escolar e as habilidades necessárias para prosperar em um mundo cada vez mais digital. À medida que a tecnologia molda as formas de comunicação, trabalho e acesso à informação, é crucial que a educação abrace essas mudanças para preparar o aluno adequadamente.

Nesse contexto, a robótica educacional é uma ótima abordagem para incluir a tecnologia como uma ferramenta de aprendizado. Ela consiste em um ambiente em

que o aluno é apresentado à montagem de sistemas, automação e controle de dispositivos mecânicos e eletrônicos que podem ser controlados pelo computador (S. P.; VICTOR, 2020). Os trabalhos desenvolvidos por (DANIELA; MILTIADIS, 2019) com o objetivo de avaliar as implicações de seu uso com crianças concluíram que a robótica educacional é uma ótima ferramenta para construção de conhecimento e facilita o processo de aprendizado para os estudantes.

Entretanto, a robótica é um campo multidisciplinar e para trabalhar com ela é necessário o domínio de muitas áreas, como mecânica, controle, computação, eletrônica e a combinação destas. Por isso, é importante ter conhecimento de ferramentas que possam auxiliar na construção de um robô tornando a tarefa mais simples. Dentre as áreas citadas a computação possui uma ferramenta muito interessante que auxilia na escrita do código, o *Robot Operating System* (ROS) (QUIGLEY et al. 2009). Escrever um *software* para um robô pode ser complexo principalmente um robô autônomo que utiliza uma variedade de hardwares, além disso, dado que a amplitude de conhecimento necessária está muito além das capacidades de uma única pessoa, as arquiteturas de *software* de robótica também devem apoiar esforços de integração de código. Com o foco na resolução desses problemas, surge o ROS, um sistema operacional de robôs de código aberto amplamente utilizado na comunidade robótica. O ROS fornece uma estrutura flexível e modular para o desenvolvimento de software de robôs, facilitando a integração de componentes e a implementação de algoritmos complexos.

Além dos desafios de cunho intelectual, no âmbito do Brasil persistem os desafios estruturais. Visto que nem todas as escolas e universidades têm condições de ter um laboratório estruturado com os equipamentos e plataformas robóticas necessárias para que um professor realize um trabalho adequado com os seus alunos dificultando a aplicação da robótica educacional. Esse desafio será detalhado e exemplificado no próximo tópico.

1.1 APRESENTAÇÃO DO PROBLEMA

O principal desafio para a implantação da robótica educacional no cenário brasileiro é financeiro como apontado no trabalho de Moreira Maciel, Araújo Leal (2022). Pois é necessário um ambiente preparado com um conjunto de equipamentos que possibilitem a realização de atividades práticas alinhadas com o

os conteúdos teóricos. Um professor com acesso aos equipamentos certos aplicando uma didática direcionada pode apresentar na prática o método científico, pesquisa, testes e análise dos resultados através da utilização da robótica.

No mercado existem plataformas robóticas que possuem tudo que é necessário para o uso educacional, entre as mais populares estão o *Duckiebot*, *TurtleBot* 3, *TurtleBot* 4 e o robô Tiago. Nos tópicos a seguir serão detalhadas as informações sobre cada plataforma, enfatizando as suas principais características e ao final, será apresentada a Tabela 1 que resume os principais componentes das plataformas.

1.1.1 *Duckiebot*

Duckiebot é um robô móvel autônomo desenvolvido principalmente para fins educacionais e de pesquisa em robótica. O projeto *Duckiebot* foi iniciado no Laboratório de Robótica do *Massachusetts Institute of Technology* (MIT) e se concentra em ensinar conceitos de robótica, visão computacional e aprendizado de máquina de forma acessível e prática. O *Duckiebot* é projetado para operar em ambientes internos e é equipado com sensores como câmeras e outros componentes que permitem a navegação autônoma. O projeto *Duckiebot* utiliza o *software open source* para desenvolvimento de robôs. Como resultado, o *Duckiebot* é amplamente utilizado em universidades e instituições de pesquisa como uma ferramenta prática e educacional para a exploração de conceitos de robótica e sistemas autônomos no geral.

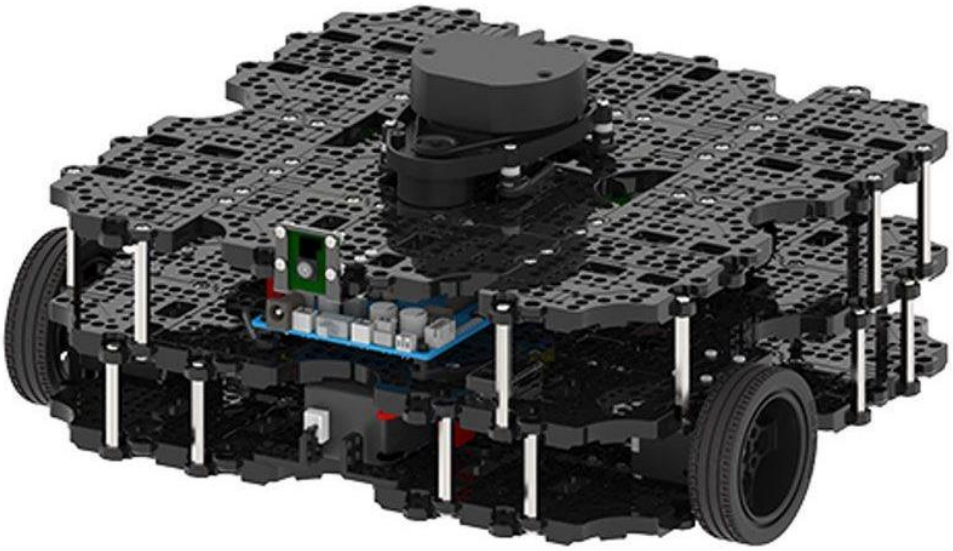
Figura 1 – Robô *Duckiebot*.

Fonte: nvidia (2023).

1.1.2 *TurtleBot* 3

O *TurtleBot* é um robô móvel bastante conhecido no mercado por sua modularidade, capacidade de customização, custo relativamente baixo e compatibilidade com *frameworks* de *software* robótico, tornando-o uma opção interessante para robótica educacional. Equipado com componentes destinados a diversas aplicações, o robô incorpora um lidar 360º para mapeamento de ambientes e prevenção de obstáculos, a placa de controle OpenCR, utilizada como interface de baixo nível para os atuadores, um sensor *Inertial Measurement Unit* (IMU) e um *Raspberry Pi* encarregado da programação em níveis superiores.

Figura 2 – Robô turtleboat 3 waffle



Fonte: Turtlebot(2023)

1.1.3 *TurtleBot* 4

O *TurtleBot* 4 é uma versão avançada do *TurtleBot* 3, oferecendo mais potência, sensores aprimorados e uma experiência de usuário de qualidade. Equipado com uma base móvel *iRobot® Create3, Raspberry Pi 4* e sensores como câmera estéreo OAK-D e LiDAR 2D, proporciona uma plataforma pronta para desenvolvimento em robótica. Os sensores integrados incluem câmera, LiDAR, IMU

e outros, permitindo customização para diversas aplicações. Sendo open source, seus códigos e modelos estão disponíveis para a comunidade contribuir e aprimorar.

Figura 3 – Robô *turtlebot* 4.

Fonte: *Turtlebot*(2023)

1.1.4 TIAGo

O TIAGo, desenvolvido pela *PAL Robotics*, é um robô de assistência móvel projetado para operar em ambientes internos. Destacando-se por sua mobilidade autônoma em espaços fechados, possui rodas omnidirecionais ou pernas, dependendo do modelo. Com um tronco extensível e braço manipulador, o TIAGo é adequado para tarefas de percepção, manipulação e navegação. Equipado com sensores, como câmeras e sensores de proximidade, o robô é capaz de interagir com o ambiente. Sua compatibilidade com *softwares* de código aberto facilita a integração em pesquisas acadêmicas e contextos de robótica.

Figura 4 – Robô TIAGo.

Fonte: wiki.ros.org (2023).

Tabela 1 – Componentes das plataformas

Características	**Duckiebot**	**Turtle bot 3**	**Turtle bot 4**	**Tiago**
Câmera	160 graus fov	Raspberry Pi câmera	OAK-D-PRO	RGB-D camera
Computador	NVIDIA Jetson Nano 4GB	Raspberry Pi 3	Raspberry Pi 4B (4 GB)	CPU intel i5/i7 RAM 8/16 GB SSD 250/500GB
Lidar	Não apresenta	LDS-01	RPLIDAR-A1	Lidar 2D
Liberdade de câmera	Não apresenta	Não apresenta	Não apresenta	Apresenta
Locomoção diferencial	Apresenta	Apresenta	Apresenta	Depende do modelo
Preço	R$ 2.182,18	R$ 8.151,43	R$ 10.328,14	R$ 259.584,79-R$486.272,58

Fonte: O autor (2023).

Apesar de serem ótimas opções e atenderem aos requisitos para trabalhar com robótica, os robôs apresentados na Tabela 1 enfrentam um problema em comum, custo elevado somado às taxas de importação, tornando-os pouco acessíveis financeiramente, especialmente considerando a realidade socioeconômica brasileira. A falta de alternativas acessíveis cria uma barreira para a implementação da robótica nas escolas nacionais, limitando o acesso dos alunos a essa valiosa ferramenta de aprendizado.

1.2 OBJETIVOS

O avanço da tecnologia facilita o acesso ao conhecimento e diminui o custo tecnologias principalmente em centros urbanos. Mesmo que não se tenha o equipamento necessário para fabricar um componente é possível comprar o produto final. Sendo assim, com conhecimento certo e acesso a alguns equipamentos é possível construir uma plataforma robótica própria mais barata que as comerciais e que atende os requisitos de um curso de robótica.

O objetivo deste trabalho consiste em desenvolver uma plataforma educacional de robótica móvel, que seja não apenas acessível em termos de custo de construção, mas também capaz de enfrentar uma variedade de desafios. Esta plataforma será implementada como uma ferramenta prática de aprendizado e experimentação em ambientes acadêmicos, especificamente nas salas de aula e para atividades de pesquisa na Universidade Federal de Pernambuco (UFPE). Os atuadores, sensores e centrais de controle serão criteriosamente escolhidos para aproximar essa plataforma das situações reais enfrentadas por robôs comerciais, tornando-a uma ferramenta valiosa para a formação em engenharia.

Todos os modelos CAD e códigos desenvolvidos serão disponibilizados no github (Plataforma_robótica), permitindo a replicação e aprimoramento do projeto por outros interessados. Os objetivos específicos que norteiam este trabalho incluem:

- Desenvolvimento de um modelo tridimensional (3D) em um software CAD educacional, que será utilizado como estrutura do robô;
- Desenvolver *drivers* na linguagem de programação *Python* utilizando o *framework* ROS para acessar os *hardwares* utilizados na construção da plataforma;
- Desenvolver um Software na linguagem de programação *Python* utilizando o *framework* ROS, capaz de superar o desafio de movimentação autônoma;
- Disponibilizar a documentação e as informações necessárias para que seja possível replicar e melhorar a plataforma;
- Colaboração com o Professor Dr. Adrien Durand-Petiteville para a criação de material didático que explore as potencialidades da plataforma robótica desenvolvida, visando enriquecer o aprendizado em contextos acadêmicos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ROBÔS MÓVEIS

Um robô móvel é um dispositivo mecânico montado sobre uma base não fixa, que age sob o controle de um sistema computacional, equipado com sensores e atuadores que o permitem interagir com o ambiente. Nesse aspecto um robô deve ser capaz de perceber o ambiente, processar estas informações e atuar sobre o meio. A utilização de robôs móveis permite o aumento do grau de autonomia e flexibilidade de sistemas automatizados, facilitando a integração e o controle. Eles podem contribuir para uma melhora significativa da produtividade e eliminar a presença humana em ambientes potencialmente agressivos e perigosos para a saúde como em indústrias de mineração, química, nuclear e militar.

A classificação de um robô móvel pode ser realizada conforme vários fatores, um deles é quanto ao meio de locomoção, ou seja, construção física, onde tem-se três tipos: aéreo, aquáticos e terrestres. Os robôs aéreos, geralmente são de asas fixas ou rotativas e entram na classificação de *UAVs (Unmanned Aerial Vehicle)*, atualmente os exemplos mais populares são os drones (ver Figura 5). Os robôs voadores são uma ferramenta popular para monitorar e realizar vigilância de alvos em várias aplicações militares e civis, incluindo vigilância do tráfego terrestre, inspeção de campos agrícolas, monitoramento e inspeção de infraestruturas, vigilância de áreas de desastres naturais, missões de resgate e monitoramento terrestre para fins de segurança. Estes apresentam algumas limitações em termos de duração do voo, carga útil e tamanho.

Figura 5 – Drone de pulverização utilizado no setor agro

Fonte: Max Pixel(2021)

Para o meio aquático existem os *Autonomous Underwater Vehicle* (AUVs) que são principalmente utilizados para a exploração de áreas oceânicas desconhecidas. Eles podem mapear o leito do oceano, coletar dados sobre a topografia do fundo do mar, identificar recursos minerais e estudar ecossistemas submarinos. Na Figura 6 temos um exemplo, o robô submarino da empresa *Terradepth* feito para investigar os oceanos.

Figura 6 – Robô submarino

Fonte: Terradepth (2020)

Os robôs terrestres podem ser divididos em 3 grupos dependendo do tipo de locomoção sendo elas rodas, esteiras ou articulados. Eles são amplamente utilizados na agricultura para tarefas como plantio, colheita, irrigação e pulverização de culturas. Eles podem ser programados para seguir trajetórias precisas e otimizar

o uso de recursos, como água e fertilizantes. Também podem ser utilizados para logística e transporte de carga. Eles podem ser empregados em armazéns para mover mercadorias e em locais como portos e aeroportos para transporte de cargas pesadas. Veja um exemplo na Figura 7.

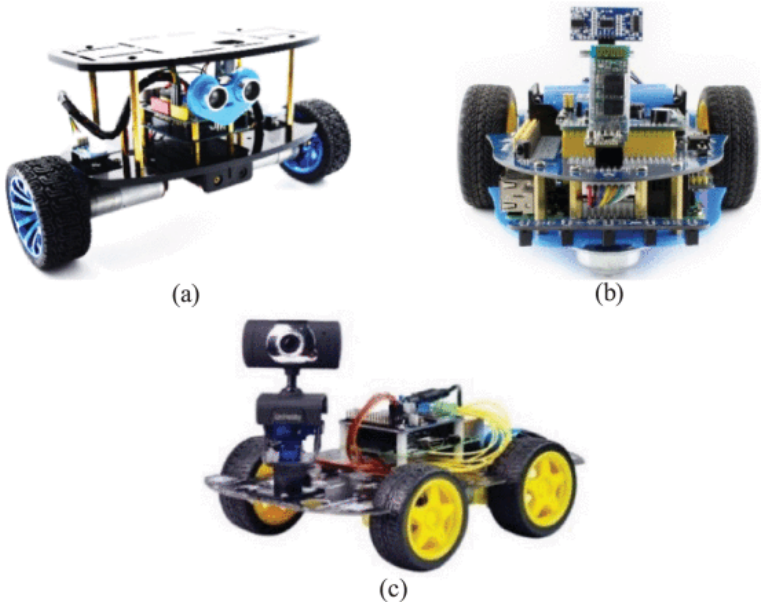
Figura 7 – Robô terreste

Fonte: XAG(2021)

No âmbito de robôs terrestres destacam-se os robôs móveis com rodas. Estes são utilizados para gerenciamento de estoque, automação industrial, vigilância militar, entre outros. São classificados em diferentes categorias, como robôs semelhantes a carros, robôs omnidirecionais e robôs de acionamento diferencial; sendo os robôs de acionamento diferencial uma classe proeminente (R MATHEW; S. S. HIREMATH 2019).

O *Differential Drive Wheeled Mobile Robot* (DDWMR) tem sido cada vez mais notado e amplamente aplicado na área da Robótica. Ele possui diversas vantagens, como ampla capacidade de movimentação, uma estrutura simples e custos de produção baixos. O DDWMR pode ser encontrado em diversos modelos, classificados pela quantidade de rodas, DDWMR de duas rodas, três rodas ou quatro rodas. O tipo de 3 rodas é a forma mais comum, compreendendo duas rodas fixas motorizadas montadas nos lados esquerdo e direito da plataforma do robô e uma roda passiva de apoio usada para equilíbrio e estabilidade (A. STEFEK et al. 2020). A Figura 8 exemplifica os 3 modelos.

Figura 8 – Robôs diferenciais de duas, três e quadro rodas



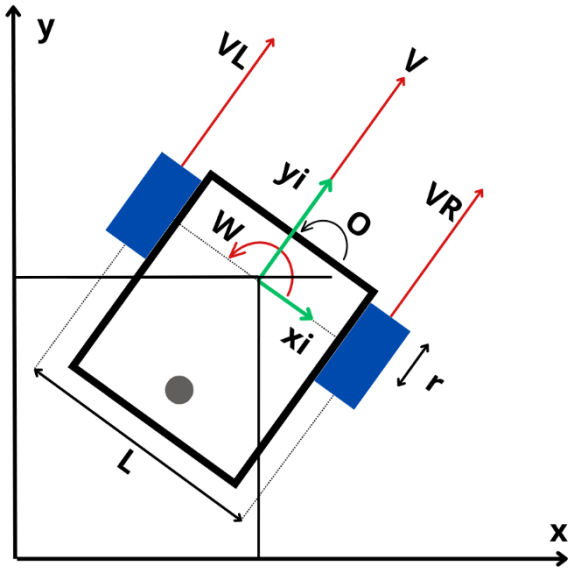
Fonte: A. STEFEK. et al (2020)

2.2 CINEMÁTICA DO ROBÔ DIFERENCIAL

O projeto desenvolvido nesse trabalho é de um robô diferencial (DDWMR), de três rodas, que consiste em duas rodas paralelas acionadas de forma independente que giram sobre um mesmo eixo e um *caster ball* como terceira roda e ponto de equilíbrio para estabilizar o robô na horizontal. O esboço do robô é ilustrado na Figura 9 com todos os detalhes para o cálculo da sua cinemática.

Esse tipo de robô móvel possibilita movimentar-se para frente e para trás, e também controlar o ângulo de direção através da modificação da velocidade linear da roda direita (Vr) e da roda esquerda (Vl). Devido a essa configuração, esses robôs demonstram grande manobrabilidade e capacidade de realizar curvas fechadas, tornando-os ideais para navegar em espaços reduzidos e executar manobras precisas. A simplicidade desse modelo frequentemente resulta em custos de construção mais acessíveis quando comparados a outros tipos de robôs. Além disso, sua facilidade de controle proporciona alta manobrabilidade e a habilidade de girar em torno do centro do robô (P. PETROV; V. GEORGIEVA, 2018). O DDWMR de três rodas, em particular, possui uma estrutura simples, sendo adequado para diversas aplicações práticas, popular e predominante no mercado.

Figura 9 – Cinemática robô diferencial 2 rodas



Fonte: O autor (2023).

As variáveis da Figura 9 estão descritas na Tabela 2 abaixo

Tabela 2 – Variáveis do modelo

Variáveis	Descrição
x	Eixo global x
y	Eixo global y
xi	Eixo robô xi
yi	Eixo robô yi
VR	Velocidade linear da roda direita
WR	Velocidade angular da roda direita
VL	Velocidade linear da roda esquerda
WL	Velocidade angular da roda esquerda
L	Distância entre os centros das duas rodas
W	Velocidade angular do robô
V	Velocidade linear do robô
O	Orientação do robô ao eixo x
r	Raio da roda

Fonte: O autor (2023).

As variáveis descritas na tabela 2 serão utilizadas para descrever a cinemática do modelo. Aqui temos que Vx representa a velocidade linear em relação ao eixo x global, Vy representa a velocidade linear em relação ao eixo y global e W representa a velocidade de rotação do robô em seu próprio eixo.

$$Vx = \frac{r}{2}(WR + WL)cos(O) \quad (1)$$

$$Vy = \frac{r}{2}(WR + WL)sin(O) \quad (2)$$

$$W = \frac{r}{2L}(WR - WL) \quad (3)$$

2.3 SENSORES E ATUADORES

Os robôs móveis possuem diferentes configurações e podem usar diversos hardwares embarcados para realizar as tarefas para as quais foram projetados. Dentre os vários subsistemas de um robô os mais pertinentes são os atuadores, comunicação, unidade de controle (ECU) e sensores. Qualquer sistema robótico pode incluir muitos desses componentes ou simplesmente alguns (KURDILA; BEM-TZVI, 2020). Tomando como referências os robôs educacionais do mercado, que foram apresentados no início desse trabalho, destacam-se os seguintes componentes para a construção de uma plataforma educacional de robótica móvel.

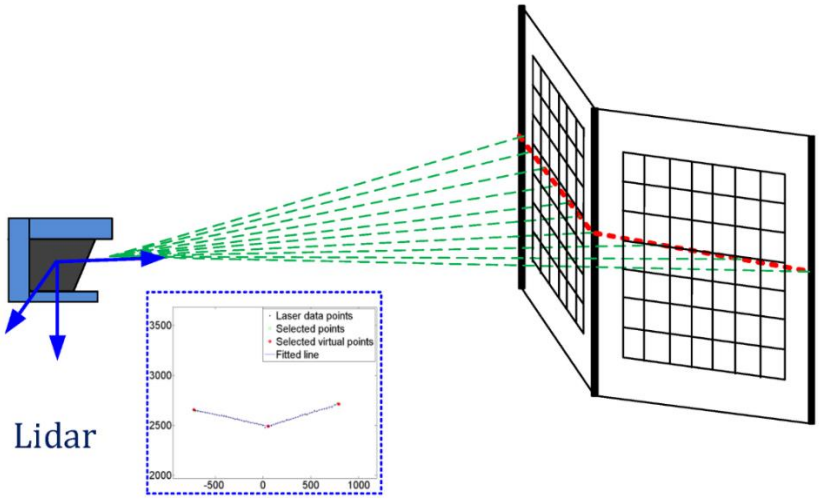
2.3.1 Lidar

O *Light Detection and Ranging* (LIDAR) em robôs têm se destacado como uma abordagem fundamental para habilitar sistemas robóticos na realização de tarefas de detecção de objetos e mapeamento. Os benefícios dos LIDARs incluem sua capacidade de operar em diferentes condições ambientais, como baixa luminosidade, o que inviabilizaria o uso de câmeras (CSABA BENEDEK et al. 2021). Além disso, sua precisão na medição de distâncias e na criação de mapas detalhados torna-os essenciais para a autonomia de robôs em ambientes dinâmicos e complexos.

O LIDAR é uma tecnologia ativa de sensoriamento remoto que utiliza ondas eletromagnéticas na faixa óptica para detectar um objeto, determina a distância entre o alvo e o sensor e mede outras propriedades físicas da superfície do alvo, como espalhamento e reflexão (J.C.F. DIAZ et al. 2017). O sensor calcula a distância dos objetos-alvo a partir do tempo de eco do feixe de laser emitido e detectado, onde o feixe se propaga com a velocidade da luz. O resultado da medição é uma

nuvem de pontos 3D como ilustra a Figura 10, onde as coordenadas dos pontos são fornecidas em um sistema de coordenadas local ou global, dependendo do tipo do sistema LIDAR e da área de aplicação.

Figura 10 – Lidar em funcionamento



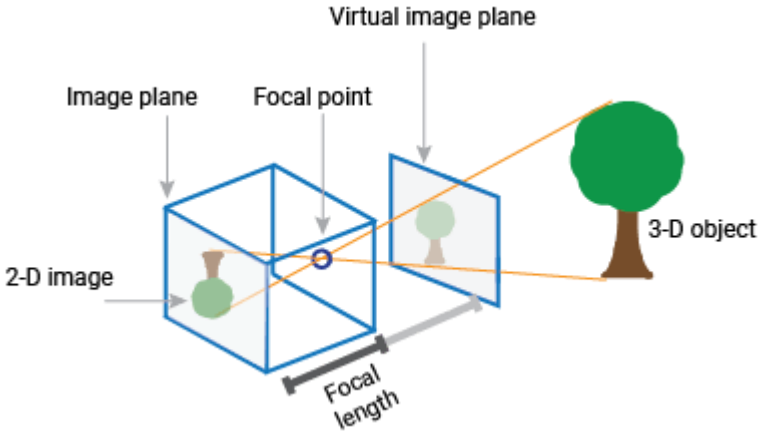
Fonte: superannotate (2023)

2.3.2 Câmera

Uma câmera opera com base em princípios ópticos e eletrônicos. Ela consiste em um sistema óptico composto por lentes que capturam a luz refletida pelos objetos ao redor, como ilustra a Figura 11. Essa luz é focalizada em um sensor de imagem, geralmente um dispositivo semicondutor conhecido como *charge-coupled device* (CCD) ou *Complementary Metal-Oxide-Semiconductor* (CMOS). O sensor converte a luz em sinais elétricos, que são então processados para formar uma imagem (MOHD JAVAID, 2021).

A câmera desempenha um papel fundamental nos sistemas de percepção, orientação e navegação de robôs autônomos. A literatura destaca uma variedade de aplicações que demonstram a versatilidade desse componente, como a detecção e identificação de obstáculos, facilitando a navegação segura de robôs em ambientes dinâmicos, auxiliando robôs industriais na localização precisa, agarre e movimentação de itens. Todos os robôs comerciais apresentados na introdução deste trabalho possuem câmeras.

Figura 11 – Funcionamento de uma câmera



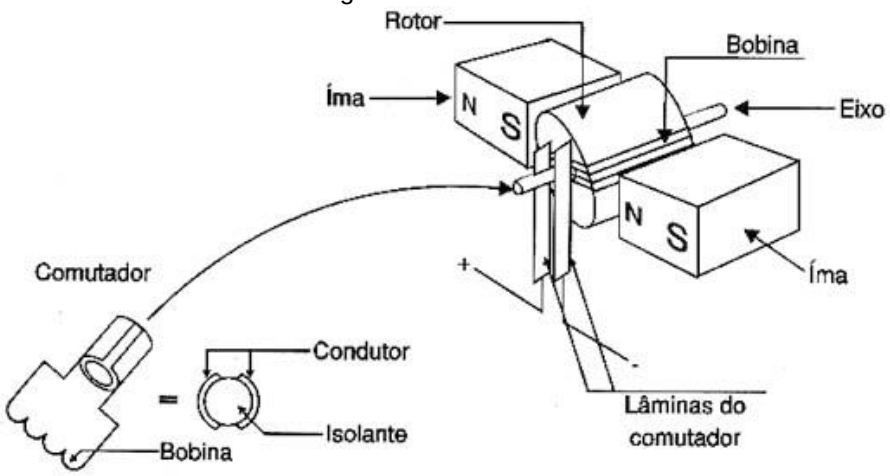
Fonte: Mathworks(2023)

2.3.3 Motor DC

Para que o robô móvel diferencial se movimente é necessário a utilização de um sistema de locomoção com motores, neste trabalho serão utilizados motores dc. Um motor DC (Corrente Contínua) é um dispositivo eletromecânico que converte energia elétrica em movimento mecânico rotativo. Seu princípio de funcionamento baseia-se na interação entre o campo magnético produzido por ímãs permanentes e a corrente elétrica que flui através de uma bobina, geralmente chamada de rotor. A corrente elétrica, fornecida por uma fonte de alimentação, cria um campo magnético ao redor da bobina, interagindo com o campo magnético dos ímãs e gerando um torque que resulta no movimento rotativo do rotor. Os motores DC podem ser classificados em motores de escova e sem escova, sendo os primeiros caracterizados pelo uso de escovas para a comutação da corrente (CHAPMAN, STEPHEN J., 2013).

Os motores DC são muito utilizados na robótica oferecendo uma solução versátil para o controle preciso de movimentos em diversas aplicações. Sua eficiência e capacidade de proporcionar diferentes níveis de torque e velocidade fazem deles escolhas ideais para sistemas de locomoção robótica e manipulação de objetos, a Figura 12 ilustra um motor DC e seus componentes.

Figura 12 – Motor DC



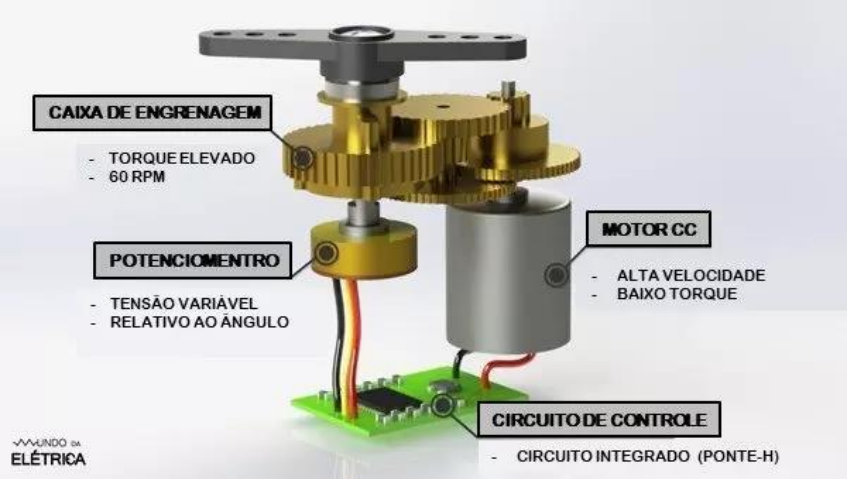
Fonte: Newtoncbraga(2023).

2.3.4 Servomotor

Os servo motores de corrente contínua são dispositivos, notáveis por seu controle preciso de posição e velocidade. A construção básica de um servo motor DC inclui um motor de corrente contínua, um sistema de feedback para monitorar a posição do eixo, e um controlador incorporado. O motor converte a energia elétrica em rotação, enquanto o sistema de feedback fornece informações em tempo real sobre a posição angular do eixo. O controlador utiliza esses dados para ajustar continuamente a energia fornecida ao motor, garantindo um posicionamento preciso. A Figura 13 representa o sistema de um servo motor.

Os servo motores DC desempenham um papel crucial na robótica, onde a precisão e a resposta dinâmica são essenciais. Em sistemas robóticos, esses motores são frequentemente empregados para controlar articulações e movimentos precisos. Sua capacidade de feedback em tempo real permite que os robôs ajustem suas posições e movimentos com precisão

Figura 13 – Servo motor

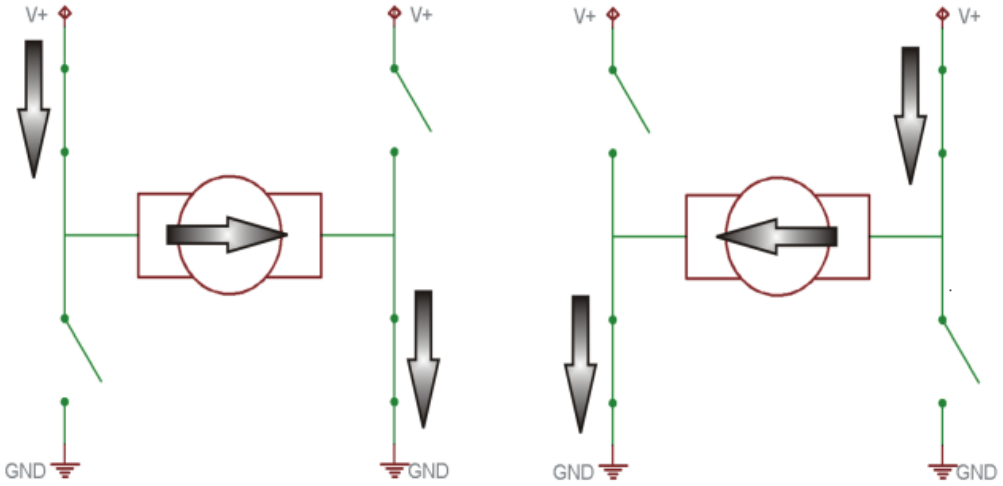


Fonte: mundodaeletrica(2023).

2.3.5 Controlador

Para que o robô realize sua locomoção da forma desejada é fundamental que a movimentação do motor seja controlada. Dentre os tipos mais comuns, destaca-se o controlador baseado na Ponte H, que utiliza uma configuração de quatro interruptores para controlar a polaridade da corrente que flui através do motor, permitindo movimento bidirecional como exibe a Figura 14.

Figura 14 – Funcionamento de uma ponte H

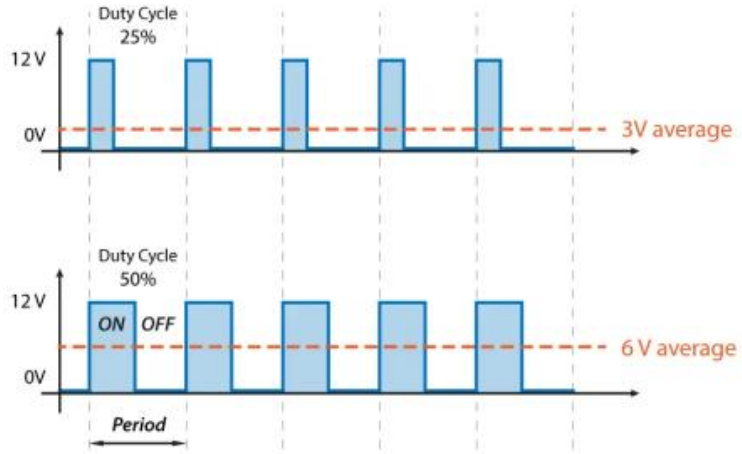


Fonte: Robocore(2023).

Além disso, a técnica de *Pulse Width Modulation* (PWM) é frequentemente empregada nesses controladores para ajustar a velocidade do motor. O PWM regula a média da potência entregue ao motor variando a largura dos pulsos, para obter

resultados analógicos com meios digitais. O controle digital é usado para criar uma onda quadrada, um sinal alternado entre ligado e desligado como exibe a Figura 15. Este padrão on-off pode simular tensões entre o Vcc total da placa (por exemplo, 5 V) e desligado (0 V), alterando a parte do tempo que o sinal fica ligado versus o tempo que o sinal fica desligado (SOHAN, MD SHAHED. 2023).

Figura 15 – Sinal PWM



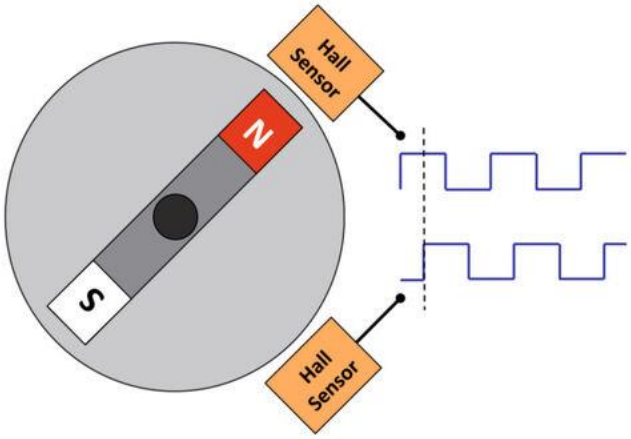
Fonte: SOHAN, MD SHAHED. (2023)

2.3.6 Hall encoder

Um sensor Hall é um dispositivo que utiliza o efeito Hall para medir campos magnéticos. Consiste em um material semicondutor com uma corrente elétrica fluindo por ele. Quando exposto a um campo magnético externo, ocorre uma deflexão nos portadores de carga, gerando uma diferença de potencial, conhecida como sinal Hall. Este sinal é proporcional à intensidade do campo magnético, permitindo a detecção precisa de sua presença e intensidade.

Na robótica, o sensor Hall é amplamente empregado para medir e controlar a velocidade de motores DC. Posicionado estrategicamente próximo ao componente rotativo do motor, o sensor Hall detecta as mudanças no campo magnético gerado pelos ímãs permanentes no rotor. Essas variações são convertidas em sinais elétricos pulsados cuja frequência é proporcional à velocidade de rotação (Figura 16).

Figura 16 – Hall encoder e seu sinal



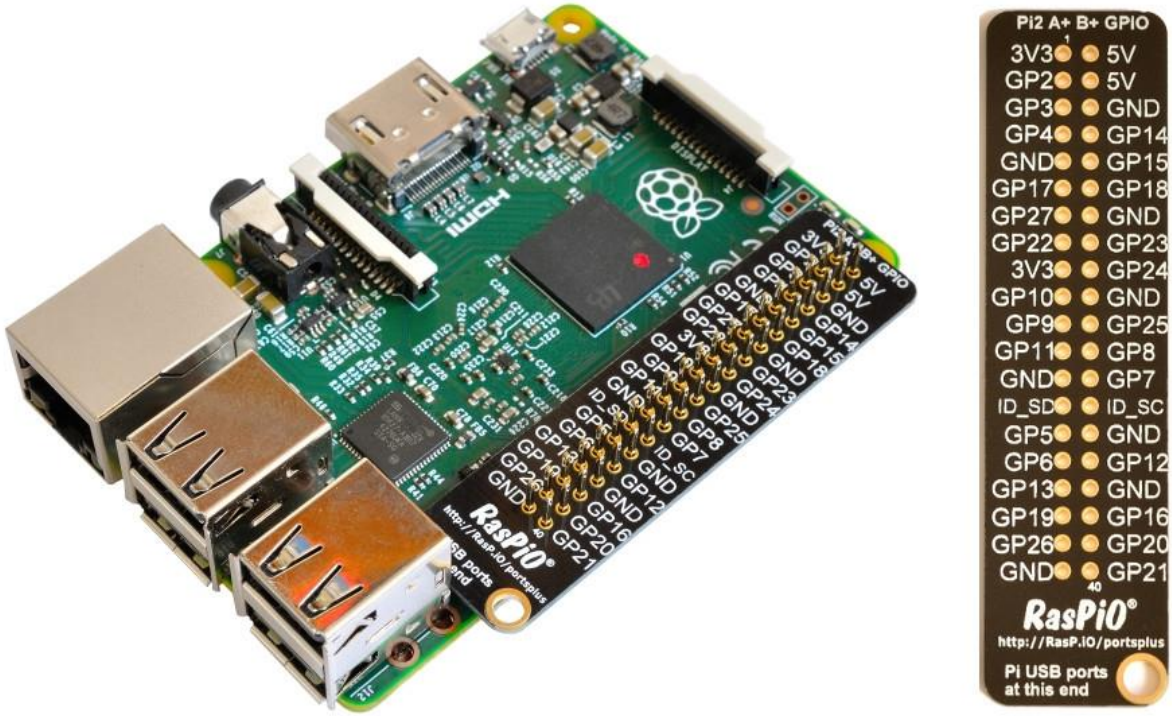
Fonte: elektronikpraxis(2023)

2.3.7 GPIO Entrada/Saída de Propósito Geral

Para facilitar o acesso a propriedade internas dos microcontroladores possuem uma interface disponível chamada de *General Purpose Input/Output* (GPIO) ele permite múltiplas interações e aplicações simultâneas. Esses pinos podem ser programados como entradas, onde dados de alguma fonte externa são inseridos no sistema para serem manipulados em um momento e local desejados. Saídas também podem ser realizadas nos GPIOs, permitindo a transmissão eficiente de dados formatados para dispositivos externos. Isso fornece um mecanismo simples para programar e retransmitir dados com base nas preferências do usuário por meio de uma única interface de porta (Raspberry 2023).

Nesses pinos sinais podem ser enviados ou recebidos de outros dispositivos. Em muitas aplicações, os GPIOs podem ser configurados como linhas de interrupção, permitindo que a CPU sinalize o processamento imediato das linhas de entrada. Em muitos designs mais recentes, eles também têm a capacidade de controlar e usar Acesso Direto à Memória para transferir blocos de dados de maneira mais eficiente. Essencialmente, todas as portas podem ser adaptadas para atender a metas de design específicas e proporcionar reutilização em aplicações, ver Figura 17.

Figura 17 – GPIO do *Raspberry*



Fonte: *Raspberry* (2023)

2.4 MODELAGEM 3D

Com os principais equipamentos que compõem um robô descrito, a próxima etapa do desenvolvimento é a modelagem da estrutura que suportará esses equipamentos e dará forma ao robô e para isso é importante entender mais sobre as ferramentas de *computer aided design* (CAD). Os *softwares* são elementos essenciais na engenharia, especialmente nos campos da mecânica e robótica, desempenhando um papel crucial ao longo de décadas de desenvolvimento tecnológico. A importância dessas ferramentas reside na eficiência proporcionada ao processo de design, na garantia de precisão e na promoção da inovação em componentes mecânicos (KIRPES C. et al. 2022).

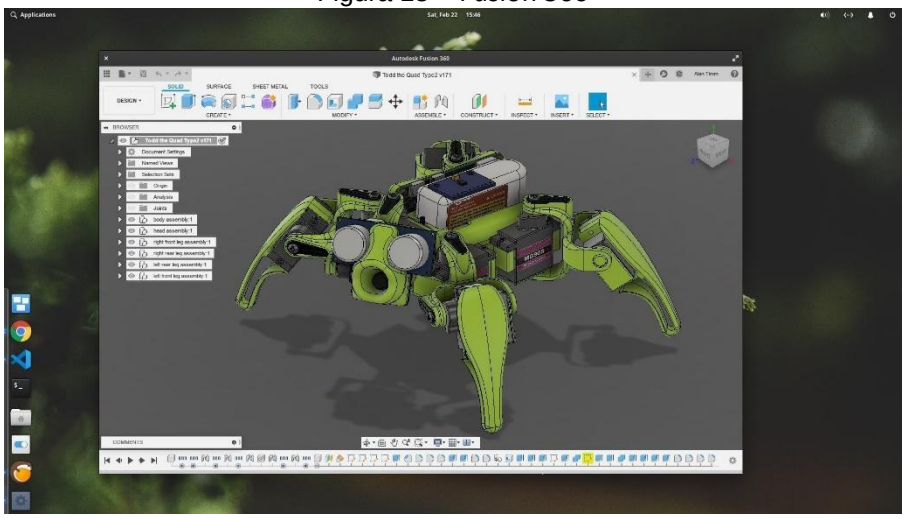
No âmbito das aplicações, as ferramentas CAD encontram vasto emprego na engenharia mecânica, no design de máquinas e na criação de sistemas robóticos complexos. Essas aplicações abrangem desde o projeto de componentes individuais até a simulação do movimento de robôs inteiros antes da construção física. Além disso, o CAD é fundamental na prototipagem rápida e na colaboração eficiente entre equipes multidisciplinares. Sua evolução ao longo das décadas possibilitou não apenas maior acessibilidade, mas também a incorporação de tecnologias

avançadas, solidificando seu papel como uma ferramenta indispensável na engenharia moderna.

No mercado existem muitas ferramentas CAD, dentre elas se destaca o Fusion 360, um software de design e engenharia assistido por computador (CAD/CAM/CAE) desenvolvido pela Autodesk. Ele oferece diversas funcionalidades para projetos de design, engenharia e fabricação. O Fusion 360 permite criar modelos 3D detalhados de peças e montagens, facilitando a visualização e a análise de projetos complexos. Isso é especialmente útil para estudantes de engenharia.

O Fusion é baseado na nuvem, o que significa que você pode acessar seus projetos de qualquer lugar com uma conexão à internet. Isso facilita a colaboração em equipe, pois várias pessoas podem trabalhar no mesmo projeto simultaneamente. A Autodesk oferece uma versão gratuita do Fusion 360 para estudantes universitários, o que torna a ferramenta acessível para uso educacional. Além disso, possui uma comunidade ativa que produz e disponibiliza de maneira gratuita diversos modelos de equipamentos e peças. O que resulta em uma escolha popular para projetos universitários. A Figura 18 exibe a tela principal do Fusion 360.

Figura 18 – Fusion 360



Fonte: Autodesk (2023)

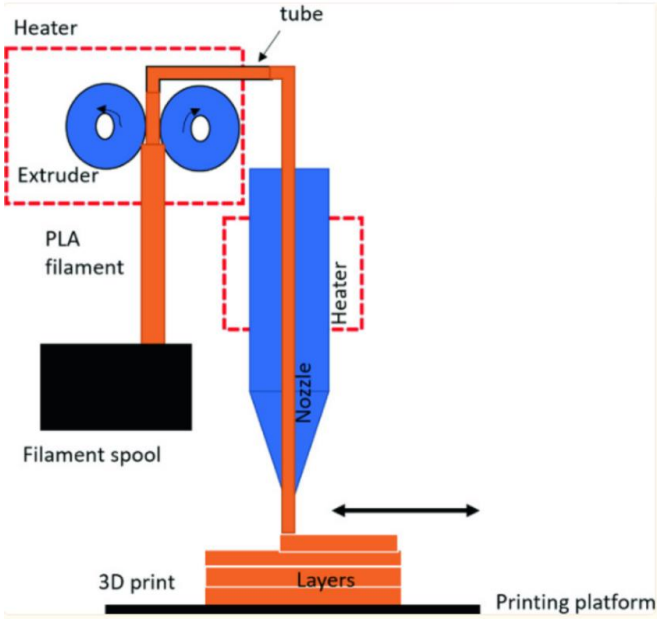
2.5 MANUFATURA ADITIVA (FDM)

A etapa posterior a modelagem de uma peça, componente ou estrutura é a sua fabricação. Existem uma infinidade de métodos de fabricação que dependem do objetivo final, recursos financeiros e equipamentos disponíveis para o uso. Neste

trabalho será utilizado um método de impressão 3D, portanto é necessário ter maior conhecimento sobre esse processo.

O método de fabricação aditiva mais difundido é a *Fused Deposition Modeling* (FDM), devido à sua plataforma de baixo custo e ao movimento de *open source* (ANDI D.; GEORGE-CHRISTOPHER V., 2018). Esse método utiliza polímeros como material bruto em forma de filamento. O princípio do processo FDM é ilustrado em um diagrama esquemático na Figura 19. Como mostrado, o filamento é continuamente alimentado através do extrusor e bico da máquina por meio de dois rolos girando em direções opostas. O material é depositado na mesa de construção camada por camada até que a forma e o tamanho do produto desejados sejam alcançados. Durante o processo de camadas, o bico da impressora navega sobre o plano conforme as coordenadas espaciais do modelo CAD original nos arquivos de código G até que o tamanho e a forma desejados do componente sejam produzidos. Geralmente, a resolução e a eficácia da extrusão dependem em grande parte das propriedades do filamento termoplástico e, como tal, diferentes impressoras 3D são projetadas para materiais específicos. Entretanto existem impressoras com capacidade de imprimir diversos tipos de filamento sem a necessidade de alterações em sua estrutura sendo o ácido polilático (PLA) o material mais comum (MWEMA FM; AKINLAB ET., 2020).

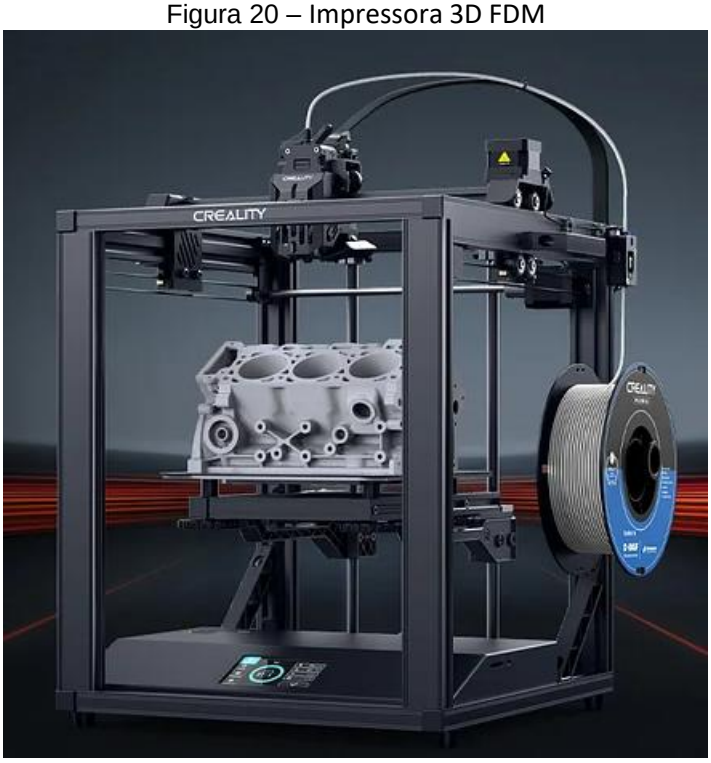
Figura 19 – Funcionamento de uma Impressora 3D FDM



Fonte: Adaptado de (MWEMA FM; AKINLAB ET., 2020).

Essa técnica é amplamente aplicada na fabricação de estruturas e geometrias complexas a partir de modelos 3D. A impressão 3D que incorpora diversos métodos, materiais e equipamentos, evoluiu ao longo dos anos e tem sido capaz de revolucionar os processos de produção e logística. A produção aditiva pode ser encontrada em diferentes indústrias, incluindo indústria de defesa, indústria de aviação, construção, prototipagem e biomecânica. A crescente acessibilidade dessa tecnologia se deve, em grande parte, ao término de patentes antigas, que abriu caminho para que fabricantes desenvolvessem novos modelos de impressoras 3D. Além disso, avanços recentes resultaram na redução dos custos dessas impressoras, tornando-as amplamente disponíveis e aplicáveis em uma variedade de ambientes, como escolas, residências, bibliotecas e laboratórios. Seu uso ajuda a minimizar os custos adicionais associados ao processo de desenvolvimento de produtos. Na Figura 20 é possível ver um modelo de impressora 3D completa.

Figura 20 – Impressora 3D FDM



Fonte: Creality (2023).

2.6 FATIAMENTO PRUSA SLICER

Os modelos 3D para serem fabricados pelo processo de manufatura aditiva, precisam passar por uma etapa de preparação. Nessa fase um *software* de fatiamento 3D é usado para converter o arquivo de modelagem CAD para formato de arquivo de impressão o G-code. O *PrusaSlicer*, ótima alternativa de *software* de fatiamento desenvolvido pela Prusa *Research*, a mesma empresa por trás das populares impressoras 3D Prusa, é amplamente utilizado na comunidade de impressão 3D, devido à sua compatibilidade e integração com uma variedade de impressoras 3D no mercado.

Ele é gratuito e *open source*, graças a grande e forte comunidade de desenvolvedores da Prusa *Research* o *software* continua recebendo novas funcionalidades e sempre atualizado. Essa grande comunidade também contribui testando diversos parâmetros de impressão para cada modelo e fabricante de impressoras 3D, assim o Prusa armazena os melhores parâmetros que podem ser selecionados pelo usuário e garantir uma impressão perfeita. A sua interface é limpa e simples de usar o que facilita o aprendizado de novos usuários. Ele permite a customização de suportes, que podem ser configurados diretamente no modelo 3D dentro do *software* e também é possível usar malhas personalizadas como bloqueadores e reforços.

2.7 SISTEMA OPERACIONAL UBUNTU MATE

Quando a plataforma robótica está fabricada e montada, a próxima etapa é a configuração do ambiente de software que dependerá do computador utilizado na plataforma. Neste trabalho será utilizado o Ubunto MATE. O Ubuntu MATE representa uma escolha adequada de sistema operacional, destacando-se por sua estabilidade e facilidade de uso, proporcionando um desempenho otimizado em computadores, tanto modernos quanto mais antigos, devido a seus requisitos de hardware modestos (Ubuntu, 2023). O site oficial do Ubuntu MATE oferece imagens pré-configuradas para diversos modelos de Raspberry Pi, simplificando significativamente o processo de instalação. A Figura 21 apresenta o ubuntu Mate em uma *raspyberry pi 4.*

O ambiente de desktop MATE é uma implementação abrangente que abarca um gerenciador de arquivos para acesso local e em rede, editor de texto, gerenciador de arquivos compactados, visualizador de imagens, leitor de documentos, monitor de sistema e terminal. Todos esses componentes são altamente personalizáveis e gerenciados por meio de um centro de controle, proporcionando facilidade para usuários envolvidos no desenvolvimento de aplicações nesse sistema operacional. Dada a natureza deste projeto, que busca criar uma plataforma educacional de robótica móvel acessível, o Ubuntu MATE surge como uma excelente opção, oferecendo uma interface completa, amigável e demandando requisitos modestos de hardware, o que contribui para a economia na implementação.

Figura 21 – Ubuntu mate na *Raspberry*

Fonte: O autor (2023).

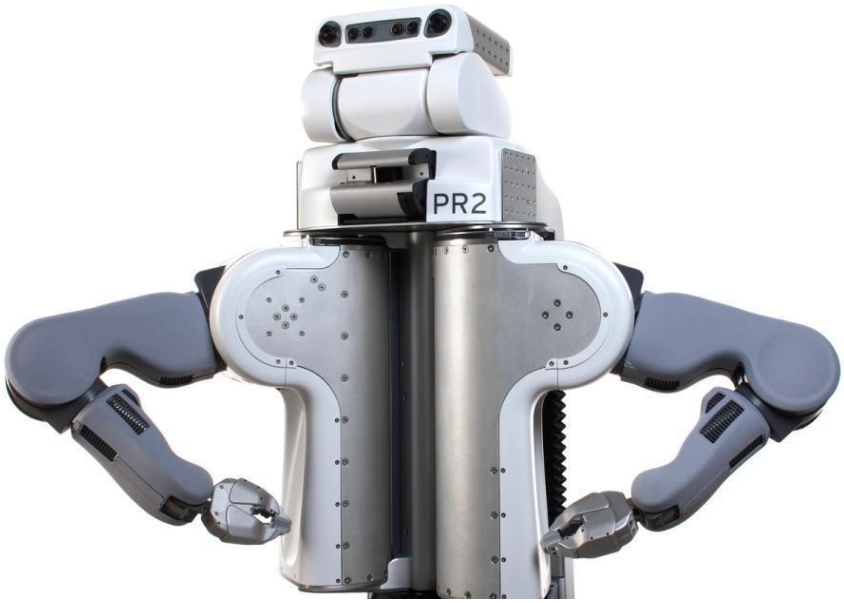
2.8 ROBOT OPERATED SISTEM (ROS)

Com o ambiente de *software* preparado, desenvolver o *software* que comandará as ações de um robô autônomo, com sua variedade de hardwares, é uma tarefa desafiadora. Nesse contexto surge o ROS, ou *Robot Operating System*, representa um *framework* de *software* que emergiu como a interface padrão para robôs no cenário acadêmico. Este *framework* é composto por uma coleção abrangente de ferramentas, bibliotecas e convenções projetadas para simplificar a complexa tarefa de desenvolver comportamentos robóticos em diversas plataformas. Desde sua concepção, o ROS foi estruturado para fomentar o desenvolvimento colaborativo em robótica, tanto em escalas menores quanto em projetos de grande

envergadura. Sua arquitetura foi elaborada de maneira a facilitar a colaboração entre especialistas de diferentes áreas, permitindo que eles trabalhem simultaneamente em projetos e, posteriormente, integrem suas contribuições sem complicações significativas.

Na Figura 22, podemos observar o robô PR2, uma plataforma frequentemente utilizada como ambiente de aprendizado para o ROS. Essa escolha é indicativa da relevância do ROS não apenas como uma ferramenta robusta para pesquisa e desenvolvimento, mas também como uma plataforma de ensino crucial para aqueles que buscam compreender e trabalhar com sistemas robóticos avançados.

Figura 22 – Robô PR2.



Fonte: robotsguide (2023)

Segundo (QUIGLEY et al.(2009)) os principais aspectos do ROS, que representam a filosofia por trás da sua criação são:

- *Peer To Peer*: o sistema ROS consiste em vários pequenos programas de computador que se conectam e trocam mensagens continuamente, a forma dessas mensagens e a taxa de envios ou recebimento podem ser configuradas e modeladas de acordo com o objetivo do projeto. As mensagens viajam, de um programa para o outro, não há um serviço de roteamento central
- *Multi-lingual*: muitas tarefas são mais fáceis de realizar em linguagens de *script* de “alta produtividade” como Python ou Ruby, outras exigem um maior

desempenho e requerem uso de linguagens mais otimizadas como C++. Atualmente o ROS aceita C++, Python, Java, Javascript, Ruby, R, Julia e outras.

- *Tool based:* para gerenciar as complexidades do ROS, ele foi feito com um *design* de micro *kernel*, onde um grande número de ferramentas são usadas para executar e criar os vários componentes, tornando o sistema totalmente modular.
- *Open source:* o código fonte do ROS está totalmente disponível, isso acelera o desenvolvimento do sistema. Ele é distribuído sobre termos de licença permissiva BSD que permite o uso comercial e não comercial.

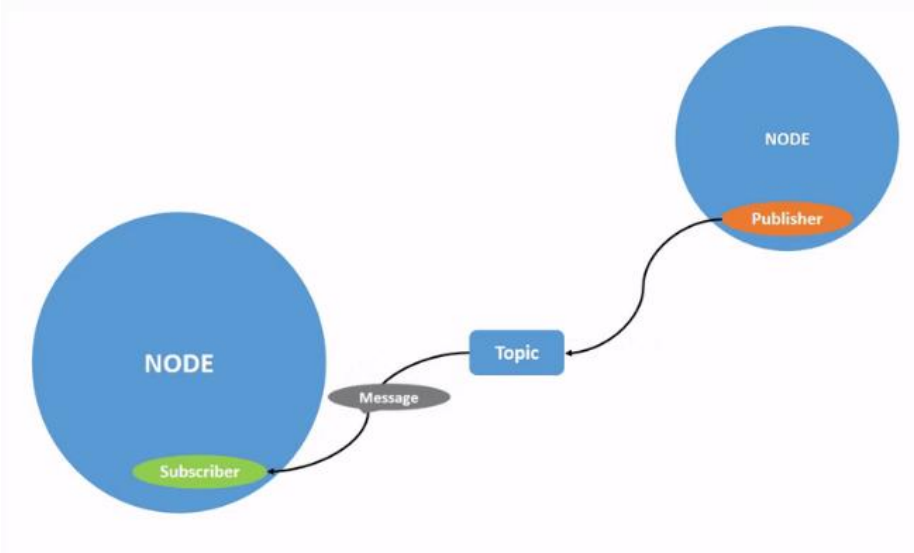
O ROS é organizado em Pacotes, cada um dos quais contém alguma combinação de código, dados e documentação. Por ter uma comunidade ativa hoje existem milhões de pacotes no ecossistema ROS, cobrindo aspectos relacionados a robótica, carros autônomos, drones, desde implementações de prova de conceito de novos algoritmos a recursos de qualidade industrial. A comunidade de usuários do ROS se baseia em uma infraestrutura comum para fornecer um ponto de integração e assim oferecer acesso a drivers de *hardware*, recursos genéricos de robôs, ferramentas de desenvolvimento e bibliotecas externas úteis.

Para iniciar o aprendizado com o ROS além de entender a sua filosofia é importante saber os conceitos fundamentais de nós, tópicos e serviços.

2.8.1 Nós ROS

Cada nó no ROS deve ser responsável por um único propósito, esse conceito facilita a modularização do seu projeto. Os nós podem ser usados para controlar os motores das rodas ou publicar os dados do sensor de um medidor de alcance a laser. Cada nó pode enviar dados e receber dados de outros nós por meio de tópicos, serviços, ações ou parâmetros. Na figura 23, o nó que realiza o envio é o *Publisher* e o que recebe é *Subscriber.*

Figura 23 – Exemplo de nós *Publisher* e *Subscriber*.

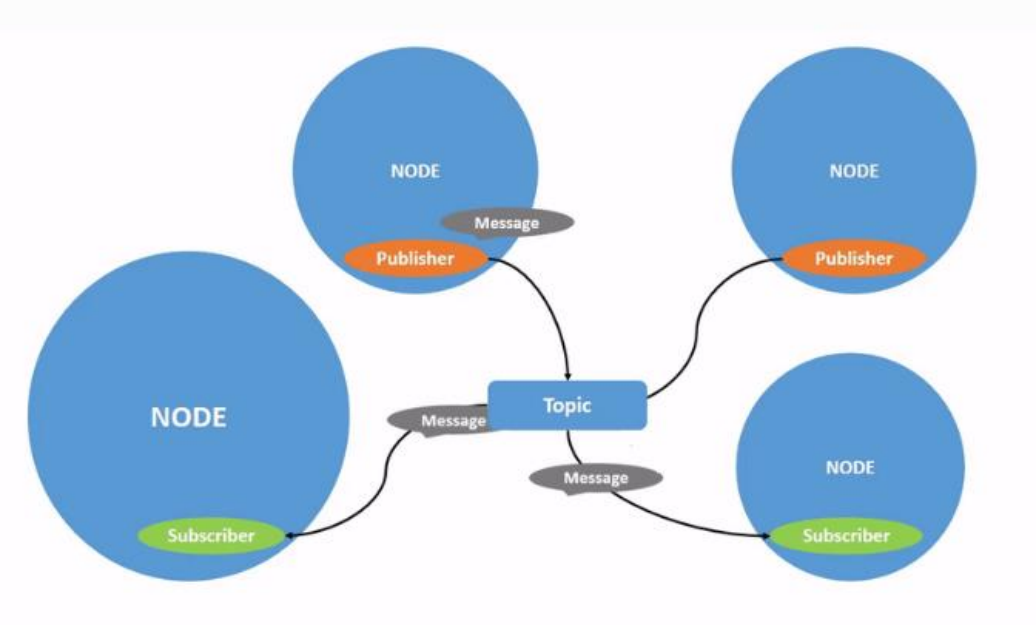


Fonte: ROS (2023)

2.8.2 Tópicos ROS

Tópicos são canais de comunicação que permitem que diferentes partes de um sistema ROS troque informações entre si. Eles desempenham um papel fundamental na arquitetura de comunicação do ROS. Um nó do ROS envia e recebe informações para um ou mais nós através de tópicos. Eles permitem uma forma assíncrona de comunicação, assim um nó não precisa esperar que o outro esteja pronto para receber dados, ele faz a publicação de uma mensagem em um tópico e todos que estiverem inscritos recebem. Os dados trocados em tópicos são estruturados em mensagens. As mensagens definem o formato e o tipo de dados que estão sendo transmitidos. O usuário pode criar o seu tipo de mensagem, mas já existem uma ampla variedade de mensagens criadas pela comunidade. Na Figura 24 podemos ver quatro nós que estão trocando mensagens através de um único tópico.

Figura 24 – Tópicos em ROS

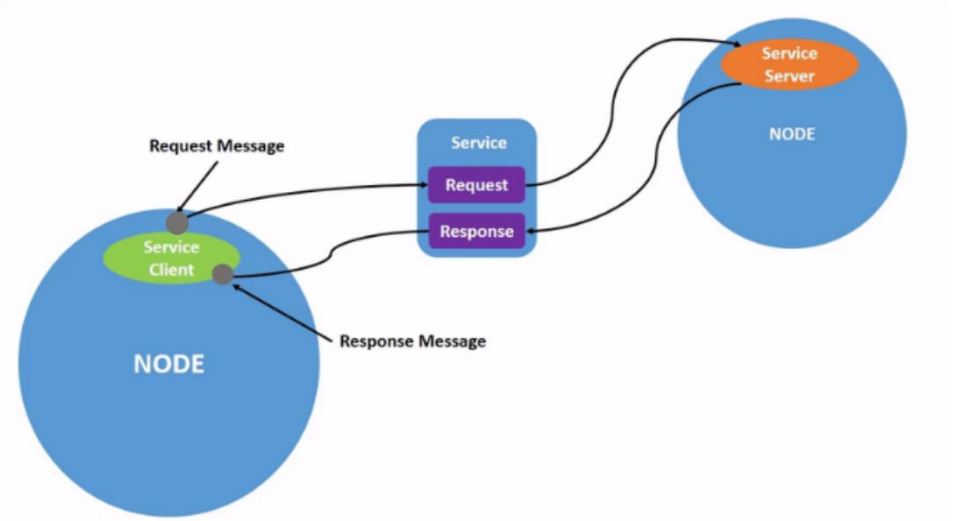


Fonte: ROS(2023)

2.8.3 Serviços ROS

Os serviços são outro método de comunicação para nós. Os serviços são baseados em um modelo de chamada e resposta, ao contrário do modelo de publicação e assinatura dos tópicos. Enquanto os tópicos permitem que os nós se inscrevam em fluxos de dados e recebam atualizações contínuas, os serviços fornecem dados apenas quando são especificamente chamados por um cliente, como exibe a Figura 25.

Figura 25 – Exemplo Serviço ROS.



Fonte: ROS(2023)

2.9 BIBLIOTECAS UTILIZADAS

Outra ferramenta que auxilia o desenvolvimento de *softwares*, são as bibliotecas. Bibliotecas são conjuntos de código pré-desenvolvido que oferecem funcionalidades específicas, eliminando a necessidade de desenvolver o *software* do zero a cada novo projeto. Essas ferramentas são fundamentais para acelerar o processo de programação, reduzir erros e promover a reutilização de código. No contexto robótico, onde a complexidade dos sistemas exige soluções especializadas, as bibliotecas desempenham um papel fundamental. Elas abrangem desde rotinas básicas até implementações avançadas de algoritmos, oferecendo aos desenvolvedores uma variedade de recursos prontos para uso.

Essa abordagem não apenas simplifica o desenvolvimento, mas também facilita a colaboração entre equipes, pois a padronização proporcionada pelas bibliotecas promove uma compreensão consistente e compartilhada do código. Ao empregar bibliotecas, os desenvolvedores de robótica podem se concentrar nas características exclusivas de seus projetos, aproveitando soluções confiáveis e testadas pela comunidade. Isso não apenas economiza tempo, mas também contribui para a criação de sistemas mais robustos e inovadores. A seguir, serão apresentadas bibliotecas utilizadas no desenvolvimento deste trabalho.

2.9.1 Biblioteca RPI

A biblioteca RPI é amplamente usada para interagir com os pinos GPIO de um *Raspberry Pi*. GPIO é a maneira pela qual o *Raspberry Pi* pode se comunicar com dispositivos externos e sensores, permitindo que você controle e leia informações de componentes eletrônicos. A biblioteca "RPI.GPIO" facilita o acesso aos pinos GPIO usando a linguagem de programação *Python*.

A biblioteca "RPI.GPIO" é uma escolha popular entre os desenvolvedores que desejam criar um projeto, pois oferece uma interface Python simples e eficaz para interagir com os recursos da *Raspberry Pi*. Isso torna mais fácil para os iniciantes em eletrônica e programação de hardware aproveitarem a versatilidade do *Raspberry Pi* em seus projetos.

2.9.2 Biblioteca OpenCV

A biblioteca *OpenCV*, que significa *Open Source Computer Vision Library* (Biblioteca de Visão Computacional de Código Aberto), é uma biblioteca *open source* amplamente utilizada para processamento de imagens e visão computacional. Ela fornece um conjunto de funções e algoritmos que permitem que os desenvolvedores processem imagens e vídeos, detectem objetos, façam análises de imagem e realizem uma variedade de tarefas relacionadas à visão por computador.

A OpenCV foi originalmente desenvolvida pela Intel e, desde então, tornou-se uma das bibliotecas mais populares e respeitadas na comunidade de visão por computador. Ela é escrita em C++ e possui interfaces para várias outras linguagens de programação, incluindo Python, Java e MATLAB, tornando-a acessível para uma ampla gama de desenvolvedores. Ela é uma ferramenta essencial para qualquer pessoa envolvida em projetos de visão por computador, aprendizado de máquina ou processamento de imagens

3 METODOLOGIA

O desenvolvimento da plataforma educacional de robótica móvel requer uma abordagem metódica e bem estruturada para garantir a eficiência e a replicabilidade do projeto. Neste capítulo, será apresentado detalhadamente a metodologia adotada, delineando cada etapa e decisão tomada ao longo do processo. A realização deste trabalho está fundamentada em uma proposta clara, que abrange desde a concepção inicial até a entrega de uma plataforma robusta e acessível financeiramente para fins educacionais e de pesquisa.

Como pilares desse trabalho a acessibilidade e eficácia guiaram as escolhas de materiais, modelagem 3D, fabricação, montagem e o desenvolvimento do software. Cada etapa é cuidadosamente planejada para assegurar que a plataforma atenda ao escopo de um curso de robótica, proporcionando uma experiência prática e educativa. Ao longo dos próximos tópicos será descrito além do “como”, o “porquê” de cada decisão tomada. Essa abordagem transparente não apenas facilita a compreensão do processo, mas também capacita outros interessados a replicarem e

aprimorarem este trabalho.

3.1 PROPOSTA DO TRABALHO

A proposta central deste trabalho é a criação de uma plataforma educacional de robótica móvel que funciona com ROS, caracterizada por sua acessibilidade financeira e capacidade de enfrentar diversos desafios, proporcionando um recurso prático e inovador para aprendizado e experimentação em ambientes acadêmicos, especialmente em salas de aula e atividades de pesquisa na Universidade Federal de Pernambuco (UFPE). A plataforma será desenvolvida com o intuito de ser mais econômica em comparação com modelos comerciais, mantendo, ao mesmo tempo, a qualidade necessária para um curso de robótica eficaz.

Nesse contexto, a seleção de atuadores, sensores e controlador para a plataforma busca simular situações enfrentadas por robôs comerciais, tornando-se uma ferramenta valiosa para a formação em engenharia. Será feito o compartilhamento de modelos CAD, códigos-fonte e documentação dando a possibilidade de replicação e aprimoramento do projeto, enquanto a colaboração com o Professor Dr. Adrien Durand-Petiteville busca enriquecer o aprendizado por meio da criação de material didático explorando as potencialidades da plataforma robótica desenvolvida.

3.2 ESCOPO DO PROJETO

O escopo do projeto foi definido por meio de um mapeamento dos requisitos que o protótipo do robô deverá atender. Isso envolveu uma análise dos materiais e componentes disponíveis, bem como um estudo dos modelos de robôs comerciais e dos softwares mais relevantes na área. O escopo do projeto inclui:

- Posicionamento estratégico de componentes: os motores devem ser posicionados na parte da frente, será utilizada uma *ballcaster* como terceiro ponto de apoio. O alinhamento do eixo das rodas com o frame do Lidar otimizará a sua utilização.
- Sensores de percepção: O Lidar deve ser estrategicamente posicionado em uma altura ideal para detecção de obstáculos, proporcionando um campo de

visão superior a 180 graus. A câmera será posicionada frontalmente, à frente do eixo das rodas, integrada com um servo para um campo livre de visão de 180 graus.

- Manutenção simplificada: a acessibilidade para a manutenção foi considerada, garantindo que cabos e conexões sejam facilmente conectados. Será priorizada a utilização da menor quantidade de fios possível para reduzir o risco de mau contato.
- Modularidade e atualização: a carcaça deve ser projetada em módulos para permitir futuras melhorias, possibilitando a atualização de um módulo específico sem impactar os demais.
- Locomoção e controle: o robô deverá se locomover com base em inputs de velocidade linear e angular desejada, incorporando um sistema de controle em malha fechada a partir do sinal do encoders para assegurar essa funcionalidade.
- Integração em ROS: exemplos práticos de nós em ROS para acessar os *hardwares* gerais do robô, como locomoção, motores, câmera, servo e Lidar, serão disponibilizados. Ademais, também será disponibilizado um exemplo de atuação completa envolvendo o robô como um todo.
- *Software* livre: a opção por softwares livres será feita visando possibilitar melhorias e contribuições da comunidade. A ideia é que o trabalho seja um ponto de partida para futuros aprimoramentos.
- Material de construção: o material escolhido para a construção do protótipo será o PLA (poliácido lático), amplamente utilizado na fabricação de peças 3D e compatível com a impressora disponível no laboratório.
- Testes e verificações: deverão ser realizados testes unitários indoor para cada sistema, garantindo que o robô atenda às expectativas. Em seguida, serão realizados testes completos, envolvendo todos os sensores e atuadores, em um ambiente com obstáculos fixos para simular condições reais.

Com este escopo, busca-se não apenas construir um protótipo funcional, mas também estabelecer as bases para uma plataforma educacional de robótica móvel acessível, inovadora e passível de evolução.

3.4 MATERIAIS UTILIZADOS:

Aqui serão detalhados todos os materiais, componentes e recursos utilizados no desenvolvimento do projeto, bem como a descrição de suas características. Isso não apenas aumenta a transparência do processo, mas também possibilita que outros pesquisadores ou interessados repliquem e ampliem esse estudo.

3.4.1 *Raspberry pi 4 model B*

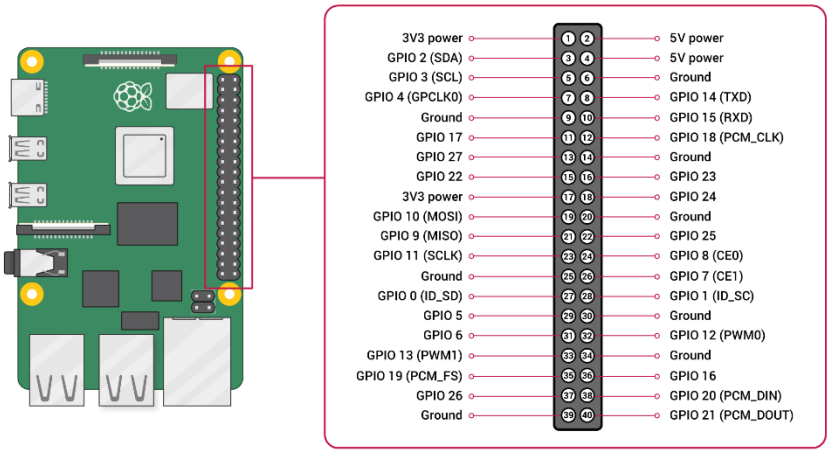
A *Raspberry Pi 4 Model B*, ver Figura 26, é uma escolha altamente recomendada para aplicações em robótica. Este pequeno computador de placa única oferece um equilíbrio ideal entre desempenho, tamanho compacto e eficiência energética. Com um processador quad-core ARM Cortex-A72, 4 GB de RAM e portas USB 3.0, a Raspberry Pi 4 proporciona potência computacional suficiente para lidar com tarefas complexas, como o controle de motores e a interpretação de dados de sensores em tempo real.

Figura 26 –Rasberry pi 4 model b

Fonte: robocore (2023)

A capacidade de conectividade é outra vantagem significativa, com a presença de portas HDMI duplas e suporte a *Ethernet Gigabit*, proporcionando flexibilidade na comunicação e visualização de dados. A possibilidade de expansão através dos pinos *General Purpose Input/Output* (GPIO) permite a conexão direta a uma variedade de sensores e atuadores, tornando-a altamente adaptável a diferentes necessidades de projeto. A Figura 27 apresenta os pinos GPIO da *raspberry pi 4*.

Figura 27 – Pinos GPIO *raspberry pi 4 model b*



Fonte: raspberrypi (2023)

Essa placa suporta o sistema operacional *ubuntu mate*, apresentado anteriormente nesse trabalho, que possui uma interface amigável e é semelhante aos sistemas operacionais utilizados em desktops. Além disso, a *Raspberry Pi 4* é conhecida por suportar linguagens de programação de alto nível, como Python, facilitando o desenvolvimento de software. Sua popularidade na comunidade de desenvolvedores garante um amplo suporte técnico, com uma vasta quantidade de recursos e tutoriais disponíveis online.

Essas características fazem do *Raspberry Pi 4* uma escolha excepcional como placa de controle para o projeto, proporcionando uma experiência de utilização mais acessível, especialmente para aqueles usuários menos familiarizados com ambientes mais técnicos, tornando-o uma opção ideal para um amplo público.

3.4.2 Locomoção

O sistema da locomoção é composto por rodas ligadas a motores de DC de 6 volts acoplados com um sensor de rotação e uma redução, Mancal para o motor, e rodas de 67 mm de diâmetro, ver Figura 29 .O kit apresentado na Figura 28 facilita a montagem do robô e apresenta uma boa relação custo-benefício. O sensor de rotação será o utilizado para montar o controle de velocidade. A Tabela 3 mostra as principais características do motor e a Figura 30 apresenta os pinos de conexão.

Figura 28 – Componentes da locomoção

Fonte: baú da eletrônica (2023)

Figura 29 – Medidas locomoção

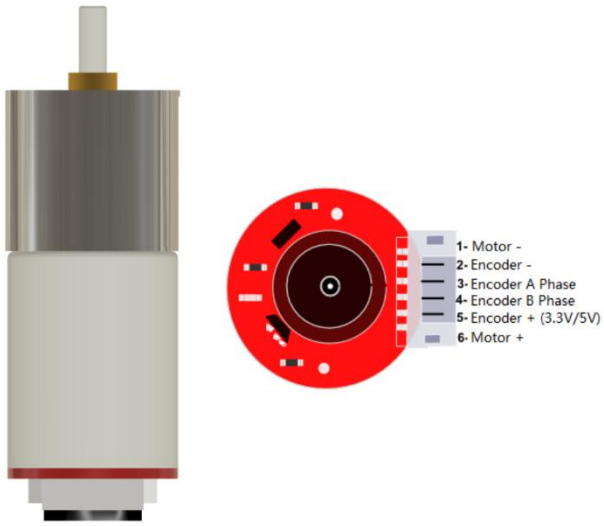
Fonte: O autor (2023)

Tabela 3 – Características do motor da locomoção

Tensão máxima no motor	6V
Velocidade máxima do motor	130 RPM
Razão de redução	46

Fonte: O autor (2023).

Figura 30 – Pinos de conexão motor da locomoção



Fonte: O autor (2023)

O motor possui um conjunto de fios coloridos para realizar a conexão, eles seguem o padrão da Tabela 4.

Tabela 4 – Descrição dos conectores do motor

Vermelho	Fonte de alimentação positiva do motor (+)
Branco	Fonte de alimentação negativa motor (-)
Amarelo	Sinal feedback (uma volta do motor tem 11 sinais)
Verde	Sinal feedback (uma volta do motor tem 11 sinais)
Azul	Fonte de alimentação positiva do codificador (+)
Preto	Fonte de alimentação positiva do codificador (-)

Fonte: O autor (2023).

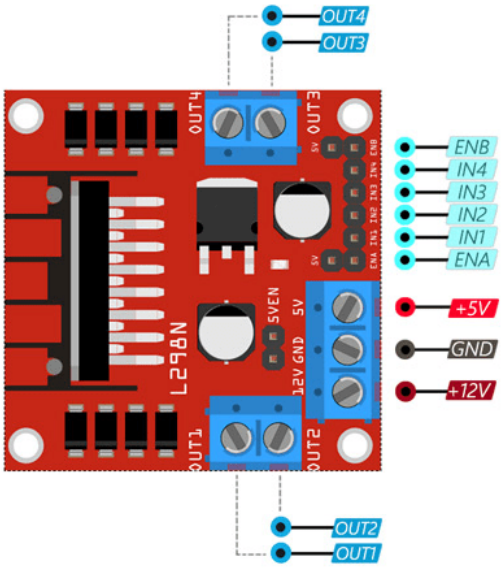
3.4.3 Driver ponte H

Para o acionamento dos motores da locomoção foi selecionado o chip L298N que é um dispositivo integrado fabricado pela STMicroelectronics e é composto principalmente por 2 pontes H, cada uma delas possibilitando o controle de um motor elétrico DC, permitindo movimentação em ambas as direções. Esse componente permite o controle direto de dois motores elétricos, por meio de controles lógicos de baixa potência. O L298N, ver Figura 31, necessita de duas fontes de energia independentes para funcionar, uma tensão destinada à seção de

potência, que será empregada para fornecer energia aos motores por meio de transistores de potência e uma tensão destinada à seção de controle, que será utilizada para alimentar toda a lógica de controle, incluindo os mencionados transistores de potência.

Os pinos de controle da ponte (ENA e ENB) possibilitam a ativação ou desativação dos motores. Importante notar que essas entradas podem ser alimentadas de forma binária (ligado ou desligado, resultando em rotação máxima dos motores) ou por meio de sinal PWM, o que permite controlar a velocidade de rotação. Os pinos de seleção da ponte (IN1, IN2, IN3 e IN4) possibilitam a escolha da direção de rotação dos motores. Como estamos lidando com o controle de dois motores, há quatro entradas disponíveis, correspondentes às quatro possíveis direções de rotação: motor direito para frente, motor direito para trás, motor esquerdo para frente e motor esquerdo para trás. A Tabela 5 descreve as conexões do L298n.

Figura 31 – L298N



Fonte: hibit (2023)

Tabela 5 – Conexões L298n

ID	Nome	Descrição
1	VCC	Fonte de alimentação positiva da placa (+)
2	GND	Fonte de alimentação negativa da placa (-)
3	5V	Saída de 5V
4	ENA	PWM do motor A
5	IN1	Sentido de rotação do motor A
5	IN2	Sentido de rotação do motor A
6	IN3	Sentido de rotação do motor B
6	IN4	Sentido de rotação do motor B
7	ENB	PWM do motor B
8	OUT1	Alimentação negativa do motor A (-)
8	OUT2	Alimentação negativa do motor A (-)
9	OUT3	Alimentação negativa do motor B (-)
9	OUT4	Alimentação negativa do motor B (-)

Fonte: – O autor (2023)

3.4.4 Lidar

Este sensor permite a captura de uma nuvem de pontos que mede a distância do sensor a obstáculos. Esta nuvem permite o robô fazer o mapeamento de ambientes internos e assim auxilia a navegação, localização e identificação de obstáculos. No presente trabalho será utilizado o Ydlidar G2, ver Figura 32.

Figura 32– Ydlidar G2

Fonte: Ydlidar.com (2023)

A Tabela 6 mostra as características do lidar.

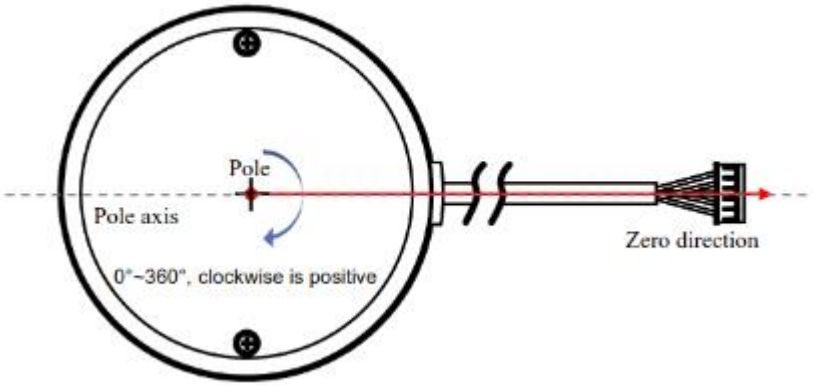
Tabela 6 – Características do Lidar G2

Características	Descrição
Frequência de medição	5000HZ
Frequência do motor	5 a 12HZ
Distância de medição	0,12 a 16m
Campo de visão	0 a 360 graus
Erro sistemático	0,02 m
Faixa de intensidade luminosa	0 a 1023
Resolução angular	0,36 a 0,864 graus

Fonte: O autor (2023).

O lidar G2 define um sistema de coordenadas polares. As coordenadas polares do sistema consideram o centro do núcleo rotativo de G2 como o polo, e o ângulo especificado é positivo no sentido horário (vista superior) como exibe a Figura 33. O ângulo zero está localizado na direção da saída da linha de interface G2 PH2.0-5P.

Figura 33 – Polo do lidar



Fonte: Ydlidar.com(2023)

3.4.5 Raspberry Pi Câmera 2

Câmera da Raspberry, ver Figura 34, é de ótima qualidade e compacta, possui foco automático ultrarrápido como o seu padrão, além disso conta com uma rica biblioteca de comandos de software já desenvolvidas, isso significa que é possível ajustar todo os parâmetros e ter controle total sobre o que focar e como fazê-lo. Ela apresenta um campo de visão amplo de 120º que vai ajudar no reconhecimento do ambiente. A Tabela 7 apresenta as principais características do câmera.

Figura 34 – *Raspberry Pi Camera* Módulo 2

Fonte: *Raspyberrypi*(2023)

Tabela 7 – Características *Raspberry Pi Camera* Módulo 2

Características	Valor
Tamanho	25x24x9 mm
Massa	3 g
Resolução	8 Megapixels
Sensor	Sony IMX219
Resolução do sensor	3282 x 2464 pixels
Área de imagem do sensor	3.68 x 2.76 mm (diagonal de 4.6 mm)
Tamanho do pixel	1.12 µm x 1.12 µm
Tamanho óptico	¼”
Foco	Ajustável
Profundidade de campo	10 cm – Inf
Distância focal	3.04 mm
Campo de visão horizontal	62.2. graus
Campo de visão vertical	48.8. graus
Relação focal	F2.0
Tempo máximo e exposição	11.76

Fonte: O autor (2023).

3.4.6 Servo Motor

Para ampliar o campo de percepção do robô, esse projeto está equipado com um servomotor TowerPro MG996R, ver Figura 35, que permite orientar a câmera. Estas características e a descrição dos fios são dados nas Tabelas 8 e 9. Os servos são controlados pela largura do pulso, a largura do pulso determina o ângulo. Uma largura de pulso de 1500 µs move o servo para o ângulo 0. Cada aumento de 10 µs

na largura de pulso normalmente move o servo 1 grau mais no sentido horário. Cada diminuição de 10 µs na largura de pulso normalmente move o servo 1 grau mais no sentido anti-horário. Servos normalmente recebem 50 pulsos por segundo (50Hz).

Figura 35 – Servo motor MG 996R

Fonte: Robocore (2023)

Tabela 8 – Características do Servo mg 996r

Tensão de operação	4,8 a 6,0V
Tipo de frenagem	Metálica
Modulação	Digital
Velocidade de operação	0,19 seg/60 graus (4,8 V sem carga
Velocidade de operação	0,15 seg/60 graus (6 V sem carga)
Torque (stall)	9,4 kgf.cm (4,8 V) e 11,0 kgf.cm (6 V)

Fonte: O autor (2023).

Tabela 9 – Conexões servo mg 996r

Marrom	Fonte de alimentação negativa do motor(-) (0 V)
Vermelho	Fonte de alimentação positiva do motor(+) (5 V)
Amarelo	Sinal de controle PWM

Fonte: O autor (2023).

3.4.7 *Ballcaster*

O *ballcaster*, ver Figura 36, é um componente mecânico utilizado na robótica e outras aplicações que requerem mobilidade e facilidade de movimentação em diferentes direções. É constituída por uma esfera de material resistente, como plástico ou metal, que gira em torno de um eixo central. Ela servirá como o terceiro ponto de contato com o chão do robô.

Figura 36– *Ballcaster*

Fonte: polulu (2023)

3.4.8 *Power bank*

O Carregador Portátil i2GO, ver Figura 37, com 10000mAh, destaca-se como a opção ideal para fornecer energia a um protótipo de robô móvel. Oferece uma fonte estável e eficiente de fornecimento de energia. Sendo pequeno, leve e de ampla compatibilidade com dispositivos, torna ele perfeito para acompanhar o robô como uma escolha confiável para alimentação

Figura 37 – *Powerbank*

Fonte: I2go(2023)

3.5 CIRCUITO E CONEXÃO DE COMPONENTES ELETRÔNICOS

O desenvolvimento do robô abrange não apenas a sua estrutura física, mas também a integração dos componentes eletrônicos que serão utilizados. Antes de iniciar a modelagem da estrutura do robô, é de suma importância compreender a conexão e a disposição desses componentes. Isso implica entender quais fios e cabos serão utilizados, estabelecer conexões precisas entre cada elemento (como bateria, motores, sensores e placas de controle) e definir a estratégia de alimentação elétrica. Esse esquema de conexão não apenas simplifica o processo de modelagem 3D, mas também resulta em inúmeros benefícios.

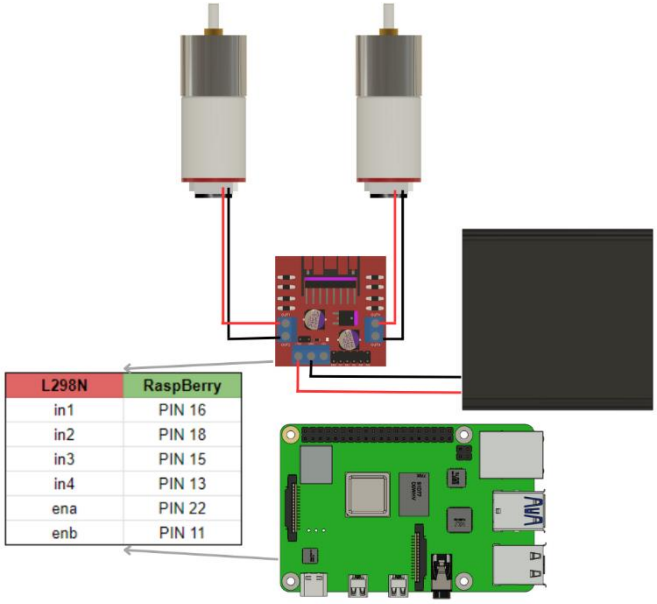
Um dos principais benefícios de uma abordagem centrada na conexão dos componentes eletrônicos é a otimização do espaço disponível na estrutura do robô. Ao planejar com antecedência como os fios, cabos e componentes serão dispostos, é possível aproveitar ao máximo o espaço interno, garantindo que cada sensor e atuador esteja posicionado de forma eficiente. Além disso, a consideração de

espaços para a passagem de fios é fundamental para evitar problemas futuros relacionados à organização e à manutenção.

Outro benefício significativo desse planejamento é a capacidade de aproveitar 100% da capacidade de sensores e atuadores. Ao entender como os componentes se relacionam eletricamente, é possível evitar conflitos de conexão e garantir que todos os recursos de hardware sejam plenamente utilizados. Isso contribui para o desempenho ideal do robô e a maximização de suas funcionalidades.

A Figura 38 mostra as conexões dos componentes eletrônicos para o sistema de locomoção.

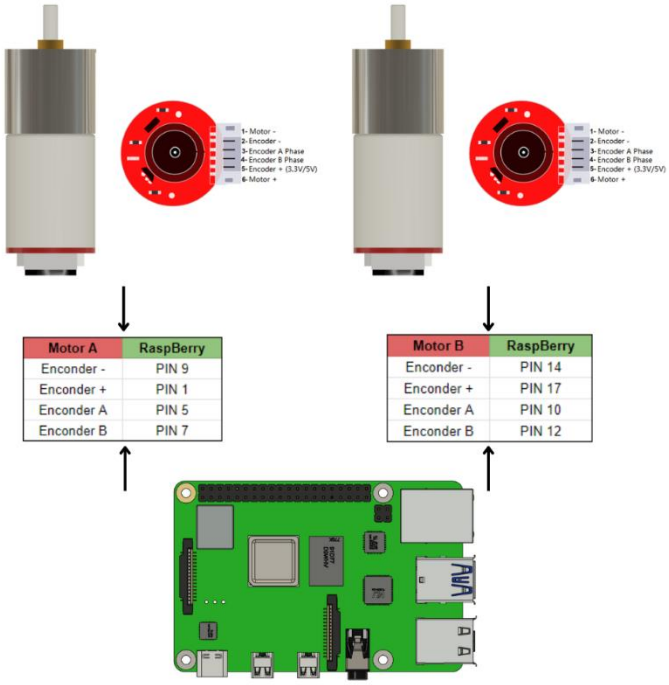
Figura 38 – Conexão locomoção



Fonte: O autor (2023)

A Figura 39 representa a conexão com o encoder de cada motor DC com a *Raspberry*.

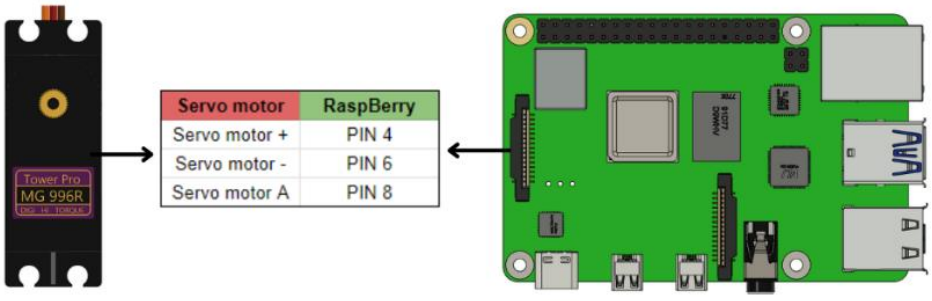
Figura 39 – Conexão Encoders locomoção



Fonte: O autor (2023)

A Figura 40 representa a conexão com o servo motor que movimenta a câmera com a *Raspberry*.

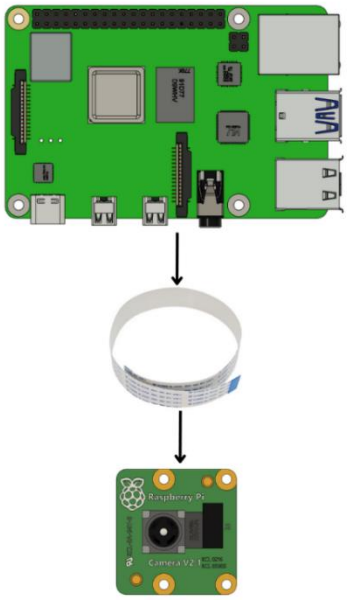
Figura 40 – Conexão servo motor



Fonte: O autor (2023)

A Figura 41 representa a conexão a câmera com a *Raspberry*.

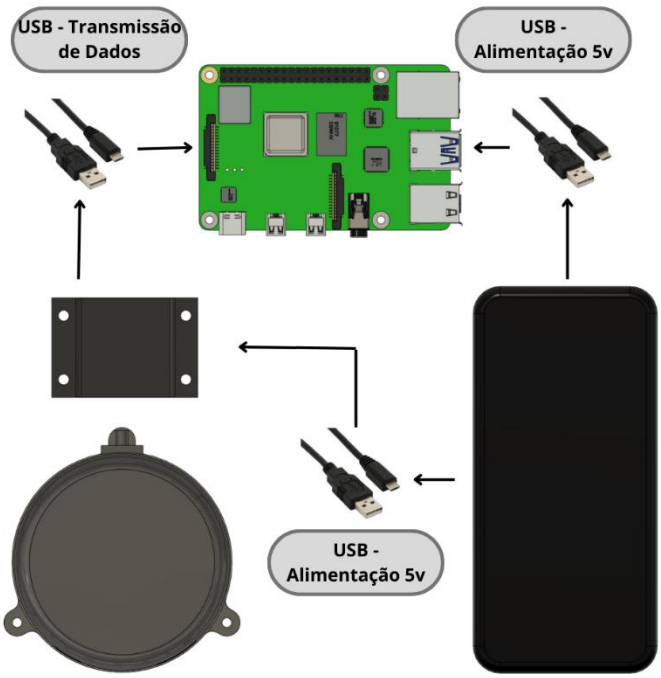
Figura 41 – Conexão câmera



Fonte: O Autor(2023)

A Figura 42 representa a conexão do lidar, alimentação e *Raspberry*.

Figura 42 – Conexão Lidar com raspberry e alimentação



Fonte: O autor(2023)

A sobreposição de todos os sistemas compõe o sistema completo do protótipo.

3.6 MODELAGEM 3D

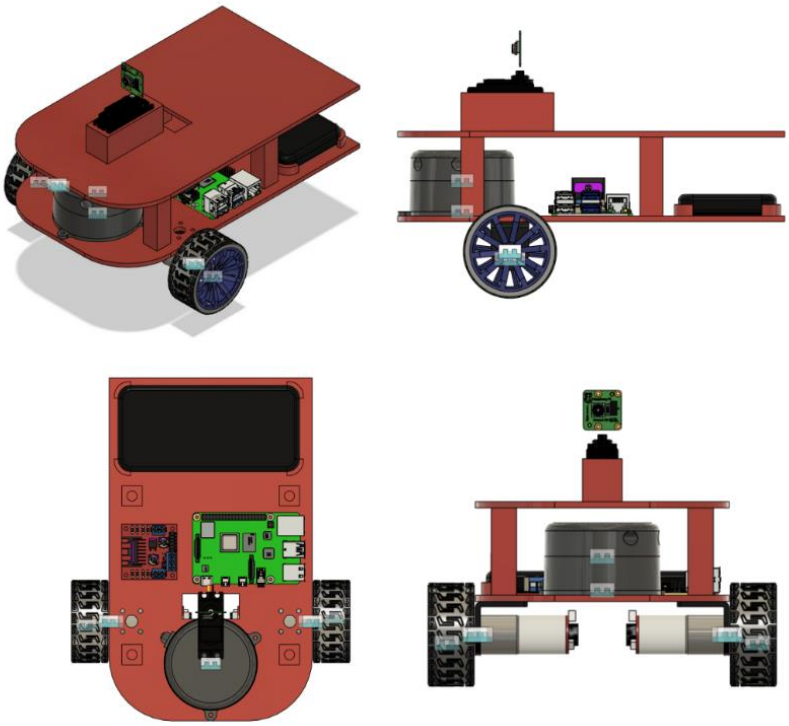
Na etapa de modelagem 3D do protótipo, uma pesquisa aprofundada foi realizada em relação a diversos robôs comerciais disponíveis no mercado que estão aproximadamente dentro do escopo do projeto. Isso permitiu compreender os padrões de design e as melhores práticas em relação à estrutura e posicionamento de componentes de robôs semelhantes. Com esse conhecimento, a modelagem do protótipo foi conduzida com o auxílio do *software* Fusion 360. Durante o processo de modelagem, foram desenvolvidas cerca de 9 diferentes versões do protótipo de maneira iterativa, as quais evoluíram à medida que *feedback*, testes e provas de conceitos eram conduzidos. Essas iterações resultaram em melhorias significativas no design, garantindo que o protótipo atendesse de maneira eficaz aos requisitos estabelecidos para o projeto.

Nos próximos tópicos serão apresentadas três diferentes versões que carregam consigo as evoluções mais significativas das rodadas de iteração do projeto. Bem como a descrição e justificativa do design da estrutura e posicionamento de componentes.

3.6.1 Versão 1

A Figura 43 Ilustra a visão geral da primeira versão do modelo 3D

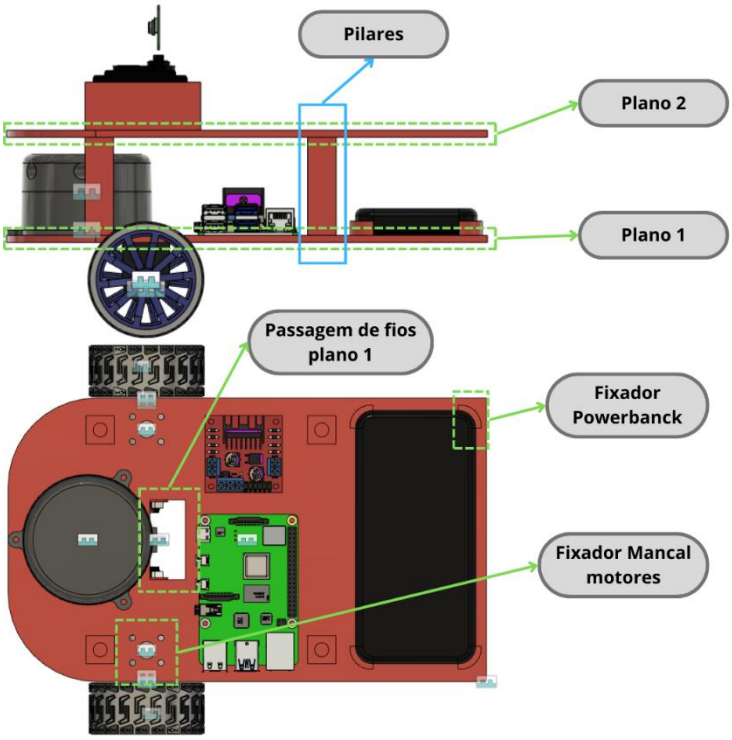
Figura 43 – Visão geral modelo 1



Fonte: O autor(2023)

Essa versão apresenta uma proposta de divisão de construção em dois planos, ver Figura 44, sendo o primeiro responsável suportar na sua parte superior o *powerbank*, *raspberry pi 4*, lidar e o *driver* de ponte h. Esse plano apresenta um rasgo retangular para passagem de fios até os motores, que estão fixados pelos mancais na superfície abaixo do primeiro plano. Ele também conta com estruturas para manter o posicionamento o *powerbank* fixo em uma determinada posição.

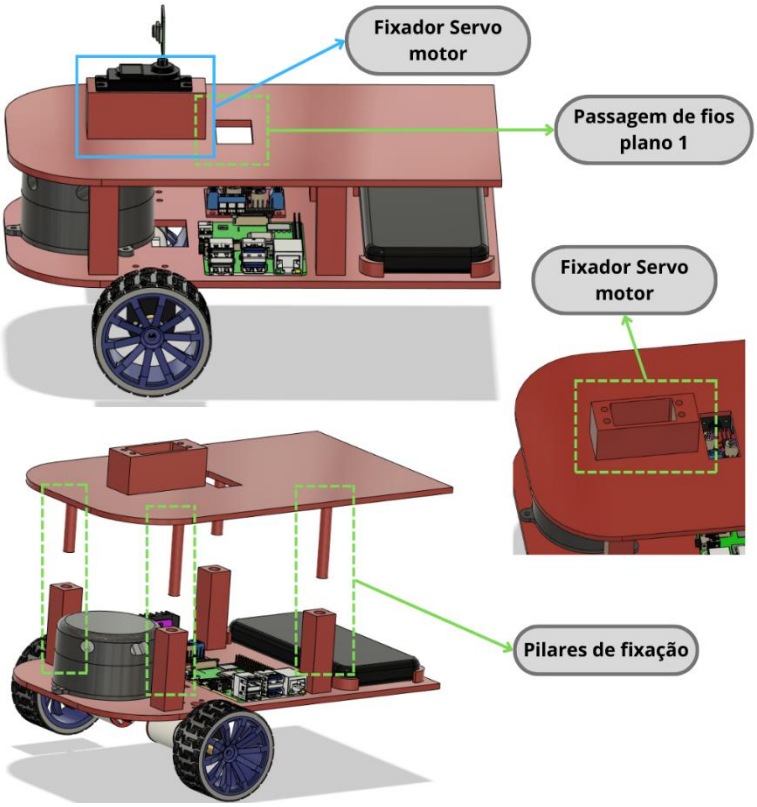
Figura 44 – Planos do modelo 1



Fonte: O autor (2023)

No segundo plano estão dispostos o servo motor e a câmera. Ele possui um rasgo retangular para passagem de fios que vão conectar o servo e a câmera na *raspberry*. O servo motor está fixado em uma “caixa” com um rasgo para saída de fios na parte inferior. Os dois planos estão unidos por quatro estruturas de conexão. A Figura 45 enfatiza esses detalhes do texto.

Figura 45 – Fixadores modelo 1



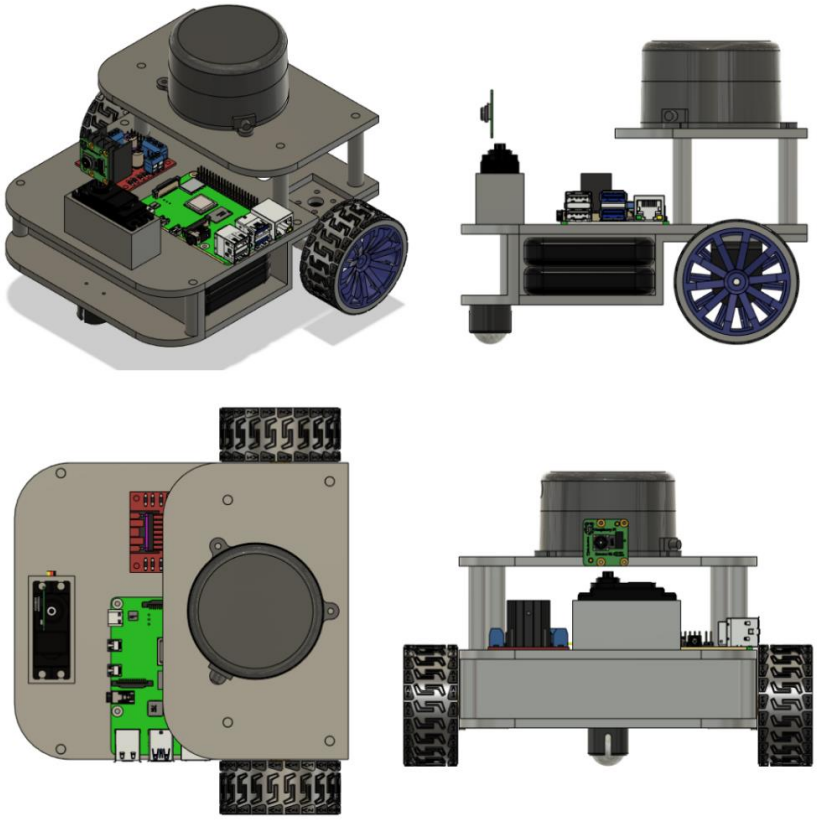
Fonte: O autor (2023)

Após algumas análises notou-se que o lidar não estava em uma posição ideal, pois estava a frente do eixo das rodas e por conta das estruturas de conexão entre os planos haveria uma limitação do range de captação do Lidar, o modelo era maior em área do que as expectativas e o centro de massa estava distante do chão.

3.6.2 Versão 2

A Figura 46 exibe a visão geral da versão dois do modelo 3D.

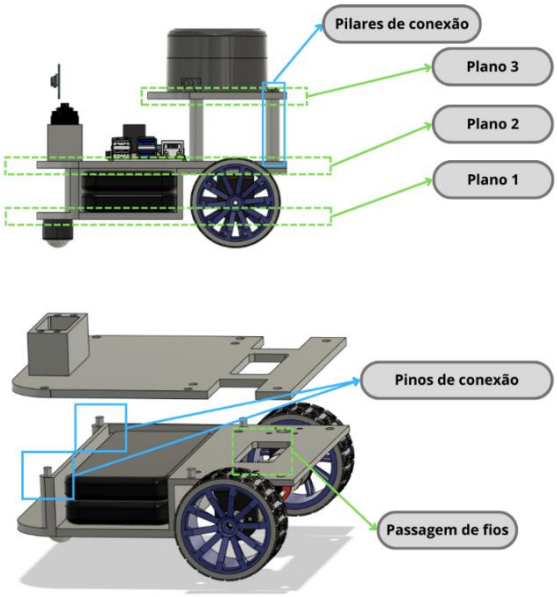
Figura 46 – Visão geral modelo 2



Fonte: O autor (2023)

Ele apresenta uma proposta de divisão de construção em três planos, sendo o primeiro responsável por guardar os dois *powerbanks* os motores e a *ballcaster*. A lateral onde estão os *powerbanks* seria aberta para a passagem dos cabos de alimentação. Esse plano possui pinos cilíndricos usados para realizar a conexão com segundo plano, como mostra a Figura 47.

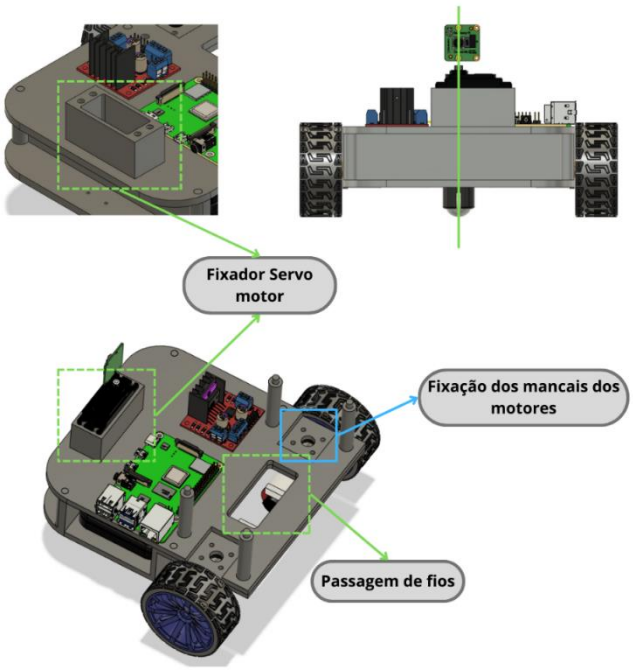
Figura 47– Planos modelo 2



Fonte: O autor (2023)

No segundo plano estão a *raspberry* pi 4, driver de ponte h, o servo motor e a câmera. Semelhante a versão 1 esse plano apresenta um rasgo retangular para passagem de fios até os motores, que ficam no primeiro plano. Também foi mantida a estratégia de fixação do servo motor utilizando uma (caixa) tendo o cuidado de manter o seu eixo na linha central do robô, ver Figura 48.

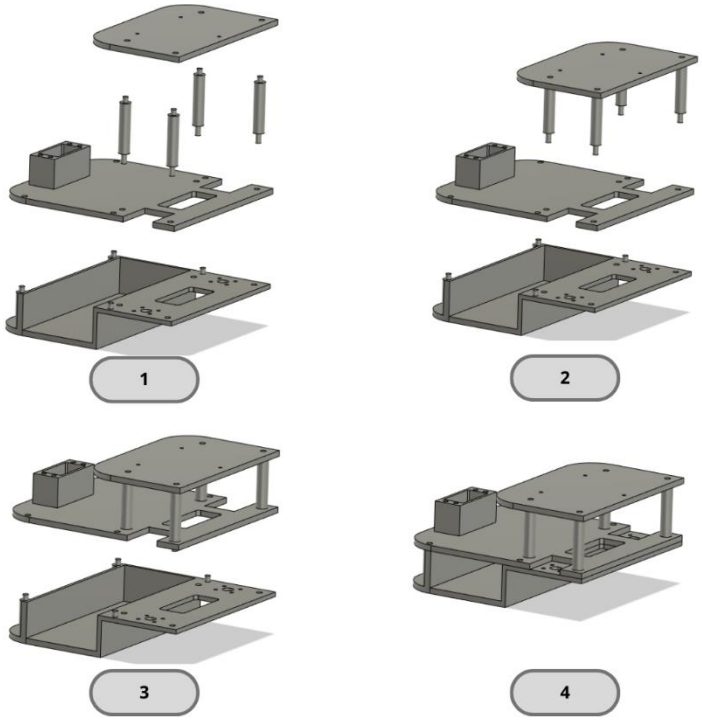
Figura 48 – Fixadores modelo 2



Fonte: O autor (2023)

No terceiro plano temos o lidar, posicionado de modo que consiga captar dados em um range de 360º. Os planos dois e três estão unidos por estruturas de conexão cilíndrica, ver Figura 49. A imagem abaixo enfatiza esses detalhes do texto.

Figura 49 – Montagem estrutura modelo 2



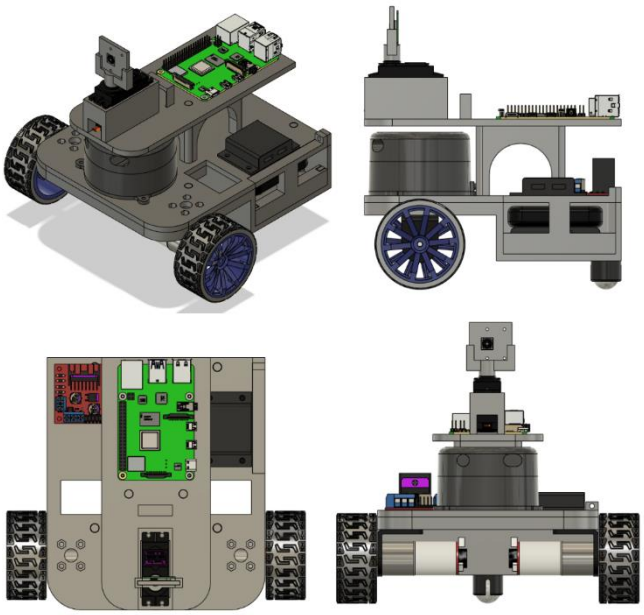
Fonte: O autor (2023)

Após algumas análises e conversas com o orientador notou-se que o lidar estava posicionado em uma altura elevada, o que tornaria difícil perceber obstáculos menores. O modelo era menor que a versão 1 em área e o centro de massa estava mais próximo do chão. Outro ponto levantado durante a coleta de feedback estava relacionado ao posicionamento da câmera que poderia ser mais bem aproveitada se fosse posicionada em um local mais elevado. Além disso, deveria inverter a frente do robô deixando a câmera mais próxima do eixo das rodas.

3.6.3 Versão 3

A Figura 50 mostra a visão geral da versão três do modelo do modelo 3D.

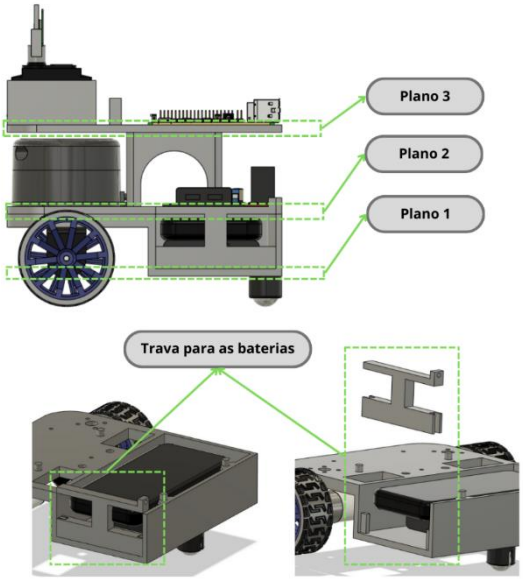
Figura 50 – Visão geral modelo 3



Fonte: O autor (2023)

Ele apresenta uma proposta de divisão de construção em três planos, ver Figura 51, sendo o primeiro responsável por suportar os dois *powerbanks*, motores e a *ballcaster*, semelhante a versão 2. Buscando uma otimização de espaço foram removidas algumas estruturas diminuindo a área do robô. Na lateral onde estão posicionadas os *powerbanks* foi modelado um sistema que fecha a abertura, mantendo os componentes no seu interior fixos impedindo que eles caiam com o movimento do robô e permitindo a passagem dos cabos de alimentação.

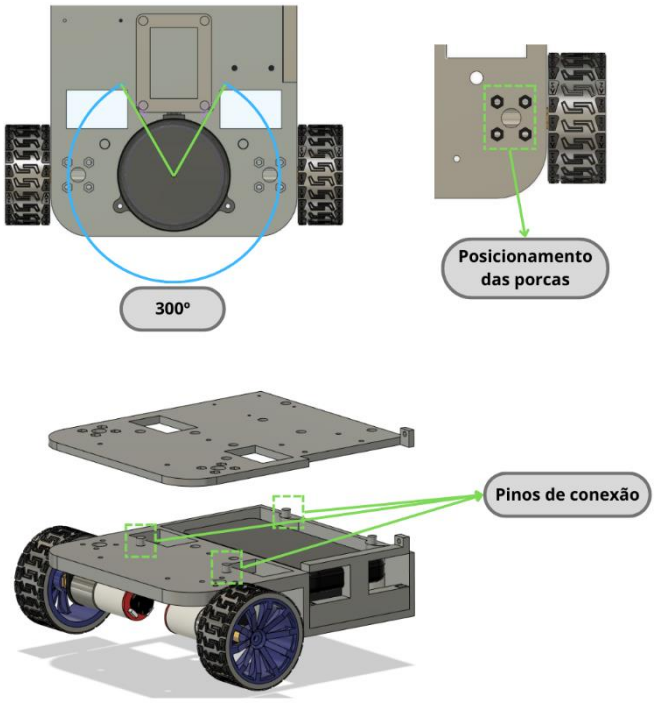
Figura 51 – Planos modelo 3



Fonte: O autor (2023)

Logo acima, no segundo plano estão, driver de ponte h e o lidar posicionado mais próximo ao chão do que na versão 2 possibilitando a identificação de objetos menores, ele também consegue captar dados em um range de 300°, ver Figura 52, o suficiente para o objetivo do protótipo. O plano 1 e o plano dois estão conectados por pinos cilíndricos semelhantes à versão 2 e ambos apresentam dois rasgos retangulares para passagem de fios até os motores, que ficam no primeiro plano. O plano dois tem rasgos hexagonais para posicionar as porcas que são utilizadas para fixar os mancais dos motores.

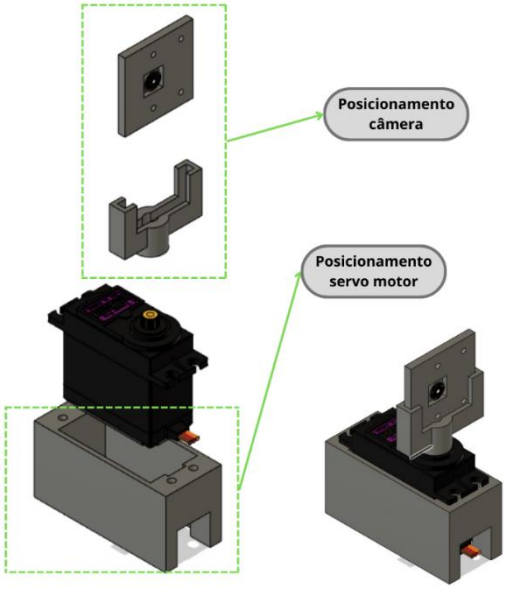
Figura 52 – Fixação modelo 3



Fonte: O autor (2023)

No terceiro plano temos o servo motor, câmera e a *raspberry*. A câmera está posicionada em uma localização mais elevada que na versão dois, possibilitando uma maior captação de informações. Foram modeladas estruturas para sua sustentação e conexão com o servo motor, a Figura 53 exibe essas peças.

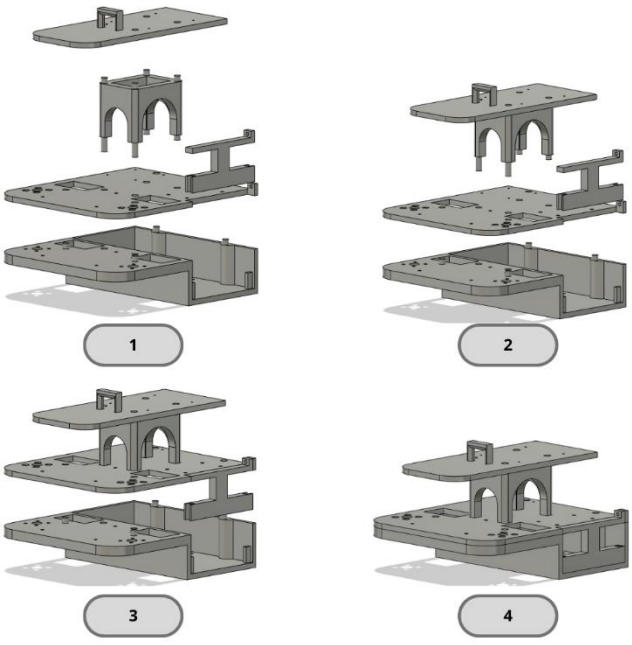
Figura 53 – Sistema da câmera modelo 3



Fonte: O autor (2023)

Entre os planos 2 e 3 foi utilizado uma estrutura de conexão com arcos, ela apresenta uma maior estabilidade quando comparada com as estruturas cilíndricas da versão dois e ainda permite a passagem de fios pelo seu interior. A Figura 54 apresenta os detalhes do modelo citados anteriormente no texto

Figura 54 – Montagem estrutura modelo 3



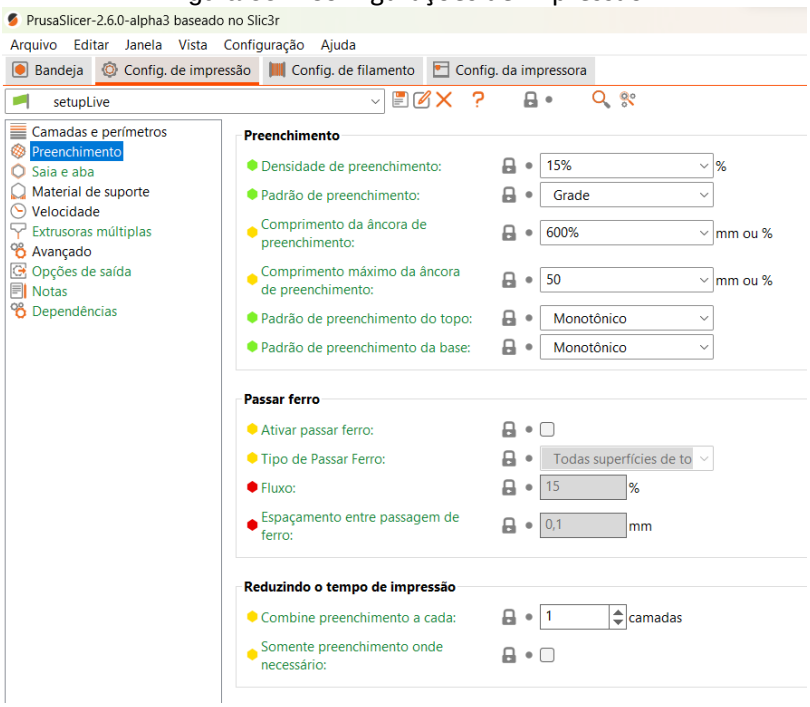
Fonte: O autor (2023)

3.7 FABRICAÇÃO

O método escolhido para produzir o protótipo foi o de manufatura aditiva e o material utilizado foi o PLA. Ao longo da etapa de modelagem, algumas peças foram fabricadas para validar algumas hipóteses e premissas relacionadas a usabilidade, funcionalidade de componentes, estabilidade e o processo de montagem. Com os modelos físicos foi possível identificar defeitos e oportunidades de melhorias que não seria possível notar apenas com o modelo 3D no software. As correções foram aplicadas de maneira iterativa ao projeto.

Antes de realizar a impressão completa do modelo final, também foram feitos testes de impressão para fazer ajustes de dimensionalidade do protótipo em geometrias críticas. Devido a peculiaridade de cada modelo de impressora 3D, filamento e as configurações escolhidas no software de fatiamento as dimensões definidas em software sofrem alteração no resultado da impressão. Para superar esses desvios dimensionais as geometrias críticas foram isoladas, impressas e após análises, o modelo 3D foi corrigido e finalizado para a impressão completa. A Figura 55 mostra algumas das configurações selecionadas no software de fatiamento.

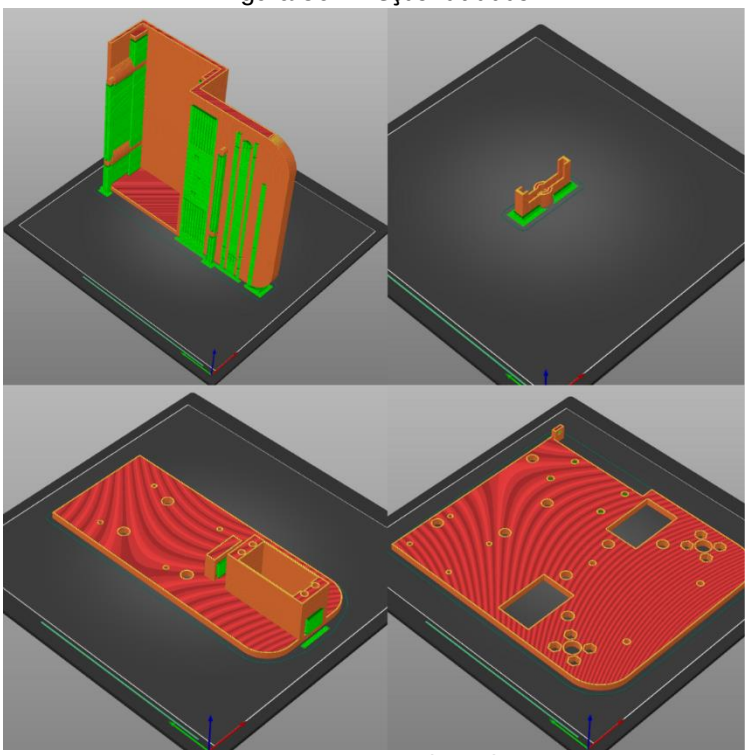
Figura 55 – Configurações de impressão



Fonte: O autor (2023)

Com os ajustes de dimensionalidade feitos os componentes 3D foram exportados para o formato STL, importados no software de fatiamento, ver Figura 56, responsável por gerar o Gcode utilizado na impressora 3D, com as mesmas configurações aplicadas nos testes. Na imagem abaixo é possível ver algumas das peças com seus suportes.

Figura 56 – Peças fatiadas



Fonte: O autor (2023)

Para produzir o protótipo foram impressas um total de 7 peças com duração total de 33h36min. As informações detalhadas com duração e massa de cada peça podem ser vistas na Tabela 10.

Tabela 10 – Tempos de impressão e massa das peças

Nome	Tempo de impressão	Massa (g)
Base 1	16h50min	169,65
Base 2	6h40min	72,63
Base 3	5h16min	55,71
Tampa bateria	1h20min	10,35
Conexão base2-3	3h10min	26,25
Suporte câmera	20min	6,22

Fonte: O autor (2023).

Ao todo foram fabricados 4 modelos que estão sendo utilizados na disciplina programação robótica oferecida pelo departamento de engenharia mecânica da UFPE.

3.8 MONTAGEM

Um dos requisitos básicos do projeto é ter uma montagem simples e que permita a evolução dele trocando os módulos. No tópico anterior, que apresenta a modelagem 3D, são enfatizadas as peças utilizadas para realizar as conexões entre os componentes. Essas em sua maioria são de encaixe, diminuindo a utilização de parafusos, mas esses ainda são utilizados na fixação dos motores, lidar e placas eletrônicas.

3.9 CONFIGURAÇÃO DO AMBIENTE DE SOFTWARE

O processo para configurar o ambiente de software na *Raspberry Pi* é simples, mas requer algumas atenções. Com objetivo de facilitar a replicação de projeto serão apresentadas as etapas necessárias, junto com o *link* de acesso aos *sites* para realizar a instalação.

1. Instalação do *Raspberry Pi Imager,* ele será utilizado para instalar a imagem do sistema operacional ubuntu mate no cartão de memória da *raspberry pi 4*. O processo de instalação do *Raspberry Pi Imager,* pode ser encontrado em *Raspberry Pi Imager*.
2. Com o *Raspberry Pi Imager* instalado, agora é necessário fazer o *download* da imagem do *ubuntu mate* para *raspberry pi 4.* Ela pode ser encontrada em *ubuntu mate*. Caso não ache a versão desejada do ubuntu mate, você pode acessar as releases antigas aqui. Utilize o *Raspberry Pi Imager* para salvar a imagem no cartão de memória da imagem.
3. Agora que a imagem já está no cartão de memória, coloque-o na *raspberry pi 4* e instale. Conecte os periféricos necessários como: cabo hdmi para o monitor, teclado e *mouse.* Ligue a *Raspberry* à energia e siga as etapas de configuração.

4. Com a *Raspberry* ligada e com o ubuntu mate, será necessário instalar o ROS. Neste trabalho estamos utilizando o ROS noetic. O site do ROS traz o tutorial para instalação aqui, ROS tutorial.
5. Com o ROS instalado, será necessário instalar as bibliotecas apresentadas anteriormente. A RPI, seguindo os passos do seguinte link e a Opencv, seguindo as etapas deste tutorial.
6. Para utilizar o lidar, será necessário fazer o *download* do *Software development kit* SDK do modelo G2 no site ydlidar g2.

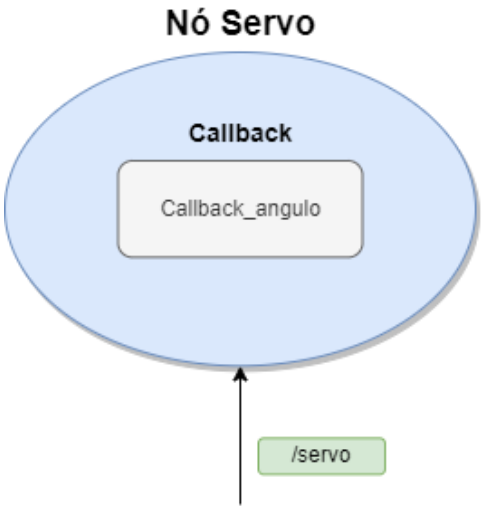
3.10 DRIVERS FORNECIDOS

Para facilitar a utilização do projeto foram desenvolvidos *drivers* para acessar e controlar os hardwares utilizados. Esses *drivers* serão entregues como nós do ROS, cada um com sua respectiva funcionalidade. São eles o servo motor, *camera*, *encoders*, *Controller base* e lidar. As suas características individuais serão apresentadas nos próximos tópicos.

3.10.1 Servo motor

O nó do servo motor é responsável por controlar a atuação do servo. Essa função é importante para aumentar o campo de visão do robô melhorando sua capacidade de superar desafios. Esse nó recebe uma mensagem do tipo Int32(), do módulo de mensagens padrão do ROS, que contém a angulação que o servo deve assumir. O range da mensagem vai de -90 até 90. Essa mensagem percorre o tópico */servo.* A Figura 57 ilustra o nó e o tópico.

Figura 57 – Nó servo_motor

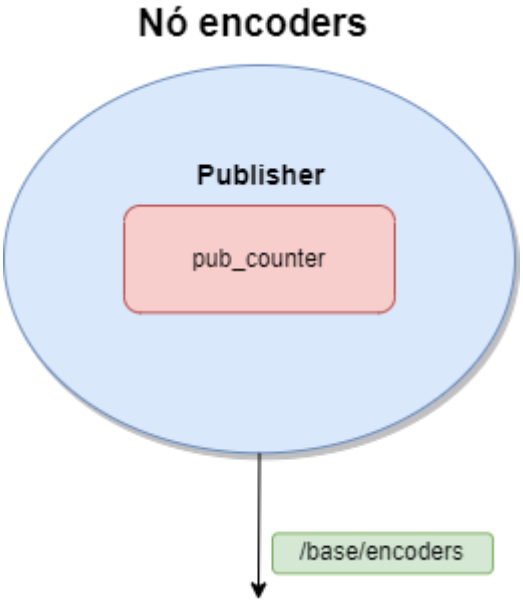


Fonte: O autor (2023)

3.10.2 *Encoder*

O nó *encoders* é responsável por computar o sinal do *encoders* em cada motor por meio de interrupções. Esse processamento é importante para medir a velocidade de cada roda. A cada acionamento do *encoder* é enviado pelo tópico */base/encoders* uma mensagem em formato de *array*, de tamanho dois, com a contagem de acionamentos de cada *econder*. A Figura 58 exibe o tópico e o nó.

Figura 58 – Nó *encoders*

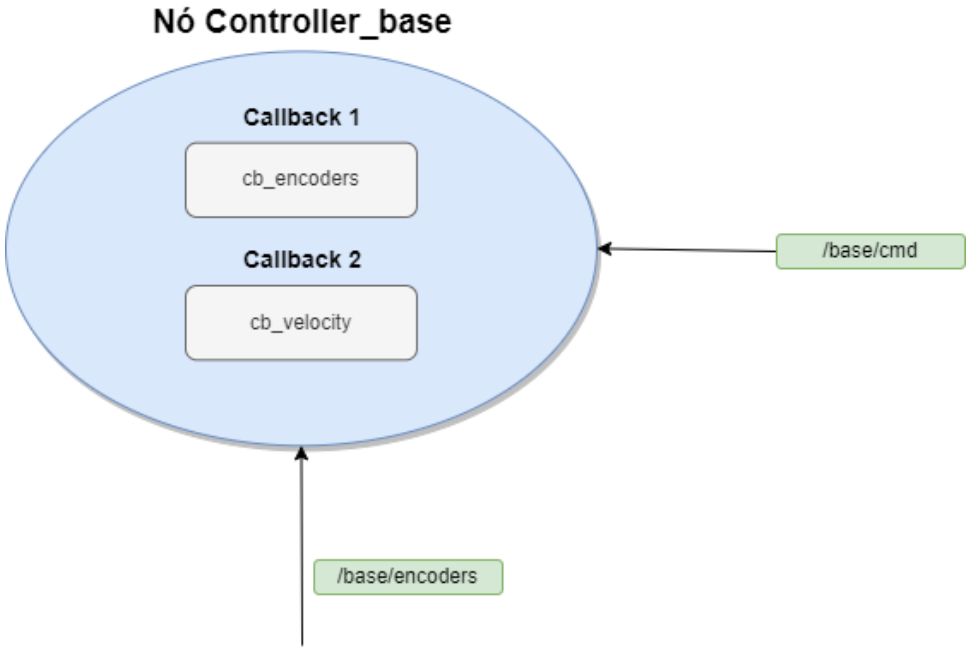


Fonte: O autor (2023)

3.10.3 *Controller base*

O nó de Controller base desempenha a função de controlar o acionamento dos motores. Ele recebe a mensagem que contém a velocidade linear e angular desejada no formato *Twist* do módulo de mensagens *geometry_msgs.msg* pelo tópico */base/cmd*. Ele também recebe pelo tópico */base/encoders* a mensagem com a quantidade de acionamentos de cada *encoder.* Utiliza essas informações para calcular a velocidade de cada roda e aplicar técnicas de controle para atingir e manter a velocidade desejada. A Figura 59 apresenta o nó e o tópico.

Figura 59 – Nó *Controller_base*

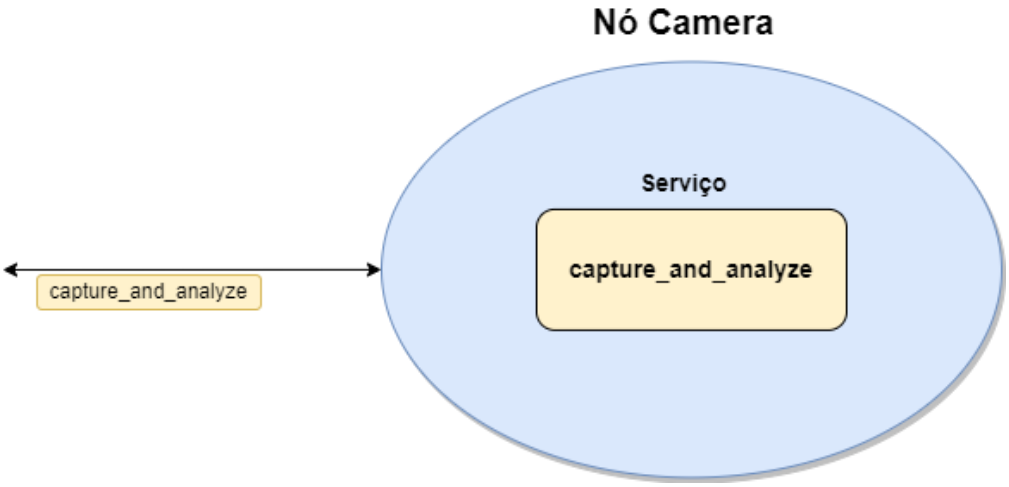


Fonte: O autor (2023)

3.10.4 Câmera

O nó Câmera foi estruturado como um serviço, responsável por acionar a câmera raspberry, capturar uma imagem e realizar um processamento para identificar se é possível avançar em determinada direção. Esse nó, ilustrado na figura 60, retorna a resposta do processamento para o nó cliente.

Figura 60 – Nó *camera*

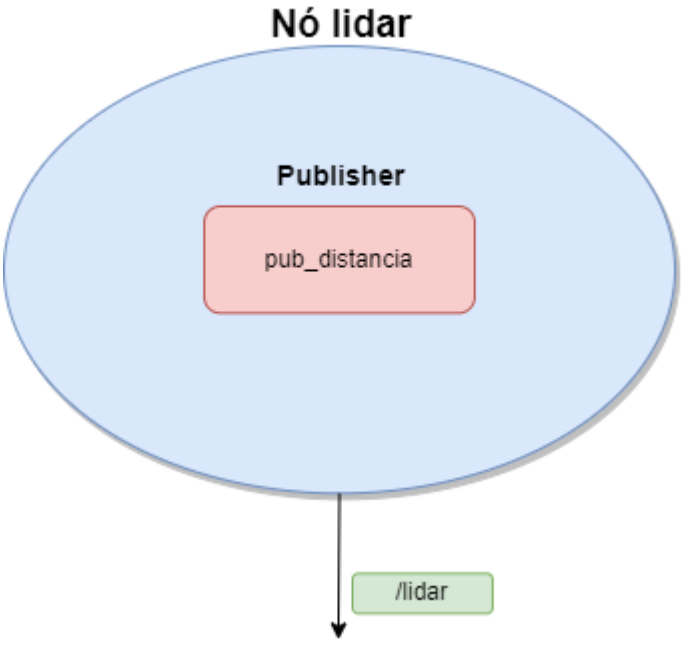


Fonte: O autor (2023)

3.10.5 Lidar

O nó lidar, ver Figura 61, é responsável por acionar o Ydlidar G2 e realizar a varredura do ambiente medindo as distâncias para os objetos que estão no campo de visão do robô e assim evitar a colisão frontal. No caso deste trabalho, foi restringido o campo de visão para que ele só retorne a distância do objeto mais próximo dentro do range de 120 graus, tomando como referência a parte frontal do Robô. Essa informação deve ser enviada pelo tópico /*lidar* .

Figura 61 – Nó do lidar

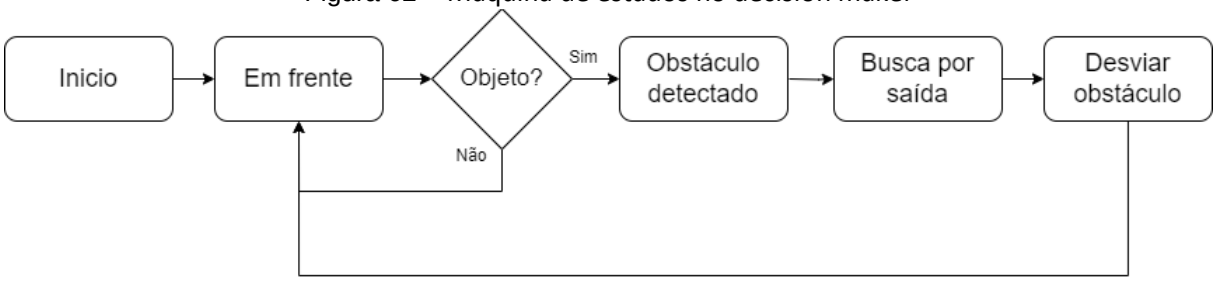


Fonte: O autor (2023)

3.11 EXEMPLO DE APLICAÇÃO.

Para facilitar a utilização do projeto foi desenvolvido um exemplo de aplicação que utiliza todos os drivers para realizar um desafio. Nesse desafio o robô deve se locomover em um ambiente controlado, evitando a colisão com obstáculos. A lógica para superar esse desafio foi implementada em um novo nó em ROS chamado *DescisionMaker*. O nó *Decisionmaker* é responsável por controlar a sequência de ações do robô de acordo com os eventos. Ele possui uma estrutura de máquina de estados que segue o fluxo mostrado na figura 62.

Figura 62 – Máquina de estados nó decision maker

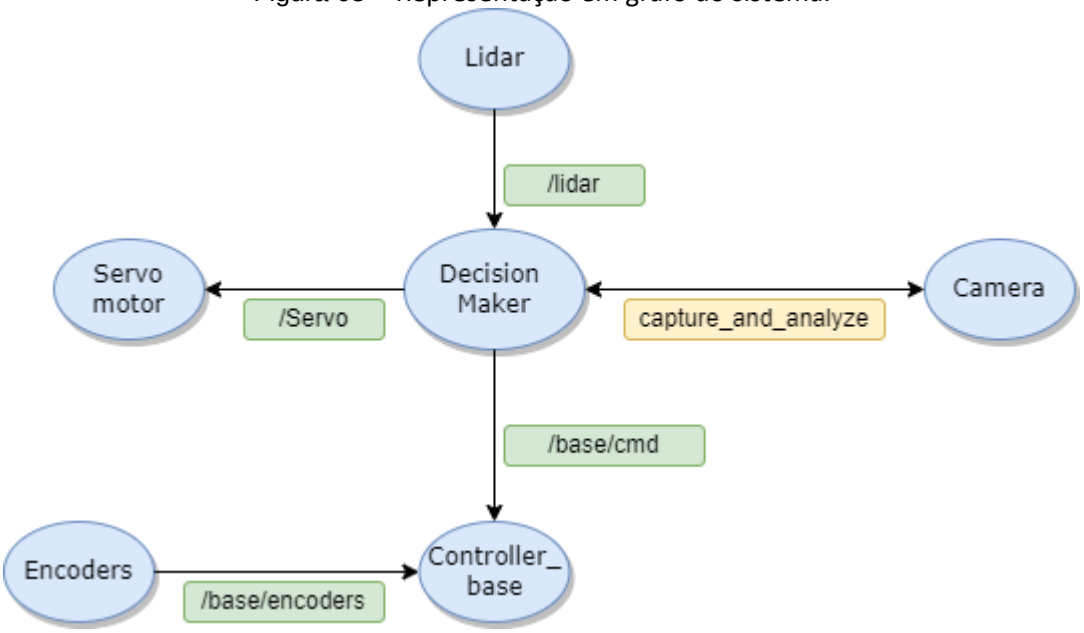


Fonte: O autor (2023)

- Início: em início o robô inicializa e espera 10 segundos para que todos os nós realizem suas rotinas de início.
- Em frente: nesse estado o robô ativo os motores com uma determinada velocidade e começa a se mover para frente até que encontre um obstáculo. Caso encontro ele muda para o estado de obstáculo detectado.
- Obstáculo detectado: Para o os motores da locomoção.
- Busca por saída: Utiliza o servo motor e câmera para buscar uma saída na direita e depois na esquerda.
- Desviar obstáculo: Com as informações do seu contorno ele realiza a manobra de saída e volta para o estado em frente.

O diagrama do sistema representa a união de todos nós. Na imagem abaixo é possível ver uma estrutura em grafos que representa os nós em azul, tópicos em verde e serviços em amarelo ver Figura 63.

Figura 63 – Representação em grafo do sistema.



Fonte: O autor (2023)

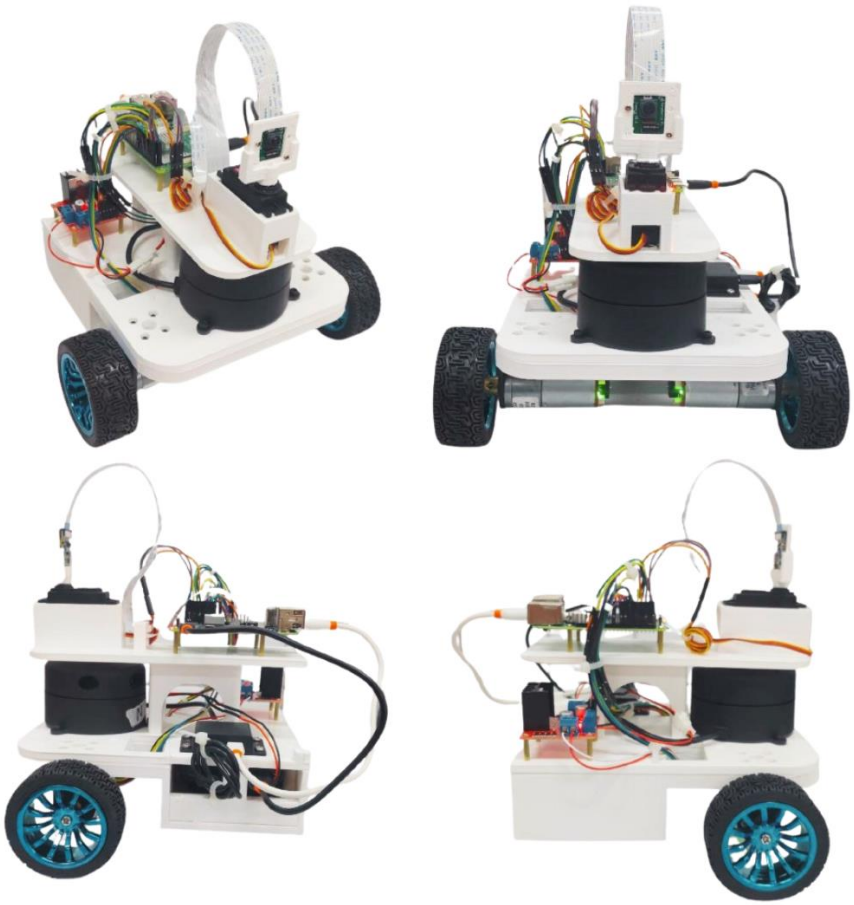
4 RESULTADOS

Neste capítulo serão apresentados os resultados alcançados no desenvolvimento da plataforma. Primeiro a o resultado geral do modelo fabricado, depois a validação dos drivers para acionamento e leitura de hardwares utilizados no projeto, a performance da plataforma na resolução do desafio e por último a comparação com as plataformas comerciais existentes.

4.1 RESULTADO GERAL

Como resultado do trabalho foram construídos quatro modelos da plataforma robótica que estão sendo utilizados na disciplina de programação robótica disponibilizada pelo departamento de engenharia mecânica da UFPE. A versão final da plataforma respeita o escopo inicial do projeto. Sendo modular, robusto, de fácil montagem e acesso aos componentes. Na Figura 64 é possível ver o modelo montado.

Figura 64 – Plataforma robótica fabricada.



Fonte: O autor (2023)

As versões do modelo 3D estão disponíveis no (github) o que permite a replicação do projeto e a implementação de melhorias. Junto com os modelos 3D estão disponíveis os nós desenvolvidos para a utilização dos hardwares. Além disso, foi elaborado junto com o professor Adrien uma ficha educacional que descreve os comportamentos básicos dos hardwares e propõe um desafio de robótica para ser superado com o uso do modelo em sala de aula, esse material se encontra no apêndice G.

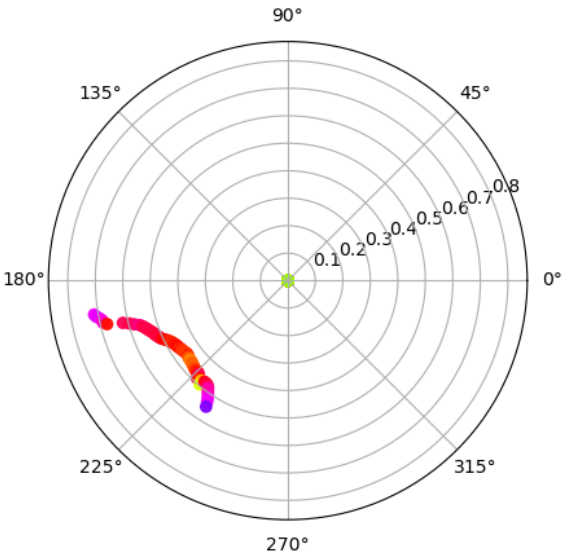
4.2 VALIDAÇÃO DOS DRIVERS.

Cada nó apresentado anteriormente foi testado exaustivamente de maneira individual. Os resultados dos testes serão exibidos nos próximos tópicos. Para a validação do envio das informações por alguns nós será utilizado uma função do ROS que permite ver o eco das mensagens enviadas por um tópico especificado. Essa função exibe no terminal as mensagens que estão percorrendo o tópico.

4.2.1 Lidar

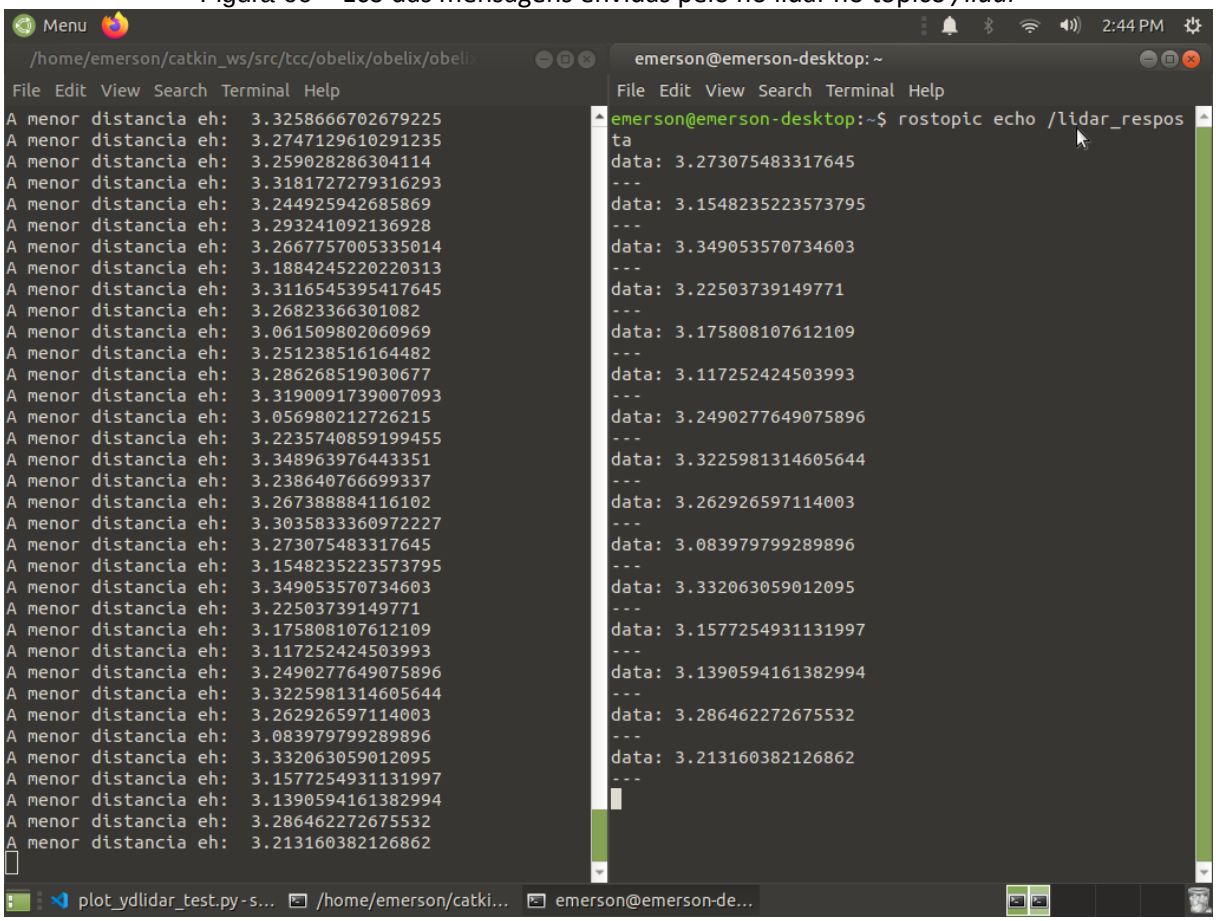
O nó do lidar deve retornar continuamente o valor da distância do objeto mais próximo. A Figura 65 exibe um gráfico com as distâncias captadas pelo lidar em um range de 120°. Esse range foi definido para focar em evitar objetos na parte frontal do robô. A Figura 66 apresenta a saída no terminal dos objetos mais próximos captados pelo nó do lidar.

Figura 65 – Gráfico das distâncias do lidar até os objetos.



Fonte: O autor (2023)

Figura 66 – Eco das mensagens envidas pelo nó lidar no tópico */lidar*

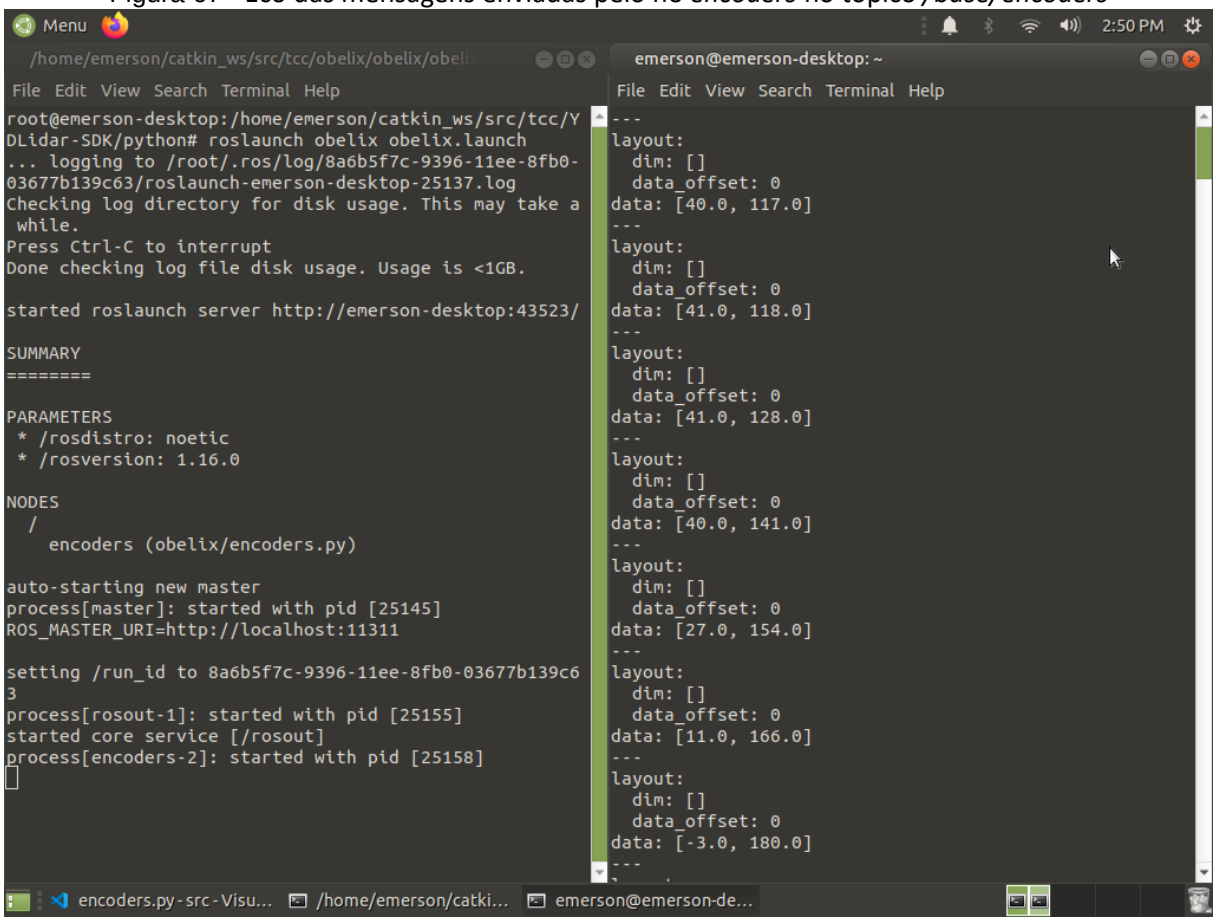


Fonte: O autor (2023)

4.2.2 *Encoders*

O nó *encoders* deve ler cada acionamento do *encoder* nos motores. Esses acionamentos são contabilizados e enviados. A Figura 67 os ecos das mensagens enviadas pelo no tópico */base/encoders* que contêm a contagem dos acionamentos dos *encoders*.

Figura 67– Eco das mensagens enviadas pelo nó *encoders* no tópico */base/encoders*



Fonte: O autor (2023)

4.2.3 *Controller base*

O nó *controller base*, atua recebendo os valores desejados de velocidade linear e angular, recebe o valor acumulado da leitura dos *encoders* e aplica técnicas de controle para atingir a velocidade desejada. A Figura 68 exibe a velocidade desejada e a velocidade atual ao longo do tempo.

Figura 68 – Variação da velocidade do robô no processo para atingir a velocidade desejada

Fonte: O autor (2023)

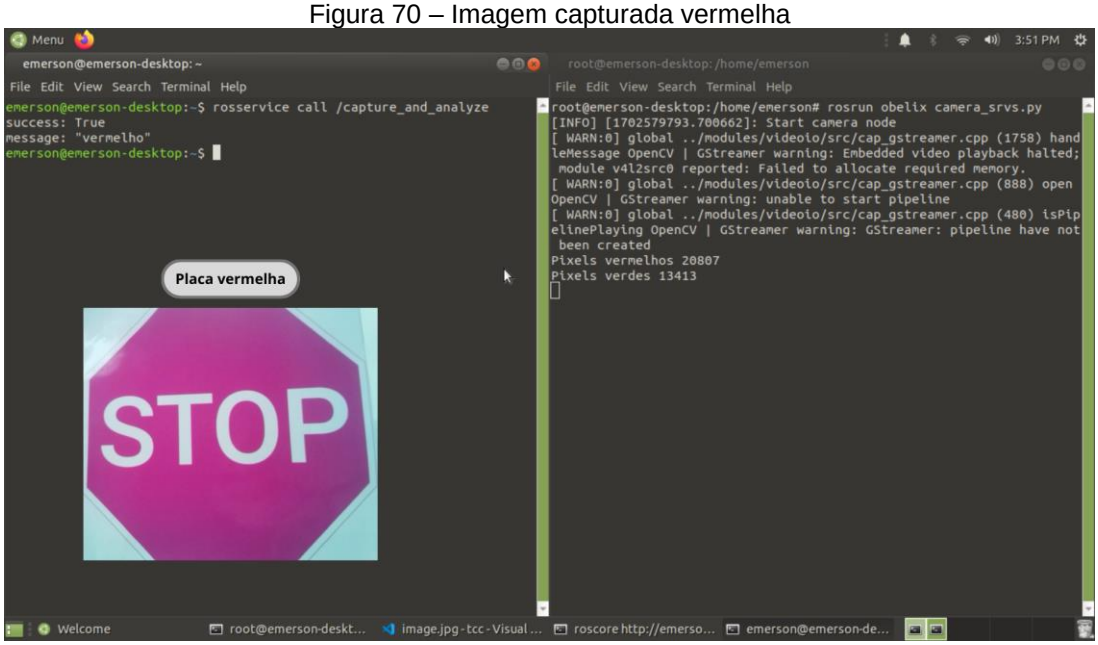
4.2.4 *Camera*

O nó da *camera* funciona como um serviço que quando solicitado aciona a câmera, processa a imagem realizando a contagem de pixels verdes e vermelhos que compõe a imagem e de acordo com a quantidade de pixels verdes e vermelhos informa ao robô se ele tem passagem livre. A Figura 69 exibe a imagem verde capturada pelo robô a Figura 70 exibe a imagem vermelha capturada pelo robô.

Figura 69 – Imagem capturada verde

Fonte: O autor (2023)

Figura 70 – Imagem capturada vermelha

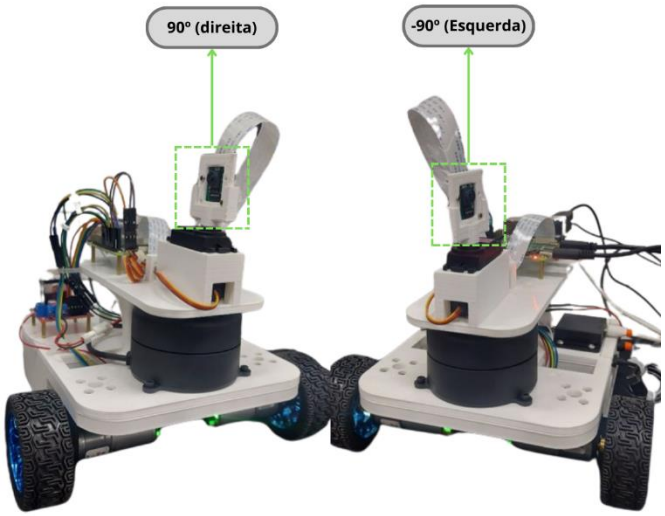


Fonte: O autor (2023)

4.2.5 Servo motor

O nó do servo motor, quando solicitado deve rotacionar a câmera na quantidade do valor em graus recebido. A Figura 71 exibe o resultado do seu acionamento de 90° e de -90º.

Figura 71 – Rotação do servo para direita e esquerda (90 e -90)



Fonte: O autor (2023)

4.3 PERFORMANCE NO DESAFIO

O desafio foi realizado em uma sala de aula da UFPE. O comportamento do robô pode ser visto no vídeo no drive. Ele atingiu as expectativas e evitando as colisões com os obstáculos e se movimentando da maneira esperada.

4.4 COMPARAÇÃO COM MODELOS COMERCIAIS

Como a proposta do trabalho é desenvolver uma plataforma educacional de robótica móvel competitiva e acessível financeiramente é importante analisar os custos para o seu desenvolvimento. O custo total para o desenvolvimento do projeto foi de R$ 2.663,47. A Tabela 11 abaixo traz os valores individuais de cada componente considerando os preços e o valor de conversão de real para dólar atuais.

Tabela 11 – Preços dos componentes

Nome	Preço
Raspberry pi 4 model B	R$ 700,00
Rasp cam	R$ 230,00
Motores (locomoção)	R$ 400,00
Servo motor	R$ 55,00
Powerbank	R$ 129,00
Caixa pilha	R$ 35,00
Ponte h l298n	R$ 21,90
Lidar ydlidar g2	R$ 963,57
Filamento	R$ 109,00
Parafusos	R$ 20,00

Fonte: O autor (2023).

A comparação da plataforma desenvolvida neste trabalho, chamada de Obelix, com os 4 modelos de robôs educacionais encontrados no mercado, que foram descritos na introdução deste trabalho, pode ser vista na Tabela 12.

Tabela 12 – Comparações entre robôs

Características	**Turtle bot 3**	**Turtle bot 4**	**Tiago**	**Duckiebot**	**obelix**
Dimensões	(281x306x141) mm	(341x339x351) mm	(1100x1450x540) mm	(340x150x230) mm	(220x185x196) mm
Peso	1.8 kg	3.9 kg	70 kg	1.8 kg	1.8Kg
Velocidade máxima	0.26 m/s	0.31 m/s	1.5 m/s	0.4 m/s	0.24m/s
Câmera	Raspberry Pi câmera	OAK-D-PRO	RGB-D camera	160 graus fov	Raspberry Pi câmera
Computador	Raspberry Pi 3	Raspberry Pi 4B (4 GB)	CPU intel i5/i7 RAM 8/16 GB SSD 250/500GB	NVIDIA Jetson Nano 4GB	Raspberry Pi 4B (4 GB)
Lidar	LDS-01	RPLIDAR-A1	Lidar 2D	Não apresenta	Ydlidar G2
Liberdade de câmera	Não apresenta	Não apresenta	Apresenta	Não apresenta	Apresenta
Software	ROS	ROS 2	ROS	ROS	ROS
Preço	R$ 8.151,43	R$ 10.328,14	R$ 259.584,79-R$486.272,58	R$ 2.182,18	R$ 2.663,47

Fonte: O autor (2023).

Focando na comparação com os modelos Turtlebot 3 e Duckiebot, o Obelix se destaca por suas dimensões compactas (220x185x196 mm) e peso leve de 1.8 Kg, proporcionando vantagens em termos de manobrabilidade. Sua velocidade máxima de 0.24 m/s é ligeiramente menor que a do TurtleBot3 (0.26 m/s) e Duckiebot (0.4 m/s). Tanto o Obelix quanto o Duckiebot utilizam a Raspberry Pi câmera, enquanto o TurtleBot3 opta por uma câmera RGB-D. Em relação aos computadores embarcados, o Obelix utiliza o Raspberry Pi 4B (4 GB), superior aos

modelos do TurtleBot3, que possui o Raspberry Pi 3, e o Duckiebot, que usa a NVIDIA Jetson Nano 4GB.

Analisando o lidar, tanto o Obelix quanto o TurtleBot3 utilizam, oferecendo varreduras 2D a laser, enquanto o Duckiebot não apresenta o Lidar. A liberdade de câmera é um diferencial entregue pelo Obelix, sendo ausente nos outros modelos. Todos os robôs adotam o ROS como software base. Em termos de preço, o Obelix destaca-se significativamente ao ser 67% mais barato que o TurtleBot 3, apresentando componentes semelhantes. Comparado ao Duckiebot, o Obelix é um pouco mais caro, mas possui componentes superiores em quase todos os aspectos.

5 CONCLUSÃO

O Robô Obelix se destaca como uma ótima escolha como plataforma educacional de robótica móvel quando comparado com os robôs comerciais. Ele atende a todos os requisitos propostos para esse trabalho. Devido às suas dimensões compactas e peso leve, promove facilidade de transporte e manipulação. Obelix oferece deslocamento eficiente e seguro para atividades práticas. Equipado com câmera e computador embarcado, o robô mantém um equilíbrio entre desempenho e custo, facilitando a integração em ambientes de aprendizado. Além disso, apresenta LIDAR que proporciona uma rica experiência de sensoriamento, junto com liberdade de movimento para a câmera. Utilizando o ROS, o Obelix integra-se perfeitamente a ambientes educacionais, proporcionando uma plataforma robusta para o aprendizado prático de robótica. E por fim, com um preço acessível oferece uma solução econômica e eficaz, tornando-se uma escolha vantajosa para instituições educacionais que buscam enriquecer a experiência de ensino em robótica.

Entretanto ainda apresenta algumas limitações. Para seu completo funcionamento necessita um ambiente controlado com um piso de superfície plana. Os drivers oferecidos são básicos, mas dá o acesso completo aos hardwares do projeto e permite o desenvolvimento de novos trabalhos para melhoria dos mesmo. Com o preço relativamente mais baixo que os robôs comerciais, para a realidade brasileira ainda pode apresentar dificuldade de implementação, mas devido a sua modularidade, pode ser removido o Lidar que é o componente com maior custo,

diminuindo assim significativamente o seu preço.

6 REFERÊNCIAS

Claro, aqui estão todas as referências reorganizadas em ordem alfabética:

A. STEFEK, T. V. PHAM, V. KRIVANEK AND K. L. PHAM, **Energy Comparison of Controllers Used for a Differential Drive Wheeled Mobile Robot** in IEEE Access, vol. 8, pp. 170915-170927, 2020, doi: 10.1109/ACCESS.2020.3023345.

ANDI DINE, GEORGE-CHRISTOPHER VOSNIAKOS, **On the development of a robot-operated 3D-printer, Procedia Manufacturing**, Volume 17, 2018, Pages 6-13, ISSN 2351-9789, https://doi.org/10.1016/j.promfg.2018.10.004.

Autodesk Fusion 360. 2023. Disponível em: <https://lutris.net/games/autodesk-fusion-360/> Acesso em 05 junho 2023.

Chapman, Stephen J. Fundamentos de máquinas elétricas. 5. ed. Porto Alegre: AMGH, 2013.

CSABA BENEDEK, ANDRAS MAJDIK, BALAZS NAGY, ZOLTAN ROZSA, TAMAS SZIRANYI, **Positioning and perception in LIDAR point clouds**, Digital Signal Processing, Volume 119, 2021, 103193, ISSN 1051-2004.

DANIELA AND M. D. LYTRAS, **Educational robotics for inclusive education, Technology, Knowledge and Learning**, vol. 24, no. 2, pp. 219–225, Jun 2019.

DA SILVA PONTES, PAULO RICARDO; VICTOR, VALCÍ FERREIRA. **Robótica educacional: uma abordagem prática no ensino de lógica de programação**. Revista Sítio Novo, v. 6, n. 1, p. 57-71, 2022.

Get Your AI in Gear with a JetBot AI Robot Kit. 2023. Disponível em: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetbot-ai-robot-kit/> Acesso em 05 junho 2023.

How to use the L298N motor driver module. Janeiro de 2023. Disponível em: <https://www.hibit.dev/posts/89/how-to-use-the-l298n-motor-driver-module> Acesso em 05 junho 2023.

Impressive 3D Ender 5 S1. 2023. Available at: <https://www.creality.com/br/products/ender-5-s1-3d-printer?spm=..index.products_tab_1.1> Accessed on June 05, 2023.

Introducing the TurtleBot 4. May 2023. Available at: <https://www.openrobotics.org/blog/2022/5/3/introducing-the-turtlebot-4> Accessed on June 05, 2023.

JAHN, UWE et al. **A Taxonomy for Mobile Robots: Types, Applications, Capabilities, Implementations, Requirements, and Challenges**. Robotics, 2020. Available at: https://www.mdpi.com/2218-6581/9/4/109#cite. Accessed May 12, 2021.

J.C.F. DIAZ, W.E. CARTER, R.L. SHRESTHA, C.L. GLENNIE. **LiDAR Remote Sensing Springer International Publishing**, Cham (2017), pp. 929-980, 10.1007/978-3-319-23386-4_44.

JULIANO GIANOTTO. **ANAC is collecting information on the use of drones for spraying in Brazil**. July 2021. Available at: <https://aeroin.net/anac-esta-coletando-informacoes-sobre-uso-de-drones-para-pulverizacao-no-brasil/>. Accessed June 05, 2023.

KIRPES C, HU G, SLY D. **The 3D Product Model Research Evolution and Future Trends: A Systematic Literature Review**. Applied System Innovation. 2022; 5(2):29. https://doi.org/10.3390/asi5020029.

KURDILA, ANDREW J.; BEM-TZVI, PINHAS. **Dynamics and Control of Robotic Systems**. John Wiley & Sons Ltd: 2020.

LiDAR: What it is, how it works, how to annotate it. 2023. Available at: < https://www.superannotate.com/blog/what-is-lidar-annotation> Accessed June 05, 2023.

MOHD JAVAID, ABID HALEEM, SHANAY RAB, RAVI PRATAP SINGH, RAJIV SUMAN, **Sensors for daily life: A review, Sensors International**, Volume 2, 2021, 100121, ISSN 2666-3511, https://doi.org/10.1016/j.sintl.2021.100121.

MOREIRA MACIEL, ARAÚJO LEAL. **Educational Robotics: challenges and perspectives in Brazilian Education**, Conjecturas. DOI: 10.53660/CONJ-1116-Q55.

Motor DC 6V 280RPM with Encoder and Wheel. 2023. Available at: <https://www.baudaeletronica.com.br/produto/motor-dc-6v-280-rpm-com-encoder-e-roda.html> Accessed June 05, 2023.

MWEMA FM, AKINLABI ET. **Basics of Fused Deposition Modelling (FDM). Fused Deposition Modeling.** 2020 May 30:1–15. doi: 10.1007/978-3-030-48259-6_1. PMCID: PMC7257444.

Ocean data faster, greener, and cheaper. Available at: <https://www.terradepth.com/>. Accessed June 05, 2023.

P. PETROV AND V. GEORGIEVA, **Adaptive Velocity Control for a Differential Drive Mobile Robot,** 2018 20th International Symposium on Electrical Apparatus and Technologies (SIELA), Bourgas, Bulgaria, 2018, pp. 1-4, doi: 10.1109/SIELA.2018.8447091.

Pololu Ball Caster with 3/4″ Metal Ball. 2023. Available at: <https://www.pololu.com/product/955> Accessed June 05, 2023.

PR2. 2023. Available at: <https://robotsguide.com/robots/pr2> Accessed June 05, 2023.

QUIGLEY, M., GERKEY, B., CONLEY, K., FAUST, J., FOOTE, T., LEIBS, J., BERGER, E., WHEELER, R. E NG, A. **ROS: an open-source Robot Operating System**, (2009), at: International Conference on Robotics and Automation.

QUIGLEY, MORGAN & CONLEY, KEN & GERKEY, BRIAN & FAUST, JOSH & FOOTE, TULLY & LEIBS, JEREMY & WHEELER, ROB & NG, ANDREW. **ROS: an open-source Robot Operating System**. (2009). ICRA Workshop on Open Source Software. 3.

R. MATHEW AND S. S. HIREMATH, **Development of Waypoint Tracking Controller for Differential Drive Mobile Robot** 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT), Paris, France, 2019, pp. 1121-1126, doi: 10.1109/CoDIT.2019.8820389.

Raspberry Pi 4 8GB - Model B Anatel. 2023. Available at: <https://www.robocore.net/placa-raspberry-pi/raspberry-pi-4-8gb> Accessed June 05, 2023.

Raspberry Pi Camera Module 2. 2023. Available at: <https://www.raspberrypi.com/products/camera-module-v2/> Accessed June 05, 2023.

Raspberry Pi Documentation. 2023. Available at: <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html> Accessed June 05, 2023.

SOHAN, MD SHAHED. (2023). **Accident Reduction System Design by Detecting Drowsiness**. 10.5281/zenodo.8265893.

Transform for Every Agricultural Mission. Available at: <https://www.xa.com/en/r150-2022> Accessed June 05, 2023.

Turtlebot 3 overviews. Available at: <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/> Accessed June 05, 2023.

TIAGo Tutorials. August 2023. Available at: <https://wiki.ros.org/Robots/TIAGo/Tutorials> Accessed June 05, 2023.

Understanding Resolution in optical and magnetic Encoders Available at: <https://www.elektronikpraxis.de/understanding-resolution-in-optical-and-magnetic-encoders-a-427263/>

APÊNDICE A – Nó camera:

```
#!/usr/bin/env python3

# Node information:
# Name: camera
# Description: Camera node as as service that captures and return image analysis

# importing libraries
import rospy
import cv2
import numpy as np
from std_srvs.srv import Trigger, TriggerResponse

class Camera():
    '''Class to manage the node'''
    def __init__(self):
        self.pixelRed = 0
        self.pixelGreen = 0
        self.threshold = 50
        self.new_dimension_image = (480, 320)

        # service
        self.camera_service = rospy.Service('capture_and_analyze', Trigger,
self.capture_and_analyze)



    def capture_and_analyze(self, req):
        '''Capture ande analyze image'''

        self.camera_image = cv2.VideoCapture(0)

        try:
```

```
            if self.camera_image.isOpened():
                validation, frame = self.camera_image.read()
                if validation:
                    cv2.imshow("Camera",frame)
                    frame = cv2.rotate(frame, cv2.ROTATE_180)

cv2.imwrite('/home/emerson/catkin_ws/src/tcc/obelix/imagens/image.jpg', frame)

                    frame_resized = cv2.resize(frame, self.new_dimension_image,
interpolation = cv2.INTER_AREA)

                    for i in range(self.new_dimension_image[0]):
                        for j in range(self.new_dimension_image[1]):
                            if (frame_resized[j][i][1]<self.threshold):
                                self.pixelRed = self.pixelRed + 1
                            if (frame_resized[j][i][2]<self.threshold):
                                self.pixelGreen = self.pixelGreen + 1

                    if (self.pixelRed>self.pixelGreen):
                        color = 'vermelho'
                        print("Pixels vermelhos",self.pixelRed)
                        print("Pixels verdes",self.pixelGreen)
                        self.camera_image.release()
                        return TriggerResponse(success=True, message=color)

                    else:
                        color = 'verde'
                        print("Pixels vermelhos",self.pixelRed)
                        print("Pixels verdes",self.pixelGreen)
                        self.camera_image.release()
                        return TriggerResponse(success=True, message=color)

        except KeyboardInterrupt:
            self.camera_image.release()
```

```
        self.camera_image.release()
        color="sem_cor"
        return TriggerResponse(success=True, message=color)

##################
# --- MAIN --- #
##################

def main():
    rospy.init_node('camera_srvs')
    camera = Camera()
    rospy.loginfo('Start camera node')

    rate = rospy.Rate(10)

    while not rospy.is_shutdown():

        rate.sleep()

if (__name__ == '__main__'):
    main()
```

APÊNDICE B – Nó servo motor:

```
#!/usr/bin/env python3

##########
#Informações sobre esse Node:
#Nome: servo_motor
#Descrição: movimentar o servo motor

from gpiozero import AngularServo
from time import sleep
import rospy
from std_msgs.msg import Int32

PIN_SERVO = 14

class Servo(object):

    def __init__(self):
        #setup

        self.servo = AngularServo(PIN_SERVO, min_pulse_width=0.0004,
max_pulse_width=0.0025)

        self.angulo_servo = rospy.Subscriber('servo', Int32, self.callback_angulo)

    def callback_angulo(self,msg_angulo):
        '''Altera o angulo do servo'''

        self.servo.angle = msg_angulo.data


def main():
    rospy.init_node('servo_motor')
```

```
    servo = Servo()
    rospy.loginfo('Inicio no locomocao')

    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        rate.sleep()

if (__name__ == '__main__'):
    main()
```

APÊNDICE C – Nó Lidar:

```
#!/usr/bin/env python3

# Node information:
# Name: Lidar
# Description: captures data from the lidar and sends the average distance

# importing libraries
import os
import sys
import time
from matplotlib.patches import Arc
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import ydlidar
import rospy
from std_msgs.msg import Float64

RMAX = 32.0

class Lidar():

    def __init__(self):
        #self.setup_plot_lidar()
        self.setup_lidar()

        self.lidar_resposta = Float64()
        #pub
        self.pub = rospy.Publisher('lidar_resposta', Float64, queue_size=1)


    def setup_plot_lidar(self):
```

```
        '''setup plot lidar'''

        self.fig = plt.figure()
        self.fig.canvas.set_window_title('YDLidar LIDAR Monitor')
        self.theta_offset_deg = 45
        self.theta_offset_rad = np.radians(self.theta_offset_deg)
        self.lidar_polar = plt.subplot(polar=True, theta_offset=self.theta_offset_rad)
        self.lidar_polar.set_rmax(RMAX)
        self.lidar_polar.autoscale_view(True,True,True)
        self.lidar_polar.grid(True)

    def setup_lidar(self):
        self.ports = ydlidar.lidarPortList()
        self.port = "/dev/ydlidar"
        for key, value in self.ports.items():
            self.port = value

        self.laser = ydlidar.CYdLidar();
        self.laser.setlidaropt(ydlidar.LidarPropSerialPort, self.port);
        self.laser.setlidaropt(ydlidar.LidarPropSerialBaudrate, 512000)
        self.laser.setlidaropt(ydlidar.LidarPropLidarType, ydlidar.TYPE_TRIANGLE);
        self.laser.setlidaropt(ydlidar.LidarPropDeviceType,
ydlidar.YDLIDAR_TYPE_SERIAL);
        self.laser.setlidaropt(ydlidar.LidarPropScanFrequency, 10.0);
        self.laser.setlidaropt(ydlidar.LidarPropSampleRate, 9);
        self.laser.setlidaropt(ydlidar.LidarPropMaxAngle, -90.0);
        self.laser.setlidaropt(ydlidar.LidarPropMinAngle, -180.0);
        self.laser.setlidaropt(ydlidar.LidarPropMaxRange, 10.0);
        self.laser.setlidaropt(ydlidar.LidarPropMinRange, 0.08);
        self.laser.setlidaropt(ydlidar.LidarPropSingleChannel, False);
        self.scan = ydlidar.LaserScan()

    def animate(self, num):
```

```
    r = self.laser.doProcessSimple(self.scan);
    if r:
        angle = []
        ran = []
        intensity = []
        for point in self.scan.points:
            angle.append(point.angle);
            ran.append(point.range);
            intensity.append(point.intensity);
        range_sem_zeros = np.array(ran)[np.array(ran)!=0]
        if len(range_sem_zeros) > 0:
            print("A menor distancia eh: ", np.mean(range_sem_zeros))
        self.lidar_polar.clear()
        self.lidar_polar.scatter(angle, ran, c=intensity, cmap='hsv', alpha=0.95)

def visualzar_animacao(self):
    ret = self.laser.initialize();
    if ret:
        ret = self.laser.turnOn();
        if ret:
            ani = animation.FuncAnimation(self.fig, self.animate, interval=50)
            plt.show()
        self.laser.turnOff();
    self.laser.disconnecting();
    plt.close();

def iniciar_lidar(self):
    ret = self.laser.initialize();
    if ret:
        ret = self.laser.turnOn();

def finalizar_lidar(self):

    self.laser.turnOff();
```

```
        self.laser.disconnecting();

    def ler_distancia(self):

        r = self.laser.doProcessSimple(self.scan);
        if r:
            angle = []
            ran = []
            intensity = []
            for point in self.scan.points:
                angle.append(point.angle);
                ran.append(point.range);
                intensity.append(point.intensity);

            range_sem_zeros = np.array(ran)[np.array(ran)!=0]

            if len(range_sem_zeros) > 0:
                self.lidar_resposta.data = np.mean(range_sem_zeros)
                print("A menor distancia eh: ", self.lidar_resposta.data)
                self.pub.publish(self.lidar_resposta)

def main():
    rospy.init_node('lidar')
    lidar = Lidar()
    rospy.loginfo('Inicio no lidar')

    lidar.iniciar_lidar()

    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        lidar.ler_distancia()
        rate.sleep()
```

```
    lidar.finalizar_lidar()

if (__name__ == '__main__'):
    main()
```

APÊNDICE D – Nó Encoders:

```
#!/usr/bin/env python3

# Node information:
# Name: encoders
# Description: Counts encoders pulses

# importing libraries
import RPi.GPIO as GPIO
import rospy
from std_msgs.msg import Float32MultiArray

contar_pulso = 0

# Setup the pin numbers

# For the right encoder
ENC_R1 = 3
ENC_R2 = 4

# For the left encoder
ENC_L1 = 15
ENC_L2 = 18

class encoders:

    def __init__(self):

        ############
        # Publisher

        self.pub = rospy.Publisher('/base/encoders', Float32MultiArray, queue_size=1)
        self.lr_counter = Float32MultiArray()
```

```
    self.lr_counter.data = [0]*2

    ####################
    # Configure the pin

    GPIO.setmode(GPIO.BCM)

    # Right encoder
    GPIO.setup(ENC_R1,GPIO.IN)
    GPIO.setup(ENC_R2,GPIO.IN)
    GPIO.add_event_detect(ENC_R1,GPIO.RISING,callback=self.counter_er1)

    # Left encoder
    GPIO.setup(ENC_L1,GPIO.IN)
    GPIO.setup(ENC_L2,GPIO.IN)
    GPIO.add_event_detect(ENC_L1,GPIO.RISING,callback=self.counter_el1)

# Definition of the callback functions
def counter_el1(self, channel):
    # self.left_counter +=1
    if GPIO.input(ENC_L2):
        self.lr_counter.data[0] +=1
    else:
        self.lr_counter.data[0] -=1

def counter_er1(self, channel):
    # self.left_counter +=1
    if GPIO.input(ENC_R2):
        self.lr_counter.data[1] -=1
    else:
        self.lr_counter.data[1] +=1

# Publisher function
def pub_counter(self):
```

```
        self.pub.publish(self.lr_counter)


##################
# --- MAIN --- #
##################

def main():

    rospy.init_node('encoders')
    enc = encoders()

    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        enc.pub_counter()
        rate.sleep()

if (__name__=='__main__'):
    main()
```

APÊNDICE E – Nó Controller base:

```
#!/usr/bin/env python3

# Node information:
# Name: controller_base
# Description: Kinematics and base control

# importing libraries
import RPi.GPIO as GPIO
import time
import math
#from apscheduler.schedulers.background import BackgroundScheduler

import rospy
from std_msgs.msg import Float32MultiArray
from geometry_msgs.msg import Twist

class controller:

    def __init__(self):

        #############
        # Subscriber
        rospy.Subscriber('/base/encoders', Float32MultiArray, self.cb_encoders,
queue_size=1)
        rospy.Subscriber('/base/cmd', Twist, self.cb_velocity, queue_size=1)

        ######################
        # Robot's parameters
        self.interWheels = 0.2 ### to be checked
        self.reductionRatio = 46
        self.nbPoles = 20
        self.wheelDiameter = 0.067
```

```
self.pulseToRad = 2*math.pi / (self.reductionRatio*self.nbPoles)
self.v = 0.1 # linear velocity
self.w = 0.0 # angular velocity

#########################
# Controller's paramters
self.ts = 0.1 # sampling time
self.kp = 3
self.ki = 0
self.kd = 0.2
self.leftInt = 0
self.rightInt = 0
self.refLeftVelocity = 0.3
self.refRightVelocity = 0.3
self.maxAngVelocity = 130/60*math.pi*2
self.initEncoder = False

########################################
# Variables to compute the current speed
self.timeInterval = 0.1
self.previousLeft = 0.0
self.previousRight = 0.0
self.leftVelocity = 0.0
self.rightVelocity = 0.0

########################
# Setup the pin numbers

# For the right motor
self.mr_fwd = 23    #in1
self.mr_bck = 24    #in2
self.mr_en = 25     #ina

# For the left motor
```

```
        self.ml_fwd = 22    #in3
        self.ml_bck = 27    #in4
        self.ml_en = 17     #inb

        ####################
        # Configure the pin

        GPIO.setmode(GPIO.BCM)

        # Right motor
        GPIO.setup(self.mr_fwd,GPIO.OUT)
        GPIO.setup(self.mr_bck,GPIO.OUT)
        GPIO.setup(self.mr_en,GPIO.OUT)
        GPIO.output(self.mr_fwd,GPIO.LOW)
        GPIO.output(self.mr_bck,GPIO.LOW)
        self.mr=GPIO.PWM(self.mr_en,1000)
        self.mr.start(0) #46% -> 60 RPM

        # Left motor
        GPIO.setup(self.ml_fwd,GPIO.OUT)
        GPIO.setup(self.ml_bck,GPIO.OUT)
        GPIO.setup(self.ml_en,GPIO.OUT)
        GPIO.output(self.ml_fwd,GPIO.LOW)
        GPIO.output(self.ml_bck,GPIO.LOW)
        self.ml=GPIO.PWM(self.ml_en,1000)
        self.ml.start(0)

    def __del__(self):

        self.ml.ChangeDutyCycle(0)
        self.mr.ChangeDutyCycle(0)

        GPIO.output(self.mr_fwd,GPIO.LOW)
        GPIO.output(self.mr_bck,GPIO.LOW)
```

```
        GPIO.output(self.ml_fwd,GPIO.LOW)
        GPIO.output(self.ml_bck,GPIO.LOW)

        GPIO.cleanup()

    # Sign function
    def my_sign(self, x):
        return (x > 0) - (x < 0)

    # Callback encoder
    def cb_encoders(self, msg):
        '''
            Le o pulso dos encoders,
            atualiza os pulsos de cada motor
            Calcula a velocidade de cada motor
            Atualiza o valor da velocidade
        '''
        # Save the current number of counts
        left = float(msg.data[0])
        right = float(msg.data[1])

        # Init the previous values to deal with non zero initial encoders
        if not self.initEncoder:
            self.previousLeft = left
            self.previousRight = right
            self.initEncoder = True

        # Compute the motor velocity
        self.leftVelocity = (left - self.previousLeft) / self.timeInterval * self.pulseToRad
        self.rightVelocity = (right - self.previousRight) / self.timeInterval *
self.pulseToRad

        # Update the previous number of counts
        self.previousLeft = left
```

```
    self.previousRight = right

# Callback velocities
def cb_velocity(self, msg):
    '''Recebe o valor desejado de velocidade'''

    # Update the velocities
    self.v = msg.linear.x
    self.w = msg.angular.z

# PID controller
def pid(self):

    # Convert the base velocities into wheels velocities
    # converte as velocidades desejadas em velocidade da roda
    delta = self.interWheels*self.w / (self.wheelDiameter/2)
    l_vel = self.v/(self.wheelDiameter/2) - delta/2
    r_vel = self.v/(self.wheelDiameter/2) + delta/2

    # Adjustment based on feedbacks
    leftErr = l_vel - self.leftVelocity
    self.leftInt += leftErr*self.ts
    leftDer = leftErr / self.ts
    #l_vel_send += self.kp*leftErr + self.kd*leftDer + self.ki*self.leftInt
    l_vel_send = l_vel+ self.kp*leftErr + self.kd*leftDer + self.ki*self.leftInt

    rightErr = r_vel - self.rightVelocity
    self.rightInt += rightErr*self.ts
    rightDer = rightErr / self.ts
    #r_vel_send += self.kp*rightErr + self.kd*rightDer + self.ki*self.rightInt
    r_vel_send = r_vel + self.kp*rightErr + self.kd*rightDer + self.ki*self.rightInt

    # Adjust velocties accordind to motors bounds
    if abs(l_vel_send) > self.maxAngVelocity:
```

```
            ratio = self.maxAngVelocity/abs(l_vel_send)
            l_vel_send = self.my_sign(l_vel_send)*self.maxAngVelocity
            r_vel_send *= ratio

        if abs(r_vel_send) > self.maxAngVelocity:
            ratio = self.maxAngVelocity/abs(r_vel_send)
            r_vel_send = self.my_sign(r_vel_send)*self.maxAngVelocity
            l_vel_send *= ratio

        # Detect the wheels direction
        if l_vel_send >= 0:
            l_forward = GPIO.HIGH
            l_backward = GPIO.LOW
        else:
            l_forward = GPIO.LOW
            l_backward = GPIO.HIGH

        if r_vel_send >= 0:
            r_forward = GPIO.HIGH
            r_backward = GPIO.LOW
        else:
            r_forward = GPIO.LOW
            r_backward = GPIO.HIGH

        # Convert wheels velocity into duty cycle
        l_dc = math.floor(abs(l_vel_send) / self.maxAngVelocity*100)
        r_dc = math.floor(abs(r_vel_send) / self.maxAngVelocity*100)

        #print("Desired left: {} - Measured left : {} - Send left:
{}".format(l_vel,self.leftVelocity,l_vel_send))
        #print("Desired right: {} - Measured right : {}- Send right:
{}".format(r_vel,self.rightVelocity,r_vel_send))

        print("Desired left: {} - Measured left : {}".format(l_vel,self.leftVelocity))
```

```
        print("Desired right: {} - Measured right : {}".format(r_vel,self.rightVelocity))

        # Update the duty cycle
        self.ml.ChangeDutyCycle(l_dc)
        self.mr.ChangeDutyCycle(r_dc)

        # Update the motors direction
        GPIO.output(self.ml_fwd,l_forward)
        GPIO.output(self.ml_bck,l_backward)
        GPIO.output(self.mr_fwd,r_forward)
        GPIO.output(self.mr_bck,r_backward)


##################
# --- MAIN --- #
##################

rospy.init_node('controller')
cont = controller()

# rospy.spin()

rate = rospy.Rate(1/cont.ts)

while not rospy.is_shutdown():
    cont.pid()
    rate.sleep()
```

APÊNDICE F – Nó DecisionMaker:

```
#!/usr/bin/env python3

##########
#Informações sobre esse Node:
#Nome: decison_maker
#Descrição: configura a maquina de estados geral

import os
import sys
import time
import numpy as np
import rospy
from std_msgs.msg import Float64, Int32
from geometry_msgs.msg import Twist
from std_srvs.srv import Trigger

TEMPO_ROTACAO_ESQUERDA = 2.5
TEMPO_ROTACAO_DIREITA = 2.5
TEMPO_MOVER_PARA_TRAS = 2

class Decision_maker():

    def __init__(self):
        # estado inicial
        self.estado = self.alternador_de_estados('iniciar')

        # mensagem_resposta
        self.eventos = {
            'resposta_lidar': 2.1,
            'resposta_camera': str(),
            'direita': 'vermelho',
            'esquerda': 'vermelho',
```

```
            'trazeira': 'vermelho'
        }

        # mensagens_enviadas
        self.mensagem_base = Twist()
        self.mensagem_camera = Int32()
        self.mensagem_servo = Int32()

        # pub
        #self.pub_camera = rospy.Publisher('camera', Int32, queue_size=1)
        self.pub_servo = rospy.Publisher('servo', Int32, queue_size=1)
        self.pub_base = rospy.Publisher('/base/cmd', Twist, queue_size=1)

        # sub
        #self.camera = rospy.Subscriber('camera_resposta', Int32,
self.callback_camera)
        self.lidar = rospy.Subscriber('lidar_resposta',Float64,self.callback_lidar)

        # cliente
        self.cliente_camera = rospy.ServiceProxy('capture_and_analyze', Trigger)

    #CALLBACKS

    def callback_camera(self, msg_camera):
        '''Recebe a mensagem da camera'''

        self.eventos['resposta_camera'] = msg_camera.data
        #print("resposta_camera: ", self.resposta_camera)

    def callback_lidar(self, msg_lidar):
        '''Recebe a mensagem do lidar'''

        self.eventos['resposta_lidar'] = msg_lidar.data
        #print("resposta_lidar: ", self.resposta_lidar)
```

```
# Maquina de estados

def alternador_de_estados(self, estado):
    '''todos os estados'''
    self.estados = {
        'iniciar': self.setup_robo,
        'em_frente': self.frente,
        'obstaculo_detectado': self.obstaculo_detectado,
        'procurar_saidas': self.procurar_saidas,
        'evitar_obstaculo': self.evitar_obstaculo
    }
    return self.estados[estado]

def em_evento(self, eventos):
    '''em evento'''
    self.estado = self.estado(self.eventos)

# Estados

def setup_robo(self, eventos):
    '''Realiza a rotina para iniciar o robo'''
    print("Inicinando robo em ... ")
    time.sleep(2)
    print('3 ...')
    time.sleep(2)
    print('2 ...')
    time.sleep(2)
    print('1 ...')
    return self.alternador_de_estados('em_frente')

def frente(self, eventos):
    '''ir em frente'''
```

```
        print("Vamos em frente !")

        if (eventos['resposta_lidar'] > 0.2):
            self.mover_robo_frente()
            return self.alternador_de_estados('em_frente')
        return self.alternador_de_estados('obstaculo_detectado')

    def obstaculo_detectado(self, eventos):
        print('Obstaculo detectado')
        print('Esperando motor para...')
        self.parar()
        time.sleep(3)
        # if (eventos['resposta_lidar'] > 0.2):
        #     return self.alternador_de_estados('em_frente')
        return self.alternador_de_estados('procurar_saidas')

    def procurar_saidas(self, eventos):
        self.rotacionar_servo_direita()
        print('procurando na direita')
        eventos['direita'] = self.solicita_analise_camera()
        print(f"lado direito {eventos['direita']}")
        self.rotacionar_servo_esquerda()
        print('procurando na esquerda')
        eventos['esquerda'] = self.solicita_analise_camera()
        print(f"lado esquerda {eventos['esquerda']}")
        self.rotacionar_servo_frente()
        time.sleep(2)
        return self.alternador_de_estados('evitar_obstaculo')

    def evitar_obstaculo(self, eventos):

        if (eventos['direita']=='vermelho')&(eventos['esquerda']=='vermelho'):
            print('desviar de obstaculo pela direita')
            self.rotacionar_robo_direita()
```

```
            return self.alternador_de_estados('em_frente')
        elif (eventos['direita']=='vermelho')&(eventos['esquerda']=='verde'):
            print('desviar de obstaculo pela direita')
            self.rotacionar_robo_direita()
            return self.alternador_de_estados('em_frente')
        elif (eventos['direita']=='verde')&(eventos['esquerda']=='vermelho'):
            print('desviar de obstaculo pela esquerda')
            self.rotacionar_robo_esquerda()
            return self.alternador_de_estados('em_frente')
        elif (eventos['direita']=='verde')&(eventos['esquerda']=='verde'):
            print('desviar de obstaculo indo para tras')
            self.mover_para_tras()
            self.rotacionar_robo_direita()
            return self.alternador_de_estados('em_frente')

    # Metodos gerais

    def solicita_analise_camera(self):
        rospy.wait_for_service('capture_and_analyze')
        try:
            response = self.cliente_camera()
            self.eventos['resposta_camera'] = str(response.message)
            return self.eventos['resposta_camera']
        except rospy.ServiceException as e :
            rospy.logerr(f'Erro ao chamar servico: {e}')
        return 4

    def velocidade_desejada(self, linear=0.0, angular=0.0):
        '''setar velocidade desejada'''

        self.mensagem_base.linear.x = linear
        self.mensagem_base.angular.z = angular
        self.pub_base.publish(self.mensagem_base)
```

```
def posicao_desejada_servo(self, angulo=0):
    '''Muda a posicao do servo para a desejada'''

    self.mensagem_servo.data = angulo
    self.pub_servo.publish(self.mensagem_servo)

def mover_robo_frente(self):
    angular = 0.0
    linear = 0.3
    self.velocidade_desejada(linear,angular)

def rotacionar_robo_esquerda(self):
    '''Rotacionar o robo para esquerda'''

    angular = 1.0
    linear = 0.0
    self.velocidade_desejada(linear,angular)
    time.sleep(TEMPO_ROTACAO_ESQUERDA)
    self.velocidade_desejada(0,0)

def rotacionar_robo_direita(self):
    '''Rotacionar o robo para direita'''

    angular = -1.0
    linear = 0.0
    self.velocidade_desejada(linear,angular)
    time.sleep(TEMPO_ROTACAO_DIREITA)
    self.velocidade_desejada(0,0)

def parar(self):
    '''parar'''

    angular = 0.0
    linear = 0.0
```

```
        self.velocidade_desejada(linear,angular)

    def mover_para_tras(self):
        angular = 0.0
        linear = -0.3
        self.velocidade_desejada(linear,angular)
        time.sleep(TEMPO_MOVER_PARA_TRAS)
        self.velocidade_desejada(0,0)

    def rotacionar_servo_esquerda(self):
        '''Rotacionar servo para esquerda'''

        self.posicao_desejada_servo(90)

    def rotacionar_servo_direita(self):
        '''Rotacionar servo para direita'''

        self.posicao_desejada_servo(-90)

    def rotacionar_servo_frente(self):
        '''Rotacionar servo para direita'''

        self.posicao_desejada_servo(0)

def main():
    rospy.init_node('decision_maker')
    decision_maker = Decision_maker()
    rospy.loginfo('Inicio no Decision_maker')

    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        decision_maker.em_evento(decision_maker.eventos)
        rate.sleep()
```

```
if (__name__ == '__main__'):
    main()
```

APÊNDICE G – Ficha educacional

Projeto ROS
UFPE - Demec
ME653 Programação Robótica

1 Descrição do projeto

Este projeto visa colocar em prática em uma plataforma robótica os conceitos relacionados ao ROS vistos em sala de aula. O projeto está dividido em duas partes principais. A primeira consiste em implementar os nós que fazem a ligação entre o hardware e um usuário. Este trabalho é semelhante ao realizado por um engenheiro que **construindo** o robô. A segunda parte exigiu a implementação de algoritmos semelhantes aos do usuário para resolver os desafios. Este trabalho é semelhante ao realizado por um engenheiro usando o robô. Este trabalho é semelhante ao realizado por um engenheiro **usando** o robô.

2 Plataforma robótica

2.1 Modelagem

Neste projeto, utilizamos um robô diferencial, que consiste em duas rodas acionadas independentemente que giram sobre o mesmo eixo, bem como um caster ball que mantém o robô na horizontal. Denotamos o raio das rodas como r e a distância entre as rodas como $2d$. Além disso, definimos u_L e u_R come a velocidade angular da roda esquerda e direita.
As velocidades linear v e angular ω do robô podem ser obtidas usando as equações seguinte:

$$v = \frac{r}{2}u_L + \frac{r}{2}u_R \tag{1}$$

$$\omega = -\frac{r}{2d}u_L + \frac{r}{2d}u_R \tag{2}$$

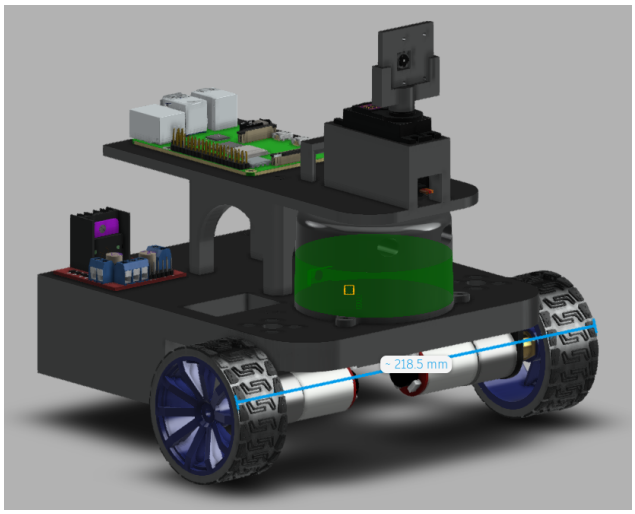


Figura 1: Modelo

2.2 Material

O robô usado neste projeto consiste nos seguintes elementos:

- 1x estrutura de plástico
- 2x motores DC 3-6V com caixa de redução
- 2x codificadores magnéticos
- 1x servomotor
- 1x câmera Raspberry Pi v2 8MP
- 1x laser YDlidar G2
- 1x mini-computador Raspberry Pi 4
- 1x placa ponte H
- 2x baterias

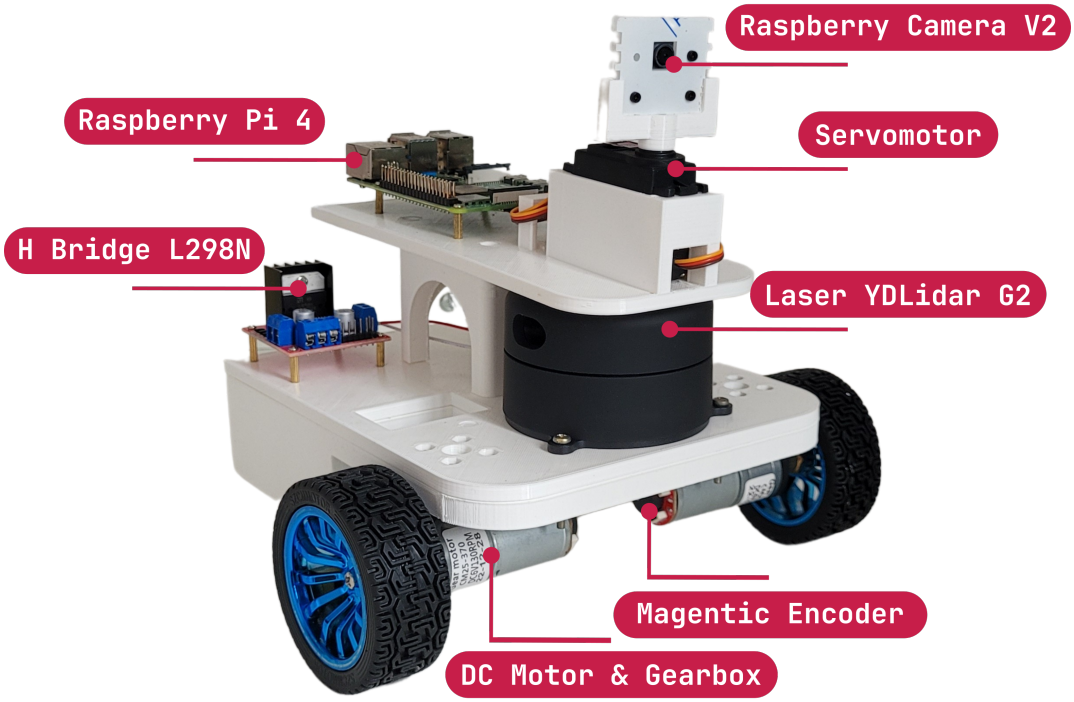


Figura 2: Plataforma robótica

Figura 3: Bateria 1 (Raspberry Pi & Laser)

Figura 4: Bateria 2 (Motores & codificadores)

3 Raspberry Pi 4

3.1 Hardware

O Raspberry Pi 4 (RP4) é um mini-computadores de placa única multiplataforma cujos componentes principais são mostrados na Fig. 5.

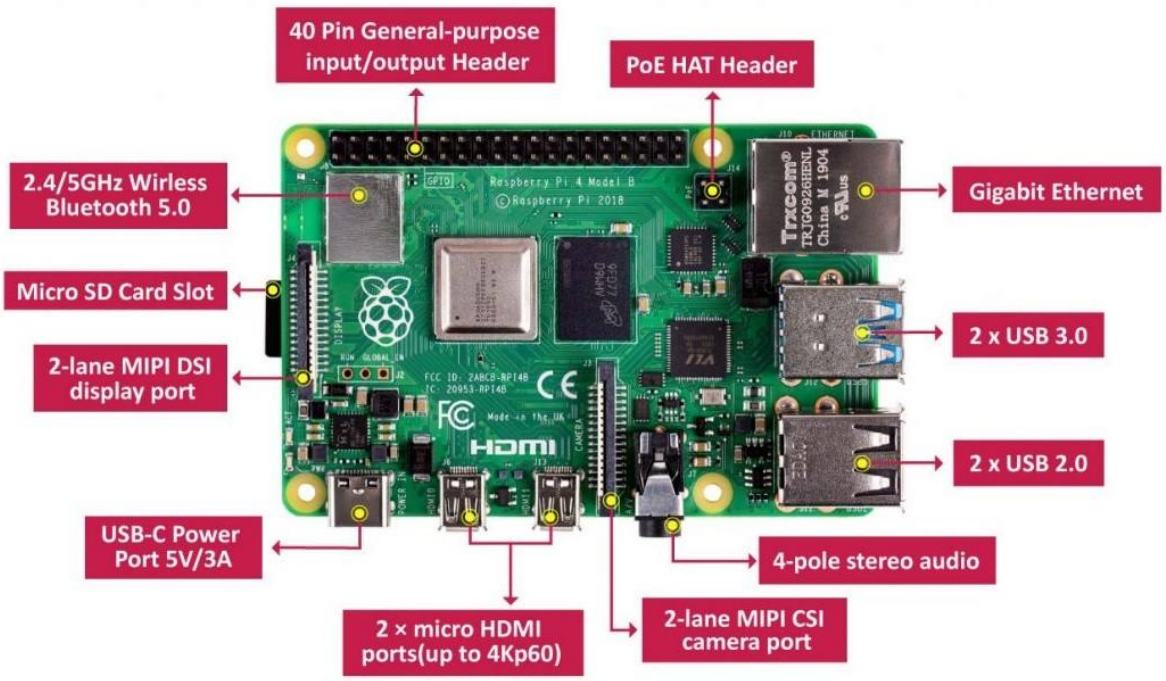


Figura 5: Raspberry Pi 4

3.2 Software

O Raspberry Pi4 está equipado com um cartão microSD contendo:

- O sistema operacional Ubuntu Mate 20.04
- O middleware ROS Noetic
- A linguagem de programação Python 3
- A biblioteca RPi
- A biblioteca OpenCv

.

Como o RP4 está a bordo do robô, não é prático usar um teclado e uma tela para interagir com ele. Neste projeto, portanto, usamos os comandos `ssh` e `scp` para interagir remotamente com o RP4. **Para usar os comandos apresentados nesta seção, o computador do usuário deve estar conectado à rede *meca_robo* com a senha *meca_robo*.**

3.2.1 ssh

Secure Shell (SSH) é um protocolo de rede criptográfico para operação de serviços de rede de forma segura sobre uma rede insegura. O SSH fornece um canal seguro sobre uma rede insegura em uma arquitetura cliente-servidor, conectando uma aplicação cliente SSH com um servidor SSH. Aplicações comuns incluem login em linha de comando remoto e execução remota de comandos.

Sintaxe:
`ssh [opções] usuário@host`
Exemplo:
`ssh student@168.192.1.100`
Todos os RP4 tem um usuário "adp". A lista do IP da cada robô é a seguinte:

Computador	Usuário	IP	Senha
asterix	robot	192.168.1.102	Mec@tr0n
obelix	robot	192.168.1.100	Mec@tr0n
idefix	robot		Mec@tr0n
panoramix	robot		Mec@tr0n

3.2.2 scp

Secure Copy, em Português cópia segura, ou simplesmente SCP, é um meio seguro de transferência de arquivos entre um servidor local e um remoto ou entre dois servidores remotos, usando o protocolo SSH.
Tipicamente a sintaxe do programa `scp` é parecida com a sintaxe do `cp`:
`scp /ArquivoFonte usuário@host:/diretório/ArquivoAlvo`
`scp usuário@host:/diretório/ArquivoFonte /ArquivoAlvo`
Usamos a opção `-r` para copiar/colar uma pasta.

HARDWARE

Conectar o Raspberry Pi à bateria 1 (ou fonte de alimentação)

ROS

Criar um `workspace` *robot_ws*
Criar um `package` *drivers*
Criar um `package` *maze*

3.3 Pinos GPIO

O RP4 possui pinos GPIO (General Purpose Input Output), portas programáveis de entrada e saída de dados, para permitir interação e o controle de LEDs, interruptores, sinais analógicos, sensores e outros dispositivos. Como os pinos GPIO não tem função definida e por padrão não são usadas, é possível usar a biblioteca RPi para programar os pinos, cujo mapa é mostrado na Fig. 6.

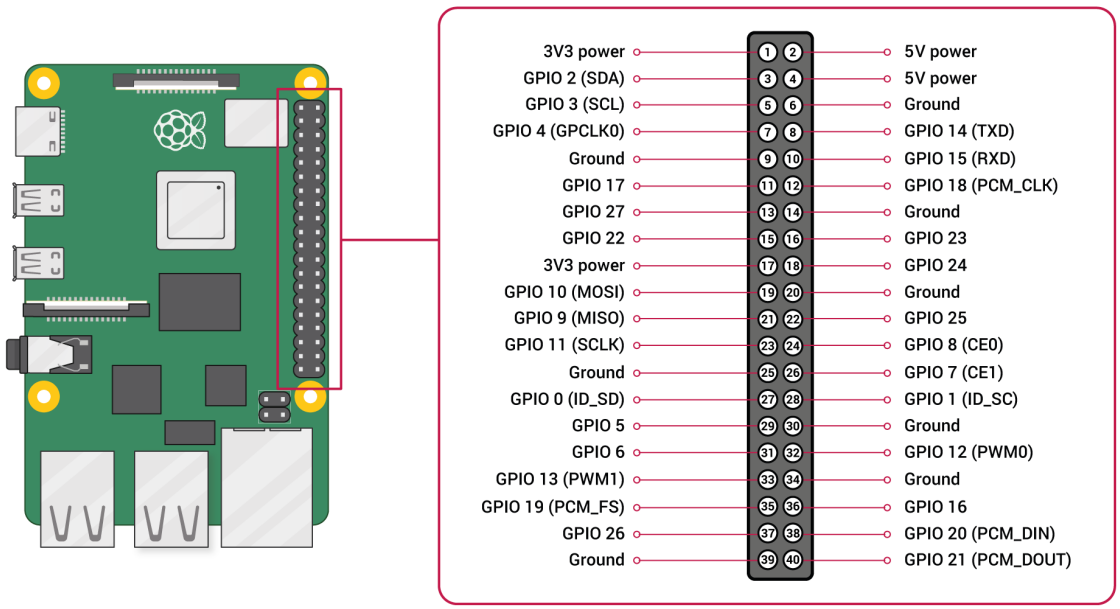


Figura 6: Pinos GPIO

3.3.1 Output

Os pinos GPIO podem ser usados como saída lógica: baixo (0 V) ou alto (5 V).

Listing 1: Output

```
#!/usr/bin/env python3

# To import the RPi.GPIO module
import RPi.GPIO as GPIO

# Set up the board layout
GPIO.setmode(GPIO.BCM)

# Set up pin 24 as output
GPIO.setup(24,GPIO.OUT)

# Turn OFF pin 24
GPIO.output(24,GPIO.LOW)

# Turn ON pin 24
GPIO.output(24,GPIO.HIGH)

# Cleanup at the end of the program
GPIO.cleanup()
```

3.3.2 PWM

Pulse Width Modulation, ou PWM, é uma técnica para obter resultados analógicos com meios digitais. O controle digital é usado para criar uma onda quadrada, um sinal alternado entre ligado e desligado. Este padrão on-off pode simular tensões entre o Vcc total da placa (por exemplo, 5 V) e desligado (0 V), alterando a parte do tempo que o sinal fica ligado versus o tempo que o sinal fica desligado. A duração do "on time"é

chamada de largura de pulso ou 'Pulse Width'. Para obter valores analógicos variados, você altera ou modula essa largura de pulso. Se você repetir esse padrão liga-desliga, o resultado será como se o sinal fosse uma tensão constante entre 0 e Vcc (Fig. 7).

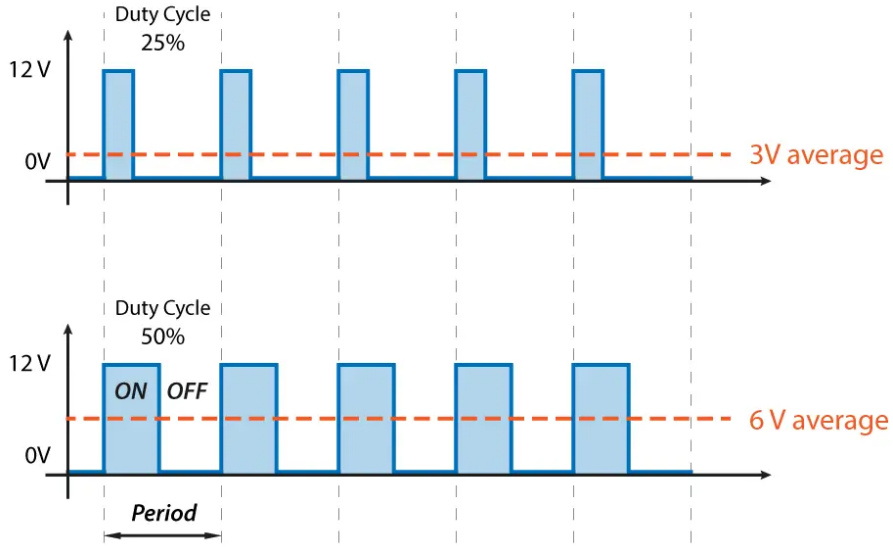


Figura 7: Pulse With Modulation

Listing 2: PWM

```
#!/usr/bin/env python3

# To import the RPi.GPIO module
import RPi.GPIO as GPIO

# Set up the board layout
GPIO.setmode(GPIO.BCM)

# Set up pin 25 as output
GPIO.setup(25,GPIO.OUT)

# Set up pin 25 as PWM with a 1000 Hz frequency
pwm25=GPIO.PWM(25,1000)

# Start with a duty cycle of 50%
pwm25.start(50)

# Change the frequency
pwm25.ChangeFrequency(500)

# Change the duty cycle
pwm25.ChangeDutyCycle(100)

# Stop
pwm25.stop()

# Cleanup at the end of the program
GPIO.cleanup()
```

3.3.3 Input

Existem várias maneiras de obter entrada GPIO em seu programa. A primeira e mais simples maneira é verificar o valor de entrada em um ponto no tempo. Isso é conhecido como 'polling' e pode potencialmente perder uma entrada se o seu programa ler o valor na hora errada. O polling é executado em loops e pode consumir muito do processador. A outra maneira de responder a uma entrada GPIO é usando 'interrupções' (detecção de borda). Uma borda é o nome de uma transição de HIGH para LOW (borda descendente) ou LOW para HIGH (borda ascendente).
RPi.GPIO executa um segundo thread para funções callback. Isso significa que as funções callback podem ser executadas ao mesmo tempo que seu programa principal, em resposta imediata a uma borda.

Listing 3: Input

```
#!/usr/bin/env python3

# To import the RPi.GPIO module
import RPi.GPIO as GPIO

# Definition of the callback function for event detection
def cb_function(channel):
    print("Hello World")


GPIO.setmode(GPIO.BCM)

# Set up pin 10 as input
GPIO.setup(10,GPIO.IN)

# Add event detection for pin 10
GPIO.add_event_detect(10,GPIO.RISING,callback=cb_function)

# Remove event detection for pin 10
GPIO.remove_event_detect(channel)
```

4 Motores DC e Codificadores

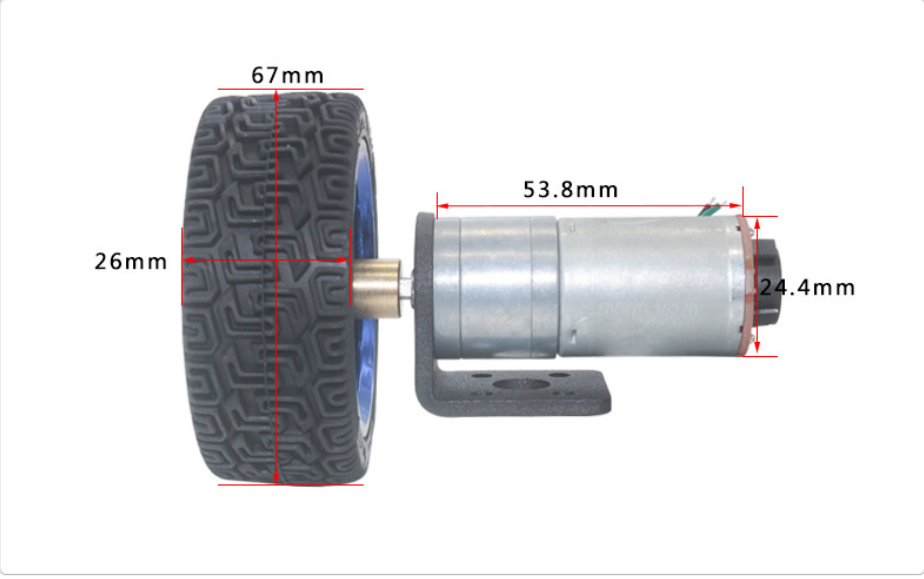


Figura 8: Roda, motor, caixa de redução e codificador

O robô é equipado com rodas acionadas por motores DC acoplados a caixas de redução, com mostrado na Fig. 8. As características dos motores, das caixas de redução e dos codificadores são dadas na Tab. 1.

Tensão maximal no motor	6V
Velocidade maximal do motor	130 RPM
Razão de redução da caixa	46

Tabela 1: Características dos motores com caixa de redução

O código de cores dos fios do par motor-codificador é dado na Tab. 2

Vermelho	Fonte de alimentação positiva do motor(+)
Branco	Fonte de alimentação negativa do motor(-)
Amarelo	Sinal feedback (uma volta do motor tem 11 sinais)
Verde	Sinal feedback (uma volta do motor tem 11 sinais)
Azul	Fonte de alimentação positiva do codificador (+)(3.3-5v)
Preto	Fonte de alimentação negativa do codificador (-)(3.3-5v)

Tabela 2: Código de cores dos fios do par motor-codificador

HARDWARE

O computador deve estar desligado
Conectar os cabos de feedback dos codificadores ao Raspberry Pi (usar pinhos com uma única função)
Conectar os cabos de alimentação dos codificadores ao Raspberry Pi
Verificar as conexões com o professor

ROS

Nome: Codificador
Entrada: -
Saída: Publisher (Float32MultiArray)
Descrição: Conta o número de sinais de cada codificador e publica os resultados

5 Ponte H

O chip L298N é um circuito integrado da STMicroelectronics que contém principalmente:

- 2 pontes H, cada uma permitindo acionar 1 motor elétrico DC (em uma direção ou outra)
- Uma lógica de controle de "baixa corrente", para conduzir essas pontes de "alta corrente"

Resumindo, o L298N permiti o controle direto de dois motores elétricos, por meio de controles lógicos de "baixa potência".
O L298N requer 2 fontes de alimentação separadas para operar:

- Uma tensão para a parte de potência, que será utilizada para alimentar os motores, através de transistores de potência
- Uma tensão para a parte de controle, que será usada para alimentar toda a parte lógica de controle, incluindo esses transistores de potência

No nível de controle lógico, distinguimos:

- Pinos de acionamento da ponte (ENA e ENB), que permitem a partida ou parada dos motores. Observe que essas entradas podem ser alimentadas em tudo ou nada (assim os motores irão “girar” na velocidade máxima), ou em PWM, para controlar sua velocidade de rotação
- Pinos de seleção de ponte (IN1, IN2, IN3 e IN4), que permitem selecionar os sentidos de rotação dos motores (e como aqui existem dois motores controláveis, existem 4 entradas, correspondentes às 4 possibilidades de sentido de rotação)

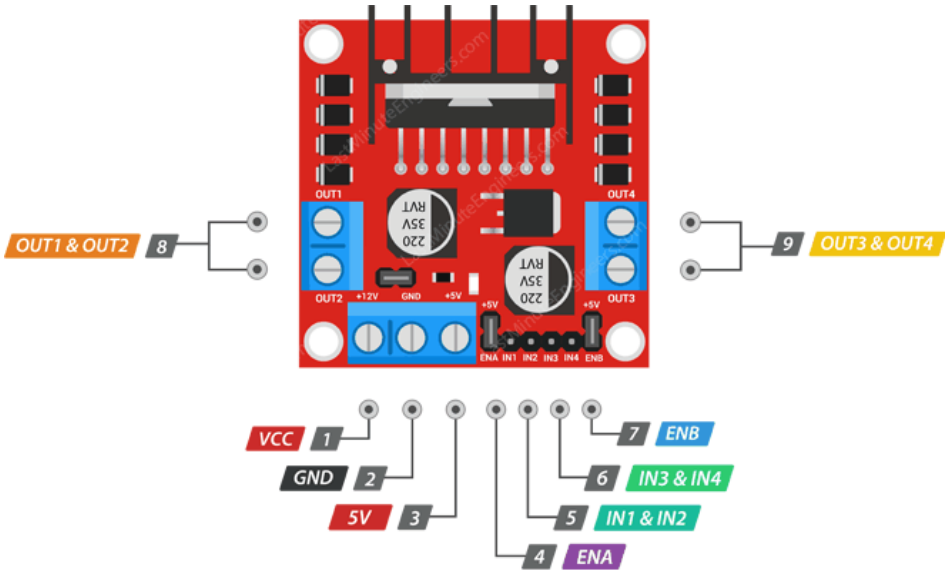


Figura 9: Ponte H L298N

1	VCC	Fonte de alimentação positiva da placa (+)
2	GND	Fonte de alimentação negativa da placa (-)
3	5V	Saída de 5V
4	ENA	PWM do motor A
5	IN1	Sentido de rotação do motor A
5	IN2	Sentido de rotação do motor A
6	IN3	Sentido de rotação do motor B
6	IN4	Sentido de rotação do motor B
7	ENB	PWM do motor B
8	OUT1	Alimentação negativa do motor A (-)
8	OUT2	Alimentação positiva do motor A (+)
9	OUT3	Alimentação positiva do motor B (+)
9	OUT4	Alimentação negativa do motor B (-)

Tabela 3: Entradas-saídas do L298N

HARDWARE

O computador deve estar desligado
Conectar os cabos de alimentação dos motores à placa L298N
Conectar as entradas logicas ao Raspberry Pi
Conectar o GND da placa L298N ao GND do Raspberry Pi
Conectar as fontes de alimentação da placa L298N à **bateria 2**
Verificar as conexões com o professor

ROS

Nome: Base_controller
Entrada: Subscriber (Float32MultiArray) - Subscriber (Twist)
Saída:
Descrição: 1: Calcula a velocidade angular dos motores - 2: Calcula a velocidade angular desejada - 3: Calcula o sinal PWM de cada motor usando um controlador PID - 4: Envia os sinais elétricos à placa ponte H

6 Servomotor

Figura 10: Servomotor TowerPro MG996R

O robô está equipado com um servomotor TowerPro MG996R que permite orientar a câmera. Estas características e o código colorido dos fios são dados nas Tab. 4 e 5
Os servos são controlados pela largura do pulso, a largura do pulso determina o ângulo. Uma largura de pulso de 1500 μs move o servo para o ângulo 0. Cada aumento de 10 μs na largura de pulso normalmente move o servo 1 grau mais no sentido horário. Cada diminuição de 10 μs na largura de pulso normalmente move o servo 1 grau mais no sentido anti-horário. Servos normalmente recebem 50 pulsos por segundo (50 Hz). Alguns cálculos em 50 Hz para larguras de pulso de amostra.

$$500/20000 = 0.025 \text{ ou } 2.5\ \%\ \text{dutycycle}$$

Tensão de operação	4,8 a 6,0 V
Tipo de Engrenagem	Metálica
Modulação	Digital
Velocidade de operação	0,19 seg/60 graus (4,8 V sem carga)
Velocidade de operação	0,15 seg/60 graus (6 V sem carga)
Torque (stall)	9,4 kgf.cm (4,8 V) e 11,0 kgf.cm (6 V)

Tabela 4: Código de cores dos fios do par motor-codificador

Marão	Fonte de alimentação negativa do motor(-) (0 V)
Vermelho	Fonte de alimentação positiva do motor(+) (5 V)
Amarelo	Sinal de controle PWM

Tabela 5: Código de cores dos fios do par motor-codificador

$$1000/20000 = 0.05 \text{ ou } 5.0\ \%\ \text{dutycycle}$$

$$1500/20000 = 0.075 \text{ ou } 7.5\ \%\ \text{dutycycle}$$

$$2000/20000 = 0.1 \text{ ou } 10.0\ \%\ \text{dutycycle}$$

$$2500/20000 = 0.125 \text{ ou } 12.5\ \%\ \text{dutycycle}$$

HARDWARE

O computador deve estar desligado
A câmera deve estar desconectada
Conectar os cabos de alimentação do servomotor à placa L298N
Conectar a entrada logica ao Raspberry Pi
Verificar as conexões com o professor

ROS

Nome: Servo_controller
Entrada: ServiceRequest (JointTrajectory)
Saída: ServiceResponse (livre)
Descrição: O nó deve oferecer um serviço de posicionamento da câmera

7 Lidar

Figura 11: Lidar YDlidar G2

O primeiro sensor exteroceptivo a bordo do robô é um lidar, modelo YDlidar G2. Estas características são dados na Tab. 6.

Ranging frequency	5000 Hz
Motor frequency	5 − > 12 Hz
Ranging distance	0.12 - 16 m
Field of view	0-360 Deg
Systematic error	2 cm
Luminous intensity range	0 - 1023
Angle resolution	0.36 − > 0.864 Deg

Tabela 6: Código de cores dos fios do par motor-codificador

G2 define internamente um sistema de coordenadas polares. As coordenadas polares do sistema consideram o centro do núcleo rotativo de G2 como o pólo, e o ângulo especificado é positivo no sentido horário (vista superior). O ângulo zero está localizado na direção da saída da linha de interface G2 PH2.0-5P.

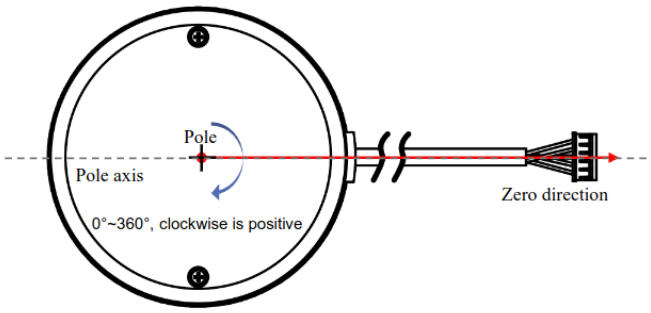


Figura 12: Lidar YDlidar G2

Um exemplo de código Python que permite trabalhar com o Lidar está disponível em 4.

Listing 4: Lidar

```
import os
import ydlidar
import time

if __name__ == "__main__":
    ydlidar.os_init();
    ports = ydlidar.lidarPortList();
    port = "/dev/ydlidar";
    for key, value in ports.items():
        port = value;
        print(port);
    laser = ydlidar.CYdLidar();
    laser.setlidaropt(ydlidar.LidarPropSerialPort, port);
    laser.setlidaropt(ydlidar.LidarPropSerialBaudrate, 115200);
    laser.setlidaropt(ydlidar.LidarPropLidarType, ydlidar.TYPE_TRIANGLE);
    laser.setlidaropt(ydlidar.LidarPropDeviceType, ydlidar.
   YDLIDAR_TYPE_SERIAL);
    laser.setlidaropt(ydlidar.LidarPropScanFrequency, 10.0);
    laser.setlidaropt(ydlidar.LidarPropSampleRate, 3);
    laser.setlidaropt(ydlidar.LidarPropSingleChannel, True);
    laser.setlidaropt(ydlidar.LidarPropMaxAngle, 180.0);
    laser.setlidaropt(ydlidar.LidarPropMinAngle, -180.0);
    laser.setlidaropt(ydlidar.LidarPropMaxRange, 16.0);
    laser.setlidaropt(ydlidar.LidarPropMinRange, 0.08);
    laser.setlidaropt(ydlidar.LidarPropIntenstiy, False);

    ret = laser.initialize();
    if ret:
        ret = laser.turnOn();
        scan = ydlidar.LaserScan();
        while ret and ydlidar.os_isOk() :
            r = laser.doProcessSimple(scan);
            if r:
                print("Scan received[",scan.stamp,"]:",scan.points.size(),
   "ranges is [",1.0/scan.config.scan_time,"]Hz");
            else :
                print("Failed to get Lidar Data")
            time.sleep(0.05);
        laser.turnOff();
    laser.disconnecting();
```

HARDWARE

O computador deve estar desligado
Conectar o cabo de comunicação ao Raspberry Pi (USB 3)
Conectar a alimentação do Lidar à bateria 1
Verificar as conexões com o professor

ROS

Nome: Lidar
Entrada:
Saída: Publisher (LaserScan)
Descrição: Publica os dados do lidar

8 Câmera

O segundo sensor exteroceptivo a bordo do robô é uma câmera, modelo Raspberry Pi module V2. Essas características são dados na Tab. 7.

Figura 13: Raspberry Pi Camera Module v2

Size	25x24x9 mm
Weight	3g
Resolution	8 Megapixels
Sensor	Sony IMX219
Sensor resolution	3280 x 2464 pixels
Sensor image area	3.68 x 2.76 mm (4.6 mm diagonal)
Pixel size	1.12 μm x 1.12 μm
Optical size	1/4”
Focus	Adjustable
Depth of field	10 cm to inf
Focal length	3.04 mm
Horizontal Field of View (FoV)	62.2 degrees
Vertical Field of View (FoV)	48.8 degrees
Focal ratio (F-Stop)	F2.0
Maximum exposure times (seconds)	11.76

Tabela 7: Parâmetros do Raspberry Pi Camera Module v2

Um exemplo de código Python que permite trabalhar com a câmera está disponível em 5.

Listing 5: Lidar

```
import cv2

# open camera
cap = cv2.VideoCapture('/dev/video0', cv2.CAP_V4L)

# set dimensions
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 2560)
```

```
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1440)

# take frame
ret, frame = cap.read()
# write frame to file
cv2.imwrite('image.jpg', frame)
# release camera
cap.release()
```

HARDWARE

O computador deve estar desligado
Conectar a câmera ao Raspberry Pi **Verificar as conexões com o professor**

ROS

Nome: Camera
Entrada: ServiceRequest (livre)
Saída: ServiceResponse (Image)
Descrição: O nó deve fornecer as imagens da câmera via um serviço