UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES
CAMPUS PATOS DE MINAS

OTÁVIO AUGUSTO ROCHA DA CRUZ

DESENVOLVIMENTO DE UM ROBÔ MÓVEL *OPEN SOURCE* BASEADO EM ROS 2 PARA PESQUISA EM ROBÓTICA COOPERATIVA

PATOS DE MINAS - MG
2022

OTÁVIO AUGUSTO ROCHA DA CRUZ

DESENVOLVIMENTO DE UM ROBÔ MÓVEL *OPEN SOURCE* BASEADO EM ROS 2 PARA PESQUISA EM ROBÓTICA COOPERATIVA

Trabalho de conclusão de curso apresentado à banca examinadora como requisito de avaliação da disciplina de TCC2 da graduação em Engenharia Eletrônica e de Telecomunicações, da Faculdade de Engenharia Elétrica, da Universidade Federal de Uberlândia, Campus Patos de Minas.

Orientador: Prof. Daniel Costa Ramos

Patos de Minas - MG
2022

OTÁVIO AUGUSTO ROCHA DA CRUZ

DESENVOLVIMENTO DE UM ROBÔ MÓVEL *OPEN SOURCE* BASEADO EM ROS 2 PARA PESQUISA EM ROBÓTICA COOPERATIVA

Trabalho de conclusão de curso apresentado à banca examinadora como requisito de avaliação da disciplina de TCC2 da graduação em Engenharia Eletrônica e de Telecomunicações, da Faculdade de Engenharia Elétrica, da Universidade Federal de Uberlândia, Campus Patos de Minas.

Orientador: Prof. Daniel Costa Ramos

Patos de Minas, 18 de fevereiro de 2022.

Banca Examinadora

_____

Prof. Dr. Daniel Costa Ramos – FEELT/UFU (Orientador)

_____

Prof. Dr. Jeovane Vicente de Sousa – FEELT/UFU (Membro 1)

_____

Prof. Dr. Laurence Rodrigues do Amaral – FACOM/UFU (Membro 2)

_____

Prof. Dr. Eliana Pantaleão – FACOM/UFU (Membro 3)

RESUMO

A robótica móvel é uma área em constante evolução, com constantes pesquisas. Neste aspecto, é comum a necessidade de uma plataforma robótica para averiguar hipóteses de uma pesquisa científica na prática. Assim sendo, faz-se necessária a aquisição ou montagem de um robô. Contudo, a construção pode ser complexa ou não estar diretamente relacionada ao escopo do estudo (como a implementações de funcionalidades de *software*), o que acarreta em perda significativa de tempo, enquanto que a compra pode ser frustrada pelos altos preços. Ademais, encontrar robôs móveis de baixo custo e bem documentados que possibilitem a reprodução não é algo trivial. O presente trabalho procurou preencher estas lacunas, a partir de uma plataforma robótica móvel, denominada SciCoBot (*Science, Coffee* e *Robot*), de baixo custo, *open source*, modular e expansiva, baseada em ROS 2, micro-ROS, Arduino Due e Raspberry Pi. Sendo a plataforma idealizada como ferramenta de pesquisa para temas relacionados a robótica cooperativa, principalmente para emprego na Universidade Federal de Uberlândia, campus Patos de Minas. Para isso, uma versão inicial com alguns sensores foi montada e uma arquitetura de código modular para o Arduino e o Raspberry foi desenvolvida. Procurando validar a versão construída, aplicações de *software* com os sensores foram exemplificadas e documentadas no GitHub. Com a plataforma robótica pronta e funcional, foi possível compará-la com outras estruturas, onde a SciCoBot se destaca para uso especifico na UFU, já que dispõe de dispositivos disponíveis no laboratório, o que potencializa o baixo custo e possibilita que projetistas da UFU utilizem seus conhecimentos prévios para compreensão do robô. Além disso, este trabalho fornece uma boa revisão de estruturas robóticas disponíveis na comunidade *open source.*

Palavras-chave: Robótica móvel. ROS2. Arduino. Raspberry Pi. Scicobot.

ABSTRACT

Mobile robotics is an area in constant evolution, with constant research. In this regard, it is common to need a robotic platform to investigate hypotheses of scientific research in practice. Therefore, the acquisition or assembly of a robot is necessary. However, the construction can be complex or not be directly related to the scope of the study (such as the implementation of *software* features), which leads to reduced time loss, while that the purchase can be frustrated by the high prices. Furthermore, finding low-cost, well-documented mobile robots that make reproduction possible is not trivial. Seeking to fill these gaps, the present work proposes the development of a mobile robotic platform, called SciCoBot (Science, Coffee and Robot), of low cost, modular and expansive, based on ROS 2, micro-ROS, Arduino Due and Raspberry Pi. The platform is idealized as a research tool for topics related to a robotic cooperative, mainly for employment at Universidade Federal de Uberlândia, Patos de Minas campus. For this, an initial version with some sensors was assembled and a modular code architecture for Arduino and Raspberry was developed. Seeking to validate the built version, software applications with the sensors were exemplified and documented on GitHub. With the robotic platform ready and functional, it was possible to compare it with other structures, where SciCoBot stands out for specific use in the UFU, since it has devices available in the laboratory, which enhances the low cost and allows UFU designers to use their previous knowledge to understand the robot. Furthermore, this work provides a good review of robotic structures available in the open-source community.

Keywords: Mobile robotics. ROS2. Arduino. Raspberry Pi. Scicobot.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AGPL	*Affero General Public License*
AUVs	*Autonomous Underwater Vehicle*
BOM	*Bill of Materials*
DART	*Dynamic Animation and Robotics Toolkit*
DDS	*Data Distribution Service*
GPL	*General Public License*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronic Engineers*
IMU	*Inertial Measurement Unit*
IR	*InfraRed*
kB	*Kilobyte*
LARCC	Laboratório de Robótica, Comunicação e Controle
LED	*Light Emitting Diode*
LGPL	*Lesser General Public License*
LiDAR	*Light Detected And Ranging*
MHz	*Megahertz*
MPL	*Mozilla Public License*
MQTT	*Message Queuing Telemetry Transport*
MRS	*Multi Robot Systems*
NASA	*National Aeronautics and Space Administration*
ODE	*Open Dynamics Engine*
OSRF	*Open Source Robotics Foundation*
OSS	*Open Source Software*
POO	Programação Orientada a Objetos
QoS	*Quality of Service*
RAM	*Random Access Memory*
RCL	*ROS Client Library*
RCLC	*ROS Client Library for C*
RCLCPP	*ROS Client Library for C++*

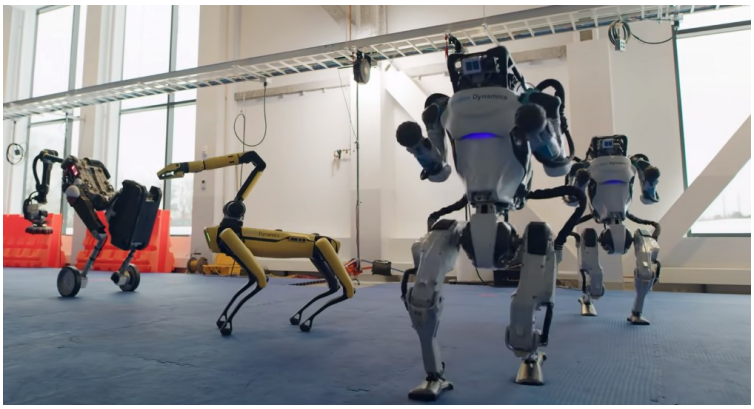
RCLPY	*ROS Client Library for the Python*
ROS	*Robot Operating System*
RTPS	*Real-Time Publish Subscribe*
RViz	*ROS Visualization*
SciCoBot	*Science Coffee and Robot*
SLAM	*Simultaneous Localization and Mapping*
SO	Sistema Operacional
SSH	*Secure Shell*
UAV	*Unmanned Aerial Vehicle*
UDP	*User Datagram Protocol*
UFU	Universidade Federal de Uberlândia
UGV	*Unmanned Ground Vehicle*
USB	*Universal Serial Bus*
VNC	*Virtual Network Computing*
Wi-Fi	*Wireless Fidelity*
WLAN	*Wireless Local Area Network*

SUMÁRIO

1 INTRODUÇÃO

A robótica, palavra popularizada devido ao escritor de ficção cientifica Isaac Asimov, é um ramo multidisciplinar associado à construção e concepção de robôs. Um dos principais polos de desenvolvimento robótico são os Estados Unidos, onde existe um setor consolidado, com uma série de instituições proeminentes, como *startups* de robótica e as maiores empresas de tecnologia do mundo (ESTOLATAN et al., 2021). Dentre as *startups*, destaca-se a Boston Dynamics, empresa famosa pela divulgação de seus robôs pela *internet,* Figura 1.1 (KÜSTER; SWIDERSKA; GUNKEL, 2020; YOUTUBE, 2018).

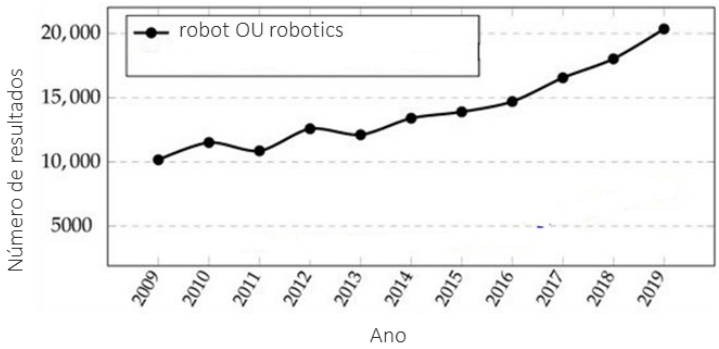
Figura 1.1 - Robôs da Boston Dynamics dançando.



Fonte: (YOUTUBE, 2020).

A popularidade da robótica reflete em mais pesquisas neste campo. A Figura 1.2 indica a quantidade de resultados ao pesquisar a palavra *robotics* no banco de dados *Institute of Electrical and Electronic Engineers* (IEEE) *Xplore*, em agosto de 2020 (IEEE, 2021; JAHN et al., 2020), onde é perceptível que desde 2016 a quantidade de pesquisas na área apresenta um crescimento.

Figura 1.2 - Quantidade de resultados ao pesquisar termos relacionados a robótica no IEEE *Xplore*.



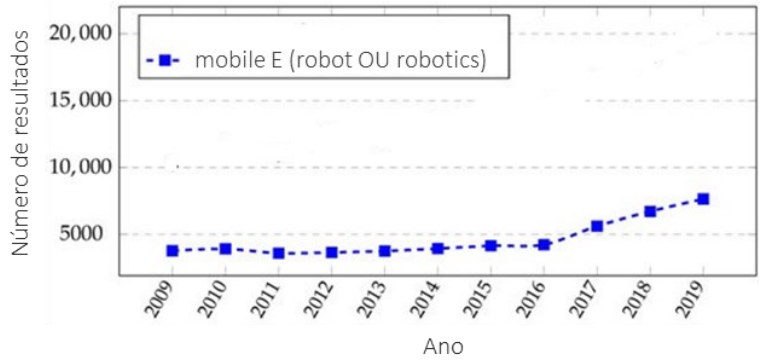
Fonte: modificado (JAHN et al., 2020).

A evolução e a utilização da robótica é uma realidade, mas o Brasil não está entre os países que mais vêm participando desse progresso. Segundo dados da Federação Internacional

de Robótica, na Coreia do Sul o número de robôs para cada 10 mil operários ultrapassa 500, já em Cingapura, Japão e Alemanha a média é superior a 300. Enquanto isso, no Brasil, avalia-se que sejam cerca de 10 robôs para cada 10 mil operários. Além disso, a quantidade de robôs instalados no Brasil equivale a 0,6% dos instalados no mundo, número muito inferior à China e EUA. Sendo que, no Brasil, o setor automobilístico sozinho possui mais de 50% da quantidade total de robôs utilizados no país (FENERICK; VOLANTE, 2020).

Os robôs móveis são uma categoria de robôs que possuem a capacidade de se locomover, podendo ser aéreos, aquáticos ou terrestres (HENRIQUE, 2019; JAHN et al., 2020). Os robôs terrestres podem ser subdivididos de acordo com o dispositivo de locomoção utilizado: rodas, esteiras e pernas, sendo os robôs sobre rodas os mais simples e populares. Neste contexto, a robótica móvel tem sido objeto de constante pesquisa, a Figura 1.3 apresenta o número de publicações na IEEE *Xplore* ao pesquisar por *mobile robot* e *mobile robotics*, onde, percebe-se um crescente aumento desde 2016 (JAHN et al., 2020).

Figura 1.3 - Quantidade de resultados ao pesquisar termos relacionados a robótica móvel no IEEE *Xplore*.



Fonte: modificado (JAHN et al., 2020).

Em geral, a construção de um robô é uma tarefa complexa, que envolve conceitos de mecânica, elétrica, eletrônica e programação. Neste sentido, foram desenvolvidos ao longo dos anos algumas estruturas robóticas que buscam facilitar e padronizar implementações, especialmente para uso em ensino e pesquisa. Os robôs para pesquisa têm como objetivo oferecer aos pesquisadores a capacidade de validar rapidamente vários algoritmos e soluções, já que dependendo da situação, a implementação de um robô desde o início pode ser inviável, dada toda a complexidade envolvida (RHOADES; SABO; CONRAD, 2017). Na Tabela 1.1 são indicadas algumas plataformas desenvolvidas para fins educacionais e/ou de pesquisa, sendo a maioria produzida no exterior. Na tabela, é possível observar que os custos apresentam grandes oscilações e o desenvolvimento de novas plataformas é recorrente.

Tabela 1.1 - Alguns robôs usados na educação e/ou pesquisa. Conversão para real com uma Libra equivalente a R$ 7,12, um Euro R$ 5,93 e um dólar R$ 5,27, realizada 01/02/2022.

Robô	Ano	Custo [R$]
Pioneer 3DX	1995	19171,69
Amigobot	2001	18679,5
E-puck	2009	4984,00
Turtlebot 2	2012	9920,89
Rice r-one	2013	1424,00
Colias	2014	178,00
Edison	2014	305,87
Khepera IV	2016	17784,07
MicroMVP	2017	498,40
Turtle 3 (burger)	2017	3474,68
Turtle 3 (waffle)	2017	8296,07
Rosbot 2.0	2018	9849,73
Husky A200	-	163370,00
Summit-XL	-	76415,00
Robotino	-	44795,00
E-puck2	-	5270,00
Turtlebot 2i	-	11594,00

Fonte: Modificado (AMSTERS; SLAETS, 2019; ARVIN et al, 2019; MARÍN, 2018).

Um dos componentes mais importantes de qualquer sistema robótico é a arquitetura do *software*, isso é ainda mais evidente ao usar uma equipe de robôs. No entanto, o projeto de uma arquitetura de *software* é uma área completa de estudo. Diante disso, projetistas recorrem a arquiteturas existentes para estruturar seus projetos. Dentre as arquiteturas de *software* robóticos existentes, *Robot Operating System* (ROS) é um dos mais populares, sendo utilizado tanto na comunidade acadêmica quanto na indústria (GARZÓN, 2017; LEMOS; MENDONÇA, 2020; SANTOS; CUNHA; MACEDO, 2019). Em suma, ROS possui um conjunto de ferramentas de *software*, bibliotecas e convenções, com o objetivo de simplificar a tarefa de desenvolver *softwares* para um robô. Isso facilita e agiliza a implementação, não sendo necessário recomeçar a cada novo trabalho (TSARDOULIAS; MITKAS, 2017). Um dos diferenciais do ROS é que ele possui uma comunidade ativa, que compartilha inúmeros módulos reutilizáveis (ALVARO; KOZIOL, 2020). Atualmente, tem-se uma nova versão do

ROS, denominada ROS 2, com foco em operações descentralizadas, podendo ser um marco para as pesquisas em sistemas multi-robôs.

A constituição eletrônica de um robô pode variar muito. Sabe-se que o *hardware* influencia diretamente no desempenho do robô, pois permite a aplicação de algoritmos complexos em uma dada qualidade (DE SOUSA et al., 2018). Dentre os diversos eletrônicos utilizados destaca-se aqui o Raspberry Pi e Arduino, estruturas abertas bastante difundidas (BENTO, 2018; MORENO; PAEZ, 2017; VAUTIER et al., 2018). O Arduino é uma plataforma de *hardware* e *software* que roda em Mac, Windows e Linux; ele possui baixo custo; dispõe de fácil acesso e implementação; comunidade ativa; é facilmente adaptável e extensível (ARDUINO, 2018; BENTO, 2018). Por outro lado, o Raspberry Pi, é um mini computador de baixo custo, suas principais características são: grande quantidade de exemplos publicados na Internet, o que auxilia o desenvolvimento de projetos; suporte de sensores essenciais para direção autônoma, como *Light Detected And Ranging* (LiDAR) e possibilidade de instalação do sistema operacional Ubuntu, oficialmente compatível com ROS (BENTO, 2018; NAKAMOTO; KOBAYASHI, 2019).

Raspberry, Arduino e ROS são peças convenientes para aplicações robóticas, inclusive em casos envolvendo grupos de robôs cooperativos de até média complexidade com recursos limitados de *hardware*, como em: ESCOBAR et al. (2020); KASSAWAT e CERVERA (2019); SAYED, AMMAR e SHALABY (2020); e VERMA (2019). Ou seja, em situações onde é necessária uma intensa e correta interação entre robôs para executar uma tarefa de certa complexidade, como transporte de carga, busca, resgate e exploração de área (VERMA, 2019).

Neste contexto, a padronização da plataforma é vital para redução dos custos e da complexidade de manutenção. Nota-se, também, a importância do desenvolvimento da robótica em nosso país, de forma a acompanhar as transformações dessa tecnologia. Neste sentido, para desenvolver métodos e algoritmos para solucionar problemas nessa área, é preciso validar o procedimento de alguma forma, seja via simulação, teoria ou robô real. É desejável que esta validação seja feita em um robô real, sendo os resultados potencializados se este for parte de um *framework* padronizado, tanto de *software* quanto de *hardware*, evitando retrabalhos e facilitando a aprendizagem do usuário.

1.1 TEMA DO PROJETO

Este trabalho tem como objetivo o desenvolvimento de um protótipo robótico móvel básico, para pesquisas em robótica cooperativa, principalmente no que se refere ao *software*. A plataforma, denominada SciCoBot *(Science, Coffee e Robot)* ou, ainda, Scicobot, deve ser capaz

de utilizar o sistema ROS 2 em conjunto com Arduino e Raspberry, além disso, deve atender a requisitos de ser de fácil acesso, ser de uso *indoor* e, sobretudo, apresentar *software* modular e ser viável para implementação em pesquisas com robótica cooperativa, realizadas na Universidade Federal de Uberlândia (UFU) campus Patos de Minas.

1.2 PROBLEMATIZAÇÃO

Na ciência é comum haver comparações entre trabalhos similares, como quando quer-se destacar certos progressos. Um problema que pode ocorrer nesse sentido, é quando necessita-se comparar implementações de *software*, como: técnicas de controle, navegação ou cooperação, mas a comparação é inviável devido à grande discrepância entre *hardwares* utilizados. Logo, a padronização entre as plataformas utilizadas pode facilitar este processo.

Dentre as plataformas robóticas existentes comercialmente, têm-se as de projeto fechado, sendo difícil a modificação ou a substituição dos elementos de *hardware* e *software*, onde as alterações estão geralmente limitadas às fornecidas pelo fabricante (HENRIQUE, 2019). Além disso, têm estruturas robóticas com proposta semelhante à deste trabalho, porém, a maioria não considera a realidade financeira brasileira, Tabela 1.1. Já no que diz respeito as propostas brasileiras, que poderiam ser uma boa alternativa, encontraram-se dificuldades nas que se teve conhecimento, como as de: DE SOUSA (2018), HENRIQUE (2019) e REZECK et al. (2017), geralmente envolvendo descontinuidade ou falta de informações para reprodução.

Tendo em vista a criação do Laboratório de Robótica, Comunicação e Controle (LARCC) na UFU campus Patos de Minas, onde uma das áreas de pesquisa será a utilização de robôs cooperativos e a influência da rede de comunicação na conclusão de suas tarefas, deseja-se a criação de uma plataforma padronizada para pesquisa, onde os discentes consigam desenvolver soluções que possam ser reaproveitadas em projetos futuros. Nesta consideração, o custo é um dos fatores mais relevantes, junto com a utilização de tecnologias disponíveis no laboratório.

A utilização do protótipo proposto neste trabalho pode possibilitar que pesquisadores da UFU e externos, foquem e aprofundem em temas robóticos da atualidade, como robótica cooperativa, mapeamento de ambiente desconhecidos, *Simultaneous Localization And Mapping* (SLAM) e autonomia robótica. Além disso, o uso de uma estrutura padronizada nos projetos pode facilitar a comparação de desempenho entre as soluções desenvolvidas. Para tal, neste projeto, será avaliada a disponibilidade e preço dos materiais a serem utilizados, considerando o cenário brasileiro e do curso de Engenharia Eletrônica e de Telecomunicações da UFU, campus Patos de Minas. Ainda, por se tratar de um projeto aberto (do termo em inglês

open source), existe uma ampla comunidade de usuários que podem, com o tempo, contribuir com sugestões de melhorias. Ou seja, a modularidade proposta pode possibilitar a otimização e inserção de novas funcionalidades e estilos de estrutura, tanto a nível de *software* quanto de *hardware*.

Outro ponto importante é a utilização do ROS 2 *Foxy Fritzing* como arquitetura de *software*, que, devido seu recente desenvolvimento, ainda não possui muitos materiais disponíveis, especialmente em português, o que dificulta sua ampla utilização por discentes de graduação.

1.3 HIPÓTESE

É levantada a hipótese de que é possível elaborar um protótipo robótico modular e compatível às necessidades de pesquisa no campus, sobretudo quanto ao *software* e comunicação, para emprego em pesquisas na área de robótica cooperativa.

1.4 OBJETIVOS

1.4.1 Objetivos Gerais

O objetivo geral deste trabalho é a elaboração e montagem de uma estrutura robótica móvel de baixo custo e expansível, que utilize ROS 2, Arduino e Raspberry, voltada para pesquisa, principalmente para as realizadas no curso de Engenharia Eletrônica e de Telecomunicações da UFU, campus Patos de Minas. O foco deste trabalho consiste na implementação do *software* robótico baseado em ROS 2. Para isto, será montada uma estrutura física para testes. Após a montagem, será desenvolvida uma documentação descritiva do projeto, para que outros pesquisadores possam facilmente reproduzir a estrutura.

1.4.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

1. Realizar estudo sobre estruturas já existentes para estipular os requisitos mínimos de *hardware* e definir os componentes que o protótipo deve possuir;
2. Montar um protótipo com sistema operacional baseado em Linux no Raspberry Pi, ROS 2 e conectividade com Arduino, garantindo o funcionamento adequado dos mesmos;
3. Documentar todo o desenvolvimento do protótipo e disponibilizar o projeto *online*.

Com a conclusão desses objetivos, espera-se ter um protótipo robótico modular com arquitetura básica de *hardware* e *software* pronta para uso em pesquisas.

1.5 JUSTIFICATIVAS E TRABALHOS RELACIONADOS

Atualmente, existem plataformas robóticas profissionais para uso em pesquisas, no geral, essas plataformas possuem alta qualidade e estão disponíveis comercialmente, tendo como exemplo da Tabela 1.1: Husky A200, Summit-XL, Robotino, Khepera IV, E-puck2 e o Turtlebot 2i (MARÍN, 2018). Muitas delas são apresentadas como estruturas de fácil acesso e de baixo custo, o que pode não ser válido para o Brasil, que tem realidade mercadológica e financeira diferente do país de origem dessas estruturas. Isto faz com que na prática, muitas dessas estruturas acabem sendo de difícil acesso, já que precisam ser importadas, e de alto valor, como pode-se notar nos valores anteriormente especificados. Ainda há aquelas plataformas que foram descontinuadas, como: Pioneer e AmigoBot. Tem-se, ainda, estruturas comerciais criadas ou montadas no Brasil, que poderiam resolver o problema de custo e acessibilidade, como RoboDeck (XBOT, 2017), mas neste caso, a empresa foi descontinuada.

Para a Tabela 1.1 foi considerada a cotação de Libra igual a R$ 7,12, do Euro igual a R$ 5,93 e do dólar igual R$ 5,27, todas realizadas no dia 01 de fevereiro de 2022. Dessa forma, a fim de estabelecer um texto mais fluído e evitar a repetição desnecessárias de informação, considere essas cotações para conversões que não possuem informações sobre a cotação utilizada, para este e os próximos capítulos,

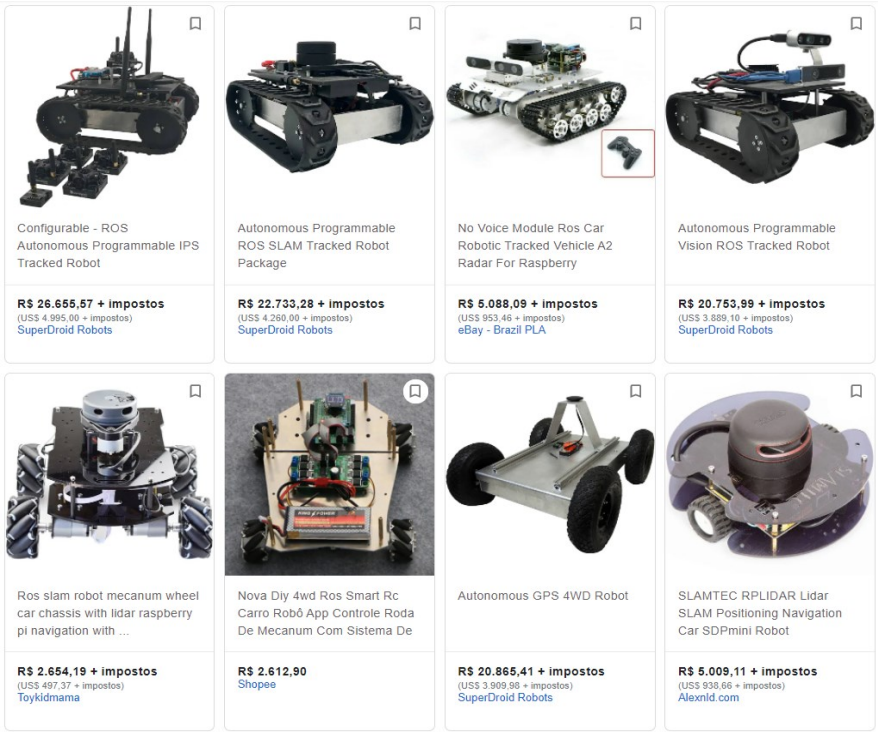
O TurtleBot é um robô móvel comercialmente famoso, ele é conceitualmente modular, customizável, de baixo custo e baseado em ROS, o que o torna uma boa plataforma para projetos de SLAM. No entanto, em termos comparativos, no Brasil o modelo TurtleBot3 Burger, por exemplo, custa por volta de R$ 7463,78 em ALIEXPRESS (2021), valor elevado. A Figura 1.4 mostra um TurtleBot3 Burger, que é equipado com: LiDAR 360°; OpenCR, como placa de baixo nível para controle dos atuadores; sensor *Inertial Measurement Unit* (IMU); e um Raspberry Pi, usado para programação de alto nível (ELKILANY et al., 2020).

Figura 1.4: TurtleBot3 Burger.

Fonte: Modificado (ELKILANY et al., 2020).

Caso o projetista avalie e descarte o uso de plataformas comercialmente conhecidas no meio robótico, como as apresentadas anteriormente, pode-se procurar no Google Shopping alguma alternativa. Nesse sentido, pesquisou-se inicialmente “robô ros”, mas a maioria dos resultados eram pequenos robôs humanoides, então, procurou-se “*robot ros”*. A Figura 1.5 mostra alguns dos resultados ao efetuar essa pesquisa, onde os robôs dispõem de um visual robusto e altos preços.

Figura 1.5: Alguns resultados do Google Shopping para: *robot ros,* em 29 de abril de 2021.

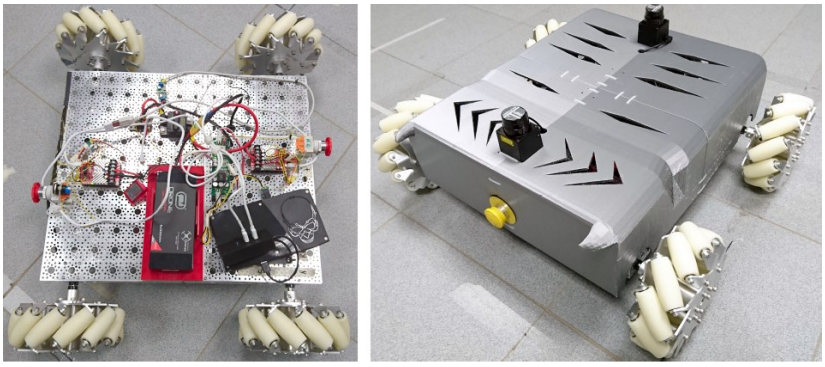


Fonte: Adaptado da Internet pelo Autor.

Tendo em vista esse cenário, a comunidade científica mundial vem propondo arquiteturas que podem ser construídas pelo próprio pesquisador, uma vez que são mais baratas (MARÍN, 2018) e podem se adequar às condições financeiras e especificas de cada projeto ou universidade. A seguir são descritas algumas dessas propostas.

O trabalho de MARÍN (2018) propõe uma estrutura robótica baseada em ROS 2, de *hardware* aberto, modular e expansível para pesquisa, com custo menor que plataformas profissionais, mas com desempenho e recursos semelhantes. Com os componentes selecionados, o custo da plataforma proposta é de aproximadamente R$ 8959,00, sem a unidade de controle, que deve ser escolhida conforme a necessidade, podendo ser um Raspberry Pi 3B+. A estrutura desenvolvida, Figura 1.6, é larga e possui rodas do tipo *mecanum*, uma espécie de roda omnidirecional que tem sido bastante usada nos últimos anos (LI et al., 2018).

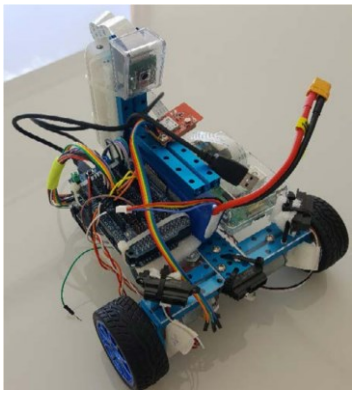
Figura 1.6: Estrutura proposta por Marín.



Fonte: (MARÍN, 2018).

Uma plataforma robótica de baixo custo destinada ao ensino e pesquisa é apresentada por DE SOUSA (2018). A Figura 1.7 mostra o robô, ele dispõe de alguns sensores; Arduino Due, para execução de códigos de controle; e um Raspberry Pi, utilizado para processamento de imagens e comunicação wireless, sendo a comunicação baseada no protocolo *Message Queuing Telemetry Transport* (MQTT). A plataforma apresenta custo de aproximadamente R$ 1500,00, podendo ser reproduzida por meio de impressora 3D, apesar de não estar explicito onde foram disponibilizados os arquivos.

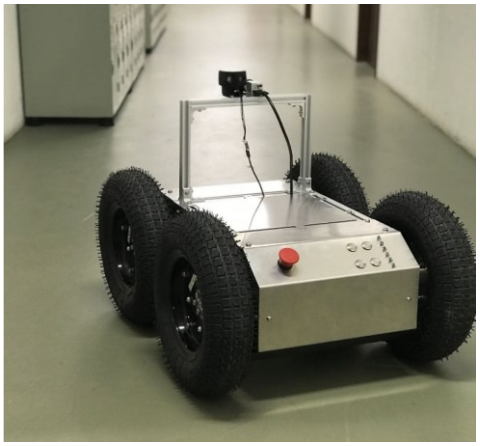
Figura 1.7 - Estrutura aberta e totalmente montada



Fonte: (DE SOUSA, 2018).

Um robô móvel *open source* e de baixo custo é projetado por HENRIQUE (2019), Figura 1.8. O trabalho descreve o desenvolvimento do robô e disponibiliza link para reconstrução da plataforma, entretanto, ao averiguar o link fornecido observou-se que encaminhava para uma página inexistente. A plataforma utiliza o Intel Nuc NUC6i5SYB, dispositivo com preço de R$ 1212,10 (STM, 2021). Além disso, a plataforma dispõe de um Sweep LiDAR, de valor aproximado de R$ 1839,23 (KARLSSON ROBOTICS, 2021). Considerando-se apenas estes dois componentes, tem-se um custo de cerca de R$ 3051,33.

Figura 1.8 - Estrutura aberta e totalmente montada



Fonte: (HENRIQUE, 2019).

O HeRo (Figura 1.9) é uma plataforma robótica para aplicações em robótica de enxame da Universidade Federal de Minas Gerais. O objetivo é disponibilizar uma plataforma de baixo custo (R$ 94,86), pequeno porte, aberta, de fácil montagem, modular, impresso em impressora 3D e integrada a ROS (REZECK et al., 2017). O robô possui como componentes básicos: ESP8266, sensor IR TCRT5000, bateria e motores. O ESP8266 é um microcontrolador amplamente usado, compatível com *Wireless Fidelity* (Wi-Fi) e Bluetooth, porém, comparado ao Raspberry, possui capacidade computacional muito menor (AUFRANC, 2020). O material para implementação está disponível em um repositório público chamado GitHub (VERLAB, 2017), onde estão dispostas as informações para construir o pacote ROS desenvolvido para a plataforma.

Figura 1.9 - Estrutura aberta e totalmente montada

Fonte: (REZECK et al., 2017).

O SpotMicro, Figura 1.10, é um robô quadrúpede, similar ao Spot da Boston Dynamics (BOSTON DYNAMICS, 2021). Trata-se de um projeto aberto (CHEN et al., 2020), que possui todos os modelos de impressão 3D disponíveis em seu repositório. Os principais componentes utilizados no SpotMicro são: NVIDIA Jetson Nano/ Raspberry Pi 4, sensor ultrassônico HC-SR04 e giroscópio MPU-6050. Além disso, é fornecido o modelo de simulação robótica via

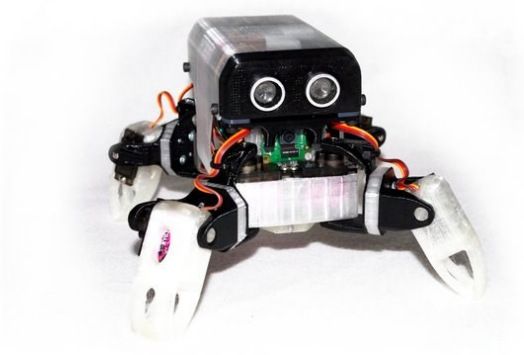
PyBullet. Tem-se, ainda, o Stanford Doggo (KAU et al., 2019; KAU, 2018) e Spot Micro (MIKE4192, 2020), estruturas similares ao SpotMicro e aparentemente promissoras, porém, com alto custo e complexidade. O Doggo, por exemplo, custa cerca de R$ 15810,00 (KAU et al., 2019).

Figura 1.10 - SpotMicro.

Fonte: (CHEN et al., 2020).

LoCoQuad (BERNAL; CIVERA, 2020), Figura 1.11, é um robô quadrúpede de propósito geral, projetado para pesquisa e educação. O projeto propõe um robô de baixo custo (entre R$ 790,50 – 869,55) e alto grau de flexibilidade, podendo ser produzido em impressora 3D. Onde os componentes básicos utilizados foram: Raspberry Pi 3 modelo B, PCA-9685, IMU GY-521, HC-SR04, câmera Raspberry Pi, bateria de *LiPo* e *buzzer*. O *hardware* e *software* são abertos e podem ser encontrados no GitHub (BLACKROAD, 2019), onde é indicado que as próximas versões devem ter todos os recursos ROS implementados e prontos para uso.

Figura 1.11: LoCoQuad.



Fonte: (BERNAL; CIVERA, 2020).

Encontram-se, ainda, propostas de plataformas com esteiras, como o do trabalho de THÁCIK, BREZINA e JADLOVSKÁ (2019), onde é prevista uma plataforma para emprego em pesquisas e ensino, com baixo custo, arquitetura modular e de fácil montagem, sendo o chassi montado em impressora 3D. O protótipo básico possui Raspberry e STM32.

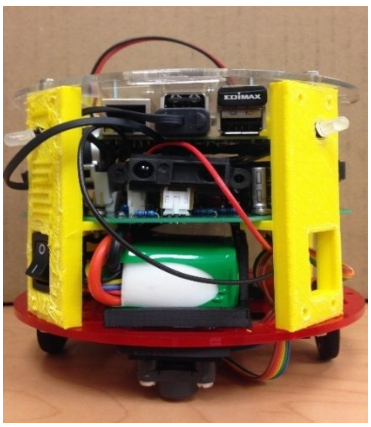
Posteriormente, a fim de validar o protótipo, são adicionados módulos para sensor ultrassônico, infravermelho, giroscópio e câmera (Figura 1.12).

Figura 1.12: Robô esteira proposto.

Fonte: (THÁCIK; BREZINA; JADLOVSKÁ, 2019).

A *Arizona State University* desenvolveu o Pheeno (Figura 1.13) em 2015, trata-se de um robô para pesquisa projetado para ser barato (peças de aproximadamente R$ 1686,40), modular e para uso em experimentos com vários robôs. O robô em si é um dispositivo de duas rodas controlado por um Arduino Pro Mini e Raspberry Pi, além disso, possui alguns sensores, *encoder*, câmera, IMU entre outros, que auxiliam em suas funções (ACS LABORATORY, 2017a; WILSON, 2015). Ademais, foi desenvolvido e disponibilizado suporte ROS para o Pheeno em (ACS LABORATORY, 2017b).

Figura 1.13: Pheeno.



Fonte: (ACS LABORATORY, 2017a).

E-puck (Figura 1.14) é um robô usado para pesquisa e educação que possui suporte Bluetooth; microcontrolador com 8 Kilobyte (kB) de *Random Access Memory* (RAM), 144 kB de memória flash e frequência de 60 Megahertz (MHz); e vários sensores de proximidade infravermelho (*InfraRed* - IR) (MILLARD et al., 2017). Ele pode ser usado para aplicações multi-robôs, onde, por exemplo, um grupo de robôs e-puck são usados para coordenar e

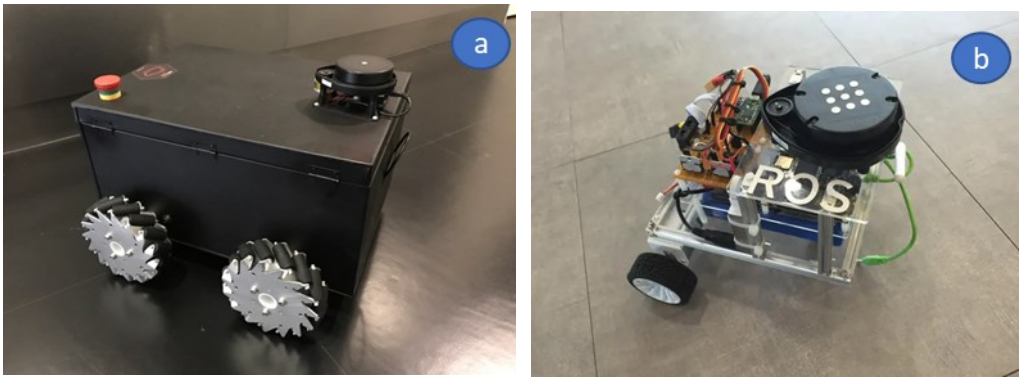
sincronizar ações a fim de movimentar um cubo o mais longe possível de sua posição inicial (ALKILABI; NARAYAN; TUCI, 2017). O e-puck possui alguns modelos, dentre eles destaca-se o Pi-puck, que dispõe de um Raspberry Pi zero W com suporte para ROS 2 (CYBERBOTICS, 2020).

Figura 1.14: E-puck.

Fonte: Modificado (MONDADA, 2010).

Linorobot (JIMENO, 2021) é um projeto que propõe um conjunto de robôs compatíveis com ROS 1 e ROS 2 de código aberto para estudantes, desenvolvedores e pesquisadores que necessitam de uma plataforma robótica de baixo custo na criação de novos aplicativos baseado em ROS. O Linorobot possui versões com duas e quatro rodas, como mostrado na Figura 1.15 (a) e (b), respectivamente. A proposta é interessante, no entanto, o baixo custo é questionável, as plataformas propostas utilizam: sensores a laser, microcontrolador Teensy e motores com *encoder*, componentes onerosos. Além disso, a descrição para montagem do chassi é pouco descritiva.

Figura 1.15: Linorobot. (a) Quatro rodas. (b) Duas rodas.

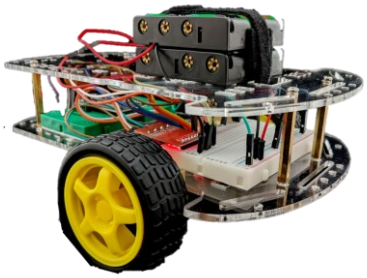


Fonte: (JIMENO, 2021).

O Hadabot, Figura 1.16, é um robô de código aberto construído a partir de um ESP32, sua estrutura base pode ser adquirida por R$ 526,74, havendo a possibilidade de expansões que adicionam sensores *encoder* e de distância a base robótica. O robô possui material didático relacionado a ROS 2 e descrição detalhada da montagem. E, ainda, destaca-se o suporte Docker

e a interface web disponibilizada pelo projeto. No entanto, trata-se de uma estrutura de nível básico, em termos de funcionalidade, onde, inicialmente, tem-se apenas a função de locomoção, com a adição de sensores a estrutura adquire mais funcionalidades e possibilidades de aplicação, mas, ainda, com pouco potencial para aplicações que demandam maior poder de processamento (HADABOT, 2021).

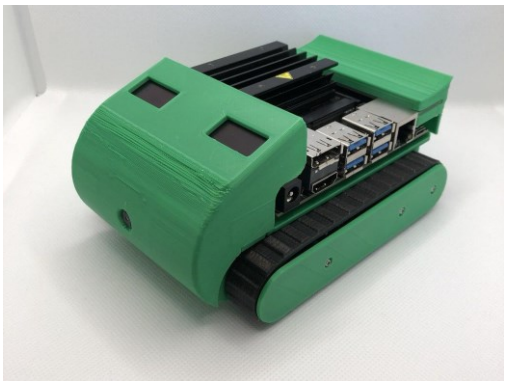
Figura 1.16: Hadabot.



Fonte: modificado (HADABOT, 2021).

O Nanosaur (BONGHI, 2022a), é um robô esteira compacto (Figura 1.17), imprimível em 3D, de código aberto e que utiliza uma placa NVIDIA Jetson Nano com ROS 2. Em suma, a Jetson Nano é como um pequeno computador, otimizado para uso de Inteligência Artificial (IA). Tecnicamente, essa placa possui 40 pinos, divididos entre: GPIO, I2C, I2S, SPI e UART; cpu quad-core ARM 1.43 GHz; 4 GB de memória; saída HDMI; e GPU 128-core Maxwell. O projeto especifica a montagem e apresenta exemplos de implementação interessantes. No entanto, a utilização da placa Jetson Nano aumenta significativamente o custo da arquitetura, já que seu valor é aproximadamente R$ 1850,00 (CASAS BAHIA, 2022).

Figura 1.17: Nanosaur.



Fonte: (BONGHI, 2022a).

O cenário apresentado mostra que nos últimos anos surgiram várias plataformas robóticas com objetivos semelhantes aos propostos por este trabalho (baixo custo, modular, terrestre, baseada em ROS e *open source*), sendo que, muitas das atuais ainda são caras e/ou apresentam várias dificuldades para replicar, não apresentando uma lista de peças completa ou

instruções de montagem (MARÍN, 2018; YU et al., 2017), por exemplo. Neste contexto, as principais contribuições deste trabalho são: fornecer material explicativo *online* sobre cada etapa de desenvolvimento e construção, visto que, a documentação fornecida pelos desenvolvedores é muitas vezes confusa, insuficiente e/ou em outra língua; considerar a realidade brasileira para escolha dos componentes físicos do robô; e cogitar possíveis modificações, como acréscimo de componentes de *hardware* e *software* para projetos mais complexos. Sendo que, o escopo do trabalho tem como principal foco o *software*/comunicação utilizando ROS 2.

Dessa forma, o presente trabalho irá fornecer um robô especialmente planejado para o campus, para influenciar e auxiliar novas pesquisas. Em um segundo momento, espera-se que esta plataforma seja continuamente desenvolvida, podendo ser usada como meio educacional, como em matérias relacionadas a programação, controle e eletrônica.

1.6 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados os problemas e as motivações do trabalho, apresentando pontos positivos que uma estrutura robótica básica, modular, de baixo custo e acessível pode trazer para pesquisadores. Além disso, foi discutido sobre estruturas atuais, que, em geral, possuem pouco conteúdo auxiliar, foram descontinuadas, são caras e/ou de difícil acesso no Brasil. No próximo capítulo será abordado a parte teórica, utilizada para o desenvolvimento do protótipo.

2 REFERENCIAL TEÓRICO

Neste capítulo serão abordadas as teorias básicas consideradas necessárias para compreensão do trabalho. Serão apresentados conceitos sobre: robótica; Arduino; Raspberry; *encoder*; sensores de obstáculo; ponte H, GitHub e Git, licenças *open source*, ROS, programação e acesso remoto.

2.1 ROBÓTICA

A palavra “robô” origina-se da palavra tcheca “robota” que significa “trabalho” ou “trabalho forçado”. Um sistema robótico pode ser definido como “um sistema mecânico reprogramável, controlado por computador, que pode sentir e reagir aos atributos do ambiente à medida que executa tarefas atribuídas com algum grau de autonomia” (KURDILA; BEM-TZVI, 2020).

Os robôs são mais adequados para aplicações que envolvem tarefas repetitivas, trabalhos que requerem precisão ou operação em ambientes perigosos. Dentre as aplicações robóticas, as médicas estão se tornando cada vez mais comuns, onde os robôs auxiliam os cirurgiões na realização de cirurgias. Esses robôs possuem alta precisão, podendo operar através de incisões muito menores com movimentos mais precisos (NIKU, 2020).

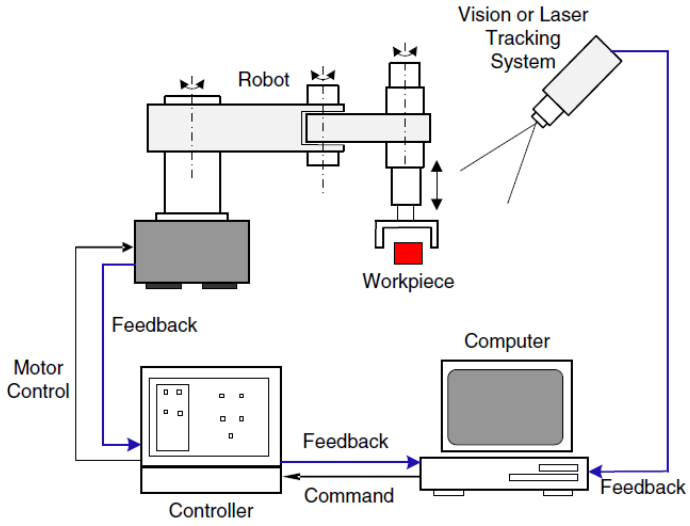
Para o desenvolvimento de um sistema robótico são necessários vários conceitos e técnicas, de muitos campos, especialmente engenharia mecânica, elétrica e tecnologia da informação. Dentre os vários subsistemas de um robô destacam-se: atuadores, comunicação, unidade de controle, manipulador e sensores. Qualquer sistema robótico pode incluir muitos desses componentes ou simplesmente alguns (KURDILA; BEM-TZVI, 2020).

Os atuadores servem como músculos do sistema, são responsáveis, por exemplo, por produzirem movimento. Já o subsistema de comunicação possibilita a comunicação ou acesso entre o robô e um servidor. Além disso, uma característica essencial de qualquer robô é que ele possua algum nível de autonomia ou inteligência, para tal, uma unidade de controle é vital. Um robô, pode precisar manipular mecanicamente o ambiente, como um objeto, nesses casos é comum o uso de um manipulador, como um braço robótico. Uma vez que um robô deve interagir com seu ambiente, muitos robôs também incluem conjuntos de sensores (KURDILA; BEM-TZVI, 2020).

A Figura 2.1 ilustra um manipulador robótico típico, neste sistema o robô é equipado com sensores que retornam medições de movimento angular dos motores, que auxiliam na definição da posição. Além disso, uma câmera ou *laser* podem ser usados para fornecer

medições da posição da garra ou de algum objeto importante para o sistema, sendo esta medição enviada ao computador para auxiliar no controle (KURDILA; BEM-TZVI, 2020).

Figura 2.1: Robô manipulador.



Fonte: (KURDILA; BEM-TZVI, 2020).

2.1.1 Robótica Móvel

Um robô móvel pode ser definido como um sistema mecânico capaz de se mover em um ambiente de forma autônoma (JAULIN, 2019). A classificação de um robô móvel pode ser realizada conforme vários fatores, um deles é quanto a anatomia, ou seja, construção física, onde tem-se três tipos: aéreo, aquáticos e terrestres. Logo, tem-se, por exemplo, os *Unmanned Aerial Vehicle* (UAVs), também conhecidos como drones, que são basicamente aeronave remotamente pilotada. Outro exemplo são os *Autonomous Underwater Vehicle* (AUVs), robôs autônomos que viajam embaixo da água, são frequentemente usados como plataformas de pesquisa para mapear o fundo do mar ou caracterizar propriedades físicas, químicas ou biológicas da água (COCHRAN et al., 2019). E, por último, tem-se os robôs terrestres, que são subdivididos em outros três grupos, conforme o dispositivo utilizado para locomoção, sendo eles: rodas, esteiras ou pernas. Destacam-se especialmente os robôs com rodas, foco deste trabalho, que em comparação com os demais, necessitam de *hardware* de menor complexidade, o que os tornam mais simples (HENRIQUE, 2019).

Uma revisão de temas relacionados à robótica móvel é realizada no trabalho de JAHN et al. (2020), onde foram pesquisados temas importantes da área na biblioteca digital IEEE *Xplore*, a fim de analisar o cenário no contexto do tema e da robótica como um todo. A Tabela 2.1 mostra parte dos tópicos abordados, onde navegação e autonomia são os que apresentam maior quantidade de resultados. Além disso, o trabalho de JAHN et al. (2020) apresenta gráficos

temporais com a quantidade de resultado de pesquisa para cada termo, sendo que, para os termos apresentados na Tabela 2.1, menos modularidade e computação em nuvem, há uma tendência de crescimento mais acentuada a partir de 2016.

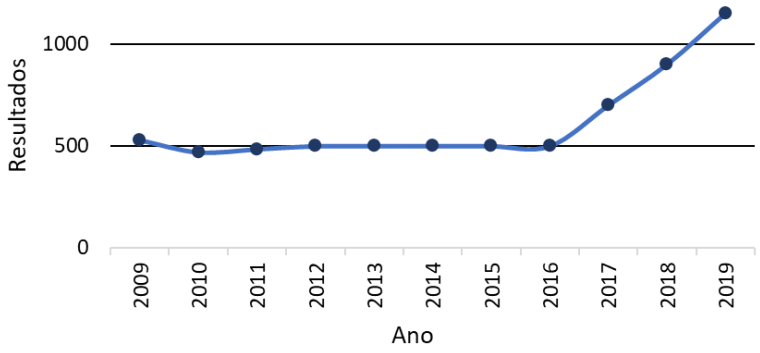
Tabela 2.1: Pesquisa e número de resultados para recursos de robôs móveis.

Tópicos	Termos Pesquisados	Resultados
Navegação	mobile E robot E (navigation OU mapping OU slam OU "collision avoidance" OU "path planning")	34423
Autonomia	mobile E robot E (autonomy OU autonomous)	24626
Cooperação Multi-robôs	mobile E robot E (swarm OU "multi robot" OU "networked robots")	9265
Capacidade de Tempo Real	mobile E robot E ("real time" OU realtime)	8874
Machine Learning	mobile E robot E ("neural network" OU "neural networks" OU "machine learning" OU "deep learning")	5768
Visão computacional	mobile E robot E ("computer vision" OR cv OR "object recognition")	4207
Computação em Nuvem	mobile E robot E (cloud OU server)	2864
Modularidade	mobile E robot E modular	1438

Fonte: Modificado (JAHN et al., 2020).

Além disso, destaca-se especificamente a cooperação multi-robôs. A Figura 2.2 mostra que desde 2016 a quantidade de resultados vem crescendo, chegando a quase dobrar em um intervalo de dois anos, enquanto que antes de 2016, tinha-se quantidade em torno de 500 por ano.

Figura 2.2: Ano x Resultados, para multi-robôs.

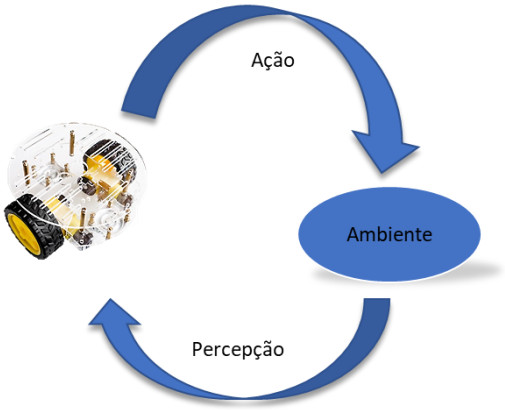


Fonte: Modificado (JAHN et al., 2020).

Um aspecto importante em um robô móvel são suas habilidades de navegação. O objetivo é que o robô se mova facilmente de um local para outro em um ambiente conhecido ou desconhecido (RUBIO; VALERO; LLOPIS-ALBERT, 2019). Para isso, o robô precisa de sensores para perceber o ambiente e atuadores para reagir aos comandos referentes a essa percepção, como mostrado na Figura 2.3. Nesta perspectiva, segundo RUBIO, VALERO e LLOPIS-ALBERT (2019); e ALATISE, HANCKE (2020) os sensores mais comuns usados na robótica são:

- Torque de força: para saber qual força o robô está aplicando;
- *Encoder*s: para saber a velocidade do robô;
- Infravermelhos: são sensores baseados em luz, usados para medir distância ou detectar objetos;
- Visão: câmeras, usadas para capturar imagens;
- Ultrassônicos: são sensores baseados em som, usados para medir distância ou detectar objetos;
- IMU: combinação de acelerômetros, giroscópios e às vezes magnetômetros. Sendo que o acelerômetro é usado para medir a aceleração, o giroscópio para medir as velocidades angulares e a orientação do robô e o magnetômetros para estimar posição e/ou orientação de um objeto relativo a um ponto de partida.

Figura 2.3: Navegação de um robô móvel autônomo.



Fonte: Autor.

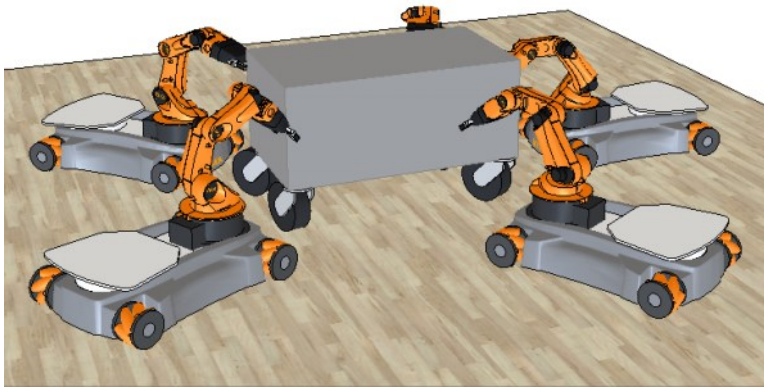
Um robô móvel também pode ser analisado quanto a sua dinâmica de controle, como para movimentar de um ponto para outro. Para tal, pode-se considerar três etapas: primeiro o sistema de percepção fornece informações sobre o ambiente, o próprio robô e a relação entre eles a partir dos dados fornecidos pelos sensores. Posteriormente, essas informações são processadas e o nível cognitivo (código) determina qual ação deve ser executada, para então,

os comandos apropriados serem enviados aos atuadores, que movimentam a estrutura mecânica (RUBIO; VALERO; LLOPIS-ALBERT, 2019).

2.1.2 Multi-robôs

Os pesquisadores vêm sendo motivados a projetar e construir equipes de robôs com capacidade cooperativa (DARMANIN; BUGEJA, 2017). Neste aspecto, os Sistemas Multi-Robôs (*Multi Robot Systems* - MRS) usam a ideia de que muitos robôs podem cooperar para completar uma determinada tarefa mais rápido do que um (NEDJAH; JUNIOR, 2019). MRS são aplicados em vários domínios diferentes, como: vigilância, busca e resgate, exploração, manipulação cooperativa e transporte de objetos, entre outros (DARMANIN; BUGEJA, 2017). Por exemplo, PETITTI et al. (2016) propõem a manipulação de carga em um avião por uma equipe de robôs (Figura 2.4). Além disso, outros exemplos podem ser verificados nos trabalhos de ALONSO-MORA (2017) e VERMA (2019).

Figura 2.4: MRS cooperativo, usado para carregar uma carga.



Fonte: (PETITTI et al., 2016).

Pode-se classificar um sistema MRS baseando-se no número de agentes que compõem o sistema. Neste sentido, sistemas de unidades múltiplas referem-se a equipes com vários membros, onde o número de unidades é relativamente pequeno em relação à área/tarefa. Além disso, outra possível classificação é enxames, nesta tem-se um número muito grande de membros, normalmente quase idênticos e com capacidades limitadas (GARZÓN et al., 2017). Ademais, também, classifica-se MRS quanto à morfologia, sendo (GARZÓN et al., 2017):

- Idênticos: os membros são homogêneos, tanto em mecânica quanto em capacidades. Isso significa que eles podem ser substituídos por qualquer outro membro da equipe;
- Homogêneo: os membros têm morfologia ou capacidades semelhantes, mas não exatamente as mesmas. Isso significa que os membros da equipe podem realizar a mesma tarefa até certo ponto, já que alguns dos recursos são exclusivos;

- Heterogêneos: os membros são diferentes entre si, tanto em recursos quanto em estrutura física.

A comunicação de um robô executa uma importante função, ela permite tarefas cooperativas entre robôs, bem como monitoramento e detecção remotos (RIBEIRO; LOPES 2020). Nesse sentido, é comum se perguntar qual comunicação deve ser usada em um projeto robótico. O trabalho de GARZÓN et al. (2017) tenta responder essa questão, onde Wi-Fi se mostrou a solução mais flexível, devido às suas características e custo.

2.2 ARDUINO

O Arduino trabalha com dois conceitos, as placas e o ambiente de desenvolvimento integrado (*Integrated Development Environment* - IDE). As placas Arduino possuem baixo custo e são comumente usadas como ferramenta educativa, elas são equipadas com entradas e saídas digitais e analógicas, que podem ser conectadas a diferentes placas de expansão, também conhecidas como *shields* (HU, 2019). Além disso, tem-se o Arduino IDE, um *software* mundialmente conhecido, usado para codificar, compilar e fazer *upload* de código em placas Arduino. O Arduino IDE possui suporte para muitas bibliotecas e uma comunidade ativa, facilitando implementações. Além disso, ele pode ser executado em Mac OS X, Windows e Linux. Na robótica, o Arduino é amplamente usado na construção e teste de robôs (HU, 2019; SINGH; ROHILLA, 2020). Essa placa pode funcionar como um computador de baixo nível, recebendo dados de sensores, controlando motores ou executando algoritmos de controle (DE SOUSA et al., 2018; VAUTIER et al., 2018).

Em seu site, a Arduino disponibiliza alguns modelos de placas, cada uma com suas concepções. A Tabela 2.2 possibilita a comparação entre as versões Arduino Due, Nano, Uno Rev3 e Mega, sendo a primeira e a última recomendada para projetos mais complexos e as demais para projetos básicos, que não necessitam de muitos recursos (Arduino, 2021). Isto é notório ao analisar a Tabela 2.2, nela o Nano e o Uno possuem características semelhantes, já o Due se destaca por possuir mais memória flash, SRAM e velocidade. Ainda, comparativamente, o Due possui custo benefício melhor que o Mega, já que ambos possuem mesmo valor, mas o Mega dispõe de menos recursos. Ademais, ao utilizar o Due é importante se atentar quanto à tensão de operação, 3.3 V, já que alguns sensores disponíveis no mercado operam em 5 V.

Tabela 2.2: Característica de alguns Arduino.

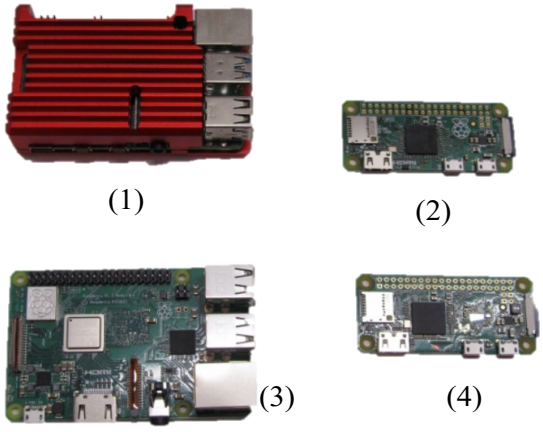
Características	Arduino Due	Arduino Nano	Arduino Uno Rev3	Arduino Mega
Preço [R$]	238,98	122,75	136,39	238,98
Tensão de operação [V]	3.3	5	5	5
Pinos I/O digital	54 (12 PWM)	22 (6 PWM)	14 (6 PWM)	54 (15 PWM)
Pinos analógicos	12	8	6	16
SRAM [kB]	96	2	2	8
Flash [kB]	512	32	32	256
Velocidade [MHz]	84	16	16	16

Fonte: Modificado (ARDUINO, 2022a).

2.3 RASPBERRY

Raspberry é um mini computador de baixo custo, bastante usado como meio educativo. Atualmente o Raspberry está na quarta geração, Raspberry Pi 4. Desde o primeiro lançamento, em 2012, foram anunciados diversos modelos e versões, têm-se os da Figura 2.5 como exemplo (KURNIAWAN, 2018; RASPBERRY PI, 2021). Muitos recursos de computadores "comuns" estão inclusos no Raspberry, como: saída de áudio, Wi-Fi, chip gráfico, ethernet, portas Universal Serial Bus (USB), para teclado e *mouse*, por exemplo, e conector HDMI (EHRMANN; EHRMANN, 2020). Além disso, os Raspberries possuem pinos *General Purpose Input/Output* (GPIO), que são portas programáveis de entrada e saída de dados, utilizadas como interface entre periféricos e o microprocessador (VAIDYA; RUGHANI, 2020).

Figura 2.5: Modelos Raspberry Pi. (1) Raspberry Pi 4B. (2): Raspberry Pi Zero. (3): Raspberry Pi 3B+. (4): Raspberry Pi Zero W.



Fonte: Modificado (EHRMANN; EHRMANN, 2020).

Ao logo do tempo ocorreram várias melhorias envolvendo, principalmente, o processador, a quantidade de memória e a conectividade (GUSE, 2020), a Tabela 2.3 traz mais detalhes quanto às características de certos modelos. Desses, ressalta-se o novo Raspberry Pi 4, que pode chegar a 8 Gb de memória RAM. Ademais, destaca-se que os preços dispostos nessa tabela podem sofrer ajustes, principalmente em relação a cotação do Euro e a custos de importações, que não foram considerados no cálculo do valor especificado.

Tabela 2.3: Características de alguns modelos de Raspberry Pi.

Características	**Raspberry Pi 4B**	**Raspberry Pi 3B+**	**Raspberry Pi Zero W**
Preço [R$]	225,34 – 462.54	219,41	65,23
Clock CPU [GHz]	1.5	1.4	1
RAM [GB]	1-8 LPDDR4	1 DDR2	0.512
USB 3.0	2	-	-
USB 2.0	2	4	-
Vídeo	2 micro HDMI	HDMI	mini HDMI
Áudio	3.5 mm jack	3.5 mm jack	-
Wi-Fi	802.11 b/g/n/ac	802.11 b/g/n/ac	802.11 n
Bluetooth	5.0	4.2 BLE	4.1
Potência	1.25 A @ 5 V	1.13 A @ 5 V	180 mA @ 5 V

Fonte: Modificado (EHRMANN; EHRMANN, 2020).

2.3.1 LINUX

Linux é um Sistema Operacional (SO) de código aberto criado por Linus Torvalds (PALAKOLLU, 2021). Esse sistema é usado como base para distribuições populares, como Debian, Android, Fedora e Ubuntu. O Android é o sistema mais utilizado em *smartphones*, por isso, pode-se dizer que o Linux tem a maior base instalada de todos os sistemas operacionais de uso geral. Ademais, o Linux é um SO com destaque em aplicações relacionadas a servidores e sistemas embarcados. Os principais diferenciais do Linux são estabilidade, performance e segurança, dado que o código é constantemente revisado por um grande número de desenvolvedores (UFSC, 2000).

Além disso, o Ubuntu, SO construído a partir do núcleo Linux, tem sido a principal plataforma para ROS. No qual, para uma dada versão de ROS tem-se uma determinada versão do Ubuntu, como por exemplo, para ROS 2 *Foxy Fitzroy* utilizada neste trabalho, onde a versão recomendada é o Ubuntu Focal Fossa (20.04) 64-bit.

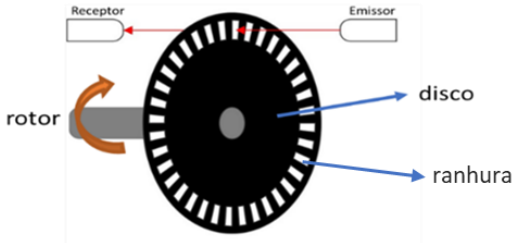
2.4 ENCODER

Odometria é o uso de dados de sensores de movimento para estimar alteração de posição. A odometria a partir de *encoders* é muito utilizada, já que fornece boa precisão, possui baixo custo financeiro e permite altas taxas de amostragem (SALAROLLI; MATTA; CUADROS, 2017). No entanto, métodos utilizando *encoder*s podem ser afetados pelo deslizamento da roda e a rugosidade do chão, que produzem erros (ROMADON et al., 2019).

Os *encoder*s ópticos são um tipo comum de sensor na robótica e podem ser classificados de acordo com o tipo de funcionamento, em incremental ou absoluto. O *encoder* incremental possui um emissor e receptor de luz, Figura 2.6 (a). Para que ele funcione, tem-se um disco com ranhuras acoplado ao eixo de rotação do rotor, como o da Figura 2.6 (b). Esse disco gira junto com o rotor. Quando o motor está funcionando, o disco rotaciona e o feixe de luz é recebido se há uma abertura e interrompido caso haja um obstáculo, a parte preta do disco. Sendo que, quando a luz é recebida um pulso elétrico é gerado. Logo, sabendo-se a quantidade total de ranhuras do disco e a quantidade de pulsos gerados pelo *encoder* em um determinado intervalo de tempo, é possível estimar a velocidade de rotação da roda de um robô (SANTOS, 2020; TAUFIQQUROHMAN; SARI, 2018). Por exemplo, uma contagem de pulso igual a 50 durante um segundo com um disco de 20 ranhuras, indica que a roda efetuou duas voltas e meia em um segundo.

Figura 2.6: *Encoder* incremental.

(a) Disco e o encoder.



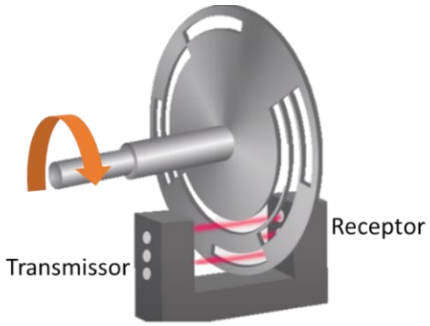
Fonte: Modificado (ELETRICAMENTE FALANDO, 2011).

(b) Disco acoplado ao rotor.

Fonte: Autor.

Já o *encoder* absoluto, Figura 2.7, fornece a leitura do sentido de rotação e velocidade do rotor; para tal, ele pode apresentar uma combinação exclusiva de ranhuras, as quais estão dispostas de forma que na saída tenha-se um código binário para cada posição angular (SHIBUKAWA, 2018).

Figura 2.7: *Encoder* absoluto.



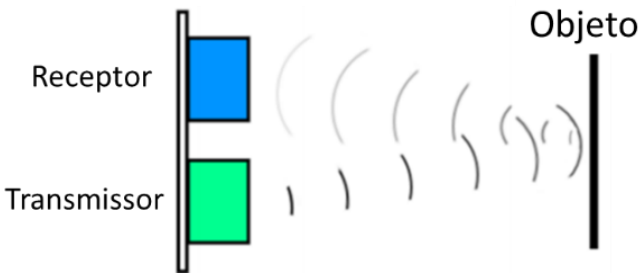
Fonte: Modificado (REALPARS, 2019).

2.5 SENSORES DE OBSTÁCULO

Navegação e prevenção de obstáculos são problemas fundamentais da robótica móvel. É comum encontrar soluções onde um robô é equipado com sensores que possibilitem um movimento autônomo (ou semiautônomo), tais como sensores: ultrassônicos, infravermelho, câmera, entre outros (PANDEY; PANDEY; DR, 2017).

HC-SR04 é um módulo ultrassônico de baixo custo usado para detectar obstáculos ou medir distâncias entre objetos. O módulo, Figura 2.8, possui um transmissor e receptor, onde o transmissor emite ondas sonoras em frequências de 40 kHz. Quando as ondas atingem um objeto elas são refletidas e posteriormente captadas pelo receptor. Então, a partir do tempo entre transmissão e recepção é possível obter informação de distância entre o objeto e o módulo (KAMAL; HEMEL; AHMAD, 2019; ZHMUD et al., 2018).

Figura 2.8: HC-SR04.

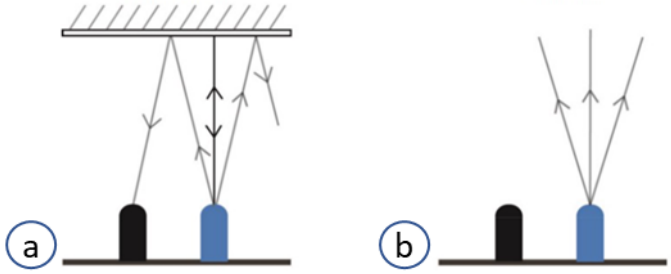


Fonte: (ZHMUD et al., 2018).

O sensor IR é outra forma utilizada para detectar obstáculos. Ele apresenta lógica de funcionamento similar ao ultrassônico, com transmissor e receptor, só que ao invés de ondas de

som, utiliza luz infravermelha. A Figura 2.9 (a), mostra o funcionamento padrão, onde a luz que incide sobre uma superfície lisa e brilhante é refletida e captada pelo receptor. Diferente do sensor ultrassónico, o sensor IR pode não detectar um objeto a sua frente se este tiver uma cor preta (Figura 2.9 (b)), isso acontece porque a onda infravermelha é absorvida pelo objeto (MANZOOR, 2020).

Figura 2.9: Sensor IR. Incidência sobre (a) superfície lisa e brilhante. (b) superfície preta.

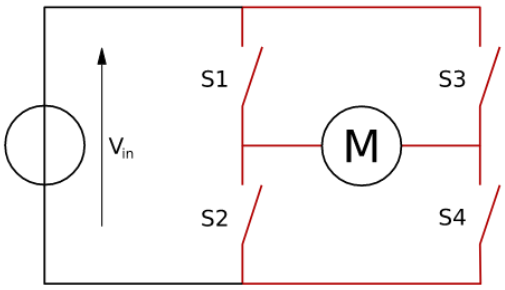


Fonte: (MANZOOR, 2020).

2.6 PONTE H

A ponte H é um conceito utilizado para inverter a polaridade de uma carga, sendo comumente utilizada para girar motores DC em sentido horário ou anti-horário. A Figura 2.10 mostra um circuito ponte H com chaves S1, S2, S3 e S4. Onde o sentido de rotação do motor depende da disposição das chaves, sendo que, quando as chaves S1 e S4 estão fechadas o motor roda em um sentido, em contrapartida, quando as chaves S3 e S2 estão fechadas o motor gira em outro sentido (ALMEIDA, 2014).

Figura 2.10: Ponte H com chaves.



Fonte: (ALMEIDA, 2014)

2.7 GITHUB E GIT

GitHub é uma plataforma de hospedagem de código e arquivos, desde seu lançamento, uma ampla quantidade de usuários aderiu a plataforma, incluindo empresas, estudantes, universidades e agências governamentais (CHENG et al., 2020).

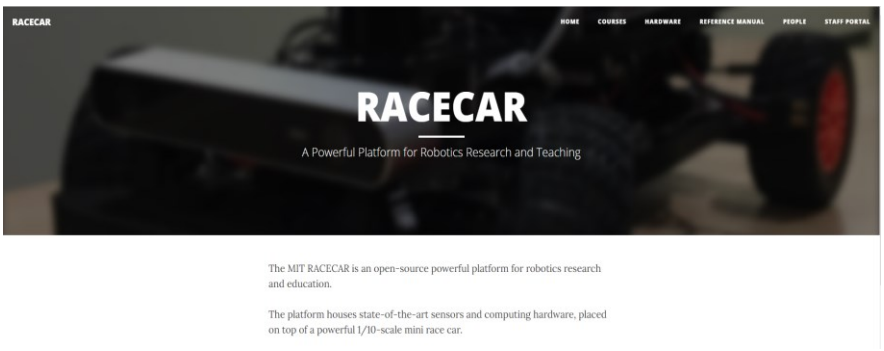
O GitHub é modelado no sistema de controle de versão Git. Quando o Git é inicializado dentro de um diretório, o mesmo se torna um repositório Git. Feito isso, o programador pode registrar as alterações realizadas em um projeto, organizando versões anteriores, de forma que, quando acontece algum erro, pode-se recuperar versões passadas. Em suma, a diferença entre Git e GitHub é que o Git é um sistema de controle de versão e o GitHub é um repositório de *backup* gratuito na internet, que fornece ferramentas de colaboração e gerenciamento de projeto (THORNBERRY; WHITE, 2020).

Muitos projetos *open source* são disponibilizados no GitHub, inclusive códigos baseados em ROS (ROS, 201X; ZHANG et al., 2017). Particularmente, na robótica, existem várias tecnologias hospedadas no GitHub, como: Racecar (KARAMAN, 2017); código SDK Spot (BOSTON DYNAMICS, 2020); Gazebo; biblioteca kilolib (ACORNEJO, 2013), para compilar programas para kilobots; OpenBot (INTEL ISL, 2020); Webots; e JPL (*Open Source Rover*) (NASA, 2018), o que torna o GitHub uma importante ferramenta da área.

Um dos serviços oferecidos pelo GitHub é o GitHub Pages, que possibilita a hospedar um site estático de forma gratuita. Esse serviço, pega arquivos HTML, CSS e Java Script diretamente de um repositório no GitHub para montar o site. Tem-se a Figura 2.11 como exemplo (GITHUB, 2021; UTOMO; FALAHAH, 2020). No entanto, o GitHub Pages tem algumas limitações, sendo elas (GITHUB, 2021):

- Não oferece suporte a linguagens do lado do servidor, como PHP, Ruby ou Python;
- Não se destina a uso comercial, como para site de comércio eletrônico;
- Não pode ter mais de 1 GB;
- Têm um limite de *soft bandwidth* de 100 GB por mês;
- Têm limite de 10 compilações por hora.

Figura 2.11: GitHub Pages do MIT RACECAR.



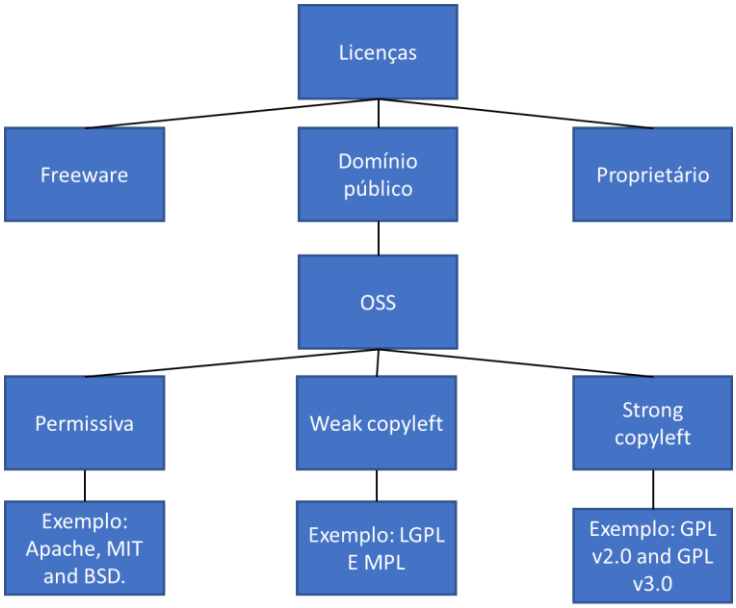
Fonte: (KARAMAN et al., 2017).

2.8 LICENÇAS *OPEN SOURCE*

Uma licença de *software* é um instrumento legal que autoriza ou restringe o uso de um *software*. A licença é definida pelos desenvolvedores, sendo importante para proteger os autores originais e definir parâmetros de utilização para terceiros.

O Scicobot possui pretensões *open source*, dentre as categorias de licença de código, tem-se as licenças de *software* de código aberto (*Open Source Software* - OSS), que garantem ao usuário o direito de usar, modificar e redistribuir o código-fonte em sua forma original ou modificada, o que condiz com os objetivos do robô. As OSS podem ser divididas em três categorias: permissivas, *weak copyleft* e *strong copyleft*, como apresentado pela Figura 2.12 (ALAMOUDI, 2019).

Figura 2.12: Tipos de licenças.



Fonte: Modificado (ALAMOUDI, 2019).

As licenças permissivas são a categoria mais liberal, não há restrições para devolver as modificações à comunidade, apenas a exigência de que os licenciados deem crédito aos contribuidores originais. Ou seja, as pessoas podem bifurcar o código-fonte e liberá-lo sob uma licença proprietária. Os exemplos mais comuns desta categoria são o Apache e MIT (ALAMOUDI, 2019).

Já as licenças do tipo *weak copyleft* aplicam-se apenas ao trabalho original, ou seja, obras licenciadas sob licenças *copyleft* fracas podem ser incluídas em projetos com licença diferente. Um efeito disso é a possibilidade de vincular bibliotecas com *weak copyleft* a projetos proprietários. Alguns exemplos desse tipo de licença são a Mozilla Public License (MPL) e a GNU Lesser General Public License (LGPL) (ALAMOUDI, 2019).

Por outro lado, as licenças *strong copyleft*s possuem uma natureza viral, que exige que quando um *software* contém parte de código *strong copyleft*, o *software* como um todo deve ser distribuído sobre a mesma licença *strong copyleft*.

Em geral, uma licença pode ser alternada para uma licença mais restritiva, mas não para uma mais permissiva. Além disso, apenas o tipo de licença permissiva pode ser modificado para proprietário (ALAMOUDI, 2019).

Algumas das licenças mais usadas são (ALAMOUDI, 2019):

- MIT: é uma licença permissiva, permite que as pessoas façam qualquer coisa com o código com atribuições próprias e sem garantias (ESCOLHAUMALICENCA, 2022);
- Licença Apache 2.0: também é uma licença permissiva, no entanto, inclui cláusulas de rescisão e indenização de patentes (GNU, 2022).
- *GNU General Public License* (GPL) 3.0: a principal característica desta licença é a imposição de que quaisquer trabalhos derivados ou modificações sejam publicados sob as mesmas licenças GPL, garantindo que o projeto nunca seja convertido em um produto proprietário. De acordo com a Black Duck KnowledgeBase, cerca de 30% de todos as OSS são lançadas sob diferentes versões GPL (ALAMOUDI, 2019)
- *GNU Affero General Public License* v3 (AGPL): é semelhante à GPL v3, com um parágrafo adicional para permitir que os usuários que interagem com o *software* licenciado em uma rede recebam a fonte desse programa (GNU, 2022).

2.9 ROBOT OPERATING SYSTEM

O ROS é um *framework* de *software* robótico que se tornou uma interface padrão para robôs no mundo acadêmico. Dentre as principais características do ROS tem-se: integração de robôs às redes; comunidade de usuários grande e ativa; e facilidade de integração com visão computacional, ferramentas de modelagem e simulação 3D, como: OpenCV, RViz, Gazebo e Matlab (LOPEZ-RODRIGUEZ; CUESTA, 2021). A ideia de construir ROS veio quando se percebeu que inúmeras pessoas que pretendiam inovar em *softwares* robóticos gastaram cerca de 90% do tempo reescrevendo códigos que outros já haviam escrito antes e construindo um protótipo para teste. Ou seja, apenas 10% do tempo eram gastos efetivamente em inovação (WYROBEK, 2017).

ROS pode ser complicado de entender quanto ao seu uso e importância, portanto, considere uma situação hipotética onde um desenvolvedor quer trabalhar no desenvolvimento de *software* de um robô, neste caso, há algumas alternativas: o desenvolvedor pode desenvolver

uma lógica nova, de sua autoria, ou seguir a estrutura (*framework*) de um projeto já existe. Porém, na criação se gasta muito tempo tentando implementar elementos e lógicas de comportamento (como uma maneira de realizar a comunicação) e para o aproveitamento de projetos existentes tem-se soluções que usam uma lógica diferente, prevista para uma aplicação em particular, com pouca documentação e padrão complicado de entender. Ainda, se cada um desenvolve uma estrutura para validar alguma implementação, sem haver reaproveitamento, as pesquisas ficam ineficientes, atrasando o desenvolvimento da robótica. Nesta perspectiva, ROS vem como proposta de solução. Ele permite que os desenvolvedores se concentrem em sua aplicação, usando uma base existente em vez de tentarem reinventar recursos a cada nova pesquisa. Além disso, ROS pode ser usado em qualquer robô, seja ele um robô móvel, braço robótico, drone ou barco (ROBOTICS BACK-END, 201X). Em suma, tem-se os seguintes benefícios ao utilizar ROS:

- Todos os robôs do sistema conversando em um mesmo padrão;
- Facilidade para reaproveitar código;
- Possui fácil de integrar com algumas ferramentas importantes para a robótica, como o Gazebo;
- Facilidade na compreensão de projetos, já que utiliza uma lógica relativamente simples, de assinante e editor, e bem documentada;
- Facilidade de comparação entre projetos a nível de comunicação.

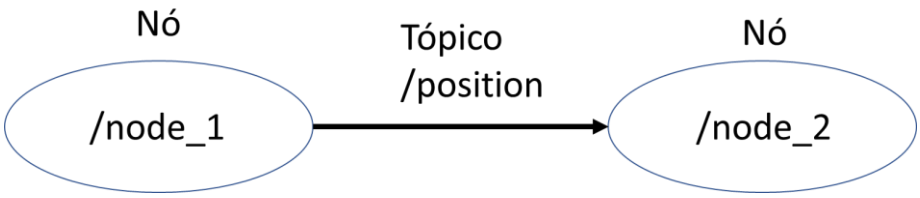
ROS apresenta abstração de *hardware*, controle de dispositivo de baixo nível, implementação de funcionalidades comuns, troca de mensagens entre processos e gerenciamento de pacotes. Ao comparar sistemas operacionais com ROS, pode-se dizer que o Ubuntu, por exemplo, se difere de ROS por ser completo, com alto grau de independência (ROBERT, 2020; YU et al., 2017).

Em termos gerais, ROS proporciona uma estrutura de comunicação de alto nível, que fornece um padrão organizado, fácil e relativamente confiável de troca de informação baseado em assinante, mensagem e publicador, semelhante ao MQTT. As definições para os três elementos citados anteriormente são apresentadas a seguir (ROS, 201X):

- Nós (*Nodes*): um executável que usa ROS para se comunicar com outros nós. Os nós também podem fornecer ou usar um serviço;
- Mensagens (*Messages*): tipo de dados ROS usado ao assinar ou publicar um tópico;
- Tópicos (*Topics*): os nós podem publicar mensagens em um tópico, bem como se inscrever em um tópico para receber mensagens.

Um exemplo simples da arquitetura ROS é mostrado na Figura 2.13, onde há dois nós (node_1 e node_2) e um tópico (position), nesse sistema, o node_1 publica e o node_2 está inscrito no tópico position, dessa forma, quando node_1 publica uma mensagem em position, é como se o node_1 estivesse mandando uma mensagem para node_2.

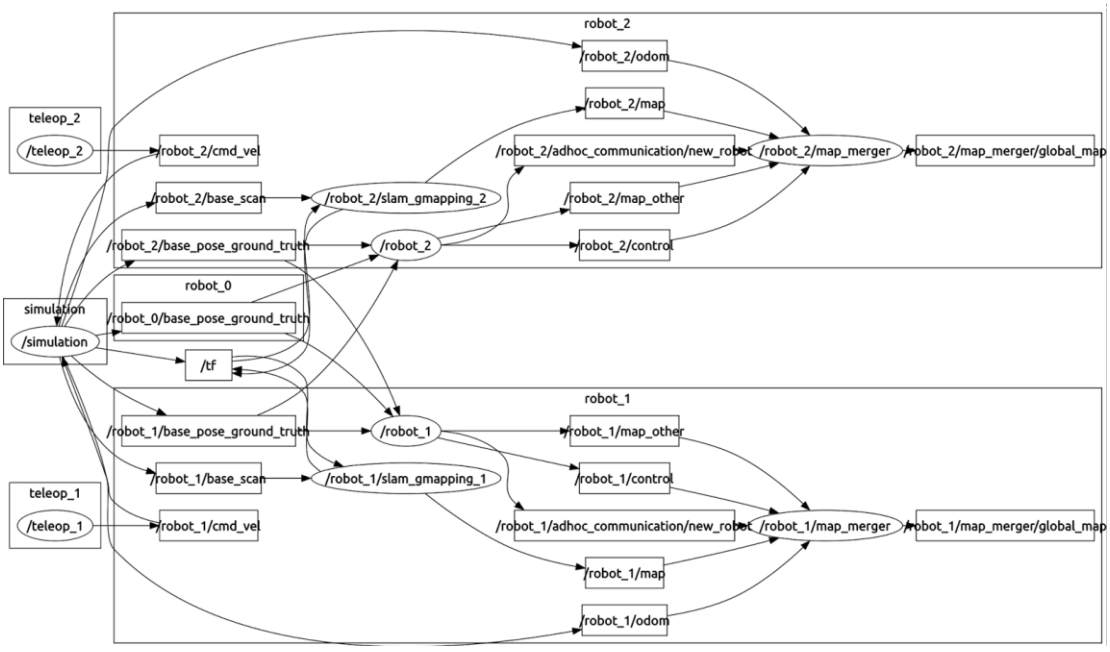
Figura 2.13: Comunicação entre dois nós.



Fonte: Modificado (MATHWORKS, 2021).

Conforme um projeto cresce e de acordo com a organização adotada, têm-se estruturas mais ou menos complexas. Nessa perspectiva a Figura 2.14 demostra uma estrutura mais robusta, com vários nós e tópicos, onde dois robôs estão publicando mapas individuais usando "slam_gmapping". Os mapas produzidos são, então, mesclados no nó "map_merger" e posteriormente disponibilizados no RVIZ (SKOROSPELOV, 2015).

Figura 2.14: Possível configuração de nós de um projeto.

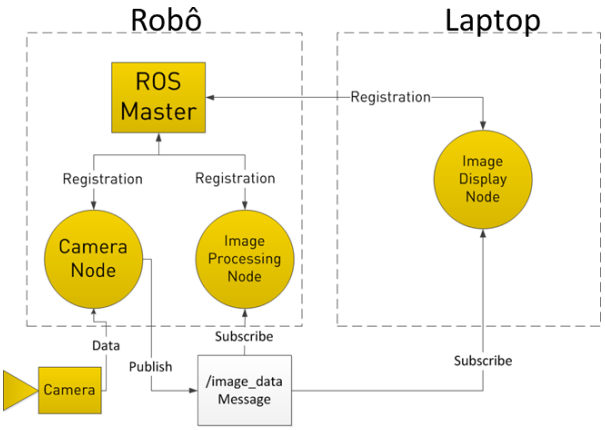


Fonte: (SKOROSPELOV, 2015).

Na prática, para o correto funcionamento de ROS tem-se que executar um nó mestre (master), que cuida dos serviços de nomenclatura e registro para que outros nós ROS possam se encontrar e se comunicar na rede (ERŐS et al., 2019). A título de exemplo, na Figura 2.15 tem-se três nós: um nó de câmera, que pega as informações da câmera (como vídeo ou imagem);

um nó de processamento de imagem, que processa dados de imagem no robô; e um nó de exibição de imagem, que exibe imagens em uma tela de computador. Todos os nós foram registrados a partir do mestre, ele é como um banco de dados, onde todos os nós vão para saber para onde enviar mensagens (CLEARPATH ROBOTICS, 2015).

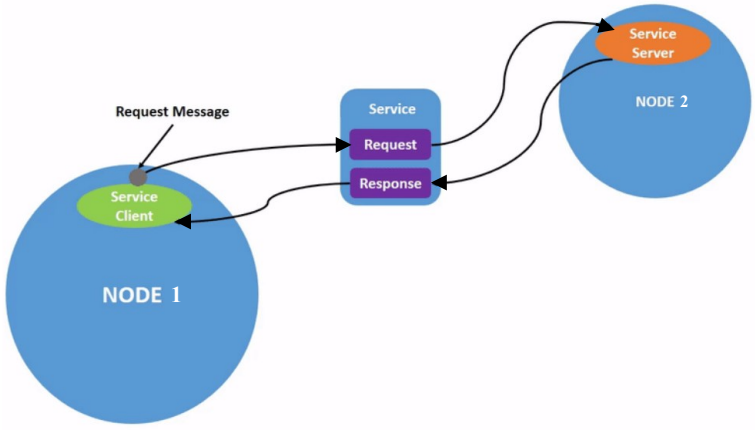
Figura 2.15: Nós de um projeto se comunicando com o mestre.



Fonte: Modificado (CLEARPATH ROBOTICS, 2015).

Os serviços são outra forma de comunicação entre nós ROS. Diferente dos tópicos, os serviços permitem que os nós enviem uma solicitação e recebam uma resposta, ou seja, enquanto os tópicos são unilaterais os serviços propiciam interações (NOORI et al., 2017; ROS, 2020). Dois nós são mostrados na Figura 2.16, onde o serviço se dá de *service client* para o *service server* (solicitação) e depois de server para *client* (resposta).

Figura 2.16: Serviço ROS.



Fonte: Modificado (ROS2, 2022a).

O lançamento do ROS é programado para todo mês de maio, com suporte de dois a cinco anos (JALIL, 2018). A seguir, na Tabela 2.4, são mostradas algumas distribuições ROS.

Tabela 2.4: Distribuições ROS.

Distribuição	Ano de Lançamento	Suporte
Melodic Morenia	2018	2023
Lunar Lorgerhead	2017	2019
Kinect Kame	2016	2021
Jade Turtle	2015	2017
Indigo Igloo	2014	2019
Hydro Medusa	2013	2015
Groovy Galapagos	2012	2014
Fuerte Turtle	2012	-
Eletric Emys	2011	-
Diamondback	2011	-
C Turtle	2010	-
Box Turtle	2010	-

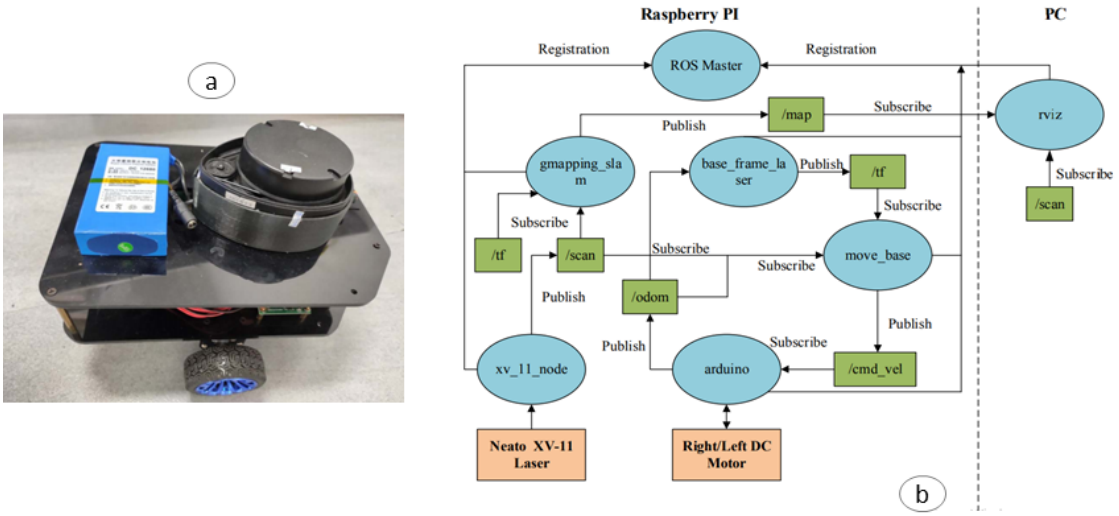
Fonte: Modificado (JALIL, 2018).

2.9.1 Exemplos de Trabalhos Baseados em ROS

LIAO (2019) apresenta um robô, Figura 2.17 (a), para SLAM. O sistema executa ROS dentro de uma placa Raspberry Pi 3B e possui controlador Arduino Mega2560, *laser* Neato XV-11 e dois motores DC com *encoder*s integrados. A arquitetura ROS utilizada está disposta na Figura 2.17 (b), os nós são:

- move_base: esse nó assina os tópicos /tf e /scan e publica em /cmd_vel, para controle de movimento conforme o caminho planejado;
- arduino: recebe os comandos de velocidade por /cmd_vel para efetuar o controle dos motores e publica os dados de odometria, como do *encoder*, em /odom;
- base_frame_laser: recebe os dados de odometria através de /odom. Os dados recebidos são transformados para, então, serem publicados no tópico /tf;
- xv_11_node: os novos dados do laser são processados e publicado em /scan;
- gmapping_slam: este nó assina /tf e /scan para receber dados de odometria e do laser. Esses dados são publicados em /map e usados para corrigir a posição do robô no mapa;
- rviz: posição do robô e o mapa construído são exibidos no PC dinamicamente via RViz, uma ferramenta gráfica do ROS.

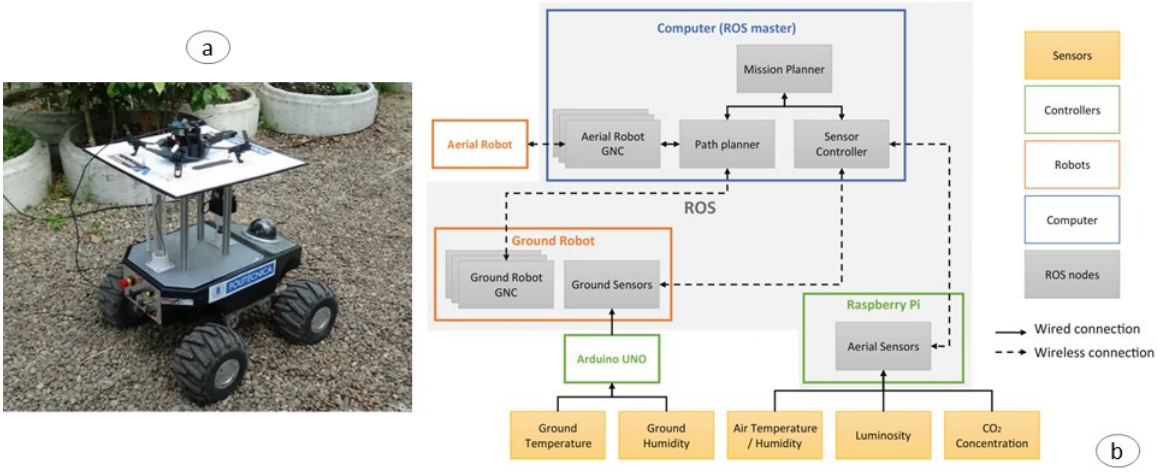
Figura 2.17: Robô para prática SLAM. (a) Robô proposto. (b) Arquitetura ROS, em azul os nós e em verde os tópicos.



Fonte: (LIAO, 2019).

Um sistema heterogêneo de monitoramento para estufas é proposto por ROLDÁN (2016), onde um robô aéreo e outro terrestre, Figura 2.18 (a), são usados para medir propriedades do solo e do ar. No UAV tem-se um Raspberry Pi com ROS *Hydro* e em *Unmanned Ground Vehicle* (UGV) um Arduino UNO e Summit XL, também com ROS. É usada uma rede *Wireless Local Area Network* (WLAN) para conectar todos os robôs e dispositivos, enquanto um computador é usado para coletar, processar e armazenar os dados ambientais (GARZÓN, 2017). Na Figura 2.18 (b) é mostrada a arquitetura de *software* implementada para a execução das tarefas, destacam-se os nós ROS (ROS *nodes*) definidos, que estabelecem a forma de comunicação entre computador-Raspberry Pi e computador-robô terrestre.

Figura 2.18: Sistema para monitoramento de estufas. (a) UGV e UAV implementados. (b) Arquitetura de *software* utilizada.



Fonte: (GARZÓN, 2017; ROLDÁN, 2016).

2.9.2 Robot Operating System 2

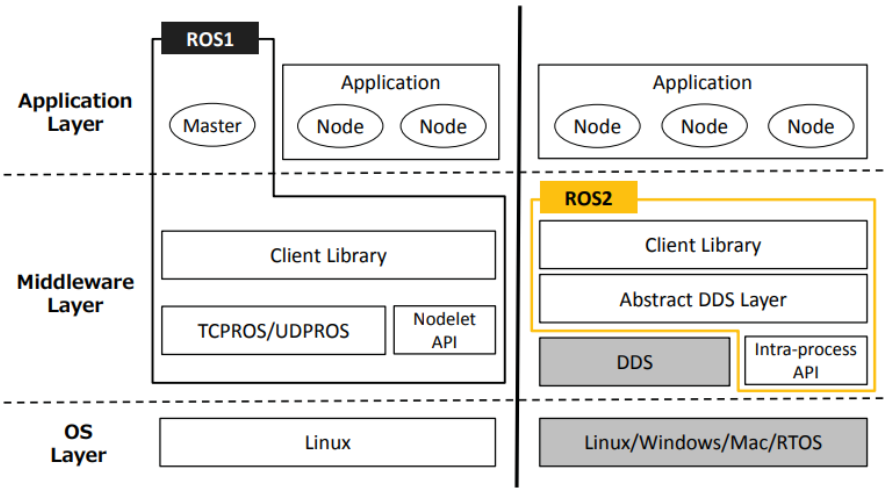
Desde o lançamento de ROS, em 2007, muita coisa mudou na comunidade de robótica. A fim de se adaptar a essas mudanças foi lançado o ROS 2, sendo ROS *Noetic* a última versão de ROS (também conhecido como ROS 1, termo usado para evitar eventuais confusões entre as versões).

ROS 2 ainda utiliza o mecanismo de publicação e assinatura (nó, tópico e mensagem), mas com middleware DDS/RTPS, que fornece descoberta não centralizada, serialização e transporte. Sendo *Data Distribution Service* (DDS) um padrão da indústria, amplamente utilizado para sistemas distribuídos em tempo real, e *Real-Time Publish Subscribe* (RTPS) um protocolo usado pelo DDS para se comunicar na rede (KRONAUER, 2021; ROS 2, 2021). Diferentemente de ROS, em ROS 2 (THE ROBOTICS BACK-END, 2021):

- Pode-se usar *cpp* versão 11 e 14 por padrão;
- Não é necessário iniciar um nó mestre. Cada nó tem a capacidade de descobrir outros nós;
- Têm a capacidade de escolher entre as opções de políticas de qualidade de serviço (*Quality of Service* – QoS), o que permite modificar ainda mais a camada de transferência;
- Pode ser usado no Ubuntu, Mac OS e Windows 10;
- Suporta resposta em tempo real (ERŐS et al., 2019). Essa é uma das principais alterações, já que ROS não atendia aos requisitos para operação em tempo real (MARUYAMA; KATO; AZUMI, 2016).

A Figura 2.19 mostra a arquitetura usada por ROS 1 (à esquerda) e por ROS 2 (à direita). Em comparação é possível averiguar algumas das características anteriormente especificadas, sendo uma delas a centralização definida pelo nó mestre (master) para ROS 1 (MARUYAMA; KATO; AZUMI, 2016).

Figura 2.19: Arquitetura ROS 1 e ROS 2.



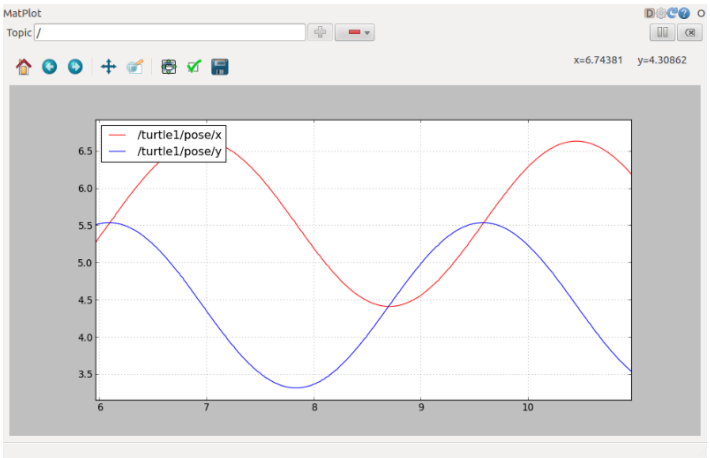
Fonte: (MARUYAMA; KATO; AZUMI, 2016).

2.9.3 Ferramentas

ROS possui algumas ferramentas de simulação e plotagem de dados auxiliares. A seguir são apresentadas breves descrições de algumas dessas ferramentas.

- O rqt_plot é um recurso usado para exibir um gráfico 2D para plotagem de dados dos tópicos (ROS, 2019). A Figura 2.20 mostra a plotagem de dois dados, /turtle1/pose/x e /turtle1/pose/y, a título de exemplo.

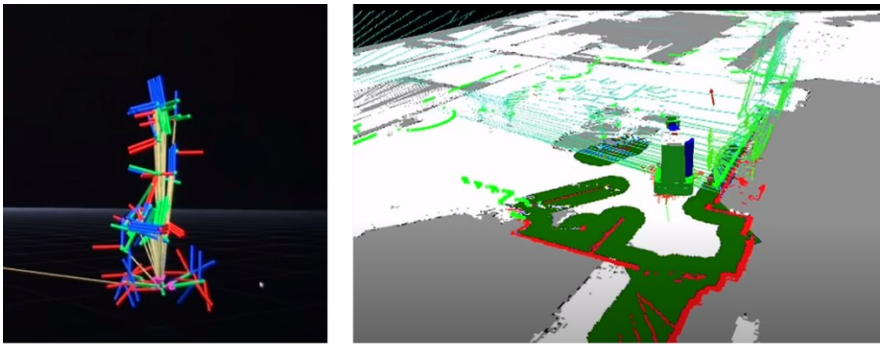
Figura 2.20: Usando rqt_plot para plotar dois dados.



Fonte: (ROS, 2019).

- ROS *Visualization* (RViz) é um visualizador tridimensional para ROS usado para ver o mundo através dos olhos dos robôs (NOORI et al., 2017). Ele permite que o usuário visualize o modelo do robô simulado, Figura 2.21 (a), reproduza as informações dos sensores do robô, como câmeras, lasers, Kinect e outros dispositivos 3D ou 2D, na forma de nuvens de pontos, Figura 2.21 (b), ou imagens (FAIRCHILD; HARMAN, 2016).

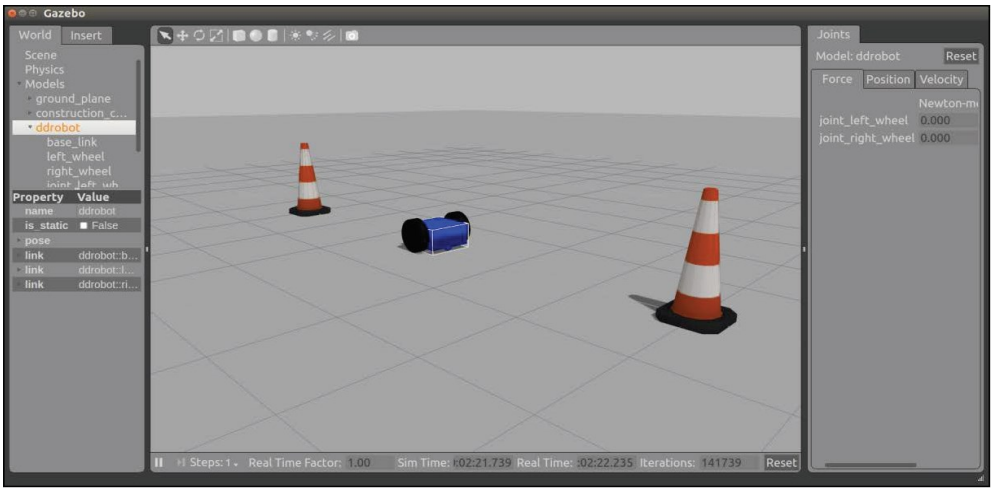
Figura 2.21: RViz: (a) Representação da estrutura de um robô. (b) SLAM.



Fonte: Modificado (WILLOWGARAGEVIDEO, 2010).

- A simulação é uma importante ferramenta para os que trabalham na área da robótica. O Gazebo, Figura 2.22, é um *software* gratuito suportado por Linux e Mac OS X que oferece a capacidade de simular com precisão e eficiência um ou mais robôs em ambientes complexos. Para isso, ele usa mecanismo de física robusto, gráficos de alta qualidade e interfaces gráficas convenientes. Por padrão, o Gazebo utiliza o *Open Dynamics Engine* (ODE) como motor de física para simular o comportamento físico do ambiente e dos robôs (como atrito e massa). No entanto, o Gazebo também suporta *Bullet*, *Simbody* e *Dynamic Animation and Robotics Toolkit* (DART). Atualmente, a *Open Source Robotics Foundation* (OSRF) mantém o Gazebo e o ROS, sendo que os dois são compatíveis através de um conjunto de pacotes, que fornece a interface necessária para simular um robô que utiliza ROS no Gazebo (GAZEBO, 2014; NOORI et al., 2017).

Figura 2.22: Simulando o comportamento de um robô no Gazebo.



Fonte: (FAIRCHILD; HARMAN, 2016).

2.9.4 Bibliotecas Cliente

A biblioteca cliente ROS para C++ (*ROS Client Library for C++* - RCLCPP) e a biblioteca cliente ROS para Python (*ROS Client Library for the Python* - RCLPY) são bibliotecas cliente suportadas oficialmente, elas permitem aos usuários implementar seu código ROS a partir de um núcleo comum, denominado biblioteca cliente (*ROS Client Library* - RCL) (ROS2, 2022a).

Para aqueles que necessitam implementar projetos em alguma biblioteca cliente é recomendável um estudo aprofundado, neste caso, os exemplos de ROS2 (2022a), denominados "*Writing a simple publisher and subscriber (C++)*" e "*Writing a simple publisher and subscriber (Python)*", fornecem um bom auxilio inicial. Esses exemplos, definem uma arquitetura ROS 2 simples, com um editor e assinante, escritos a partir de RCLCPP ou RCLPY, respectivamente. Além disso, é altamente aconselhável o estudo da documentação oficial (RCLCPP, 2022; RCLPY, 2022) junto com os exemplos.

2.9.5 ROS para microcontroladores

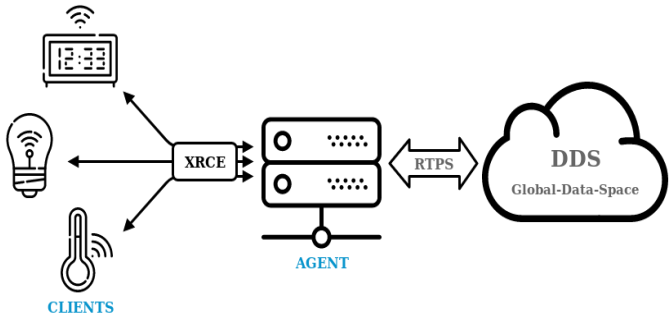
Um microcontrolador é comumente utilizado na robótica para acessar sensores e atuadores. Esses dispositivos possuem recursos limitados, como o espaço da RAM, velocidade e ausência de conexão Wi-Fi. As limitações desses dispositivos impossibilitam a utilização de ROS diretamente nos microcontroladores (LANGE, 2021). Assim sendo, iniciativas foram criadas a fim de viabilizar a utilização de ROS em microcontroladores. Nas subseções seguintes são mostradas algumas formas para isso.

2.9.5.1 Micro-Ros

A micro-ROS é um projeto *open source*, licença Apache, criado para possibilitar a integração entre ROS 2 e placas de microcontroladores, trazendo todos os benefícios de ROS 2 para o desenvolvimento de baixo nível (MICRO-ROS, 2022).

A micro-ROS é baseada no padrão micro XRCE-DDS (MICRO XRCE-DDS, 2018), que implementa um protocolo cliente-servidor para que dispositivos com recursos limitados (clientes) participem das comunicações DDS, como mostra a Figura 2.23. Para isso, o Agente Micro XRCE-DDS (servidor) atua em nome dos clientes.

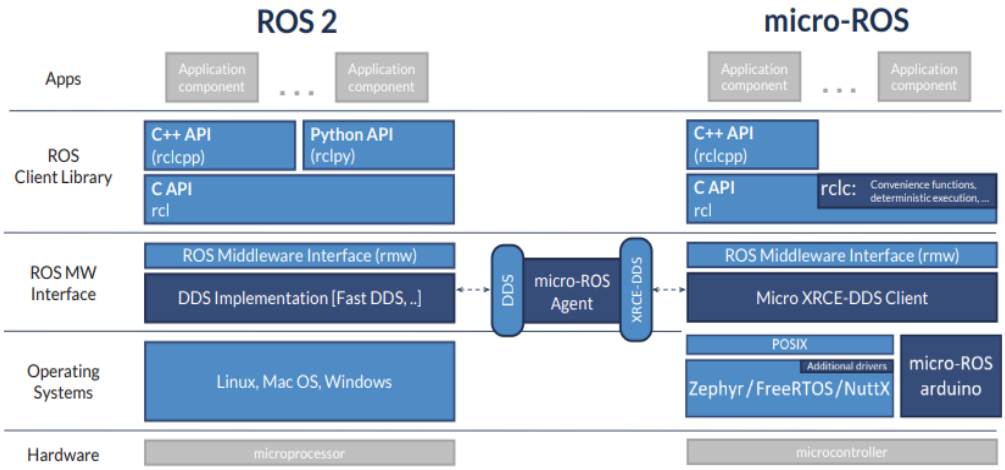
Figura 2.23: eProsima XRCE-DDS.



Fonte: (MICRO-ROS, 2022).

Para atingir seu objetivo, a micro-ROS utiliza estruturas de dados e algoritmos existentes da pilha ROS 2 e traz as alterações necessárias na pilha principal. É possível verificar isso a partir da Figura 2.24, onde a arquitetura geral de um sistema ROS 2 com dois dispositivos, um microcontrolador e um microprocessador, é mostrada. Nesta figura, tem-se a divisão em cinco camadas, desde a de *hardware*, de mais baixo nível, à da aplicação, de maior nível, sendo os componentes azuis escuros implementações especificas da micro-ROS (MICRO XRCE-DDS, 2018).

Figura 2.24: Pilha micro-ROS.



Fonte: modificado (MICRO-ROS, 2022).

Quanto ao uso de memória, XRCE-DDS exige menos que 75 kB de memória Flash e cerca de 3 kB de RAM para um aplicativo completo de editor e assinante que lida com tamanhos de mensagens na ordem de 512 B (MICRO-ROS, 2022). Neste contexto, normalmente, o problema para executar micro-ROS em um microcontrolador são as restrições de memória. Por isso, a micro-ROS busca resolver o problema de gerenciamento de memória priorizando o uso de memória estática. O rmw-microxrcedds, por exemplo, usa memória estática para alocar os recursos associados as entidades RCL e biblioteca cliente ROS para C (*ROS Client Library for C* - RCLC). Neste caso, o tamanho dos *pools* de memória pode ser definido por meio de sinalizadores CMake, como por exemplo RMW_UXRCE_MAX_PUBLISHERS, que define a

quantidade de editores. Assim sendo, restringe-se o número máximo de entidades que um aplicativo micro-ROS pode usar. Isso leva a um limite superior nas atribuições de memória, que pode ser configurado pelo usuário antes do processo de construção (MICRO-ROS, 2022).

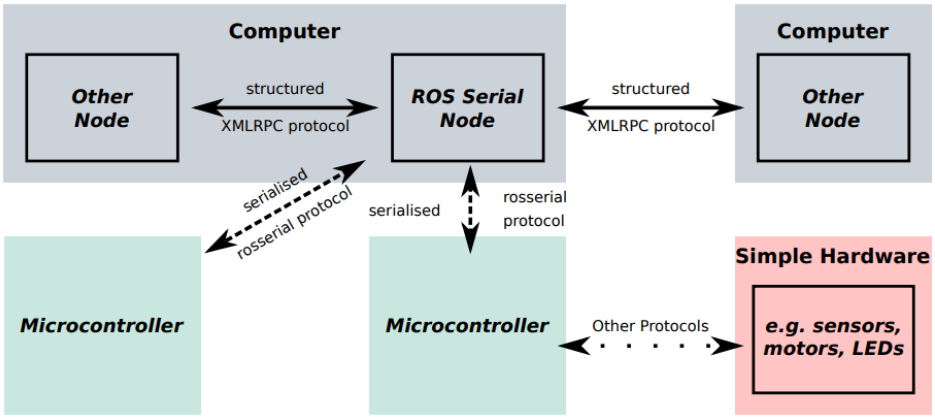
O suporte de hardware micro-ROS é dividido em duas categorias: placas oficialmente suportadas e placas com suporte da comunidade. O Arduino Due, por exemplo, é uma das estruturas com suporte da comunidade micro-ROS (MICRO_ROS_ARDUINO, 2022). Inicialmente, ela é fornecida como uma biblioteca pré compilada, no entanto, como mencionado anteriormente, os usuários podem modificar os parâmetros de construção e reconstruir, personalizando de acordo com as necessidades.

2.9.5.2 Ros-serial

O ros-serial é uma biblioteca que implementa um protocolo para encapsular mensagens ROS de microcontroladores. Para isso é usada uma topologia cliente-servidor, onde um ros-serial servidor é um computador rodando ROS e um ros-serial cliente é o microcontrolador que recebe os dados dos sensores e os transporta para o servidor na forma de mensagens ROS.

A Figura 2.25 mostra que *ROS Serial Node* é usado para comunicar os dados ROS do microcontrolador ao computador e vice versa, permitindo a interação de *hardware* não compatível com a rede ROS (ARVIN; LENNOX; WATSON, 2018).

Figura 2.25: Comportamento ros-serial.



Fonte: (ARVIN; LENNOX; WATSON, 2018).

2.9.5.3 Ros2arduino

Ros2arduino é uma biblioteca similar a micro-ROS, ela foi criada para que placas de microcontroladores possuíssem comunicação do padrão ROS2 (*Dashing* e *Crystal*) via serial ou protocolo de datagramas do usuário (*User Datagram Protocol* – UDP). Para isso, mais uma

vez é utilizado o protocolo cliente-servidor da XRCE-DDS, onde o agente é o microcontrolador e o cliente é o computador (CHO, 2020).

A ros2arduino possui as seguintes restrições (CHO, 2020):

- RAM maior ou igual a 32 kB;
- Placas suportadas: OpenCR, Arduino MKR ZERO e ESP32.

A solução proposta pela ros2arduino é interessante, no entanto, o projeto, que possui 136 estrelas e 39 bifurcações no GitHub, teve sua última modificação em julho de 2020 e, ainda, há *issues* sem resposta desde outubro de 2021, o que pode indicar a descontinuidade da biblioteca (CHO, 2020).

2.9.6 Ros1_bridge

Ros1_bridge é um pacote ROS 2 que fornece comunicação bidirecional entre ROS 1 e ROS 2. Ele permite, portanto, assinar mensagens em uma versão do ROS e publicá-las em outra versão do ROS. Para o mapeamento automático entre as mensagens ROS 1 e 2 são implementas três etapas. Na primeira os pacotes ROS 1 que terminam com _msgs são associados aos pacotes ROS 2 que terminam em _msgs ou _interfaces. Já na segunda, as mensagens com o mesmo nome são associadas umas às outras. E, por fim, na terceira etapa, os campos de mensagem são associados entre si. Logo, se o nome do pacote, o nome da mensagem e todos os nomes e tipos de campo forem os mesmos entre um pacote ROS 1 e um pacote ROS 2, as condições de sufixo são satisfeitas, sendo assim, as mensagens serão automaticamente associadas umas às outras sem especificação adicional. Além disso, é possível realizar o mapeamento personalizado a partir de algumas diretrizes (ROS1_BRIDGE, 2022).

2.10 PROGRAMAÇÃO

Os computadores são utilizados para as mais diversas tarefas, por causa disso, há uma diversidade de linguagens de programação com objetivos diferentes (SEBESTA, 2018). Para este trabalho, em especial, destacam-se as linguagens C e C++, devido à utilização em sistemas embarcados (VIARHEICHYK, 2020), como pela Arduino IDE, e por ROS 2, através de RCLCPP. Onde C++ é uma linguagem baseada em C, originada na Bell Labs, sendo uma de suas características marcantes a Programação Orientada a Objetos (POO) (SEBESTA, 2018).

Em C e C++ é comum a utilização de arquivos de cabeçalho (por padrão *.h* ou *.hpp*), são arquivos que contém as estruturas de dados, como as declarações de funções, tipos de dados e macros. Esses arquivos são, então, importados, via diretiva *#include*, em arquivos *.cpp*, que

fazem as implementações. Em suma, a *#include* insere uma cópia do arquivo de cabeçalho diretamente no arquivo *.cpp* antes da compilação (MICROSOFT, 2021).

As diretivas de pré-processador, como a *include*, são instruções direcionadas ao compilador, sendo as operações realizadas antes da compilação. Todas as diretivas de pré-processador são iniciadas com o símbolo #. Um outro exemplo útil de diretiva é *#ifndef*, que pode ser utilizada para não incluir um arquivo de cabeçalho mais de uma vez, evitando, assim, quebra de código. Neste caso, tem-se uma compilação condicional, onde essa diretiva verifica se uma determinada constante, relacionada apenas àquele arquivo, não está definida. Se estiver definida nada acontece, mas, caso contrário, a constante é definida e o código compilado (MICROSOFT, 2021).

Um erro comum de programadores é sacrificar a clareza do código em favor da velocidade de implementação, no entanto, isto gera problemas sérios, já que é normal olhar para um código escrito há algumas semanas atrás e não entender ou lembrar para o que era ou como funcionava (VALVANO, 2014). Afinal, código ruim pode funcionar, no entanto, isto pode gerar um desperdício incontável de horas. Um código é mais lido do que escrito, logo, um desenvolvedor acaba escrevendo *software* para outras pessoas ou a si mesmo (DESCHAMPS, 2019).

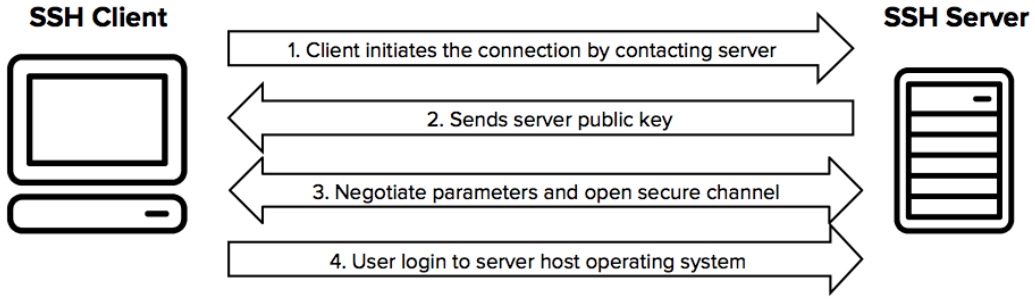
Arquitetura limpa (MARTIN, 2019) é uma boa referência para aqueles que pretendem criar códigos melhores. Nele são discutidas algumas práticas que otimizam o trabalho de refatoração e de *debug* de código. O nome é um dos temas abordados. Dá-se nome a diversas coisas relacionadas a programação, sejam funções, variáveis ou arquivos, sendo que bons nomes economizam tempo, dado que ajudam para que o código fique mais intuitivo. Ademais, outro ponto de discussão e destaque, é quanto ao tamanho das funções, esse ponto é mais abstrato, mas, em suma, tem-se que funções menores são mais fáceis para *debug*, já que um escopo menor facilita a estimativa de possíveis *bugs*. Essas práticas são simples, mas importantes para aqueles que se preocupam com a manutenção e reuso de *software*s, dado que um *software* fácil de entender é fácil de corrigir erros, verificar e adicionar recursos (VALVANO, 2014).

2.11 ACESSO REMOTO

É possível acessar um computador remoto na mesma rede usando o protocolo *Secure Shell* (SSH). O SSH usa um modelo cliente-servidor, onde o cliente é a máquina que está solicitando uma conexão e o servidor a máquina que está concedendo a conexão, Figura 2.26. Na primeira vez que a conexão for feita, as máquinas fornecem chaves umas às outras (NOTT,

2016). Quando uma conexão SSH é estabelecida, uma sessão *shell* é iniciada, possibilitando que o cliente digite comandos que serão executados no servidor.

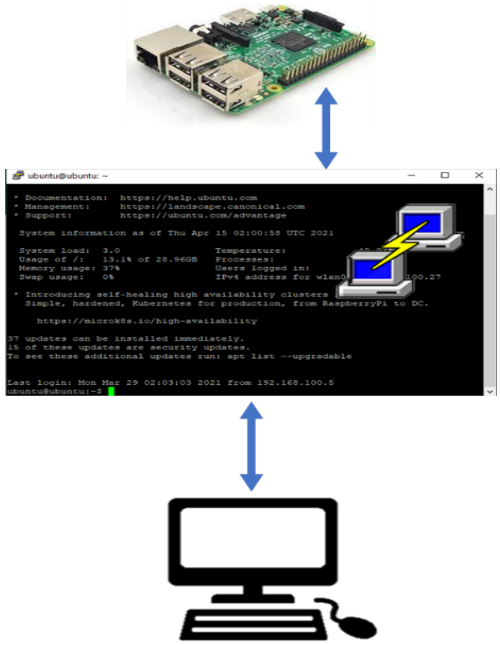
Figura 2.26: Comunicação SSH.



Fonte: (SSH, 201X).

Considerando uma situação em que não se tem à disposição um teclado, mouse e monitor para operar o SO de um Raspberry, o protocolo SSH pode ser usado como forma rápida e fácil de acesso remoto (KURNIAWAN, 2019), conforme mostrado na Figura 2.27.

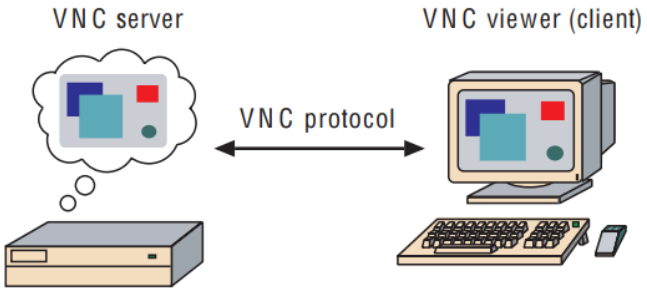
Figura 2.27: Acesso remoto através de SSH.



Fonte: Autor.

O *Virtual Network Computing* (VNC) é outra forma de acesso remoto, só que com interface gráfica. A Figura 2.28 mostra um exemplo, onde o cliente VNC visualiza a tela do servidor VNC. Sendo assim, eventos de teclado e mouse, por exemplo, são transmitidos do cliente para o servidor e, posteriormente, as atualizações gráficas ocorridas no servidor são enviadas para o cliente, que visualiza quase que em tempo real as mudanças. Caso necessário, vários clientes podem se conectar a um servidor VNC ao mesmo tempo (RICHARDSON et al., 1998).

Figura 2.28: Acesso remoto por VNC.



Fonte: (RICHARDSON et al., 1998).

2.12 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados conceitos teóricos necessários para compreensão e implementação do robô, principalmente conteúdo relacionado a ROS, determinante para implementação deste projeto. No próximo capítulo, são apresentados os materiais utilizados e a metodologia seguida para implementação deste trabalho.

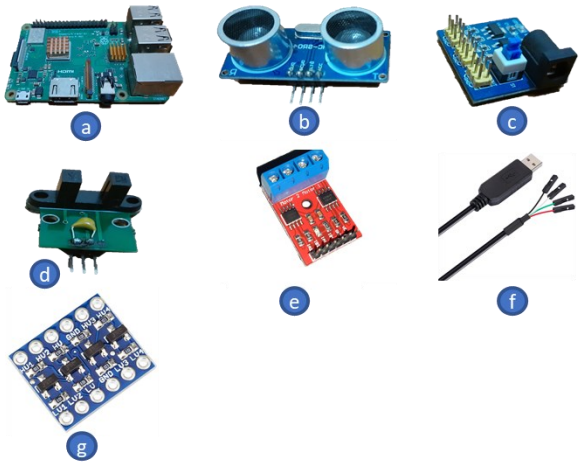
3 MATERIAIS E MÉTODOS

Tendo em vista a necessidade do robô de um certo grau de inteligência para realizar tarefas complexas em ambientes dinâmicos e considerando as expectativas futuras de que o robô seja dotado de autonomia, capacidade computacional, baixo preço, versatilidade, acessibilidade e conectividade, formam-se argumentos fortes a favor da placa Raspberry (HENRIQUE, 2019; SANTOS, FLÔR, 2019). Ainda, o Raspberry pode funcionar com sensores e dispositivos externos, no entanto, o Arduino faz isso muito melhor (CICOLANI, 2018).

Nessa circunstância, optou-se pelo uso das placas Arduino e Raspberry em conjunto. Ao combinar o Raspberry Pi e o Arduino pode-se aproveitar o que cada um faz de melhor. O Raspberry Pi oferece boa capacidade de processamento, enquanto que o Arduino fornece o controle de baixo nível sobre dispositivos externos, como sensores. Dessa forma, é possível realizar tarefas em paralelo, por exemplo: enquanto o Pi processa fluxos de dados de vídeo, o Arduino pode reunir informações dos vários sensores e aplicar uma determinada lógica (CICOLANI, 2018). Dessa maneira, usando o Raspberry em conjunto com o Arduino a capacidade computacional da plataforma é potencializada, possibilitando aplicações mais complexas.

Além do Arduino e do Raspberry outros componentes foram definidos para compor o robô, para referência futura tem-se a Figura 3.1, que mostra os principais componentes utilizados. Durante o capítulo de desenvolvimento deste trabalho serão disponibilizadas mais informações sobre os componentes e as motivações que orientaram a escolha dos mesmos.

Figura 3.1: Componentes Scicobot. (a) Raspberry Pi 3B+. (b) HC-SR04. (c) *Step Down* AMS1117. (d) *Encoder*. (e) *Driver* de motor Ponte H dupla. (f) Cabo Ttl Rs232. (g) Conversor de Nível Lógico.



Fonte: Google.

3.1 MATERIAIS

A Tabela 3.1 indica a lista de materiais (*Bill of Materials* – BOM) e os custos estimados para aquisição dos componentes necessários para montagem e utilização do Scicobot. Onde os componentes mais caros são o Raspberry 3B+ e o Arduino Due, os dois primeiros itens da tabela, que juntos somam mais de 60% do valor total.

Tabela 3.1: Materiais e o custos estimados

Nome	***Preço [R$]***	***Quantidade***
Placa Arduino Due	179,00	1
Raspberry 3B+	689,50	1
Case Raspberry	27,80	1
Chassi (com rodas e motores)	75,91	2
HC-SR04	12,89	1
Encoder Óptico	15,72	2
Resistor 330 ohms	~0,15	1
Light Emitting Diode (LED)	~2,00	1
Jumpers	~15,00	-
Conversor de Nível Lógico Bidirecional I2C - 4 canais	16,99	1
Driver de motor Ponte H dupla	9,80	1
Step Down 12V / 5V / 3.3V DC AMS1117	5,00	2
Bateria Lipo 1500mah 7.4v	174,90	1
Velcro (3 m)	25,00	-
Conversor Usb Serial Cabo Ttl Rs232	22,90	1
Cartão SD 32 GB	27,70	1
Adaptador cartão SD para USB	11,75	1
***Total*:**	***R$ 1408,64***	

Fonte: Mercadolivre e Aliexpress, 21/01/2022.

3.2 METODOLOGIA

Para desenvolver a estrutura robótica seguiu-se as seguintes etapas de desenvolvimento:

1) **Revisão e estudo bibliográfico**

Plataformas robóticas com propostas análogas a deste trabalho foram procuradas, revisadas e estudadas, esforçando-se para detectar problemas, design de *software*, ferramentas utilizadas e funcionalidades interessantes a serem implementadas. Além disso, o ambiente de trabalho foi configurado e aproveitou-se para revisitar exemplos e relembrar conceitos de ROS. Ao fim desta etapa, tinha-se uma idealização mais específica e robusta dos limites e possibilidades de implementação para o Scicobot.

2) **Comunicação entre Arduino e Raspberry**

A plataforma Scicobot utiliza um Arduino e um Raspberry, que precisam interagir entre si, no entanto, essa comunicação pode ser feita de diversas formas. Sendo assim, foi avaliada e determinada a solução adequada para o projeto, realizando as modificações necessárias. Após a escolha examinou-se exemplos fornecidos, que serviram como suporte inicial para compreensão da biblioteca.

3) **Definição dos componentes**

As estruturas avaliadas na etapa 1 foram utilizadas como referência para definição dos componentes utilizados. Dessa forma, a partir do panorama interpretado das referências e as modificações pendentes da etapa 2, foi realizado o levantamento de componentes a serem empregados no Scicobot. Para isso, considerou-se, ainda, os componentes disponíveis no laboratório da UFU, campus patos de Minas.

Cada componente escolhido foi testado a partir de exemplos e/ou implementações próprias, procurando interpretar o funcionamento e possíveis implementações de abstração de funcionalidades.

4) ***Software* Arduino**

Nesse ponto, tinha-se uma boa perspectiva da implementação de cada parte do robô, de forma separada, e a comunicação já havia sido selecionada e testada. Foi, então, que os conceitos fragmentados foram encapsulados em várias bibliotecas Arduino. Isso possibilitou que o Arduino do robô realizasse a leitura dos sensores, enviasse comandos de movimento para o motor e iniciasse a comunicação serial com o Raspberry.

5) ***Software* Raspberry**

Com a aplicação Arduino elaborada, partiu-se para o Raspberry, onde pacotes ROS 2 C++ foram criados para enviar, receber e manipular as informações ROS 2. A partir desse

estágio, o Scicobot estava funcional, podia-se utilizar os exemplos montados para o Arduino e o Raspberry a fim de execução algumas funcionalidades, como se locomover e manipular dados dos sensores no Raspberry.

6) **Simulação de sistema multi-robôs**

Para a robótica cooperativa é de suma necessidade estabelecer uma comunicação entre os robôs. Neste contexto, o Scicobot possibilita a interação entre dispositivos que estão executando um ambiente de trabalho ROS 2. Procurando validar essa afirmativa, executou-se um teste simples de trocas de mensagens entre um Scicobot e um Raspberry, que, neste caso, simula outro robô no sistema.

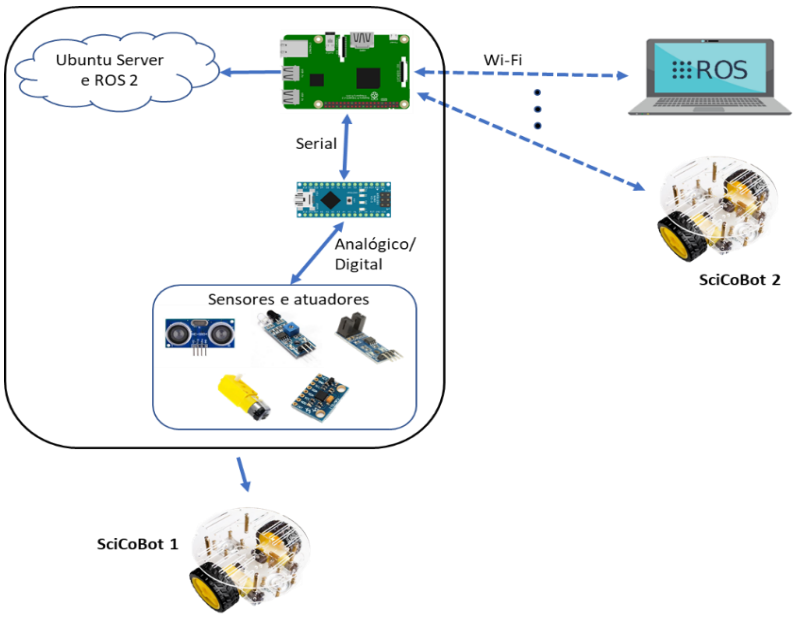
3.3 CONSIDERAÇÕES FINAIS

Neste capítulo foram mostradas as etapas e materiais utilizados para desenvolver o protótipo robótico móvel, bem como os custos e cronograma de atividades realizadas para o TCC 2. No capítulo seguinte, será descrito em detalhes o desenvolvimento da metodologia apresentada.

4 DESENVOLVIMENTO

Inicialmente, o projeto possuía esquema como o apresentado pela Figura 4.1. Onde os robôs possuem um Raspberry como componente de alto nível do sistema e um Arduino Nano como controlador e leitor de sensores. Nesta arquitetura, o Raspberry é o *hardware* responsável pela comunicação entre os dispositivos da rede. Para isso, o Pi possui um espaço de trabalho ROS 2 *Foxy* e Wi-Fi, que possibilitam a interação entre as estruturas que compõem o sistema.

Figura 4.1: Arquitetura de *hardware* e estrutura de comunicação prevista.



Fonte: Autor.

O desenvolvimento do projeto seguiu a metodologia definida no capítulo 3. Sendo nas subseções seguintes apresentados os detalhes para cada etapa.

4.1 REVISÃO E ESTUDO BIBLIOGRÁFICO

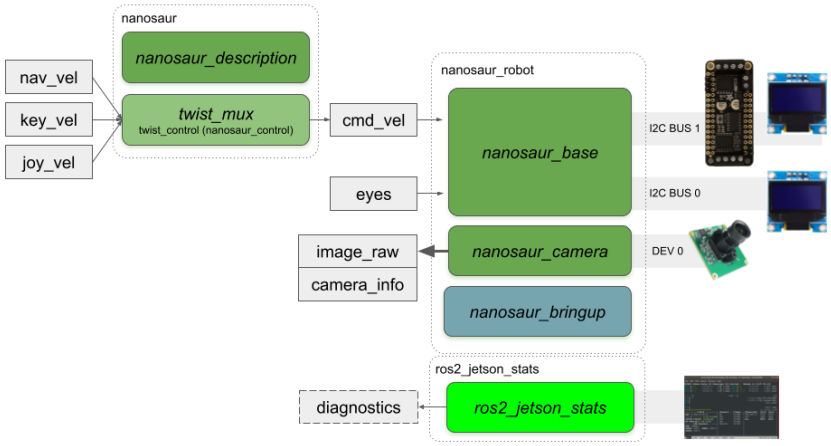
Iniciou-se executado alguns tutoriais de ROS2 (2022b) no computador com o objetivo de aprofundar e revisar conceitos de ROS 2, onde destacam-se: *Understanding ROS 2 nodes, Understanding ROS 2 topics, Understanding ROS 2 services, Creating a Workspace, Creating your first ROS 2 package* e *Writing a simple publisher and subscriber (C++)*. Em suma, os tutoriais forneceram uma boa base para implementações futuras, sendo os exemplos bastante didáticos. Após testes iniciais no computador, instalou-se o Ubuntu servidor 20.04 e ROS 2 Foxy no Raspberry, aproveitando nesse ponto para executar mais alguns testes, como experimentar a comunicação ROS 2 entre computador e Raspberry a partir da criação de nós em dos dispositivos e visualização desse mesmo nó no outro dispositivo.

Iniciou-se a revisão bibliográfica a partir das plataformas robóticas apresentadas na sessão 1.5, sendo selecionadas três estruturas para estudo mais aprofundado, sendo elas: Linorobot (JIMENO, 2021), Hadabot (HADABOT, 2021) e Nanosaur (BONGHI, 2022a). Essas estruturas foram escolhidas devido, principalmente, as suas propostas e materiais disponíveis.

4.1.1 Nanosaur

O Nanosaur é uma plataforma criada com o objetivo de ensinar os fundamentos de ROS 2 e NVIDIA Isaac ROS, que fornece pacotes para facilitar a criação de soluções ROS de alto desempenho em *hardware* NVIDIA. A Figura 4.2, apresenta a arquitetura do robô, onde nanosaur_base, nanosaur_camera e ros2_jetson_stats são nós ROS. Em resumo, tem-se que: nanosaur_base é responsável por fornecer o controle dos motores e olhos do robô; nanosaur_camera possui um editor que transmite o fluxo de dados da câmera; e, em jetson_stats, a placa NVIDIA Jetson Nano é monitorada e controlada (BONGHI, 2022a).

Figura 4.2: Estrutura de Nanosaur.



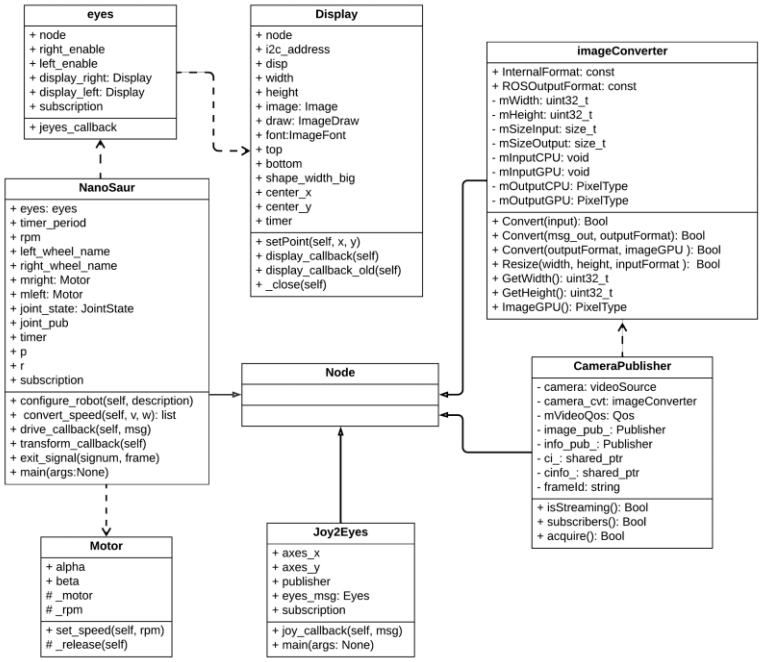
Fonte: (BONGHI, 2022a).

Ainda, destaca-se o processo de instalação, feito a partir de scripts *bash* que configuram o sistema para que, por exemplo, seja possível utilizar comandos Nanosaur no Terminal, e constrói um espaço de trabalho ROS 2. Tem-se, também, o sistema de construção, que é baseado em Docker, sendo ambos os processos, instalação e construção do *software*, simples, abstraindo bastante da complexidade. Outro destaque, é o suporte para simulação 3D do robô no aplicativo Gazebo (BONGHI, 2022a).

A Figura 4.3 destaca a interpretação realizada de parte da pilha de *software* de BONGHI (2022b). Onde, na parte esquerda da Figura 4.3, tem-se a estrutura para nanosaur_base, composta por eyes, Display, NanoSaur, Motor e Joy2Eyes, que, como especificado

anteriormente, possibilitam o controle do motor e dos olhos do Nanosaur. Onde NanoSaur parece utilizar da estrutura formada pelas outras classes para criar a dinâmica de trabalho. Ainda na Figura 4.3, são mostradas duas classes criadas em nanosaur_camera: imageConverter para conversão de formatos de imagem e CameraPublisher para aquisição e publicação dos frames.

Figura 4.3: *Software* nanosaur_base e nanosaur_camera.



Fonte: Autor.

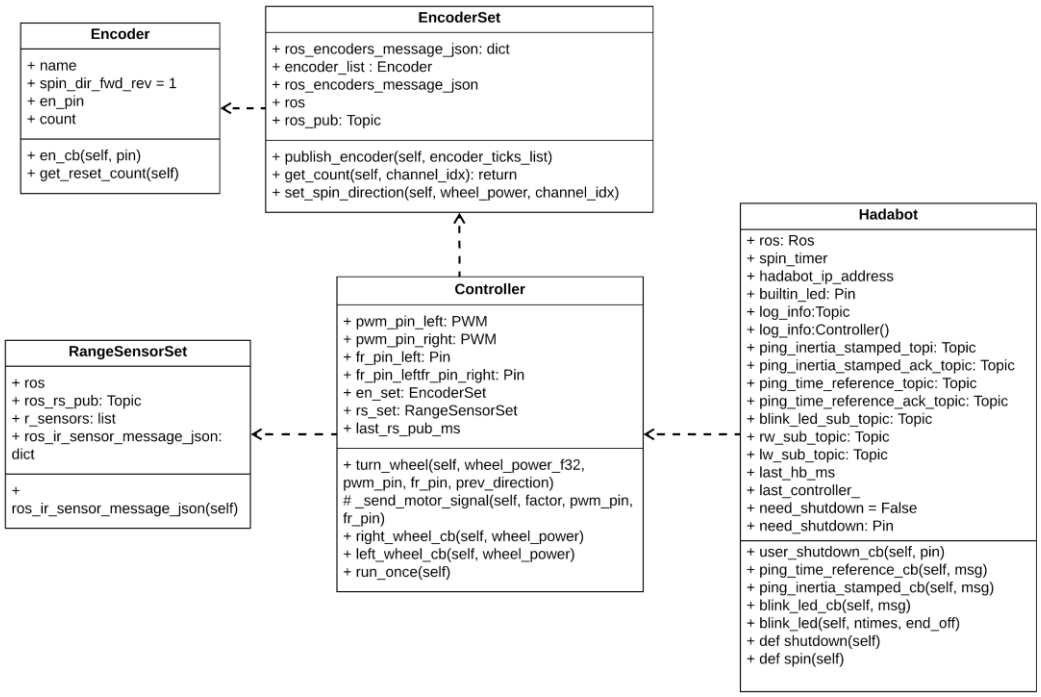
4.1.2 Hadabot

O Hadabot é um robô destinado aqueles que querem aprender conceitos de ROS 2 e robótica na prática. Dessa plataforma, destacam-se os materiais didáticos disponibilizados, que geralmente abordam algum tópico da robótica e utiliza o robô para exemplificar implementações e/ou realizar testes práticos. Alguns dos temas abordados nesses materiais são: conceitos ROS 2, localização, planejamento de movimento, localização, mapeamento, navegação e odometria.

Tentou-se identificar a lógica de programação adotada, principalmente para o microcontrolador ESP32, no entanto, a falta de uma descrição detalhada do raciocínio de implementação atrapalhou a compreensão total da estrutura de *software* utilizada. A Figura 4.4 apresenta o diagrama de classes que foi montado a partir da interpretação realizada do *software* para ESP32 (HADABOT, 2022). Onde, na classe Encoder são declaradas variáveis e funções que são utilizadas por EncoderSet que, por sua vez, realiza configurações e define instruções relacionadas ao sensor *encoder*, como uma função destinada a publicação dos dados do sensor

a partir de ROS. Já RangeSensorSet, é uma classe criada para definir os procedimentos para o sensor de distância, como Encoder e EncoderSet fazem. Ademais, a classe Controller é usada como uma ponte das outras classes com Hadabot. Nela é implementada parte da dinâmica de trabalho geral, como pode ser verificado pela definição da função *run_once(self)*, onde os valores dos sensores são acessados e publicados. Por fim, tem-se a classe Hadabot, que abstrai ainda mais as implementações das classes anteriores, iniciando o objeto Ros, que é passado como parâmetro para as outras classes, e um objeto Controller, além de atribuir os inscritos que são utilizados para receber comandos de movimento.

Figura 4.4: *Software* Hadabot.



Fonte: Autor.

4.1.3 Linorobot

Como apresentado anteriormente, o Linorobot2 é um conjunto de robôs compatíveis com ROS 2, de código aberto e que visa fornecer a estudantes, desenvolvedores e pesquisadores uma plataforma para criação de novos aplicativos. Ele possui dois projetos principais, um para o computador embarcado (JIMENO, 2022a) e outro para a placa Teensy utilizada (JIMENO, 2022b).

Do trabalho de JIMENO (2022a), destaca-se a utilização de um arquivo *bash* para instalação e configuração dos pacotes do projeto, como micro-ROS e *drivers* LiDAR. Além

disso, esse projeto também disponibiliza pacotes para um possível computador *host*, com *lauchs* para a simulação 3D no aplicativo Gazebo.

Para o microcontrolador, JIMENO (2022b) descreve a instalação de PlatformIO, plataforma utilizada para desenvolver, configurar e carregar o firmware na placa. Como Linorobot2 suporta versões diferentes de cada componentes, as bibliotecas são montadas de forma generalista, a partir do uso de várias diretivas de pré-processador para compilação condicional e arquivos para configuração de parâmetros. Especificamente, o projeto possui bibliotecas para o *encoder*, IMU, cinemática, motor, odometria e PID. No arquivo principal do projeto, *firmware.ino*, as bibliotecas são carregadas e utilizadas para executar o *pipeline* de trabalho, iniciado um nó ROS, os editores para o IMU e odometria, e um inscrito para receber comandos de movimento. Ainda, em (JIMENO, 2022b), são apresentados os diagramas de conexões elétrica para montagem do robô.

4.1.4 Contribuições

Sendo assim, as estruturas estudadas auxiliaram na idealização inicial de parte da estrutura de *software* do Scicobot. Principalmente, a ideia de modularização com classes para cada componente e o conceito de dividir as funcionalidades de um determinado componente entre duas classes, uma para as funcionalidades do componente e outra que utiliza o contexto criado pela primeira para trocar informações no mundo ROS, como Encoder e EncoderSet de Hadabot fazem. Além disso, notou-se certa preocupação quanto à impressão de informações, dada a existência de estruturas criadas para esse fim, como ros2_jetson_stats de Nanosaur e a classe Logger de Hadabot, onde, provavelmente, são empregadas para depuração de código e verificação de dados. Logo, atentou-se em otimizar a depuração do Scicobot, afinal, quando se trata de pesquisas a realização de testes é importante, sendo assim, ter à disposição alguma ferramenta que auxilie a na obtenção de dados de execução é indispensável.

4.2 COMUNICAÇÃO ENTRE ARDUINO E RASPBERRY

O desenvolvimento partiu da concepção da Figura 4.1, procurando avaliar o Arduino Nano. Nos testes realizados com alguns sensores e o Nano, sentiu-se que a expansão da plataforma poderia ser prejudicada devido à quantidade de portas digitais fornecidas, já que com três sensores e uma ponte H dupla tem-se, aproximadamente, dois terços das portas ocupadas. Chegou-se a cogitar a utilização de mais Nanos, no entanto, essa solução poderia

aumentar significativamente a complexidade do projeto, já que adicionaria mais camadas de comunicação, entre Nanos e entre Nanos e Raspberry.

Em paralelo com o panorama descrito, iniciou-se a avaliação da forma de comunicação entre Arduino e Raspberry. Em resumo, as possibilidades vislumbradas eram:

- Usar a biblioteca ros2arduino, no entanto, essa biblioteca não é compatível com o Nano, dado a restrição de RAM;
- Utilizar ROS 1 com a biblioteca ros-serial no Nano e empregar ROS 2 Foxy com ros1_bridge no Raspberry, contudo, esta possibilidade é complexa e desinteressante para a padronização vislumbrada, já que utilizaria duas versões de ROS diferentes;
- Criar um padrão de comunicação de dados no Nano que empregasse UART, I2C ou SPI. No entanto, como a anterior, esta solução escapa da padronização que se almeja, adicionando dois padrões de comunicação, um próprio e ROS 2;
- Utilizar micro-ROS, opção com maior potencial, mas não suportada pela Arduino Nano.

Todo esse contexto, formou fortes argumentos para troca do microcontrolador. As opções consideradas formavam-se em torno da Arduino Due e Teensy, principalmente porque a comunicação entre microcontrolador e Raspberry poderia ser feita utilizando a micro-ROS, solução mais interessante entre as descritas. Logo, optou-se pela troca da Arduino Nano pelo Due, visto que ela possui mais que o dobro de portas digitais, Tabela 2.2; possibilita o uso da micro-ROS; e mantém o robô na plataforma Arduino, que possui concepções que vão de acordo com a proposta deste trabalho. Além disso, a escolha da versão utilizada do Raspberry foi a 3B+, tendo em vista a disposição deste componente no laboratório.

Deve-se destacar que a micro-ROS se sobressaiu devido ao seu largo suporte, maior quantidade de recursos se comparada a ros2arduino (ROMÁN, 2017) e o envolvimento de grandes empresas no projeto, como: Bosch, LG, Amazon e Espressif (MICRO-ROS, 2022).

4.3 DEFINIÇÕES DOS COMPONENTES

A nível de *hardware*, o objetivo é possuir uma arquitetura expansiva para uso em robótica cooperativa, no entanto, este tema é muito abrangente, pode-se ter a cooperação entre robôs para as mais diversas finalidades, sendo a finalidade limitante pela escolha dos componentes do robô. Todavia, não é viável que a plataforma proposta tenha componentes suficientes para todo tipo de aplicação. Por isso, foi decidido que seria montada uma arquitetura considera básica, com alguns sensores e atuadores simples, como forma de validar e exemplificar o uso do *software* e da dinâmica proposta. Ainda, o objetivo é que haja

documentação suficiente para que pesquisadores tenham a capacidade de modificar conforme suas necessidades, criando novas versões a partir da estrutura principal (chassi, Arduino Due, cabo conversor USB-TTL e Raspberry Pi 3B+), que, por sua vez, podem ser adicionadas ao projeto GitHub como uma nova vertente.

Foi realizada uma revisão bibliográfica a fim de definir os componentes do robô. Durante esse período, procurou-se trabalhos que disponibilizassem uma perspectiva descritiva dos sensores utilizados nas principais plataformas robóticas, atentando-se à repetição de alguns componentes. Nesta perspectiva, partindo de NEDJAH e JUNIOR (2019), tem-se uma lista de robôs utilizados em estudos de robótica de enxame com seus respectivos sensores, onde quatro ou mais apresentam sensores IR, câmera, acelerômetro, luz ambiente, microfone e *encoder*. Destacando-se o sensor IR, utilizado em todas as arquiteturas, geralmente para detecção de obstáculos e durante a comunicação entre robôs. Ainda, WILSON et al. (2015) detalha algumas plataformas multi-robôs, onde se destacam os componentes: acelerômetro 3D, *encoder*, IR, câmera, luz ambiente e microfone. Por último, tem-se o trabalho de ANOOP e KANAKASABAPATHY (2017), que compara diferentes robôs, apresentando os sensores utilizados, sendo o IR e de luz visível os mais comuns.

Em conjunto, procurou-se avaliar os componentes utilizados em arquiteturas com propostas similares à deste trabalho, que poderiam ter partes aproveitáveis. A Tabela 4.1 traz essa relação, onde se destacam a câmera e o *encoder*.

Tabela 4.1: Robôs com propostas semelhantes ao Scicobot.

Robôs	***Componentes***
(THÁCIK; BREZINA; JADLOVSKÁ, 2019)	Raspberry, STM32, câmera, sonar, infravermelho e giroscópio.
Pheeno (WILSON et al, 2015)	acelerômetro 3D, magnetômetro 3D, *encoder*, IR, garra e câmera.
Linorobot (JIMENO, 2021)	Laser, IMU, Teensy, Raspberry e motor com *encoder*.
Hadabot (HADABOT, 2021)	ESP32, *encoder* e sensor de distância.
Nanosaur (BONGHI, 2022)	NVIDIA Jetson Nano, display OLED e câmera.

Fonte: Autor.

Dessa forma, observando a perspectiva apresentada e considerando os componentes disponíveis no laboratório, decidiu-se utilizar os sensores *encoder* e o ultrassônico. O *encoder* porque foi bastante citado nas referências analisadas e o ultrassônico vem como substituto do

IR, para medir distância de obstáculos, mesma função geralmente atribuída ao IR (consulte seção 2.5). Além disso, a preferência quanto a esses sensores se deve, principalmente, à disposição deles no laboratório e ao conhecimento prévio sobre eles.

Com os componentes do sistema definidos, testes foram realizados com cada um. Sendo que, parte da lógica implementada foi facilmente encontrada no Google, em alguns casos em *sites* de vendedores.

4.4 MONTAGEM

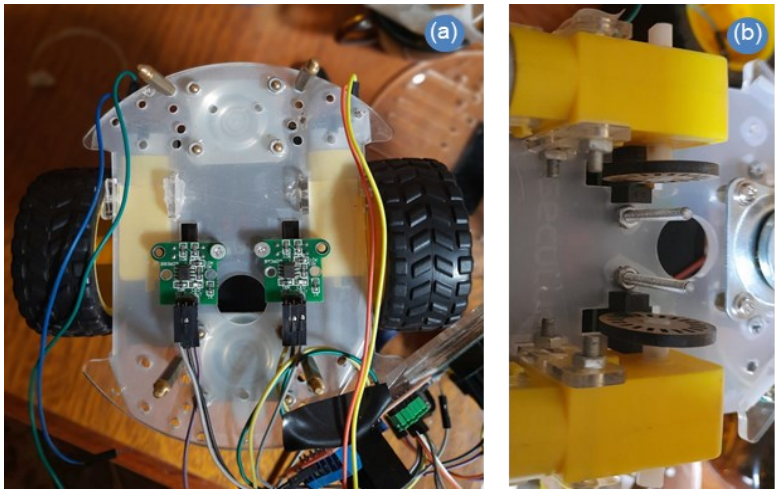
A Figura 4.5 mostra a estrutura adquirida, ele vem com um chassi, que dispõe de dois discos de acrílico; duas rodas laterais; e dois rodízios giratórios, frontal e traseiro. Onde mais discos podem ser obtidos para a adição de mais níveis na estrutura. Para o Scicobot deste trabalho três níveis foram utilizados, sendo que, não foi encontrado à disposição no mercado o disco de acrílico sozinho, foi necessário a compra de outra estrutura completa como forma de adquirir o disco.

Figura 4.5: Estrutura inicial.

Fonte: Autor.

No primeiro nível do chassi encontram-se os *encoders*, Figura 4.6 (a). Para isso, colocou-se primeiro os discos dos *encoders* na parte de baixo do chassi, Figura 4.6 (b), conectados aos eixos dos motores. Já em cima, os sensores foram facilmente posicionados, já que o disco disponibiliza duas entradas para encaixe dos mesmos.

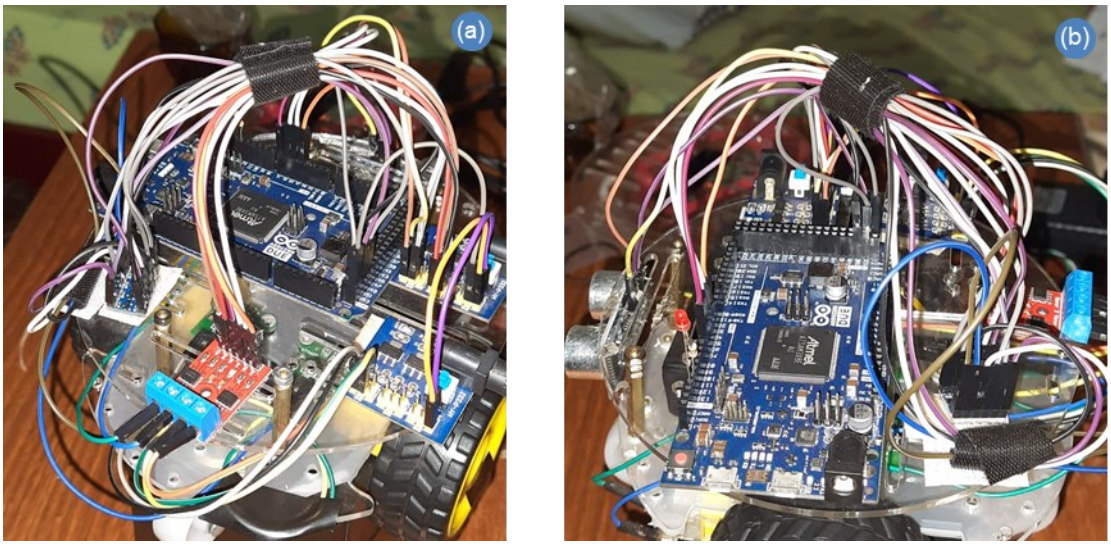
Figura 4.6: *Encoder* do Scicobot. (a) vista superior. (b) vista de baixo.



Fonte: Autor.

Colocando mais um disco, progrediu-se para o segundo nível. Nele foram alocados a ponte H, o Arduino Due, conversor de nível lógico e os conversores de tensão 12V / 5V / 3.3V, Figura 4.7 (a) e (b). Sendo o conversor de nível lógico utilizado para converter a tensão de saída e entrada de alguns componentes para níveis adequados, já que o Due opera em 3.3 V e esses componentes em 5 V. Para organizar os fios utilizou-se de velcro, apesar de não ser necessário é altamente recomendável essa prática, já que auxilia a visualizar os componentes e a fazer alterações nas conexões sem grandes problemas.

Figura 4.7: Nível dois. (a) vista traseira. (b) vista lateral.



Fonte: Autor.

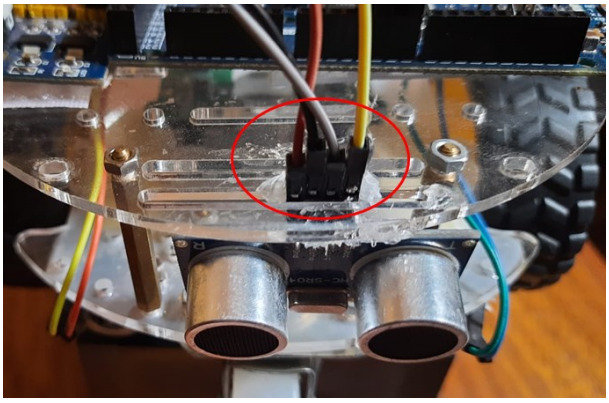
Ainda no segundo nível, destaca-se o componente conectado na porta digital 13 do Due, utilizado para debug. Trata-se de um LED e resistor 330 ohms, Figura 4.8. Para construção desse componente parte dos conectores do LED e do resistor foram cortadas e, depois, soldou-se os dois para ficar mais compacto.

Figura 4.8: Componente utilizado para debug.

Fonte: Autor.

Por outro lado, o ultrassônico foi posicionado entre os níveis um e dois, utilizando-se de uma abertura no chassi para encaixe dos *jumpers*, como destacado na Figura 4.9, sendo utilizada cola quente para fixação do mesmo no disco de acrílico.

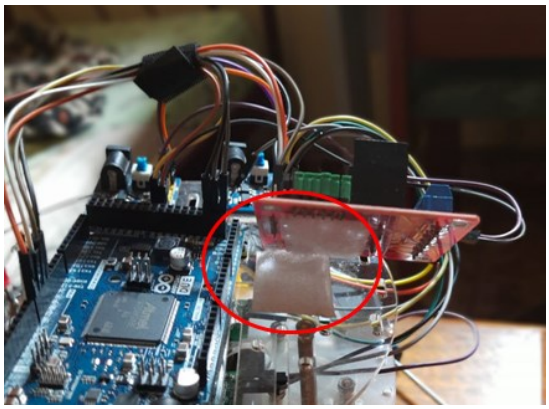
Figura 4.9: Ultrassônico do Scicobot.



Fonte: Autor.

Assim como para organização dos fios, foi utilizado velcro para fixação dos componentes no chassi, como mostra a Figura 4.10 para a ponte H de Scicobot. Isso se tornou uma solução muito interessante, já que é barata, fácil de colocar e satisfaz as necessidades.

Figura 4.10: Velcro para fixar componentes.

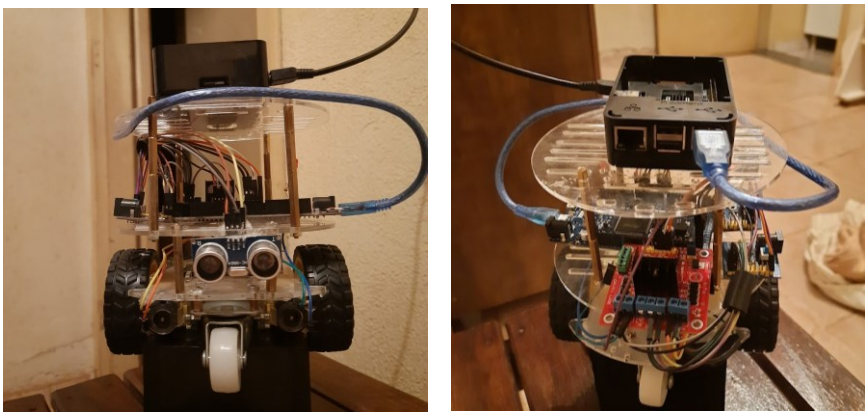


Fonte: Autor.

Por fim, foi adicionado o último nível, que inclui em seu espaço superior e inferior um Raspberry Pi 3B+ e bateria. Dessa forma, tem-se a plataforma robótica Scicobot montada,

Figura 4.11. Ademais, destaca-se que o fio azul que liga o Arduino e o Raspberry se estende de forma desagradável, mas não crítica, necessitando atenção do operador para não atrapalhar durante locomoção.

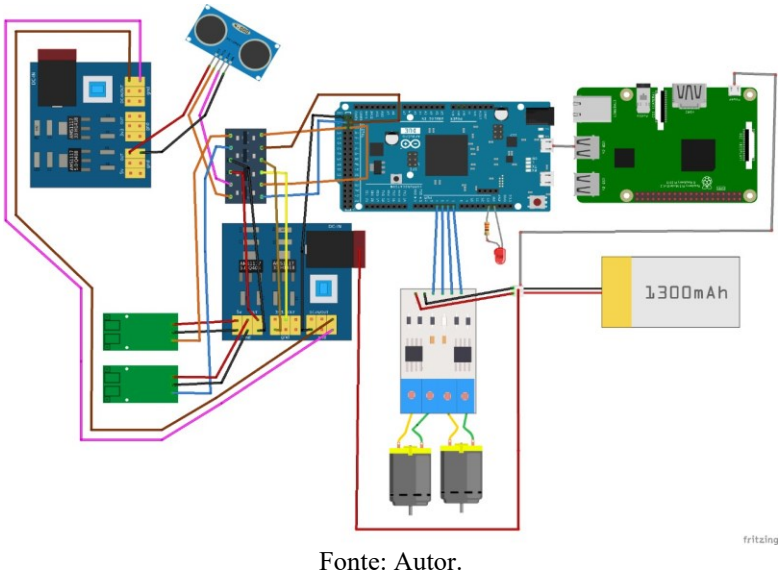
Figura 4.11: Scicobot 1, sem bateria.



Fonte: Autor.

O esquema de ligação entre os componentes do robô é apresentado na Figura 4.12, onde os fios cinzas que conectam a bateria e o Arduino ao Raspberry são representações simbólicas. Destaca-se que no esquema os dois conversores *Step Down* de 12V para 5 V e 3.3 V. Esse componente é utilizado para converter a tensão da fonte para valores de 5 V e 3.3 V, já que componentes do circuito trabalham nessas tensões. Ainda, após alimentar os componentes, tinha-se o problema de alguns componentes operando em 5 V e o Due em 3.3 V, logo, a ligação desses componentes em portas digitais do Due poderia danificar a placa ou reduzir sua vida útil, sendo assim, um conversor de nível lógico foi usado para converter 5 V em 3.3 V e 3.3 V em 5 V.

Figura 4.12: Esquema de ligação Scicobot 1.



Fonte: Autor.

Ademais, a fim de oferecer outra forma de identificar as ligações, montou-se a Tabela 4.2, onde as ligações dos componentes são especificadas. Nela, foram diferenciados os *Step Down*, como A e B, sendo *Step Down* A o que está na parte inferior da Figura 4.12.

Tabela 4.2: Ligações elétricas entre os componentes.

Componentes	Pino
Encoder Direito	VCC -> 5V (Step Down A) GND -> GND (Step Down A) OUT -> D51 (Arduino)
Encoder Esquerdo	VCC -> 5V (Step Down A) GND -> GND (Step Down A) OUT -> D52 (Arduino)
Ultrassônico	VCC -> 5V (Step Down B) GND -> GND (Step Down B) Trig -> D53 (Arduino) Echo -> D50 (Arduino)
Drive Ponte H dupla	VMS -> 3.3V (Fonte) GND -> GND (Fonte) Output A -> motor A Output B -> motor B IB1 -> D4 (Arduino) IA1 -> D5 (Arduino) IB2 -> D6 (Arduino) IA2 -> D7 (Arduino)
Arduino Due	GND -> GND (Step Down) Micro USB -> USB (Raspberry)
Step Down A	DC IN -> DC IN (Step Down B) GND IN -> GND IN (Step Down B)
LED	Polo positivo -> D13 (Arduino) Polo negativo -> GND (Arduino)
Raspberry 3B+	Micro USB -> Fonte

Fonte: Autor.

Em relação à adição de novos componentes, a estrutura pode ser expandida verticalmente, a partir de novos discos, como mostra a Figura 4.13, onde o robô à esquerda é

um Scicobot 1, desenvolvido neste trabalho, e o robô à direita uma nova versão, que possui um LiDAR, tecnologia a *laser* comumente utilizada para mapear ambientes. Logo, tem-se certas limitações quanto à expansão física, já que a adição de muitos níveis pode comprometer, por exemplo, a estabilidade e a capacidade de locomoção, visto que o balançar da estrutura pode derrubar o robô e o peso exacerbado pode não ser suportado pelos motores.

Figura 4.13: Expansão física do Scicobot.

Fonte: Autor.

4.5 *SOFTWARE* ARDUINO

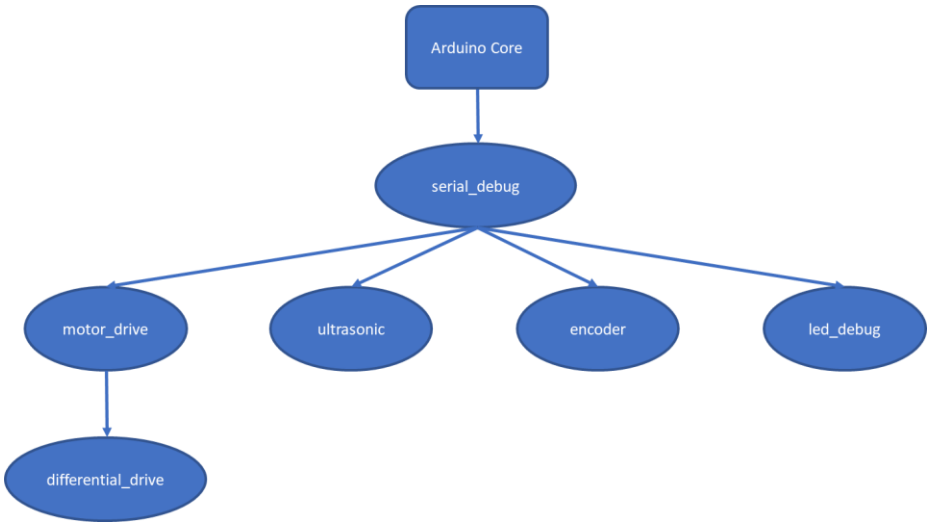
Com o conhecimento adquirido de cada parte do robô e alguns conceitos conhecidos de programação, como código limpo e diretivas de pré-processsador, criou-se as primeiras bibliotecas, em C++. Para isso, baseou-se parcialmente no material disponibilizado pela própria Arduino (ARDUINO, 2022b; ARDUINO, 2022c), que descrevem convenções para criação de uma biblioteca Arduino. As bibliotecas desta etapa são:

- Arduino core: local onde as funções básicas incorporados pela Arduino IDE estão definidas, como *pinMode()* e *analogRead()*;
- serial_debug: geralmente usa-se o monitor serial como forma de depuração no Arduino. Neste aspecto, a serial_debug é uma biblioteca que fornece macros e funções para auxiliar na depuração serial;
- led_debug: é inspirada em exemplos disponibilizados pela micro-ROS. Tem como objetivo fornecer funções e macros para debug utilizando um LED, que pisca quando um erro crítico acontece. A princípio, as implementações realizadas constam apenas com a lógica de depuração para micro_ros_arduino;
- differential_drive: biblioteca que implementa a lógica de controle diferencial;

- encoder: biblioteca que encapsula o trabalho com o *encoder*;
- ultrasonic: utilizada para adquirir e manipular informações do sensor ultrassônico;
- motor_control: implementa funções para controle dos motores DC a partir de uma ponte H.

A Figura 4.14 exprime o diagrama das bibliotecas apresentadas até aqui, sendo que as setas indicam as dependências entre elas, onde a que está mais perto do fim da seta é dependente da outra. Durante a definição das bibliotecas atentou-se principalmente a depuração, por isso, há duas bibliotecas com esse intuito, uma via serial e outra por LED. Além disso, destaca-se que a utilização de serial_debug é configurável, existe um parâmetro que habilita ou desabilita o uso desta biblioteca. Este processo foi implementado porque otimiza a execução para quando não há necessidade de depuração serial.

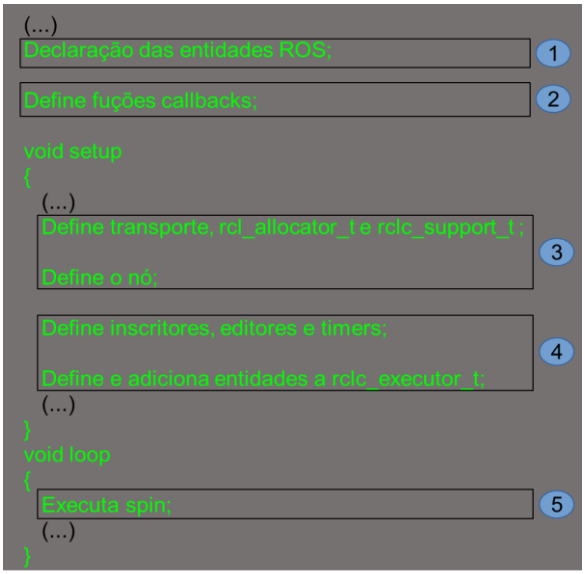
Figura 4.14: Estrutura das bibliotecas Arduino para controle de *hardware*.



Fonte: Autor.

Com o *hardware* encapsulado, partiu-se para a modularização do *middleware*, nesse caso, encapsulado pela micro_ros_arduino. Após análise dos exemplos disponibilizados por essa biblioteca, como (MICRO-ROS, 2021a) e (MICRO-ROS, 2021b), percebeu-se um certo padrão no fluxo de implementação. A Figura 4.15 exemplifica uma típica implementação de micro_ros_arduino, nela são destacados cinco pontos cruciais, tem-se: 1 - as entidades ROS são declaradas sem suas respectivas atribuição; 2 - as funções *callbacks* são definidas; 3 - transporte, nó, *allocator* e o *support* são definidos, sendo as duas últimas entidades utilizadas para criar e manipular nós, inscritores e editores; 4 - inscritores, editores e *timers* são definidos conforme estrutura ROS desejada, além disso, o *executor* também é definidos e o(s) *timer*(s) e/ou inscritor(es) são adicionados a ele; 5 - um *spin* é definido para o *executor* de antes.

Figura 4.15: Estrutura de implementação micro_ros_arduino.

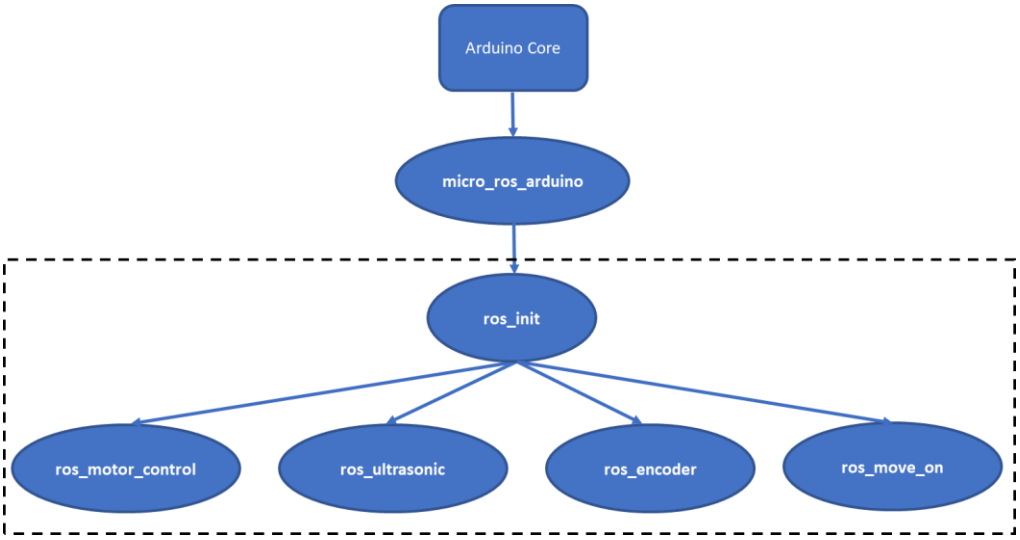


Fonte: Autor.

Dessa forma, decidiu-se estabelecer a arquitetura de *software* conforme mostrado na Figura 4.16. Em suma, o conteúdo do retângulo é um encapsulamento da micro_ros_arduino para as aplicações implementadas, ou seja, para comunicação dos sensores escolhidos e o controle de movimento do robô via teclado. As bibliotecas criadas nesta etapa são descritas a seguir, onde a primeira implementa o ponto 1 e 3 e as demais 1 e 4 da Figura 4.15. Já os pontos 2 e 5 são estabelecidos no código principal, onde fica o *main*. Além disso, as bibliotecas fornecem, funções *get* que retornam as entidades declaradas para uso posterior. Logo, tem-se:

- ros_init: responsável pela declaração e atribuição de *allocator*, *support* e nó. Para isso, considerou-se que a estrutura teria apenas um nó, denominado scicobot_arduino;
- ros_motor_control: incumbido da declaração e atribuição de um inscritor ROS, usado para receber os comandos diferencias que são utilizados no controle do robô;
- ros_ultrasonic: responsável por declarar e definir um editor ROS, útil para publicar as medidas realizadas pelo sensor ultrassônico;
- ros_encoder: utilizado para declarar e definir dois editores ROS, esses editores publicam a quantidade de pulsos contados por cada sensor *encoder*;
- ros_move_on: declara e define um inscrito ROS, empregado para definir a locomoção do robô.

Figura 4.16: Arquitetura prevista para abstração da micro_ros_arduino.

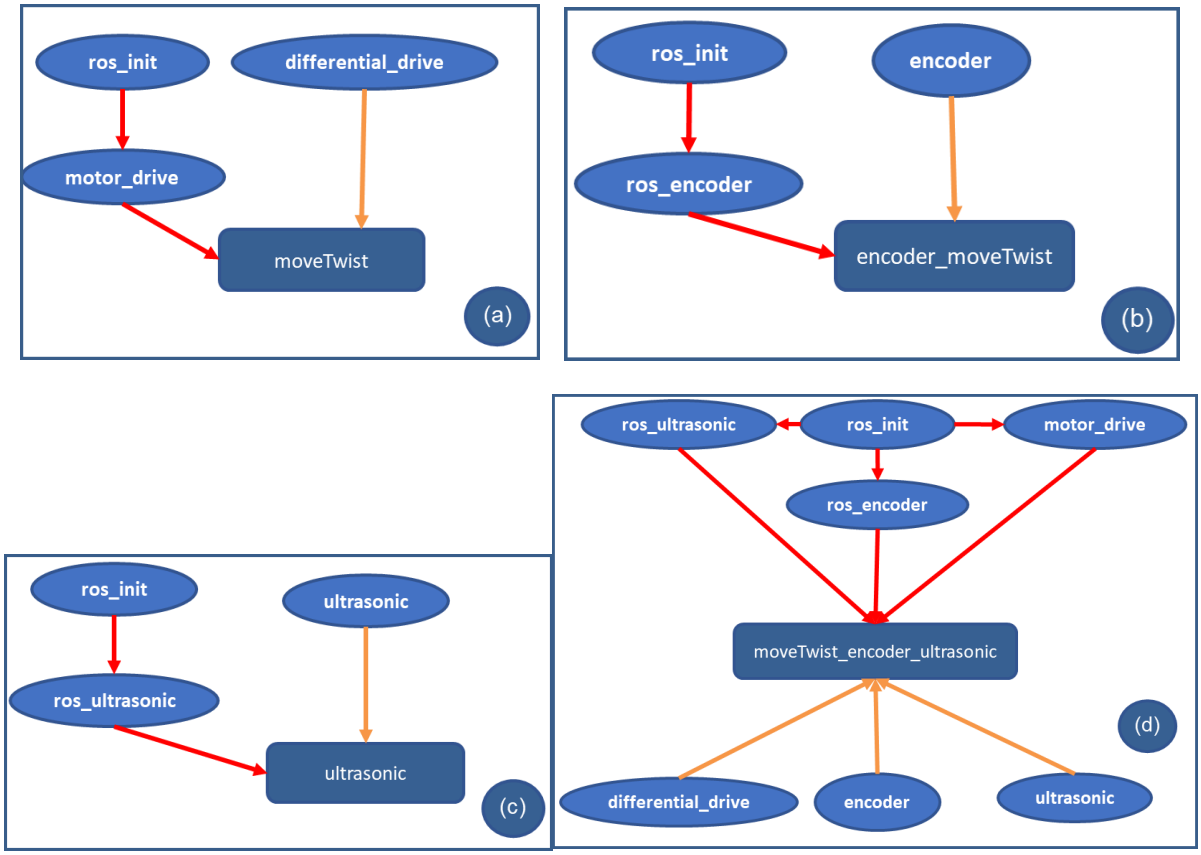


Fonte: Autor.

Com a base de bibliotecas do lado do Arduino finalizadas, desenvolveu-se um repositório GitHub (CRUZ, 2022d) que contém exemplos de implementação com os sensores e atuadores. Esses exemplos são, em suma, variações mescladas das bibliotecas criadas para micro-ROS e *hardware*.

A Figura 4.17 apresenta os diagramas dos exemplos criados, onde as setas indicam a dependência. Por exemplo, na biblioteca motor_drive da Figura 4.17 (a) tem-se o *#include* para a biblioteca ros_init. Para facilitar o entendimento as setas foram destacadas, sendo as vermelhas referentes ao *middleware* (Figura 4.16) e as laranjas ao *hardware* (Figura 4.14). Em suma, os exemplos são para: movimentar o motor via teclado, Figura 4.17 (a); movimentar o robô via teclado e realizar publicações dos dados dos sensores *encoders,* Figura 4.17 (b); publicar as medidas realizadas pelo sensor ultrassônico, Figura 4.17 (c); movimentar o robô a partir de comandos do teclado enquanto publica dados dos sensores ultrassônico e *encoders,* Figura 4.17 (d). Para a averiguação destes exemplos é necessário executar comandos ROS no Raspberry, como o pacote que publica os comandos do teclado, pode-se verificar CRUZ (2022d) para mais detalhes.

Figura 4.17: Exemplos Arduino. (a) moveTwist. (b) encoder_moveTwist. (c) ultrasonic. (d) moveTwist_encoder_ultrasonic.



Fonte: Autor.

4.6 *SOFTWARE* RASPBERRY

No Raspberry do robô é possível utilizar ROS 2 via linha de comando, a partir de uma conexão SSH com o computador. Nesta situação, é comum o uso de comandos como: *ros2 run,* para executar algum pacote e *ros2 topic echo,* para visualizar os dados de um determinado tópico. No entanto, esta não é uma forma otimizada de se trabalhar, é preferível que haja alguma aplicação que rode automaticamente, com necessidade mínima de intervenção. Seguindo este viés, montaram-se pacotes ROS 2 (CRUZ, 2022a) que executam determinadas dinâmicas de trabalho, como acessar dados de tópicos de sensores, manipulá-los e publicar o resultado. Além disso, utiliza-se de um projeto externo para envio de comandos diferenciais via teclado (ROS-INDEX, 2021).

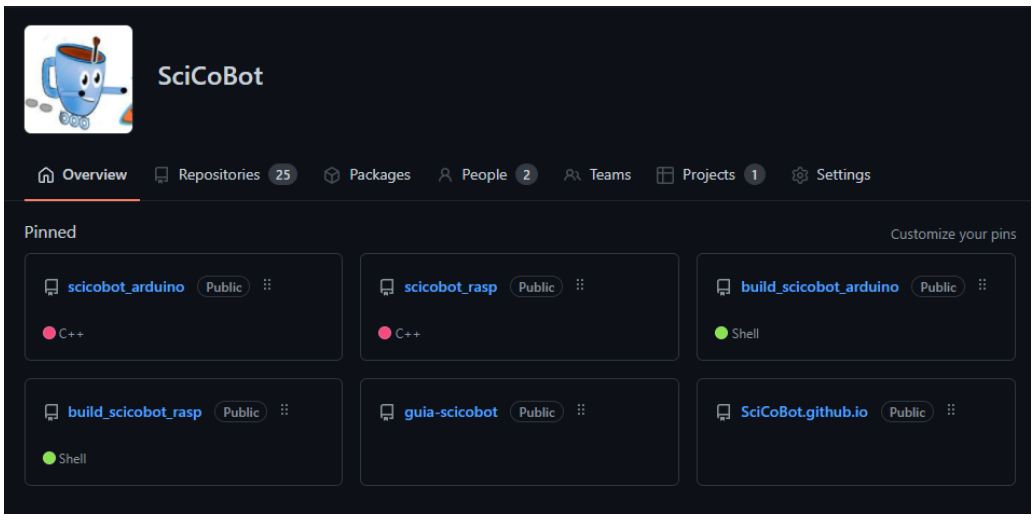
A estrutura de *software* proposta é bastante modular, com várias bibliotecas separadas, isso possibilita que o projetista importe apenas as partes necessárias, evitando o desperdício de recursos. De toda forma, foram criados dois sistemas de construção e manutenção de dependências a partir de arquivos *.bash:* um relacionado ao Arduino (CRUZ, 2022b) e outro a

Raspberry (CRUZ, 2022c), espera-se com isso diminuir a complexidade de gerenciamento de tantos arquivos.

Por fim, avaliou-se licenças de *software,* a fim de escolher uma que se adequasse à proposta do trabalho. Dentre as opções, as que melhor se alinhavam à perspectiva deste trabalho eram MIT e *GNU Affero GPL*, sendo a primeira permissiva e a última com *strong copyleft*. No caso da *GNU Affero GPL*, é interessante o aspecto de obrigar que projetistas disponibilizassem seus trabalhos para a comunidade, no entanto, acredita-se que essa exigência possa limitar a utilização da plataforma, já que o indivíduo não necessariamente quer que a plataforma seja distribuída como *GNU Affero GPL*. Ainda, o uso dessa licença pode originar uma sensação de incerteza jurídica nos projetistas, que podem não entender os limites legais da licença, fazendo com que tenham receio em utilizar o *software*. Nesse sentido, uma licença mais permissiva seria melhor, por isso, optou-se por distribuir as dependências sobre a licença MIT.

Todos os programas descritos estão disponíveis em CRUZ (2022e), uma organização GitHub que engloba toda a base de software desenvolvida para Scicobot, Figura 4.18. Inclusive com material descritivo além dos apresentados neste trabalho. Além disso, é neste local que se pretende publicar possíveis atualizações ao longo do tempo.

Figura 4.18: GitHub SciCoBot.



Fonte: Autor.

4.7 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado o desenvolvimento do *hardware* e *software* de Scicobot, conforme metodologia definida no capítulo 3. No capítulo seguinte, Resultados e Discussões, serão descritos os resultados obtidos e suas possíveis implicações.

5 RESULTADOS E DISCUSSÕES

O capítulo anterior apresentou o desenvolvimento das etapas da metodologia. Já neste capítulo, serão apresentados e discutidos os resultados referentes aos testes iniciais do *software*, simulação multi-robôs e uma breve discussão sobre os resultados e a estrutura montada.

5.1 TESTES DO *SOFTWARE*

Nesta etapa, com a plataforma física montada, foram executados testes procurando validar as implementações desenvolvidas para os componentes do Scicobot, sendo eles:

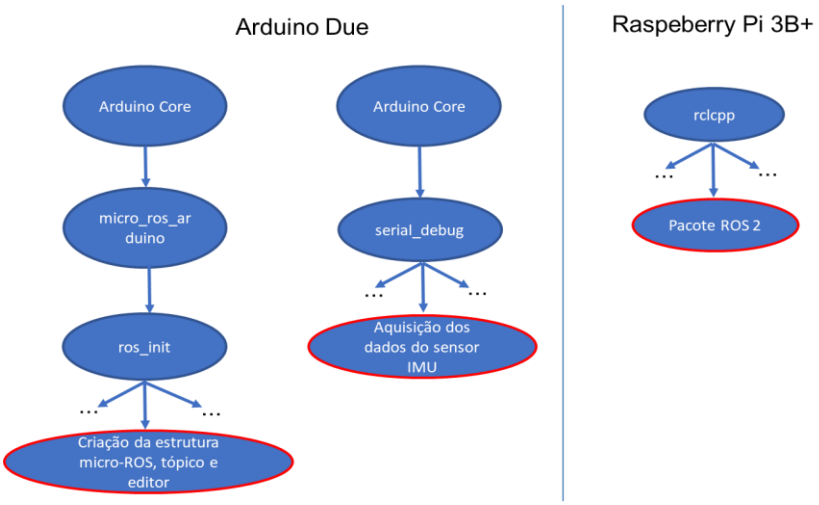
- Controle do robô por teclado: carregou-se o exemplo moveTwist de scicobot_arduino (CRUZ, 2022d) no Arduino e o pacote teleop_twist_keyboard (ROS-INDEX, 2021) no Raspberry. Isto fez com que no Arduino um inscrito recebesse os comandos publicados por um editor do Raspberry, fazendo com que os motores se movessem conforme o conjunto de parâmetros recebidos no Arduino;
- Encoder: para testar o encoder, utilizou-se do exemplo encoder_moveTwist, de scicobot_arduino (CRUZ, 2022d); do pacote scicobot_encoder, de scicobot_rasp (CRUZ, 2022a); e do pacote teleop_twist_keyboard (ROS-INDEX, 2021). Neste teste o robô foi movimentado via comando do teclado enquanto as medições do encoder eram realizadas e publicadas pelo Due. No Raspberry um inscrito acessava essas informações e as escrevia no Terminal;
- Ultrassônico: utilizou o exemplo ultrasonic de scicobot_arduino (CRUZ, 2022d) no Due e o pacote scicobot_ultrasonic (CRUZ, 2022a) no Raspberry. Essas execuções fizeram com que fossem criados um editor ROS no Due, que publicava as medições de distância, e um inscrito no Raspberry, utilizado para obtenção dos dados do sensor, sendo as medições recebidas mostradas no Terminal do Raspberry.

Destaca-se a dinâmica necessária para aferição dos testes, onde precisou-se carregar um projeto no Due e utilizar um dos pacotes ROS 2 no Raspberry, sendo essa prática essencial para qualquer tarefa que se pretende realizar utilizando as duas plataformas em conjunto com a estrutura de *software* montada. Logo, tem-se que os testes realizados com as implementações elaboradas (CRUZ, 2022a, 2022d) foram bem sucedidos e fornecem bons exemplos de implementação.

Espera-se que para novas implementações de *software* os usuários usem a base estabelecida, implementando novas funcionalidades. A título de exemplo, digamos que se pretende adicionar um IMU no robô Scicobot 1, para isso, poder-se-ia adicionar uma biblioteca

para leitura dos dados do IMU e uma ros_imu para implementação ROS 2 no Arduino, ou seja, ter-se-ia mais uma biblioteca na estrutura mostrada pela Figura 4.14 e Figura 4.16. Já no Raspberry, seria necessário montar um projeto que se inscrevesse no tópico criado, para que fosse possível acessar as informações publicadas. Sendo que o processo descrito foi destacado em vermelho na Figura 5.1.

Figura 5.1: Adição da aplicação IMU na estrutura de *software* Scicobot.

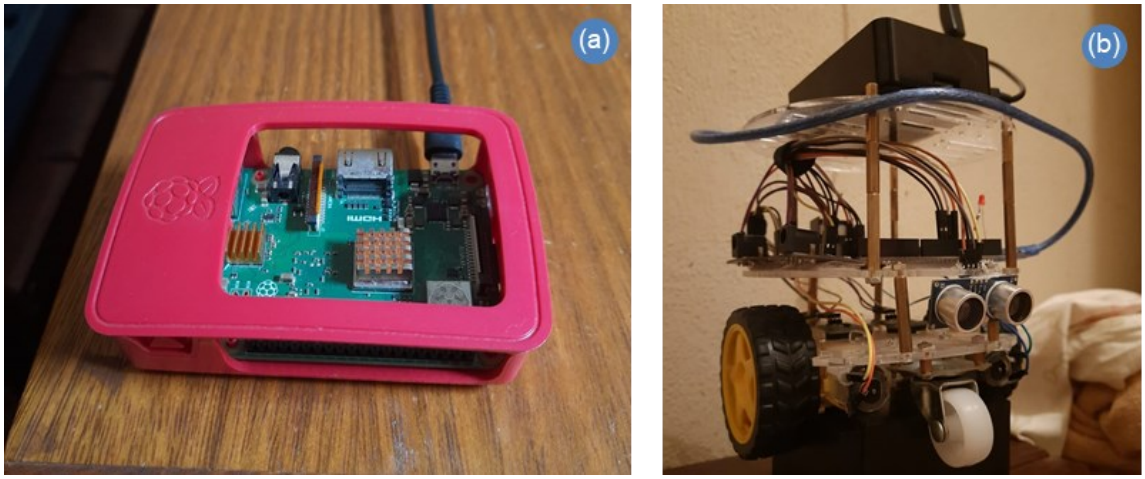


Fonte: Autor

5.2 SIMULAÇÃO MULTI-ROBÔS

A cooperação entre robôs pode ser simples, como compartilhar dados de sensores, ou complexo, como planejamento e atuação conjunta de movimentos (SIEFKE, 2020). Sendo que, qualquer aplicação cooperativa exige a comunicação entre os dispositivos do sistema. Nesta perspectiva, simulou-se uma aplicação cooperativa simples, onde tinha-se um Scicobot, um Raspberry e um computador. O Scicobot utilizado é indicado na Figura 5.2 (b), sendo ele responsável por envia os valores de leitura do sensor *encoder* para o Raspberry. Já o Raspberry, Figura 5.2 (a), envia informações de sua temperatura para o Scicobot, formando assim, uma comunicação bidirecional. Além disso, o teste contou com o auxílio de um computador, utilizado apenas como forma de acesso SSH para execução e visualização dos comandos no Raspberry e no Scicobot.

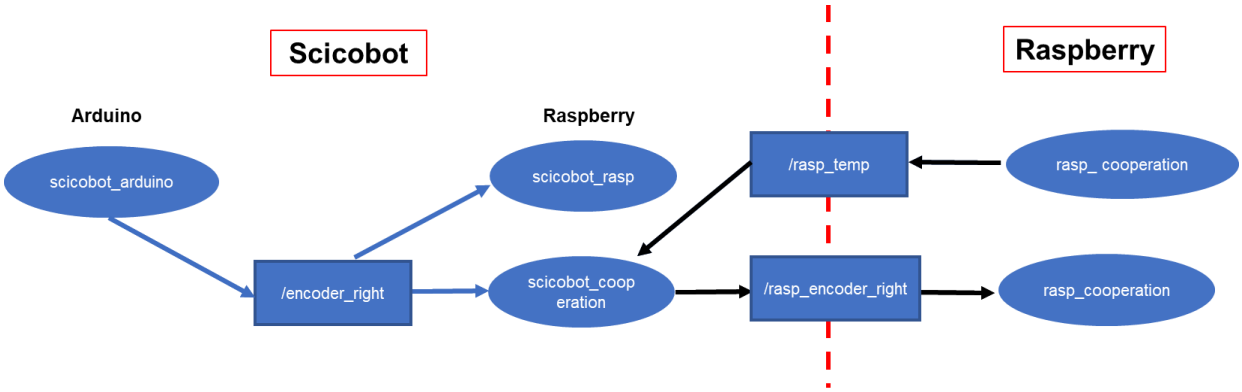
Figura 5.2: Estrutura utilizadas na simulação de sistema multi-robôs. (a) Raspberry. (b) Scicobot.



Fonte: Autor.

A Figura 5.3 mostra a arquitetura de nós implementada no teste multi-robôs, onde os dados do *encoder* direito do Scicobot são acessados pelo Raspberry a partir de um inscrito em */rasp_encoder_right,* enquanto informações de temperatura do Raspberry são publicadas no tópico */rasp_temp*.

Figura 5.3: Diagrama de nós da simulação de sistema com multi-robôs



Fonte: Autor.

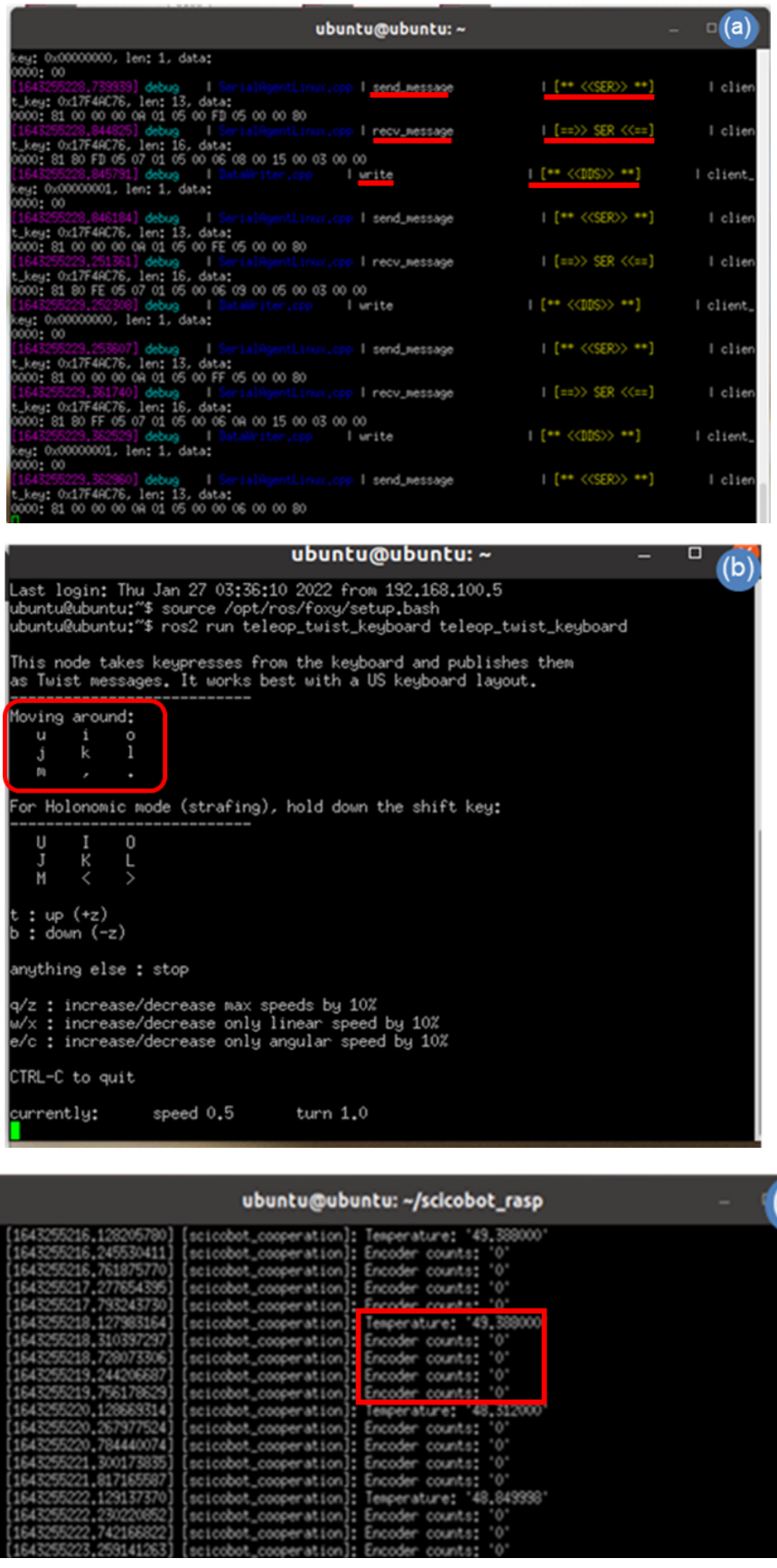
Usou-se de projetos do Scicobot e projetos externos para efetuar o teste multi-robôs. Sendo para isso carregado no Arduino o exemplo denominado encoder_moveTwist, responsável por publicar os dados do *encoder* e fornecer um inscrito para recebimento de comandos de movimento. Ainda no Scicobot, mas agora no Raspberry, executou-se os seguintes pacotes:

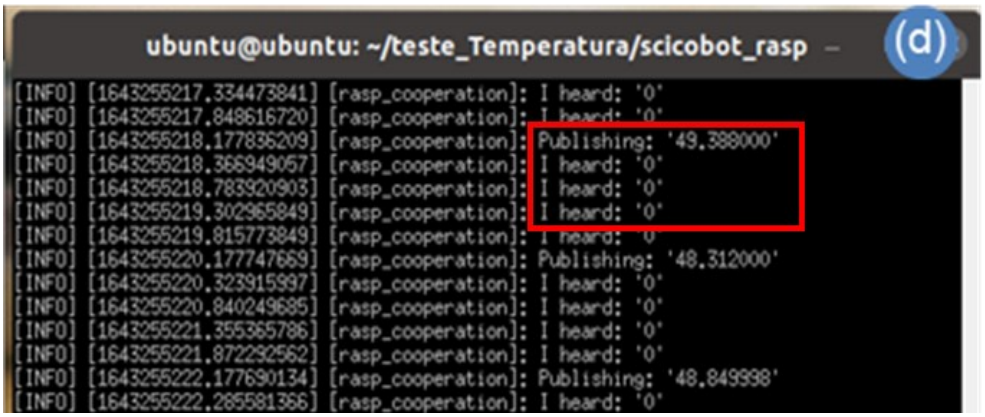
- micro_ros_agent: pacote externo que executa um agente micro-ROS para a comunicação serial entre Due e Raspberry. A Figura 5.4 (a) apresenta a tela dessa parte, onde foram destacadas algumas trocas de mensagem que ocorrem entre o servidor (Raspberry) e o cliente (Arduino);

- teleop_twist_keyboard: este é outro pacote externo a Scicobot, usado para controle de movimento via teclado. Na Figura 5.4 (b) é apresentada uma imagem de quando este pacote é utilizado, onde em vermelho estão escritas as definições das teclas para controle do robô;
- scicobot_encoder (CRUZ, 2022a): esse pacote Scicobot fornece inscritos para os tópicos dos *encoders.* Além disso, esse pacote é utilizado no teste com a finalidade de exemplificar uma aplicação com mais de um nó no Raspberry;
- test_cooperation_scicobot (CRUZ, 2022a): é utilizado para republicar as leituras dos sensores *encoders* e receber as informações de temperatura do Raspberry. A Figura 5.4 (c) apresenta as mensagens exibidas no Terminal após execução deste pacote, onde o quadrado vermelho contorna parte das mensagens exibidas, uma informação temperatura igual a 49.388000 e o valor de três publicações referentes ao *encoder*, todas iguais a zero já que o robô estava parado nesse momento.

No outro integrante do sistema, um Raspberry, executou-se o pacote test_cooperation_rasp (CRUZ, 2022a), que criar um editor para publicar as informações de temperatura do Raspberry e um inscrito para acessar as informações do *encoder*, disponibilizadas graças a test_cooperation_scicobot. A Figura 5.4 (d) mostra as impressões desse pacote no terminal, onde destaca-se a parte com a temperatura enviada igual a 49.388000 e as informações do *encoder* recebidas iguais a zero. Sendo assim, as mensagens da Figura 5.4 (c) e Figura 5.4 (d) condizem com o esperado, ou seja, a temperatura enviada por Raspberry é recebida por Scicobot, assim como as informações do *encoder* disponibilizadas por Scicobot são acessadas no Raspberry. Dessa forma, foi provado que é possível utilizar uma rede Wi-Fi e ROS 2 Foxy para comunicação entre pelo menos dois dispositivos que estão executando um espaço de trabalho ROS 2.

Figura 5.4: Saída dos Terminais para os pacotes. (a) micro_ros_agent. (b) teleop_twist_keyboard. (c) test_cooperation_scicobot. (d) test_cooperation_rasp.





Fonte: Autor.

5.3 DISCUSSÕES

As diferenças relevantes entre o Scicobot e as principais arquiteturas de referência são descritas na Tabela 5.1. Através da comparação compreende-se que os principais destaques de Scicobot é sua documentação extensa em português sobre conceitos e ferramentas relevantes na robótica, além de abranger a perspectiva de implementação na UFU, fazendo com que fossem escolhidos alguns dispositivos de uso comum ao longo do curso e disponíveis no laboratório, o que deve facilitar o uso da plataforma em pesquisas futuras para alunos da UFU, campus Patos de Minas. Ademais, Nanosaur e Linorobot parecem ser estruturas mais robustas, apropriadas para uso em projetos de alta complexidade, já a Scicobot é indicada para quando precisa-se de uma estrutura com recursos medianos, uma vez que possui mais capacidades que Hadabot e menos que Nanosaur e Linorobot.

Tabela 5.1: Comparação entre estruturas robóticas.

Robôs	**Comparação**
Hadabot (HADABOT, 2021)	- Tem estrutura física muito semelhante à de Scicobot, no entanto, utiliza um ESP32; - Usa de *hardwares* simples; - Possui um bom material descritivo.
Nanosaur (BONGHI, 2022)	- Possui bons recursos computacionais, devido ao uso da NVIDIA Jetson Nano; - É produzido em impressora 3D, para expansão deve ser necessário modificar o arquivo de impressão, o que não é trivial; - Possui boa quantidade de funcionalidades; - Tem bom material descritivo.

Linorobot (JIMENO, 2021)	- Não possui descrição detalhada da montagem; - Usa micro-ROS; - Utiliza a placa Teensy; - *Hardware* e *software* modulares; - Suporte para um conjunto expressivo de componentes.
Scicobot	- Possui material descritivo em menor quantidade que os demais robôs da tabela, destacando-se o material conceitual sobre o *software;* - Planejamento de expansão de *hardware* e *software*; - Quantidade de funcionalidades semelhantes à do Hadabot; - Estrutura de *software* e *hardware* modular; - Versão Scicobot 1 possui sensores simples; - Capacidade computacional maior que a do Hadabot e menor que a de Nanosaur e Linorobot; - Escolha de componentes considerando utilização na UFU.

Fonte: Autor

A Tabela 5.2 sintetiza os objetivos e como cada um foi atingido. Logo, tem-se que os objetivos desse estudo foram alcançados com êxito, concluindo-se que a hipótese levantada de que é possível produzir um robô modular, de baixo custo, baseado em ROS e com extensa documentação descritiva, voltada para pesquisa na UFU, campus Patos de Minas, no âmbito da robótica cooperativa, é exequível e foi desenvolvida com êxito neste trabalho. Além disso, este trabalho fornece uma boa revisão das plataformas robóticas existentes, a partir de comparações e descrição de características.

Tabela 5.2: Resumo dos objetivos e realizações deste trabalho.

Objetivos	**Como foram alcançados**
Modular	- Arquitetura de *software* modular, com bibliotecas separadas para cada parte do robô; - Chassi com possibilidade de expansão em níveis.
Baixo custo	- Uso de plataformas de baixo custo, Arduino e Raspberry.
Fácil acesso	- Material descritivo disponibilizado online.

Documentação	- Descrição detalhada do processo de criação do robô no GitHub; - Posterior disponibilização do TCC no repositório online da UFU. - Utilização de recursos com bom suporte, como: Arduino, Raspberry e micro-ROS
Potencial de aplicação em pesquisas de graduação	- Arduino Due e Raspberry Pi 3B+ possuem recursos suficientes para aplicações avançadas.
Montagem rápida e fácil	- Basta comprar os componentes da lista de materiais e efetuar a montagem, que deve demorar cerca de 1 hora.
Emprego na UFU	- Para escolha dos componentes considerou-se a disponibilidade e utilização dos mesmo durante o curso de Engenharia Eletrônica e de Telecomunicações da UFU.

Fonte: Autor

É importante salientar que a viabilidade da Scicobot depende da aplicação que se deseja desenvolver, dessa forma, caso a aplicação seja realizável a partir de um ESP32 e deseja-se utilizar uma plataforma robótica disponibilizada pela comunidade, recomenda-se a Hadabot. Em contrapartida, caso o Raspberry não possua os requisitos mínimos de recursos para a aplicação, seria indicado a avaliação do uso de Nanosaur ou Linorobot. Sendo assim, teoricamente, o emprego de Scicobot é viável para projetos de nível intermediário.

Ademais, como discutido anteriormente, o preço pode ser outro fator limitante ao tentar escolher uma plataforma robótica *open source*. A título de comparação, o preço de Hadabot com a versão dos sensores é cerca de R$ 1211,31 (HADABOT, 2022) e estima se que o de Nanosaur é entre R$ 1750,00 e R$ 2100,00, segundo levantamento feito dos componentes especificados por BONGHI (2022c). Ainda, tendo em mente os robôs da Tabela 1.1, onde treze dos dezessete possuem valor acima de R$ 3400,00, tem-se que Scicobot, que custa cerca de R$ 1408,64, oferece um bom custo-benefício. Ademais, se comparado especificamente com Hadabot, onde a diferença de preço é quase R$ 200,00, tem-se um grande ganho de poder computacional, dado principalmente pelo Raspberry, sem grande acréscimo no preço.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como tema o desenvolvimento de uma plataforma robótica de baixo custo e modular, principalmente para uso em pesquisas na UFU, campus Patos de Minas. Ao longo do trabalho foram descritos os principais conceitos teóricos, capítulo 2, onde o conteúdo relacionado a ROS foi introduzido. Posteriormente, no capítulo 3, os materiais para montagem do robô e a metodologia usada foram definidos, sendo os procedimentos metodológico descritos no capítulo 4, Desenvolvimento. Por fim, os resultados foram expostos e discutidos no capítulo 5, onde conclui-se que a proposta de implementação de um robô móvel, de baixo custo, modular, amplamente documentado e expansivo, para uso em pesquisas relacionadas a robótica cooperativa, foi atingido.

Para o desenvolvimento do Scicobot aproveitou-se de conceitos prévios e informações extraídas de outras estruturas existentes para definição do *software* e dos componentes. Então, montou-se a plataforma, criando exemplos de implementações a partir da estrutura de *software* imaginada. Por fim, validou-se o *software* e a comunicação entre dispositivos através de experimentos. Com a plataforma pronta e testada, comparou-se Scicobot com estruturas robóticas existentes, onde Scicobot se destaca ao ser considerada a aplicação na UFU, já que possibilita que alunos dessa instituição utilizem de conhecimentos prévios e componentes a disposição no laboratório, potencializando o baixo custo e facilitando o aprendizado.

Como uma plataforma nova, Scicobot tem muito o que modificar, por isso, como trabalhos futuros, sugere-se:

- Otimizar o uso da bateria: avaliar se ao retirar os LEDs das placas o robô tem economia significativa no uso de energia. Além disso, averiguar a possibilidade de estabelecer um modo dormir, onde o robô opera com poucos recursos;
- Implementar controle PID: o curso de Engenharia Eletrônica e de Telecomunicações possui uma disciplina denominada Sistemas Realimentados, nesta disciplina aprende-se o controle realimentado PID, sendo assim, pode ser interessante implementar um controlador PID para o robô, possibilitando a utilização de Scicobot nas aulas e, posteriormente, caso seja do interesse do aluno, em uma pesquisa de iniciação cientifica ou em um trabalho de conclusão de curso;
- Avaliar a utilização de um Sistema Operacional em Tempo Real, essa categoria de sistema fornece maior confiabilidade na execução de tarefas em prazos compatíveis com a ocorrência de eventos externos, um exemplo atraente nesse aspecto é Zephyr (FINOCCHIARO, 2020);

- Docker: Nanosaur e micro-ROS utilizam Docker para construir seus projetos, esta dinâmica parece possuir menor potencial de erro, isto posto, seria importante a avaliação e, talvez, implementação dessa dinâmica;
- Simulação 3D: a simulação é uma ferramenta prazerosa e importante no meio robótico, por isso, a criação de pacotes para simular Scicobot em aplicativos como CoppeliaSim e Gazebo é atrativo para novos projetistas;
- Avaliar a implementação de outras versões: apesar de Scicobot 1 ser um robô terrestre com rodas, é possível adequar as ferramentas utilizadas, principalmente ROS 2 e micro-ROS, para outras estruturas, como um robô esteira e um drone.

REFERÊNCIAS

ABUKHALIL, T. et al. *Power Optimization in Mobile Robots Using a Real-Time Heuristic.* **Journal of Robotics,** 2020. Disponível em: https://downloads.hindawi.com/journals/jr/2020/5972398.pdf. Acesso em: 11 de abril de 2021.

ACORNEJO, Alex. GitHub. Kilobot. 2013. Disponível em: https://github.com/acornejo/kilolib. Acesso em: 11 de abril de 2021.

ACS LABORATORY. Github Pages. Pheeno Construction Guide. 2017. Disponível em: https://acslaboratory.github.io/pheeno-v1/pheeno_construction_content/. Acesso em: 12 de maio de 2021.

ACS LABORATORY. Github. Pheeno ROS. 2017. Disponível em: https://github.com/ACSLaboratory/pheeno_ros. Acesso em: 12 de maio de 2021.

ALAMOUDI, Emad et al. *Open Source and Open Data Licenses in the Smart Infrastructure Era: Review and License Selection Frameworks*. 2019. (eds) Smart Infrastructure and Applications. EAI/Springer Innovations in Communication and Computing. Springer, Cham. https://doi.org/10.1007/978-3-030-13705-2_22

ALATISE, Mary B.; HANCKE, Gerhard P. *A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods.* **IEEE Access**, 2020. Disponível em: https://ieeexplore.ieee.org/document/9007654. Acesso em: 12 de maio de 2021.

ALIEXPRESS. Chegada nova programável ros robô automático navegação slam carro Turtlebot3-Burger pi3 peças a granel. 2021. Disponível em: https://pt.aliexpress.com/item/1005001608418400.html?spm=a2g0o.productlist.0.0.5da03344owtBYk&algo_pvid=b37f5b34-d759-4,9cb-be32-488c2ce6fa20&algo_expid=b37f5b34-d759-49cb-be32-488c2ce6fa20-0&btsid=0bb0623d16188804690946353eba85&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_. Acesso em 19 de abril de 2021.

ALKILABI, Muhanad H. Mohammed; NARAYAN, Aparajit; TUCI, Elio. *Cooperative object transport with a swarm of e-puck robots: robustness and scalability of evolved collective strategies.* **Swarm Intell,** 2017

ALMEIDA, Rodrigo. Ponte H com bootstrap para acionamento de motores DC. 2014. Embarcados. Disponível em: https://www.embarcados.com.br/ponte-h-bootstrap-acionamento-motores-dc/. Acesso em 23 de janeiro de 2021.

ALONSO-MORA, Javier. YouTube. Multi-robot formation control and object transport. 2017. Disponível em: https://www.youtube.com/watch?v=sDNqdEPA7pE&ab_channel=JavierAlonso-Mora. Acesso em: 12 de maio de 2021.

ALVARO, Diego; KOZIOL, Scott. ***Special Project: Husky Robot Navigation with ROS.*** 2020**.** Baylor University. Disponível em: https://repositorio.unican.es/xmlui/bitstream/handle/10902/19030/426030.pdf?sequence=1&isAllowed=y. Baylor University. Acesso em: 11 de abril de 2021.

AMSTERS, Robin; SLAETS, Peter. ***Turtlebot 3 as a Robotics Education Platform*****.** Robotics in Education, 2019. Advances in Intelligent Systems and Computing, v. 1023. Springer, Cham.

ANOOP, A. S.; KANAKASABAPATHY P.. *Review on swarm robotics platforms*. 2017. International Conference on Technological Advancements in Power and Energy, 2017, pp. 1-6. doi: 10.1109/TAPENERGY.2017.8397275.

ARDUINO. About. 2018. Disponível em: https://www.arduino.cc/en/Guide/Introduction. Acesso em: 11 de abril de 2021.

ARDUINO. Arduino Products. 2021. Disponível em: https://www.arduino.cc/en/Main/Products. Acesso em: 11 de abril de 2021.

ARDUINO. Arduino Style Guide. 2022c. Disponível em: https://docs.arduino.cc/hacking/*software*/ArduinoStyleGuide. Acesso em: 26 de janeiro de 2022.

ARDUINO. Placas Arduino. 2022a. Disponível em: https://store-usa.arduino.cc/collections/core-family. Acesso em: 22 de janeiro de 2022.

ARDUINO. Writing a Library for Arduino. 2022b. Disponível em: https://www.arduino.cc/en/Hacking/libraryTutorial. Acesso em: 26 de janeiro de 2022.

ARVIN, F. et al. *Mona: an Affordable Open-Source Mobile Robot for Education and Research.* **Journal of Intelligent & Robotic Systems**, 2019. Disponível em: https://link.springer.com/article/10.1007/s10846-018-0866-9. Acesso em: 11 de abril de 2021.

ARVIN, Farsahd; LENNOX, Barry; WATSON, Simon. *ROS Integration for Miniature Mobile Robots*. 2018. Conference: 19th Towards Autonomous Robotic Systems (TAROS) Conference. Disponível em: https://www.researchgate.net/publication/325499148_ROS_Integration_for_Miniature_Mobile_Robots. Acesso em: 23 de janeiro de 2022.

AUFRANC, Jean-Luc. Know the Differences between Raspberry Pi, Arduino, and ESP8266/ESP32. 2020. CSN. Disponível em: https://www.cnx-*software*.com/2020/03/24/know-the-differences-between-raspberry-pi-arduino-and-esp8266-esp32/. Acesso em: 11 de abril de 2021.

BENTO, Antonio Carlos. *Internet of Things: An Experiment with Residential Automation for Robotics Classes.* **International Research Journal of Management, IT & Social Sciences**, 2018. Disponível em: https://core.ac.uk/download/pdf/230598106.pdf. Acesso em: 11 de abril de 2021.

BERNAL, Manuel; CIVERA, Javier. LoCoQuad: A Low-Cost Arachnoid Quadruped Robot for Research and Education. **ArXiv**, 2020. Disponível em: https://arxiv.org/abs/2003.09025. Acesso em: 9 de maio de 2021.

BLACKROAD, Tom. GitHub. LoCoQuad: A Low-Cost Arachnoid Quadruped Robot for Research and Education. 2019. Disponível em: https://github.com/TomBlackroad/LoCoQuad. Acesso em: 11 de abril de 2021.

BONGHI, Raffaello. *Bill of Materials*. 2022c. Disponível em: https://nanosaur.ai/bill-of-materials/. Acesso em:02 de fevereiro de 2022.

BONGHI, Raffaello. Nanosaur. 2022a. Disponível em: https://github.com/rnanosaur. Acesso em: 22 de janeiro de 2022.

BONGHI, Raffaello. *ROS2 packages to drive nanosaur*. 2022b. Disponível em: https://github.com/rnanosaur/nanosaur_robot. Acesso em: 30 de janeiro de 2022.

BOSTON DYNAMICS. GitHub. Spot SDK. 2020. Disponível em: https://github.com/boston-dynamics/spot-sdk. Acesso em: 11 de abril de 2021.

BOSTON DYNAMICS. Spot. 2021. Disponível em: https://www.bostondynamics.com/spot. Acesso em: 11 de abril de 2021.

CARRILLO, M. et al. *A Bio-inspired Approach for Collaborative Exploration with Mobile Battery Recharging in Swarm Robotics.* **Bioinspired Optimization Methods and Their Applications,** 2018.

CASAS BAHIA. Jetson Nano 2gb Developer Kit Atualizado 2021. Disponível em: https://www.casasbahia.com.br/jetson-nano-2gb-developer-kit-atualizado-2021-1520218055/p/1520218055?utm_medium=Cpc&utm_source=GP_PLA&IdSku=1520218055&idLojista=120336&utm_campaign=3P_ACES_SSC&gclid=Cj0KCQiAraSPBhDuARIsAM3Js4pPrujTcIdNvo2JVv8Ay_VyEHNCCfct681s5LydlOxXOoKDUMidZgMaAiKcEALw_wcB. Acesso em: 22 de janeiro de 2021.

CHEN, Pengpeng et al. SpotMicro. 2020. Disponível em: https://spotmicroai.readthedocs.io/en/latest/. Acesso em: 12 de maio de 2021.

CHENG, Xiufeng et al. *Open collaboration between universities and enterprises: a case study on GitHub.* **Internet Research,** 2020. ISSN: 1066-2243.

CHO, HanCheol. GitHub: ros2arduino. *This library helps the Arduino board communicate with the ROS2 using XRCE-DDS.* 2020. Disponível em: https://github.com/ROBOTIS-GIT/ros2arduino. Acesso em: 23 de janeiro de 2021.

CICOLANI, Jeff. *Beginning Robotics with Raspberry Pi and Arduino*. 2018. Apress, Berkeley, CA. doi: https://doi.org/10.1007/978-1-4842-3462-4. ISBN: 978-1-4842-3462-4.

CLEARPATH ROBOTICS. *Intro To ROS.* 2015. Disponível em: http://www.clearpathrobotics.com /assets/guides/kinetic/ros/Intro%20to%20the%20Robot%20Operating%20System.html. Acesso em: 13 de maio 2021.

COCHRAN, J. K., BOKUNIEWICZ, H., YAGER, P. ***Encyclopedia of Ocean Sciences***. Academic Press, 2019, ed. 3ª. ISBN: 9780128130827

CRUZ, Otávio. GitHub build_scicobot_arduino. 2022b. Disponível em: https://github.com/SciCoBot/build_scicobot_arduino. Acesso em: 26 de janeiro de 2022.

CRUZ, Otávio. GitHub build_scicobot_rasp. 2022c. Disponível em: https://github.com/SciCoBot/build_scicobot_rasp. Acesso em: 26 de janeiro de 2022.

CRUZ, Otávio. GitHub SciCoBot. 2022e. Disponível em: https://github.com/SciCoBot. Acesso em: 18 de fevereiro de 2022.

CRUZ, Otávio. GitHub scicobot_arduino. 2022d. Disponível em: https://github.com/SciCoBot/scicobot_arduino. Acesso em: 26 de janeiro de 2022.

CRUZ, Otávio. GitHub scicobot_rasp. 2022a. Disponível em: https://github.com/SciCoBot/scicobot_rasp. Acesso em: 26 de janeiro de 2022.

CYBERBOTICS. GitHub. E-Puck *Driver for ROS2*. 2020. Disponível em: https://github.com /cyberbotics/epuck_ros2. Acesso em: 12 de maio de 2021.

DARMANIN, Rachael N.; BUGEJA, Marvin K. *A Review on Multi-Robot Systems Categorised by Application Domain.* **25th Mediterranean Conference on Control and Automation (MED),** 2017.

DE SOUSA, Raphaell Maciel et al. Desenvolvimento de um protótipo de robô móvel de baixo custo para práticas de ensino e pesquisa. **Revista Principia - Divulgação Científica e Tecnológica do IFPB.** 2018. ISSN 2447-9187. Disponível em: https://periodicos.ifpb.edu.br/index.php/principia/article/view/1249. Acesso em: 09 maio 2021.

DESCHAMPS, Filipe. Youtube: Livro Clean Code. 2019. Disponível em: https://www.youtube.com/playlist?list=PLMdYygf53DP5Sc6yFYs6ZmjsuuA2fu0TK. Acesso em: 27 de janeiro de 2022.

EHRMANN, Guido, EHRMANN, Andrea. *Suitability of common single circuit boards for Sensing and Actuating in Smart Textiles.* **Communications in Development and assembling of textile products,** 2020. Disponível em: https://journals.qucosa.de/cdatp/article/view/28/22. Acesso em: 11 de abril de 2021.

ELETRICAMENTE FALANDO. *Encoder***.** 2011. Disponível em: http://eletricamentefalando.blogspot.com/2011/10/*encoder*.html. Acesso em: 11 de abril de 2021.

ELKILANY, Basma Gh et al. *Potential Field Method Parameters Tuning Using Fuzzy Inference System for Adaptive Formation Control of Multi-Mobile Robots*. **Robotics,** 2020, Disponível em: https://doi.org/10.3390/robotics9010010. Acesso em: 12 de maio de 2021.

ERŐS, Endre et al. *A ROS2 based communication architecture for control in collaborative and intelligent automation systems.* **Procedia Manufacturing,** 2019. ISSN 2351-9789. Disponível em: https://www.sciencedirect.com/science/article/pii/S2351978920300469. Acesso em: 9 de maio 2021.

ESCOBAR, L. et al. *Multi-Robot platform with features of Cyber-physical systems for education applications.* IEEE ANDESCON, 2020, pp. 1-6.

ESCOLHAUMALICENCA. Escolher uma licença open source não precisa ser assustador. 2022. Disponível em: http://escolhaumalicenca.com.br/. Acesso em: 22 de janeiro de 2022.

ESTOLATAN, Eric et al. ***Mapping the Evolution of the Robotics Industry: a Cross Country Comparison.*** 2017. Disponível em: https://munkschool.utoronto.ca/ipl/files/2018/07/robots-final-Jul11.pdf. Acesso em: 11 de abril de 2021.

FAIRCHILD, Carol; HARMAN, Thomas L. ***ROS Robotics By Example.*** Packt Publicado Ltd. UK, 2016. ISBN 978-1-78217-519-3.

FENERICK, Jessica; VOLANTE, Carlos Rodrigo. A Evolução Das Indústrias, os Benefícios da Automação e as Perspectivas do Mercado da Robótica no Brasil e no Mundo. 2020. **Interface Tecnológica**, 2020. ISSN 2447-0864. Disponível em: https://revista.fatectq.edu.br/index.php/interfacetecnologica/article/view/805/510. Acesso em: 11 de abril de 2021.

FINOCCHIARO, Francesca. Micro-ROS: A member of the Zephyr Project and integrated into the Zephyr build system as a module!. 2020. Disponível em: https://zephyrproject.org/micro-ros-a-member-of-the-zephyr-project-and-integrated-into-the-zephyr-build-system-as-a-module/. Acesso em: 20 de fevereiro de 2022.

GARZÓN, Mario et al. *Using ROS in Multi-robot Systems: Experiences and Lessons Learned from Real-World Field Tests. In*: Koubaa A. (eds) Robot Operating System (ROS), 2017. Studies in Computational Intelligence, vol 707. Springer, Cham.

GAZEBO. Tutorial: ROS integration overview. 2014. Disponível em: http://gazebosim.org/tutorials?tut=ros_overview. Acesso em: 13 de maio 2021.

GITHUB. *About GitHub Pages.*2021. Disponível em: https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages. Acesso em: 11 de abril de 2021.

GNU. *Software Licenses*. 2022. Disponível em: https://www.gnu.org/licenses/license-list.html. Acesso em: 22 de janeiro de 2022.

GUSE, Rosana. Filipeflop. O que é Raspberry Pi?. 2020. Disponível em: https://www.filipeflop.com/blog/o-que-e-raspberry-pi/. Acesso em: 11 de abril de 2021.

HADABOT. hadabot_main. 2022. Disponível em: https://github.com/hadabot/hadabot_main/tree/master/firmware/uhadabot. Acesso em: 30 de janeiro de 2022.

HADABOT. *Learn ROS2 robotics by building a Hadabot robot*. 2021. Disponível em: https://www.hadabot.com/. Acesso em: 22 de janeiro de 2022.

HADABOT. *Purchase a Hadabot ROS 2 Robot Kit*. 2022. Disponível em: https://www.hadabot.com/purchase.html. Acesso em: 02 de fevereiro de 2022.

HENRIQUE, Luiz. **Projeto e concepção de uma plataforma robótica móvel integrada com o ROS.** 2019. UFSC, Blumenau.

HU, He. An educational Arduino robot for visual Deep Learning experiments. **International Journal of Intelligent Robotics and Applications,** 2019.

IEEE. *About*. Disponível em: https://www.ieee.org/about/at-a-glance.html. 2021. Acesso em: 11 de abril de 2021.

INTEL ISL. GitHub. OpenBot. 2020. Disponível em: https://github.com/intel-isl/OpenBot. Acesso em: 11 de abril de 2021.

JAHN, Uwe et al. *A Taxonomy for Mobile Robots: Types, Applications, Capabilities, Implementations, Requirements, and Challenges.* **Robotics***,* 2020**.** Disponível em: https://www.mdpi.com/2218-6581/9/4/109#cite. Acesso em: 12 de maio de 2021.

JALIL, Abdul. *Robot Operating System (ROS) Dan Gazebo Sebagai Media Pembelajaran Robot Interaktif.* **Journal Ilmiah,** 2018. Disponível em: http://jurnal.fikom.umi.ac.id/index.php/ILKOM/article/view/365/160. Acesso em: 11 de abril de 2021.

JAULIN, Luc. ***Mobile Robotics***. ISTE Ltd e John Wiley & Sons, 2019, ed. 2ª. London, UK

JIMENO, Juan et al. GitHub Linorobot. 2021. Disponível em: https://github.com/linorobot/linorobot. Acesso em: 20 de janeiro de 2022.

JIMENO, Juan et al. GitHub Linorobot. 2022a. Disponível em: https://github.com/linorobot/linorobot2. Acesso em: 30 de janeiro de 2022.

JIMENO, Juan et al. GitHub Linorobot. 2022b. Disponível em: https://github.com/linorobot/linorobot2_hardware. Acesso em: 30 de janeiro de 2022.

KAMAL, A. M.; HEMEL, S. H.; AHMAD, M. U. Comparison of Linear Displacement Measurements Between A Mems Accelerometer and Hc-Sr04 Low-Cost Ultrasonic Sensor. 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), 2019.

KARAMAN, Sertac et al. GitHub. *Racecar - A Powerful Platform for Robotics Research and Teaching.* 2017. Disponível em: https://mit-racecar.github.io/. Acesso em: 11 de abril de 2021.

KARLSSON ROBOTICS. Scanse Sweep. 2021. Disponível em: https://www.kr4.us/Scanse-Sweep.html. Acesso em: 11 de abril de 2021.

KASSAWAT, M.; CERVERA, E. del Pobil A.P. *Multi-robot User Interface for Cooperative Transportation Tasks. In:* Bioinspired Systems and Biomedical Applications to Machine Learning. IWINAC, 2019. Springer, Cham.

KAU, Nathan et al. *Stanford Doggo: An Open-Source, Quasi-Direct-Drive Quadruped.* **IEEE,** 2019.

KAU, Nathan. GitHub. Stanford Doggo Project. 2018. Disponível em: https://github.com/Nate711/StanfordDoggoProject. Acesso em: 11 de abril de 2021.

KRONAUER, Tobias et al. *Latency Overhead of ROS2 for Modular Time-Critical Systems.* ***ArXiv,*** *2021.* Disponível em: https://arxiv.org/pdf/2101.02074.pdf. Acesso em: 11 de abril de 2021.

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

KURNIAWAN, Agus. Cap. *Introduction to Raspberry Pi.* In: ***Raspbian OS Programming with the Raspberry Pi.*** Apress, Berkeley, CA. 2019.

KURNIAWAN, Agus. Introduction to Raspberry Pi. **Raspbian OS Programming with the Raspberry Pi,** 2018.

KÜSTER, Dennis; SWIDERSKA, Aleksandra; GUNKEL, David. *I saw it on YouTube! How online videos shape perceptions of mind, morality, and fears about robots.* **New Media & Society,** 2020. Disponível em: https://journals.sagepub.com/doi/pdf/10.1177/1461444820954199. Acesso em: 11 de abril de 2021.

LANGE, Ralph. Micro-ROS – bringing the most popular robotics middleware onto tiny microcontrollers. 2021. Disponível em: https://www.bosch.com/stories/bringing-robotics-middleware-onto-tiny-microcontrollers/. Acesso em: 23 de janeiro de 2022.

LEMOS, Ana C. R.; MENDONÇA, Kennya R. Plataforma Robótica Móvel para Estudos de Localização, Mapeamento, Navegação e Percepção 3D Baseados em um Sensor RGB-D. *In*: XXII Congresso Brasileiro de Automática, 2020. ISSN 2525-8311.

LI, Yunwang et al. *Modeling and Kinematics Simulation of a Mecanum Wheel Platform in RecurDyn.* **Journal of Robotics,** 2018. Disponível em: https://downloads.hindawi.com/journals/jr/2018/9373580.pdf. Acesso em: 11 de abril de 2021.

LIAO, M.; WANG, D.; Yang, H. *Deploy Indoor 2D Laser SLAM on a Raspberry Pi-Based Mobile Robot.* 11th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2019, pp. 7-10, doi: 10.1109/IHMSC.2019.10097.

LOPEZ-RODRIGUEZ, Francisco M.; CUESTA, Federico. *An Android and Arduino Based Low-Cost Educational Robot with Applied Intelligent Control and Machine Learning.* **Applied Sciences**, 2021. Disponível em: https://www.mdpi.com/2076-3417/11/1/48/htm. Acesso em: 11 de abril de 2021.

MANZOOR, Mariam. Arduino with Infrared Sensor. **Medium,** 2020. Disponível em: https://medium.com/illumination/arduino-with-infrared-sensor-48ad4415f320. Acesso em: 12 de maio de 2021.

MARÍN, Leonardo. *Modular Open Hardware Omnidirectional Platform for Mobile Robot Research.* IEEE, 2018.

MARTIN, Robert C. Arquitetura Limpa. Alta Books, 2019, ed. 1ª. ISBN: 978-8550804606.

MARUYAMA, Yuya; KATO, Shinpei; AZUMI, Takuya. *Exploring the performance of ROS2.* 13th International Conference on Embedded *Software*, 2016. https://doi.org/10.1145/2968478.2968502

MATHWORKS. *Get Started with ROS.* 2021. Disponível em: https://www.mathworks.com/help/ros/ug/get-started-with-ros.html. Acesso em: 13 de maio 2021.

MICRO XRCE-DDS. eProsima Micro XRCE-DDS. 2018. Disponível em: https://micro-xrce-dds.docs.eprosima.com/en/latest/. Acesso em: 9 de maio de 2021

MICRO_ROS_ARDUINO. *Micro-ros library for arduino*. Última modificação em 2022. Disponível em: https://github.com/micro-ROS/micro_ros_arduino. Acesso em: 23 de janeiro de 2022.

MICRO-ROS. Example publisher. Última modificação em 2021a. Disponível em: https://github.com/micro-ROS/micro_ros_arduino/blob/foxy/examples/micro-ros_publisher/micro-ros_publisher.ino. Acesso em: 26 de janeiro de 2022.

MICRO-ROS. Examples subscriber. 2021b. Última modificação em Disponível em: https://github.com/micro-ROS/micro_ros_arduino/blob/foxy/examples/micro-ros_subscriber/micro-ros_subscriber.ino. Acesso em: 26 de janeiro de 2022.

MICRO-ROS. Home: micro-ROS puts ROS 2 onto microcontrollers. 2022. Disponível em: https://micro.ros.org/. Acesso em: 23 de janeiro de 2022.

MICROSOFT. Arquivos de header (C++). 2021. Disponível em: https://docs.microsoft.com/pt-br/cpp/cpp/header-files-cpp?view=msvc-170&viewFallbackFrom=msvc-170%5B. Acesso em: 27 de janeiro de 2022.

MIKE4192. GitHub. *Spot Micro Quadruped*. 2020. Disponível em: https://github.com/mike4192/spotMicro. Acesso em: 11 de abril de 2021.

MILLARD, Alan G. et al. *The Pi-puck extension board: a Raspberry Pi interface for the e-puck robot platform.* **International Conference on Intelligent Robots and Systems (IROS)**, 2017.

MONDADA, F et al. E-puck: *Main Menu*. 2010. Disponível em: http://www.e-puck.org/index.php?option=com_content&view=article&id=2&Itemid=8. Acesso em: 12 de maio de 2021.

MORENO, Andrés; PAEZ, Daniel. *Performance evaluation of ROS on the Raspberry Pi platform as OS for small robots.* 2017. **Tekhnê da Universidad Distrital Francisco José de Caldas.** ISSN 1692-8407.

NAKAMOTO, Naohiro; KOBAYASHI, Hiroyuki. *Development of an Open-source Educational and Research Platform for Autonomous Cars.* **IEEE**, 2019.

NASA. GitHub. Rover Based on the Rovers on Mars. 2018. Disponível em: https://github.com/nasa-jpl/open-source-rover. Acesso em: 11 de abril de 2021.

NEDJAH, Nadia; JUNIOR, Luneque Silva. *Review of methodologies and tasks in swarm robotics towards standardization.* **Swarm and Evolutionary Computation,** 2019.

NIKU, Saeed B. ***Introduction to Robotics.*** 3. ed. John Wiley & Sons Ltd: 2020.

NOORI, Farzan M et al. *On 3D simulators for multi-robot systems in ROS: MORSE or Gazebo?*. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR), 2017.

NOTT, Christopher. *SSH Remote Access.* Study. 2016. Disponível em: https://study.com/academy/lesson/ssh-remote-access.html. Acesso em: 12 de maio de 2021.

NUNES, Guilherme. **Controle de Iluminação e Temperatura pela Plataforma Arduino via Celular.** 2015. Universidade Tecnológica Federal do Paraná. Disponível em: http://repositorio.utfpr.edu.br/jspui/bitstream/1/16893/1/PG_COAUT_2015_2_04.pdf. Acesso em: 11 de abril de 2021.

PALAKOLLU, S.M. (2021) Introduction to the Linux Environment. In: Practical System Programming with C. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6321-1_1

PANDEY, Anish; PANDEY, Shalini; DR, Parhi. *Mobile Robot Navigation and Obstacle Avoidance Techniques: A Review.* **International Journal of Robotics and Automation,** 2017. Disponível em: https://www.researchgate.net/publication/317101750_Mobile_Robot_Navigation_and_Obstacle_Avoidance_Techniques_A_Review. Acesso em: 12 de maio de 2021.

PETITTI, Antonio et al. *Decentralized Motion Control for Cooperative Manipulation with a Team of Networked Mobile Manipulators. In*: 2016 IEEE International Conference on Robotics and Automation (ICRA). 2016.

RASPBERRY PI. Products. 2021. Disponível em: https://www.raspberrypi.org/products/. Acesso em: 11 de abril de 2021.

RCLCPP. *C++ ROS Client Library API*. 2022. Disponível em: https://docs.ros2.org/latest/api/RCLCPP/. Acesso em: 23 de janeiro de 2022.

RCLPY. *RCLPY provides the canonical Python API for interacting with ROS 2*. 2022. Disponível em: https://docs.ros2.org/latest/api/RCLPY/index.html. Acesso em: 23 de janeiro de 2022.

REALPARS. YouTube. *What is the Difference between Absolute and Incremental Encoders?*. 2019. Disponível em: https://www.youtube.com/watch?v=-Qk--Sjgq78&ab_channel=RealPars. Acesso em: 11 de abril de 2021.

REZECK, Paulo et al. HeRo: An open platform for robotics research and education. 2017. Disponível em: https://www.verlab.dcc.ufmg.br/hero/. Acesso em: 12 de maio de 2021.

RHOADES, Benjamin B.; SABO, Jeremy P.; CONRAD, James M. *Enabling a National Instruments DaNI 2.0 Robotic Development Platform for the Robot Operating System.* 2017. **IEEE**, 2017. ISSN: 1558-058X. Disponível em: https://ieeexplore.ieee.org/document/7925293/authors#authors. Acesso em: 11 de abril de 2021.

RIBEIRO, A., LOPES, G. *Learning Robotics: a Review.* **Curr Robot Reports,** 2020. Disponível em: https://link.springer.com/article/10.1007/s43154-020-00002-9#citeas. Acesso em: 12 de maio de 2021.

RICHARDSON, Tristan et al. ***Virtual Network Computing.*** IEEE Internet Computing. Volume 2, 1998. Disponível em: https://quentinsf.com/publications/virtual-network-computing/vnc-ieee.pdf. Acesso em: 12 de maio de 2021.

ROBERT, John. *Hands-On Introduction to Robot Operating System (ROS).* 2020. Disponível em: https://trojrobert.github.io/hands-on-introdution-to-robot-operating-system(ros)/. Acesso em: 13 de maio 2021.

ROBOTICS BACK-END. *What is ROS?.* 201X. Disponível em: https://roboticsbackend.com/what-is-ros/. Acesso em: 12 de maio de 2021.

ROLDÁN, Juan Jesús et al. *Heterogeneous Multi-Robot System for Mapping Environmental Variables of Greenhouses.* Sensors, 2016. Disponível em: https://www.mdpi.com/1424-8220/16/7/1018/htm. Acesso em: 12 de maio de 2021.

ROMADON, Z. T. et al. Pose Estimation on Soccer Robot using Data Fusion from *Encoder*s, Inertial Sensor, and Image Data. **IEEE,** 2019.

ROMÁN, David Pérez. Mobile Robot Powered by ROS. 2017. Disponível em: https://zaguan.unizar.es/record/106412/files/TAZ-TFG-2021-2176.pdf?version=1. Acesso em: 25 de janeiro de 2022.

ROS. *ROS developer's guide.* 201X. Disponível em: http://wiki.ros.org/DevelopersGuide. Acesso em: 12 de maio de 2021.

ROS. *Understanding ROS Nodes.* 201X. Disponível em: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes. Acesso em: 13 de maio 2021.

ROS. *Understanding ROS Services and Parameters.* 2020. Disponível em: http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams. Acesso em: 13 de maio 2021.

ROS. *Understanding ROS Topics.* 2019. Disponível em: http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics. Acesso em: 11 de abril de 2021.

ROS1_BRIDGE. *Pacote ROS 2 que fornece comunicação bidirecional entre ROS 1 e ROS 2*. Última modificação em 2022. Disponível em: https://github.com/ros2/ros1_bridge. Acesso em: 23 de janeiro de 2022.

ROS2. ROS 2. Documentation. 2022a. Disponível em: https://docs.ros.org/en/foxy/index.html. Acesso em: 22 de janeiro de 2021.

ROS2. ROS 2. Tutorial. 2022b. Disponível em: https://docs.ros.org/en/foxy/Tutorials.html. Acesso em: 31 de janeiro de 2021.

ROS-INDEX. Repository teleop_twist_keyboard. 2021. Disponível em: https://index.ros.org/r/teleop_twist_keyboard/#foxy. Acessado em: 26 de janeiro de 2022.

RUBIO, Francisco; VALERO, Francisco; LLOPIS-ALBERT, Carlos. *A review of mobile robots: Concepts, methods, theoretical framework, and applications.* **International Journal of Advanced Robotic Systems,** 2019. Disponível em: https://journals.sagepub.com/doi/10.1177/1729881419839596. Acesso em: 12 de maio de 2021.

SALAROLLI, Pablo; MATTA V. da R.; CUADROS, M. A. de S. L. Fusão dos Dados do Dead Reckoning e do Giroscópio Usando o Filtro de Kalman Estendido Aplicado à Localização de uma Cadeira de Rodas Motorizada. **XIII Simpósio Brasileiro de Automação Inteligente,** 2017**.** Disponível em: https://www.ufrgs.br/sbai17/papers/paper_445.pdf. Acesso em: 11 de abril de 2021.

SANTOS, André; CUNHA, Alcino; MACEDO, Nuno. *Static-Time Extraction and Analysis of the ROS Computation Graph.* **Third IEEE International Conference on Robotic Computing (IRC)**, 2019. Disponível em: https://ieeexplore.ieee.org/abstract/document/8675667. Acesso em: 11 de abril de 2021.

SANTOS, Henrique A. Uma Avaliação da Precisão do Sistema de Localização de Robôs Móveis com Aplicação da Técnica de Fusão Sensorial. III Brazilian Humanoid Robot Workshop and IV Brazilian Workshop on Service

Robotics. 2020. Disponível em: https://www.researchgate.net/publication/343646784_Evaluation_of_mobile_robot_localization_system%27s_accuracy_based_on_sensor_data_fusion_technique. Acessado em: 12 de maio de 2021.

SANTOS, Rafael Marcelo; FLÔR, Daniela Eloise. RASPCAR – CARRO ROBÓTICO GUIADO REMOTAMENTE RASPCAR - REMOTELY GUIDED ROBOTIC CAR. **Revista Mundi Engenharia, Tecnologia e Gestão,** 2019. ISSN 2525-4782.

SATTAYASOONTHORN, Preedipat; SUTHAKORN, Jackrit. *Battery management for rescue robot operation.* **IEEE,** 2016.

SAYED, A. S.; AMMAR, H. H.; SHALABY, R. *Centralized Multi-agent Mobile Robots SLAM and Navigation for COVID-19 Field Hospitals.* 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), 2020. Disponível em: https://ieeexplore.ieee.org/abstract/document/9257919. Acesso em: 10 de maio de 2021.

SEBESTA, Robert W. Conceitos de Linguagem de Programação. Bookman, 2018, ed 11ª. ISBN: 9780133943023

SHABALINA, Ksenia; SAGITOV, Artur; MAGID, Evgeni. *Comparative Analysis of Mobile Robot Wheels Design.* **IEEE**, 2018.

SHIBUKAWA, Guilherme Hideki. **Modelagem matemática de robô SCARA de acionamento elétrico com enfoque na prevenção de infecções hospitalares por meio da automatização de luminárias do campo cirúrgico.** 2018. Engenharia de Controle e Automação - UNICESUMAR.

SIEFKE, Lennart et al. Open *Robotic Systems of Systems Based on a Decentralized Service-Oriented Architecture*. Robotics 2020, 9, 78. https://doi.org/10.3390/robotics9040078

SINGH, Anmol; ROHILLA, Yogesh. *Design and Control of Two-wheeled Self-Balancing Robot using Arduino.* **IEEE,** 2020.

SKOROSPELOV, Leonid. Multibot. 2015. Disponível em: http://fontysarcr.blogspot.com/2016/04/. Acesso em: 24 maio de 2021.

SSH. *Secure Shell - Home Page.* 201X. Disponível em: https://www.ssh.com/academy/ssh. Acesso em: 12 de maio de 2021.

STM. Intel NUC NUC6i5SYB Intel Core i5-6260U MOTHERBOARD. 2021. Disponível em: https://www.surplustechmart.com/computers-networking/computer-servers-components/boards/intel-nuc-nuc6i5syb-intel-core-i5-6260u-motherboard.html. Acesso em: 11 de abril de 2021.

TAUFIQQUROHMAN, M.; SARI, N. F. Odometry Method and Rotary *Encoder* for Wheeled Soccer Robot. IOP Conference Series: Materials Science and Engineering. 2018. Disponível em: https://iopscience.iop.org/article/10.1088/1757-899X/407/1/012103. Acessado em: 12 de maio de 2021.

THÁCIK, Milan; BREZINA, Adam; JADLOVSKÁ, Slávka. *Design of a Prototype for a Modular Mobile Robotic Platform.* **16th IFAC Conference on Programmable Devices and Embedded Systems PDES,** 2019.

THE ROBOTICS BACK-END. *ROS1 vs ROS2, Practical Overview For ROS Developers.* 2021. Disponível em: https://roboticsbackend.com/ros1-vs-ros2-practical-overview/. Acesso em: 11 de abril de 2021.

THORNBERRY, Evan; WHITE, Phil. *GitHub and Jekyll for Publishing GIS Workshop Content***. Open Journal Systems,** 2020. Disponível em: https://openjournals.uwaterloo.ca/index.php/acmla/article/view/3463. Acesso em: 11 de abril de 2021.

Torabbeigi, M.; Lim, G.J.; Kim, S.J. Drone Delivery Scheduling Optimization Considering Payload-induced Battery Consumption Rates. **Journal of Intelligent & Robotic Systems,** 2020.

TSARDOULIAS, Emmanouil; MITKAS, Pericles. *Robotic frameworks, architectures and middleware comparison.* **ArXiv**, 2017. Disponível em: https://arxiv.org/pdf/1711.06842.pdf. Acesso em: 11 de abril de 2021.

UFSC. *UNIX E LINUX*. 2000. Disponível em: http://www.inf.ufsc.br/~j.barreto/cca/sisop/unixe.htm#:~:text=Sistema%20Linux,especial%20seu%20Idealizador%2C%20Linus%20Torvalds.. Acesso em: 12 de maio de 2021.

UTOMO, Prayudi; FALAHAH. *Building Serverless Website on GitHub Pages.* IOP Conference Series: Materials Science and Engineering. 2020. Disponível em: https://iopscience.iop.org/article/10.1088/1757-899X/879/1/012077. Acesso em: 11 de abril de 2021.

VAIDYA, Neerad; RUGHANI, Parag. *A forensic study of Tor usage on the Raspberry Pi platform using open source tools.* **Computer Fraud & Security,** 2020.

VALVANO, Jonathan W. Introduction to Arm(r) Cortex -M Microcontrollers: Introduction to Arm(r) Cortex(tm)-M Microcontrollers. Createspace, 2014. ISBN: 9781477508992.

VAUTIER, Ulysse et al. *Development and test of an open source autonomous sailing robot with accessibility, generality and extendability.* International Robotic Sailing Conference, 2018. Disponível em: https://www.researchgate.net/publication/328289438_Development_and_Test_of_an_Open_Source_Autonomous_Sailing_Robot_with_Accessibility_Generality_and_Extendability. Acesso: 12 de maio 2021.

VERLAB. GitHub. HeRo: An Open Platform for Robotics Research and Education. 2017. Disponível em: https://github.com/verlab/hero_common. Acesso em: 11 de abril de 2021.

VERMA, Pulkit et al. *Loosely Coupled Payload Transport System with Robot Replacement.* **ArXiv,** 2019. Disponível em: https://arxiv.org/pdf/1904.03049.pdf. Acesso em 12 de maio de 2021.

VERMA, Pulkit. YouTube. *Loosely Coupled Payload Transport System with Robot Replacement.* Disponível em: https://www.youtube.com/watch?v=-6ivGT3dOQw&ab_channel=PulkitVerma. 2019. Acesso em: 12 de maio de 2021.

VIARHEICHYK, Igor. *Embedded Programming with Modern C++.* Packt Publishing, 2020. ISBN: 9781838821043

WILLOWGARAGEVIDEO. YouTube. Rviz. 2010. Disponível em: https://www.youtube.com/watch?v=i--Sd4xH9ZE&t=27s&ab_channel=WillowGaragevideo. Acesso em: 13 de maio 2021.

WILSON, Sean et al. Pheeno, *A Versatile Swarm Robotic Research and Education Platform.* **IEEE Robotics and Automation Letters,** 2015.

WYROBEK, Keenan. *IEEE Spectrum. The Origin Story of ROS, the Linux of Robotics.* 2017. Disponível em: https://spectrum.ieee.org/automaton/robotics/robotics-*software*/the-origin-story-of-ros-the-linux-of-robotics. Acesso em: 12 de maio de 2021.

XBOT. RoboDeck. 2017. Disponível em: http://www.xbot.com.br/educacional/robodeck/. Acesso em: 11 de abril de 2021.

YOUTUBE. ***Do You Love Me?.*** 2020. Disponível em: https://www.youtube.com/watch?v=fn3KWM1kuAw&ab_channel=BostonDynamics. Acesso em: 11 de abril de 2021.

YOUTUBE. ***Hey Buddy, Can You Give Me a Hand?.*** 2018. Disponível em: https://www.youtube.com/watch?v=fUyU3lKzoio&ab_channel=BostonDynamics>. Acesso em: 11 de abril de 2021

YU, Xinguo et al. *Design and Implementation of Platform of Educational Mobile Robots Based on ROS.* **Recent Developments in Mechatronics and Intelligent Robotics**, 2017.

ZHANG, Yun et al. *Detecting similar repositories on GitHub.* **IEEE 24th International Conference on *Software* Analysis, Evolution and Reengineering (SANER)**, 2017.

ZHMUD, V. A et al. *Application of ultrasonic sensor for measuring distances in robotics.* **Journal of Physics: Conference Series,** Volume 1015, Issue 3, 2018. Disponível em: https://iopscience.iop.org/article/10.1088/1742-6596/1015/3/032189. Acesso em: 12 de maio de 2021.