UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Turtle bot: "Relâmpago marquinhos"

Raphael Leite Diniz
Ramon M. P. Mariano
Gustavo Orlando Boçon Huziy

5 de dezembro de 2023

Conteúdo

1 Introdução

A evolução da robótica contemporânea, impulsionada por avanços notáveis, abrange uma diversidade de temas fascinantes, sendo os robôs autônomos um destaque notável. Esses inovadores autômatos encontram aplicações abrangentes, desde assistência residencial e exploração espacial até agricultura, indústria e veículos autônomos. No âmago dessas aplicações está a habilidade essencial de mapear o ambiente, desempenhando um papel crucial na localização eficiente em contextos variados.

O protótipo em questão incorpora diversas tecnologias contemporâneas, começando pelo uso de sensores infravermelhos, seguido pelo processamento desses dados utilizando o ROS (Robot Operating System) e, por fim, a implementação de motores DC para a locomoção autônoma.

O ROS, que atua como a "alma"do robô, é o sistema responsável por gerenciar todas as funcionalidades do projeto, desde a leitura dos dados infravermelhos até a tomada de decisões, culminando na interação com o nível mais baixo para controlar os motores DC, que é executada por outro microcontrolador.

O microcontrolador ESP32 assume a responsabilidade do controle em nível mais baixo, determinando a velocidade dos motores e interpretando as direções enviadas pelo ROS. Além disso, é nesse componente que ocorre a captação dos dados do acelerômetro/giroscópio, que são posteriormente enviados para o ROS.

A função principal deste projeto é não apenas permitir a locomoção autônoma do robô, mas também capacitar a geração de um mapa detalhado do ambiente explorado, que pode ser salvo para referências futuras.

Ao longo deste relatório, serão apresentadas a linha de raciocínio e o desenvolvimento que culminaram na criação desse protótipo. Desde os princípios fundamentais até a integração de componentes, cada fase do processo será explorada em detalhes, proporcionando uma compreensão aprofundada do impacto potencial dessa tecnologia na vanguarda da robótica autônoma.

2 Revisão Bibliográfica

- Razbotics (2018) fornece um guia detalhado sobre a implementação do Robot Operating System (ROS) em sistemas distribuídos, destacando a conexão entre Raspberry Pi e PC por meio de uma rede local (LAN). O autor oferece insights práticos e orientações valiosas para estabelecer sistemas distribuídos utilizando o ROS [11].

- Yahboom Technology (2023) apresenta o YDLIDAR X3/X3 Pro Lidar TOF 360° Scanning Ranging Sensor, enfatizando seu suporte para ROS1 e ROS2. O código-fonte e informações técnicas estão disponíveis no repositório do GitHub, proporcionando recursos valiosos para desenvolvedores interessados em integrar esse sensor em projetos ROS [13].

- O trabalho de zhanyiaini (2022) apresenta um abrangente *Table of Contents* disponível no GitHub para o YDLidar-SDK. Essa referência fornece uma visão geral estruturada do SDK, auxiliando os desenvolvedores na navegação eficiente pelos recursos e documentação disponíveis [3].

- RoboticArts (2020) contribui para a comunidade ROS com o pacote `ros_imu_bno055`. Este pacote é projetado para integração fácil e confiável do sensor de IMU BNO055 em sistemas ROS, simplificando o desenvolvimento de robôs com capacidades de orientação avançadas [12].

- O projeto "articubot_one"de Josh Newans (2023) disponível no GitHub oferece uma implementação prática e aberta de um robô móvel. Os desenvolvedores podem explorar e adaptar o código para suas próprias aplicações, aproveitando as contribuições e conhecimentos do autor [7].

- A documentação oficial da ROS Wiki (2020) fornece instruções detalhadas para a instalação do ROS Melodic no sistema operacional Ubuntu. Essa referência é essencial para desenvolvedores que desejam configurar um ambiente de desenvolvimento ROS em suas máquinas [2].

- A documentação oficial da ROS Wiki (2018) disponibilizou a principal ferramenta de vizualização de mapas, e interação com robótica, disponível atualmente, é de fácil utilização e tem grandes funcionalidades.[17]

- A ROS Wiki (2020) também oferece informações cruciais sobre o pacote `map_server`. Esta ferramenta é fundamental para o mapeamento em ambientes robóticos, sendo uma referência valiosa para desenvolvedores que trabalham com sistemas de navegação autônomos [1].

- O guia de instalação do Ubuntu Desktop (2023) fornecido pelo Ubuntu é uma fonte essencial para desenvolvedores que buscam configurar um ambiente de desenvolvimento baseado em Ubuntu para projetos ROS [14].

- Josh (2022) compartilha experiências e conhecimentos práticos sobre a incorporação de um Lidar em um robô móvel. Seu artigo "Making a Mobile Robot #8 - Adding a Lidar"fornece insights valiosos para desenvolvedores que estão expandindo as capacidades de seus robôs [6].

- The Robotics Back-End (2023) oferece uma série abrangente de tutoriais ROS, cobrindo vários aspectos do desenvolvimento robótico. Esses tutoriais são uma fonte valiosa de informações para desenvolvedores iniciantes e avançados no ecossistema ROS [5].

- ACKERMAN e GUIZZO (2017) compartilham uma experiência prática com o TurtleBot 3, um robô poderoso para aprendizado de ROS. O artigo "Hands-on With TurtleBot 3, a Powerful Little Robot for Learning ROS"fornece uma visão detalhada [4].
- A ROS Wiki (2018) cirou uma ferramenta de visualização de sistema para poder abrir uma interface gráfica, onde é possível ver todos os nós do sistema e como eles interagem entre si, esta ferramenta é muito útil não só para entender melhor o que está acontecendo como também para debuggar o sistema. [16]
- A ROS Wiki (2015) forneceu o pacote utilizado para controlar o robô manualmente, o teleop_twist_keyboard. O artigo é bem direto quanto a como utilizar e como ele funciona. [15].
- Pela quantidade de arquivos e linhas de código empregadas no projeto, segue o repositório referenciado criado pelos autores para os nós do ROS [10].
- Segue, também, o repositório referenciado criado pelos autores para toda a lógica do PID empregado no ESP32, o baixo nivel do software utilizado. [8].
- Foi apresentado um bug com o ESP32 saindo de controle sendo necessário reiniciá-lo para que volte ao normal, então a solução encontrada foi um script shell que desliga a porta USB e liga denovo de forma a reiniciar o ESP32.[9]

3 Hardware

3.1 Estrutura

Esta seção do relatório tem como objetivo especificar os hardwares utilizados na montagem do "Turtlebot: Relâmpago Marquinhos". Para melhor compreensão, a presente seção será dividida em estrutura, sensores e microcontroladores.

3.1.1 Atuador: Motor DC

Na estrutura do robô, foi utilizado um chassi e um kit Arduino, como mostrado na Figura 1, com motor DC em duas rodas.

Figura 1: Estrutura do Robô.

Fonte: Google imagens (2023).

3.1.2 Sistema de Baterias

O sistema de baterias do "Turtlebot: Relâmpago Marquinhos"é composto por seis baterias de íon de lítio (Li-Po) de 3,6V, conforme mostrado na Figura 2. Essas baterias são utilizadas para fornecer energia ao sistema, alimentando os diversos componentes eletrônicos e motores.

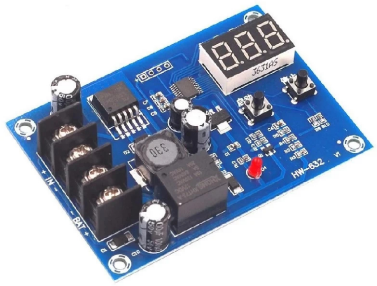
Figura 2: Baterias de Íon de Lítio

Fonte: Google imagens (2023).

3.1.3 Módulo de Carregamento XH-M603

O módulo de carregamento XH-M603, como apresentado na Figura 3, é um componente fundamental no sistema de energia do "Turtlebot: Relâmpago Marquinhos". Ele faz a gestao do carregamento, e a proteçao do circuito em caso de curtos.

Figura 3: Módulo de Carregamento XH-M603.



Fonte: Google imagens (2023).

3.2 Sensores

3.2.1 YDLIDAR

O sensor YDLIDAR X3, representado na Figura 4, é um sensor lidar (Light Distance and Range). Ele fornece capacidades de detecção e mapeamento baseadas em laser, sendo crucial para a navegação autônoma do robô.

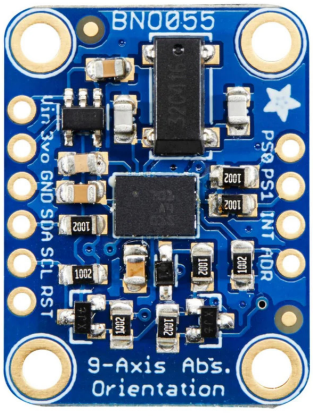
Figura 4: YDLIDAR X3.

Fonte: Google imagens (2023).

3.2.2 BNO055

O sensor BNO055, ilustrado na Figura 5, é um sensor de movimento que combina acelerômetro, giroscópio e magnetômetro. Ele é utilizado para obter informações precisas sobre a orientação do robô no espaço.

Figura 5: BNO055.

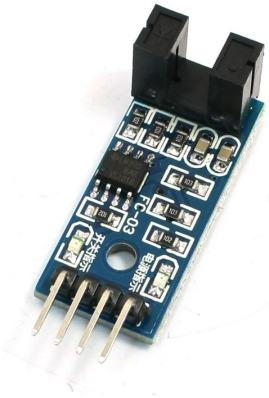


Fonte: Google imagens (2023).

3.2.3 Encoders Incrementais

Os encoders incrementais, apresentados na Figura 6, são dispositivos utilizados para medir a rotação das rodas do robô. Eles desempenham um papel fundamental no controle de movimento e na obtenção de feedback para garantir uma navegação precisa.

Figura 6: Encoders Incrementais.



Fonte: Autores (2023).

3.3 Microcontroladores

3.3.1 Raspberry Pi 4 8GB

O Raspberry Pi 4 8GB, como mostrado na Figura 7, é um computador de placa única (Single Board Computer - SBC) popular e poderoso, amplamente utilizado em projetos variados.

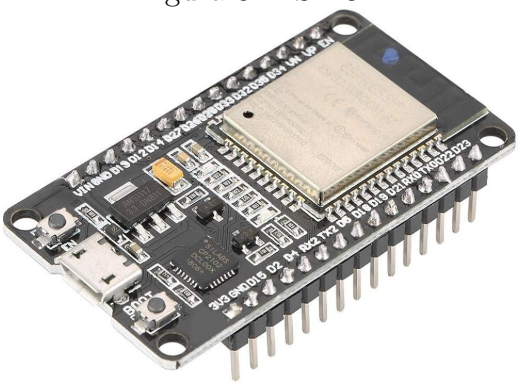
Figura 7: Raspberry Pi 4 8GB.

Fonte: Autores (2023).

3.3.2 ESP-32

O ESP-32, como mostrado na Figura 8, é um microcontrolador de baixo custo e alto desempenho amplamente utilizado em projetos de IoT.

Figura 8: ESP-32.



Fonte: Autores (2023).

4 Placa-Mãe

4.1 KiCad 7.0

O KiCad 7.0 é uma ferramenta poderosa e de código aberto para design de circuitos eletrônicos e PCB (Placa de Circuito Impresso). Ele oferece uma interface intuitiva e recursos avançados para facilitar a criação de projetos eletrônicos complexos.

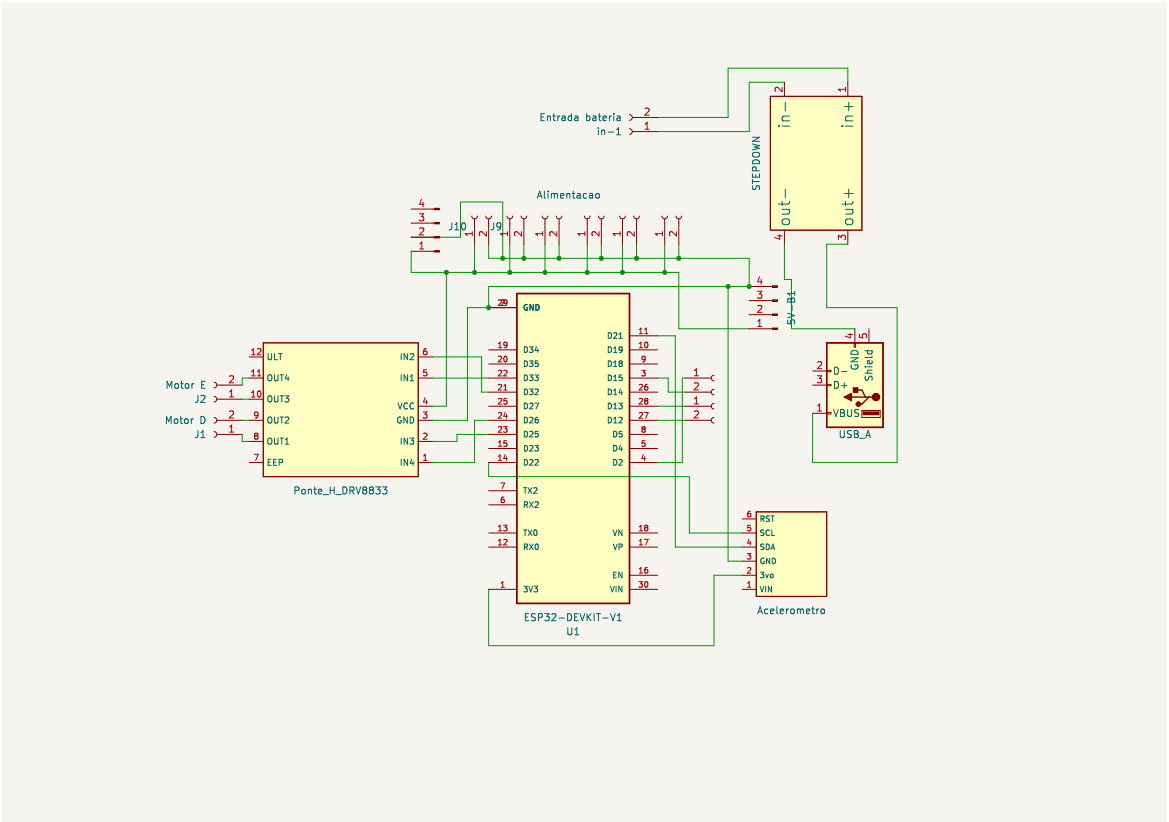
A placa mãe do "Turtlebot"Relâmpago-Marquinhos foi projetada utilizando o KiCad 7.0. No KiCad, foi possível inserir nossos hardwares, criar componentes personalizados, bem como adicionar pinos de energia e configurar o módulo de alimentação.

4.2 Esquemático Eletrônico

Um esquemático eletrônico é uma representação visual de um circuito elétrico, utilizado para descrever a conexão entre componentes eletrônicos e suas interações no sistema. Essa representação é essencial no desenvolvimento de projetos, fornecendo uma visão clara e organizada da estrutura do circuito.

A Figura 9 mostra a representaçao do esquematico do "TurtleBot:Relampago-marquinhos", no software KiCad 7.0.

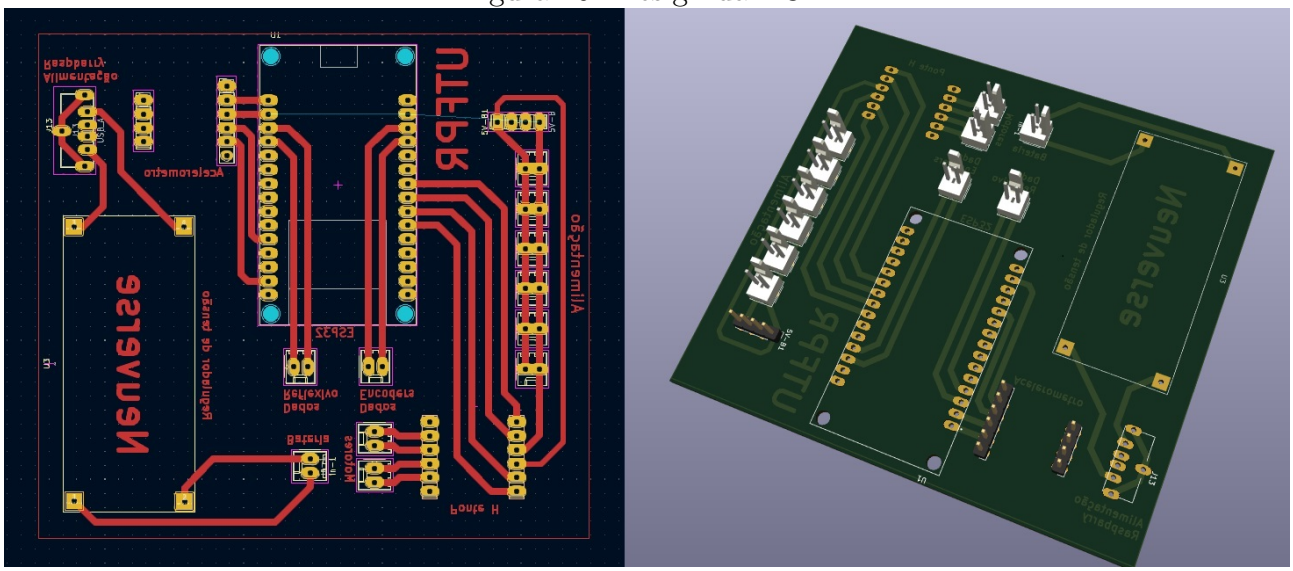
Figura 9: Esquematico no KiCad 7.0.



Fonte: Autores (2023).

A representação , representada na Figura 10, é a materialização do projeto esquemático. Este design proporciona a disposição física dos componentes eletrônicos na placa, otimizando conexões e garantindo a eficiência do circuito.

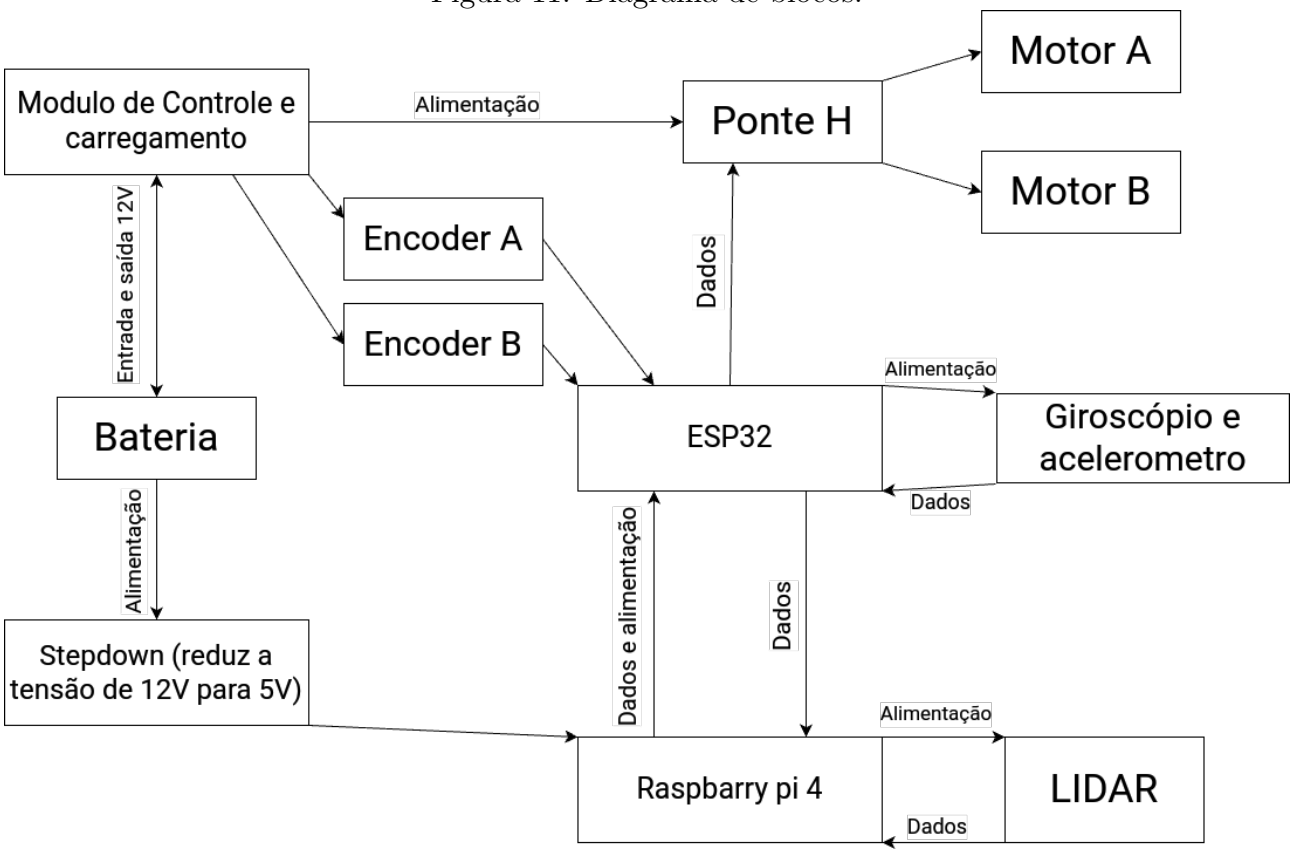
Figura 10: Design da PCB.



Fonte: Autores (2023).

A Figura 11 mostra o diagrama de blocos, aqui é estabelecida as conexões físicas entre os hardwares do robo.

Figura 11: Diagrama de blocos.



Fonte: Autores (2023).

5 Lista de materiais

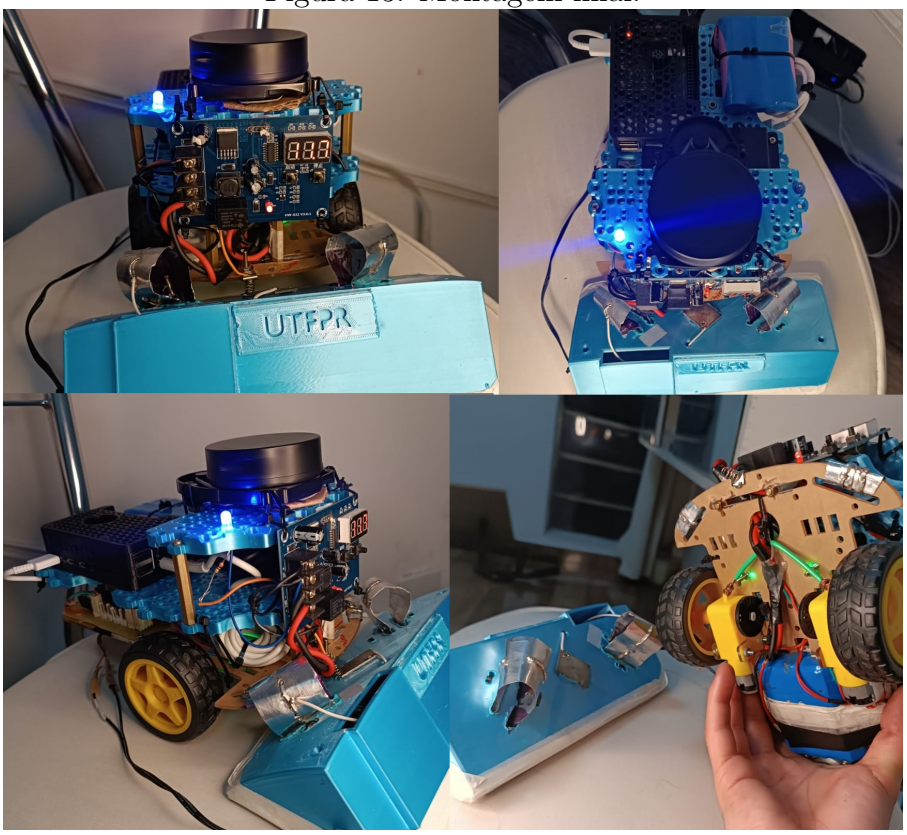
Figura 12: Lista de materiais e valores.

	Materiais utilizados	Quantidade	Valor unitário	Valor total
1	Kit (motor, roda, caixa de redução)	2	R$ 14.90	R$ 29.80
2	Chassi acrílico com roda boba giratória	1	R$ 24.99	R$ 24.99
3	Modulo sensor de velocidade (encoder)	2	R$ 7.90	R$ 15.80
4	ESP32	1	R$ 52.00	R$ 52.00
5	Raspbarry Pi 4 8GB	1	R$ 1,030.00	R$ 1,030.00
6	Ponte H DRV8833	1	R$ 11.90	R$ 11.90
7	YDLIDAR V3	1	R$ 320.00	R$ 320.00
8	Modulo regulador de tensão XL4015	1	R$ 26.90	R$ 26.90
9	Modulo controlador de carga XHM632	1	R$ 46.90	R$ 46.90
10	Baterias 3,7V 1200mAh	6	R$ 25.00	R$ 150.00
11	Módulo sensor de orientação BNO55	1	R$ 349.20	R$ 349.20
12	Peças impressas em 3D	7	R$ 30.00	R$ 210.00
13	Parafusos e porcas	8	R$ 6.90	R$ 55.20
15	Fitas/"Enforca-gatos	1	R$ 25.00	R$ 25.00
			Total	R$ 2,347.69

Fonte: Autores (2023).

A Figura 12 representa o total gasto no produto final. e a Figura 13 mostra a montagem final do chassi e a base de carregamento.

Figura 13: Montagem final.



Fonte: Autores (2023).

6 Ubuntu

O Ubuntu é uma distribuição de Linux amplamente adotada na comunidade de robótica, especialmente quando se trata de desenvolvimento com o Robot Operating System (ROS). Esta escolha é respaldada por várias razões técnicas e funcionais que contribuem para a eficácia no desenvolvimento de robôs autônomos. A instalação é bem documentada e bem fácil de ser realizada.[14]

Fundamentos do Ubuntu para ROS

- **Estabilidade e Confiabilidade:** O Ubuntu é reconhecido por sua estabilidade e confiabilidade, características cruciais para o desenvolvimento de sistemas robóticos.
- **Compatibilidade com ROS:** O ROS tem forte suporte e integração com o Ubuntu, facilitando a instalação e configuração.
- **Gerenciamento de Pacotes com apt:** O sistema de gerenciamento de pacotes `apt` do Ubuntu simplifica a instalação, atualização e remoção de software, facilitando a integração de pacotes ROS.
- **Comunidade Ativa:** A comunidade de usuários do Ubuntu é vasta e ativa, proporcionando suporte e recursos online.

Vantagens para Desenvolvimento de Robôs

- **Suporte a Hardware Diversificado:** O Ubuntu oferece suporte a uma ampla variedade de hardware, tornando-o ideal para robótica.
- **Permissões de Portas USB e Integração de Hardware:**[9] O Ubuntu permite a gestão de permissões de portas USB para uma interação adequada com dispositivos como sensores LIDAR.
- **Instalação de Drivers:**[3] A instalação de drivers, como os usados em sensores LIDAR, é facilitada no Ubuntu.
- **Remapeamento de Portas USB:** Ferramentas no Ubuntu permitem o remapeamento eficiente de portas USB.
- **Configuração do Ambiente ROS:** O Ubuntu simplifica a configuração do ambiente ROS, com comandos de inicialização em bash, como `source devel/setup.bash`.
- **Criação de Servidores para o ROS:**[11] O Ubuntu facilita a criação de servidores dedicados para o ROS, permitindo comunicação eficiente em sistemas robóticos distribuídos em máquinas diferentes.

O Ubuntu, com sua estabilidade, vasta compatibilidade e suporte robusto ao hardware, é uma escolha excepcional para o desenvolvimento em ROS. As funcionalidades integradas do Ubuntu simplificam muitos aspectos do desenvolvimento robótico, proporcionando um ambiente consistente e eficiente para a criação de sistemas autônomos avançados.

7 ROS-Robot Operating System

7.1 ROS-Robot Operating System

O desenvolvimento do software do projeto girou em torno do ROS, o Robot Operating System. [2], a partir da documentação sobre a instalação do ROS disponível foi possivel iniciar o projeto. O ROS melodic foi escolhido por ser a versão com mais suporte da comunidade internacional de robótica, uma vez que que muitos pacotes ja estão prontos e disponíveis no GitHub.

Transcrevendo [7] pode-se entender que o ROS é um sistema operacional que permite a criação de nós. Um nó é uma unidade de processamento independente no ROS, que executa uma tarefa específica. Pode ser um driver de hardware, um algoritmo de controle, ou qualquer outra parte do sistema que realiza uma função específica. Os nós são módulos autônomos que colaboram para formar um sistema robótico. Cada nó executa uma tarefa específica, promovendo a modularidade e a reutilização de código. Esses nós podem ser divididos em 4 classes diferentes, a depender da funcionalidade que ele executa na rede ROS.

7.2 Nós

7.2.1 Publisher

Um publisher é um nó que envia mensagens sobre um tópico específico. Ele produz dados para outros nós interessados em consumir essas informações. Na prática profissional, um publisher é frequentemente associado a sensores ou dados de estado que precisam ser compartilhados com outros componentes do sistema.

Em nosso projeto, foi utilizado 3 principais publishers, o cmdvel, responsável por enviar comandos de direção do robô, o laser_scan, tópico que gera os dados do LiDar e disponibiliza na rede ROS, e por fim o tópico odom que utiliza as informações do LiDar para estimar a localização do robô e gerar o mapa.

7.2.2 Subscriber

Um subscriber é um nó que recebe mensagens de um tópico específico. Ele consome os dados publicados por um publisher. Profissionalmente, um subscriber é usado para acessar informações específicas de outros nós. No caso do sistema desenvolvido o principal subscriber foi o nó responsável por fazer a movimentação autônoma do robô, onde ele era subscriber do tópico laser_scan.

7.2.3 Callback

Um callback é uma função que é chamada em resposta a um evento específico. Em ROS, muitas vezes é usado para processar mensagens recebidas de um tópico. No contexto profissional, um callback é utilizado para lidar com dados recebidos de forma assíncrona. Por exemplo, ao receber dados de um sensor, um callback pode ser usado para processar e reagir a esses dados de maneira eficiente. Foi utilizado o cmdvel como nó callback em nossa execução.

7.2.4 Service

Um serviço é uma chamada de procedimento remoto (RPC) entre nós no ROS. Permite que um nó solicite a execução de uma função em outro nó e receba uma resposta. Em um ambiente profissional, serviços são usados para comunicação síncrona entre nós. Por exemplo, um nó resposável por salvar um mapa, ele espera o nó que cria este mapa disponibiliza-lo na rede, para que ele possa salvar e encerrar o seu funcionamento, de forma que ele depende da resposta direta de outro nó para poder funcionar.

7.3 Pacotes

7.3.1 Cartographer

O pacote Cartographer é uma ferramenta avançada no ecossistema do ROS projetada para realizar mapeamento 2D e 3D de ambientes usando dados de sensores, como um scanner a laser (LIDAR) ou uma câmera RGBD. Ele é particularmente útil para robôs móveis autônomos, pois permite que eles construam mapas precisos de seus arredores, facilitando a navegação eficiente.

O Cartographer opera usando um algoritmo de SLAM (Simultaneous Localization and Mapping), que combina informações de movimento (odometria) com dados do sensor para estimar simultaneamente a posição do robô no ambiente e construir um mapa do ambiente em tempo real.

Apesar de fazer praticamente toda a lógica necessária para o mapeamento de um ambiente o cartographer precisa da ferramenta de vizualição do ROS aberta para que os seus dados possam ser vizualizados, ferramenta esta que se chama rviz e pode mostrar todo o mapeamento e localização do robô em tempo real.[17]

Principais componentes e conceitos do pacote Cartographer:

- **Nó:** O Cartographer opera como um nó no ROS, chamado "cartographer_Nó". Este nó é responsável por coordenar o processo de mapeamento, recebendo dados de sensores e publicando mapas resultantes.
- **Configuração YAML:** O Cartographer é altamente configurável, permitindo ajustar parâmetros para se adaptar a diferentes ambientes e sensores. A configuração é frequentemente feita por meio de arquivos YAML que descrevem os detalhes do sensor, a configuração do algoritmo SLAM e outros parâmetros.
- **Launch File:** No contexto do ROS, o Cartographer é frequentemente iniciado usando um arquivo de lançamento (*launch file*). Este arquivo especifica como configurar e iniciar o nó Cartographer, bem como quais tópicos são usados para entrada e saída de dados.
- **Tópicos:** O Cartographer interage com vários tópicos do ROS, como tópicos para dados de odometria, nuvem de pontos do LIDAR e comandos de controle. Esses tópicos são fundamentais para a comunicação eficiente entre o Cartographer e outros componentes do sistema.
- **Frame de Referência:** O Cartographer utiliza um sistema de coordenadas global e local para realizar a correspondência entre diferentes leituras de sensores e integrar essas informações no mapa. O frame de referência é crucial para garantir uma representação coerente do ambiente. No nosso caso, o frame escolhido foi o laserframe, de forma que o lidar é o frame de referência para o mapa.

O pacote Cartographer foi uma peça fundamental no desenvolvimento do nosso sistema de navegação autônoma e mapeamento em robótica, mostrando ser uma ferramenta poderosa para criar mapas precisos e atualizados em tempo real, para utiliza-lo é necessário digitar o seguinte comando no terminal:[13]

```
$ roslaunch transbot_nav laser_map.launch map_type:=cartographer
frame_id:=laser_link
```

7.3.2 Modo autônomo

O pacote de movimentação autônoma é responsável por controlar o movimento autônomo do robô com base nas leituras do sensor de laser. Utilizando dados do tópico `LaserScan`, o código implementa lógica de desvio de obstáculos e envio de comandos de movimento.

- **Estrutura do Código:** A estrutura do código está organizada em uma classe principal chamada `laserAvoid`, que contém os seguintes métodos:

  1. `__init__`: Configura parâmetros iniciais, inicializa variáveis de controle, e configura a comunicação serial.
  2. `cancel`: Registrado como callback para o encerramento do nó, desregistra o assinante e realiza procedimentos de encerramento.
  3. `registerScan`: Processa os dados do LaserScan para identificar obstáculos à frente, à esquerda e à direita.
  4. `robot_move`: Implementa a lógica de movimentação autônoma, desviando de obstáculos com base nas leituras do LaserScan.
  5. `send_serial_command`: Envia comandos para a porta serial, utilizada para controlar o robô.

- **Funcionamento Geral:** O código segue uma abordagem reativa para desvio de obstáculos. Durante a execução, o robô verifica as leituras do sensor de laser e toma decisões de movimento com base nas condições encontradas. As condições incluem a detecção de obstáculos à frente, à esquerda e à direita.

  Os comandos de movimento são enviados por meio da comunicação serial, presumivelmente controlando os motores do robô. A lógica de movimentação é organizada em uma série de condições, cada uma correspondendo a uma situação específica de detecção de obstáculos. Para utilizar o pacote de movimentação autonoma é necessario digitar o seguinte comando no terminal:[10]

  ```
  $ roslaunch robot_controller relampago_marquinhos.launch
  serial_port:=/dev/ttyUSBX
  ```

7.3.3 Modo Manual

O pacote de controle manual, permite a operação manual do robô por meio de comandos enviados ao tópico `/cmd_vel`. Este modo manual é projetado para ser controlado externamente, pelo usuário.

- **Estrutura do Código** A estrutura do código consiste em uma classe principal chamada `RobotController`. Essa classe possui os seguintes métodos:

1. `__init__`: Configura o nó, inicializa o assinante para o tópico `/cmd_vel`, e configura a comunicação serial.
2. `cancel`: Registrado como callback para o encerramento do nó, desregistra o assinante e realiza procedimentos de encerramento.
3. `cmd_vel_callback`: Callback para receber comandos do tópico `/cmd_vel`. Converte comandos lineares e angulares em comandos de movimento.
4. `send_serial_command`: Envia comandos para a porta serial, utilizada para controlar o robô.

- **Funcionamento Geral** O código monitora o tópico `/cmd_vel` para receber comandos de velocidade linear e angular. Com base nos valores recebidos, determina os comandos apropriados para movimento do robô. Os comandos são então enviados por meio da comunicação serial para controlar o hardware do robô.

  Os comandos são interpretados da seguinte maneira:

  – Velocidade linear positiva: Move para frente (`'S'`).
  – Velocidade linear negativa: Move para trás (`'W'`).
  – Velocidade angular positiva: Rotaciona à esquerda (`'A'`).
  – Velocidade angular negativa: Rotaciona à direita (`'D'`).
  – Valores nulos: Para o robô (`'K'`).

  O pacote de movimentação manual é utilizado da seguinte maneira:[10]

  ```
  $ rosrun robot_controller cmd_vel_subscriber.py
  ```

  Em outro terminal é necessário abrir o cmd_vel para receber os inputs do teclado do usuário:[15]

  ```
  $ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
  ```

O pacote manual, é composto por 2 pacotes que interagem entre si, o cmd_vel, que recebe as mensagens do teclado, e o nó cmd_vel_subscriber.py recebe essas mensagens e manda para o baixo nivel executá-la.

7.3.4 Pacote `map_server`

O pacote `map_server` `map_save` desempenham um papel crucial na gestão e persistência de mapas em ambientes de robótica autônoma. Essa combinação permite a criação, manipulação e salvamento eficiente de mapas utilizados para navegação autônoma de robôs.

- **Estrutura e Funcionalidades do Pacote `map_server`**

  O pacote `map_server` proporciona funcionalidades essenciais para a gestão de mapas em ambientes robóticos. Sua estrutura inclui:

  1. `map_server` Nó: Responsável por publicar mapas e fornecer serviços relacionados à informação do mapa.

2. `Serviços de Mapa`: Oferece serviços para requisitar informações sobre o mapa, como dimensões, resolução e dados brutos.
3. `Integração com Navegação`: Integrado a sistemas de navegação autônoma, permitindo a utilização de mapas para tomada de decisões durante a navegação do robô.

- **Comando `map_save`**

  O comando `map_save` é uma ferramenta associada ao pacote `map_server`, permitindo salvar a imagem do mapa para uso futuro. Suas principais características incluem:

  1. `Mapa no buffer`: Permite ao usuário salvar o mapa atualmente carregado pelo `map_server`.
  2. `Salvar`: Os mapas salvos são armazenados em arquivos, garantindo que a informação do ambiente mapeado seja preservada.
  3. `Facilidade de Uso`: O comando `map_save` oferece uma interface simples e eficiente para salvar mapas, tornando a operação acessível aos usuários.

- **Utilização do Comando `map_save`**

  Para salvar um mapa utilizando o comando `map_save`, utilize o seguinte comando no terminal:[1]

  ```
  $ rosrun map_server map_saver -f caminho/do/arquivo/mapa
  ```

  Substitua `caminho/do/arquivo/mapa` pelo caminho desejado e nome do arquivo para o mapa.

- **Considerações Finais**

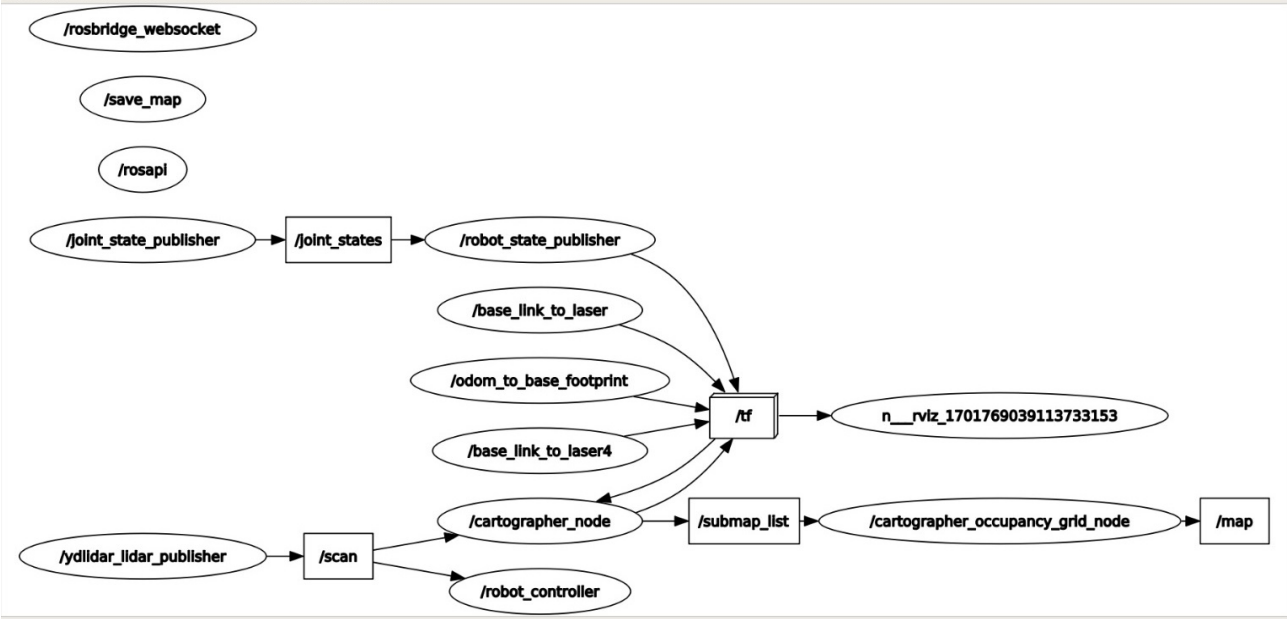
  A combinação do pacote `map_server` e o comando `map_save` é vital para a operação autônoma de robôs em ambientes mapeados. A partir da compreensão e utilização deste pacote é possível salvar os mapas gerados pelo robô.

7.4 Fluxograma

Segue abaixo um fluxograma 14 gerado pela ferramenta do ROS, rqt_graph[16], onde é possível enxergar como cada nó esta interagindo no sistema, tanto com a movimentação autônoma, quanto com a movimentação manual.

Ademais, é possível observar também como o tópico /scan interage com o pacote cartographer para gerar o mapa.—

Figura 14: Fluxograma Ros.



Fonte: Autores (2023).

8 FreeRTOS

O sistema embarcado apresenta um projeto baseado em FreeRTOS para controlar motores, encoders e um sensor inercial (IMU) BNO055. O código em questão utiliza a biblioteca FreeRTOS para gerenciar tarefas concorrentes e garantir uma execução eficiente em um ambiente de tempo real.[8]

8.1 Configuração de Hardware

- **Motores e Encoders:** Pinagem dos motores (MOTOR_A1, MOTOR_A2, MOTOR_B1, MOTOR_B2) e encoders (ENCODER_A, ENCODER_B).
- **PWM:** Utilização da biblioteca `ledc` para configurar os pinos como saída PWM, permitindo controle de potência dos motores.
- **IMU BNO055:** Utilização do sensor inercial BNO055 para obtenção de dados de orientação, giroscópio, aceleração e outros.

8.2 Controle de Encoders

O sistema monitora dois encoders (`contadorEncoderA` e `contadorEncoderB`) para calcular as rotações por minuto (RPM) dos motores. Esses valores são utilizados posteriormente para implementar um controle PID.

8.3 Tarefas FreeRTOS 15

- **Tarefa do Encoder:** A tarefa `tarefaEncoder` calcula os RPM dos motores periodicamente.
- **Tarefa IMU:** A tarefa `taskIMUCode` obtém dados do sensor inercial BNO055 e os imprime.

8.4 Controle PID e Motores

O código implementa um controle PID para manter a velocidade dos motores próxima a um valor desejado (`setPoint`). A função `computePID` é utilizada para calcular a potência dos motores com base nos erros proporcional, integral e derivativo.

8.5 Comandos Externos

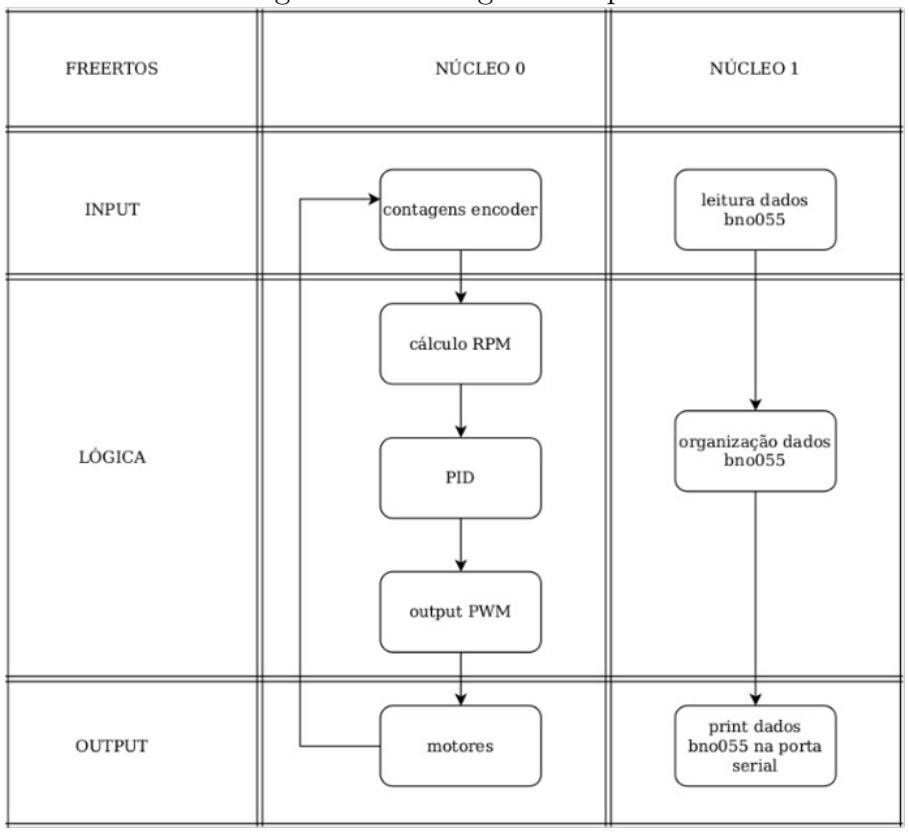
O sistema aguarda comandos externos recebidos pela porta serial para determinar a ação dos motores. Os comandos possíveis incluem movimentos para frente (`'W'`), para trás (`'S'`), rotação à esquerda (`'A'`), rotação à direita (`'D'`) e parada (`'K'`).

O código implementa um sistema embarcado robusto, utilizando FreeRTOS para tarefas concorrentes, controle PID para estabilidade dos motores e integração de um sensor inercial para monitoramento do ambiente. Essa abordagem proporciona um controle preciso e eficiente em aplicações robóticas.

8.6 Fluxograma

Segue abaixo o fluxograma da lógica do freeRTOS, explicando cada core separadamente e mostrando as atividades acontecendo em simultâneo. 15

Figura 15: Fluxograma Esp32.



Fonte: Autores (2023).

9 Resultados e discussões

9.1 Hardware

O desafio da eletronica é entender as minuncias, é nos detalhes que ficam escondidos as falhas e os erros. Diversos componentes foram queimados durante o processo, incrivelmente mais de 4 esp-32 foram queimados, dois chassis foram utilizados, 2 rodas reservas trocadas. Um encoder incremental estava com um curto muito peculiar, quando a luz era interrompida, o hardware entrava em curto e o sistema travava, ele não tinha indicação de falha. Isso custou mais de 2 semans do projeto.

O desafio de projetar a placa-mãe apareceu de forma gradativa, o primeiro prototipo foi feito em fenolite, ele comportava os componentes, com isso podia se ter uma visão melhor do projeto, e tambem evita mal contato, no total foram mais de 5 projetos de circuito para chegar no PCB final, a corrosão das placas foram feitas em casa com percloreto ferrico, papel couche 170 mg e um ferro de passar roupa. A técnica do processo de corrosão foi dominada pela equipe.

A estrutura do Robo, juntamente com a base foi impressa na impressora 3D de um membro da equipe, o desafio de projetar hardwares fisicamentes estaveis e dificil e necessita de bastante atenção e tempo.

9.2 Software

Inicialmente a escolha da equipe foi em utilizar o ROS2, que após perder um certo tempo, foi possível enxergar que não possuia ainda o suporte necessário, após pesquisar foi encontrado a versão "ROS melodic"que melhor obedecia os requisitos do projeto e da equipe, isso se deve, principalmente, ao fato do pouco conhecimento que se tinha sobre o ROS quando a escolha foi feita.

Após 4 meses, é notório o quanto trabalhar com o ubuntu e o ROS melhora as habilidades em programação. Isso se deve principalmente a liberdade de se poder criar quase qualquer coisa dentro do sistema, basta saber estruturar o fluxo de dados corretamente. Isso vale, não só para o ROS mas para todos os softwares, é sempre preciso entender qual dado se quer adquirir, onde ele deve chegar e como ele precisa chegar.

Além disso, o código específico para o ESP32, utilizando FreeRTOS, pode ser comparado ao ROS ao pensar nas tasks como nós que executam uma ação, no caso do sistema empregado no ESP32 foi utilizado variáveis globais ao invés de filas, porém é possível ver a semelhança entre o freeRTOS e o ROS em questão de funcionamento.

9.3 Conclusão

O projeto cumpre o proposto, ele tem a habilidade de se locomover autonomamente enquanto mapeia o ambiente. Sua localização é possivel por intermédio do cartographer, e por conseguinte, pode-se exporta o mapa. O controle manual do "Relampago-marquinhos"funciona, e é possivel controlar via conexão ssh. A base é capaz de carregar o sistema, acendendo um indicador led ao atracar. Com tudo é possivel dizer que o projeto foi um sucesso.

Referências

[1] map_server - ros wiki, 03 2020.

[2] melodic/installation/ubuntu - ros wiki, 03 2020.

[3] zhanyiaini . Table of contents, 06 2022.

[4] EVAN ACKERMAN and ERICO GUIZZO. Hands-on with turtlebot 3, a powerful little robot for learning ros, 05 2017.

[5] The Robotics Back-End. Ros tutorials, 2023.

[6] josh. Making a mobile robot #8 - adding a lidar, 06 2022.

[7] Josh Newans. joshnewans/articubot_one, 03 2023.

[8] Ramon Mariano Raphael Diniz. controle pid, 12 2023.

[9] Ramon Mariano Raphael Diniz. restartusb, 12 2023.

[10] Ramon Mariano Raphael Diniz. robot$_{c}ontroller$, 122023.

[11] razbotics. Ros on multiple computers — connecting raspberry pi with pc over lan, 01 2018.

[12] RoboticArts. ros_imu_bno055, 12 2020.

[13] Yahboom Technology. Ydlidar x3/x3 pro lidar tof 360° scanning ranging sensor 8m for ros robotics support ros1 ros2, 11 2023.

[14] Ubunto. Install ubuntu desktop, 2023.

[15] ROS WIKI. teleop$_{t}wist_{k}eyboard$, 052015.

[16] ROS WIKI. rqt$_{g}raph$, 052018.

[17] ROS WIKI. Rviz, 05 2018.

10 Apêndices

10.1 Links

Video - Youtube O vídeo do projeto, demonstrando o seu funcionamento e os componentes, está disponível na seguinte URL:

`https://youtube.com/Neuspace/relampago_marquinhos`

GitHub Os algoritimos desenvolvidos e utilizados estao disponiveis na seguinte URL

`https://github.com/RaphaDiniz/robot_controller`
`https://github.com/RaphaDiniz/PID_control`
`https://github.com/RaphaDiniz/restart-usb-port`