Explorando Ambientes com Precisão: Navegação e Mapeamento Autônomo de Robôs Móveis de Pequeno Porte com LiDAR

Thalita Wiederkehr Pereira*, Maria Eduarda Crema Carlos*,
Adriana Postal* and Josué Pereira de Castro*
*Curso de Ciência da Computação, Universidade Estadual do Oeste do Paraná - Unioeste, Cascavel, PR,
{thalita.pereira1, adriana.postal, josue.castro}@unioeste.br
Eduardacremacarlos2016@gmail.com
Telefone: +55 45 99107-8732

***Resumo*—This work presents the development of a mapping and autonomous navigation system for small mobile robots using a LiDAR sensor. The proposed architecture follows a client-server approach, where the data acquisition takes place on the robot and the processing is performed externally. Point cloud fusion and landmark identification techniques were applied to create consistent two-dimensional maps. A zigzag movement pattern was used to provide complete coverage of the environment during navigation. The results show the feasibility of the solution with good computing power and an accurate representation of the environment.**

***Keywords*—Mobile robotics; SLAM; Grid-based navigation.**

***Resumo*—Este trabalho apresenta o desenvolvimento de um sistema de mapeamento e navegação autônoma para robôs móveis de pequeno porte utilizando sensor LiDAR. A arquitetura proposta adota uma abordagem cliente-servidor, onde a coleta de dados ocorre no robô e o processamento é realizado externamente. Técnicas de fusão de nuvens de pontos e identificação de *landmarks* foram aplicadas para construir mapas bidimensionais consistentes. Para permitir a navegação em forma de varredura de todo o ambiente, foi utilizado o padrão de movimentação em zigue-zague. Os resultados mostram a viabilidade da solução, com bom desempenho computacional e precisão na representação do ambiente.**

***Palavras-chave*—Robótica móvel; SLAM; Navegação baseada em grade.**

I. Introdução

A robótica móvel autônoma tem se consolidado como uma das áreas mais dinâmicas da computação, impulsionada pelo avanço de sensores de alta precisão, plataformas embarcadas de baixo custo e algoritmos cada vez mais robustos. Nesse contexto, a capacidade de mapear ambientes e realizar navegação autônoma representa um desafio técnico significativo, exigindo integração eficiente entre hardware, sensores e métodos de processamento de dados.

Entre os diversos sensores disponíveis, o LiDAR (*Light Detection and Ranging*) [1] destaca-se pela precisão na geração de mapas bidimensionais e tridimensionais, sendo amplamente utilizado em veículos autônomos, drones e robôs móveis. Entretanto, a aplicação do LiDAR em robôs de pequeno porte demanda soluções de baixo custo computacional que preservem a eficiência e a confiabilidade dos dados capturados.

Este trabalho apresenta o desenvolvimento de um sistema de mapeamento e navegação autônoma para um robô móvel de pequeno porte, integrando um sensor LiDAR LD06 a uma arquitetura de software baseada em cliente-servidor. A proposta explora a coleta de dados espaciais e a geração de mapas por meio de nuvens de pontos, assim como a identificação e correspondência de *landmarks* utilizando técnicas de aprendizado de máquina. Além disso, implementa-se uma estratégia de navegação em padrão zigue-zague, capaz de adaptar-se a obstáculos detectados em tempo real.

Foram conduzidos experimentos em diferentes cenários, avaliando o desempenho computacional da plataforma embarcada, a fidelidade das nuvens de pontos e a robustez na identificação e integração de *landmarks*. Os resultados mostram que a arquitetura distribuída proposta equilibra adequadamente o processamento entre o robô e o computador cliente, permitindo a construção de mapas consistentes e a navegação segura mesmo em ambientes dinâmicos.

O restante deste artigo está organizado da seguinte forma: a Seção II apresenta os conceitos teóricos que fundamentam o trabalho; a Seção III descreve os materiais, métodos e implementações utilizados; a Seção IV apresenta os experimentos e os resultados obtidos; e, por fim, a Seção V traz as conclusões e perspectivas futuras.

II. Fundamentação Teórica

A. Mapeamento

O mapeamento é o processo de construção de uma representação espacial de um ambiente, geralmente realizado por robôs móveis ou sistemas autônomos [2]. No contexto da robótica, o mapeamento envolve a coleta de informações do ambiente através de sensores, que são então utilizadas para criar um modelo ou mapa do espaço em que o robô está inserido. Esse mapa pode ser bidimensional (2D) ou tridimensional (3D), dependendo da natureza dos sensores e da complexidade do ambiente. O objetivo principal do mapeamento é permitir que o robô se localize e navegue de forma eficiente e segura no ambiente ao seu redor, utilizando as informações capturadas para identificar obstáculos, caminhos e referências espaciais importantes [2].

B. Sensores

Sensores são componentes fundamentais na interface entre o mundo físico e os sistemas digitais, atuando efetivamente como "os olhos e ouvidos" de sistemas automatizados. Estes dispositivos convertem grandezas físicas do ambiente, como temperatura, pressão, luz, movimento ou distância, em sinais elétricos que podem ser interpretados e processados por computadores ou sistemas de controle [3].

Neste trabalho foi utilizado o sensor LiDAR (*Light Detection and Ranging*), o qual pertence a categoria de sensores infravermelhos ativos (IR ativos[1]), que utiliza pulsos de laser precisos para medir distâncias e criar nuvens de pontos detalhadas, bi ou tridimensionais, do ambiente [3].

C. Simultaneous Localization and Mapping (SLAM)

O SLAM é uma técnica que permite a um robô construir um mapa de um ambiente desconhecido ao mesmo tempo em que estima sua própria posição e orientação dentro desse espaço [5]. Esse processo combina duas tarefas principais: a localização, que envolve a estimativa da posição do robô em relação ao ambiente, e o mapeamento, que consiste na construção de uma representação do espaço a partir dos dados coletados pelo sensor [6].

A representação do ambiente pode ser feita de diferentes formas, conforme os objetivos e os sensores utilizados. Os mapas de ocupação dividem o ambiente em células discretas, atribuindo a cada uma das células uma probabilidade de estar ocupada. Essas estimativas são atualizadas continuamente com base em novas medições, tornando esse modelo útil para robôs que navegam em ambientes dinâmicos [7].

[1]Sensores que emitem sua própria energia infravermelha, a qual é refletida pelos objetos e então captada pelo sensor [4].

É possível adotar mapas topológicos, que descrevem o ambiente de maneira mais abstrata, como um grafo. Nele, os nós representam locais relevantes e as arestas indicam conexões entre esses pontos. Essa abordagem facilita o planejamento de rotas e a identificação de regiões similares por meio da análise da estrutura do grafo [8].

Outra forma comum de representação é a nuvem de pontos, composta por coordenadas 2D ou 3D obtidas por sensores como LiDAR ou câmeras de profundidade. Esse tipo de mapa capta com precisão os contornos e irregularidades do ambiente, mas demanda técnicas específicas para filtrar ruídos e otimizar o volume de dados [9]. Para este trabalho, foi utilizado esse formato de mapa.

D. Landmarks

As *landmarks* são pontos de referência no ambiente que podem ser detectados por sensores como LiDAR ou câmeras durante o processo de mapeamento e localização. Elas podem incluir características visuais, como bordas, ou elementos geométricos, como cantos e arestas. Funcionam como âncoras para que o robô corrija sua estimativa de posição e reduza as incertezas associadas aos sensores e ao movimento [10].

No contexto do SLAM, ao detectar uma *landmark*, o robô mede a distância e a orientação em relação a ela, utilizando essas informações para ajustar sua localização estimada. A repetida observação das mesmas *landmarks* ao longo do deslocamento permite ao sistema corrigir erros acumulados, mantendo o mapa consistente e preciso mesmo na presença de ruídos nas medições [5].

E. Navegação

A navegação em robôs móveis é considerada um dos pilares da robótica móvel autônoma. Trata-se da capacidade do robô de se locomover de forma segura e eficiente em um ambiente, tomando decisões sobre direção e movimento com base nas informações obtidas ao seu redor [11]. Uma das abordagens mais utilizadas é a navegação baseada em grade (*grid-based navigation*), onde o ambiente é dividido em células que podem ser marcadas como livres (com valor 0) ou ocupadas (com valor 1).

Uma técnica simples e eficiente de cobertura de área é a navegação em zigue-zague, muito comum em aplicações como mapeamento, limpeza ou patrulhamento. Esse padrão permite que o robô percorra sistematicamente todas as regiões livres do ambiente, linha por linha, reduzindo o risco de deixar áreas sem cobertura [12].

Em ambientes dinâmicos ou parcialmente desconhecidos, a habilidade de desviar de obstáculos em tempo real torna-se essencial. Sensores como o LiDAR fornecem leituras contínuas do entorno, permitindo ao robô identificar e reagir a objetos

inesperados. Essas leituras alimentam a lógica de desvio, composta por manobras temporárias que garantem a retomada da rota principal após o obstáculo.

A navegação de robôs de pequeno porte, como os utilizados neste projeto, requer soluções de baixo custo com alta capacidade de controle. Nesse contexto, o uso do Raspberry Pi, aliado ao controle de motores via GPIO[2], mostrou-se uma alternativa eficaz para a implementação da lógica de navegação.

III. Materiais e Métodos

A. Hardware

1) Robô: A arquitetura do robô foi desenvolvida a partir de uma base de chassi robótico de aço inoxidável *TH Robot Tank* produzido por Xiao R Geek Techonology[3], e para compor a movimentação das esteiras foram utilizados dois motores DC. Para permitir a captura de imagens em vídeo, há uma câmera embarcada. O robô é alimentado por uma bateria recarregável Li-ion com 4 células 18650. A Figura 1 mostra o protótipo utilizado para a realização do projeto.

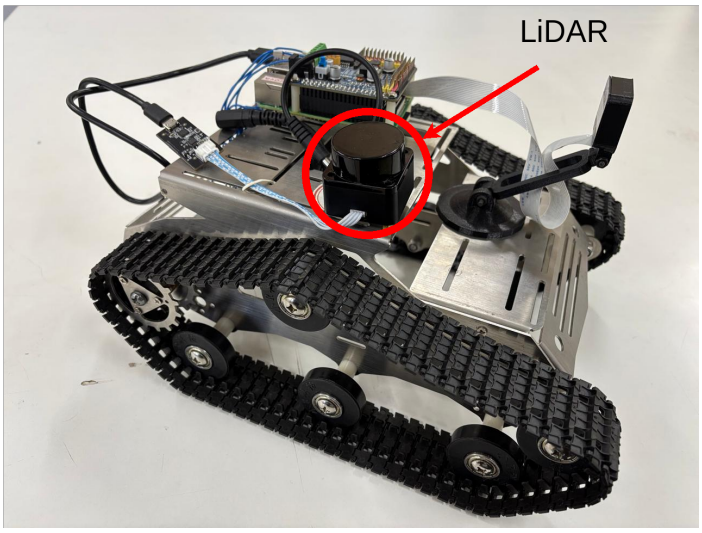


Figura 1. Robô utilizado, com LiDAR. Fonte: Os autores.

2) Placa de controle: A parte eletrônica do projeto utiliza a placa Raspberry Pi 4 Model B, um microcomputador completo em uma única placa de circuitos. A placa possui as seguintes especificações[4]:

- Processador: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memória: 4GB LPDDR4-3200 SDRAM
- Conectividade:
  - 2.4 GHz e 5.0 GHz IEEE 802.11ac wireless
  - Bluetooth 5.0, BLE
  - Gigabit Ethernet
- Portas USB:
  - 2 portas USB 3.0
  - 2 portas USB 2.0
- Vídeo e som:
  - 2 portas micro-HDMI (suporte a 4K 60Hz)
  - Porta de áudio estéreo de 3,5mm e vídeo composto
- GPIO: 40 pinos
- Armazenamento: *Slot* para cartão microSD para sistema operacional e dados

3) Sensor LiDAR: O funcionamento do sensor se baseia na emissão de pulsos de laser que são direcionados aos objetos. Após a reflexão, o laser retorna ao sensor, e o sistema cronometra o tempo de ida e volta do feixe. Esse intervalo é então convertido em uma medida de distância precisa, permitindo a construção de um mapa detalhado do ambiente ao redor do sensor [3]. A Figura 2 mostra o funcionamento do sensor LiDAR.

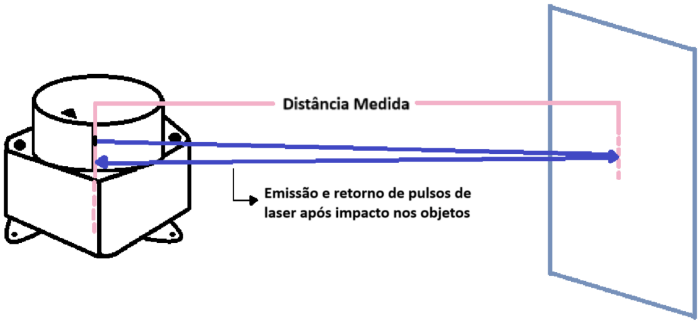


Figura 2. Funcionamento do sensor LiDAR. Fonte: Os autores.

O sensor LiDAR embarcado no robô é do modelo LD06, projetado para realizar varreduras em um plano 2D. O dispositivo gira ao redor de seu eixo para capturar dados em torno dele, medindo a distância até os objetos no seu entorno dentro de um círculo de 360 graus [1]. O sensor utilizado fornece os dados apresentados na Figura 3, sendo:

- *Start character*: comprimento de 1 Byte, valor fixo 0x54, significa o início do pacote de dados;
- *Data Length*: comprimento de 1 Byte, os três primeiros dígitos são reservados, os últimos cinco dígitos representam o número de pontos medidos em um pacote, atualmente com valor fixo 12;

[2]*General Purpopse Input Output* - Entrada e Saída de Propósito Geral.
[3]https://www.xiaorgeek.net/pt-br
[4]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

start character	data length	Radar speed		Start angle		data	end angle		Timestamp		CRCcheck
54H	2CH	LSB	MSB	LSB	MSB	.....	LSB	MSB	LSB	MSB	1Byte

measurement point 1			measurement point 2			···	measurement point n		
Distance Value		Confidence	Distance Value		Confidence		Distance Value		Confidence
LSB	MSB	1 Byte	LSB	MSB	1 Byte	···	LSB	MSB	1 Byte

Figura 3. Pacote de dados do sensor LiDAR. Fonte: [1].

- *Radar speed*: comprimento de 2 Bytes, em graus por segundo;
- *Start angle*: comprimento: 2 Bytes; unidade: 0,01 grau;
- *Data*: o comprimento de dados de uma medição é de 3 bytes;
- *Distance Value LSB*: byte menos significativo do valor de distância;
- *Distance Value MSB*: byte mais significativo do valor de distância;
- *Confidence*: 1 Byte de confiança da medição;
- *End Angle*: comprimento: 2 Bytes; unidade: 0,01 grau;
- *Timestamp*: comprimento de 2 Bytes em milisegundos;
- *CRC check*: soma de verificação de todos os dados anteriores.

Os dados são empacotados e transmitidos em formato binário. O ângulo (em graus) e a distância (em milímetros) são as principais informações para a geração do mapa [1].

Cada ponto de medição inclui um valor de distância, expresso em milímetros, e um valor que reflete a intensidade do sinal retornado pelo objeto, que é utilizado para medir a confiança; quanto maior a intensidade mais confiáveis são os valores para o ponto capturado. A intensidade é maior quando o objeto reflete mais luz, e a reflexão depende das propriedades do material do objeto [1].

Os ângulos de cada ponto são calculados por interpolação linear entre o ângulo inicial e o ângulo final do pacote de dados. A fórmula para calcular o passo (*step*) e o ângulo (*angle*) são calculadas pelas Equações 1 e 2.

$$\text{step} = \frac{\text{end\_angle} - \text{start\_angle}}{\text{len} - 1} \tag{1}$$

$$\text{angle} = \text{start\_angle} + \text{step} \times i \tag{2}$$

onde:
`start_angle`: ângulo inicial em radianos.
`end_angle`: ângulo final em radianos.
`len`: número total de pontos no pacote de dados.
`step`: incremento angular entre pontos consecutivos.
`i`: índice do ponto atual (variando de 0 a len-1).
`angle`: ângulo calculado para o ponto atual.

A Figura 4 mostra o sensor LiDAR LD06, que utiliza um sistema de coordenadas da mão esquerda, com o centro de rotação definido como a origem e o ângulo de rotação aumentando no sentido horário [1].

4) Computador: A seguir, são apresentadas as especificações de hardware do dispositivo externo:

- Processador: Intel(R) Core(TM) i5-10210U, com velocidade base de 1,60 GHz, alcançando até 2,11 GHz em frequências mais altas.
- Memória RAM: 8 GB de RAM instalada (7,83 GB utilizável).
- Armazenamento: SSD de 512 GB.
- Sistema Operacional: Windows 10

B. Softwares utilizados

1) Sistemas Operacionais: O sistema operacional da placa do robô é o Raspberry Pi OS, uma distribuição Linux baseada no Debian Bookworm, desenvolvida especificamente para o hardware do Raspberry Pi 4B. No computador externo, foi instalado o sistema operacional Windows 10. O terminal nativo do Windows é empregado para a conexão com o servidor e o processamento dos dados.

2) Linguagem de programação: A linguagem escolhida para a implementação dos códigos foi Python na versão 3.9.13, que além de estar alinhada com o modelo inicial (seção III-C1),

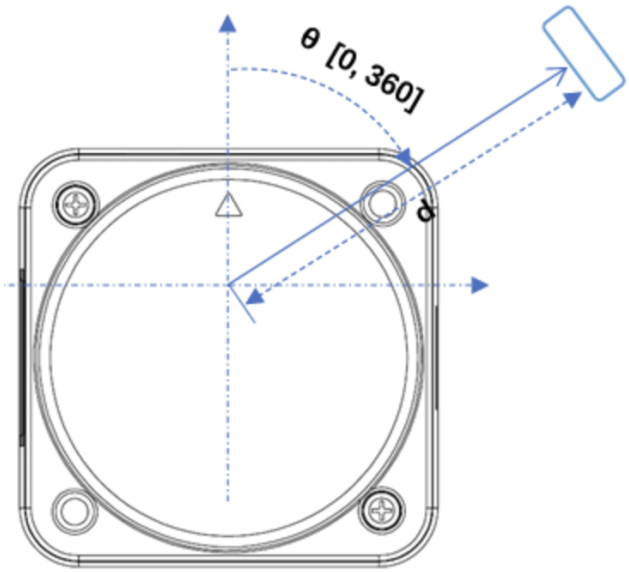


Figura 4. Sistema de coordenadas de mão esquerda (sentido de varredura horário) do LD06. Fonte: [1].

possui bibliotecas de fácil manipulação, auxiliando na conexão cliente-servidor, manipulação do arquivo .csv e representação gráfica do mapa. As bibliotecas utilizadas incluem:

- socket (*Built-in*): utilizada para a comunicação entre cliente e servidor.
- csv (versão 1.0): empregada para manipulação de arquivos CSV.
- json (versão 2.0.9): usada para formatar e interpretar dados em formato JSON.
- matplotlib.pyplot (versão 3.9.1.post1): responsável pela visualização gráfica dos dados.
- math (*Built-in*): fornece funções matemáticas.
- numpy (versão 2.0.1): utilizada para a manipulação de dados numéricos e operações matriciais.
- os (*Built-in*): fornece funcionalidades para interações com o sistema operacional.
- scikit-learn (versão 1.6.1): biblioteca para aprendizado de máquina, utilizada para técnicas como DBSCAN e KDTree.

C. Implementações realizadas

1) Modelo Inicial (Servidor): O processamento inicial dos dados do sensor LiDAR foi implementado com base no software LIDAR_LD06_python_loader, desenvolvido em Python [13], embarcado no robô. A arquitetura do software é constituída por dois componentes principais: o módulo *main*, que gerencia a interface de comunicação com o sensor LiDAR, realiza a aquisição dos dados em formato hexadecimal e os encaminha para o módulo CalcLiDARData. Este segundo componente executa a conversão dos dados hexadecimais para formato decimal e extrai os parâmetros fundamentais do pacote de dados do sensor. Durante este processamento, ocorre a conversão dos ângulos para radianos, e das distâncias para metros.

2) Arquitetura Cliente-Servidor Implementada: O sistema original foi adaptado para uma arquitetura cliente-servidor, visando otimizar a captura e o processamento dos dados. O servidor foi implementado como código principal, utilizando o protocolo TCP/IP (*main*) embarcado no robô, e realiza as seguintes operações:

- Estabelece conexão com o sensor LiDAR.
- Executa uma série de 200 leituras sequenciais.
- Encaminha cada leitura para processamento no módulo CalcLidarData.
- Agrupa conjuntos de dez leituras processadas.
- Transmite os dados agrupados para o cliente.

O segundo módulo é um cliente, que executa em um dispositivo externo (neste caso, um computador, cuja configuração está descrita no item 4 da Seção III, o qual processa os dados recebidos do servidor. Esta arquitetura distribuída transfere a carga computacional da placa do robô para um computador cliente dedicado, onde os dados são recebidos e armazenados em formato CSV. A Figura 5 apresenta o fluxo completo de dados no sistema cliente-servidor implementado.

3) Implementações para a combinação de leituras (Cliente): Com os principais dados armazenados em arquivo CSV, ângulo (radianos), distância (metros) e coordenadas x e y (metros), o sistema realiza o processamento e fusão de nuvens de pontos 2D através de diversas funções especializadas no cliente. A função `transform_points` aplica transformações geométricas nos pontos, permitindo translações, rotações e ajustes de escala para alinhar diferentes leituras do sensor realizadas de posições distintas. Esse alinhamento é necessário para realizar a identificação e *matches* de *landmarks*.

A identificação de *landmarks* é implementada pela função `extract_landmarks_improved`, que emprega o algoritmo DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) para agrupar pontos em *clusters* com base em sua densidade espacial. A abordagem incorpora um mecanismo adaptativo que otimiza o parâmetro *epsilon*[5] do

[5]Parâmetro que determina a distância máxima entre dois pontos para que sejam considerados vizinhos no espaço.

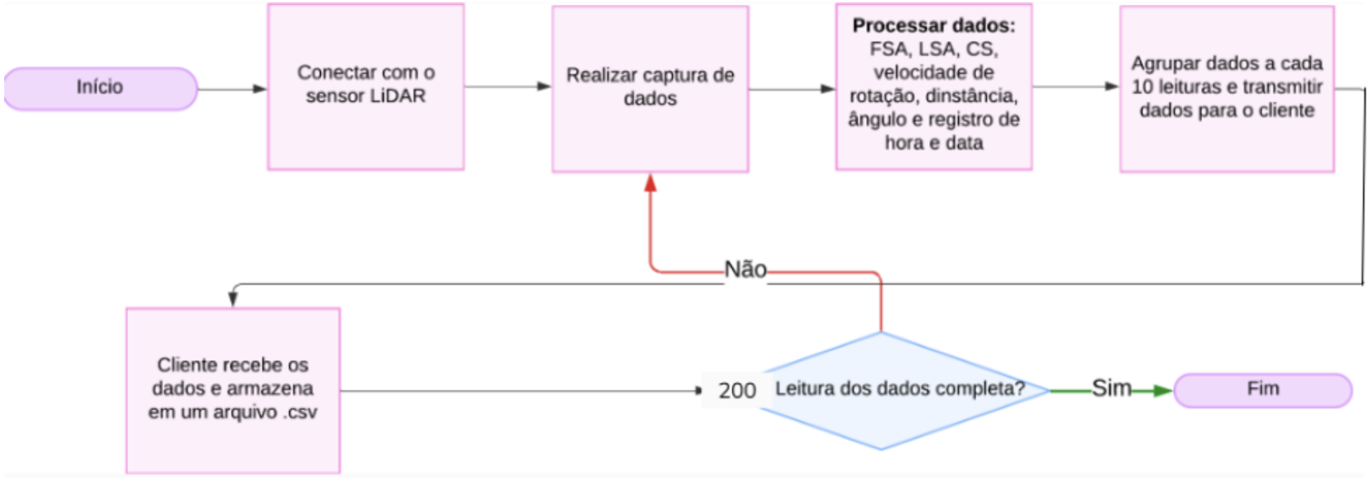


Figura 5. Diagrama de fluxo de dados para coleta e armazenamento de dados LiDAR. Fonte: Os autores.

DBSCAN através da estrutura KDTree[6]. Este método define dinamicamente o raio de vizinhança ao redor de cada ponto, adaptando-se à densidade local dos dados. Outro parâmetro do DBSCAN é número de pontos mínimos para formar cada *cluster*, definido no código como no mínimo 5 pontos. Após a formação dos *clusters*, calcula-se a mediana das coordenadas dos pontos em cada agrupamento, e este ponto mediano é designado como uma *landmark*.

O *match* entre *landmarks* existentes e recém-detectadas é realizado através da função `find_landmark_matches`, que utiliza KDTree para otimizar a busca por correspondências. O *matching* é baseado na distância euclidiana, calculada pela Equação 3.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3}$$

onde: (x_1, y_1) e (x_2, y_2) são as ordenadas das *landmarks* comparadas.

A visualização do ambiente, implementada pela função `visualize_landmarks_and_matches`, integra:

- Nuvem de pontos das duas leituras
- *Landmarks* identificadas em cada leitura
- Correspondências entre *landmarks* (representadas por linhas tracejadas)
- Posições do sensor nas diferentes leituras

[6]KDTree (*K-Dimensional Tree*) é uma estrutura de dados hierárquica que organiza pontos em um espaço k-dimensional. Sua eficiência para buscas de vizinhos próximos deriva da partição do espaço em regiões menores, reduzindo significativamente a complexidade computacional ao evitar comparações exaustivas entre todos os pontos.

A Figura 6 mostra o resultado do processo de identificação e atualização das *landmarks* em duas leituras e os *matches*.

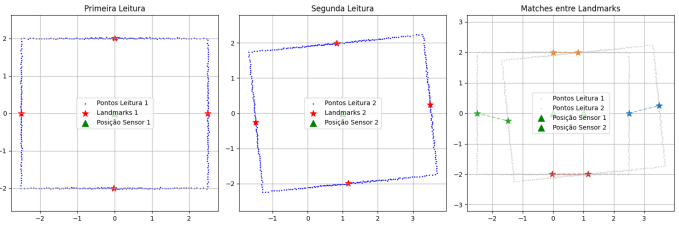


Figura 6. Visualização das *landmarks* identificadas na nuvem de pontos do LiDAR. Fonte: Os autores.

Para integrar as diferentes leituras do sensor em um mapa unificado, foi implementada uma função de combinação de nuvens de pontos que opera por meio de um algoritmo que analisa sistematicamente cada ponto da primeira nuvem, identifica possíveis correspondentes na segunda nuvem através de uma estrutura KDTree, e determina se estes estão dentro de um limiar de distância euclidiana pré-estabelecido. Quando correspondências são encontradas, o algoritmo calcula a mediana das coordenadas de todos os pontos relacionados, gerando uma representação mais robusta e precisa que a simples média aritmética, especialmente na presença de valores atípicos. Para pontos sem correspondência em ambas as nuvens, o método preserva suas coordenadas originais, garantindo que características únicas capturadas em apenas uma das leituras não sejam perdidas. Esta abordagem resulta em uma nuvem de pontos consolidada que maximiza a fidelidade da representação do ambiente, ao mesmo tempo em que minimiza a redundância

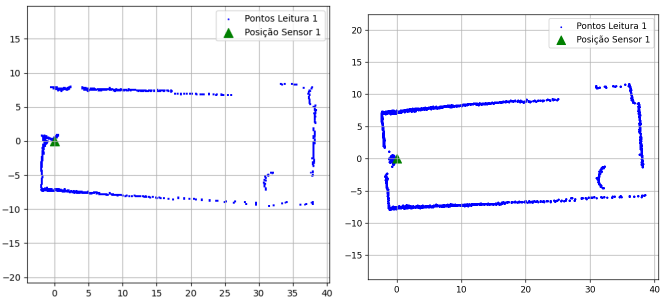
e os efeitos de possíveis pontos aleatórios.

IV. Resultados e Discussões

A. Mapeamento

1) Teste de processamento: O experimento para avaliar o processamento de captura de dados foi realizado com diferentes volumes de leituras, variando de 50 a 1000, permitindo identificar as limitações computacionais da Raspberry Pi. Durante os testes, observou-se que o consumo de CPU no cliente aumentou progressivamente de 3,8% com 20 leituras até atingir 20,7% com 1000 leituras, enquanto o uso de memória manteve-se relativamente constante entre 81,4% e 90,6%. Na Raspberry Pi, que atuou como servidor, o desempenho mostrou-se significativo mesmo considerando as limitações de hardware, com o consumo de CPU oscilando entre 7,2% e 18%, e uso de memória apresentando estabilidade entre 10,9% e 12,2% dos 4GB disponíveis. O tempo de processamento em ambos os dispositivos apresentou comportamento similar, alcançando aproximadamente 380 segundos ao processar 1000 leituras.

A análise dos resultados revelou um crescimento significativo na quantidade de pontos capturados, atingindo 6.006.000 pontos com 1.000 leituras, o que justifica a elevação no tempo de processamento. Concluiu-se que a faixa ideal de leituras situa-se entre 200 e 400 por captura, representando um equilíbrio entre o volume de dados capturados e o consumo de recursos computacionais. Capturas inferiores a 100 leituras não proporcionam dados suficientes para uma análise robusta do ambiente, enquanto aquelas acima de 400 leituras apresentam um aumento expressivo no tempo de processamento sem um ganho proporcional na qualidade dos dados coletados. Essa diferença pode ser observada nas Figuras 7a e 7b.



(a) Nuvem de pontos de 100 leituras (b) Nuvem de pontos de 200 leituras

Figura 7. Teste de Processamento de captura de dados. Fonte: Os autores.

2) Teste de ambiente: Para avaliar o ambiente, foram realizadas três sequências de mapeamento com a porta fechada e três com a porta aberta em diferentes posições (início, meio e fim), permitindo a comparação e análise das *landmarks*. Um desafio significativo foi a determinação do deslocamento do robô, que precisou ser medido manualmente devido à ausência de um sistema de localização preciso. No teste com a porta aberta, a primeira etapa de processamento, combinando as leituras da posição inicial e intermediária, mostrou desempenho consistente, com taxa de persistência[7] próxima a 0,30 e taxa de estabilidade[8] mantendo-se acima de 0,83. Ao incorporar a leitura da posição final, observou-se uma diminuição na taxa de persistência para aproximadamente 0,18, embora a taxa de estabilidade tenha apresentado leve melhora, alcançando 0,89. Este comportamento indica que, apesar da identificação de menos *landmarks* consistentes, aqueles identificados mantiveram excelente precisão posicional, como mostrado na Figura 8.

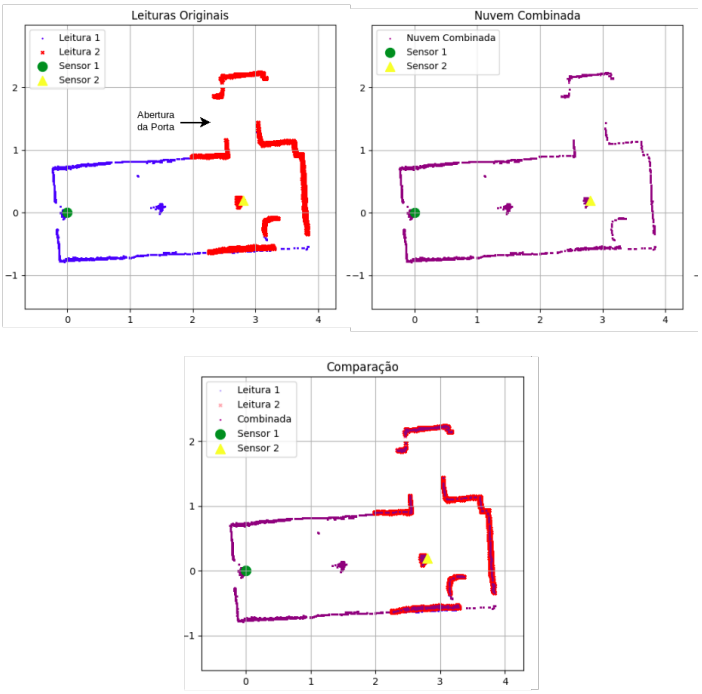


Figura 8. Resultado da combinação das leituras com a porta aberta: combinação das nuvens combinadas e a posição final. Fonte: Os autores.

O teste com a porta fechada (mostrado na Figura 9), executado da mesma maneira, apresentou na primeira etapa uma taxa de persistência variando entre 0,29 e 0,32, com taxa de estabilidade entre 0,83 e 0,85. A combinação com a leitura final mostrou melhoria significativa nas taxas de persistência, alcançando valores entre 0,44 e 0,49, enquanto a taxa de estabilidade manteve-se entre 0,67 e 0,70. O score médio de

[7]Proporção de *landmarks* que encontraram correspondência entre duas leituras.

[8]Proporção de *landmarks* correspondentes que mantiveram posições estáveis.

continuidade[9] permaneceu entre 0,43 e 0,46, indicando estabilidade na detecção de paredes no ambiente. Estes resultados evidenciam que o sistema consegue integrar múltiplas capturas mesmo em ambientes com pequenas alterações dinâmicas, como a mudança no estado da porta.

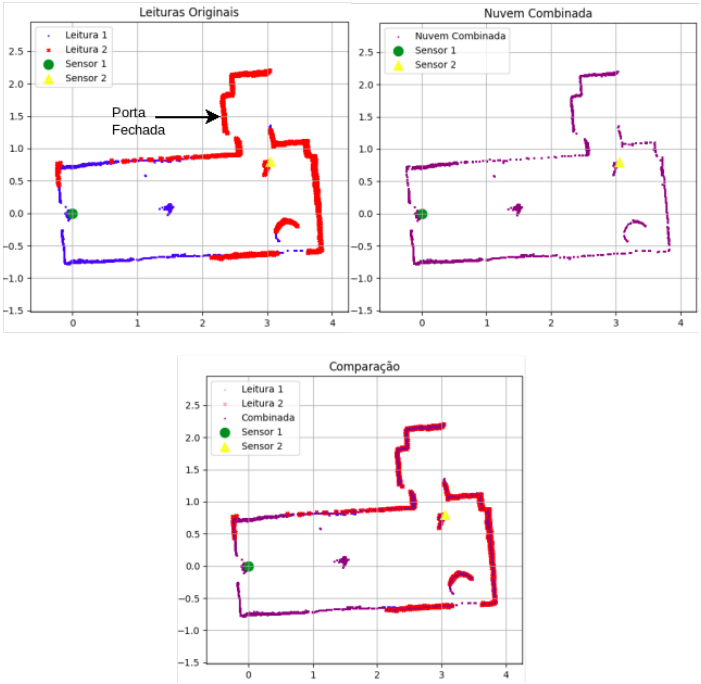
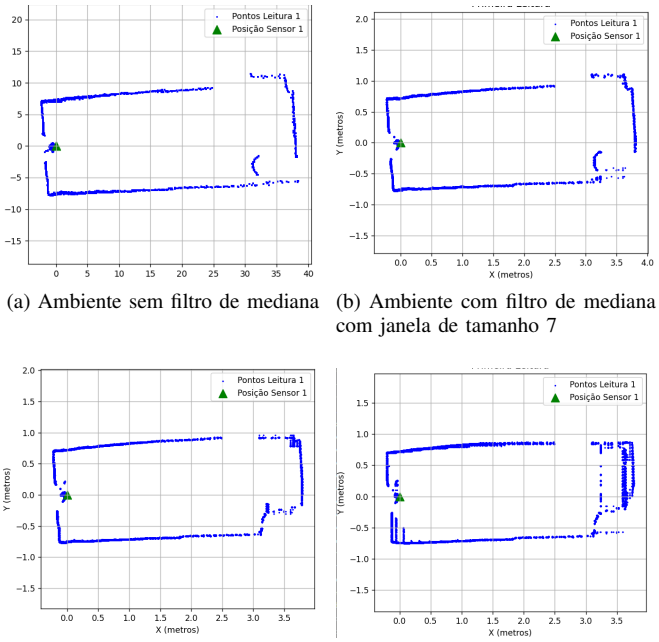


Figura 9. Resultado da combinação das leituras com a porta fechada: combinação das nuvens combinadas e a posição final. Fonte: Os autores.

3) Teste da taxa de tolerância: Para otimizar o *matching* de *landmarks*, testaram-se diferentes níveis de tolerância entre 0,1 e 1,0 unidade de distância euclidiana. Para tolerâncias muito baixas (0,1), notou-se alta precisão nas correspondências, com distância média de 0,018 e desvio padrão de 0,013, porém com número limitado de pontos incluídos (272). À medida que a tolerância aumentava, observou-se crescimento gradual tanto na distância média quanto no desvio padrão. A faixa de tolerância entre 0,2 e 0,3 proporcionou o melhor compromisso entre precisão e cobertura, enquanto valores acima de 0,5 resultaram em degradação significativa na precisão das correspondências. Assim, o valor de 0,25 foi utilizado nos testes de ambiente.

A avaliação do filtro de mediana como pré-processamento das nuvens de pontos mostrou que diferentes tamanhos de janela produzem resultados distintos. Janelas pequenas (3-11 pontos) apresentaram menor distorção dos dados originais (RMSE – *Root Mean Squared Error*) e redução de ruído moderada, preservando melhor os detalhes finos da geometria. Janelas médias (15-31 pontos) mostraram aumento gradual do RMSE e maior redução de ruído, especialmente no eixo Y, com início da perda de alguns detalhes geométricos. Já janelas grandes (51-101 pontos) resultaram em RMSE significativamente maior e perda considerável de detalhes da geometria original.

As imagens capturadas apresentam a leitura inicial e intermediária do ambiente, com diferentes tamanhos de janela para o filtro de mediana (7, 15 e 51) As Figuras 10a, 10b, 10c e 10d mostram o ambiente na posição inicial: primeiro sem filtragem e depois com a aplicação progressiva do filtro de mediana com diferentes parâmetros. Este ambiente de teste é relativamente simples, composto principalmente por paredes retas e um botijão de gás com formato semicircular como elemento de destaque.



(a) Ambiente sem filtro de mediana

(b) Ambiente com filtro de mediana com janela de tamanho 7

(c) Ambiente com filtro de mediana com janela de tamanho 15

(d) Ambiente com filtro de mediana com janela de tamanho 51

Figura 10. Comparação dos ambientes com e sem filtro de mediana. Fonte: Os autores.

Ao analisar as imagens sequencialmente, é possível observar os efeitos progressivos do filtro de mediana à medida que o tamanho da janela aumenta. Com janela de tamanho 7, nota-se uma sutil remoção de pontos dispersos e um leve refinamento na definição das paredes. Ao aumentar para janela 15, começa-se a perceber o início da suavização de cantos e pequenas descontinuidades. Já com janela 51, observa-se uma distorção significativa da geometria original, com cantos visivelmente arredondados e perda de detalhes estruturais importantes.

[9]Combinação ponderada de linearidade, uniformidade de espaçamento e consistência angular.

Um fator crítico a ser considerado é que o ambiente analisado, com aproximadamente 240 mil pontos e caracterizado predominantemente por paredes e estruturas lineares bem definidas, já apresenta um baixo nível de ruídos. Este fato explica os valores relativamente baixos de redução de ruído mesmo com janelas pequenas, como na janela de tamanho 3, onde a redução é de apenas 0,04% em X e 0,01% em Y.

Em ambientes com estas características geométricas bem definidas, o filtro de mediana com janelas grandes não atua apenas na remoção de ruídos, mas começa a modificar propriedades genuínas e estruturalmente relevantes do ambiente. Isso resulta na distorção de elementos importantes como cantos e intersecções entre paredes, que são características essenciais para uma representação fiel da geometria do ambiente. Esta degradação da fidelidade geométrica fica particularmente evidente na comparação visual entre as diferentes aplicações do filtro, especialmente ao observar como os cantos e bordas das paredes se tornam progressivamente mais arredondados e menos definidos com o aumento do tamanho da janela.

B. Navegação

A navegação do robô foi implementada utilizando uma abordagem baseada na varredura sistemática do mapa, permitindo que o agente percorresse as células livres em padrão de zigue-zague. Para tal, foram empregados mapas previamente processados e armazenados em arquivos CSV, fornecendo uma representação organizada e facilmente manipulável dos dados espaciais. Esta estratégia facilitou a localização do robô no ambiente e o controle preciso de seus movimentos, estabelecendo uma base para o planejamento de trajetória.

Durante a execução, o robô identifica cada célula livre no mapa e realiza o movimento correspondente, tomando decisões adaptativas com base na presença ou ausência de obstáculos detectados em tempo real. A integração do planejamento prévio com mecanismos de adaptação dinâmica permite que o robô não apenas siga um caminho predefinido, mas também responda rapidamente a mudanças inesperadas no ambiente, como obstáculos não mapeados, promovendo uma navegação eficiente e contínua. A Figura 11 apresenta a simulação do robô percorrendo o mapa em padrão de zigue-zague, mostrando a capacidade de evitar obstáculos e ajustar a trajetória conforme necessário[10].

A integração com o sensor LiDAR LD06 possibilitou a detecção precisa de obstáculos, independentemente de terem sido previamente mapeados. Quando uma barreira era identificada à frente, a navegação era temporariamente interrompida e a função de desvio era acionada. Esta função consiste em uma sequência estruturada de três manobras: desvio lateral, avanço paralelo ao obstáculo e retorno à linha original. Essa abordagem mostrou-se eficiente na maioria dos testes realizados, permitindo ao robô retomar sua trajetória de forma segura e previsível.

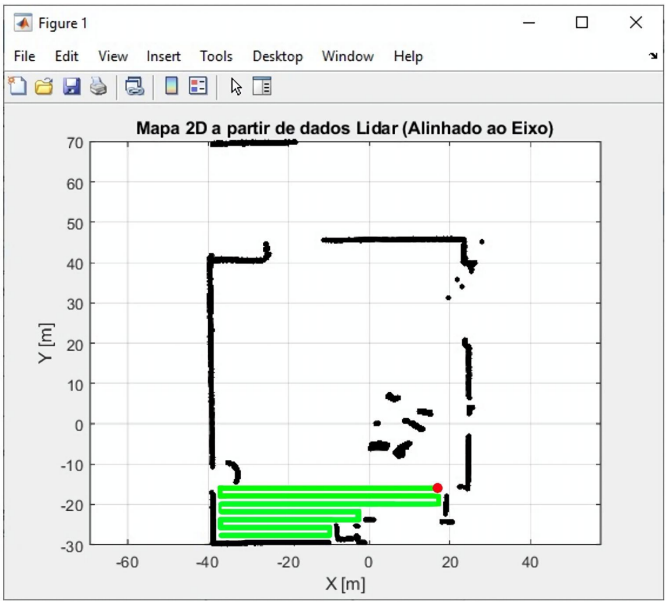


Figura 11. Movimentação e simulação do robô. Fonte: Os autores.

Os comandos de movimento são transmitidos via GPIO, utilizando controle PWM nos motores, o que garante transições suaves e controladas entre os estados de deslocamento. Entretanto, foram observadas pequenas variações na precisão das curvas e na distância percorrida por célula, indicando a necessidade de ajustes finos nos parâmetros de tempo e potência para calibração precisa.

Embora o sistema esteja funcional, diversas oportunidades de aprimoramento foram identificadas. Entre elas, destacam-se: a otimização da lógica de retorno ao trajeto após o desvio, a redução do tempo de execução das manobras e o aumento da suavidade das curvas. Adicionalmente, a realização de testes em ambientes mais complexos poderá revelar limitações não detectadas nos experimentos iniciais, fornecendo subsídios para melhorias futuras no desempenho do sistema.

V. Conclusão

Este trabalho desenvolveu e implementou um sistema de mapeamento e navegação com sensor LiDAR em um robô móvel de pequeno porte, focado na geração de mapas bidimensionais através de uma arquitetura cliente-servidor.

Os resultados mostram que o sistema atingiu seus objetivos principais com desempenho satisfatório na captura e processa-

[10] Ambiente de teste utilizado: sala quadrada contendo uma mesa, um armário e uma porta.

mento de dados espaciais. A arquitetura distribuída mostrou-se eficiente, permitindo que o robô (servidor) concentrasse-se na captura de dados enquanto o computador cliente realizava o processamento mais intensivo. Os testes identificaram que entre 200 e 400 leituras por captura oferece o melhor equilíbrio entre qualidade dos dados e consumo de recursos computacionais na Raspberry Pi 4 Model B.

O sistema mostrou robustez na detecção e *matching* de *landmarks*, apresentando taxas de persistência entre 0,29 e 0,49 e taxas de estabilidade entre 0,67 e 0,89, indicando boa precisão na identificação e localização dos pontos de referência. Verificou-se também que a implementação do filtro de mediana não é muito importante para ambientes com poucos ruídos e baixo dinamismo.

Embora os resultados obtidos tenham atendido às expectativas estabelecidas, é fundamental destacar a importância crítica de um dispositivo de localização preciso para o robô. A ausência de informações acuradas sobre o posicionamento no ambiente dificulta a combinação efetiva das leituras, sendo essencial para o correto alinhamento e integração dos dados coletados em diferentes posições. Esta dependência ressalta a necessidade de um aparelho de odometria ou localização para garantir a consistência do processo de mapeamento, especialmente quando se busca gerar um mapa global a partir de múltiplas leituras pontuais.

Como perspectivas para trabalhos futuros, pretende-se realizar testes em outros ambientes, estruturados de forma diferente e utilizando outras formas de movimentação no espaço de trabalho. Pretende-se também realizar comparações com outros sistemas similares, com objetivo de avaliar seu desempenho na produção dos mapas e na realização da navegação. Outro ponto a ser trabalhado futuramente é a inclusão de um sensor de odometria (*encoder*) para melhorar a precisão da localização do robô e a consequente realização de outros experimentos para verificar o impacto destas alterações no sistema aqui apresentado.

Agradecimentos

Ao MEC-SESU, pelo financiamento deste projeto de iniciação científica a partir do Programa de Educação Tutorial (PET).

Referências

[1] C. L. S. LDROBOT, *LiDAR LD06 Development Manual*, 2021, accessado: 04-09-2024. [Online]. Available: https://storage.googleapis.com/mauser-public-images/prod_description_document/2021/315/8fcea7f5d479f4f4b71316d80b77ff45_096-6212_a.pdf

[2] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. [s.l.]: MIT Press, 2004.

[3] P. Dong and Q. Chen, *LiDAR Remote Sensing and Applications*. Londres, England: CRC Press, 2017.

[4] F. J. V. do Coito, "Sensor de distância por infravermelhos para a caracterização do espaço de trabalho," Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Setembro 2018.

[5] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[6] Ł. Sobczak *et al.*, "Lidar point cloud generation for slam algorithm evaluation," *Sensors*, vol. 21, no. 10, p. 3313, 2021.

[7] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 1989, pp. 1166–1172.

[8] S. Thrun and A. Bücken, "Integrating grid-based and topological maps for mobile robot navigation," in *Proceedings of the 13th National Conference on Artificial Intelligence*. AAAI Press, 1996.

[9] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Menlo Park, CA, USA: IEEE, 2011.

[10] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[11] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[12] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[13] HENJIN0, "Lidar ld06 python loader," GitHub repository. https://github.cohenjin0/LIDAR_LD06_python_loder., 2021, acessado em 12 jun. 2025.