INSTITUTO FEDERAL DE EDUCAÇÃO,
CIÊNCIA E
TECNOLOGIA DO AMAZONAS
CAMPUS MANAUS DISTRITO INDUSTRIAL
ENGENHARIA DE CONTROLE E
AUTOMAÇÃO

INSTITUTO FEDERAL
Amazonas
Campus Manaus-Distrito Industrial

LUCIANO SANTOS BEZERRA

IMPLEMENTAÇÃO DE ALGORITMO HECTOR SLAM EM VEÍCULO TERRESTRE NÃO TRIPULADO EM AMBIENTE DINÂMICO E SEM ACESSO A GPS

MANAUS
2023

LUCIANO SANTOS BEZERRA

IMPLEMENTAÇÃO DE ALGORITMO HECTOR SLAM EM VEÍCULO TERRESTRE NÃO TRIPULADO EM AMBIENTE DINÂMICO E SEM ACESSO A GPS

Trabalho de Graduação apresentado ao corpo docente do departamento de Engenharia de Controle e Automação do Instituto Federal do Amazonas, como parte dos requisitos necessários à obtenção do título de Bacharel em Engenharia de Controle e Automação.

Área de concentração: Controle e Automação.

Orientador: Prof. Msc. Úrsula Vasconcelos Abecassis

MANAUS

2023

Dados Internacionais de Catalogação na Publicação (CIP)

B574i Bezerra, Luciano Santos

Implementação de algoritmo Hector Slam em veículos terrestre não tripulado em ambiente dinâmico e sem acesso a GPS. / Luciano Santos Bezerra. – Manaus, 2024.

118 f. : il. color

TCC (Graduação em engenharia de controle e automação) – Instituto Federal de Educação, Ciência e Tecnologia do Amazonas, *Campus* Manaus Distrito Industrial, 2024.

Orientadora: Prof. Msc. Úrsula Vasconcelos Abecassis

1. ROS. 2. Hector Slam. 3. Mapeamento. 4.VTNT. I. Abecassis, Úrsula Vasconcelos (orient.) II. Instituto Federal de Educação, Ciência e Tecnologia do Amazonas. III. Título.

CDD 629.8

Elabora por Fcª. Amélia Frota, registro n.858 (CRB11)

IMPLEMENTAÇÃO DE AGORITMO HECTOR SLAM EM VEÍCULO TERRESTRE NÃO TRIPULADO EM AMBIENTE DINÂMICO E SEM ACESSO A GPS

LUCIANO SANTOS BEZERRA

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Engenharia de Controle e Automação do Instituto Federal de Ciência e Tecnologia do Amazonas (IFAM), como requisito parcial para obtenção do Título de Bacharel em Engenharia de Controle e Automação.

BANCA EXAMINADORA

gov.br Documento assinado digitalmente
URSULA VASCONCELOS ABECASSIS
Data: 12/01/2024 13:11:16-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Msc. Úrsula Vasconcelos Abecassis (Presidente)

Instituto Federal do Amazonas – CMDI

gov.br Documento assinado digitalmente
JOSE FABIO DE LIMA NASCIMENTO
Data: 12/01/2024 14:52:02-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Msc. José Fábio de Lima Nascimento (Membro)

Instituto Federal do Amazonas – CMDI

gov.br Documento assinado digitalmente
RENAN CAVALCANTE SANTOS
Data: 12/01/2024 16:32:44-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Msc. Renan Cavalcante Santos (Membro)

Instituto Federal do Amazonas – CMDI

Dedico este trabalho a minha mãe e ao meu pai, que sempre me apoiaram, e ao amor da minha vida, que nunca desistiu de mim. Sem vocês eu não teria chegado até aqui.

AGRADECIMENTOS

A Deus.

A minha família que sempre foi meu alicerce e me incentivou a seguir os meus sonhos.

Aos meus amigos e colegas que dividiram a jornada comigo até aqui. Em especial a minha amiga Laleska que muito contribuiu nessa conquista.

A minha orientadora Úrsula por ter acreditado na minha proposta, mesmo quando eu não estava tão confiante assim. Você fez a total diferença nessa etapa final.

A minha noiva Deiziane que tem sido minha rocha, minha paz e minha fonte de inspiração.

RESUMO

A procura por robôs móveis que realizam a tarefa de transporte autônomo tem crescido nos institutos de pesquisa, como o Polo de Inovação Manaus/IFAM, e a tecnologia para sua fabricação, configuração e implementação em ambiente industrial abrange diversos problemas da atualidade e do próximo futuro que a robótica se propõe a resolver, problemas esses que por vezes parecem fáceis para seres humanos, como andar, se comunicar e registrar o ambiente à sua volta por meio de mapas. Tais tarefas requerem a integração de atuadores, sensores e sistemas voltados para a odometria e elaboração de registros cartográficos, os desafios aumentam quando o robô em questão incumbido pelo mapeamento deve trabalhar em um ambiente dinâmico, ou seja, onde obstáculos, que devem ser evitados, podem aparecer, sumir ou se movimentar pelo mapa. Um dos *frameworks*[1] desenvolvidos com o objetivo de alcançar essas funcionalidades é o ROS (*Robot Operating System*[2]) que disponibiliza uma plataforma para a criação de código, em Python e em C++, e de ferramentas para serem usadas em sistemas robóticos em geral, com destaque em aplicações como AGVs (*Automated Guided Vehicle*[3]) industriais, VTNTs (Veículo Terrestre Não Tripulado) militares, veículos de resposta emergencial e exploradores espaciais como ROVER MARS 2020, e talvez mais importante, para compartilhamento das soluções entre os mais diversos tipos de robôs. Este trabalho apresenta a construção de um robô teleoperado com a capacidade de mapear ambientes a partir de um sensor *LiDAR (Light Detection and Ranging*[4]*)* usando algoritmo HECTOR SLAM e empregando a plataforma ROS como o sistema robótico responsável por sincronizar sensores, atuadores e algoritmos. Embora entregue resultados satisfatórios, a plataforma robótica desenvolvida encontra dificuldades na geração de odometria devido a problemas na leitura dos sensores encoders e no escorregamento das rodas em terreno liso.

Palavras-Chaves: ROS. HECTOR-SLAM. Mapeamento. VTNT (VEÍCULO TERRESTRE NÃO TRIPULADO).

[1] Em uma tradução livre: *Software* de Estrutura
[2] Em uma tradução livre: Sistema Operacional Robótico
[3] Em uma tradução livre: Veículo Autônomo Guiado.
[4] Em uma tradução livre: Detecção e Medição Remota a Base de Luz.

ABSTRACT

The demand for mobile robots that perform the task of autonomous transport has grown in research institutes, such as Innovation Hub/IFAM, and the technology for their manufacture, configuration and implementation in an industrial environment covers several, current and future, problems that robotics proposes to solve. These problems sometimes seem easy for human beings, such as walking, communicating and recording the environment around them through maps. Such tasks require the integration of actuators, sensors and systems focused on odometry and creation of cartographic records. The challenges increase when the robot in question responsible for mapping must work in a dynamic environment, that is, where obstacles, which must be avoided, can appear, disappear or move around the map. One framework developed aiming to achieving these functionalities is ROS (Robot Operating System) which provides a platform for creating code, in Python and C++, and tools to be used in robotics systems in general, with emphasis on applications such as industrial AGVs (Automated Guided Vehicles), military UGV (Unmanned Ground Vehicle), emergency response vehicles and space explores such as ROVER MARS 2020, and perhaps most importantly, for sharing solutions between the most diverse types of robots. This work details the construction of a teleoperated robot with the ability to map indoor environments using a LiDAR (Light Detection and Ranging) sensor combined with HECTOR SLAM in the ROS platform, as the robotic system responsible for synchronizing sensors, actuators and algorithms. Although delivering satisfactory results, the developed robotic platform encounters difficulties in generating odometry due to problems reading the encoder sensors and wheel slippage on smooth terrain.

Key-words: ROS. HECTOR-SLAM. Mapping. UGV (Unmanned Ground Vehicle).

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ACRÔNIMOS

AGV	*Automated Guided Vehicle* (Veículo Autônomo Guiado)
AMR	*Autonomous Mobile Robot* (Robô Móvel Autônomo)
CAD	*Computer Aided Design* (Desenho Assistido por Computador)
CIR	Centro Instantâneo de Rotação
GPS	*Global Positioning System* (Sistema Global de Posicionamento)
IFR	*International Federation of Robots* (Federação Internacional de Robótica)
IoT	*Internet of Things* (Internet das Coisas)
IP	*Internet Protocol* (Protocolo de Internet)
ISO	*International Organization for Standardization* (Organização Internacional de Normalização)
LiDAR	*Light Detection and Ranging* (Detecção e Medição Remota a Base de Luz)
OBP	*Online Browsing plataforma* (Plataforma de Navegação Online)
OS	Sistema Operacional
PD&I	Pesquisa, Desenvolvimento e Inovação
PWM	*Pulse Width Modulation* (Modulação da Largura de Pulso)
RAM	Random-Access Memory (Memória de Acesso Aleatório)
ROS	*Robot Operating System* (Sistema Operacional Robótico)
RPM	Rotações Por Minuto
SLAM	*Simultaneous Localization and Mapping* (Mapeamento e Localização Simultânea)
VAC	Voltagem em Corrente Alternada
VDC	Voltagem em Corrente Direta
VTNT	Veículo Terrestre Não Tripulado
WLAN	*Wireless LAN* (Rede de computadores sem fio)

SUMÁRIO

1 INTRODUÇÃO

Robôs de mobilidade autônoma têm tido um maior desenvolvimento na última década e a subdivisão da robótica em questão, navegação autônoma, encontra aplicabilidade em quase todos os setores da atividade econômica. A capacidade do robô em reconhecer sua posição em relação aos limites do ambiente, obstáculos e alvos torna-se crucial para soluções diversas, abrangendo transporte, exploração, resgate, patrulhamento, limpeza e rastreamento (HUANG; SAVKIN; NI, 2021, p. 10). A importância dessas soluções é evidenciada pelos diversos modelos de robôs encontrados nos ambientes industriais, nos campos de testes militares e na exploração espacial.

Atualmente, destaca-se o desenvolvimento de soluções robóticas no âmbito da navegação autônoma está focada em cenários no qual o robô opera em ambiente fechado, onde outras formas de localização, como, por exemplo, o GPS, não se fazem úteis ou pouco efetivas em realizar a tarefa de determinar a posição relativa do robô. Visto que a tecnologia GPS comercialmente disponível não entrega precisão para ambientes internos e mesmo que o sinal esteja presente, esse não se faz útil quando há obstáculos com poucos centímetros de diâmetro. A problemática da elaboração de mapas de ambientes internos e a localização dentro deles é complexa e tem sido alvo de estudo de roboticistas e cientistas da computação por cerca de trinta anos (DURRANT-WHYTE; BAILEY, 2006, p. 107). Nesse contexto, algoritmos SLAM (*Simultaneous Localization and Mapping*[5]) apresentam soluções com alto grau de sucesso (NAGLA, 2020, p.3), cujo aspecto mais importante dessas soluções, sejam sua natureza *open-source*[6]*,* possibilitando a aplicabilidade em robôs de diferentes naturezas e funções.

1.1 JUSTIFICATIVA

Os fatores principais que justificam este trabalho são os seguintes:

- Destaque dos Sistemas Robóticos Teleoperados: Em meio ao atual avanço das soluções de transporte e patrulhamento, os sistemas robóticos teleoperados assumem um papel proeminente. Uma prova de conceito envolvendo um robô móvel teleoperado, impulsionado

[5] Em uma tradução livre: Mapeamento e Localização Simultânea.
[6] Em uma tradução livre: Código Livre.

por ferramentas ROS, oferece *insights* sobre os conhecimentos, materiais e ferramentas essenciais para tal empreendimento;

- Implementação Facilitada com ROS na Plataforma *Raspberry Pi*: A praticidade na implementação de soluções computacionais utilizando o ROS na plataforma *Raspberry Pi* é um ponto relevante. Em exemplos comerciais, robôs móveis destinados à exploração, resgate ou veículos de resposta emergencial frequentemente empregam hardwares dispendiosos e sofisticados. Entretanto, ao adotar o sistema ROS, algoritmos desenvolvidos para hardwares mais robustos podem ser adaptados para operar em robôs de menor porte e compostos por hardwares mais acessíveis;
- Elevados Custos no Desenvolvimento Robótico: O alto custo associado ao desenvolvimento robótico é um desafio notável. Exemplos provenientes de contextos comerciais, militares e acadêmicos revelam que plataformas robóticas móveis frequentemente empregam materiais e sensores onerosos, resultando em robôs cuja manutenção é exclusivamente responsabilidade dos desenvolvedores. A proposta de utilizar sensores *plug-and-play*[7], hardwares acessíveis e chassis de kits direcionados a estudantes emerge como uma alternativa viável, propiciando o desenvolvimento de soluções de baixo custo e facilitando a disseminação do conhecimento necessário

1.2 MOTIVAÇÃO

A necessidade de se reproduzir ações humanas por meio de máquinas impulsionou o desenvolvimento de robôs móveis autônomos, ao passo que a robótica móvel vem expandindo sua presença em ambientes industriais, pois em contraste a estações robóticas fixas, um robô móvel tem a capacidade de realizar tarefas combinado ao fato de não precisar ficar fixo a um local e locomover-se onde se fizer necessário (SIEGWART; NOURBAKHSH, 2004, p. 2). No desenvolvimento de soluções robóticas, os algoritmos SLAM assumem o protagonismo na navegação autônoma, pois demonstram capacidade de executar específicas tarefas de forma independente com a adicional dificuldade de agir em condições ambientais variadas, sendo cada

[7] Em uma tradução livre: Conectar e usar.

vez mais usados em robôs. Segundo ABIresearch (2019), em 2030 robôs autônomos funcionando com algum tipo de SLAM excederá a casa de 15 milhões.

Visto isso, o desenvolvimento de robôs móveis industriais tem se tornado cada vez mais importante em institutos de pesquisa como o Polo de Inovação Manaus / IFAM e com isso o estudo de sistemas como o ROS, mundialmente difundido na implementação de funcionalidades em robôs, como navegação, localização, aquisição de dados e manipulação de objetos via apêndices robóticos, também ganha protagonismo na prototipagem de soluções em automação.

1.3 PROBLEMÁTICA

A demanda por construir AGVs e AMRs (Robôs Móveis Autônomos) destinados a estabelecer uma base robótica para aplicações industriais é significativa nos institutos de pesquisa. O sistema robótico de preferência para a maioria dos profissionais nesse campo é o ROS, um *software* que possibilita a integração de atuadores e sensores em um robô. Assim como, possibilita a utilização de soluções e *scripts* originalmente feitos para outros desafios, o que concede ao sistema robótico uma grande maleabilidade. Contudo, vale salientar que este sistema ainda é pouco difundido no ambiente acadêmico do IFAM - Campus Manaus Distrito Industrial. Portanto, na atual conjuntura a construção de um robô móvel utilizando um kit de robótica acessível, com modificações que incluem a adição de peças personalizadas e manufaturadas por impressoras 3D serve como uma Prova de Conceito (POC) para a difusão da robótica no ambiente acadêmico.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Gerar mapas de ambientes internos bidimensionais a partir do desenvolvimento de um robô móvel do tipo *Skid-Steer* com capacidade de ser teleoperado e criar um sistema robótico que possibilite a integração de seus atuadores e sensores.

1.3.2 Objetivos Específicos

a. Desenvolver um robô móvel do tipo *Skid-Steer*, realizar a modelagem cinemática simétrica para a geração de fórmulas cinemáticas que preveem o seu deslocamento e projetar peças para a organização do *hardware* necessário.
b. Elaborar mapas por meio da aquisição de dados provenientes do sensor *LiDAR*.
c. Integrar sensores e atuadores e operar o robô de forma remota.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está estruturado conforme os tópicos a seguir, com uma breve descrição dos capítulos.

- **Fundamentação Teórica:** Este tópico refere-se ao capítulo 2 deste trabalho, que aborda os conceitos e definições relacionados aos componentes eletrônicos, mecânicos, hardwares, sistemas operacionais, softwares utilizados e algoritmos implementados;
- **Metodologia:** Este tópico refere-se ao capítulo 3, aqui são apresentados os passos realizados na construção do sistema robótico, abordando o aspecto mecânico e eletrônico, e de *Firmware* assim como as ferramentas usadas na construção do projeto.
- **Resultados:** Este tópico refere-se ao capítulo 4, onde serão apresentados os resultados obtidos referente ao método utilizado no capítulo anterior;
- **Conclusão e Trabalhos Futuros:** Neste tópico o trabalho escrito é finalizado, apresentando discussões pertinentes quanto ao que foi entregue (ou não entregue) e as possibilidades de melhorias futuras. Além de abordar pontos-chave durante o desenvolvimento do trabalho e um balanço das tecnologias usadas.

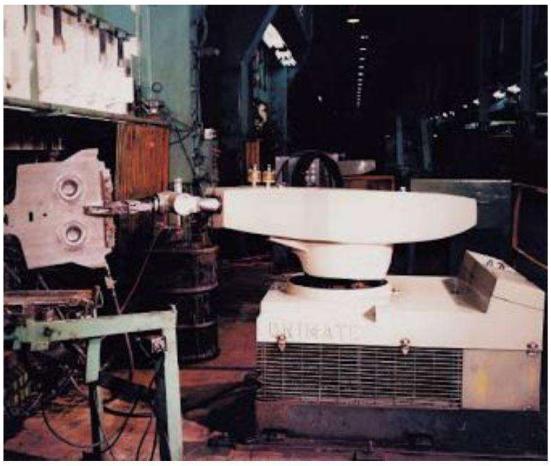
2 FUNDAMENTAÇÃO TEÓRICA

Neste trabalho a fundamentação teórica expõe uma breve apresentação da robótica, alguns conceitos importantes da navegação autônoma, a busca pela solução do problema SLAM e as ferramentas computacionais usadas na elaboração do protótipo.

2.1 ROBÔS MÓVEIS

Embora o termo robô tenha sido inventado décadas antes, foi no início dos anos 1960 que os empresários americanos George C. Devol e Joseph F. Engelberger apresentaram ao mundo o primeiro robô industrial, ilustrado na **Figura 1**, o Unimate, desenvolvido para a manipulação de materiais e instalado na fábrica de fundição em New Jersey, Estados Unidos, da empresa General Motors (CORKE, 2017, p. 2).

Figura 1 – Unimate, o primeiro robô industrial.



Fonte: Corke (2017, p. 2).

De acordo com a definição apresentada pela ISO-8373 (2021) - *International Organization for Standardization*[8] - um robô é "um mecanismo atuador programado com certo grau de

[8] Em uma tradução livre: Organização Internacional de Normalização.

autonomia para realizar locomoção, manipulação ou posicionamento, exemplos desses mecanismos são robôs manipuladores, plataformas móveis e robôs vestíveis".

Quando o foco são exemplares da robótica móvel, a definição adotada pela IFR (2021) - *International Federation of Robotics*[9] - é de que "Um robô móvel é um 'robô capaz de se locomover por conta própria'. Um robô móvel pode ser uma plataforma móvel com ou sem manipuladores. Podendo incluir ter a capacidade de operação autônoma, um robô móvel pode ter meios de ser controlado remotamente.".

2.1.1 Odometria por *Dead-Reckoning*[10]

A necessidade de se registrar o espaço percorrido é essencial no desenvolvimento de qualquer solução no ramo da robótica móvel, visto que a localização de dispositivos em um ambiente, conhecido ou não, depende diretamente da estimação do deslocamento realizado. Para atender essa necessidade, a odometria é adotada na maioria dos casos. De forma simplificada, a odometria está presente na vida da maioria das pessoas por meio do odômetro, dispositivo que computa e indica, no painel dos veículos automotivos, a distância (popularmente dita quilometragem) percorrida pelo carro (COTA, 2019, p. 12).

A posição e orientação, que na robótica móvel é chamada de "*Pose*", pode ser determinada por meio da contagem de giros realizados pelas rodas. Esse método é chamado de *Dead-Reckoning*, e embora tenha se mostrado adequado somente para curtas trajetórias (LAGES, 2008, p. 6), ainda é uma das formas mais simples e diretas de estimação de deslocamento de robôs móveis. Sua implementação é facilitada quando a fonte de contagem de giros vem de Encoders instalados diretamente nos eixos das rodas. O processo de transformar a contagem dos pulsos gerados pelo encoder em odometria se dá, primeiramente, pela determinação da velocidade angular de cada roda (v). Essa etapa requer uma restrição de tempo, chamada de delta tempo, onde as contagens de pulsos vindos do encoder possam ser consideradas (p_{enc}). Partindo do pressuposto que se sabe a quantidade total de pulsos gerados pelo encoder em uma volta completa, revolução, (p_{rev}) a fórmula da velocidade angular resume-se em dividir os pulsos gerados no delta tempo (p_{enc}) pelos pulsos de uma revolução (p_{rev}) e multiplicar pelo comprimento total deslocado pela roda ao

[9] Em uma tradução livre: Federação Internacional de Robótica.

[10] Em uma tradução livre: Reconhecimento às cegas.

longo da superfície de contato durante uma revolução, circunferência da roda, isto é, duas vezes pi vezes o raio da roda (r). O resultado pode ser visto na equação (1):

$$v = 2.\pi.r\left(\frac{p_{enc}}{p_{rev}}\right) \quad \text{[radianos/segundo]} \tag{1}$$

2.1.2 Modelo Cinemático

O modelo cinemático de um robô leva em consideração as restrições de movimento de locomoção e descreve o seu comportamento. Essas restrições são conhecidas como não holonômicas, ou seja, não integráveis, pois não podem ser expressas em derivadas de tempo em uma função generalizada de coordenadas (KOLMANOVSKY; MCCLAMROCH, 1995, p. 20).

As equações matemáticas geradas pelo modelo cinemático de um robô móvel são utilizadas pelos sistemas de controle com o objetivo de fornecer uma tradução entre comandos computacionais e a movimentação individual de cada mecanismo responsável pela locomoção do robô, e consequentemente, simplificando a sua teleoperação.

2.1.2.1 Modelo Diferencial

Esse modelo cinemático descreve um robô com duas rodas montadas em um eixo comum e impulsionadas por motores independentes, uma das mais simples e mais usadas configurações na construção de robôs móveis (SALEM, 2013, p. 253). Para fins de alcançar um modelo matemático que descreva o robô móvel diferencial, assume-se que o robô móvel diferencial possui um corpo rígido e as rodas não escorregam.

A *Pose* desse tipo de robô pode ser expressa de forma resumida em x, y e θ, em relação a um plano 2D como visto na **Figura 2**, e se dá por meio de características básicas na construção do robô, como distância entre eixos (D) e raio das rodas (r), resultando no seguinte cálculo:

Primeiramente é necessário ter as velocidades lineares de cada lado do robô, esquerda (V_l) e direita (V_r), encontradas a partir da multiplicação das velocidades angulares de cada roda com o raio das mesmas, como pode ser visto nas equações (2) e (3). O processo de obtenção das

velocidades angulares, da roda esquerda (v_l) e da roda direita (v_r), se dá pelo uso da equação (1) em cada uma das rodas. Em seguida, na equação (4), é possível calcular a velocidade linear do robô e a velocidade em que a sua orientação em torno do eixo z muda, equação (5), em função das velocidades individuais de cada lado. Onde a média das velocidades de cada lado resulta na velocidade linear total (v_k) e a velocidade angular (ω_k), ou seja, a taxa de variação da sua orientação em torno do eixo z muda, é alcançada através da diferença das velocidades individuais dividida pela distância entre as rodas (D), o detalhe a ser notado é que a subtração das velocidades é da direita para a esquerda seguindo assim um valor crescente no sentido anti-horário.

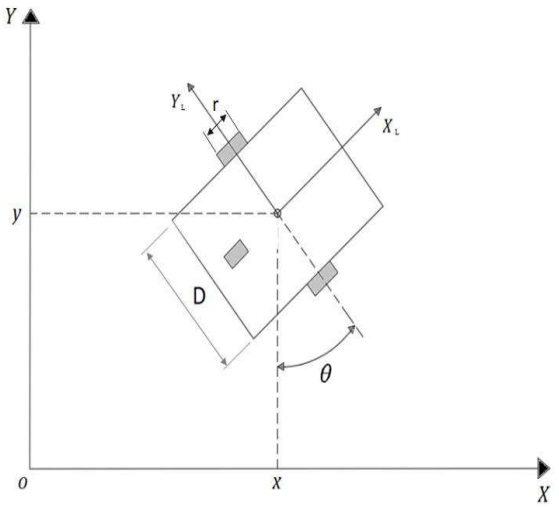
$$V_l = v_l . r \quad \text{[metros/segundo]} \tag{2}$$

$$V_r = v_r . r \quad \text{[metros/segundo]} \tag{3}$$

$$v_k = \frac{V_r + V_l}{2} \quad \text{[metros/segundo]} \tag{4}$$

$$\omega_k = \frac{V_r - V_l}{D} \quad \text{[radianos/segundo]} \tag{5}$$

Figura 2 – Relação da *Pose* do Robô diferencial com o plano 2D.



Fonte: Modificado de Rodrigues (2019, p. 19).

A partir dessas equações, de natureza instantânea, pode-se estimar o deslocamento realizado pelo robô (ΔS) e a variação no ângulo de orientação do robô (Δθ), contanto que seja acrescentado uma restrição de tempo, uma duração, aqui chamada de T_e, como mostrado nas equações (6) e (7), respectivamente.

$$\Delta S = v_k . T_e \quad \text{[metros]} \tag{6}$$

$$\Delta \theta = \omega_k . T_e \quad \text{[radianos]} \tag{7}$$

A *Pose* final do robô, posição expressa com x e y e orientação expressa com θ, é determinada pela estimativa feita no tempo percorrido, T_e, e somada com a *Pose* anterior registrada (x_{ant}, y_{ant} e θ_{ant}), isso é descrito pelas equações (8), (9) e (10):

$$x = x_{ant} + \Delta S . cos\left(\theta_{ant} + \frac{\Delta \theta}{2}\right) \quad \text{[metros]} \tag{8}$$

$$y = y_{ant} + \Delta S . sen\left(\theta_{ant} + \frac{\Delta \theta}{2}\right) \quad \text{[metros]} \tag{9}$$

$$\theta = \theta_{ant} + \Delta \theta \quad \text{[radianos]} \tag{10}$$

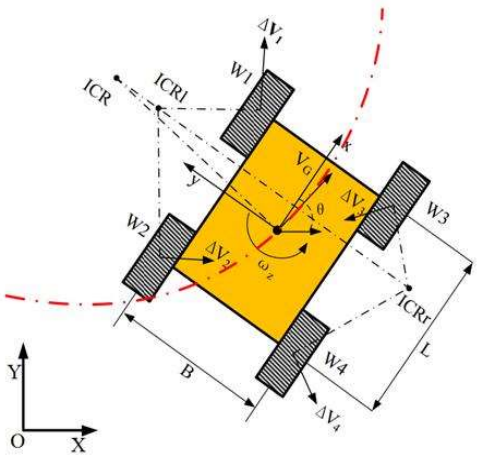
2.1.2.2 Modelo *Skid-Steer*

Robôs de configuração *Skid-Steer* mostram-se cada vez mais presentes em aplicações robóticas de campo como: navegação de terrenos, exploração, gestão de resíduos, defesa, segurança e serviços domésticos (YI et al., 2007, p 2605). Caracterizado por não possuir um sistema de direção, assim como os robôs diferenciais, robôs móveis do tipo *Skid-Steer* realizam suas manobras ao controlar as velocidades relativas do lado esquerdo e do lado direito do seu chassi. Além disso, sua robustez possibilita suportar maiores cargas em comparação com um robô

diferencial, e a capacidade de giro com raio zero tornam o modelo *Skid-Steer* excelente para a robótica teleoperada.

Na **Figura 3** tem-se o esquemático *Skid-Steer* em relação a um plano 2D, onde o *Pose* do robô é expresso em relação com a origem global (X, Y). As dinâmicas desse tipo de robô são complexas devido ao excesso de derrapagem que possa acontecer e a deformação do terreno durante o movimento (PENTZER; BRENNAN; REICHARD, 2014, p. 455).

Figura 3 – Coordenadas do frame local (x, y e ω_z)
em relação à um frame global (X e Y).



Fonte: Wu et al. (2013, p. 269).

Os esforços para a determinação de um modelo cinemático para robôs do tipo *Skid-Steer* têm se concentrado no estudo do CIR, Centro Instantâneo de Rotação, esses por sua vez, podem ser determinados usando uma correlação com os pontos de contato entre as rodas do robô e o solo. Essa abordagem pode ser vista na publicação de Martinez et al. (2005, p. 869), que trabalhando com rodas do tipo esteira determinam que: "Coordenadas locais do veículo e a CIR das suas esteiras podem ser obtidas geometricamente como funções das velocidades angulares e translacionais do próprio veículo". Isso posto, a principal consequência é a dinâmica veicular resultante no modelo cinemático, sendo determinada por apenas duas coordenadas: o CIR da esteira esquerda e o CIR da esteira direita.

Uma das propostas de construção de um robô *Skid-Steer* com esteiras seria então que os pontos CIRs coincidam com os pontos de contato das esteiras com o solo, pois dessa forma as

equações cinemáticas seriam iguais a de um robô diferencial ideal, unindo as vantagens mecânicas do anterior com a simplicidade cinemática do posterior. Garantindo que o robô se movimenta em velocidades moderadas e que o solo fosse rígido, plano e uniforme, foi possível incorporar a análise cinemática por CIRs em robôs *Skid-Steer* com rodas e gerar resultados satisfatórios, em até 30 metros, levando em consideração a precisão entre previsão matemática e medições no mundo real (MANDOW et al., 2007, p. 1227).

Visto os fatores que podem dificultar uma descrição matemática do funcionamento do robô, aliado às restrições anteriores, Wang et al. (2015, p. 9683) determina premissas a fim de alcançar equações simples e aproximadas da realidade da locomoção *Skid-Steer*, usando dos ICRs, elas são:

1. O centro de massa da plataforma móvel, e de seus apêndices, está localizado no centro geométrico do chassi;
2. As rodas de mesmo lado giram na mesma velocidade;
3. O robô está andando em uma superfície firme e todas as rodas estão sempre em contato com o chão.

Obedecendo a esses pré-requisitos, é possível descrever a velocidade do robô quanto ao seu frame local em velocidade de translação (v_x e v_y) e a sua velocidade angular (w_z). Assim como no caso do robô diferencial ideal, essas velocidades podem ser determinadas a partir das velocidades laterais individuais, isto é, V_L e V_R. No entanto a relação entre velocidades laterais individuais e a locomoção do frame do robô é uma matriz 3x2, equação (11), aqui chamada de A, de coeficientes constantes uma vez que os CIRs do robô são aproximadamente constantes quando a plataforma robótica está em movimento sobre um terreno sem variações na rigidez e na planicidade (WANG et al., 2018, p. 1). Seguindo esses estudos, Boas e Conceição (2020, p. 3) realizaram a modelagem cinemática do robô móvel comercial "*Husky*" da empresa Clearpath Robotics, ilustrado na **Figura 4**, correlacionando simulações com experimentos feitos em laboratório, encontrando a Matriz A, equação (12), pelo modelo cinemático simétrico, ou seja, os pontos do CIR do lado esquerdo e direito estão simetricamente dispostos em relação ao eixo Y, levando em consideração nas equações apenas o ponto CIR do eixo X.

$$\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix} = A \begin{pmatrix} V_l \\ V_r \end{pmatrix} \tag{11}$$

$$A = \frac{\alpha}{2x_{CIR}} . \begin{bmatrix} 0 & 0 \\ x_{CIR} & -x_{CIR} \\ -1 & 1 \end{bmatrix} \tag{12}$$

Figura 4 – AGV Husky, plataforma robótica do tipo *Skid-Steer*.

Fonte: ClearPath Robotics (2011).

Neste modelo cinemático existem dois parâmetros, o linear, representado pela letra grega α, e o rotacional, representado pelo x_{CIR}, resumindo a modelagem de um robô *Skid-Steer*, uma vez complexa e trabalhosa, em um processo simplificado de identificação de dois parâmetros via o método offline. O método de identificação de parâmetros offline consiste em realizar testes de movimentação para cada um deles:

1. Identificação offline do parâmetro α: Realizar a movimentação do robô em linha reta, registrar a distância percorrida (d), as velocidades individuais de cada lado (V_l e V_r) e o tempo de teste (t) e aplicar os valores na equação (13):

$$\alpha = \frac{2.D}{\int_0^t V_r dt + \int_0^t V_l dt} \tag{13}$$

2. Identificação offline do parâmetro x_{CIR}: Realizar um giro do robô em volta do eixo Z, registrar a diferença de ângulo do robô (θ), as velocidades individuais durante a manobra (V_l e V_r), o tempo de teste (t) e aplicar os valores na equação (14):

$$x_{CIR} = \frac{\int_0^t V_r dt - \int_0^t V_l dt}{2.\theta} \tag{14}$$

Após a identificação dos parâmetros da equação (12), a plataforma robótica ganha a capacidade de geração de odometria a partir da leitura das velocidades lineares de cada lado do robô (V_l, V_r) e a nova *Pose* do robô é obtida pela somatória de sua *Pose* anterior (x_{ant}, y_{ant} e θ_{ant}) e dos novos deslocamentos em x, y e orientação em torno do eixo z, expressa com θ, respectivamente. Esses deslocamentos são alcançados ao multiplicar o tempo registrado, aqui também chamado de T_e, em que o robô esteve sob o efeito das velocidades instantâneas e, uma vez que a abordagem do modelo simétrico tem o objetivo de permitir aplicar ao robô as equações do modelo diferencial, essas velocidades podem ser encontradas pelas fórmulas (8), (9) e (10), com a diferença no cálculo da velocidade linear total, agora encontrado pela equação (15), e pelo cálculo da velocidade angular, agora resumido na equação (16).

$$v_k = \frac{\alpha(V_l - V_r)}{2} \quad \text{[metros/segundo]} \tag{15}$$

$$\omega_k = \frac{\alpha(V_r - V_l)}{2x_{CIR}} \quad \text{[radianos/segundo]} \tag{16}$$

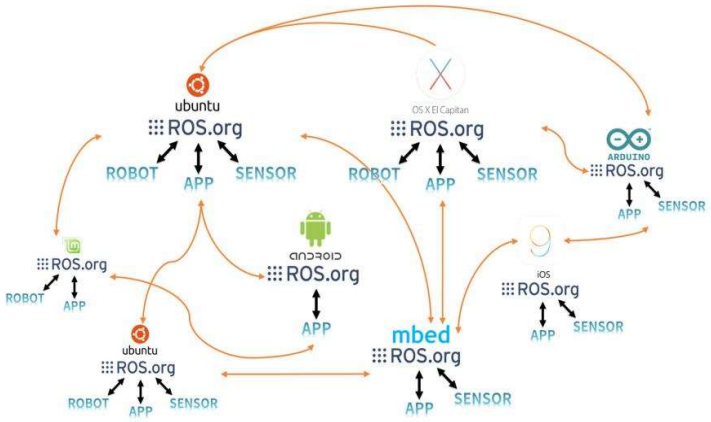
2.2 ROS: SISTEMA OPERACIONAL ROBÓTICO

ROS é um conjunto de bibliotecas de *software* que juntas proporcionam ferramentas para aplicações na robótica e funcionam de tal maneira que podem ser chamadas de um sistema meta-operacional, ou seja, ROS atua de forma semelhante a um OS (*Operating System*[11]), como Linux

[11] Em uma tradução livre: Sistema Operacional

e o Windows, pois possui camadas de abstração de *hardware*, mas diferentemente de um OS o ROS é capaz de implementar aplicações de forma combinada entre diferentes hardwares (ROS: ROBOT OPERATING SYSTEM, 2023).

Figura 5 – Comunicação ROS entre diferentes sistemas.



Fonte: Pyo et al. (2017, p. 12).

Para poder fazer uso do ROS, um sistema operacional como o Ubuntu, distribuído pelo Linux, deve ser instalado no *hardware* em questão, e em seguida o pacote de bibliotecas do ROS também deverá ser instalado. Nessa configuração o ROS usa das funções do próprio Linux e complementa-o com a capacidade de transmitir e receber informações entre hardwares diferentes que estejam usando ferramentas ROS como registro e administração de erros, monitoramento de dados e criação de camadas virtuais entre aplicações. Na prática, a interface de programação é próprio terminal do OS anfitrião, fato que leva ao ROS também ser chamado de *middleware*[12] ou mesmo *software framework* (PYO et al., 2017, p. 11). **Na Figura 5** tem-se um exemplo de como a comunicação de dados ROS acontece não somente por um sistema operacional, mas por outros sistemas que estejam usando de suas ferramentas como também outros hardwares, programas e APIs resultando em uma rede potencialmente diversa e abrangente. O ecossistema ROS permite o uso de ferramentas para plotagem de gráficos, visualização de mapas, capacidade de reproduzir condições específicas do sistema como fluxo de dados e simulação de eventos, no entanto essas funcionalidades podem ou não serem incluídas durante a instalação do ROS e podendo ainda serem

[12] Em uma tradução livre: Intermediário

baixadas e deletadas conforme a necessidade do usuário, permitindo um melhor uso dos recursos computacionais do *hardware* empregado durante e depois do desenvolvimento das soluções robóticas.

2.2.1 Origem do ROS

A origem dessa tecnologia se deu em 2006 quando dois estudantes de PhD da universidade de Standford na Califórnia, Keenan Wyrobek e Eric Berger, pesquisando sobre aplicações na área da robótica perceberam que não havia um sistema unificado e flexível que permitisse a construção de soluções de fácil desenvolvimento e compartilhamento. Com o escopo deste *software* montado, a incubadora Willow Garage decidiu investir na ideia e o protótipo do *software framework* foi feito junto do protótipo de um robô físico que rodaria o sistema (WYROBEK, 2017). Esse protótipo tinha a ideia de ser um "*Personal Robot*[13]", ou seja, um robô que pudesse atuar em ambiente residencial e realizar tarefas domésticas como arrumar a casa, chamado de PR1, ilustrado na **Figura 6**, esse protótipo ficou conhecido por ser propenso a quebras e ser lento, no entanto, o funcionamento do sistema ROS foi um sucesso.

Figura 6 – PR1: Primeiro robô pessoal a ser construído com o sistema ROS.

Fonte: Wyrobek (2017)

Figura 7 – Os 11 PR2 enviados as universidades e a equipe que os desenvolveu.

Fonte: Ackerman (2017)

Com todas as funcionalidades que configuram a maleabilidade da ferramenta ROS construídas e funcionando, o PR2 foi desenvolvido, **Figura 7**, e 11 unidades foram enviadas para

[13] Em uma tradução livre: Robô pessoal.

universidades dos estados unidos, dando início a difusão desse sistema no meio acadêmico e por consequência no ambiente *R&D*[14] de empresas voltadas a soluções robóticas.

2.2.2 Filosofia *open-source* do ROS

As soluções robóticas que se utilizam do ROS funcionam basicamente como uma estrutura de pequenos programas, "Nós", que rapidamente comunicam mensagens entre si via canais chamados "Tópicos". O particular funcionamento do ROS encoraja o reuso de softwares robóticos que estejam fora do *hardware* do robô. Essa característica do sistema de ser voltada para o reuso de recursos e códigos é alinhada com a filosofia *open-source* onde Oliveira (2019, p. 18) afirma que a intenção é facilitar o desenvolvimento de projetos robóticos e possibilitar a construção de robôs customizados, de acordo com a necessidade do projetista. Seguindo esse pensamento, os "Pacotes ROS" foram criados, esses consistem de uma coleção de "Nós" e "Parâmetros ROS" que funcionam de forma integrada e coesa, sem abrir mão da capacidade de se conectar com outros "Nós" e outros "Tópicos" fora do próprio "Pacote".

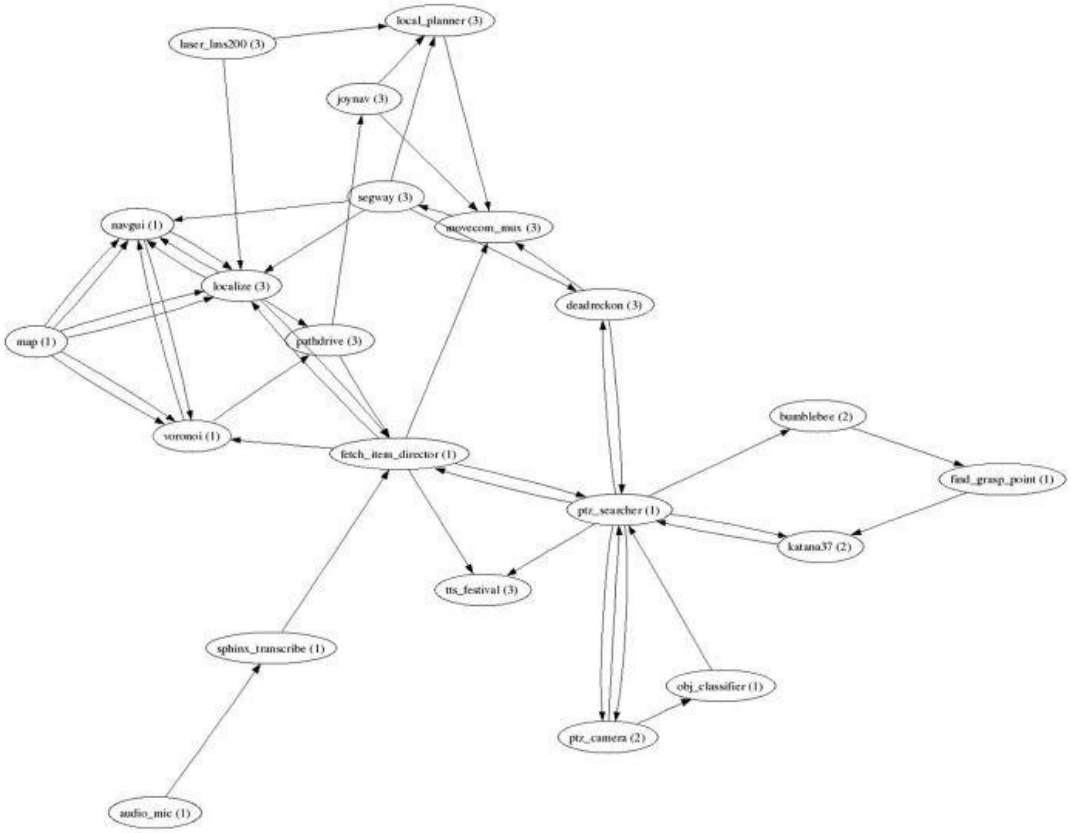
Ao iniciar uma aplicação ROS um "Nó" mestre é gerado cuja a função é registrar todos os posteriores "Nós" quando esses forem criados e incluídos na "Rede ROS", e para isso, estabelece "Parâmetros" de rede usados para padronizar a comunicação entre os "Nós" independente de qual *hardware* este esteja rodando. Os "Nós" são gerados por pequenos pedaços de programação, *scripts*, que por sua vez se subscrevem, "escutam", ou publicam, "falam" em Tópicos e essa conversa de "falar" e "escutar" se dá pela transmissão de Mensagens ROS cujo o formato é predeterminado e estruturado a fim de que dados possam ser transmitidos Nós, de forma segura e confiável. Uma aplicação robótica pode ser composta de diversos Nós, cada um responsável por uma tarefa em específico como a leitura de um sensor ou o acionamento de um motor, que juntos trabalham para um único objetivo, como por exemplo, movimentar um apêndice. A rede ROS desse exemplo pode ser vista na representação gráfica chamada de ROS *Graph*, **Figura 8,** onde os Nós, no formato oblongo, se comunicam via Tópicos, retângulos intercalados por setas.

O maior benefício desse tipo de estrutura está na capacidade de testar soluções em um protótipo ao simplesmente "desligar" um Nó dessa rede ou substituí-lo por outro com uma programação diferente e assim verificar o impacto no resultado final. Peças robóticas podem ser

[14] Em uma tradução livre: Pesquisa e Desenvolvimento.

substituídas por simulações, subsistemas de navegação podem ser trocados, algoritmos podem ser ajustados e recompilados e assim por diante. Visto que o ROS cria o pano de fundo da rede à medida que as mudanças são feitas, o sistema como um todo é feito para facilitar a interatividade e a experimentação (QUIGLEY; GERKEY; SMART, 2005, p. 11).

Figura 8 – ROS *Graph* mostrando os Nós e Tópicos de uma aplicação robótica.



Fonte: Quingley; Gerkey; Smart (2015, p. 10)

Por último, uma das ferramentas mais úteis do sistema ROS é o arquivo “launch” que facilita a implementação de vários Nós ao mesmo tempo. Esse arquivo funciona como uma “chamada” de *scripts*, ou seja, quando o arquivo launch é executado, todos os *scripts*, incluindo outros arquivos launch, listados na sua programação são implementados na Rede ROS um de cada vez. Isso permite a implementação de diferentes funções de um sistema robótico, haja visto que esses geralmente são compostos de mais de um Nó e, por consequência, mais de um script precisa ser executado. Uma vez que o arquivo Launch é “lançado”, esse já se ocupa da criação do Nó

Mestre e, portanto, já criando o pano de fundo para a adição de novos Nós e Tópicos. Essa funcionalidade é mostrada de forma clara no exemplo de Sear-Collings (2019), englobando desde a criação das pastas dentro do sistema operacional até o "lançamento" de nós gerados por *scripts* diferentes que "conversam" entre si na mesma rede ROS.

2.2.3 Biblioteca TF: Transformação da árvore de quadros

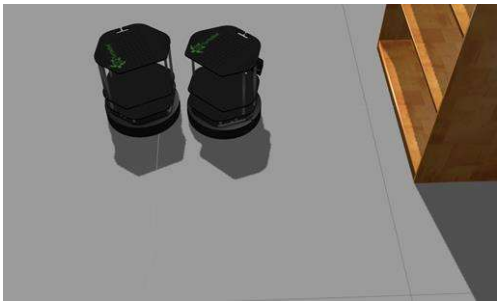
A transformação da árvore de quadros, ou como ela é conhecida originalmente TF (*Transforms*[15]), é uma biblioteca desenvolvida pela *Open Source Robotics Fundation* que cuida de um problema muito importante na robótica móvel: rastrear e registrar a relação (transformação) de várias partes de um sistema robótico (quadros de coordenadas) e o mundo a sua volta ao longo do tempo (FOOTE, 2019). Quando implementada ela disponibiliza uma comunicação que pode assumir duas formas, "Comunicador" ou "Ouvinte", sobre a localização do robô, suas rodas, seus apêndices, juntas, links e qualquer outra parte robótica que for de interesse em um sistema robótico. Essa biblioteca incorpora as limitações físicas do robô e de suas partes e consegue rastrear onde cada um desses elementos se encontra em relação ao restante do sistema robótico e em relação a um mapa, ou ao mundo ("*/world*") nos casos de simulações, a **Figura 9** demonstra como dois robôs, turtle1 e turtle2, se apresentam na simulação e na **Figura 10** é ilustrado a representação gráfica da árvore de quadros que liga ambos os robôs e sua relação com o mundo simulado, ao passo que mostra os fatores usados na matemática necessária para calcular essas relações.

A árvore de quadros é atualizada toda vez que um dos seus itens, como no exemplo robô móvel da **Figura 9** "*turtle1*", se locomove, pois assim ele gera uma modificação na sua posição e orientação, essa modificação é atualizada e o restante dos itens que fazem parte da Rede ROS, seguindo o exemplo da **Figura 9** "*turtle2*", agora podem consultar a sua nova *Pose*. Os itens a serem incluídos na árvore de quadros são identificados pela sua ID, alguns exemplos são: "*base_frame*", parte chassi do robô, "*odom_frame*", quadro que corresponde ao centro de gravidade do robô e, portanto, serve de referência para o cálculo de odometria. Também são incluídos os pontos de referência que servem para todas as partes de um mesmo robô ou de vários robôs que estejam usando a mesma Rede ROS como "*map_frame*" e "*world_frame*", o quadro de

[15] Em uma tradução livre: Transformações

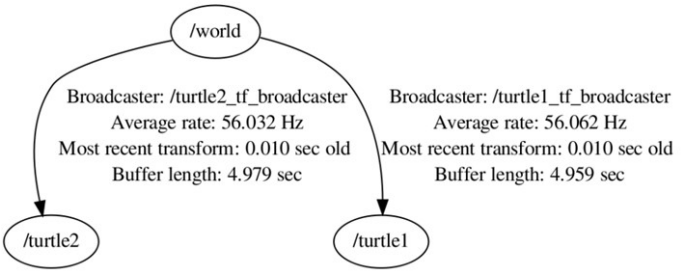
referência na criação de mapas ou na navegação autônoma e a origem do mundo onde os robôs se encontram em uma simulação, respectivamente.

Figura 9 – Simulação de dois robôs móveis (*turtle1* e *turtle2*) em uma única Rede ROS.



Fonte: Silliman (2015)

Figura 10 – Visualização da árvore de quadros (TF) e a relação entre os robôs e o mundo simulado.



Fonte: Foote (2013, p. 3)

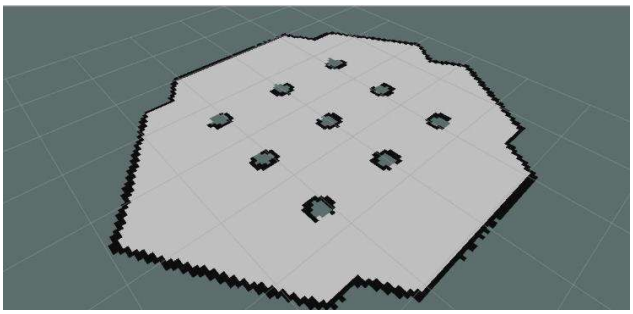
Durante o desenvolvimento de protótipos robóticos é comum que diferentes hardwares entreguem diferentes informações sobre as suas coordenadas dentro do sistema e cada uma dessas informações podem variar na taxa de leitura, latência ou mesmo apresentar falha na comunicação. Para tal problema a biblioteca TF, desenvolvida no código base do ROS, possui a capacidade de entrada de dados assíncrona, tornando-a uma das ferramentas mais robustas do sistema robótico em se tratando de atrasos e perdas de informação (FOOTE, 2013, p. 2).

2.3 SLAM

SLAM é o acrônimo de Localização e Mapeamento Simultâneo, e consiste na problemática em volta da criação do mapa de um ambiente usando sensores robóticos e, em paralelo, na localização do robô dentro do mapa (JONSSON, 2020, p. 11). Funcionando de forma contínua, os algoritmos desenvolvidos para solucionar esse problema fazem uso de dados advindos de sensores a base de reflexão de ondas eletromagnéticas ou mesmo mecânicas, no caso de sensores ultrassons, e combinam com dados de odometria da base robótica implementando iterações onde as leituras anteriores são comparadas com novas leituras para a construção do mapa e para atualização da *Pose* do robô no mesmo. Dentro do âmbito da robótica móvel que utiliza de algoritmos SLAM, a definição de mapas e a sua geração é padronizado, pela subdivisão ISO 8373 focada no vocabulário

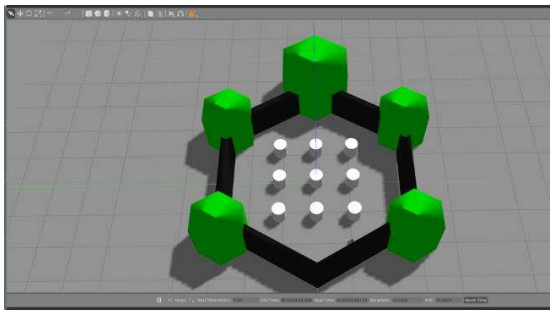
da robótica chamada de OBP (2021), acrônimo em inglês para *Online Browsing Platform*[16], e consiste em: “construir mapas que possam descrever ambientes e suas características geométricas e detectáveis, assim como pontos de referência e obstáculos.”. Para tanto a movimentação da base robótica quase sempre se faz necessária, em outras palavras, o robô precisa explorar o ambiente a ser mapeado. Usando simulações e ferramentas ROS, testes de mapeamento podem ser feitos como no caso da **Figura 11** onde o mapa construído pode-se visualizar, no *software* RVIZ, o mapa navegável em uma escala de cinza e os limites e obstáculos são representados em preto. Ao lado, na **Figura 12**, o ambiente de simulação onde o robô, conectado à Rede ROS, explorou.

Figura 11 – Map gerado por algoritmo SLAM e visualizado na ferramenta ROS, RVIZ.



Fonte: Autoria Própria (2023)

Figura 12 – Simulação do ambiente com obstáculos no *software* Gazebo.



Fonte: Autoria Própria (2023)

A movimentação da base robótica acarreta outros problemas, visto que os dados de odometria não são perfeitos e possuem erros, a diferença entre a *Pose* correta do robô e a estimativa levada em consideração pelo sistema robótico acumula ao passo que o robô explora o ambiente. Para solucionar isso, os algoritmos SLAM fazem uso de representações matemáticas estatísticas em ordem de ponderar as incertezas em relação a *Pose* do robô e a localização dos limites do mapa e dos obstáculos (CAMARA, 2019, p. 18).

2.3.1 Algoritmo *HECTOR MAPPING*

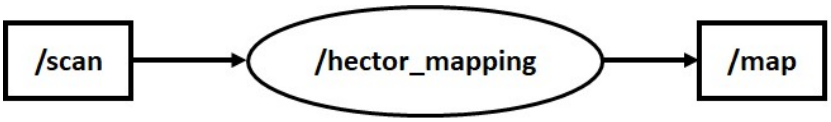
Algoritmos SLAM, em geral, precisam de dados de odometria para seu correto funcionamento e podem ser elaborados com foco em equipamentos caros e especializados, no

[16] Em uma tradução livre: Plataforma de Navegação Online.

entanto "*Hector Mapping*" é considerado um algoritmo estado da arte, e bastante acessível, sendo empregado em sensores de baixo custo, pois além de usar um mapeamento a base de filtro de partículas ele não precisa odometria para funcionar (SAAT et al., 2020, p. 2). Enquanto a maioria das soluções em mapeamento e localização necessitam da combinação de mais de um sensor, na última década, houve um maior desenvolvimento de métodos que utilizam apenas um sensor atuante capaz de altas taxas de medição telêmetro, que dependem fortemente de leituras consecutivas, resultando em um mapa composto de uma grade ocupacional multiresolucional.

O Pacote *Hector Mapping* que pode ser baixado para o sistema ROS no *hardware* onde se deseja instalar o sensor, quando empregado em uma rede ROS, necessita de poucos tópicos para o correto funcionamento, ilustrado na **Figura 13**, vantagem que proporcionou ser implementado com sucesso em diversas plataformas robóticas como VTNTs, VSNTs (Veículos de Superfície Não Tripulado), dispositivos de mapeamento segurados a mão e drones quadrotores não tripulados (ALMEIDA, 2021).

Figura 13 – Tópicos relacionados ao Nó *"/hector_mapping*".



Fonte: Autoria Própria (2023)

O tópico "*/scan*" fornece uma alta taxa de dados, que pode chegar a mais de 4000 amostras em uma única leitura, no formato de Mensagem ROS "*/LaserScan*", detalhes no **Anexo A**, e o tópico "*/map*" é o resultado do trabalho computacional feito pelo Nó *"/hector_mapping*" onde o mapa é organizado a partir dos dados de leitura e do cálculo de *Pose* feito pelo algoritmo *Hector* SLAM. Em paralelo também há os Parâmetros SLAM criados exclusivamente para o funcionamento desse pacote, esses focam em habilitar, ou não, as configurações do Nó como a organização da TF, tópicos necessários a serem subscrevidos e os tópicos publicados pelo Nó "*/hector_mapping*", os parâmetros mais importantes são organizados na **Tabela 1**. A relação de odometria gerada pelas ferramentas probabilísticas do algoritmo é estimada e a relação entre a base do robô (Parâmetro *base_frame*) e o próprio mapa (Parâmetro *map_frame*) são calculadas através da árvore de quadros (TF) e constantemente atualizada durante o mapeamento.

Visto que a principal função do pacote é a geração do mapa em tempo real a partir dos dados provindos do sensor telêmetro, futuras aplicações como a salvar o mapa gerado e visualizar os dados em forma de gráficos devem ser realizados utilizando outros pacotes. Mesmo depois de o mapa ser salvo a aplicação do *hector_mapping* não termina sendo possível continuar a mapear e a atualizar o status de obstáculos e das limitações do mapa.

Tabela 1 – Principais Parâmetros ROS usados pelo Nó "*/hector_mapping*" durante o mapeamento.

Parâmetro ROS	**Tipo**	**Função**
base_frame	String	A identificação (ID) do quadro base, chassi, do robô.
map_frame	String	ID do mapa a ser gerado pelo Nó.
odom_frame	String	ID da odometria, ou seja, da estimação de *Pose* feita pela ferramenta estatística do Nó.
scan_id	String	A identificação (ID) do tópico a se subscrever e receber os dados do sensor.

Fonte: Autoria Própria (2023)

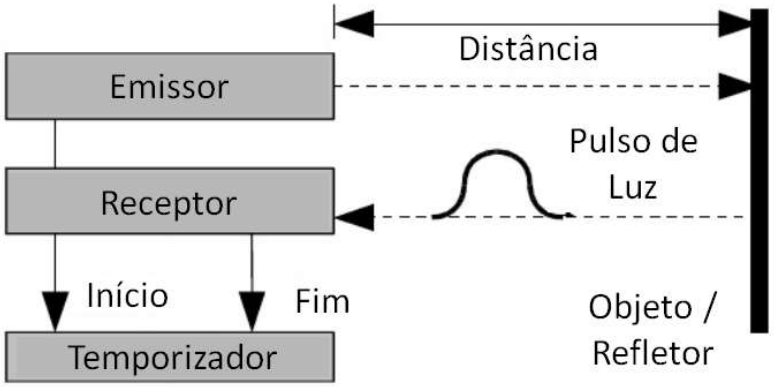
2.3.2 Tecnologia *LiDAR* no SLAM

As soluções de mapeamento e localização na robótica móvel, em sua maioria, necessitam da implementação de ao menos um sensor telêmetro, ou seja, um sensor capaz de fazer a medição da distância entre o sensor e um obstáculo e embora há casos de implementação de sensores ultrassônicos (BUTTERLY; DALY; MORRISH, 2014, p.16), os sensores do tipo *LiDAR* são os mais indicados uma vez que as medições acontecem na velocidade da luz e uma maior facilidade de integrar esses sensores em Redes ROS como no caso de projetos de navegação robótica, monitoramento de estacionamentos e escaneamento 3D de interiores (SLAMTEC, 2022).

O mecanismo de dispositivos *LiDAR* funciona baseado no princípio de *time-of-flight*[17], que utiliza o cálculo do tempo de voo de um feixe de luz contando do momento onde ele é emitido, passando pela sua reflexão no obstáculo e então detectado no momento de seu retorno, na **Figura 14** expõe-se o esquemático do pulso sendo emitido, refletido e detectado assim como o início e o fim da temporização.

[17] Em uma tradução livre: Tempo de voo.

Figura 14 – Princípio de funcionamento de um sensor *LiDAR*.



Fonte: Modificado de Wallhoff et al. (2007, p. 1)

2.4 CONSTRUÇÃO DO SISTEMA ROBÓTICO

2.4.1 Softwares

Os softwares utilizados no planejamento e execução deste trabalho podem ser categorizados de acordo com a área específica do robô em que foram utilizados, sendo elas: projeto mecânico, projeto eletrônico e projeto de *Firmware*. No projeto mecânico tem-se o *Onshape*, no projeto eletrônico, *Fritzing* e no projeto de *Firmware*, o *Raspberry Pi Imager*, que, por consequência engloba o sistema operacional usado no *Firmware* do robô, neste trabalho sendo Linux Ubuntu 20.04 Server.

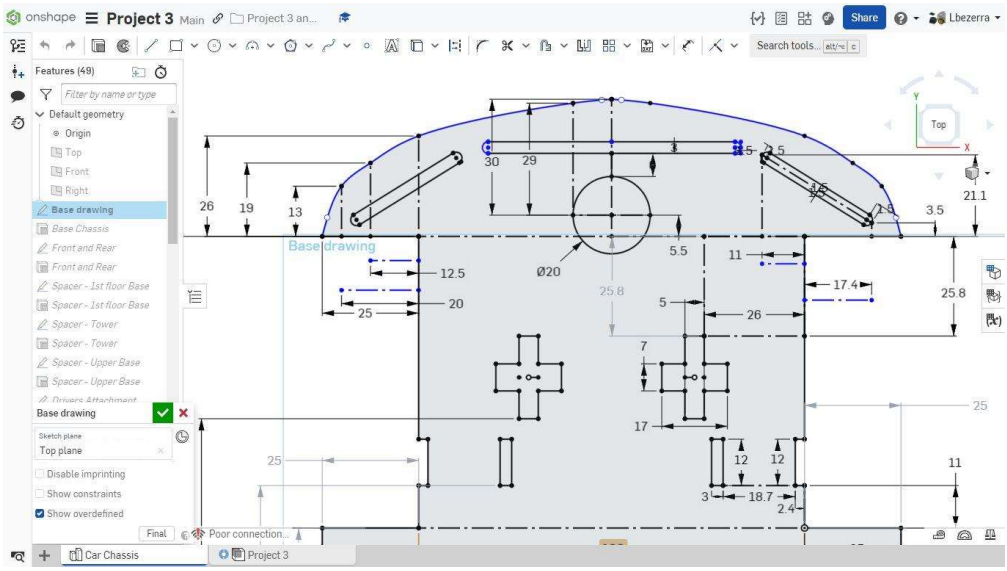
2.4.1.1 *Onshape*

Desenvolvido para atuar como um *software* CAD (*Computer Aided Design*[18]), *Onshape* se destaca da concorrência por ter sido um dos primeiros a usar em toda a sua plataforma a tecnologia de computação em nuvem, ou seja, usuários podem fazer acesso ao *software* e ao conjunto de ferramentas que ele disponibiliza através de qualquer navegador Web, no endereço: https://cad.onshape.com/, ou mesmo pelo aplicativo nas plataformas iOS e Android. Sua rápida

[18] Em uma tradução livre: Desenho Assistido por Computador.

difusão entre projetistas profissionais, hobbyistas e estudantes das áreas de engenharia se dá pela facilidade em se compartilhar arquivos e projetos permitindo que esses sejam modificados e visualizados por membros diferentes de uma mesma equipe em tempo real. Além da familiaridade que se tem com outros exemplos de *software* CAD, o que ajuda na inclusão de experientes projetistas, também é oferecido oficinas de aprendizagem passo a passo desde o nível básico até o avançado, disponível no endereço: https://learn.onshape.com/. Na **Figura 15** é ilustrado o ambiente de trabalho *Onshape* onde foi desenhado o chassi do robô deste projeto.

Figura 15 – Ambiente de trabalho *Onshape* onde se realiza os desenhos 2D.



Fonte: Autoria Própria (2023)

Por ser uma plataforma de aplicação em nuvem, elimina a dificuldade na preparação de computadores para a instalação do *software* CAD, visto que é necessário apenas conexão com a internet. Em contrapartida, no caso de *softwares* concorrentes como SolidWorks, AutoDesk ou Fusion, o computador que se deseja instalar a aplicação precisa de requisitos mínimos de *hardware* como determinados Gigabytes de memória RAM e relativamente avançadas capacidades de processamento. A solução proposta pelo *Onshape* está sendo adotada pelas concorrentes, uma vez que fóruns de compartilhamento de desenhos 2D e 3D – práticas frequentes nas comunidades

maker[19] estimuladas pela popularização das impressoras 3D - se tornam cada vez mais relevantes no âmbito da robótica. A interface online e amigável possibilita o trabalho com diversos tipos de arquivos que facilitam as etapas de planejamento, execução e teste de protótipos robóticos. O que antes era um diferencial da plataforma agora se configura como uma tendência a ser seguida pelas empresas do ramo.

2.4.1.2 *Fritzing*

Fruto da filosofia *open-source*, como outros softwares e sistemas operacionais abordados neste trabalho, o *software Fritzing* tem o objetivo de permitir a criação de *hardware* eletrônico de forma simples e acessível, destacando as bibliotecas de módulos comercialmente difundidos como placas da linha Arduino e da linha *Raspberry Pi*, disponível para download em https://fritzing.org/download/. Possui ferramentas de visualização de diagramas esquemáticos e de layout de PCBs (Placas de Circuito Impresso) no caso onde se faça necessário a confecção de módulos próprios, no entanto, um dos pontos que diferencia esse *software* dos seus concorrentes seja a verossimilhança dos hardwares quando necessários incluir nos esquemáticos e a interface lúdica que permite bastante flexibilidade na hora de se organizar os elementos de um projeto eletrônico. Neste trabalho utilizou-se a versão 0.9.10 para a correta ligação entre o *hardware Raspberry Pi 3B*, computador a bordo do robô, com os *drivers* de movimentação das rodas assim como a conexão dos encoders.

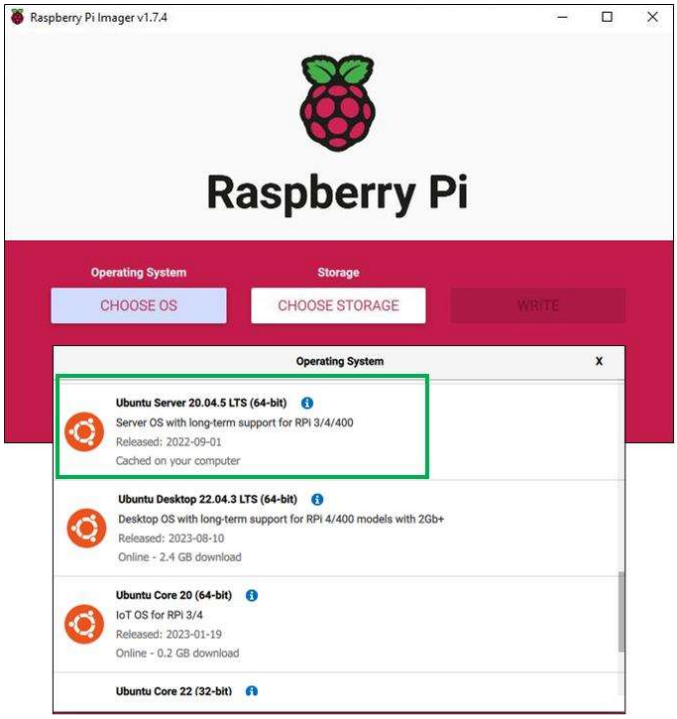
2.4.1.3 *Raspberry Pi Imager*

Software desenvolvido pelos mesmos criadores da linha de computadores *Raspberry Pi* e disponível para download no endereço: https://www.raspberrypi.com/software/, tem a função de facilitar uma das primeiras etapas necessárias no uso desses computadores: a instalação de um sistema operacional. Podendo ser instalado em um computador rodando Windows, macOS ou Ubuntu, essa ferramenta permite ter acesso a biblioteca de imagens de sistemas operacionais desenvolvidas exclusivamente para rodar nos computadores *Raspberry Pi*, ou seja, versões que

[19] Definição: Abordagem educacional focada na aprendizagem através da execução de projetos.

levam em consideração as limitações dessa linha de computadores e que, portanto, necessitam de menos memória RAM e são otimizados para conseguirem ser armazenados em um cartão SD.

Figura 16 – Tela de seleção do sistema operacional no *software Raspberry Pi Imager*.



Fonte: Autoria Própria (2023)

Uma vez selecionado o sistema operacional a ser rodado no computador *Raspberry Pi*, é necessário também escolher em qual *hardware* específico da família *Raspberry* será instalado o sistema e em quais capacidades como "*Desktop*", onde todas as ferramentas possíveis são instaladas incluindo uma interface gráfica, "*Server*", uma versão mais leve e com menos ferramentas, pois entende-se que o *hardware* será acessado apenas remotamente, ou "*IoT*", do inglês *Internet of Things*[20], a mais simples das versões onde o hardware será empregado como uma extensão de uma automação cujo o sistema principal estará alocado em outro hardware pertencente à mesma rede. Neste trabalho utilizou-se a versão 1.7.4 e na **Figura 16** tem-se a tela de seleção do sistema operacional Ubuntu 20.04 Server, a sua versão, data de lançamento da versão, tamanho do

[20] Em uma tradução livre: Internet das Coisas.

arquivo e configuração de capacidades selecionadas para esse projeto assim como quais os computadores da linha *Raspberry* são compatíveis.

2.4.1.4 *Gazebo*

Criado para ser de fácil acesso, *Gazebo* é um *software* de simulação grátis e de filosofia open-source, foi desenvolvido pela mesma empresa criadora do ROS, e possui ferramentas voltadas para a elaboração e teste de sistemas robóticos. Disponível para download no endereço: https://gazebosim.org/docs, possui versões para os principais sistemas operacionais: Linux, Windows e MacOS.

Em meio a outras soluções, como os softwares *MORSE*, *RoboDK* e *COPELLIA* (anteriormente conhecido como *V-REP*), o *Gazebo* recebe preferência em ser usado por muitos desenvolvedores e *hobbyistas* na área da robótica devido a sua compatibilidade com ROS, algo que está presente desde a sua concepção, ou como Quigley, Brian e Smart (2015, p. 96) detalham:

> "ROS integra-se de forma meticulosa através do pacote 'gazebo_ros'. Este pacote disponibiliza um plugin que permite uma comunicação bidirecional entre o Gazebo e o ROS. Sensores simulados e os dados físicos podem ser transmitidos do Gazebo para o ROS e os comandos de atuação podem ser transmitidos do ROS de volta para o Gazebo."

2.4.2 Hardwares

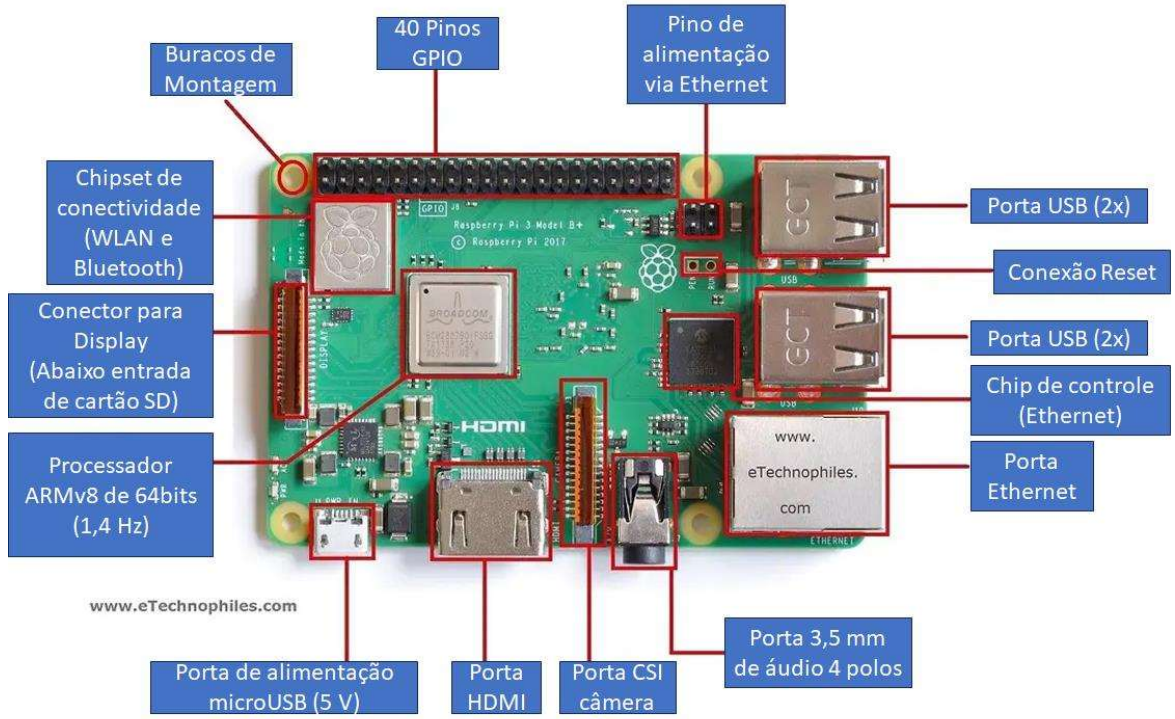
As placas comerciais integrantes desse projeto podem ser divididas entre sistemas embarcados computacionais e os sensores usados no robô. Nos hardwares embarcados tem-se o computador *Raspberry Pi 3B* e o *driver* de ponte H (L298N), a definição dessas placas e a identificação dos seus pinos, portas, *chips* e partes físicas. Nos sensores tem-se os Encoder Óptico FC-03 e o *RPLidar A1*, assim como uma breve explicação do princípio de funcionamento e a interface entre fenômeno físico e sinal elétrico enviado ao sistema robótico.

2.4.2.1 *Raspberry Pi 3B+*

Trata-se de um *hardware* desenvolvido para atender a demanda de projetos embarcados que necessitam de maior capacidade de processamento do que um microcontrolador. A linha de

computadores Raspberry se caracterizam por possuírem uma placa única, do tamanho de um cartão de crédito, com conectividade de rede sem fio (*WLAN*[21]) e tecnologia *Bluetooth*. Em 2018, foi lançada a versão da placa *Raspberry Pi 3B+,* sendo assim, a terceira geração do *hardware Raspberry Pi*, utilizada neste trabalho, é ilustrada na **Figura 17** com o esquemático de suas partes.

Figura 17 – Placa *Raspberry Pi 3B+* e a identificação de suas partes, portas, chips e pinos.



Fonte: Modificado de Negi (2020)

A placa comercial, desenvolvida pela *Raspberry Pi Fundation*, tem o foco de disponibilizar uma plataforma para educação e experimentação na programação de computadores, pois pode ser usado como um computador desktop comum para executar funções como processamento de texto, planilhas, vídeos de alta definição, jogos e programação (POLOLU, 2023). Para isso, é necessário realizar a instalação de um sistema operacional que suporte todas essas funções, conforme exemplificado no tópico **2.4.1.3 *Raspberry Pi Imager***. Com o auxílio de um teclado, e no caso de versões do sistema que oferecem uma interface gráfica, um mouse, é possível até mesmo navegar

[21] Em uma tradução livre: Wireless LAN (Rede de computadores sem fio).

na internet. A placa é equipada com um processador ARMv8 de 64 bits com capacidade de funcionar em até 1,4 GHz quando necessário rodar vídeos de alta definição e fazer interface com telas através da sua porta HDMI. Além disso, a placa *Raspberry Pi 3B+* também possui a presença de pinos de entrada e saída que funcionam em uma faixa de tensão variando entre 3,3 V e 5V que se traduz em lógica binária, permitindo uma interface com dispositivos eletrônicos e robóticos no mais baixo nível de programação como o acionamento de atuadores e a leitura de sensores.

2.4.2.2 "*TT Motor*" e o Kit Chassi 4WD

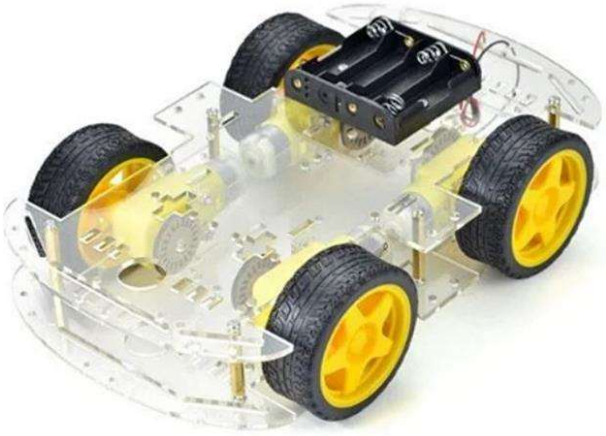
Os motores "*TT Motor*" são motores com caixa de redução acoplada e, nominalmente, trabalham com tensões na faixa de 3 a 6 VDC onde sua velocidade de rotação também varia de 120 até 250 RPM, respectivamente (ADAFRUIT, 2023). Nessas velocidades, contadas antes de passarem pela caixa redutora de razão 1:48, as correntes drenadas pelo motor podem chegar até 160 mA, no entanto, para nível de projeto eletrônico, deve-se considerar os casos onde a roda fica obstruída, ou seja, não há movimento do motor resultando em uma corrente de pico de 1,5 A, o que fatalmente danificaria qualquer *hardware* de controle como microcontroladores, microprocessadores ou placas comerciais com *chipsets* sensíveis, fazendo-se assim necessário um *driver* junto dos motores.

Na **Figura 18** é ilustrado o "*TT Motor*" usado neste trabalho aberto para a visualização da caixa redutora. Esse motor foi utilizado no projeto pois faz parte do kit comercial vendido em lojas locais de eletrônica, com preço acessível e voltado para o público estudantil chamado de kit 4WD, ilustrado na **Figura 19**. Esse Kit apresenta rigidez no chassi durante o movimento e torque suficiente para movimentar toda a plataforma robótica com baterias, sensores e computadores embarcados como no trabalho realizado por Singh et al. (2023, p. 196). Ele é composto de dois chassis de acrílico de espessura 3mm com diversos furos para a fixação de apêndices robóticos ou hardwares e para a fixação do segundo chassi existem seis espaçadores de 30 mm que fornecem estrutura para a base robótica, além dos quatro motores, cada motor acompanhado de uma roda e um disco com 20 furos para serem usados junto de encoders ópticos. Embora o kit também acompanhe um porta pilhas AA, nesse projeto optou-se por não usá-lo devido a necessidade de diferentes níveis de potência para a correta alimentação dos diferentes tipos de *hardware*.

Figura 18 – Motor comercial "*TT Motor*" com carenagem aberta.

Fonte: Adafruit (2023)

Figura 19 – Kit comercial 4WD com chassi de acrílico.



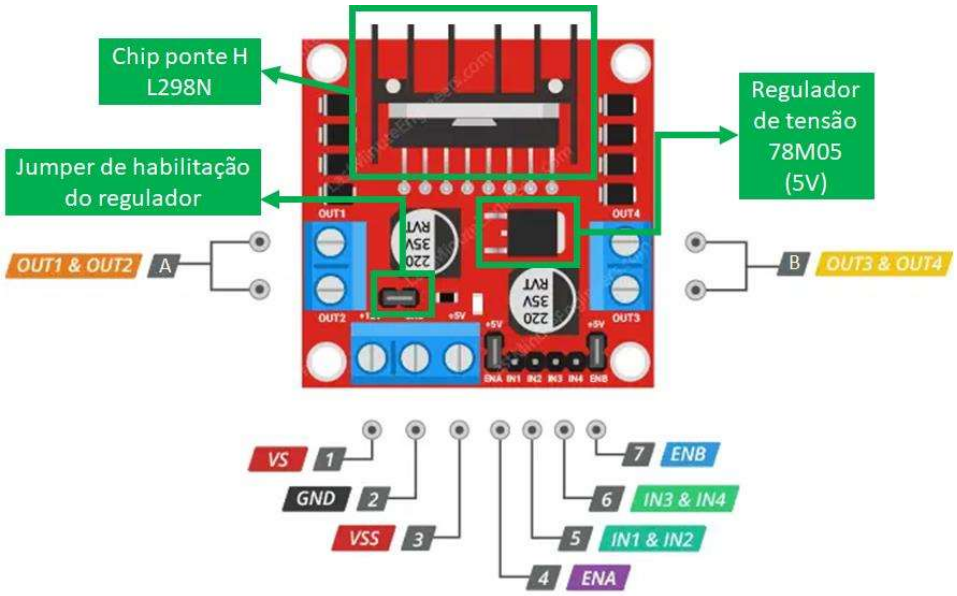
Fonte: Smart Projects Brasil (2023)

2.4.2.3 *Driver* L298N

A robótica aborda métodos de controle de sensores e atuadores, desde o alto nível de fluxo de dados como em algoritmos quanto em baixo nível de comandos binários como o controle de motores através de pontes H. Essa ferramenta de controle, amplamente usado em motores DC e motores de passo, apresenta uma eletrônica mais robusta e capaz de lhe dar com cargas indutivas que demandam maior corrente (CARDOSO, 2017) e, portanto, servem de interfaces de potência

entre o sistema de controle, em geral uma placa Arduino ou um computador *Raspberry*, e os atuadores de um robô. A placa *driver* equipada com o chip comercial ponte H L298N é a solução adotada para a necessidade de interface de potência entre o *hardware* embarcado neste trabalho e os motores com caixa de redução acoplada, chamados de "*TT Motor*", usados na movimentação das rodas do robô, devido a capacidade desse *driver* de suportar correntes até 2 A por canal, no caso de o motor estar obstruído e entrar em regime de corrente máxima. Ilustrado na **Figura 20** está a placa contendo o chip L298N, chip regulador de tensão de 5V 78M05, pinos de entrada e de saída e a identificação dos seus canais, canal A e canal B.

Figura 20 – Indicação dos pinos, chips, circuitos e canais da placa comercial *Driver* L298N.



Fonte: Modificado de Last Minute Engineers (2023)

Em referência a **Figura 20**, a explicação dos itens enumerados e a correlação entre os pinos, chips, circuitos e canais pode ser conferida a seguir:

1. Pino VS é responsável pela alimentação do circuito ponte H, ou seja, da tensão que será fornecida aos canais A e B e foi projetado para funcionar com uma alimentação de 5 a 12 V ainda que seja possível aplicar tensões até 35 V sendo restrito apenas a capacidade de dissipação do chip L298N de 25 W.
2. Pino GND, o fio terra da placa, e deve ser aterrado junto dos outros hardwares embarcados no robô.

3. O pino VSS pode possuir duas funções dependendo o jumper de habilitação do regulador, elas são:
    - Jumper conectado: A tensão fornecida pelo pino VS é regulada pelo chip 78M05 que garante estabilidade na tensão 5 V requerida para o correto funcionamento do circuito lógico da placa. Nesse modo o pino VSS se torna um pino de saída (5 V 0,5 A) que serve para alimentar outras placas como microcontroladores ou sensores. Nesse modo assume-se que a tensão advinda do pino VS não ultrapasse os 7 V, tensão na qual o regulador consegue trabalhar corretamente.
    - Jumper não conectado: A entrada VS é isolada do restante da placa e o regulador 78M05 é desligado. A alimentação do circuito lógico deve ser feita pelo pino VSS, ou seja, 5 V devem ser fornecidos de forma constante e estável, em geral através de outro *hardware* como um microcontrolador, microcomputador ou mesmo uma fonte retificadora comercial.
4. Responsável por habilitar ou não o canal A, o pino ENA recebe comandos que podem funcionar na forma *On* e *Off*, como no exemplo de se ligar ou desligar o motor conectado ao canal A, mas que podem também funcionar de forma proporcional variando a velocidade de rotação do motor, ou seja, de 0 V até o valor máximo de 5 V.
5. Os pinos IN1 e IN2 são responsáveis por determinar o sentido de giro do motor conectado no canal A baseado na diferença de tensão entre cada pino do canal, Out1 e Out 2. A configuração do sentido do giro do motor a partir do sinal digital lógico de entrada pode ser visto na **Tabela 2**.
6. A mesma lógica é aplicada nos pinos do item 5 é aplicada nos pinos IN3 e IN4, porém em relação ao canal B, Out3 e Out4.
7. Pino ENB controla o canal B da mesma forma que o item 4 controla o canal A. É importante ressaltar que o método de controle PWM pode ser usado a fim de se obter a variação proporcional compreendida entre mínimo, 0 V, e máximo, 5V.

Visto o funcionamento de suas partes, neste trabalho o *driver* L298N faz parte do circuito de controle e movimentação dos motores "*TT Motor*" e embora também possa ser empregado em

servomotores e motores-de-passo, quando implementado para o controle do sentido de rotação de motores DC, seu funcionamento lógico pode ser mapeado de acordo com a **Tabela 2.**

O comportamento da roda acoplada ao motor ligado no canal A é ditado pelas entradas de sinais lógicos nos pinos IN1 e IN2. A definição de sinal lógico adotada aqui pode ser traduzida na tensão voltaica aplicada nos pinos: "BAIXO" sendo 0 V e ALTO sendo o valor determinado para a rotação do motor, neste trabalho sendo 12 V. O comportamento mapeado para o canal A pode ser empregado no canal B, sendo necessário apenas substituir os pinos de entrada IN1 e IN2 pelos pinos IN3 e IN4, respectivamente.

Tabela 2 – Mapeamento do comportamento do motor nos canais do *Driver*.

IN1 (Nível Lógico)	**IN2 (Nível Lógico)**	**Movimentação do motor (canal A)**
BAIXO	BAIXO	Desligado
BAIXO	ALTO	Horário
ALTO	BAIXO	Anti-Horário
ALTO	ALTO	Freio

Fonte: Autoria própria (2023)

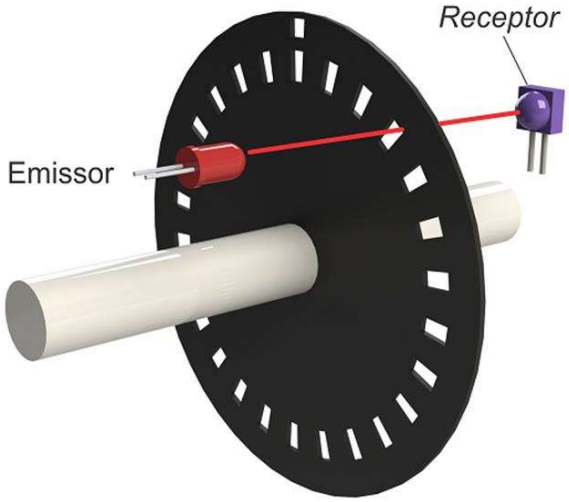
2.4.2.4 Encoder Óptico FC-03

Encoder são dispositivos/sensores eletro-mecânicos, cuja funcionalidade é transformar posição em sinal elétrico digital. Com a utilização de encoders, é possível quantizar distâncias, controlar velocidades, medir ângulos, número de rotações, realizar posicionamentos, rotacionar braços robóticos e etc. (ALMEIDA, 2017). Encoders podem ser classificados em tipo, arquitetura de construção (princípio de funcionamento) e resolução. Um encoder pode ser do tipo incremental ou absoluto, pode possuir ter um princípio de funcionamento óptico, mecânico ou magnético e possuir uma resolução geralmente compreendida entre 16 a 512 pulsos por revolução (COTA, 2019, p. 18).

Quando empregado em robôs móveis com rodas, o tipo de encoder mais usado em projetos robóticos de baixo orçamento é o incremental óptico, **Figure 21**, e seu funcionamento baseia-se em um disco acoplado com furos espaçados em distâncias iguais ao eixo no qual deseja-se medir a rotação junto de um emissor de luz e de um receptor, caracterizando o aspecto óptico, e o passo que o eixo rotaciona, a luz apenas alcança o receptor quando passa pelas regiões do disco com furos gerando assim pulsos que quando somados representam a rotação realizada pelo eixo,

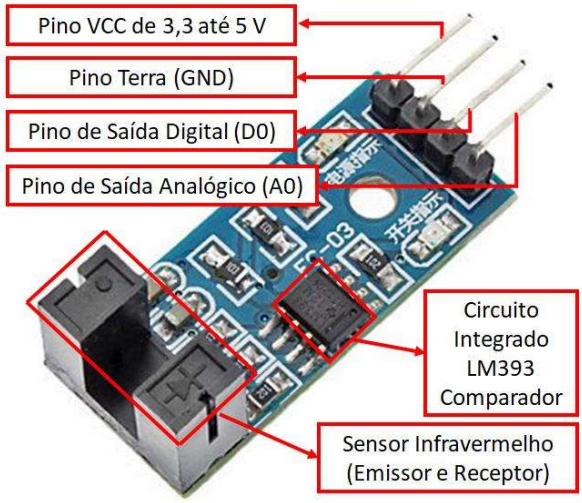
comportamento que o configura como sensor incremental, em outras palavras, um sensor integrador (NIKU, 2010, p. 325). Na **Figura 22**, a placa comercial Encoder Óptico FC-03 baseado na emissão de luz infravermelha é apresentada com a identificação de seus pinos e de suas partes e nesse projeto são usadas quatro placas dessas nos quatro conjuntos roda-motor da plataforma robótica Kit chassi 4WD.

Figura 21 – Princípio de funcionamento do Encoder Óptico com o disco acoplado.



Fonte: Almeida (2017)

Figura 22 – Placa comercial FC-03 e a identificação das suas partes.



Fonte: Autoria Própria (2023)

2.4.2.5 Sensor *RPLidar A1*

Para o desenvolvimento de protótipos robóticos em laboratório, os sensores *LiDAR* da linha *RPLidar*, desenvolvidos pela empresa *SLAMTEC*, são ideais devido a capacidade de realizar uma leitura em 360º, trabalhar em uma faixa de 2 até 10 Hz de rotação e um alcance de leitura nominal em até 12 metros, combinado com uma razoável acessibilidade e grande difusão no mercado de robôs e de hobbyistas na área de *hardware* (JIN, 2021, p. 19).

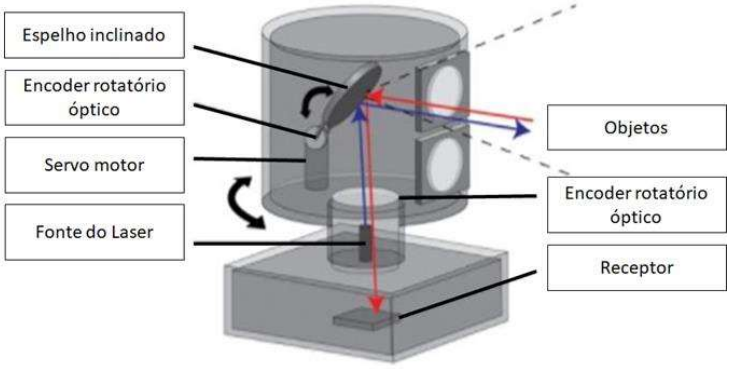
O *RPLidar A1*, **Figura 23**, é o *hardware* de entrada da sua linha de sensores, possui uma rápida e descomplicada interface via porta USB facilitando a implementação desse sensor em sistemas robóticos rodando ROS. Na **Figura 24** possível visualizar o mecanismo que algumas soluções comerciais apresentam: o sensor em cima de um servomotor que ao girar dispara vários

feixes e combinado a um encoder para o correto rastreamento da angulação do feixe de luz e, portanto, em uma leitura se é capaz de ter a geometria compreendida no ângulo de varredura, no caso da linha de sensores *RPLidar*, há uma adaptação da eletrônica para ser possível uma revolução completa, resultando em um perfil completo da geometria que cerca o sensor quando executado uma única leitura.

Figura 23 – Sensor *RPLidar A1*.

Fonte: SLAMTEC (2022).

Figura 24 – Esquemático de sensor *LiDAR* que possui capacidade de varredura em 2D.



Fonte: Modificado de Saat et al. (2020, p. 2).

A *SLAMTEC* projetou esse sensor para trabalhar em ambientes fechados, abrigado da luz solar, ele é relativamente pequeno e leve, ocupando uma área de 55 mm por 96,8 mm e pesando 170 gramas, possui baixo consumo energético de 0,5 W, sendo necessário uma fonte de alimentação com 5 V e entrega leituras com uma resolução de 1°, além de ser seguro para o olho humano, características que o tornam de fácil implementação em plataformas móveis e sua altura de 70,3 mm permite experimentação quanto ao lugar de instalação.

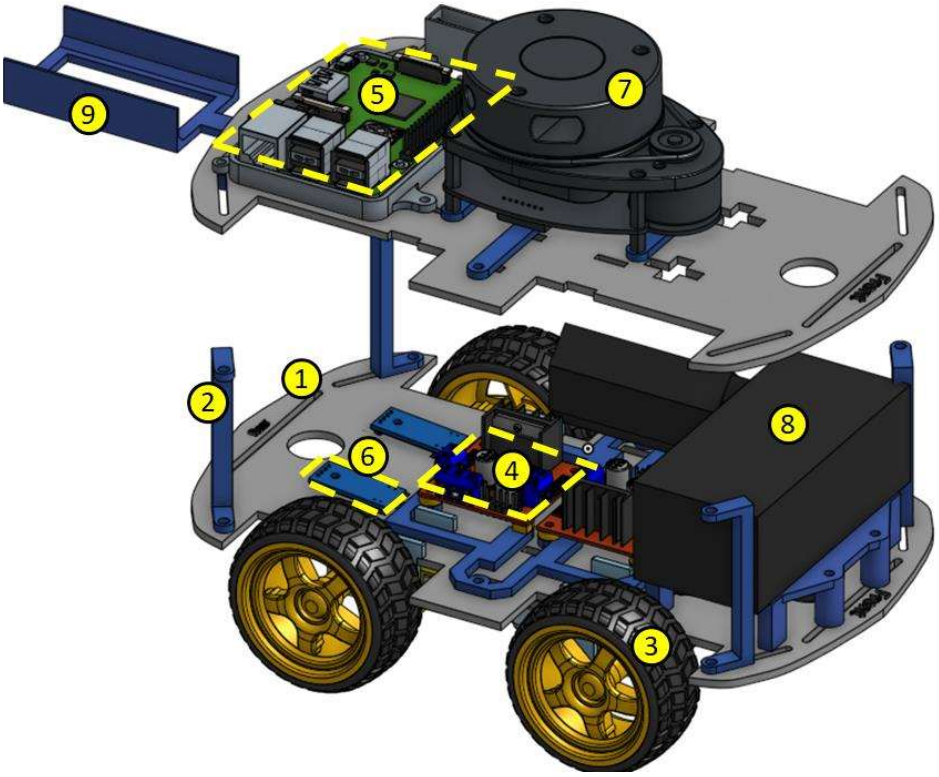
3 METODOLOGIA

Nesse trabalho a metodologia demonstra como os conceitos, materiais e tecnologias apresentados no capítulo anterior se combinam para a construção da plataforma robótica móvel, assim como suas peças mecânicas, seus sensores e atuadores, seus circuitos eletrônicos e seu sistema embarcado usado na integração de todos esses aspectos do projeto.

3.1 VISÃO GERAL DO PROJETO

O modelo 3D da plataforma robótica desenvolvida neste trabalho, foi desenhado e renderizado na ferramenta CAD *Onshape* e possui o *Mock-up*[22] das placas comerciais, sensores e fontes de alimentação de potência usadas no robô e pode ser visto na **Figura 25**.

Figura 25 – Robô móvel desenvolvido, com chassis espaçados e itens identificados e inumerados.



Fonte: Autoria Própria (2023)

[22] Em uma tradução livre: Maquete Digital.

O robô móvel composto de dois andares, dois chassis idênticos conectados por espaçadores, expandidas para melhor visualização e algumas peças foram ocultadas da imagem para melhor identificar as partes principais do projeto e seus periféricos eletrônicos, ao passo que essas são identificadas e inumeradas para serem comparadas com a **Tabela 3**.

Tabela 3 – Denominação e comentários sobre as partes da plataforma robótica enumerados.

Número	**Denominação**	**Comentários**
1	Chassi do robô	Parte original do Kit Chassi 4WD.
2	Espaçador	Projetado para abrigar o *hardware* e fonte de potência.
3	Conjunto Roda-Motor	Parte original do Kit Chassi 4WD.
4	*Driver* L298N	Responsável por fazer a interface de potência entre *Raspberry Pi 3B+* e os motores DC.
5	*Raspberry Pi 3B+*	Computador embarcado responsável por rodar o ROS.
6	Encoder Óptico Incremental	Sensor acoplado às rodas para determinação da Odometria.
7	Sensor *RPLIDAR A1*	Sensor *LiDAR* para mapeamento.
8	Transformador	Fonte de Potência para os hardwares embarcados.
9	"Cordão Umbilical"	Projetado para suportar o *Plug* de alimentação AC.

Fonte: Autoria Própria (2023)

A metodologia na elaboração e construção do projeto foi dividido em três partes, na seguinte ordem: Projeto Eletrônico, Projeto Mecânico e Projeto de *Firmware*[23]. Os próximos tópicos deste trabalho apresentam o desenvolvimento de cada uma dessas partes.

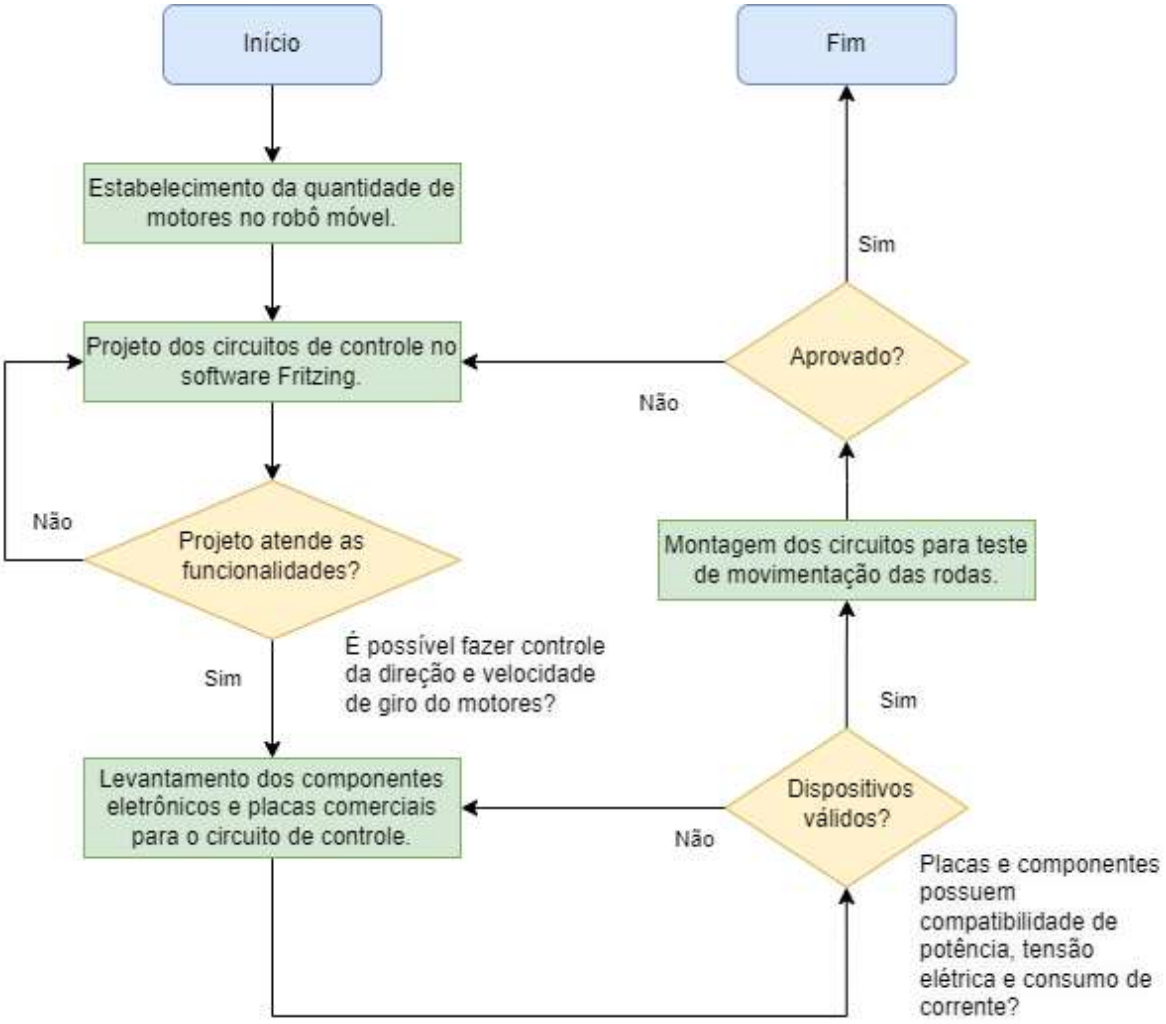
3.2 PROJETO ELETRÔNICO

A primeira etapa de desenvolvimento deste trabalho, uma vez determinada a função do robô e de seu sistema robótico, é a elaboração do projeto eletrônico, quais componentes serão embarcados no robô, onde as placas comerciais serão empregadas e quais serão as conexões entre hardwares, sensores e atuadores. Este aspecto do projeto compreende a elaboração dos circuitos

[23] Em uma tradução livre: *Software* desenvolvido para o sistema robótico.

responsáveis pela movimentação dos motores, sendo possível fazer o acionamento das quatro rodas do robô de forma independente, proporcionando um controle de sentido de rotação e de velocidade. Na **Figura 26** é possível visualizar o fluxograma de ações tomadas na elaboração e execução do circuito de controle.

Figura 26 – Fluxograma de ações tomadas na elaboração e execução do projeto eletrônico.



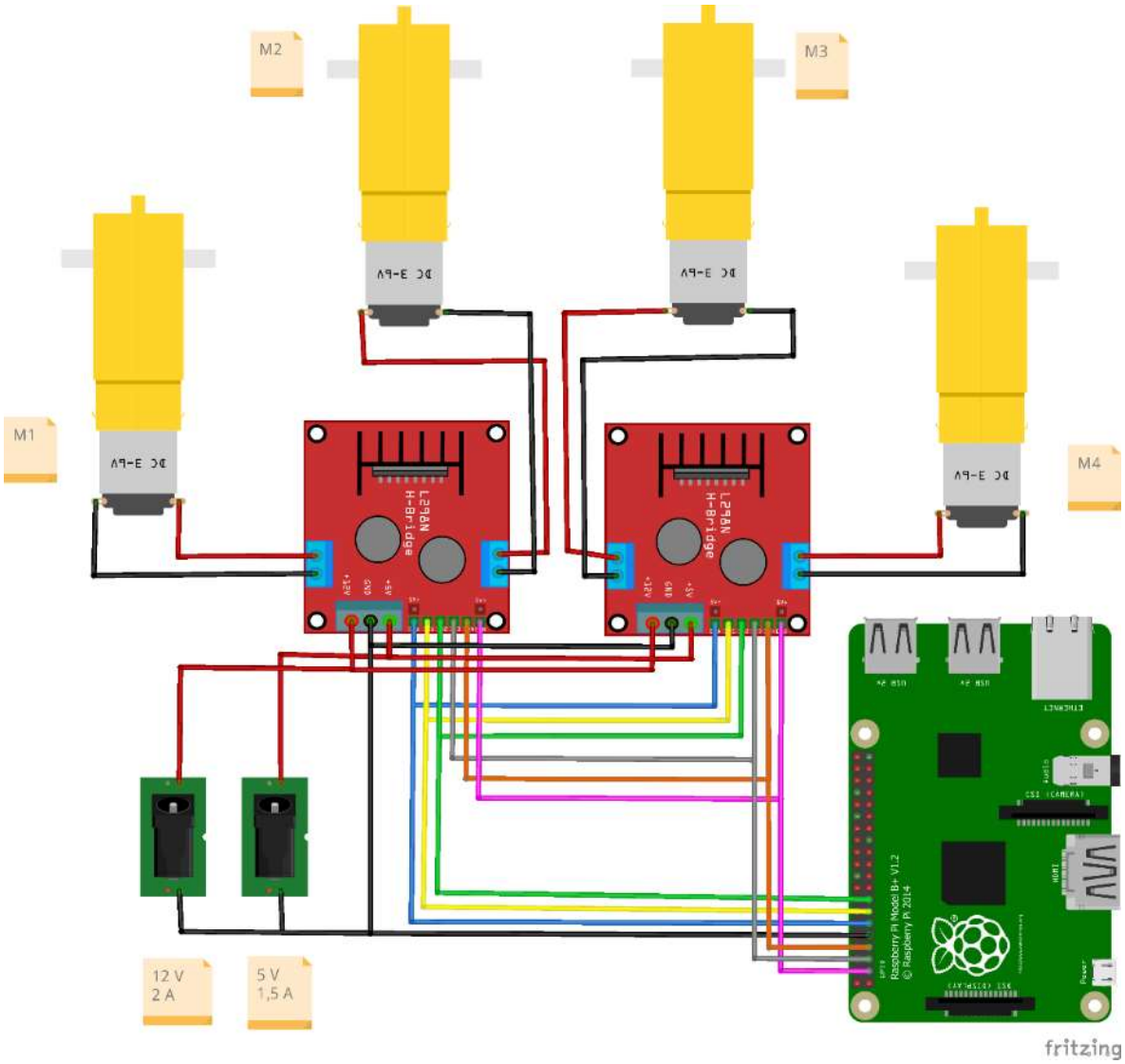
Fonte: Autoria Própria (2023)

3.2.1 Circuitos de Controle

Na **Figura 27** está ilustrado o circuito eletrônico elaborado no *software Fritzing* para o controle das rodas do robô, assim como as linhas de alimentação necessárias para o correto

funcionamento do circuito, no canto esquerdo inferior representadas pelas conexões fêmeas do tipo P4. Devido a capacidade da uma placa *Driver* L298N de controlar dois motores DC ao mesmo tempo com suas duas pontes-H, para o controle dos quatro motores, é necessário usar duas placas cujo os comandos de entrada são feitos de forma paralela. Dessa forma, os motores M1 e M3 são acionados ao mesmo tempo, por estarem ligados ao canal A, embora estes estejam ligados à diferentes *Driver*s. O mesmo acontece com os motores M2 e M4, pois ambos estão nos canais B de seus respectivos *Driver*s.

Figura 27 – Circuito de controle dos quatro motores DC do robô móvel.



Fonte: Autoria Própria (2023)

As conexões entre os pinos GPIO (com as localizações no barramento) da placa *Raspberry Pi 3B+* e os pinos de comando das placas *Driver*s L298N são divididos por canal e podem ser conferidos na **Tabela 4.**

Tabela 4 – Conexão entre Placa *Raspberry Pi 3B+* e os *Driver*s L298N.

Canal	**Pino dos *Driver*s**	**Pinos GPIO a Placa *Raspberry Pi* 3B+ (Pino Numérico[24])**	**Dispositivos Acionados**
A	ENA	GPIO 17 (11)	M1 e M3
	IN1	GPIO 27 (13)	
	IN2	GPIO 22 (15)	
B	ENB	GPIO 2 (3)	M2 e M4
	IN3	GPIO 3 (5)	
	IN4	GPIO 4 (7)	
	VSS	GPIO 5V (2)	Circuito Lógico Interno do *Driver*
	GND	GPIO GROUND (6)	

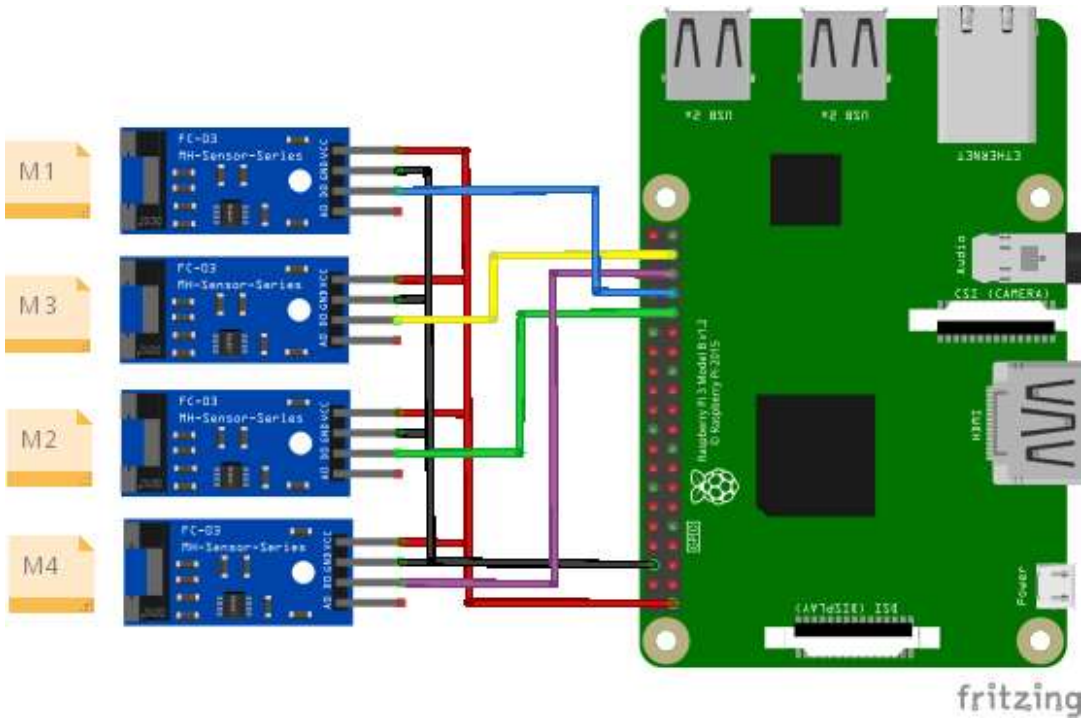
Fonte: Autoria Própria (2023)

Na **Figura 28** está o esquemático de ligação das quatro placas comerciais Encoder Óptico Incremental FC-03 usadas neste trabalho, abrangendo sua alimentação e a transmissão dos pulsos para o computador embarcado, *Raspberry Pi 3B+*. Cada uma das quatro placas é instalada junto do disco furado acoplado em cada deixo do conjunto motor-roda e permite a geração de odometria de forma individual de cada mecanismo locomotor do robô móvel. As conexões entre o computador embarcado e as placas Encoder são divididas em alimentação e coleta de dados e são listadas na **Tabela 5**.

A partir desses dois esquemáticos, é possível determinar que duas fontes de potência externas ao robô se fazem necessárias: Uma alimentação para o acionamento dos motores, de 12 V, porém limitada pelo *Driver* L298N em até 2 A, totalizando 24 W e uma alimentação para o computador embarcado *Raspberry Pi 3B+*, com a necessidade de uma fonte especificada pelo fabricante de 5V e 3 A, totalizando 15 W.

[24] Numeração no Anexo B: Relação entre pinos numéricos e pinos GPIO.

Figura 28 – Circuito de controle dos quatro sensores encoder óptico incremental



Fonte: Autoria Própria (2023)

Tabela 5 – Conexão entre Placa *Raspberry Pi 3B+* e as placas FC-03.

Pinos do Encoder	**Encoder relativo ao motor**	**Pino GPIO do Computador Embarcado (Pino Numérico[25])**
D0	M1	GPIO 13 (33)
	M2	GPIO 26 (37)
	M3	GPIO 6 (31)
	M4	GPIO 19 (35)
VCC	Todos	GPIO 3V3 (1)
GND	Todos	GPIO GROUND (9)

Fonte: Autoria Própria (2023)

Devido ao baixo consumo de potência de uma placa FC-03, sendo ele de até 0,06 W, mesmo com a somatória de quatro placas no total, a placa *Raspberry Pi 3B+* é capaz de suprir a demanda necessária para o correto funcionamento nesse projeto através do pino 1 do seu

[25] Numeração no Anexo B: Relação entre pinos numéricos e pinos GPIO.

barramento que fornece 3,3 VDC. O mesmo acontece para o relativo baixo consumo de potência, embora elevado se comparado com as placas Encoders, do circuito lógico interno das duas placas *Driver* L298N de consumo de potência unitário de até 0,18 W, no entanto a alimentação dos 5V necessários para esse circuito são fornecidos pelo pino 2 do barramento do computador embarcado.

3.2.2 Fontes de Potência

Para suprir a necessidade de duas fontes diferentes de potência, ainda na terceira etapa do fluxograma - Levantamento dos componentes eletrônicos e placas comerciais para o circuito de controle - dois transformadores foram selecionado de acordo com as especificações do projeto, eles são: a fonte de alimentação de 12 V e 1,5 A voltada para o acionamento dos motores, ilustrada na **Figura 29**, e a fonte de 5V e 3A projetada para fornecer potência para o computador a bordo, ilustrada na **Figura 30**, ambos com a necessidade de serem alimentados com tensão monofásica alternada de 127 V. Essa escolha de projeto acarreta desafios a serem solucionados no aspecto mecânico como a alocação de espaço na plataforma robótica para o *plug* de alimentação 127 VAC, chamado de "cordão umbilical", para as fontes transformadoras em si e os componentes adicionais que vem junto delas, no caso da fonte para os motores é a presença de uma conexão fêmea do tipo P4 e no caso da fonte da placa *Raspberry Pi 3B+* é a presença de um botão liga/desliga, não sendo necessário conector específico, visto que a conexão dessa fonte é do tipo USB-B e a placa já possui uma entrada desse tipo.

Um dos motivos para a escolha de uma fonte de 12 V para o acionamento dos motores está relacionado com uma característica do funcionamento da placa *Driver* L298N, onde essa ao permitir a passagem de corrente para um de seus canais, canal A ou canal B, apresenta uma queda de tensão de 2 V, ou seja, nesse caso a tensão entregue aos motores, na sua configuração máxima, será de 10 V. Levando em consideração uma fonte de tensão mais comum como a de 5 V, quando acionado, mesmo na sua configuração máxima, apenas 3 V chegariam ao motor e está tensão se mostrou muito baixa para a movimentação da plataforma robótica, uma vez que nele é embarcado as fontes transformadoras e o sensor *LiDAR* cujo os pesos somados representam significativa carga mecânica e tornam difícil a movimentação do robô quando baixas tensões elétricas são aplicadas.

Figura 29 – Fonte destinada ao acionamento dos motores com conexão P4, 12 V e 1,5 A.

Fonte: Smart Projects (2023)

Figura 30 – Fonte projetada para a placa comercial *Raspberry Pi 3B+*, com botão liga/desliga.

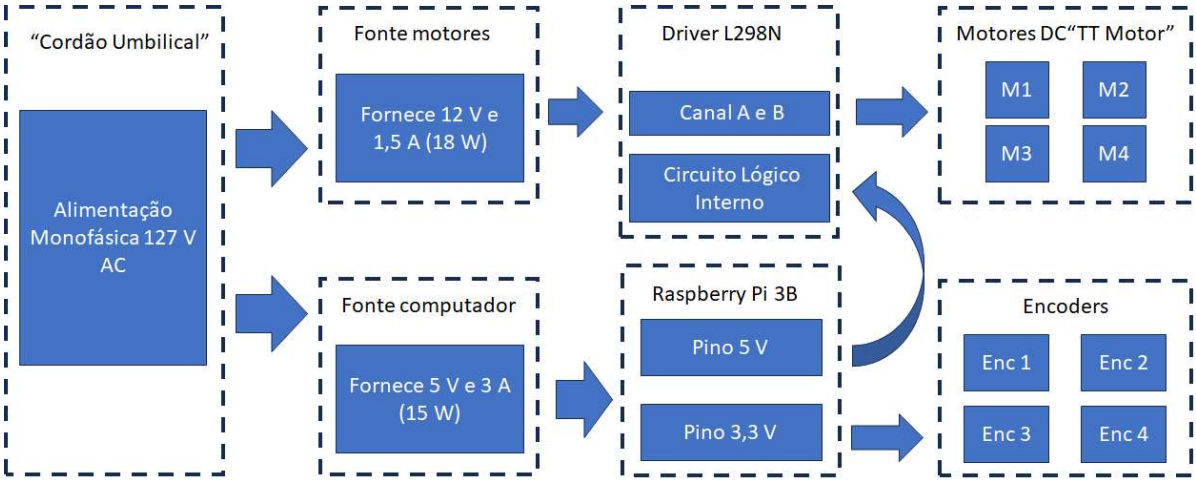
Fonte: Smart Projects (2023)

3.2.3 Montagem do Projeto Eletrônico

Antes da montagem dos circuitos no robô, cabe a análise dos níveis de potência presentes no projeto eletrônico e são resumidos na **Figura 31**, sendo organizados da esquerda para a direita. Começando desde a fonte externa ao robô, monofásica de 127 V e corrente alternada, passando pelas duas fontes transformadoras embarcadas no robô e chegando ao último nível de potência, configurado por baixos níveis de tensão elétrica para a alimentação dos sensores Encoder e do circuito interno lógico do *Driver* L298N.

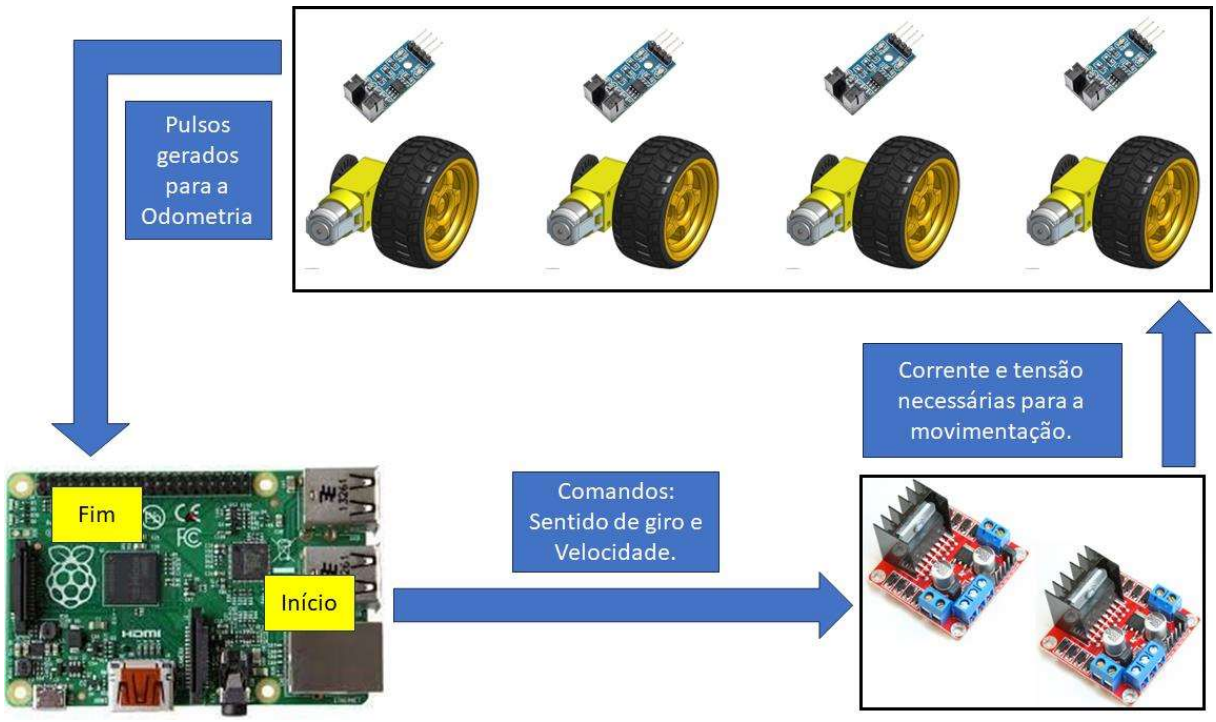
A última etapa da execução do projeto eletrônico – Montagem do circuito para teste de movimentação – pode ser organizada pela lógica de comando-execução-sensoriamento-computação. Essa lógica começa e termina no computador embarcado, como visto na **Figura 32**, onde os comandos são enviados, eletronicamente por nível lógico digital, dos pinos do *Raspberry Pi 3B+,* passam pelas placas *Driver* L298N para então acontecer a movimentação dos motores "TT Motor", esses por sua vez movem o eixo da roda que nela estão acoplados os discos furados que geram os pulsos nas placas Encoder FC-03, esses pulsos são enviados para os pinos do *Raspberry Pi 3B+* para então serem computados e a odometria por *Dead-Reckoning* ser gerada.

Figura 31 – Níveis de potência elétrica presentes no projeto eletrônico.



Fonte: Autoria Própria (2023)

Figura 32 – Lógica de funcionamento do projeto eletrônico.



Fonte: Autoria Própria (2023)

Por fim a montagem teste foi realizada onde os comandos e o retorno sensorial de pulsos pode ser feito e a compatibilidade de potência se provou adequada permitindo o correto

funcionamento da plataforma robótica sem haver a sobrecarga em nenhum componente ou placa. Os resultados do teste são explanados no **Tópico 4 RESULTADOS** e a lista de componentes, partes elétricas e placas comerciais usadas neste trabalho podem ser conferidas na **Tabela 6**.

Tabela 6 – Lista de partes eletrônicas embarcadas na plataforma robótica.

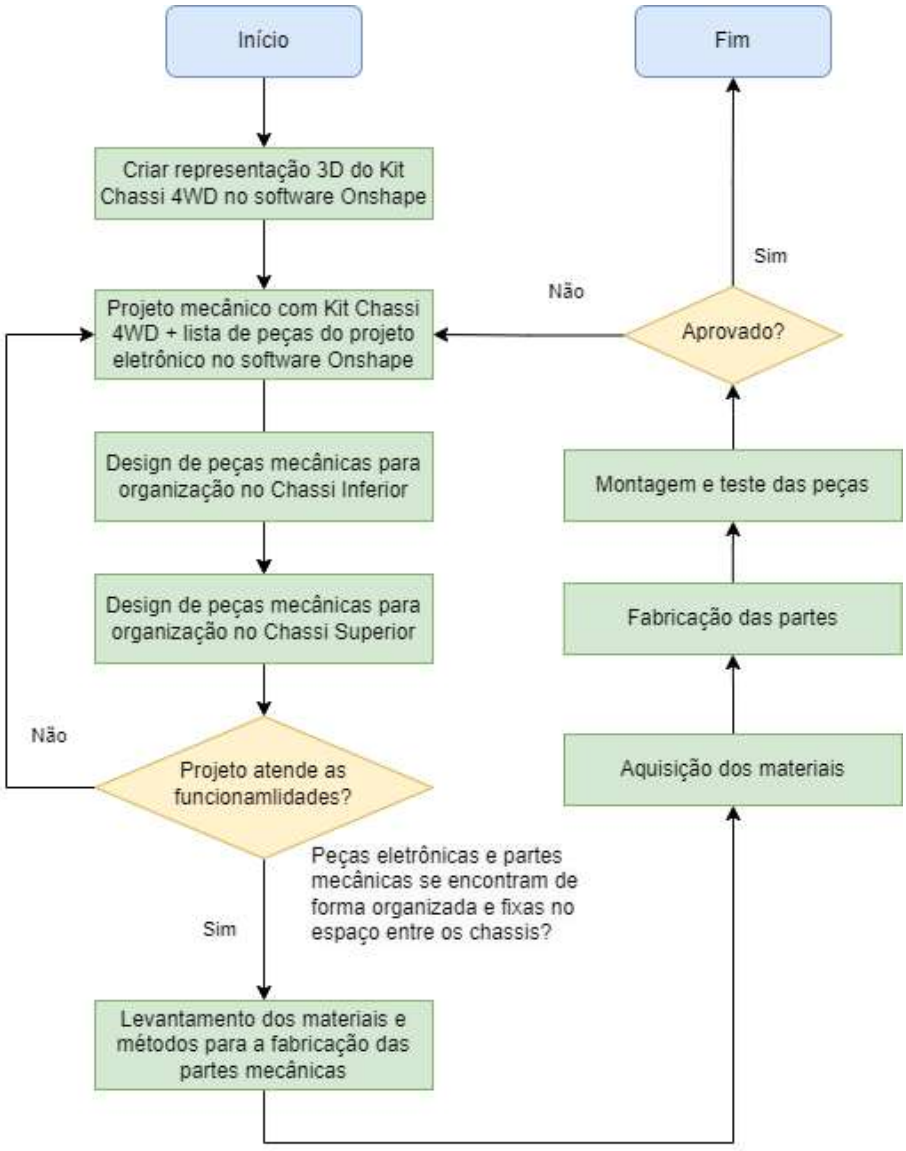
Nome	**Qtd**	**Função**
Raspberry Pi 3B+	1	Computador embarcado responsável por enviar os comandos de sentido de giro e velocidade aos *driver*s. Responsável por registrar os pulsos das placas Encoder.
Driver L298N	2	Placa comercial com dois circuito ponte-H que servem de interface de potência entre o computador embarcado e a fonte de tensão de 12 V necessária para o correto acionamento dos motores.
"*TT Motors*"	4	Motores DC responsáveis pela movimentação das quatro rodas do robô móvel.
Encoder FC-03	4	Encoder Óptico Incremental que funciona combinado ao disco furado acoplado a um eixo de motor, responsável por enviar os pulsos referentes a movimentação de uma roda.
Fonte de potência para os Motores DC	1	Fonte de alimentação de 12 V e 1,5 A responsável por fornecer a potência necessária para o correto funcionamento dos quatro motores do robô móvel. Possui uma conexão P4.
Fonte do computador embarcado	1	Fonte de alimentação especificado pelo fabricante da placa *Raspberry Pi 3B+* que entrega 5 V e 3 A. Possui conexão USB-B e um botão liga/desliga.

Fonte: Autoria Própria (2023)

3.3 PROJETO MECÂNICO

Uma vez definido o projeto eletrônico e quais as peças serão embarcadas no robô móvel, o foco passou a ser o projeto mecânico que abrange a organização das peças eletrônicas e o design de peças mecânicas para melhor alocá-las e fixá-las no espaço contido entre os chassis do robô. Devido ao fato das dimensões do chassi e do conjunto motor-roda já serem definidas a partir da compra do Kit Chassi 4WD, o projeto mecânico começa a ser pensado partindo desse ponto e tem suas etapas de elaboração e execução resumidas no fluxograma ilustrado na **Figura 33**.

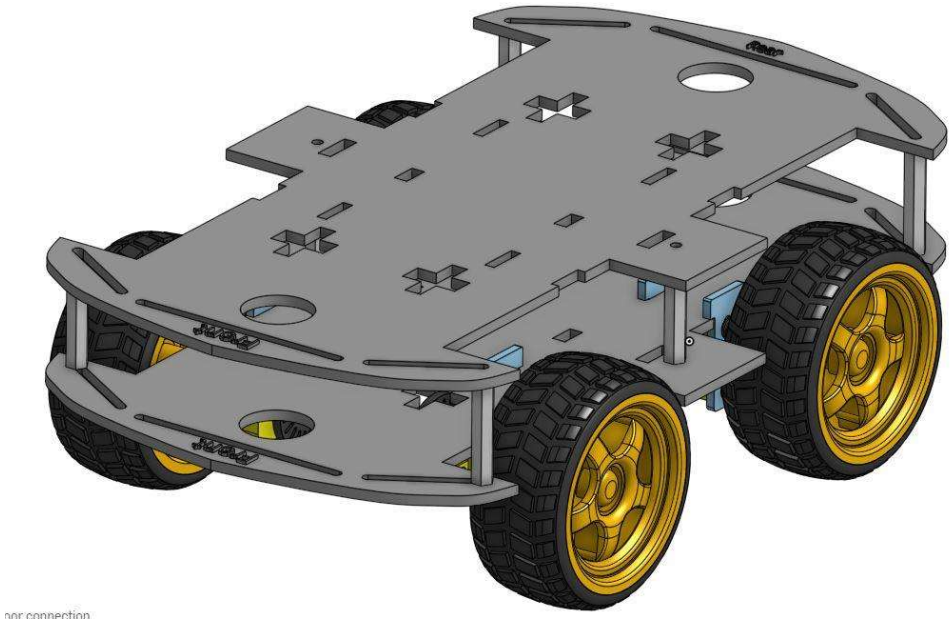
Figura 33 – Fluxograma de ações tomadas na elaboração e execução do projeto mecânico.



Fonte: Autoria Própria (2023)

A representação digital na plataforma *Onshape* do Kit usado como base para esse projeto robótico, como parte do primeiro passo do fluxograma de elaboração e execução do projeto mecânico, pode ser visto na **Figura 34**. Por ser uma solução comercial muito difundida em projetos de robótica, o conjunto motor-roda do Kit comprado foi desenhado de forma detalhada e compartilhado previamente à execução desse projeto, por usuários da plataforma *Onshape*.

Figura 34 – *Mock-up* no *software Onshape* do Kit Chassi 4WD.



Fonte: Autoria Própria (2023)

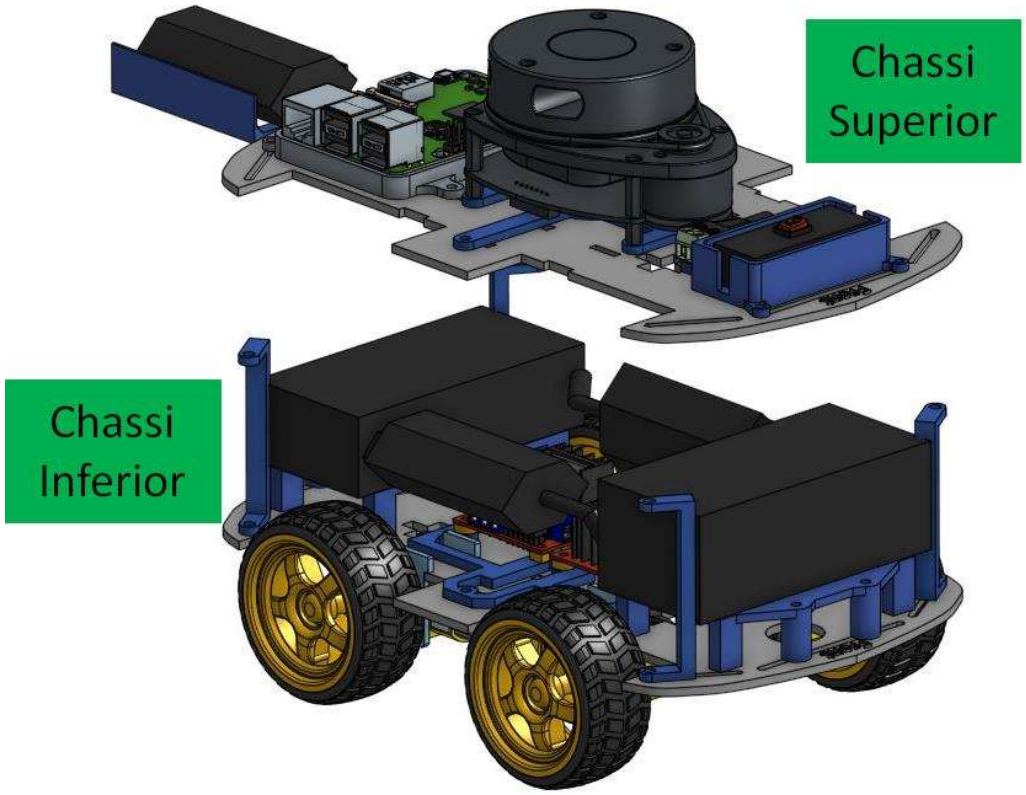
A adoção de tais desenhos neste trabalho foi uma forma rápida e eficaz de realizar a primeira etapa do fluxograma - Criar representação 3D do Kit Chassi 4WD no *software Onshape* – além de estar alinhado com a filosofia *open-source*. Isso foi possível devido ao compartilhamento de *Mock-ups*, ou seja, representações digitais de peças reais, em uma área da plataforma *Onshape* dedicada chamada de "*Public*[26]", voltada para a disseminação de conhecimento e compartilhamento de desenhos entre os membros da própria comunidade *Onshape* (LEARN ONSHAPE, 2023). Entretanto, o desenho preciso do chassi foi feito a partir de medidas realizadas no exemplar do Kit comprado, pois embora os conjuntos motor-roda sejam amplamente usado em diversas plataformas robóticas, os chassis variam de acordo com os diversos Kits comerciais disponíveis. O desenho do chassi realizado na plataforma *Onshape* pode ser encontrado no **Anexo B**.

O projeto mecânico desenvolvido neste trabalho pode ser organizado seguindo seus dois chassis: Chassi Inferior e Chassi Superior, ilustrado na **Figura 35**. O Chassi Superior é o chassi prioritário em se tratando de alocação de espaço, visto que esse receberá o sensor *RPLidar*. Sendo instalado na parte de cima do robô, seu campo de visão de 360º não deve ser obstruído e para tanto,

[26] Em uma tradução livre: Público.

deve-se manter qualquer outra parte mecânica ou eletrônica a baixo da linha de 5 cm de altura, configurando como o único pré-requisito para a organização das peças em ambos os chassis do robô. Primeiramente o Chassi Inferior será explorado com as soluções encontradas para a organização de suas peças, posteriormente o mesmo será feito com o Chassi Superior e seu pré-requisito de organização.

Figura 35 – Visão expandida do robô móvel: Chassi Superior e Chassi inferior.



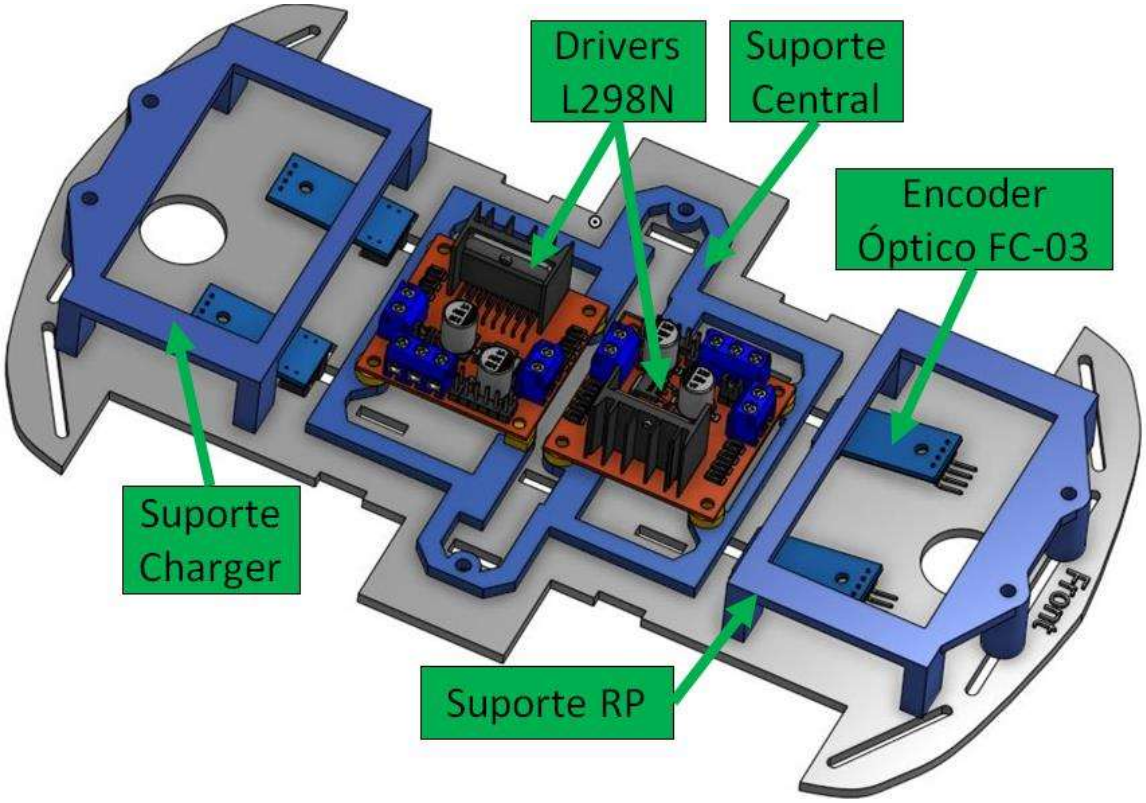
Fonte: Autoria Própria (2023)

3.3.1 Chassi Inferior

Uma vez que os motores e os discos furados acoplados aos eixos das rodas ocupam o espaço abaixo do chassi inferior, esses já possuem sua organização bem definida e não são levados em consideração no design de peças adicionais. A posição dos Encoders já está definida pelas frestas alinhadas aos eixos dos motores no Chassi, dessa forma a organização das demais peças deve levar em consideração o posicionamento e o espaço ocupado pelas placas comerciais FC-03.

Na **Figura 36,** está exposta a organização final das peças mecânicas e eletrônicas do chassi inferior. Os conjuntos motor-roda foram escondidos, assim como as duas fontes transformadoras responsáveis pela alimentação do sistema robótico, ainda que essas estejam alocadas no chassi inferior, para uma melhor visualização da organização das peças.

Figura 36 – Organização das peças mecânicas e eletrônicas no Chassi Inferior.



Fonte: Autoria Própria (2023)

Primeiramente, os *Driver*s foram alocados para o centro do Chassi, uma vez que isso os colocariam perto dos quatro motores e, portanto, facilitaria na instalação da afiação, prevista no projeto eletrônico. Para a correta fixação dessas placas, uma foi desenvolvido uma peça, chamada de “Suporte Central”, que se uni aos *Driver*s pelos seus pontos de fixação e se fixa ao chassi inferior através de dois parafusos simetricamente distantes do centro. A peça “Suporte Central” possui uma elevação de 3 mm para que os pinos elétricos das placas não tocassem no chassi e proporcionasse instabilidade mecânica, além de possuir um desenho específico projetado para

evitar furos do chassi, passagens de fios e outras peças mecânicas e eletrônicas que fazem parte da organização do Chassi Inferior como os Encoders e as peças de fixação do conjunto motor-roda.

Em seguida as duas fontes de potência se posicionam de forma assimétrica em cada extremidade longitudinal do Chassi, no entanto precisando estar a no mínimo 6 mm de altura devido à presença dos Encoders. Essa organização concede ao robô uma boa distribuição de peso e embora pernas sejam adicionadas às peças para suprir a necessidade de altura mínima, dois pontos de fixação por parafusos são incluídos no design para garantir estabilidade durante a movimentação do robô. Na extremidade frontal foi desenhado a peça de nome "Suporte *Charger*" para a fonte de 12V e 1,5 A, usada na alimentação dos motores, enquanto que a fonte de 5V e 3 A, específica para a alimentação do computador embarcado está localizada na outra extremidade, a traseira, e com ela uma peça desenhada de acordo com as suas dimensões chamada de "Suporte RP". No **Anexo B** encontram-se os desenhos das peças mecânicas do Chassi Inferior. Na **Tabela 7** estão listados os itens eletrônicos embarcados no andar inferior e as peças mecânicas desenhadas para a fixação no Chassi.

Tabela 7 – Lista de itens embarcados no Chassi Inferior.

Item	**Tipo de peça**	**Parte do Chassi**
*Driver*s L298N	Eletrônica	Central
Suporte Central	Mecânica	Central
Encoders Ópticos FC-03	Eletrônica	Espalhados
Fonte 12V	Eletrônica	Traseira
Suporte Charger	Mecânica	Traseira
Fonte 5 V	Eletrônica	Dianteira
Suporte RP	Mecânica	Dianteira

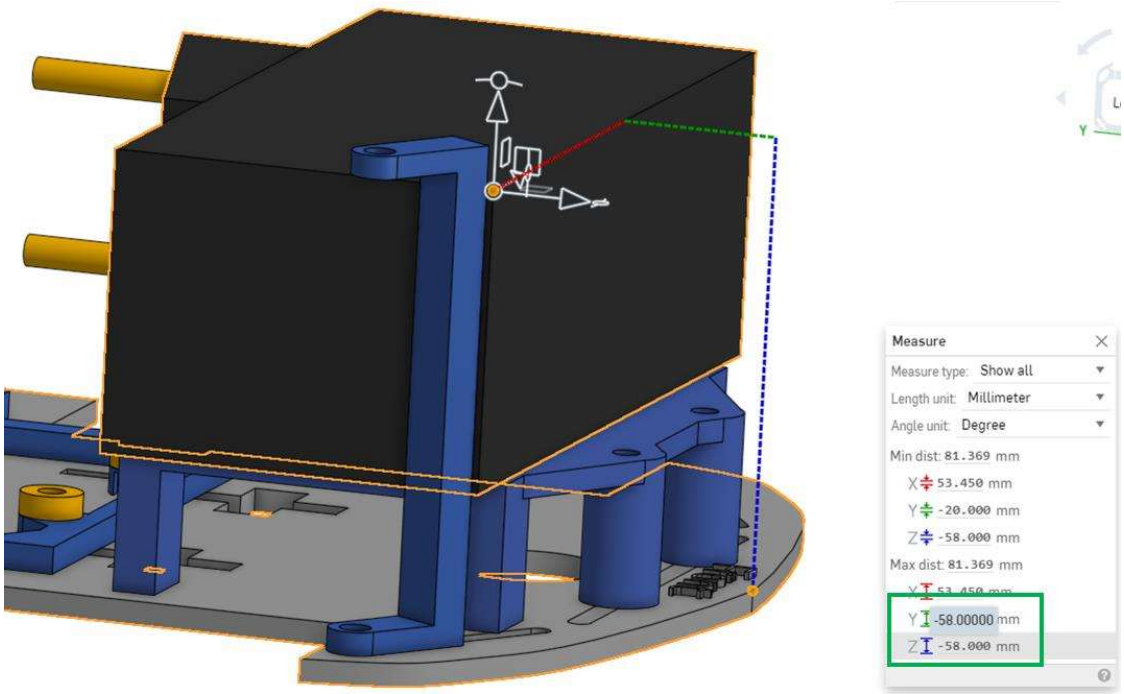
Fonte: Autoria Própria (2023)

3.3.2 Espaçadores

Após a montagem dessas peças, no ambiente digital de ferramenta *Onshape* chamado de "*Assembly*", se faz necessário o projeto de um novo espaçador que separe o Chassi Inferior do Superior, uma vez que os espaçadores originais do Kit, de 30 mm, não mais atendem as especificações do projeto mecânico. Para tanto, foi feita a medição da altura entre a face superior

do chassi inferior e a face superior da fonte de potência que mais ocupa espaço, dessa forma foi possível determinar a altura mínima para acomodar todos os itens do Chassi Inferior. Essa medida, 58mm, e o novo espaçador, projetado para permitir uma folga de 6mm, podem ser vistas na **Figura 37**. No intuito de conceder rigidez e estabilidade entre os chassis, foram fabricados seis espaçadores, onde quatro deles se fixam nos quatro cantos do chassi, e os restantes dois espaçadores são colocados nas extremidades laterais centrais do robô.

Figura 37 – Medida da altura mínima para acomodação dos itens (Espaçador com folga de 6 mm).



Fonte: Autoria Própria (2023)

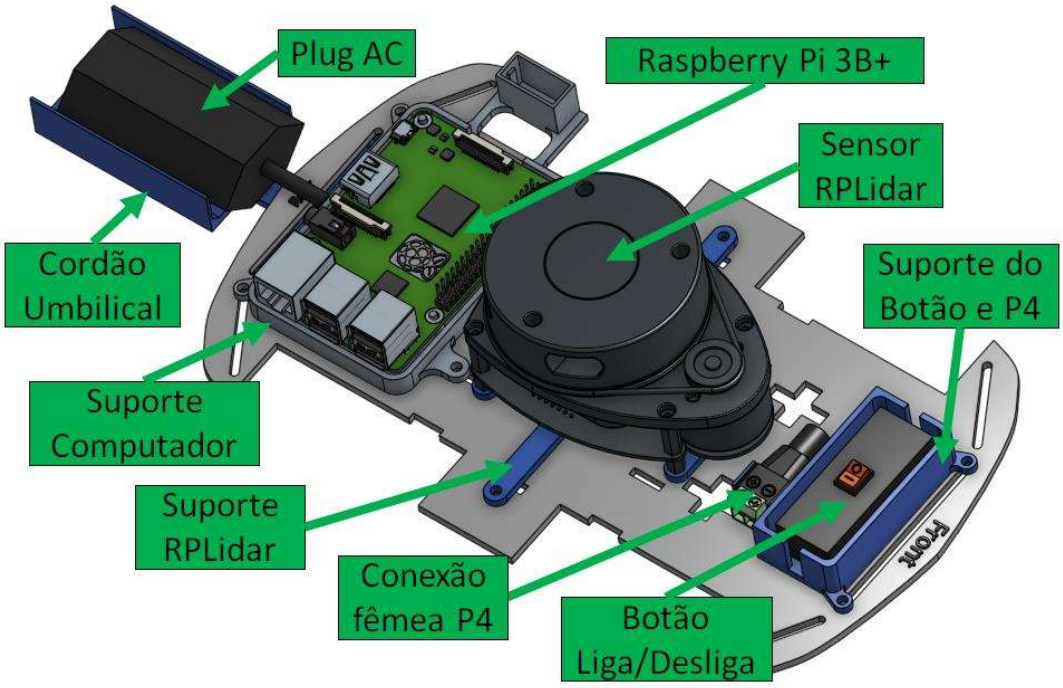
3.3.3 Chassi Superior

A segunda etapa do planejamento do projeto mecânico resume-se na organização das peças eletrônicas no Chassi Superior, esse é responsável por abrigar a peça mais importante do robô, o sensor *RPLidar*. Além do pré-requisito do sensor, o Chassi Superior herdou as peças eletrônicas que não conseguiram serem alocadas na etapa anterior de planejamento, elas são: a conexão fêmea do tipo P4 compatível com a fonte de 12V, o botão liga/desliga da fonte do computador embarcado,

além do próprio, a placa *Raspberry Pi 3B+* e o "Cordão Umbilical", usado no apoio do *plug* elétrico e a fiação que conecta à fonte de alimentação externa ao robô.

A controle da distribuição de peso foi um dos motivos que levaram a decisão de alocar as peças eletrônicas de maior peso no Chassi Inferior, dessa forma o centro de massa do robô estaria o mais próximo o possível do chão, concedendo maior estabilidade quando o robô se movimentar e manobrar dentro do ambiente a ser explorado. Outro aspecto que foi considerado no planejamento de ambos os chassis, foi a distribuição de peso ao longo do plano X, Y do robô, uma vez que quanto mais ao centro deste plano estiver o centro de massa, mais preciso será o modelo simétrico, explorado no **Tópico 2.1.2.2 Modelo *Skid-Steer***, adotado neste trabalho. Na **Figura 38,** está exposta a organização final das peças eletrônicas assim como as peças mecânicas projetadas para a fixação de todos os itens do Chassi Superior. Seguindo essas diretrizes, o sensor *RPLidar* é posicionado no centro do Chassi Superior, e para sua correta fixação, uma peça é projetada a fim de se conectar ao sensor pelos seus quatro pés e se fixar nos furos laterais do Chassi, ela é identificada pelo nome "Suporte *RPLidar*".

Figura 38 – Organização das peças mecânicas e eletrônicas no Chassi Superior.



Fonte: Autoria Própria (2023)

O computador embarcado *Raspberry Pi 3B+* pode ser colocado ao lado do sensor *RPLidar*, mais especificamente na parte traseira do Chassi, pois ele não oferece obstrução à linha de detecção do sensor. Uma peça mecânica foi desenhada para fixar a placa ao chassi, tomando cuidado com suas portas periféricas e com a entrada de cartão SD. A peça, identificada como "Suporte Computador", possui quatro pontos de fixação na placa por parafusos, quatro pontos de fixação no chassi por parafuso e um apêndice dedicado a acomodação de pinos de conexão usados na fiação entre o computador embarcado e os Encoders.

Para abrigar o botão liga/desliga da fonte do computador embarcado e a conexão fêmea compatível com a fonte de 12V, uma peça foi projetada para fixá-los, via encaixe, na parte dianteira do Chassi. Visto que nessa posição, essas peças ficam longe do computador embarcado e de suas afiações, além de proporcionarem um contrapeso aos mesmos.

A última peça a ser projetada do Chassi Superior abriga o *plug* fêmea que conecta a afiação do robô com uma tensão 127V AC externa, e permite alimentação necessária para as fontes transformadoras alocadas no Chassi Inferior. Dessa forma o robô, mesmo sendo móvel, necessita de constante conexão elétrica com uma fonte externa de potência e, portanto, possui um fio que pode ser associado a um "cordão umbilical". A lista de peças eletrônicas embarcadas no Chassi Superior e as peças mecânicas desenhadas para fins de fixação e organização é ilustrada na **Tabela 8**, assim como a indicação de qual parte do chassi a peça ocupa.

Tabela 8 – Lista de itens embarcados no Chassi Superior.

Item	**Tipo de peça**	**Parte do Chassi**
Sensor *RPLidar*	Eletrônica	Central
Suporte *RPLidar*	Mecânica	Central
Raspberry Pi 3B+	Eletrônica	Traseira
Suporte Computador	Mecânica	Traseira
Suporte Botão e P4	Mecânica	Dianteira
Botão Liga/Desliga	Eletrônica	Dianteira
Conexão Fêmea P4	Eletrônica	Dianteira
Plug Fêmea AC	Eletrônica	Traseira
Cordão Umbilical	Mecânica	Traseira

Fonte: Autoria Própria (2023)

Por fim, as etapas finais do fluxograma do projeto mecânico estão relacionadas com o levantamento do material e fabricação das peças mecânicas projetadas para ambos os chassis com o intuito de realizar a montagem e teste. As peças foram fabricadas em material PLA nas impressoras 3D disponibilizadas no laboratório de prototipagem do IFAM Campus Manaus Distrito Industrial. Os desenhos de todas as peças projetadas para o Chassi Superior são encontrados no **Anexo B**.

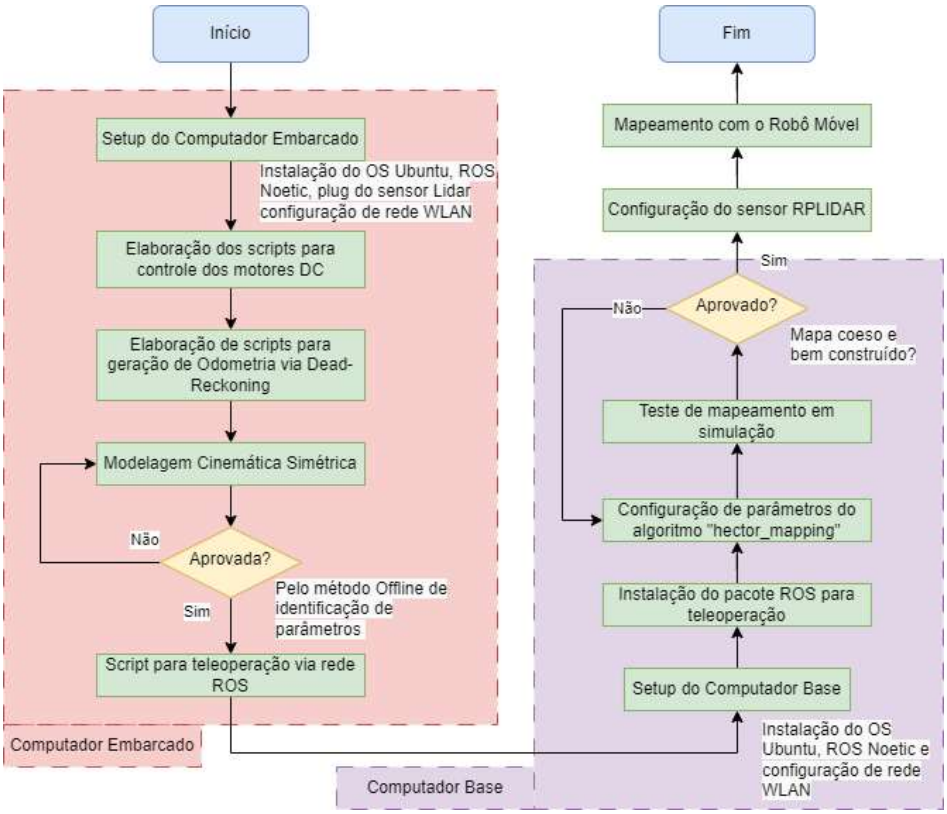
3.4 PROJETO DO *FIRMWARE*

Esse projeto utiliza dos sistemas eletrônicos e mecânicos montados e funcionais para que o *Firmware* robótico, conjunto de programas dentro do sistema ROS, possa coordenar seus recursos em prol do objetivo do robô: explorar e mapear ambientes internos. Essa etapa de implementação de algoritmos e *scripts* de controle de motores e sensores está englobado no projeto de *Firmware* e sua elaboração e execução são resumidos no fluxograma da **Figura 39**.

Uma vez que o objetivo do sistema robótico está definido, o desenvolvimento do seu *Firmware* foi dividido e organizado para rodar em duas centrais computacionais diferentes: o robô em si, composto pelo computador embarcado e do sensor *LiDAR*, e o computador base, composto por uma máquina que esteja na mesma rede *WLAN* que o computador embarcado, equipado com o OS Ubuntu e que possua as ferramentas visuais do ROS necessárias para rodar o algoritmo "*hector_mapping*". Para realizar a exploração do ambiente interno, comandos são enviados do computador base, de forma remota, para o robô, que é encarregado de executar os comandos de movimentação usando circuito de controle dos motores. Ao ponto que o robô se movimenta de forma teleoperada, as leituras feitas pelo sensor *LiDAR* são entregues ao computador embarcado e enviadas para o computador base pela mesma rede por onde recebeu os comandos de exploração.

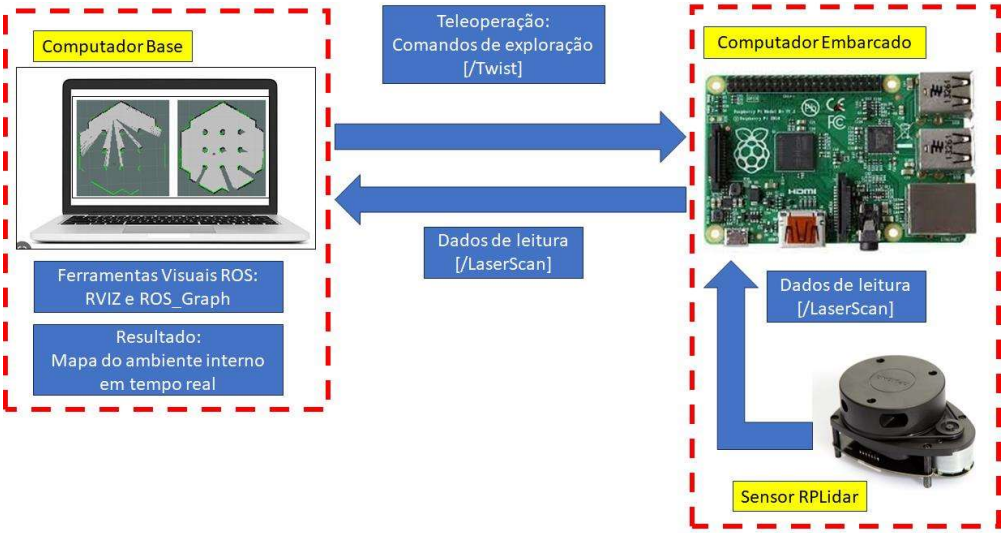
Esses dados são usados pelo algoritmo "*hector_mapping*", rodando no computador base, cujo resultado é o mapa, atualizado em tempo real, do ambiente interno onde o robô se encontra. A **Figura 40** apresenta o funcionamento macro, constituído da comunicação "Robô-Base" com o fluxo de comandos exploratórios e dados sensoriais.

Figura 39 – Fluxograma de elaboração e execução do Projeto *Firmware*.



Fonte: Autoria Própria (2023)

Figura 40 – Fluxo de informações entre as duas centrais computacionais.



Fonte: Autoria Própria (2023)

3.4.1 Robô Móvel

É a entidade física composta pela integração dos chassis e dos circuitos de controle, motores e sensores, coordenados pelo computador embarcado que resultam na modelagem cinemática simétrica do robô, capacidade de teleoperação e o envio, em tempo real, dos dados coletados pelo sensor *LiDAR* para o mapeamento. A primeira etapa do desenvolvimento do *Firmware* projetado a ser implementado no computador embarcado é o setup, ou seja, a instalação do sistema operacional usado para hospedar o ROS, a instalação específica da distribuição ROS e a correta configuração da rede *WLAN* necessária para a comunicação do robô móvel com sua base.

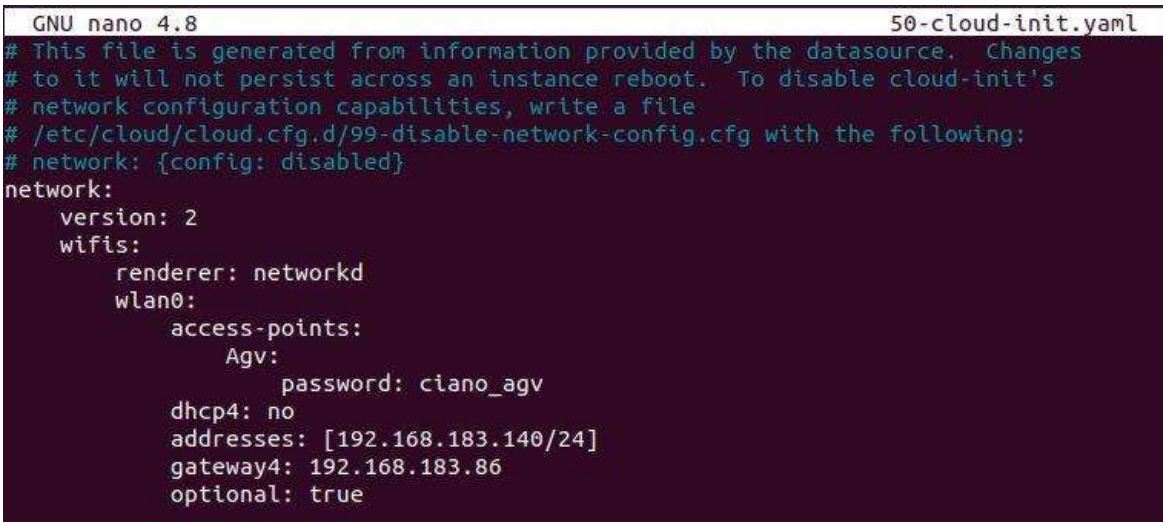
3.4.1.1 Configuração do Computador Embarcado

O sistema operacional usado neste trabalho é o Linux Ubuntu 20.04, como mostrado no **Tópico 2.4.1.3 *Raspberry Pi Imager***, onde uma versão adaptada para os usar os recursos limitados do computador embarcado, *Raspberry Pi 3B+*, é escolhida de acordo com as necessidades do projeto. A instalação é feita de forma automática ao simplesmente inserir o cartão SD cuja imagem do sistema foi gravada via o programa citado. Como essa unidade do projeto robótico não necessita de ferramentas gráficas, optou-se por apenas instalar as configurações básicas do sistema operacional, em outras palavras, o OS no computador embarcado não possui interface gráfica e dessa forma as demais configurações são feitas via terminal.

A configuração de acesso a rede *WLAN* é a etapa seguinte, uma vez que o sistema operacional está instalado corretamente, e a rede usada nesse projeto é providenciada por um aparelho celular com essa capacidade. Dessa forma cria-se uma rede exclusiva para a execução do projeto, facilitando no debug de conexão com a rede e evitando problemas que possam ser introduzidos ao sistema via dispositivos fora do escopo do projeto. Para tanto, seguindo as instruções de Young (2022), as configurações de conexão (rede e senha) e o estabelecimento de um IP estático para a placa de interface *WLAN* do computador embarcado foram realizadas. Esse processo também é feito no computador base, assim durante os inúmeros ciclos de ligar e desligar as duas centrais computacionais, essas conectam-se de forma automática sem maiores inputs do usuário. Resultando em um sistema robótico, aqui composto por Robô Móvel (Computador

Embarcado) e Computador Base, que se auto configura para funcionar de forma integrada logo após a sua energização, a tela de configuração no terminal do computador embarcado pode ser conferida na **Figura 41**.

Figura 41 – Tela configuração WLAN do computador embarcado



Fonte: Autoria Própria (2023)

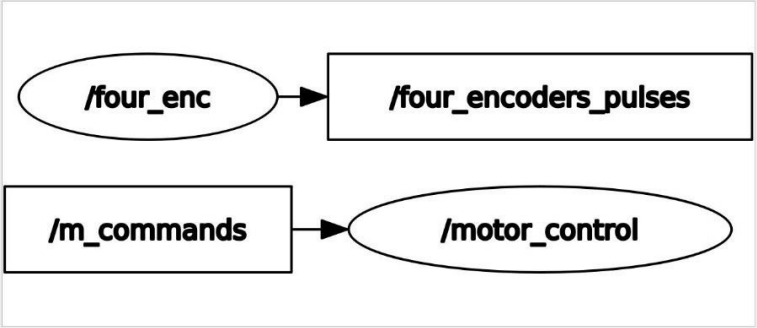
A distribuição ROS usada nesse projeto é a Noetic 1.16.0, devido a ser o mais recente da sua família, o ROS 1, no momento da instalação e ainda possuir suporte da *Open Robotics*, além de ser compatível com o OS instalado no sistema embarcado, sendo importante fazer a correta configuração do repositório Ubuntu (SAITO, 2023). Essa distribuição permite que as ferramentas ROS necessárias para funcionamento da plataforma robótica possam ser empregadas, porém, contando com as limitações computacionais da placa *Raspberry Pi 3B+*, a versão instalada é a "ROS *Base*", também conhecida como *Bare-Bones*[27], onde as ferramentas de simulação 2D/3D e ferramentas gráficas não são incluídas na instalação.

Uma vez que as principais bibliotecas do ROS são instaladas, o sistema é capaz de iniciar e sustentar uma rede de Nós de acordo com as instruções contidas nos *scripts*. Para a criação dos *scripts* de controle dos motores DC, voltados para a locomoção do robô, e do registro dos pulsos gerados nos quatro encoders, voltados para a geração de Odometria, foi necessário a instalação da

[27] Em uma tradução livre: Apenas os ossos.

biblioteca "*RPi.GPIO*" que possibilita o controle dos pinos da interface lógica para periféricos da placa *Raspberry Pi 3B+* (SOUZA, 2020). Na **Figura 42** é exposta a rede ROS composta pelo Nó **"***/motor_control*" que faz interface com os *Driver*s para a movimentação dos motores, a partir de instruções enviadas pelo tópico *"/m_commands*" e pelo Nó *"/four_enc*" que faz a interface com os pinos dos quatro Encoders para então publicar os registros pelo Tópico **"***/four_encoders_pulses*". Detalhes da programação desses *scripts* encontram-se no **Anexo A**.

Figura 42 – Rede ROS durante controle dos quatro motores e leitura dos quatro Encoders.



Fonte: Autoria Própria (2023)

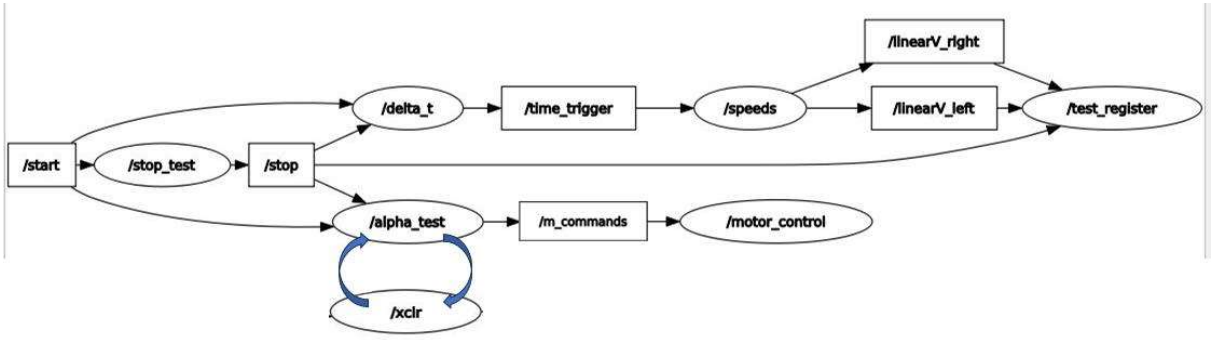
3.4.1.2 Modelagem Cinemática Simétrica

Nessa etapa do desenvolvimento do projeto de *Firmware*, as equações apresentadas no estudo da técnica de geração de Odometria por *Dead-Reckoning* e as fórmulas para determinação do modelo cinemático simétrico para robôs do tipo *Skid-Steer* são usadas. Uma vez que os *scripts* de movimentação dos motores e do registro de pulsos dos encoders estão funcionando e conectados na rede ROS, para a realização dos testes de identificação *offline* dos parâmetros, faz-se necessário a implementação de um limite temporal, em outras palavras, um delta tempo. Essa limitação temporal se traduz como uma frequência de cálculo da velocidade angular de cada uma das rodas do robô e, por consequência, das velocidades laterais individuais que levam a estimação da *Pose* do robô. Para além dessa restrição, estes testes requerem um limite de duração, ou seja, por quanto tempo cada teste será feito. Para ambos os testes, linear (α) e angular (x_{CIR}), foi determinado que a frequência de cálculo das velocidades seria de 5 Hz, ou em termos de delta tempo, 200ms, e a duração do tempo total do teste α seria de 2-s e do teste x_{CIR} seria de 1-s.

Aplicando essa nova organização da rede, resultando no ROS *Graph* da **Figura 43**, que mostra a intercambialidade dos dois Nós *"/alpha_test*" e *"/xcir*". Há a presença de mais três Nós além da substituição do Nó responsável pela interface e registro dos Encoders, acarretando quatro pontos de mudanças:

1. Respondendo a necessidade de da frequência de cálculo, apresenta-se o Nó ***"****/delta_t*", que envia os sinais de limitação temporal via Tópico "*/time_trigger*";
2. Determinando a duração total do teste, tem-se o Nó "*/stop_test*" que envia o sinal para os outros Nós dessa rede, por meio do Tópico "*/stop*", quando devem parar suas sub-rotinas;
3. Abrangendo a interface com os encoders e adicionando o cálculo da velocidade linear de cada lado do robô, esquerdo e direito, o antigo Nó "*/four_enc*" é substituído pelo Nó "*/speeds*" que disponibiliza as velocidades a cada fim do ciclo do "*/delta_t*" ("*/time_trigger*");
4. Por fim as variáveis a serem registradas, velocidades lineares de cada lado do robô, são capturadas e salvas pelo Nó "*/test_register*" para que o método *offline* possa ser implementado.

Figura 43 – Rede ROS nos testes de identificação *offline* de parâmetros.



Fonte: Autoria Própria (2023)

A organização dos Nós permite a realização dos testes e o registro das variáveis de forma automática e sem a necessidade de maiores inputs externo, uma vez que os testes se iniciam ao se enviar um sinal no tópico "*/start*" e a seleção de qual teste será realizado acontece pela retirada do Nó "*/alpha_test*" e a substituição pelo Nó "*/xcir*", ou o contrário.

Na **Tabela 9** são listadas o resultado da aplicação das fórmulas (13) e (14) a partir do registro das variáveis nas cinco repetições de ambos os testes, assim como as suas médias aritméticas, identificados como valores finais de α e $x_{CIR,}$ que quando aplicados na fórmula (12) permitem encontrar a Matriz A característica desse robô móvel. Em ordem de se atender às restrições de terreno estabelecidas para a correta modelagem cinemática de robôs do tipo *Skid-Steer*, os testes foram realizados nas salas de laboratório do Polo de Inovação Manaus / IFAM.

Tabela 9 – Valores dos parâmetros encontrados nos cinco testes realizados

Número	**Valor de α**	**Média dos Valores (Valor Final)**	**Valor de x_{CIR}**	**Média dos Valores (Valor Final**
1	0,55013	0,46721	-0,0285	-0,03256
2	0,39297		0,04504	
3	0,43321		-0,03648	
4	0,56198		0,02087	
5	0,39780		0,01625	

Fonte: Autoria Própria (2023)

O resultado dessa etapa do Projeto de *Firmware* é a criação de um script para a construção de um Nó chamado "*/odometry*" que realize as seguintes etapas:

1. Utilize a Matriz A característica desse robô, combine com dados de velocidade lineares, obtidos a partir dos Tópicos gerados pelo Nó "*/speeds*";
2. Gere a estimação da *Pose* em tempo real das coordenadas x, y e orientação no eixo z, usando as equações (8), (9) e (10), porém, primeiro adaptando as equações de velocidade linear total e velocidade angular do robô usando as equações (15) e (16), respectivamente;
3. E as disponibilize no formato padronizado, Mensagem ROS "*Pose*", através do Tópico "/*Pose_robot*".

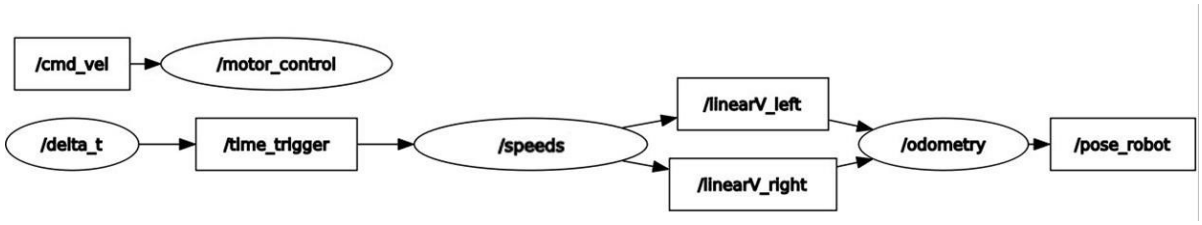
O Nó "*/test_register*", uma vez que cumpriu seu papel de registrar as variáveis que determinam os parâmetros durantes os testes, é substituído pelo novo Nó, "*/odometry*" que pertence a configuração final da Rede ROS referente ao computador embarcado. Programação, em

Python, dos *scripts* dessa etapa do projeto *Firmware* e detalhes dos testes podem ser conferidos no **Anexo A**.

3.4.1.3 Teleoperação - Robô Móvel

Por último, na sequência de desenvolvimento do projeto de *Firmware* no computador embarcado, o robô móvel deve adquirir a capacidade de ser teleoperado, resultando na possibilidade de receber comandos de exploração de um dispositivo externo ao chassi robótico e integrar esses comandos à já existente rede ROS. A principal mudança no sistema é a modificação no Nó "*/motor_control*" que embora continue a fazer a interface com os *driver*s para a movimentação controlada dos motores, agora recebe comandos pelo tópico "*/cmd_vel*", configurado para receber Mensagens ROS do tipo "*Twist*". Dessa forma o projeto se torna compatível com um dos Pacotes ROS mais difundidos para o controle de plataformas móveis (PURVIS, 2015).

Figura 44 – Rede ROS em funcionamento após as modificações para a teleoperação.



Fonte: Autoria Própria (2023)

O resultado da rede ROS até este ponto do desenvolvimento do projeto de *Firmware* pode ser conferido na **Figura 44** que ilustra a rede ROS em vigência. Uma reorganização da rede é feita e os tópicos "*/start*" e "*/stop*", relacionados com os testes de modelagem cinemática, não são mais utilizados dando espaço para uma cadeia simples de detecção dos pulsos dos encoders, geração da odometria em tempo real através dos Nós "*/speeds*" e "*/odometry*" e publicada da estimativa da *Pose* pelo Tópico "*/Pose_robot*", ainda obedientes à frequência de cálculo ditada pelo Nó "*/delta_t*". A rede ROS se apresenta aqui ainda mais simples e menos populada por Nós, tendência que aparecerá nas demais etapas de desenvolvimento do Projeto de *Firmware* do robô móvel. A

programação, em Python, dos *scripts* que conferem ao robô ser teleoperado pode ser conferida no **Anexo A**, assim como a estruturação da nova Mensagem ROS usada nessa etapa.

3.4.2 Computador Base

É o Computador Laptop equipado com OS compatível com o que foi instalado no computador embarcado e possui a mesma distribuição ROS, dessa forma a comunicação entre as duas centrais computacionais acontece sem maiores problemas, sendo necessário apenas a configuração da rede WLAN. A elaboração dessa parte do Projeto de *Firmware* é focada na preparação dos Parâmetros ROS com o intuito de proporcionar a integração dos Nós e Tópicos desenvolvidos neste computador com o robô móvel e sua Rede ROS. Antes de estabelecer comunicação entre algoritmo "*/hector_mapping*" e o Nó de comandos teleoperados, é necessário testar e configurar a aplicação de mapeamento em simulação, etapa do desenvolvimento desempenhada dentro do Computador Base, que também possui *software* de simulação de robôs.

3.4.2.1 Configuração do Computador Base

A configuração da máquina destinada a ser o Computador Base começa com a instalação do sistema operacional Ubuntu, e uma vez que esse deve ser compatível com o OS do computador embarcado, a versão 20.04 é selecionada. Após esse passo, o mesmo é feito com a seleção da distribuição ROS, a Noetic, no entanto a versão completa é instalada com todas as ferramentas de simulação 2D/3D e com os recursos visuais de meta-análise, tendo em vista que será nesse computador que visualização do mapa acontecerá assim como a visualização da árvore TF e do ROS *Graph*.

Da mesma forma que foi necessário estabelecer um IP estático no computador embarcado, o mesmo é feito nessa unidade computacional e uma vez que essas configurações são feitas, a etapa de configuração dos parâmetros ROS se inicia, sendo eles o *ROS_MASTER_URI* e o *ROS_IP*. Essa etapa, na verdade, é feita nas duas centrais computacionais, pois se for feita em somente uma, a conexão entre as redes ROS não consegue ser estabelecida e o fluxo de mensagens não consegue funcionar corretamente acarretando na perda de dados (NIEWINSKI, 2020).

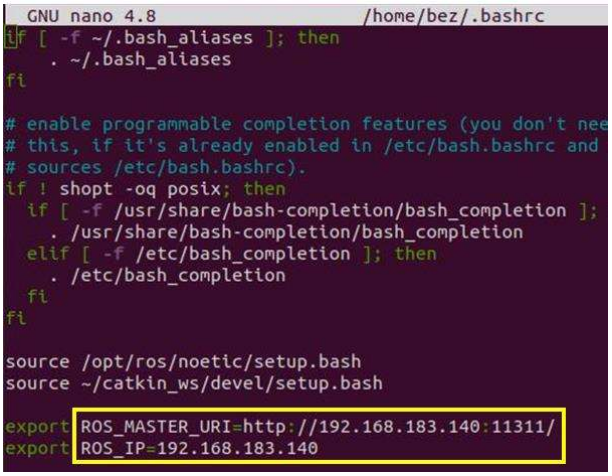
A modificação dos Parâmetros ROS é ilustrada na **Figura 45**, onde a tela do computador embarcado é exposta, e na **Figura 46**, a tela do computador base. A configuração funciona da seguinte forma:

1. Primeiramente a decisão de onde alocar o Nó Mestre deve ser feita, nesse caso o computador embarcado, de endereço IP: 192.168.183.140, é escolhido.
2. Tomada essa decisão, o Parâmetro ROS chamado de *ROS_MASTER_URI* deve receber o endereço IP do computador escolhido para abrigar o Nó Mestre, através do comando "export ROS_MASTER_URI = http://192.168.183.140:11311/", e essa modificação deve estar presente nos dois computadores, exatamente igual para ambos.
3. A configuração do Parâmetro *ROS_IP* deve ser feito de acordo com o endereço IP de cada computador, logo no caso do computador embarcado e do computador base, devem ser, respectivamente, da seguinte forma:

   "export ROS_IP = http://192.168.183.140".

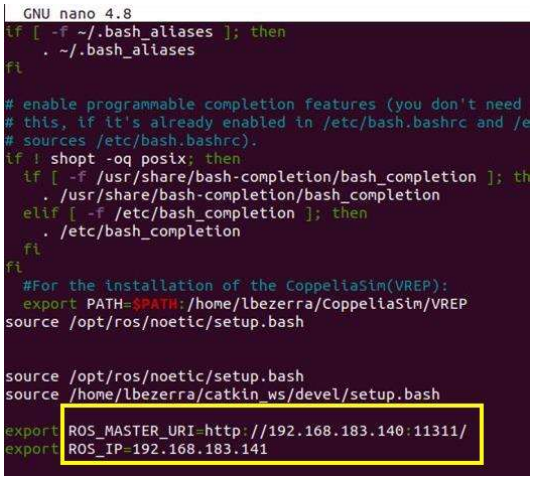
   "export ROS_IP = http://192.168.183.141".

Figura 45 – Tela de configuração dos Parâmetros ROS no computador embarcado



Fonte: Autoria Própria (2023)

Figura 46 – Tela de configuração dos Parâmetros ROS no computador base.



Fonte: Autoria Própria (2023)

Esses comandos são escritos nos documentos "bashrc" de cada computador, localizados nas suas respectivas pastas *root*[28]. Dessa forma, qualquer atividade que ROS executar por qualquer

[28] Em uma tradução livre: Raiz, endereço de documentos mais básico da OS.

que seja o computador, ela virá com seu IP de identificação e sempre estará se referenciando ao mesmo endereço de Nó Mestre, o que evita erros de transporte de informações entre Nós e Tópico, independente de qual *hardware* esteja enviando, recebendo ou processando os dados.

3.4.2.2 Teleoperação – Computador Base

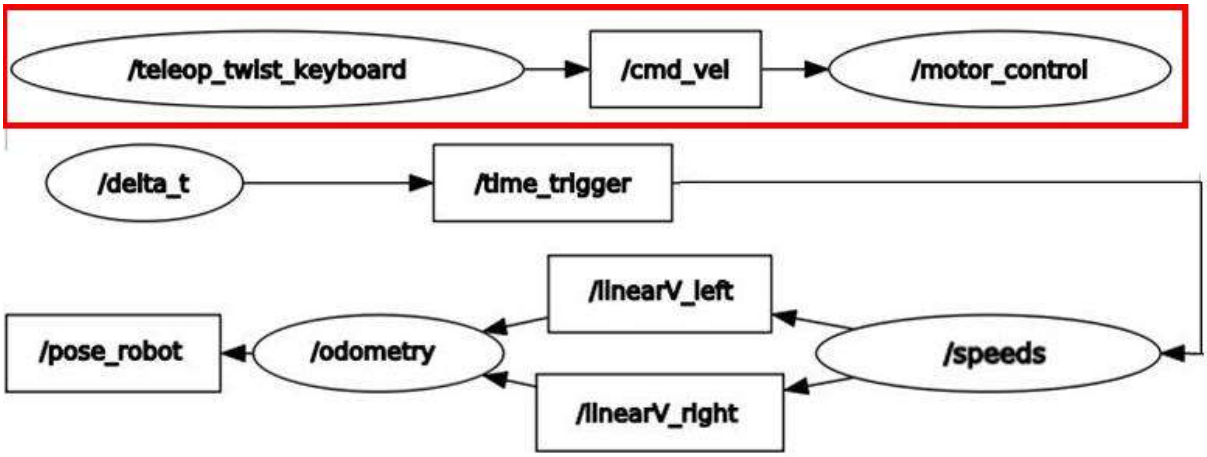
O computador base tem a função de enviar os comandos de exploração ao robô e proporcionar uma interface de controle ao usuário. Essa demanda da robótica já foi solucionada pela equipe de design do próprio ROS ao disponibilizarem um pacote que fornece uma rápida e simples forma de controlar projetos robóticos, tanto para aplicações em campo, como aplicações de simulação (VILLENA, 2017).

Pensando na compatibilidade de comandos, na rede ROS gerada pelo computador embarcado existe um Nó que recebe os comandos pelo Tópico "*/cmd_vel*", padronizado para receber Mensagens ROS do tipo "*Twist*" e no computador base, que tem o pacote em questão instalado, aplica-se o script que vem pré-programado, gerando o Nó "*/teleop_twist_keyboard*" que apresenta uma interface de controle para o usuário, **Figura 47**, e envia os comandos de controle para o Tópico "cmd_vel" no formato compatível com o que foi programado no computador embarcado, como ilustrado na **Figura 48**, onde a rede ROS anterior é conectada pelo Nó de teleoperação, em destaque, desenvolvido na etapa atual do Projeto *Firmware*.

Figura 47 – Tela de interface do script "teleop_keyboard_twist.py".

Fonte: Autoria Própria (2023)

Figura 48 – Rede Ros da Teleoperação habilitada em ambos os computadores.



Fonte: Autoria Própria (2023)

3.4.2.3 Teste do Mapeamento em Simulação

Se tornando cada vez mais frequente no desenvolvimento de projetos robóticos, a prática de testar e ajustar algoritmos em ambientes de simulação oferece a capacidade de experimentar novas abordagens e novas técnicas sem pôr em risco a integridade de robôs físicos e, em uma ótica mais profissional, permite o desenvolvimento das soluções de *Firmware* de um projeto robótico sem precisar esperar pela fabricação de protótipos, processo que pode demorar meses. Uma vez que, se projetado corretamente, a construção de uma Rede ROS pode ter o Nó de simulação de um robô móvel removido e substituído pelo Nó de um robô real, funcionando sem problemas de compatibilidade entre Tópicos e Mensagens Ros padronizadas e, portanto, sem prejuízos no resultado final.

A etapa de simulação, neste trabalho, foi feita com *software Gazebo* rodando o robô virtual do *DiffBot*, **Figura 49**, que foi projetado, construído e previamente simulado pelo projetista Pucher (2021). No robô *Diffbot*, a integração do controle dos circuitos eletrônicos, teleoperação, aquisição e envio de dados dos sensores, é feito pelo ROS, no entanto há duas diferenças entre os projetos que se destacam:

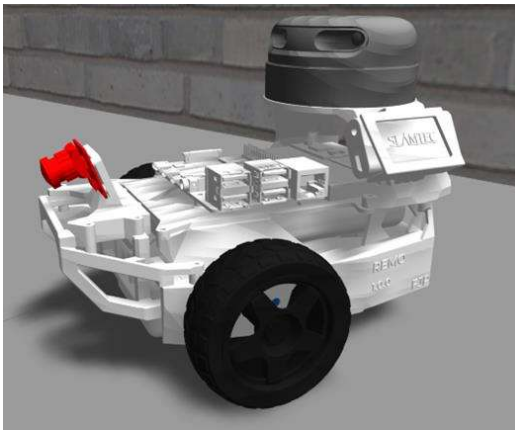
1. O sensor *LiDAR* usado no *Diffbot* é o *RPLidar A2*, indicado na **Figura 50**, ou seja, da mesma linha de sensores da *SLAMTEC* que é empregado no robô físico deste

trabalho, o que proporciona compatibilidade de Tópicos e a padronização de Mensagens ROS.

2. A construção do robô é do tipo diferencial, que embora tenha uma cinemática diferente do robô físico deste trabalho, modelo *Skid-Steer*, não causa impacto no funcionamento da Rede ROS pois é abstraída pela simulação no *Gazebo*, sendo necessário apenas usar do pacote "*teleop_twist_keyboard*" para executar a teleoperação, da mesma forma que será usado no robô físico.

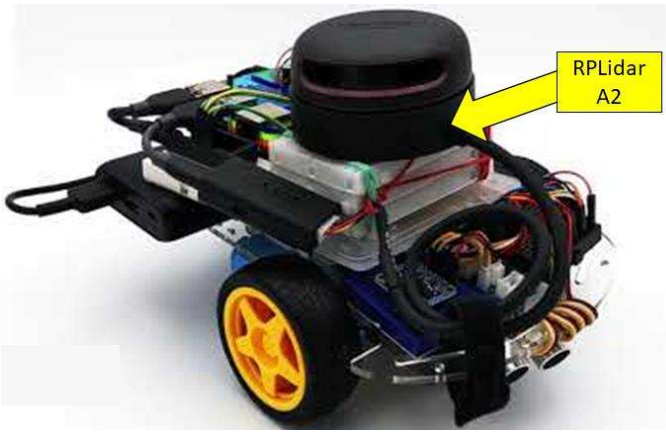
A integração da simulação no *Gazebo* do robô *Diffbot* com o algoritmo *hector_mapping* se dá, de forma contida, no computador base e a primeira etapa é fazer o download dos documentos virtuais que simulam o robô no endereço: https://github.com/ros-mobile-robots/diffbot#pencil-license. No lado esquerdo da **Figura 51** tem-se a tela do *Gazebo* mostrando a simulação do robô explorando o ambiente e os feixes de luz em azul do sensor *LiDAR*, essa configuração pode ser alcançada ao seguir as instruções contínas no link, logo em seguida o algoritmo *hector_mapping* é lançado, e com alguns ajustes de Parâmetros ROS, se integra a rede ROS gerada assim como no caso do pacote de teleoperação, cuja a interface pode ser vista no centro da **Figura 51**, gerando o mapa do ambiente simulado em tempo real, como pode ser visto na tela da ferramenta visual do ROS chamada de RVIZ ao lado direito da **Figura 51**.

Figura 49 – *Mock-up* virtual do *DiffBot* no *software Gazebo*.



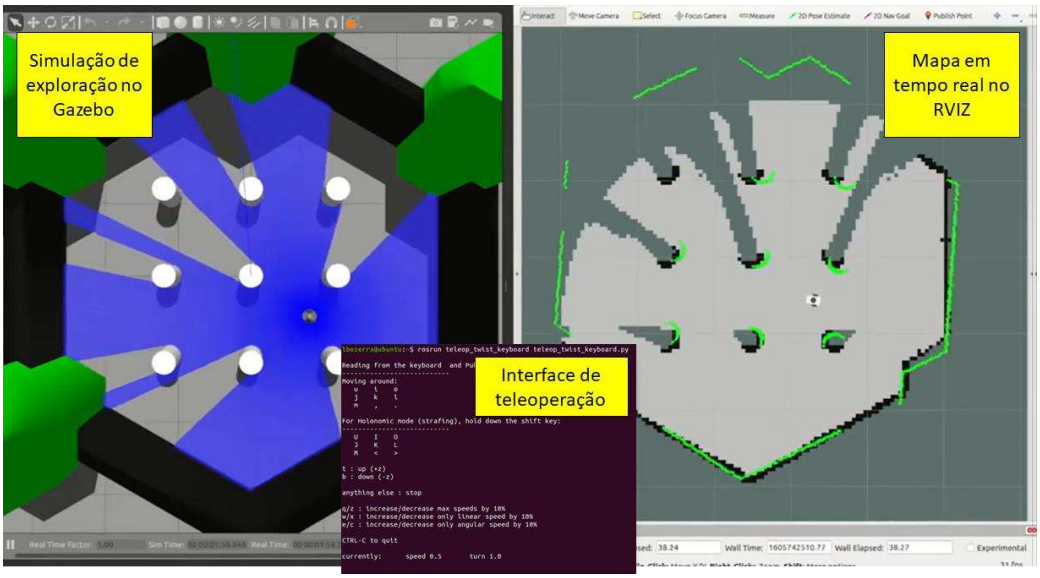
Fonte: Pucher (2021)

Figura 50 – Robô *DiffBot* real com indicação do seu sensor *RPLidar A2*.



Fonte: Modificado de Pucher (2021)

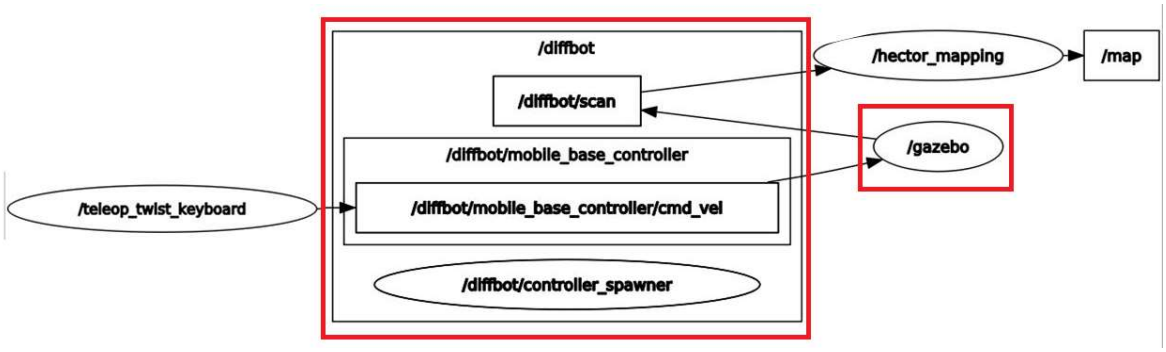
Figura 51 – Tela do computador base durante exploração e mapeamento do ambiente simulado.



Fonte: Autoria Própria (2023)

Após o mapeamento ser feito por completo, a função "*saver*" do pacote "*map_server*" é acionada, ao digitar o seguinte comando no terminal: "$ rosrun map_server map_saver -f 'file' ", onde "*file*" é o nome a ser dado para o novo mapa a ser salvo. A rede Ros gerada a partir dessa etapa do projeto de *Firmware* pode ser vista na **Figura 52**, e em destaque vermelho, tem-se os Nós e tópicos que serão substituídos pela Rede ROS proveniente computador embarcado no robô móvel físico construído neste trabalho. Os detalhes de configuração dos Parâmetros ROS usados nos testes de mapeamento em simulação podem ser conferidos no **Anexo A**.

Figura 52 – Rede ROS durante a exploração e mapeamento do ambiente simulado.



Fonte: Autoria Própria (2023)

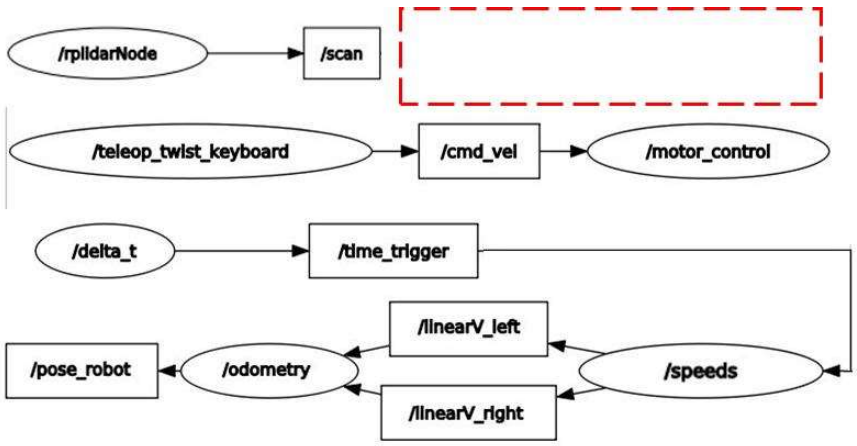
3.4.3 Integração Final – Robô/Base

Essa é a última etapa de desenvolvimento do projeto *Firmware* e consiste em usar da rede ROS elaborada até o momento, das configurações do algoritmo de mapeamento testadas em simulação e aplicar no robô físico de forma que ambas as centrais computacionais funcionem como um só sistema robótico. Para tanto, configurações no sensor principal, sensor *LiDAR*, devem ser feitas e a lógica de fluxo de dados para a execução do mapeamento deve ser implementada.

3.4.3.1 Configuração do Sensor *RPLidar*

A linha de sensores da *SLAMTEC* tem a proposta de proporcionar dispositivos de fácil manuseio, o sensor *RPLidar A1* possui uma interface simples e uma vez conectado ao computador embarcado, via USB, ele está pronto para uso, pois essa porta já estabelece a fonte de alimentação necessária para seus circuitos e providencia uma comunicação rápida e segura com a unidade computacional. No entanto, para fins de uso de forma automatizada pela programação contida nos *scripts* da Rede ROS, Desai (2022) ensina como o endereço periférico do sensor deve ser salvo na memória do computador que o usa, nesse caso sendo o computador embarcado, para que os dados captados consigam ser disponibilizados via Tópico "/scan" em formato de mensagem padronizado.

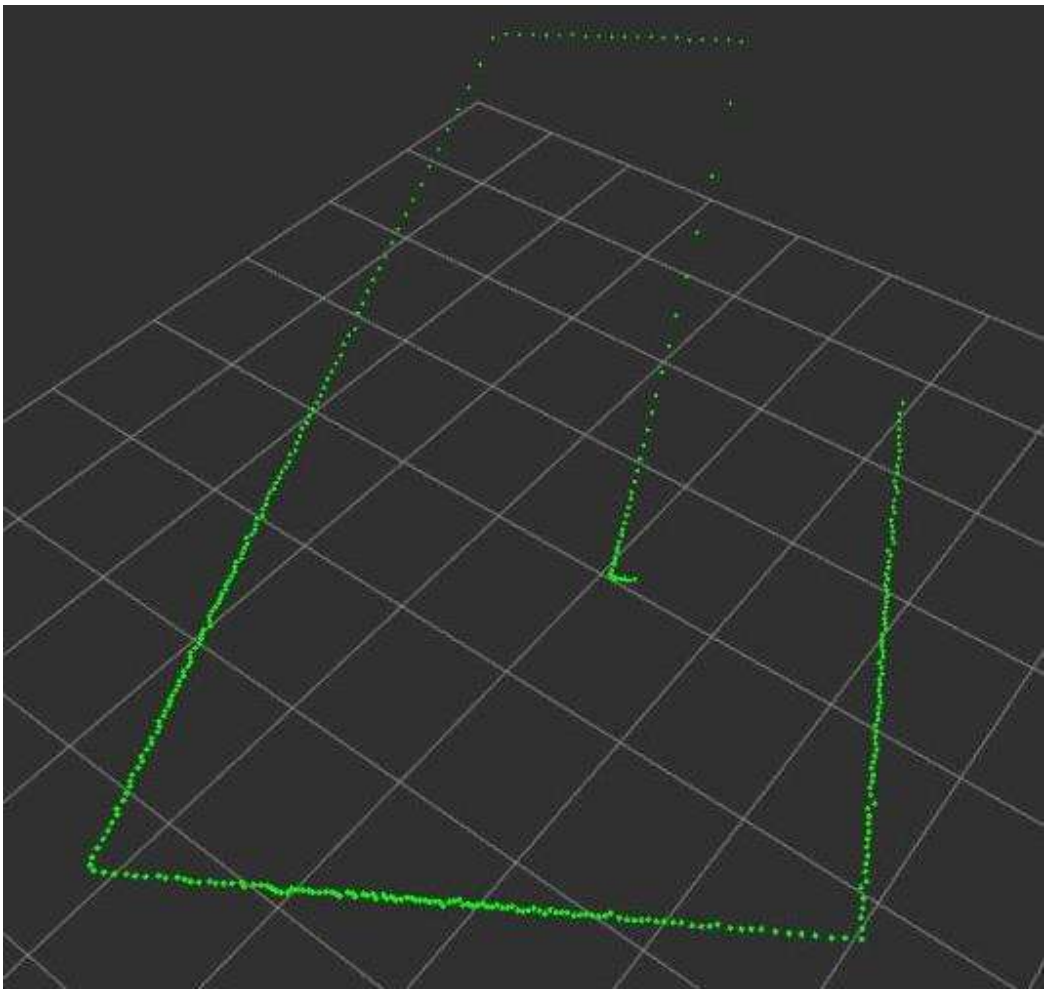
Figura 53 – Nó do Sensor *RPLidar* funcionando e integrado a Rede ROS estabelecida.



Fonte: Autoria Própria (2023)

Com essa configuração feita, o sensor está pronto para ser integrado a rede ROS e ao executar o pacote "*/rplidar_ros*", que pode ser baixado pelo link: https://github.com/Slamtec/RPLidar_ros, o novo Nó "*/rplidarNode*" faz a interface entre o sensor e a Rede Ros estabelecida, como pode ser visto na **Figura 53**, detalhe para o retângulo tracejado que indica onde o algoritmo de mapeamento se encaixa futuramente assim como ele se conectou na simulação. A validação do correto funcionamento da Rede ROS pode ser feita pela visualização, na ferramenta RVIZ, da sua leitura do ambiente em que o sensor está contido, na **Figura 54**.

Figura 54 – Validação do funcionamento do sensor *RPLidar* pelo RVIZ.



Fonte: Autoria Própria (2023)

3.4.3.2 Mapeamento com o Robô Móvel

Com todas as partes eletrônicas e mecânicas montadas e funcionando, com todas e todas as configurações de Nós e Tópicos ROS finalizadas, e com o algoritmo de mapeamento apto e validado, o sistema robótico como um todo está pronto para ser inicializado. Para realizar tal função, dois arquivos do tipo launch são executados, um em cada central computacional, no computador embarcado o arquivo cria a Rede ROS, pois ele abriga o Nó Mestre, e "chama" os *scripts* que exercem as funções relacionadas ao robô móvel, o mesmo acontece no arquivo launch no computador base. A relação de central computacional e das funções "chamadas", desenvolvidas nessa etapa do Projeto *Firmware*, está exposto na **Tabela 10**. A programação dos arquivos launch e dos *scripts* chamados são expostos no **Anexo A**.

Uma vez que todas as funções estejam online na Rede ROS, a teleoperação do robô para a exploração do ambiente interno pode começar, os dados vindos do sensor *RPLidar* são enviados ao algoritmo *hector_mapping* e o mapa em construção pode ser visualizado na ferramenta RVIZ. A rede ROS pode em funcionamento é ilustrada na **Figura 55**, pela ferramenta de meta-análise ROS *Graph*, sendo possível apenas no computador base, uma vez que somente ele possui essa capacidade. Os fluxos de informações são organizados pelos quadros roxo, verde e vermelho, responsáveis pela aquisição e envio dos dados de leitura do sensor *LiDAR* para a geração do mapa, pela teleoperação, e pela geração da odometria, respectivamente.

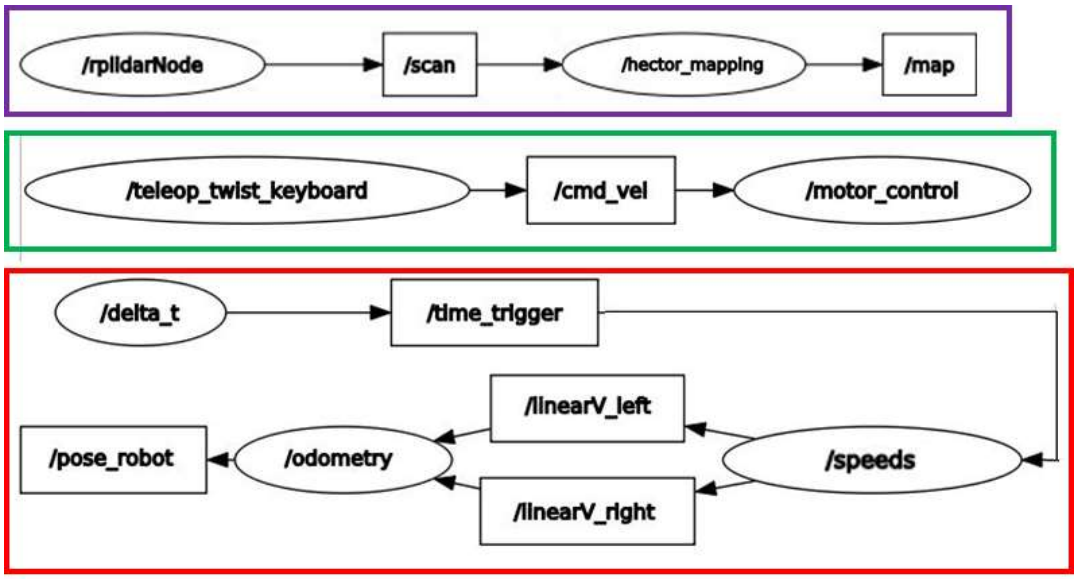
Para além dessa representação gráfica, a árvore de quadros (TF) do projeto robótico, é ilustrada na **Figura 56**, onde os quadros "*map*" (mapa), "*base_link*" (base robótica móvel) e "*rplidar_laser_link*" (Sensor *RPLidar*), além do "*scanmatcher_frame*", nativo da aplicação *hector_mapping*, são organizados e suas relações entregam a estrutura que o algoritmo necessita para construir o mapa. Ao fim do mapeamento, o mesmo processo de salvamento das informações via pacote "*map_server*", realizada na etapa de simulação, é executado de forma manual pelo usuário, finalizando assim a apresentação das funcionalidades do sistema robótico. Essa estrutura gerada pelo próprio algoritmo, e suplementado pela execução do pacote "*tf2_ros*" no arquivo launch do computador embarcado, cria as condições de odometria a partir dos dados do sensor *LiDAR* necessárias para que o pacote "*hector_mapping*" não precise de uma geração de odometria fora de seu próprio escopo.

Tabela 10 – Lista de *scripts*, pacotes e parâmetros "chamados" em cada arquivo launch.

Central Computacional (nome do arquivo launch)	**Itens**	**Função**
Robô (robot.launch)	Script "*motor_control*"	Nó responsável por receber os comandos de movimentação pelo Tópico "*/cmd_vel*" e acionar os motores do robô móvel através da interface de pinos do *Raspberry Pi 3B +*
	Script "*delta_t*"	Responsável por gerar a frequência de cálculo para a geração de odometria.
	Script "*Speeds*"	Nó responsável por registrar os pulsos dos quatro encoders do robô móvel e entregar ao nó "*/odometry*".
	Script "*/odometry*"	Nó responsável por gerar a odometria a partir dos dados enviados pelo Nó "*/wheels_status*".
	Pacote *RPLidar_ros*	Criação do Nó responsável por fazer a interface sensor *RPLidar* e Rede Ros, entregar os dados de leitura pelo Tópico "*/scan*".
Computador Base (base.launch)	Pacote de Teleoperação	Criação do Nó responsável por oferecer interface de comandos a serem enviados ao robô móvel, ao se conectar com o Tópico "*cmd_vel*".
	Parâmetros "*frame_names*"	Criação dos quadros "*map*", "*base_link*" e "*scanmatch_frame*".
	Parâmetros do mapa	Estabelecer os valores de criação do mapa como resolução, tamanho máximo, *threashold*, correção de ângulo mínimo e máximo, localização do ponto inicial do mapa.
	Pacote *tf2_ros*	Criação do quadro TF do sensor *RPLidar* (*RPLidar_laser_link*) e sua relação com a base robótica (*base_link*).
	Pacote *hector_mapping*	Algoritmo que usa dos parâmetros previamente configurados e dos dados do sensor para realizar o mapeamento.
	Pacote RVIZ	Ferramenta de visualização dos dados do sensor *RPLidar* e do mapa sendo gerado pelo algoritmo.
	Pacote *rqt_graph*	Ferramenta de visualização da rede ROS.
	Pacote *rqt_tf_tree*	Ferramenta de visualização da árvore de quadros (TF) da rede ROS.

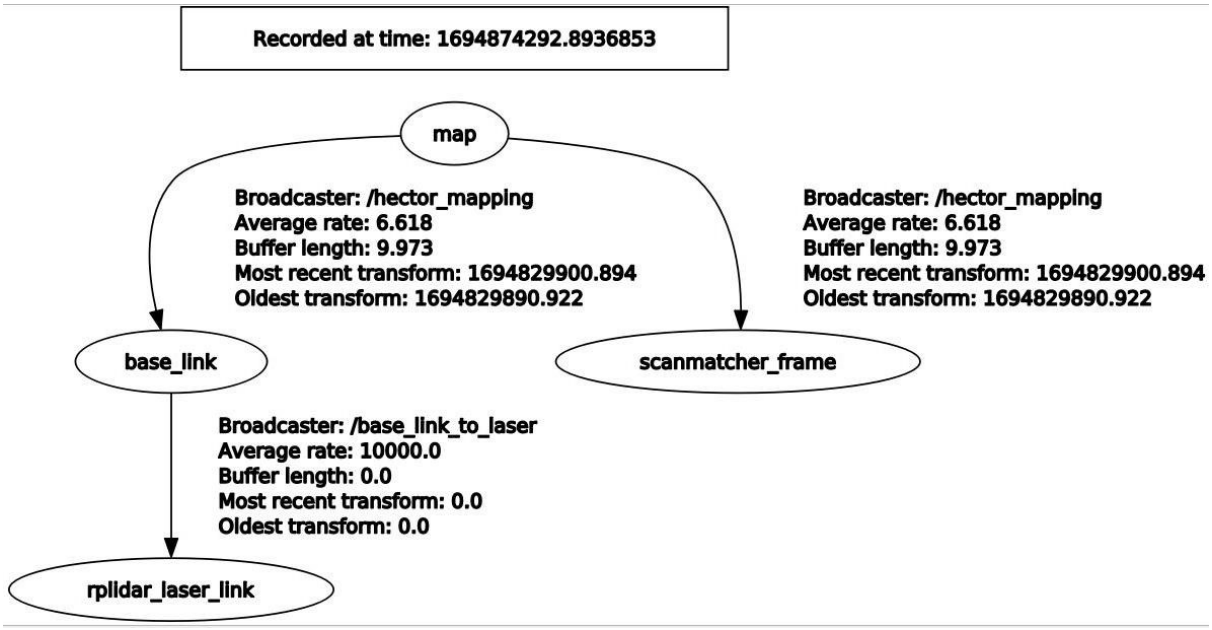
Fonte: Autoria Própria (2023)

Figura 55 – Rede ROS da integração final de todas as funções do Projeto *Firmware*.



Fonte: Autoria Própria (2023)

Figura 56 – Árvore de quadros TF da integração final do Projeto *Firmware*.



Fonte: Autoria Própria (2023)

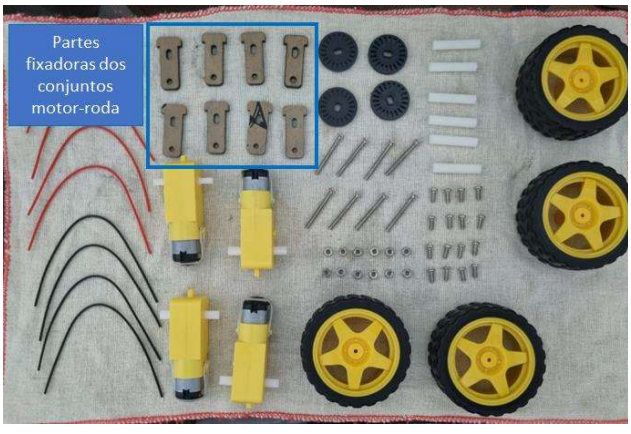
4 RESULTADOS

Nesta seção os resultados previstos na metodologia são expostos e analisados, organizados pela montagem das partes eletrônicas e mecânicas do robô móvel e pelos mapas resultantes da exploração de ambientes com obstáculos.

4.1 MONTAGEM DO ROBÔ MÓVEL

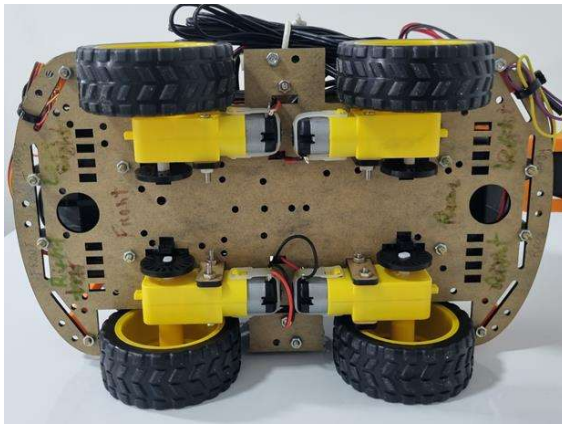
A montagem do robô móvel tem início nos quatro conjuntos motor-roda, no chassi inferior, usando das peças auxiliares que acompanham o Kit Chassi 4WD, como pode ser visto na **Figura 57**. As rodas do robô móvel, de acordo com o Projeto Mecânico, devem ficar na face inferior do chassi inferior, ilustrado na **Figura 58**, concedendo estabilidade durante a locomoção da plataforma robótica e distribuindo bem a carga entre as quatro rodas. A análise que pode ser feita da montagem das rodas e do seu funcionamento está condicionada à velocidades baixas, visto que essa é uma das imposições necessárias para a correta modelagem cinemática, resultando em uma locomoção uniforme e sem solavancos na tentativa de mitigar possíveis derrapagens durante a execução de manobras.

Figura 57 – Exposição das peças originais do kit Chassi 4WD.



Fonte: Autoria Própria (2023)

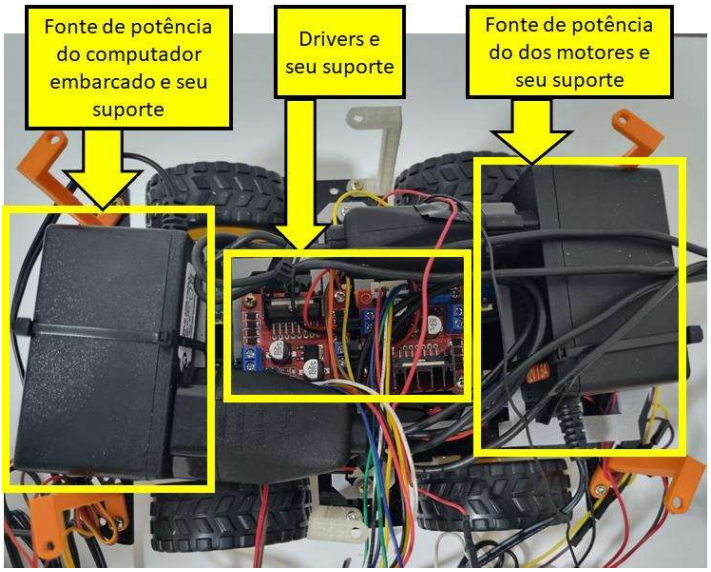
Figura 58 – Visão inferior do robô: Montagem dos conjuntos motor-roda.



Fonte: Autoria Própria (2023)

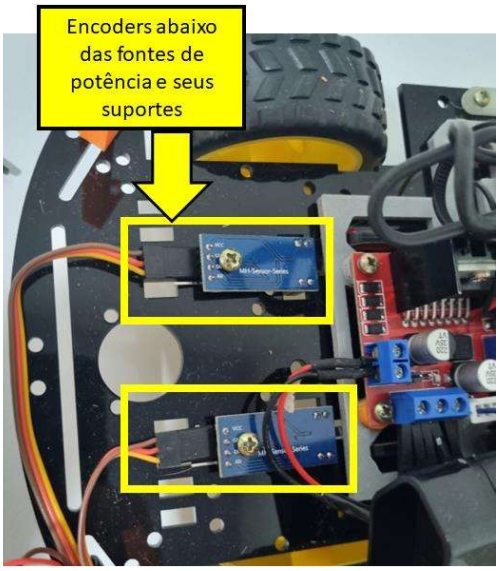
As peças mecânicas projetadas para a organização e fixação das peças eletrônicas, em ambos os chassis, foram fabricadas na impressora 3D e a montagem do chassi inferior, no seu estágio final: com a afiação conectada, pode ser conferida na **Figura 59**, e na **Figura 60** é exposto o detalhe do encaixe das placas Encoder que se localizam abaixo das fontes de transformação e seus suportes. A montagem da afiação, seguindo os esquemáticos do Projeto Eletrônico, teve início na etapa da montagem dos itens designados ao Chassi Inferior, uma vez que todos os itens eletrônicos desse andar do robô fazem interface com o computador embarcado localizado no Chassi Superior.

Figura 59 – Montagem final, com afiação, dos elementos do Chassi Inferior.



Fonte: Autoria Própria (2023)

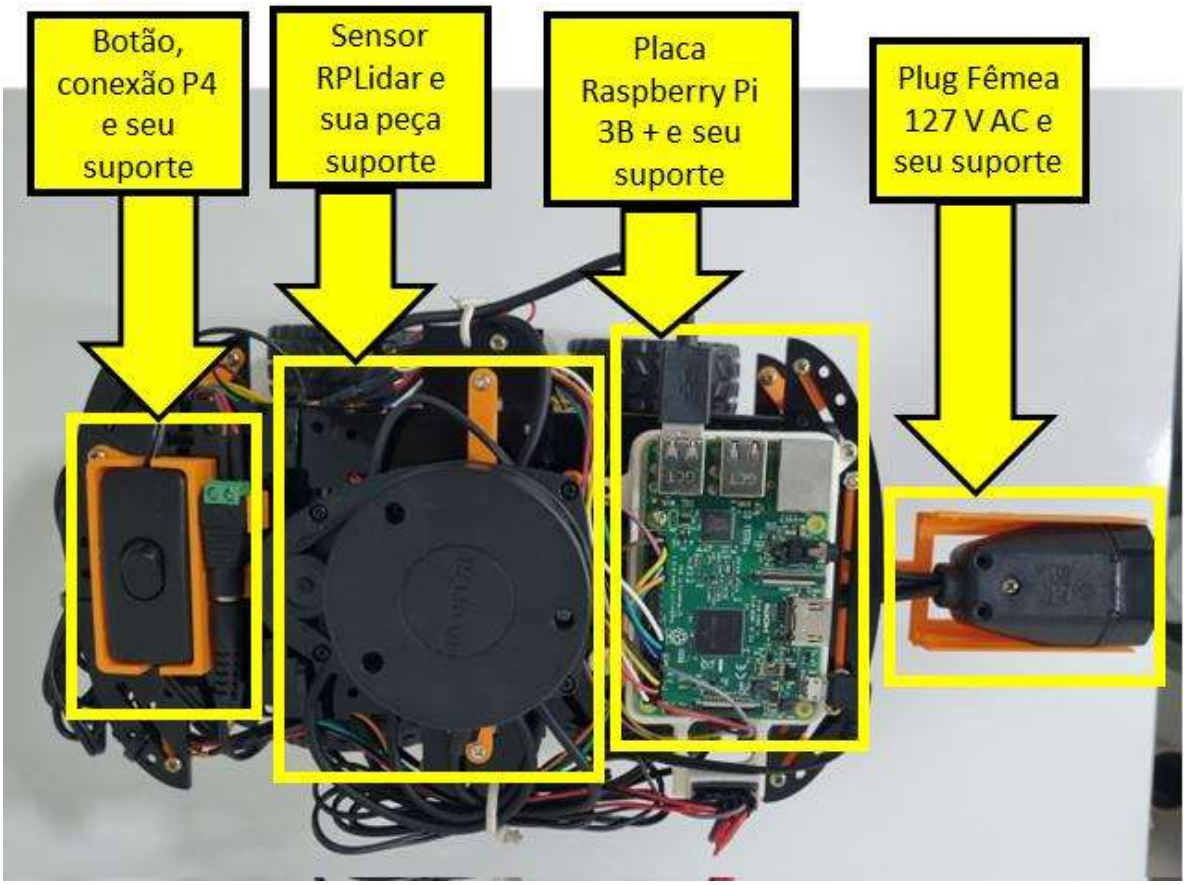
Figura 60 – Detalhe do encaixe das placas Encoder FC-03.



Fonte: Autoria Própria (2023)

Na **Figura 61** está exposta a montagem dos itens designados ao Chassi Superior. A distribuição de peso resultado dos dois chassis concedeu à plataforma um baixo centro de gravidade que por sua vez possibilitou centros instantâneos de rotação próximos o suficiente para que pudessem ser considerados simétricos e, portanto, possibilitando testes para a modelagem cinemática simétrica.

Figura 61 – Montagem final, com afiação, dos elementos do Chassi Superior.



Fonte: Autoria Própria (2023)

A validação da Teleoperação e da geração de Odometria do robô pode ser feita ao realizar o seguinte teste: Comandos de movimentação são enviados, locomovendo o robô para a frente por 1s, então, um giro no sentido horário é realizado por 1 s e, por fim, uma movimentação para trás por 1 s, para que a verdadeira locomoção do robô e a sua orientação no eixo z seja medida e comparada com a *Pose* registrada pelo Nó "*/odometry*" da rede ROS. Os resultados encontrados nos testes são listados e comparados com a estimação realizada pelo próprio robô na **Tabela 11**.

Embora a plataforma robótica apresente boa dirigibilidade para um usuário, contanto que esse consiga ter visão da movimentação do robô, e entregue rápidas respostas aos comandos de teleoperação, o recurso de previsão da *Pose* do robô de forma automatizada, não consegue ser preciso e os erros acumulam com ao passo que os comandos são executados.

A diferença entre os valores medidos e os valores calculados pelo *script* de odometria é grande devido à derrapagem que acontece nos pneus durante as manobras. Isso acontece pela

influência dos diferentes níveis de atrito que o piso, onde o desenvolvimento do protótipo e os testes foram realizados, oferece. Outro fenômeno que pode estar associado à diferença dos valores de Pose é a leitura dos pulsos computados. Isso pode se dar por erros de leitura nas placas Encoder FC-03 ou durante o registro dos sinais feito pelo computador embarcado, resultando em um número de pulsos divergente com a real rotação das rodas.

Tabela 11 – Teste de validação da teleoperação e geração de odometria.

Etapa	Ação executada pelo robô	Tempo de execução (s)	*Pose* estimada pelo Nó "*/odometry*" em x, y e θ	Medidas realizadas no laboratório
1	Movimento para frente	1	0,53 m	0,94 m
2	Giro no sentido horário	2	67 º	308º
3	Movimento para trás	1	0,65 m	0,37 m

Fonte: Autoria Própria (2023)

Os motores da plataforma robótica possibilitam poucas formas de controle fino das rodas, tirando o controle de PWM realizado, o que leva a uma das rodas girar de forma dissonante das demais, causando erros de leitura por parte do algoritmo, o que impacta grandemente no resultado. As derrapagens acontecem de forma aleatórias e atingem as quatro rodas de forma que não foi possível mapear quais rodas estavam mais propensas, no escopo de tempo deste trabalho. A pressão dos pneus é um dos itens que podem ser modificados a fim de melhor controlar a derrapagem das rodas, segundo Mandow et al. (2007, p. 1224), no entanto esse recurso não está presente nas rodas do Kit Chassi 4WD. Levando em consideração projetos robóticos que conseguiram implementar, com sucesso, recursos de geração de odometria via *Dead-Reckoning*, como os AGVs e AMRs desenvolvidos no Polo de Inovação Manaus / IFAM e no caso do trabalho de Boas e Conceição (2020, p. 5), um dos fatores decisivos está no projeto mecânico que deve levar em consideração a relação rpm-toque na escolha dos motores usados nos robôs móveis, assim como a construção de um sistema de amortecimento, dessa forma, mesmo em pisos lisos, o dimensionamento do sistema como um todo possibilita medidas que mitigam os erros de leitura

Embora, para um robô móvel, seja importante a geração de odometria devido a seu uso em atividades como a localização simultânea em ambiente previamente mapeados ou na exploração independente de bases computacionais, para o objetivo principal deste trabalho, mapeamento de ambientes internos usando algoritmo "*hector_mapping*", a odometria não interfere no resultado final, sendo ela desenvolvida neste trabalho como a tentativa de tornar o robô uma plataforma robótica mais completa e pronta para receber outras funções como as citadas.

A **Tabela 12** apresenta os valores de cada peça mecânica projetada e fabricada com impressora 3D (valores referentes a quantidade de material PLA usado), cada *hardware* comprado e o valor final de todos os itens empregados na construção do robô móvel.

Tabela 12 – Lista de peça usadas no robô móvel e seus valores.

Item	Qtd	Preço unitário	Valor
Peças mecânicas fabricadas na impressora 3D.	91,5 g	R$ 0,1/g	R$ 9,15
Raspberry Pi 3B+	1 un	R$ 659	R$ 659
Sensor *RPLidar A1*	1 un	R$ 989	R$ 989
Sensor Encoder FC-03	4 un	R$ 12,5	R$ 50
Driver L298N	2 un	R$ 24,3	R$ 48,6
Fonte de potência 12 V e 1,5 A	1 un	R$ 20	R$ 20
Fonte de potência 5 V e 3 A	1 un	R$ 77	R$ 77
		Total	R$ 1.852,75

Fonte: Autoria Própria (2023)

4.2 MAPAS DOS AMBIENTES INTERNOS

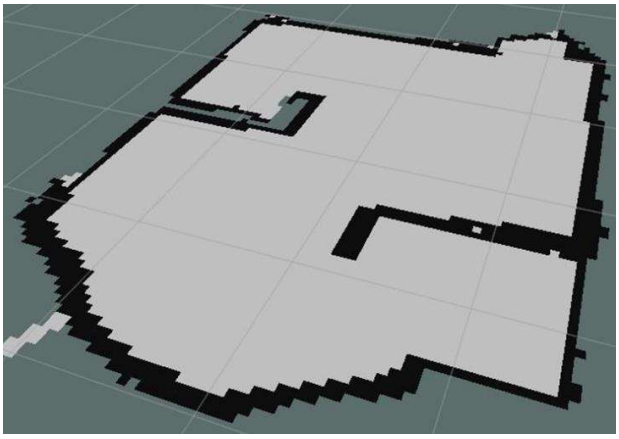
A função de gerar mapas de ambientes internos a partir da sua exploração foi testada em dois ambientes diferentes, um dos laboratórios do Polo de Inovação Manaus / IFAM e um ambiente residencial. Construídas com caixas e outros objetos, além do próprio layout de cada sala, esses ambientes forçam o robô a executar manobras, passando e repassando por pontos em comum durante a exploração, assim como mudando diversas vezes sua orientação no eixo z. A integração do sistema robótico: teleoperação, manobrabilidade do robô móvel e a aquisição, transporte e uso dos dados do sensor *LiDAR* embarcado são testados, além de apresentar desafios para a capacidade do algoritmo de construir um mapa coeso mesmo com grandes variações de movimento, limitados

sempre ao plano 2D. Na **Figura 62** e **Figura 63**, é mostrado o robô móvel no meio do processo de mapeamento do ambiente laboratório e o resultado do mapa salvo no sistema, respectivamente.

Figura 62 – Ambiente laboratório sendo mapeado pelo robô móvel.

Fonte: Autoria Própria (2023)

Figura 63 – Resultado do mapeamento do ambiente laboratório.



Fonte: Autoria Própria (2023)

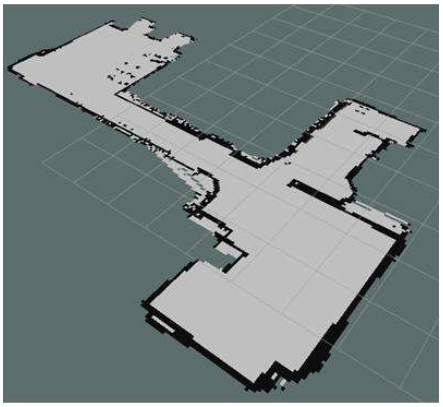
O ambiente residencial, muito maior que o ambiente anterior, apresenta mais um desafio ao algoritmo ao proporcionar trepidações ao longo do eixo z, devido à inconstância do terreno composto de azulejos que não estão perfeitamente alinhados. Esses ruídos nos dados de leitura combinam-se com a grande área a ser mapeada e adicionam mais um nível de dificuldade para se alcançar um mapa claro e coeso.

Na **Figura 64** e **Figura 65**, é mostrado um dos cômodos do ambiente residencial mapeado e o resultado do mapa salvo no sistema, respectivamente. Esse exemplo de aplicação, embora mais ruidoso, apresenta uma cópia fiel do espaço explorado pelo robô e caracteriza a plataforma robótica como apta a ser teleoperada por corredores e distâncias maiores que 10 metros sem que houvesse queda de conexão ou uma queda significativa na qualidade do mapeamento.

Figura 64 – Um dos cômodos do ambiente residencial.

Fonte: Autoria Própria (2023)

Figura 65 – Resultado do mapeamento.



Fonte: Autoria Própria (2023)

5 CONCLUSÃO

Neste capítulo são apresentadas as conclusões referentes ao trabalho realizado, uma breve análise dos objetivos definidos e propostas de melhorias para o sistema robótico desenvolvidos.

5.1 ANÁLISE DOS OBJETIVOS DO TRABALHO

- O robô móvel foi desenvolvido conforme proposto, caracterizado por ser do tipo *Skid-Steer*, testes foram realizados para alcançar fórmulas de geração de odometria pela modelagem cinemática simétrica, embora a previsão de deslocamento apresenta diferenças nos resultados reais devido ao deslizamento das rodas. O projeto mecânico foi desenvolvido previamente na plataforma *Onshape*, com peças projetadas para organizar e distribuir os itens provenientes do projeto eletrônico e, uma vez que essas etapas estavam certificadas e aprovadas, iniciou-se o processo de fabricação e montagem das peças mecânicas.
- O *Firmware* do sistema robótico foi desenvolvido usando o sistema ROS, dividido em duas centrais computacionais, uma embarcada no robô móvel e outra estática servindo de interface ao usuário, a rede ROS construída as unifica em um sistema coeso de transporte remoto de comandos e dados, concedendo ao robô a capacidade de ser teleoperado.
- Os mapas gerados a partir da leitura do sensor *LiDAR* embarcado apresentam fidelidade com os ambientes físicos e o processo de exploração se mostrou de fácil execução devido a boa manobrabilidade da plataforma robótica assim como a rápida resposta dos comandos.
- Os resultados foram satisfatórios, visto que o sistema robótico entrega recursos de teleoperação para missões de exploração e mapeamento de ambientes internos, além de disponibilizar as velocidades individuais de cada roda, através da leitura dos encoders ópticos.

5.2 CONCLUSÕES FINAIS

Ao final da execução deste trabalho, conclui-se que os objetivos estabelecidos foram alcançados e representam o desenvolvimento do conhecimento robótico alinhado aos projetos de pesquisa da lei de PD&I no ambiente acadêmico do IFAM Distrito Industrial, de modo que contribuiu para o aprendizado do autor e possibilita uma porta de entrada para projetos usando o sistema ROS.

O sistema robótico desenvolvido neste trabalho apresenta compatibilidade com diversas soluções e algoritmos que usam do sistema ROS como meio de implementação, podendo ser aplicado em vários ambientes e missões característicos da robótica móvel, contanto que novos sensores e novos subsistemas de *Firmware* sejam acrescentados, assim como soluções mecânicas que concedam ao robô maior robustez e flexibilidade de atuação quanto ao terreno, ou seja, é uma plataforma robótica que permite um grande leque de configurações e possibilidades de aplicação.

5.2.1 Trabalhos Futuros

- Incluir o dimensionamento das rodas no projeto mecânico para que permitam melhor aderência em superfícies lisas.
- Incluir o projeto de amortecedores individuais para cada uma das rodas com o intuito de compensar variações de cargas durante as manobras e conceder maior estabilidade à plataforma robótica.
- Implementação de algoritmos de visão computacional ao final do mapeamento a fim de eliminar ruídos e criar mapas com bordas mais definidas;
- Desenvolver uma interface mais amigável para o usuário, que uma todas as funcionalidades do sistema, possua ícones de interação, botões de comandos de exploração e uma janela de visualização do mapa sendo gerado em tempo real.

REFERÊNCIAS

HUANG, H; SAVKIN, A. V.; NI, W. Online UAV Trajectory Planning for Covert Video Surveillance of Mobile Targets. *IEEE Transactions on Automation Science and Engineering*, p. 1-12, 2021. Doi: 10.1109/TASE.2021.3062810. Citado na página: 12.

DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, p. 99-110, 2006. Doi: 10.1109/MRA.2006.1638022. Citado na página: 12.

NAGLA, S. 2D Hector SLAM of Indoor Mobile Robot Using 2D LiDAR. *2ND INTERNATIONAL CONFERENCE ON POWER, ENERGY, CONTROL AND TRANSMISSION SYSTEMS (ICPECTS)*, 2020, Chennai India, P. 1-4. Doi: 10.1109/ICPECTS49113.2020.9336995. Citado na página: 12.

SIEGWART, R; NOURBAKHSH, I. R. *Introduction to Autonomous Mobile Robots*. Cambridge Massachusetts, Estados Unidos da América: MIT Press, 2004. Citado na página: 13.

ABIRESEARCH. SLAM Software Paving the Path for a New Generation of Industrial Robots - AMRs. 2019. Disponível em:<https://www.abiresearch.com/press/slam-software-paving-path-new-generation-industrial-robots-amrs/>. Acessado em: 27 de outubro de 2023. Citado na página: 14.

CORKE, P. *Robotics, Vision and Control*. Cham, Suíça, Editora Springer, segunda edição, 2017. Citado duas vezes na página 16: Texto e Imagem.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). ISO 8373: Robots and robotic devices. Disponível em: <https://www.iso.org/obp/ui/en/#iso:std:iso:8373:ed-3:v1:en>. Acessado em: 05 setembro 2023. Citado na página 16.

INTERNATIONAL FEDERATION OF ROBOTICS (IFR). Robotics Standardization: Mobile Robot. Disponível em: <https://ifr.org/standardisation>. Acessado em: 03 setembro 2023. Citado na página: 17.

COTA, Eduardo. **Implementação e avaliação de técnicas de odometria aplicadas a um dispositivo móvel**. Dissertação (Mestrado) - Universidade Federal de Ouro Preto. Escola de Minas. Departamento de Engenharia e Automação e Técnicas Fundamentais. Ouro Preto, Minas Gerais, 2019. Citado nas páginas 17 e 44.

LAGES, Walter Fetter. **Estimação de Posição e Orientação**. Artigo (Pós-Graduação) – Universidade Federal do Rio Grande do Sul, Escola de Engenharia, 2008. Disponível em: < https://www.ece.ufrgs.br/~fetter/ele00070/mobrob/estimation.pdf>. Acessado em: 08 de março de 2023. Citado na página: 17.

KOLMANOVSKY, I.; MCCLAMROCH, N. H. Developments in Nonholonomic Control Problems. *IEEE Control Systems Magazine*, v. 15, n. 6, p. 20–36, 1995. Doi: 10.1109/37.476384. Citado na página: 18.

SALEM, F. A. Dynamic and Kinematic Models and Control for Differential Drive Mobile Robots. *International Press Corporation (INPRESSCO) – International Journal of Current Engineering and Technology*, v. 3, n. 2, Arabia Saudita, Taif, p. 253-263, 2013. Disponível em: < https://inpressco.com/wp-content/uploads/2013/03/Paper6253-2632.pdf >. Acessado em: 25 de maio de 2023. Citado na página: 18.

RODRIGUES, R. M. **Construção, Modelagem e Controle de Um Robô Móvel de Acionamento Diferencial**. Monografia (Graduação) – Universidade Tecnológica Federal do Paraná, Toledo, Paraná, 2019. Citado na página: 19.

YI, J.; SONG, D.; ZHANG, J.; GOODWIN, Z. Adaptive Trajectory Tracking Control of *Skid-Steer*ed Mobile Robots. In: *PROCEEDINGS 2007 IEEE INTERNATIONAL CONFERENCE ON*

ROBOTICS AND AUTOMATION (ICRA), 2007, Roma, Itália, p. 2605-2610. Doi: 10.1109/ROBOT.2007.363858. Citado na página: 20.

PENTZER, J.; BRENNAN, S.; REICHARD, K. Model-Based Prediction of *Skid-Steer* Robot Kinematics Using Online Estimation of Track Instantaneous Centers of Rotation. *Jornal of Field Robotics*, v. 31, n. 3, p. 455-476, 2014. Doi:10.1002/rob.21509. Citado na página: 21.

WU, Y.; WANG, T.; LIANG, J.; CHEN, J.; ZHAO, Q.; YANG, X.; HAN, C. Experimental Kinematics Modeling Estimation for Wheeled *Skid-Steer*ing Mobile Robots. In: *PROCEEDING 2013: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND BIOMIMETICS (ROBIO)*, 2013, Shenzen, China, p. 268-273. Doi: 10.1109/ROBIO.2013.6739470. Citado na página: 21.

MARTINEZ, J. L; MANDOW, A; MORALES, J; PEDRAZA, S; GARCIA, A. Approximating Kinematics for Tracked Mobile Robots. *The International Journal of Robotics Research*. v. 24, n. 10, p. 867-878, 2005. Doi: 10.1177/0278364905058239. Citado na página: 21.

MANDOW, A; MARTINEZ, J.L.; MORALES, J.; BLANCO, J.L.; GARCIA-CEREZO, A.; GONZALES, J. Experimental kinematics for wheeled *Skid-Steer* mobile robots, In: *PROCEEDINGS 2007: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS*, 2007, San Diego, California, Estados Unidos, p. 1222-1227. Doi: 10.1109/IROS.2007.4399139. Citado na página: 22 e 87.

WANG, T.; WU Y.; LIANG, J.; HAN, C.; CHEN, J.; ZHAO, Q. Analysis and Experimental Kinematics of a *Skid-Steer*ing Wheeled Robot Based on a Laser Scanner Sensor. *Sensors*, v.15, p. 9681-9702, 2015. Doi: 10.3390/s150509681. Citado na página: 22.

WANG, C; WENJUN, L.; XIAOCHUAN, L.; MINGLIANG, M. Terrain Adaptive Estimation of Instantaneous Centres of Rotation for Tracked Robots. *Complexity*, vol. 2018, p. 1-10, 2018. Doi: 10.1155/2018/4816712. Citado na página: 22.

BOAS, L; CONCEIÇÃO, A. Modelagem Cinemática de Robôs Móveis da Classe *Skid-Steer*. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 23. 2020. **Anais**: Sociedade Brasileira de Automática, v. 2, n. 1. Doi: 10.48011/asba.v2i1.1165. Citado na página: 22 e 87.

CLEARPATH ROBOTICS. Husky: Unmanned Ground Vehicle. Disponível em: <https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/>. Acessado em: 10 de agosto de 2023. Citado na página: 23.

ROS - ROBOT OPERATING SYSTEM. What is ROS. Disponível em: <https://www.ros.org/>. Acessado em: 27 de julho de 2023. Citado na página: 25.

PYO, Y; CHO, H; JUNG, R; LIM, T. *ROS ROBOT PROGRAMMING – From the basic concept to the practical programming and robot application*. Seul, Coreia do Sul, Editora ROBOTIS, Primeira Edição, 2017. Citado duas vezes na página 25: Imagem e Texto.

WYROBEK, K. The Origin Story of Ros, The Linux of Robotics. *IEEE SPECTRUM – For the Technology Insider*, 2017. Disponível em: <https://spectrum.ieee.org/the-origin-story-of-ros-the-linux-of-robotics>. Acessado em: 01 de fevereiro de 2023. Citado duas vezes na página 26: Texto e Imagem.

ACKERMAN, E; GUIZZO, E. Wizards of Ros: Willow Garage and The Making of The Robot Operating System. *IEEE SPECTRUM – For the Technology Insider*, 2017. Disponível em: <https://spectrum.ieee.org/wizards-of-ros-willow-garage-and-the-making-of-the-robot-operating-system>. Acessado em: 01 de fevereiro de 2023. Citado na página: 26.

OLIVEIRA, R. **Construção de Sistema Robótico de Baixo Custo para Fins Didáticos**. Monografia (Graduação) – Instituto Federal de Educação Ciência e Tecnologia do Amazonas, Manaus, Amazonas, 2019. Citado na página: 27.

QUIGLEY, M; GERKEY, B.; SMART W. D. *Programming Robots with ROS – A Practical Introduction to the Robot Operating System*. Estados Unidos da América, OREILLY, Primeira Edição, 2015. Citado duas vezes na página 28: Texto e Imagem, e uma vez na página 38.

SEARS-COLLINS, A. How to Create and Execute ROS Launch Files. *Automatic Addison*, 2019. Disponível em: <https://automaticaddison.com/how-to-create-and-execute-ros-launch-files/>. Acessado em: 7 de junho de 2023. Citado na página: 29.

FOOTE, T. tf2: Package Summary (Noetic). *ROS Documentation*, 2019. Disponível em: <https://wiki.ros.org/tf2>. Acessado em: 10 de março de 2023. Citado na página: 29.

SILLIMAN, M. Multiple TurtleBots in Concert. *Learn TurtleBot and ROS*, 2015. Disponível em: <https://learn.turtlebot.com/2015/02/04/6/>. Acessado em: 10 de março de 2023. Citado na página: 30.

FOOTE, T. tf: The Transform Library. In: *2013 IEEE CONFERENCE ON TECHNOLOGY FOR PRATICAL ROBOT APPLICATIONS (TePRA)*, 2013, Open Source Robotics Foundation, Mountain View, Califórnia, Estados Unidos da América. Doi: 10.1109/TePRA.2013.6556373. Citado duas vezes na página 30: Imagem e Texto.

JONSSON, M. **SLAM in Low Speed Scenarios Using Ultrasonic Sensors**. Dissertação (Mestrado) – *Lund University. Department of Computer Science*. Lund, Suécia. 2020. Disponível em: <https://www.lunduniversity.lu.se/lup/publication/9024868>. Acessado em: 12 de março de 2023. Citado na página: 30.

OBP (ONLINE BROWSING PLATFORM). ISO 8373: Robotics Vocabulary. 2021. Disponível em: <https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-3:v1:en>. Acessado em: 08 de outubro de 2023. Citado na página: 31.

CAMARA, J. **Map Slammer – Densifying Scattered KSLAM 3D Maps with Estimated Depth**. Monografia (Graduação) – *Universidad de Alicante, Escola Politécnica Superior*, Alicante Espanha, 2019. Disponível em: <http://hdl.handle.net/10045/94751>. Acessado em: 23 de julho de 2023. Citado na página: 31.

SAAT, S; RASHID, W; TUMARI, M; SAEALAI, M. HectorSLAM 2D Mapping for Simultaneous Localization and Mapping (SLAM). In: *JOURNAL OF PHYSICS CONFERENCE SERIES*, vol. 1529, n. 4, 2020, Melaka, Malásia. Doi: 10.1088/1742-6596/1529/4/042032. Citado nas páginas: 32 e 46.

ALMEIDA M. hector_mapping: Package Summary (Noetic). *ROS Documentation*, 2021. Disponível em: < https://wiki.ros.org/hector_mapping>. Acessado em: 27 de julho de 2023. Citado na página: 32.

BUTTERLY, P; DALY, J; MORRISH, L. **Implementing odometry and SLAM Algorithms on a Raspberry Pi to Drive a Rover**. Monografia (Graduação) – *Institute of Technology Blanchardstown*, Dublin, Irlanda, 2014. Doi: 10.13140/RG.2.1.1747.8569. Citado na página: 33.

SLAMTEC GLOBAL NETWORK. RPLidar A1 – Entry-Level LiDAR, 2022. Disponível em: <https://www.slamtec.com/en/LiDAR/A1/>. Acessado em: 20 de fevereiro de 2023. Citado nas páginas: 33 e 46.

WALLHOFF, F; RUB, M; RIGOLL, G; GOBEL, J; DIEHL, H. Surveillance and Activity Recognition with Depth Information. In: *IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO (ICME)*, 2007, Pequim, China, p. 1103-1106. Doi: 10.1109/ICME.2007.4284847. Citado na página: 34.

NEGI, A. Raspberry Pi 3 B+ Pinout with GPIO functions, Schematic and Specs in detail. *eTechnophiles,* 2020. Disponível em: <https://www.etechnophiles.com/raspberry-pi-3-b-pinout-

with-gpio-functions-schematic-and-specs-in-detail/>. Acessado em: 05 de abril de 2023. Citado na página: 39.

POLOLU – ROBOTICS & ELETRONICS. Raspberry Pi 3 Model B+. Disponível em:<https://www.pololu.com/product/2797>. Acessado em: 05 de abril de 2023. Citado na página: 39.

ADAFRUIT. DC Gearbox Motor – "TT Motor" – 200RPM – 3 to 6VDC. Disponível em: <https://www.adafruit.com/product/3777>. Acessado em: 11 de maio de 2023. Citado nas páginas: 40 e 41.

SINGH, A. V.; AGRAWAL, Y.; GUPTA, R; KUMAR, A.; BOHARA, V. A. ANROL: Autonomous Navigation based on ROS and Laser Odometry. In: *15TH INTERNATIONAL CONFERENCE ON COMMUNICATION SYSTEMS & NETWORKS (COMSNETS)*, 2023, Bangalore, India. p. 195-197. Doi: 10.1109/COMSNETS56262.2023.10041273. Citado na página: 40.

SMART PROJECTS BRASIL. Kit Chassi Carro 4WD 4 Rodas Robótica Arduino. Disponível em: <https://www.smartprojectsbrasil.com.br/kit-chassi-carro-4wd-4-rodas-robotica-arduino>. Acessado em: 01 de fevereiro de 2023. Citado na página: 41.

CARDOSO, D. Driver Motor com Ponte H L298N – Controle de Motor DC com Arduino. Vida de Silício, 2017. Disponível em: <https://portal.vidadesilicio.com.br/driver-motor-com-ponte-h-l298n/>. Acessado em: 11 de maio de 2023. Citado na página: 41.

LAST MINUTE ENGINEER. Interface L298N DC Motor Driver Module with Arduino. Disponível em: <https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/>. Acessado em: 12 de maio de 2023. Citado na página: 42.

ALMEIDA, F. O que é Encoder? Para que serve? Como Escolher? Como Interfacear?. HI Tecnologia: Automação Industrial, 2017. Disponível em:

<https://materiais.hitecnologia.com.br/blog/o-que-%C3%A9-encoder-para-que-serve-como-escolher-como-interfacear/>. Acessado em: 17 de março de 2023. Citado nas páginas: 44 e 45.

NIKU, S. *Introduction to Robotics: Analysis, Control, Applications*. Estados Unidos da América, segunda edição, Editora: *Pearson Education*, 2011. Citado na página: 45.

JIN, X. **Implementing Hector SLAM and AMCL Algorithm for Unmanned Ground Vehicle (UGV) in GPS Denied and Dynamic Environment**. Dissertação (Mestrado) – *California State Polytechnic University* (Cal Poly), Pomona, Califórnia, 2021. Disponível em: < http://hdl.handle.net/20.500.12680/zk51vp17d> Acessado em: 25 de agosto de 2023. Citado na página: 45.

SMART PROJECTS BRASIL. Fonte Chaveada Carregador Plug P4 - 12V 1,5A. Disponível em: <https://www.smartprojectsbrasil.com.br/fonte-chaveada-carregador-plug-p4-12v-1a>. Acessado em: 01 de fevereiro de 2023. Citado na página: 54.

SMART PROJECTS BRASIL. Fonte Vertical para Raspberry Pi 3 - 5V 3A com Interruptor. Disponível em: < https://www.smartprojectsbrasil.com.br/fonte-vertical-para-raspberry-pi-3-5v-3a-com-interruptor>. Acessado em: 01 de fevereiro de 2023. Citado na página: 54.

LEARN ONSHAPE. Learning Pathways. Disponível em: <https://learn.onshape.com/>. Acessado em: 27 de agosto de 2023. Citado na página: 58.

YOUNG, E. Configuring Ubuntu 20.04 to use a Static IP Address. *PiMyLifeUp*, 2022. Disponível em: <https://pimylifeup.com/ubuntu-20-04-static-ip-address/>. Acessado em: 08 de fevereiro de 2023. Citado na página: 67.

SAITO, I. Ubuntu - Install of ROS Noetic. *ROS Documentation*, 2023. Disponível em: <https://wiki.ros.org/noetic/Installation/Ubuntu>. Acessado em: 04 de janeiro de 2023. Citado na página: 68.

SOUZA, F. Como usar a GPIO da Raspberry Pi usando Python. *Medium, 2020*. Disponível em: <https://medium.com/vacatronics/este-%C3%A9-um-tutorial-sobre-como-usar-a-gpio-da-raspberry-pi-usando-a-biblioteca-python-e3b5bd5c890c>. Acessado em: 20 de junho de 2023. Citado na página: 69.

PURVIS M. Teleop_Twist_Keyboard Package: Installing, Running and Controls. *ROS Documentation*, 2015. Disponível em: <https://wiki.ros.org/teleop_twist_keyboard#Running>. Acessado em: 03 de abril de 2023. Citado na página: 72.

NIEWINSKI, D. Troubleshooting ROS Network Issues. CLEARPATH ROBOTICS – ROBOT TUTORIAL & GUIDES, 2020. Disponível em: <https://support.clearpathrobotics.com/hc/en-us>. Acessado em: 21 de abril de 2023. Citado na página: 73.

VILLENA A. Teleop with Teleop_Twist_Keyboard. *ROS Documentation*, 2017. Disponível em: <http://wiki.ros.org/stdr_simulator/Tutorials/Teleop%20with%20teleop_twist_keyboard>. Acessado em: 19 de abril de 2023. Citado na página: 75.

PUCHER F. Welcome to DiffBot Documentation. *DiffBot Differential Drive Mobile Robot*, 2021. Disponível em: <https://ros-mobile-robots.com/>. Acessado em: 11 de agosto de 2023. Citado na página: 76 e duas vezes na página 77: Imagem 1 e Imagem 2.

DESAI C. How to build RPLidar ros package. *Github*, 2022. Disponível em: <https://github.com/robopeak/RPLidar_ros/wiki>. Acessado em: 29 de abril de 2023. Citado na página: 79.

ANEXOS

ANEXO A

Anexo A.1 - Estrutura das Mensagens ROS usadas neste trabalho.

Mensagem ROS	Estrutura Mãe	Definição Compactada
/LaserScan	sensor_msgs	std_msgs/Header header
		float32 angle_min
		float32 angle_max
		float32 angle_increment
		float32 time_increment
		float32 scan_time
		float32 range_min
		float32 range_max
		float32[] ranges
		float32[] intensities
/Header.msg	std_msg	uint32 seq
		time stamp
		string frame_id
/Twist	geometry_msgs	geometry_msgs/Vector3 linear
		geometry_msgs/Vector3 angular
/Vector3	geometry_msgs	float64 x
		float64 y
		float64 z
/String	std_msg	string data
/Float32	std_msg	float32 data
/Float32MultiArray	std_msg	std_msgs/MultiArrayLayout layout
		float32[] data
/MultiArrayLayout	std_msg	std_msgs/MultiArrayDimension[] dim
		uint32 data_offset
/MultiArrayDimension	std_msg	string label
		uint32 size
		uint32 stride

Anexo A.2 - Lista de arquivos Launch usados no projeto e as configurações dos parâmetros e seus valores.

Arquivo	Configuração dos Parâmetros
hector_mapping_diff.launch	Mesma programação do hector_mapping.launch. Principal modificação: <arg name="scan_topic" default="diffbot/scan"/>
RPLidar.launch	<launch> <node name="RPLidarNode" pkg="RPLidar_ros" type="RPLidarNode" output="screen"> <param name="serial_port" type="string" value="/dev/ttyUSB0"/> <param name="serial_baudrate" type="int" value="115200"/><!--A1/A2 --> <!--param name="serial_baudrate" type="int" value="256000"--><!--A3 --> <param name="frame_id" type="string" value="RPLidar_laser_link"/> <param name="inverted" type="bool" value="false"/> <param name="angle_compensate" type="bool" value="true"/> </node> </launch>
hector_mapping.launch	<launch> <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/> <arg name="base_frame" default="base_link"/> <arg name="odom_frame" default="base_link"/> <!-- try to introduce the map_frame as the base and odom above --> <arg name="map_frame" default="map"/> <!-- It Worked! --> <arg name="pub_map_odom_transform" default="true"/> <arg name="scan_subscriber_queue_size" default="5"/> <!-- Here is where the original code refer to "/diffbot/scan" --> <arg name="scan_topic" default="/scan"/> <arg name="map_size" default="2048"/> <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="screen"> <!-- Frame names --> <param name="map_frame" value="map"/> <param name="base_frame" value="$(arg base_frame)"/> <param name="odom_frame" value="$(arg odom_frame)"/> <!-- Tf use --> <param name="use_tf_scan_transformation" value="false"/> <param name="use_tf_*Pose*_start_estimate" value="false"/> <param name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)"/> <!-- Map size / start point --> <param name="map_resolution" value="0.050"/> <param name="map_size" value="$(arg map_size)"/> <param name="map_start_x" value="0.5"/> <param name="map_start_y" value="0.5"/> <param name="map_multi_res_levels" value="2"/> <!-- Map update parameters --> <param name="update_factor_free" value="0.4"/> <param name="update_factor_occupied" value="0.9"/> <param name="map_update_distance_thresh" value="0.4"/> <param name="map_update_angle_thresh" value="0.06"/> <param name="laser_z_min_value" value="-1.0"/>

```
<param name="laser_z_max_value" value="1.0"/>
<!--  Advertising config  -->
<param name="advertise_map_service" value="true"/>
<param name="scan_subscriber_queue_size" value="$(arg
scan_subscriber_queue_size)"/>
<param name="scan_topic" value="$(arg scan_topic)"/>

<param name="tf_map_scanmatch_transform_frame_name" value="$(arg
tf_map_scanmatch_transform_frame_name)"/>
</node>
</launch>
```

robot.launch

```
<launch>

  <node name="motor_control" pkg="ciano_agv" type="teleop2command.py" />

  <node name="delta_t" pkg="ciano_agv" type="delta_t.py" />

  <node name="speeds" pkg="ciano_agv" type="speed.py" />

  <node name="odometry" pkg="ciano_agv" type="odometry.py" />

  <include file="$(find RPLidar_ros)/launch/RPLidar.launch">
  </include>

</launch>
```

base.launch

```
<node name="teleop_twist_keyboard" pkg="“teleop_twist_keyboard”
type="teleop_twist_keyboard.py" />

<include file="$(find agv_bringup)/launch/hector_mapping.launch">
</include>

<node pkg="tf2_ros" type="static_transform_publisher"
name="base_link_to_laser" args="0 0 0 0 0 0 base_link RPLidar_laser_link" />

<node pkg="rviz" type="rviz" name="rviz" />

<node name="rqt_graph" pkg="rqt_graph" type="rqt_graph" />

<node name="rqt_tf_tree" pkg="rqt_tf_tree" type="rqt_tf_tree" />
```

Anexo A.3 - Programação, em Python, dos Scripts usado no Projeto *Firmware*.

Etapa **Figura 42 –** Rede ROS durante controle dos quatro motores e leitura dos quatro Encoders.

Script /motor_control

Programação

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

import rospy
import std_msgs.msg
from std_msgs.msg import String
import geometry_msgs.msg
from geometry_msgs.msg import Twist
import RPi.GPIO as PINS
```

```
left_A = 17
left_in1 = 27
left_in2 = 22

right_A = 2
right_in1 = 3
right_in2 = 4

def set_speed(value):
	en_left.ChangeDutyCycle(value)
	en_right.ChangeDutyCycle(value)


def forward(speed_value):
	PINS.output(left_in1, PINS.LOW)
	PINS.output(left_in2, PINS.HIGH)
	PINS.output(right_in1, PINS.LOW)
	PINS.output(right_in2, PINS.HIGH)

	set_speed(speed_value)

def backward(speed_value):
	PINS.output(left_in1, PINS.HIGH)
	PINS.output(left_in2, PINS.LOW)
	PINS.output(right_in1, PINS.HIGH)
	PINS.output(right_in2, PINS.LOW)

	set_speed(speed_value)

def left(speed_value):
	PINS.output(left_in1, PINS.LOW)
	PINS.output(left_in2, PINS.HIGH)
	PINS.output(right_in1, PINS.HIGH)
	PINS.output(right_in2, PINS.LOW)

	set_speed(speed_value)

def right(speed_value):
	PINS.output(left_in1, PINS.HIGH)
	PINS.output(left_in2, PINS.LOW)
	PINS.output(right_in1, PINS.LOW)
	PINS.output(right_in2, PINS.HIGH)

	set_speed(speed_value)

def stop_function():
	PINS.output(left_in1, PINS.LOW)
	PINS.output(left_in2, PINS.LOW)
	PINS.output(right_in1, PINS.LOW)
	PINS.output(right_in2, PINS.LOW)

	set_speed(0)

def move_it_cb(msg):
	direction = msg.linear.x
```

```
        speed = msg.linear.z

        if direction == 1:
            print("Moving Foward")
            forward(speed)
        if direction == -1:
            print("Moving Backward")
            backward(speed)
        if direction == 10:
            print("Moving Left")
            left(speed)
        if direction == -10:
            print("Moving Right")
            right(speed)
        if direction == 0:
            print("Stoping It All")
            stop_function()

    if __name__ == '__main__':
        rospy.init_node('motors_control')
        rospy.loginfo("Drivers online, waiting for instructions...")

        PINS.setmode(PINS.BCM)
        PINS.setwarnings(False)

        PINS.setup(left_A, PINS.OUT)
        PINS.setup(left_in1, PINS.OUT)
        PINS.setup(left_in2, PINS.OUT)

        PINS.setup(right_A, PINS.OUT)
        PINS.setup(right_in1, PINS.OUT)
        PINS.setup(right_in2, PINS.OUT)

        en_left = PINS.PWM(left_A, 1000)
        en_right = PINS.PWM(right_A, 1000)

        en_left.start(10)
        en_right.start(10)

        move_it = rospy.Subscriber("/m_commands", Twist, move_it_cb)

        rospy.spin()
```

Rede

Figura 42 – Rede ROS durante controle dos quatro motores e leitura dos quatro Encoders.

Script

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

import rospy
import std_msgs.msg
from std_msgs.msg import Empty
from std_msgs.msg import Int32MultiArray
import RPi.GPIO as PINS


ticks = Int32MultiArray()
```

```
FL = 6
FL_count = 0

FR = 13
FR_count = 0

RL = 19
RL_count = 0

RR = 26
RR_count = 0

def pub_and_show():
	global FL_count
	global FR_count
	global RL_count
	global RR_count
	global ticks

	ticks.data = [FL_count, FR_count, RL_count, RR_count]

	pub.publish(ticks)

	print("the acount is at: ")
	print("FL: ", FL_count)
	print("FR: ", FR_count)
	print("RL: ", RL_count)
	print("RR: ", RR_count)

def FL_detection(FL):
	global FL_count
	FL_count = FL_count + 1
	pub_and_show()

def FR_detection(FR):
	global FR_count
	FR_count = FR_count + 1
	pub_and_show()

def RL_detection(RL):
	global RL_count
	RL_count = RL_count + 1
	pub_and_show()

def RR_detection(RR):
	global RR_count
	RR_count = RR_count + 1
	pub_and_show()

if __name__ == '__main__':
	rospy.init_node('four_enc')
	rospy.loginfo("Vaaaaai")

	pub = rospy.Publisher("four_encoders_pulses", Int32MultiArray, queue_size = 10)

	PINS.setmode(PINS.BCM)
```

```
    PINS.setup(FL, PINS.IN, pull_up_down = PINS.PUD_UP)
    PINS.setup(FR, PINS.IN, pull_up_down = PINS.PUD_UP)
    PINS.setup(RL, PINS.IN, pull_up_down = PINS.PUD_UP)
    PINS.setup(RR, PINS.IN, pull_up_down = PINS.PUD_UP)

    PINS.add_event_detect(FL, PINS.FALLING, FL_detection)
    PINS.add_event_detect(FR, PINS.FALLING, FR_detection)
    PINS.add_event_detect(RL, PINS.FALLING, RL_detection)
    PINS.add_event_detect(RR, PINS.FALLING, RR_detection)

    rospy.spin()
```

Rede **Figura 43 –** Rede ROS nos testes de identificação offline de parâmetros.

Script /speed

Programação

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

import rospy
import std_msgs.msg
from std_msgs.msg import String
from std_msgs.msg import Float32
import RPi.GPIO as PINS

FL = 6
FL_c = 0

FR = 13
FR_c = 0

RL = 19
RL_c = 0

RR = 26
RR_c = 0

period_in_secs = 0.2
delta_t = 60 * (1/period_in_secs)
ticks_per_rev = 20
rpm2rad = 0.10471975512
rad2deg = 57.29578
tire_radius = 0.033
displacement_per_rev = 0.207

v_left = Float32()
v_right = Float32()

ef status_generator(time_trigger_msg):
    global FL_c
    global FR_c
    global RL_c
    global RR_c

    global ticks_per_rev
    global rpm2rad
    global rad2deg
```

```
    global tire_radius

    global v_left
    global V_right


    FL_rpm = ((FL_c / 2) / ticks_per_rev) * delta_t
    FL_ang_speed_rad = FL_rpm * rpm2rad
    FL_ang_speed_deg = FL_ang_speed_rad * rad2deg
    FL_lin_speed = FL_ang_speed_rad * tire_radius

    FR_rpm = (FR_c / ticks_per_rev) * delta_t
    FR_ang_speed_rad = FR_rpm * rpm2rad
    FR_ang_speed_deg = FR_ang_speed_rad * rad2deg
    FR_lin_speed = FR_ang_speed_rad * tire_radius

    RL_rpm = (RL_c / ticks_per_rev) * delta_t
    RL_ang_speed_rad = RL_rpm * rpm2rad
    RL_ang_speed_deg = RL_ang_speed_rad * rad2deg
    RL_lin_speed = RL_ang_speed_rad * tire_radius

    RR_rpm = (RR_c / ticks_per_rev) * delta_t
    RR_ang_speed_rad = RR_rpm * rpm2rad
    RR_ang_speed_deg = RR_ang_speed_rad * rad2deg
    RR_lin_speed = RR_ang_speed_rad * tire_radius

    v_left.data = (FL_lin_speed + RL_lin_speed) / 2
    v_right.data = (FR_lin_speed + RR_lin_speed) / 2

    print("Left Velocity: ", v_left.data)
    print("Right Velocity: ", v_right.data)

    v_l_status.publish(v_left)
    v_r_status.publish(v_right)

    FL_c = 0
    FR_c = 0
    RL_c = 0
    RR_c = 0

if __name__ == '__main__':
    rrospy.init_node('wheels_status')
    rospy.loginfo("Reading encoders and generating wheels status")
    rospy.loginfo("Waiting for delta time input")

    time_trigger = rospy.Subscriber("/time_trigger", String, status_generator)

    v_l_status = rospy.Publisher("/linearV_left", Float32, queue_size = 10)
    v_r_status = rospy.Publisher("/linearV_right", Float32, queue_size = 10)

    PINS.setmode(PINS.BCM)

    PINS.setup(FL, PINS.IN, pull_up_down = PINS.PUD_UP)
    PINS.setup(FR, PINS.IN, pull_up_down = PINS.PUD_UP)
    PINS.setup(RL, PINS.IN, pull_up_down = PINS.PUD_UP)
    PINS.setup(RR, PINS.IN, pull_up_down = PINS.PUD_UP)
```

```
    PINS.add_event_detect(FL, PINS.FALLING, FL_count)
    PINS.add_event_detect(FR, PINS.FALLING, FR_count)
    PINS.add_event_detect(RL, PINS.FALLING, RL_count)
    PINS.add_event_detect(RR, PINS.FALLING, RR_count)

    rospy.spin()
```

Etapa **Figura 43 –** Rede ROS nos testes de identificação offline de parâmetros.
Script /alpha_test
Programação

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

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

commands = Twist()
to_the_left = True

def start_test_cb(msg):
    global commands

    commands.linear.x = 0.5
    commands.linear.z = 30

    m_commands.publish(commands)

def stop_test_cb(msg):
    global commands

    print("Stopping the motors")
    commands.linear.x = 0
    commands.linear.z = 0
    m_commands.publish(commands)

if __name__ == '__main__':
    rospy.init_node('alpha_motor')
    rospy.loginfo("The commands to arduino can be sent now!")

    m_commands = rospy.Publisher("/cmd_vel", Twist, queue_size = 10)

    start_signal_sub = rospy.Subscriber("/start", String, start_test_cb)
    stop_signal_sub = rospy.Subscriber("/stop", String, stop_test_cb)

    rospy.spin()
```

Etapa **Figura 43 –** Rede ROS nos testes de identificação offline de parâmetros.
Script /xcir
Programação

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

```
import rospy
import std_msgs.msg
from std_msgs.msg import String
import geometry_msgs.msg
from geometry_msgs.msg import Twist

commands = Twist()
to_the_left = True


def start_test_cb(msg):
    global commands

    commands.angular.z = -1.0

    m_commands.publish(commands)

def stop_test_cb(msg):
    global commands

    print("Stopping the motors")
    commands.linear.x = 0
    commands.linear.z = 0
    commands.angular.z = 0
    m_commands.publish(commands)

if __name__ == '__main__':
    rospy.init_node('xcir_motor')
    rospy.loginfo("The commands to motors can be sent now!")

    m_commands = rospy.Publisher("/cmd_vel", Twist, queue_size = 10)

    start_signal_sub = rospy.Subscriber("/start", String, start_test_cb)
    stop_signal_sub = rospy.Subscriber("/stop", String, stop_test_cb)

    rospy.spin()
```

Etapa	**Figura 44 –** Rede ROS em funcionamento após as modificações para a teleoperação.
Script	/motor_control
Programação	Mesma programação do script anterior de mesmo nome. + move_it = rospy.Subscriber(“cmd_vel”, Twist, move_it_cb)

Etapa	**Figura 44 –** Rede ROS em funcionamento após as modificações para a teleoperação.
Script	/delta_t
Programação	Mesma programação do scipt anterior de mesmo nome. + Retirar a subscrição dos Tópicos Start e Stop

Etapa	**Figura 44 –** Rede ROS em funcionamento após as modificações para a teleoperação.
Script	/odometry
Programação	#!/usr/bin/env python3

```
import rospy
import math
import std_msgs.msg
from std_msgs.msg import String
from std_msgs.msg import Float32
from std_msgs.msg import Float32MultiArray

alpha = 0.476721
xcir = 0.01625
delta_t = 0.2

x_ant = 0
y_ant = 0
theta_ant = 0

left_speed = 0
left_check = False
right_speed = 0
right_check = False


new_position = Float32MultiArray()
new_position.data = [0, 0, 0]

def l_speed_cb(msg):
        global left_speed
        global left_check

        left_speed = msg.data
        left_check = True

def r_speed_cb(msg):
        global right_speed
        global left_check

        right_speed = msg.data
        left_check = True

def check_cb(msg):
        global left_check
        global right_check

        while not left_check or not right_check:
                rospy.loginfo("Data not ready")
        generate_n_pub()
        rospy.loginfo("Pose generated and published")
        left_check = False
        right_check = False

def generate_n_pub():
        global alpha
        global xcir
        global delta_t
        global x_ant
        global y_ant
        global theta_ant
```

```
        global left_speed
        global right_speed
        global new_position


#       Linear velocity
        vk = (alpha * (left_speed - right_speed)) / 2

#       Total displacement
        delta_S = vk * delta_t

#       Angular velocity
#       wk = (alpha * (right_speed - left_speed) / (2 * xcir)

#       delta_theta = wk * delta_t

#       x, y and theta
#       x = x_ant + delta_S * math.cos(theta_ant + (delta_theta/2))
#       x = y_ant + delta_S * math.sin(theta_ant + (delta_theta/2))
#       theta = theta_ant + delta_theta

#       Publish the the Pose
        new_position.data[0] = x
        new_position.data[0] = y
        new_position.data[0] = theta
        Pose_robot_pub.publish(new_position)

#       Save new Pose
        x = x_ant
        y = y_ant
        theta = theta_ant


if __name__ = '__main__':
        rospy.init_node("odometry")

        left_speed = rospy.Subscriber("/linearV_left", Float32, l_speed_cb)
        right_speed = rospy.Subscriber("/linearV_right", Float32, r_speed_cb)
        time_trigger = rospy.Subscriber("/time_trigger", String, check_cb)

        Pose_robot_pub = rospy.Publisher("/Pose_robot", Float32MultiArray, queue_size = 10)

        rospy.spin()
```

Etapa	**Figura 48 –** Rede Ros da Teleoperação habilitada em ambos os computadores.
Script	/teleop_twist_keyboard
Programação	rosrun teleop_twist_keyboard teleop_twist_keyboard.py

Etapa	**Figura 52 –** Rede ROS durante a exploração e mapeamento do ambiente simulado.
Script	roslaunch hector_slam hector_mappin
Programação	roslaunch diffbot_gazebo diffbot.launch world_name:='$(find diffbot_gazebo)/worlds/turtlebot3_world.world'

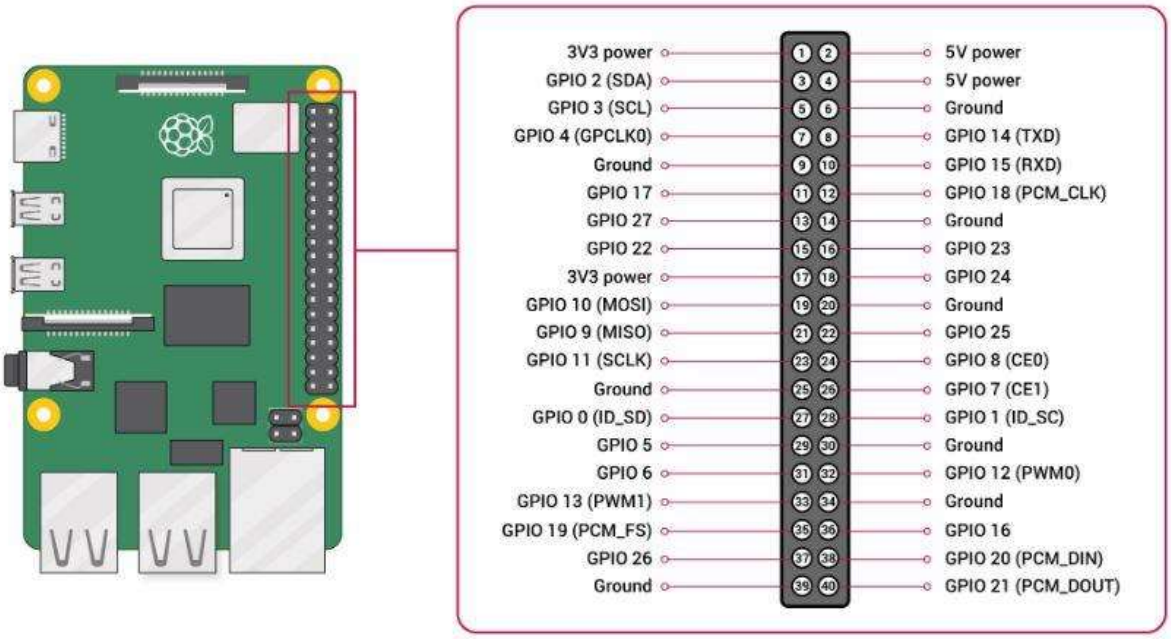
Etapa	**Figura 52 –** Rede ROS durante a exploração e mapeamento do ambiente simulado.
Script	/hector_mapping
Programação	roslaunch hector_mapping_diff.launch

Etapa	**Figura 53 –** Nó do Sensor RPLidar funcionando e integrado a Rede ROS estabelecida.
Script	/RPLidarNode
Programação	roslaunch RPLidar_ros RPLidar.launch

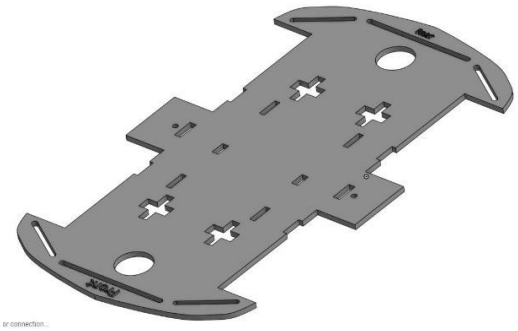
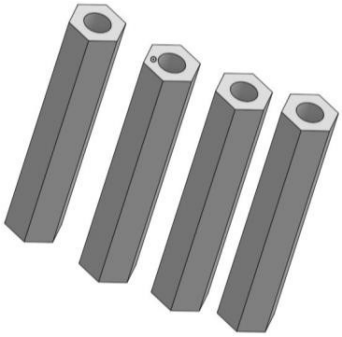
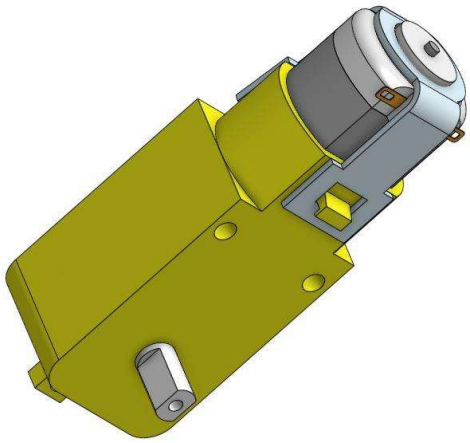
Etapa	**Figura 55 –** Rede ROS da integração final de todas as funções do Projeto *Firmware*.
Script	/hector_mapping
Programação	roslaunch agv_bringup hector_mapping.launch

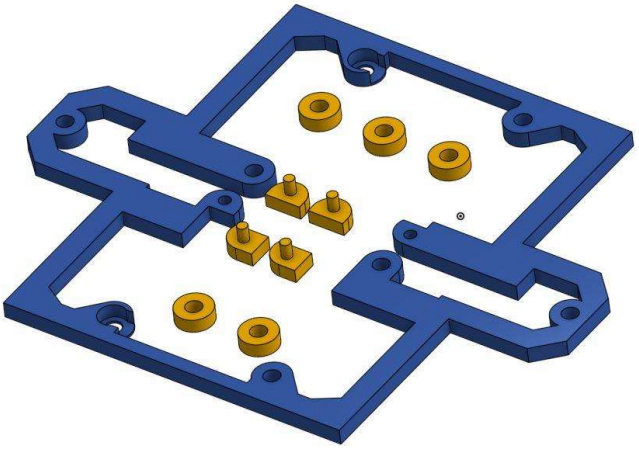
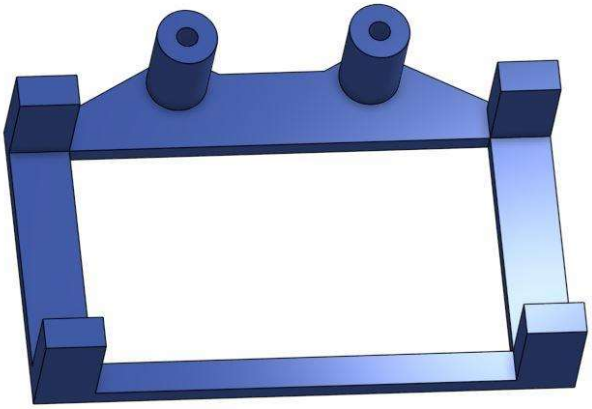
ANEXO B

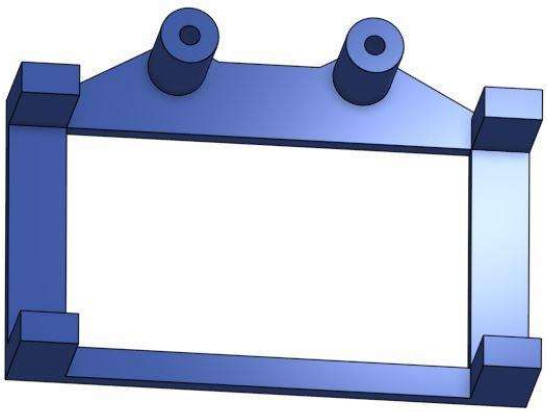
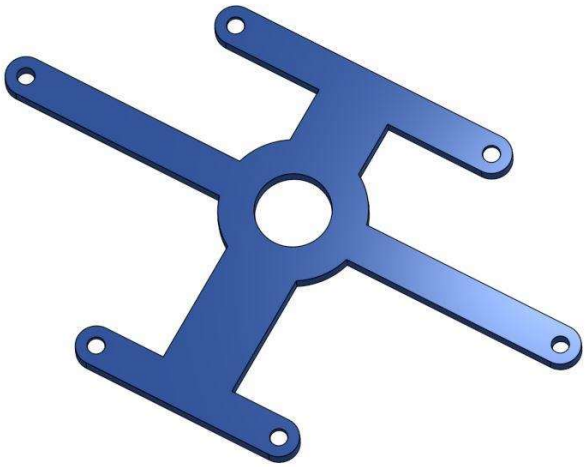
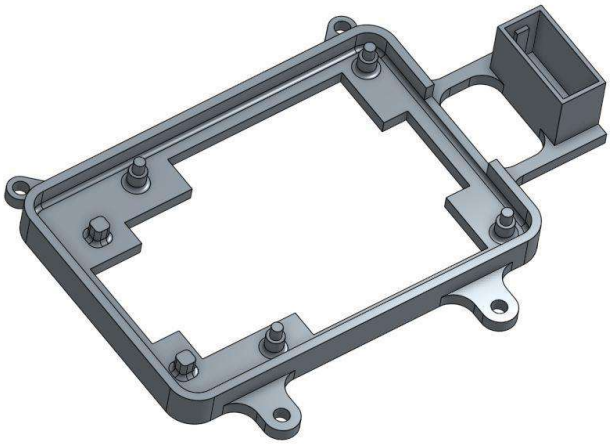
Anexo B.1 - Esquemático dos pinos do barramento GPIO da placa Raspberry Pi 3B+:

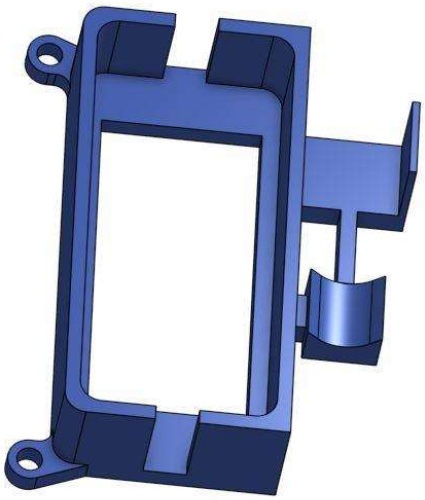
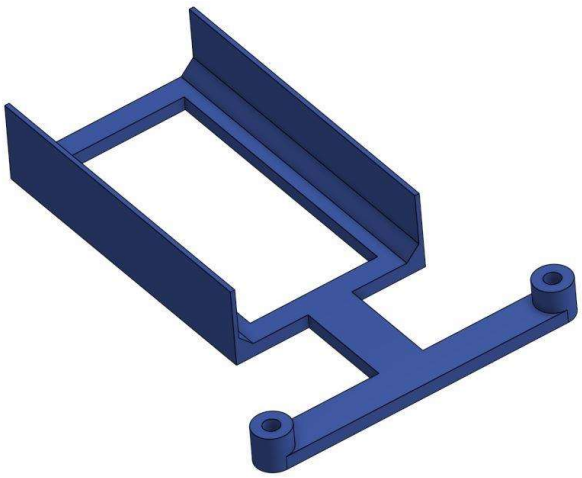


Anexo B.2 - Lista de Peças usadas no Projeto Mecânico

Origem	Nome	Peça
Kit Chassi 4WD	Chassi	
Kit Chassi 4WD	Espaçadores Originais	
Kit Chassi 4WD	Motores “TT Motor”	

Kit Chassi 4WD	Rodas	
Projeto Mecânico: Chassi Inferior	Suporte Central (e seus espaçadores)	
Projeto Mecânico: Chassi Inferior	Suporte Charger	

Projeto Mecânico: Chassi Inferior	Suporte RP	
Projeto Mecânico: Chassi Superior	Suporte RPLidar	
Projeto Mecânico: Chassi Superior	Suporte Computador	

Projeto Mecânico: Chassi Superior	Suporte Botão e P4	
Projeto Mecânico: Chassi Superior	Cordão Umbilical	
Projeto Mecânico	Novos Espaçadores