UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA

JOSÉ MÁRIO N. DE ALBUQUERQUE
GUSTAVO F. LEWIN
DAVI RIITI

ROBÔ MAPEADOR DE BAIXO CUSTO

RELATÓRIO

CURITIBA
2022

JOSÉ MÁRIO N. DE ALBUQUERQUE

GUSTAVO F. LEWIN

DAVI RIITI

ROBÔ MAPEADOR DE BAIXO CUSTO

Low-cost robot with mapping capabilities

Relatório apresentado para a diciplina ELE41 do curso Engenharia Eletrônica, do Departamento Acadêmico de Eletrônica, da Universidade Tecnológica Federal do Paraná (UTFPR).

Prof.: Ronnier Frates Rohrich
Prof.: Daniel Rossato de Oliveira

CURITIBA

2022

SUMÁRIO

1 INTRODUÇÃO

Figura 1 – TurtleBot Burguer 3.

Fonte: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (acesso em 28-08-2022).

Este projeto é inspirado no TurtleBot Burger (TURTLEBOT3, 2017), mostrado na Figura 1, que é um robô de *hardware* e *software* abertos, servindo de plataforma de entrada para a robótica móvel. No entanto, alguns componentes do TurtleBot são de difícil acesso devido ao custo – especificamente os motores e o *Lidar*. Em razão disso, devido ao custo e a disponibilidade de alguns componentes, o projeto usa peças diferentes.

O robô do projeto deve ser capaz de navegar, mapear e gerar o mapa de uma área, usando o *Robot Operating System* (ROS) (ROS, 2007) como base. Para isso, alguns passos são importantes:

- implementar a odometria, para que seja possível estimar a posição do robô em relação a seu ponto inicial;
- implementar um sensor para desempenhar o papel de *Lidar* (i.e. medir a distância até obstáculos próximos);
- por fim, incorporar essas informações e gerar o mapa em tempo real.

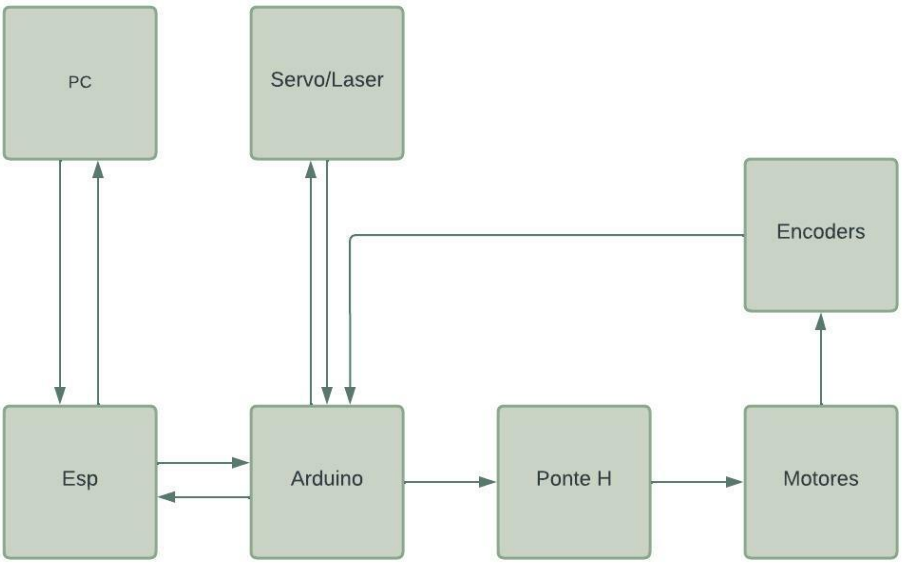
2 METODOLOGIA

2.0.1 Visão geral

Todos os componentes foram montados em dois chassis de acrílico, sendo que a Figura 2 mostra o diagrama do projeto. A lista de componentes é dada por:

- um par de motores DC;
- um par de *encoders* ópticos;
- Arduino Mega;
- NodeMCU ESP8266;
- servo motor;
- sensor *laser* VL53L0X;
- *shield* controlador de motor L293D;
- *powerbank* de 10000 mAh.

Figura 2 – Diagrama de blocos do projeto.



Fonte: Autoria própria.

O comando de deslocamento do robô vem de algum outro nó conectado na rede do robô. Esse comando possui duas componentes: velocidade linear (v_x) e velocidade angular (w_z). O robô processa esses dois valores e, através do modelo cinemático (ver 2.0.7), calcula quais valores de velocidade de cada roda são necessários para atingir v_x e w_z, utilizando um sistema de controle que faz uso dos *encoders* (ver 2.0.6).

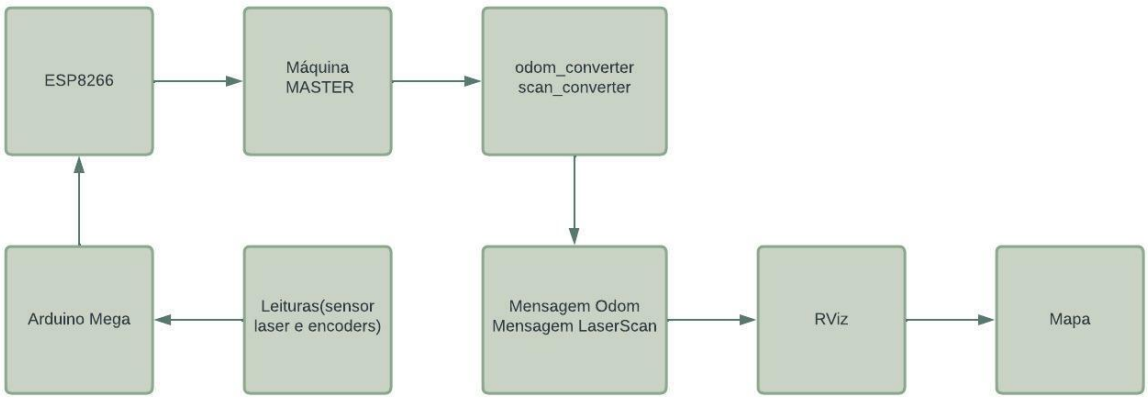
Para realizar o mapeamento, são necessárias três informações: (i) a distância lida pelo sensor *laser*, (ii) o ângulo do servo e (iii) a odometria do robô. O primeiro e segundo itens são obtidos de forma direta pela leitura dos sensores. Já a odometria, precisa ser calculada a partir da contagem de interrupções dos *encoders* em certos intervalos de tempo, o que também faz uso do modelo cinemático (ver 2.0.7).

Para tornar possível a visualização das informações dos sensores através do ROS, elas tiveram de ser convertidas nos formatos aceitos pelo programa. Foram programados dois nodos ROS para rodarem na máquina MASTER da rede:

- **odom_converter.cpp**: converte a informação de odometria em uma mensagem de Odom do ROS
- **scan_converter.cpp**: converte a informação do sensor laser e servo em uma mensagem LaserScan do ROS

Por fim, o mapa é gerado na máquina MASTER usando o RViz, uma ferramenta de visualização 3D do ROS, o código pode ser visualizado por inteiro no repositório do GitHub (TURTLEBOT_OFICINA. . . , 2022). A Figura 3 mostra como a informação é transportada e transformada durante o funcionamento do robô.

Figura 3 – Diagrama de blocos das informações.



Fonte: Autoria própria.

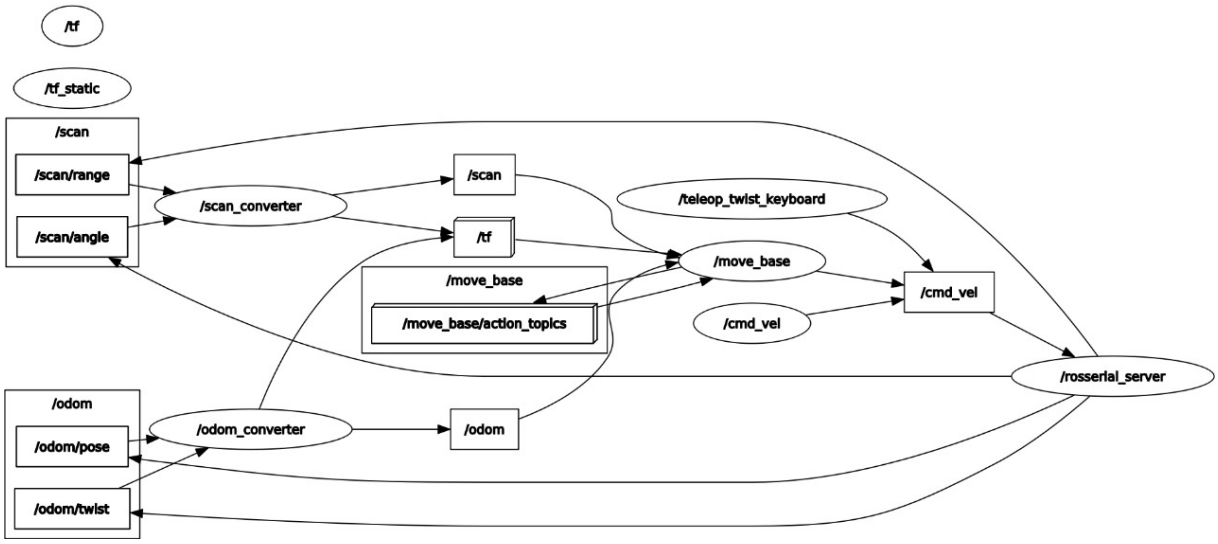
2.0.2 *Robot Operating System* (ROS)

Para servir de sistema de comunicação base do robô, via rede WiFi, o ROS foi utilizado. É um conjunto de bibliotecas de *software* e ferramentas voltados ao desenvolvimento de robôs. Para usá-lo, é necessário uma máquina MASTER, que deverá executar de forma intermitente o programa ROSCORE, que gerencia todos os serviços necessários para interfacear os nós envolvidos na rede do robô. Os seguintes pacotes do ROS foram utilizados:

- **move_base:** implementa um algoritmo de *Simultaneous Localization and Mapping* (SLAM), muito utilizado em robôs autônomos (e.g. robô aspirador);
- **rosserial:** define um protocolo de serialização de mensagens nos padrões do ROS, nesse caso seja via rede local (WiFi);
- **rviz:** é uma ferramenta de visualização 3D para o ROS, o que permite a visualização da odometria e outros sensores.

A Figura 4 exibe os nós conectados no ROS, bem como as mensagens trocadas entre eles.

Figura 4 – Nós e mensagens da rede ROS

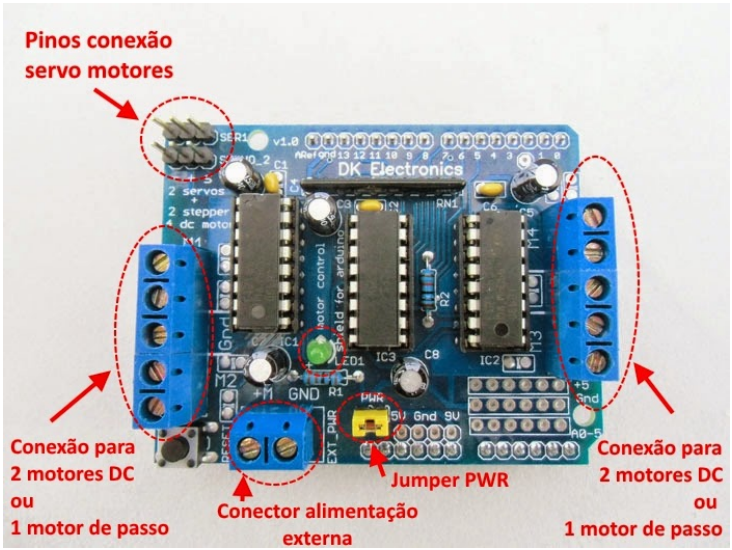


Fonte: Gerado pelo pacote rqt_graph.

2.0.3 Arduino Mega e Shield L293D

Para controlar os motores DC, foi utilizado o shield com ponte H L293D acoplado a um Arduino Mega. Conforme a Figura 5, ele é capaz de controlar até 4 motores DC.

Figura 5 – Módulo com ponte H L293D

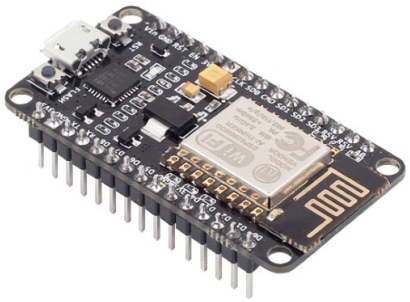


Fonte: https://www.arduinoecia.com.br/ (acesso em 28-08-2022).

2.0.4 NodeMCU ESP8266

O módulo NodeMCU ESP-12E (Figura 6) contém o chip ESP8266, que possibilita a comunicação WiFi da placa. Ele foi utilizado para interfacear o Arduino Mega com o computador MASTER da rede. A biblioteca *Rosserial Arduino* foi utilizada para conectar ao ROS.

Figura 6 – NodeMCU ESP8266

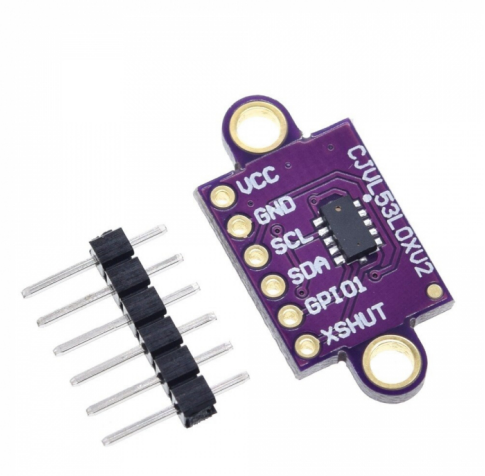


Fonte: https://www.filipeflop.com/ (acesso em 28-08-2022).

2.0.5 *Lidar*

Devido ao fato de que um *Lidar* dos mais baratos custaria em torno de R$500,00, foi concluído que um modelo próprio deveria ser criado, do zero, para o projeto. Para isso, o sensor *laser* VL53L0X (Figura 7) foi acoplado a um servo motor cujo alcance vai de 0 a 180 graus (Figura 8). Um suporte para acoplar o sensor laser no servo foi projetado e fabricado via impressora 3D. Dessa maneira, foi desenvolvido um sensor capaz de realizar a varredura de obstáculos por meio da técnica de tempo-de-vôo (Figura 9).

Figura 7 – Sensor *laser* VL53L0X.

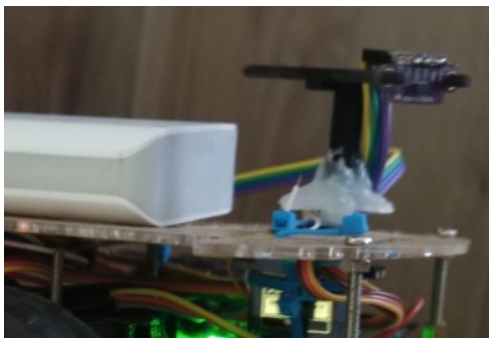


Fonte: https://curtocircuito.com.br (acesso em 05-12-2022).

Figura 8 – Servo motor SG90.

Fonte: https://www.filipeflop.com/ (acesso em 05-12-2022).

Figura 9 – Conjunto do servo e sensor *laser* (*Lidar*).



Fonte: Autoria própria.

O servo é controlado pelo Arduino, que a cada 40 ms incrementa o ângulo em uma quantidade calibrável de acordo com a necessidade do operador (normalmente 1 grau, para uma varredura lenta de 180 pontos, ou 5 graus para uma varredura rápida de 36 pontos). Quando o ângulo é incrementado, o sensor *laser* realiza a leitura da distância e então o Arduino envia a distância e o ângulo registrados para o ESP.

2.0.6 Motores e *encoders*

Foram utilizados motores de plástico 3-6V, com redução de plástico. Foram percebidos problemas no controle de qualidade desses motores. Logo, teria de ser implementado um controle um pouco mais avançado para estabilizá-los.

Os *encoders* detectam quando há uma interrupção no sinal óptico, geradas por discos acoplados aos motores. Sabendo o número de buracos no disco, podemos estimar a velocidade das rodas usando um contador de interrupções e registrando o intervalo de tempo entre cada estimativa. Dessa maneira, foi possível calcular as rotações por minuto (RPM) de cada roda, o que por sua vez possibilitou a implementação de um sistema de controle proporcional que visa estabilizar as rotações de cada motor. A Figura 10 exibe o conjunto de motor, roda e *encoder*.

Figura 10 – Conjunto motor, roda e *encoder*

Fonte: https://shopee.com.br/ (acesso em 28-08-2022).

2.0.7 Modelo cinemático e odometria

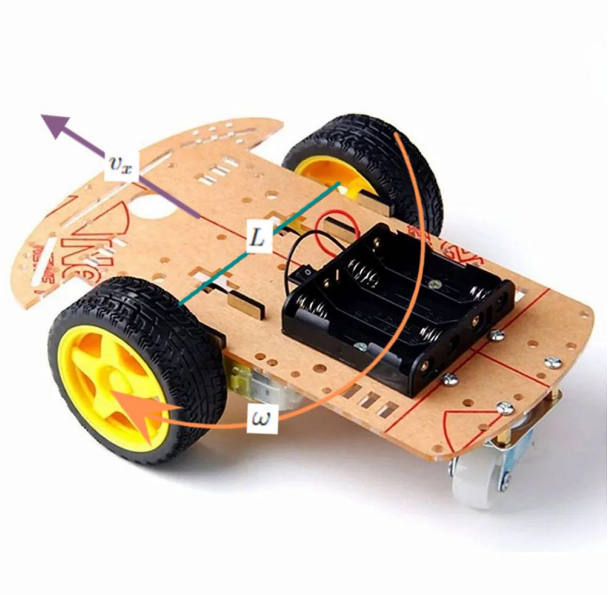
Foi necessário relacionar as velocidades das RODAS com as velocidades linear e angular do ROBÔ. No caso de um carro de duas rodas, as equações 1 e 2 determinam, respectivamente, a velocidade angular da roda direita e da roda esquerda.

$$\partial\varphi_{right}/\partial t = (\omega L/2r) + (v_x/r) \tag{1}$$

$$\partial\varphi_{left}/\partial t = (-\omega L/2r) + (v_x/r) \tag{2}$$

Conforme a Figura 11, ω corresponde à velocidade angular do carro, v_x à velocidade linear, L seria o comprimento do eixo das rodas e r o raio das rodas.

Figura 11 – Chassi com par de motores



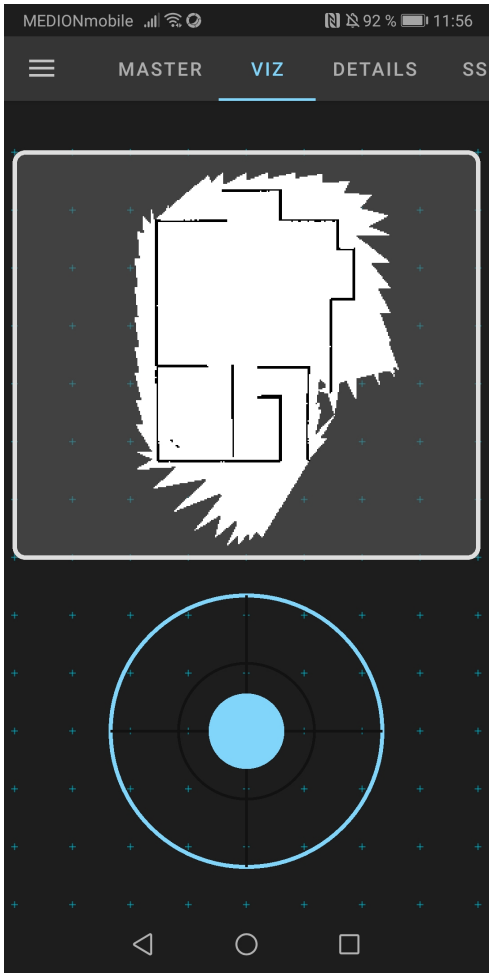
Fonte: https://daeletrica.com.br/ (acesso em 28-08-2022)

Além disso, o modelo cinemático também foi usado para calcular a variação de posição do robô, e assim estimar sua odometria.

2.0.8 Aplicativo

O aplicativo ROS Mobile foi configurado para controlar o robô via celular, conforme Figura 12.

Figura 12 – ROS Mobile



Fonte: https://github.com/ROS-Mobile/ROS-Mobile-Android (acesso em 28-08-2022)

3 RESULTADOS E DISCUSSÃO

3.0.1 Controle proporcional

Como prova de conceito para o controle proporcional aplicado às rodas, primeiramente foi tentado estabilizar o motor em 60 RPM. Foi observado que o motor realmente estava tendo um tique (indicação sonora) por segundo, o que também é mostrado na Figura 13.

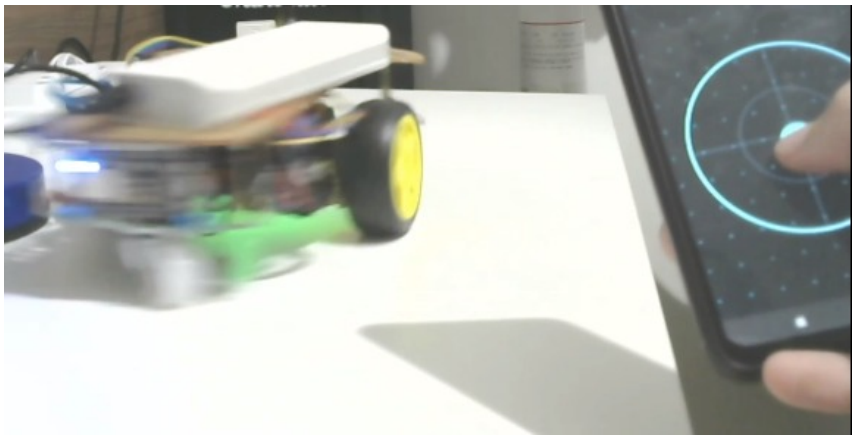
Figura 13 – Motor estabilizado em 60 RPM.

Fonte: Autoria própria.

3.0.2 Teleoperação pelo celular

O controle pelo celular foi implementado com sucesso e a responsividade foi notadamente rápida. A Figura 14 exibe o celular controlando o robô.

Figura 14 – Imagem mostrando o robô recebendo comandos do aplicativo.



Fonte: Autoria própria.

3.0.3 Odometria

A posição do robô é perceptivelmente atualizada no Rviz de forma aceitável. Entretanto, o robô por vezes perde atrito com o chão – ocasionando em rotação dos motores sem realmente produzir deslocamento do robô, o que gera erros na odometria.

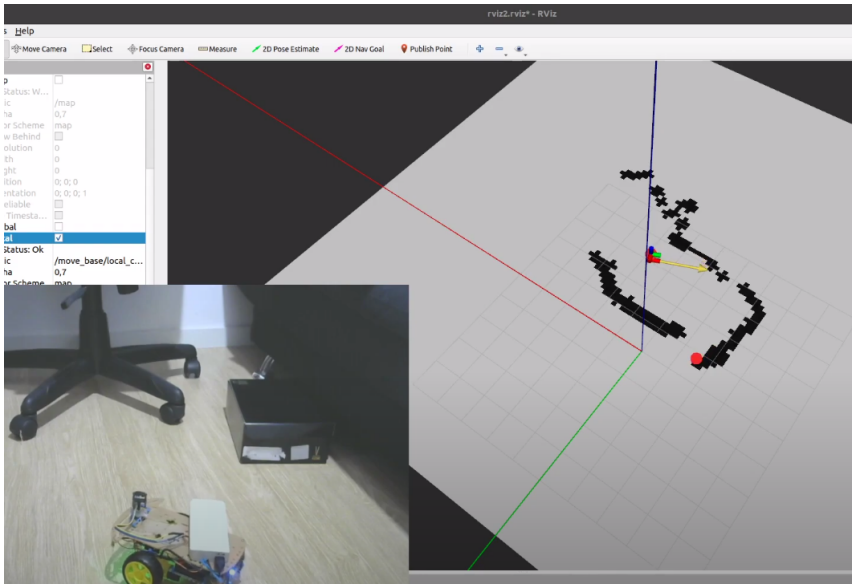
3.0.4 Mapeamento estático

O robô consegue mapear áreas até um metro de distância com precisão. Para distâncias maiores, o sensor *laser* perde rigor e não é capaz de medir mais do que cinco metros. Ele possui um modo específico de medir distâncias um pouco maiores do que as testadas, mas isso não foi experimentado, já que o foco estava em objetos próximos.

3.0.5 Mapeamento dinâmico

Foi possível gerar um mapa que se completa à medida que a posição do robô varia. No entanto, quando ocorre o problema de odometria exposto em 3.0.3, ocorre uma perda de precisão significativa na construção do mapa. O mapeamento e a estimativa de posição utilizando o RViz são mostrados na Figura 15.

Figura 15 – Odometria e mapa no RViz.

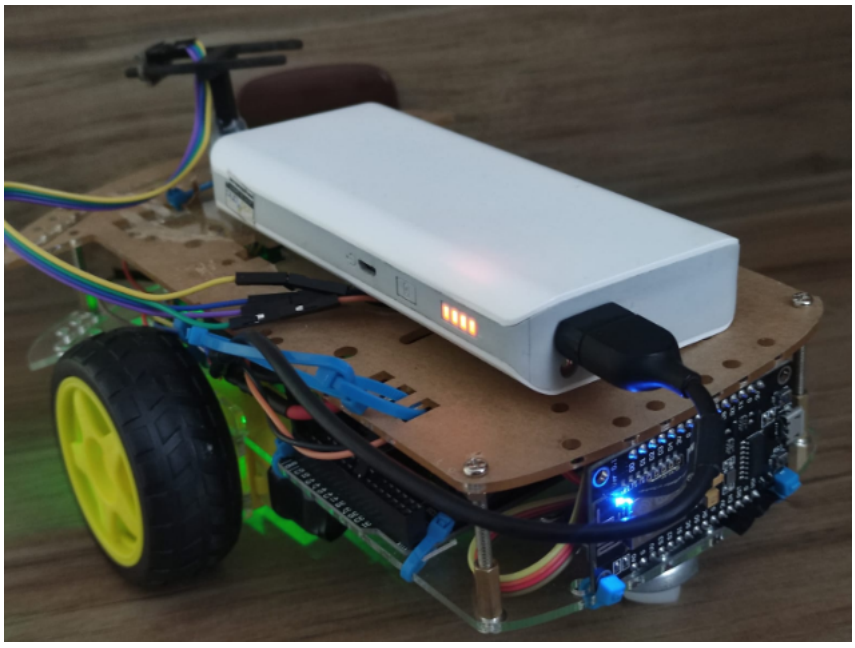


Fonte: Autoria própria.

3.0.6 Estrutura final do robô

A Figura 16 mostra de perto o estado final do robô.

Figura 16 – Estado final do robô.



Fonte: Autoria própria.

4 CONCLUSÕES E PERSPECTIVAS

Os objetivos do projeto foram alcançados. Isto é, o robô é capaz de navegar, mapear e gerar um mapa de uma área. No entanto, apareceram problemas de odometria que não foram 100% resolvidos, situação que possivelmente podia ser solucionada com motores de torque superior e/ou com caixa de redução metálica.

Outra dificuldade do projeto foi a solda do sensor *laser* que, devido ao movimento intenso do servo, soltava com facilidade.

REFERÊNCIAS

ROS. 2007. https://www.ros.org/. Accessado: 28-08-2022.

TURTLEBOT3. *[S.l.]*, 2017. Accessado: 28-08-2022.

TURTLEBOT_OFICINA no GitHub. 2022. https://github.com/gustavoflw/turtlebot_oficina/. Accessado: 06-12-2022.