INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

COORDENAÇÃO DO CURSO DE ENGENHARIA ELÉTRICA

Matheus Lucas de Lucena Pereira

Implementação de Algoritmo de Planejamento de Trajetória com Robô Diferencial

João Pessoa
2022

Matheus Lucas de Lucena Pereira

Implementação de Algoritmo de Planejamento de Trajetória com Robô Diferencial

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, como parte dos requisitos para a obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Patric Lacouth

João Pessoa
2022

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Nilo Peçanha do IFPB, *campus* João Pessoa

P436i Pereira, Matheus Lucas de Lucena.

Implementação de algoritmo de planejamento de trajetória com robô diferencial / Matheus Lucas de Lucena Pereira. – 2022.
65 f. : il.
TCC (Graduação – Engenharia Elétrica) – Instituto Federal de Educação da Paraíba / Coordenação do Curso Superior de Engenharia Elétrica, 2022.
Orientação : Profª D.r Patric Lacouth.

1. Algoritmo de planejamento de trajetória - análise. 2. Algoritmo A- star Search. 3. Inteligência artificial. 4. Robô diferencial. I. Título.

CDU 004.021 (043)

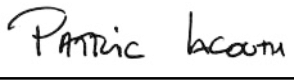
Lucrecia Camilo de Lima
Bibliotecária – CRB 15/132

Matheus Lucas de Lucena Pereira

Implementação de Algoritmo de Planejamento de Trajetória com Robô Diferencial

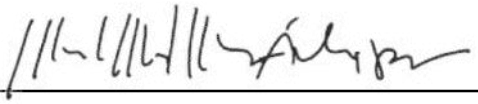
Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, como parte dos requisitos para a obtenção do grau de Engenheiro Eletricista.

BANCA EXAMINADORA



Prof. Patric Lacouth, Dr. - IFPB
Orientador

Prof. Lincoln Machado de Araújo, Dr. - IFPB
Membro da Banca



Prof Marcelo Magalhaes Avila Paz, Dr - IFPB
Membro da Banca

João Pessoa, 07 de fevereiro de 2022.

Agradecimentos

Primeiramente, gostaria de agradecer a Deus, por me permitir trilhar este caminho, ultrapassando os obstáculos encontrados. Em segundo lugar, agradecer a minha família por todo o apoio que sempre me deram, em especial, ao meu pai, Marinézio, que infelizmente não chegou a ver este trabalho finalizado, a minha mãe, Luciana e ao meu irmão, Mário.

Também gostaria de agradecer a minha noiva, Dayanne Regina, por todo o incentivo, tão precioso, durante o curso, aos meus caros colegas, Ruan Castro, Mateus Gomes e Carlos Henrique, por todas as horas e estudos que compartilhamos neste percurso e ao Profº Dr Patric Lacouth, o qual a ajuda e orientação foi indispensável no desenvolvimento deste trabalho.

AUTOR

RESUMO

Neste trabalho estão descritas a implementação e a análise de um algoritmo de planejamento de trajetória, utilizando como base o Algoritmo *A-Star Search*, e algumas de suas variações, para encontrar o caminho de menor custo, bem como evitar obstáculos, em um mapa previamente conhecido, representado por uma matriz, cujas células seriam interpretadas como nós de um grafo pelo algoritmo. A implementação foi realizada por meio de um um robô diferencial, com a movimentação em 4 direções, controlado por uma *Raspberry PI 1 Model B Rev 2*, utilizando Algoritmos de Inteligência Artificial.

Palavras-chave: Planejamento de Trajetória, A-Star Search*, Inteligência Artificial*

ABSTRACT

This work describes the implementation and analysis of a path planning algorithm, based on the A-Star Search Algorithm, and some of its variations, to find the lowest cost path, as well as avoid obstacles, in a previously known map, represented by a matrix, whose cells would be interpreted as nodes of a graph by the algorithm. The implementation was carried out through a differential robot, with movement in 4 directions, controlled by a *Raspberry PI 1 Model B Rev 2*, using Artificial Intelligence Algorithms.

Keywords: Path Planning, A-Star Search, Artificial Intelligence

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	*Associação Brasileira de Normas Técnicas*
IFPB	*Instituto Federal de Educação, Ciência e Tecnologia da Paraiba*
IFRN	*Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte*
pxWU	*Piecewise Convex Upward (COnvexo por partes Ascendentes)*
pxWD	*Piecewise Convex Downward (Convexo por partes Descendentes)*
GPIO	*General Purpose Input/Output* - Entrada/Saída de uso geral
UART	*Universal asynchronous receiver/transmitter* - Transmissor/Receptor Universal Assíncrono
I2C	*Inter-Integrated Circuit* - Circuito Inter Integrado
SPI	*Serial Peripheral Interface* - Interface Periférica Serial
USB	*Universal Serial Bus* - Barramento Serial Universal
LED	*Light Diode Emissor* - Diodo emissor de luz
RAM	*Random Access Memory* - Memória de Acesso Aleatório
CPU	*Central Process Unit* - Unidade Central de Processamento
GPU	*Graphics Process Unit* - Unidade de Processamento Gráfico

SUMÁRIO

1 INTRODUÇÃO

O uso de algoritmos de Inteligência Artificial visando a otimização de processos e a automação das tarefas, tem sido crescente nas últimas décadas, indo desde a criação de carros e robôs autônomos, até aplicações em jogos eletrônicos, para determinar os movimentos dos personagens. Para tal, se fez necessário o uso de algoritmos que otimizem a busca pelo melhor trajeto que possa ser percorrido entre 2 ou mais pontos, evitando obstáculos, e/ou reduzindo os custos de locomoção e/ou podendo levar em consideração os diferentes tipos de terreno pelo trajeto.

O objetivo deste trabalho foi implementar um algoritmo de planejamento de trajetória que pudesse ser aplicado em robôs autônomos num cenário real, como um cenário industrial, que apresentasse uma alta eficiência em encontrar a rota com menor custo e em desviar de obstruções, controlado por meio de um *Raspberry PI 1 Model B Rev 2*, microcomputador amplamente utilizado para prototipagem, e que não exija um grande poder computacional, como os métodos de Aprendizagem de Máquina, podendo ser aplicados em robôs mais simples, como o próprio protótipo de robô diferencial utilizado.

Deste modo, a família de algoritmos conhecida como *A-Star Search*, (HART; NILSSON; RAPHAEL, 1968), se mostrou uma alternativa interessante para ser utilizada como base deste trabalho. Inicialmente proposto em 1968, o algoritmo utiliza funções heurísticas para determinar o melhor trajeto em um determinado mapa, cuja representação possa ser feita por meio de um ou mais grafos ou mesmo através de uma representação matricial do ambiente, onde cada nó ou célula, apresentam os custos relativos a sua posição em relação ao ponto inicial e ao destino pretendido.

O Algoritmo *A-Star Search* pode ser descrito como "um algoritmo de busca heurística altamente eficiente em encontrar um caminho viável e de baixo custo, usando uma função de avaliação para analisar os nós", em tradução livre, (YAO; BINBIN; QINGDA, 2009).

Este trabalho está organizado da seguinte forma: no Capítulo 1, são apresentados alguns aspectos introdutórios a cerca dos algoritmos de planejamento de trajetória, no Capítulo 2, é apresentada a fundamentação teórica, no Capítulo 3 é descrita a metodologia, no Capítulo 4, são mostrados os resultados obtidos e no Capítulo 5, são apresentadas as conclusões do trabalho.

2 Fundamentação teórica

Neste capítulo, são descritos os embasamentos teóricos utilizados para o desenvolvimento do algoritmo e do protótipo.

2.1 Algoritmos de Planejamento de Trajetória

Os algoritmos de planejamento de trajetórias são utilizados para a otimização da escolha de um determinado percurso, levando em consideração fatores como a distância percorrida, o tempo o percurso, características geográficas do trajeto, bloqueios encontrados, dentre outros, que impactam na escolha do melhor trajeto. Cabe a tais algoritmos, encontrar caminhos diferentes, classificar e calcular o custo final do caminho.

> "O planejamento de trajetória refere-se a, em um ambiente coexistente com obstáculos estáticos e dinâmicos, se o robô pode encontrar um caminho, desde o início até o fim determinado, que atenda a certos critérios de avaliação, enquanto o robô possa evitar todos os obstáculos com segurança e confiabilidade durante a viagem", em tradução livre, (ZHANG; ZHAO, 2014).

Os algoritmos de planejamento de trajetória clássicos são o Algoritmos de Dijkstra, (DIJKSTRA, 1959) e o Algoritmo *A-Star Search*, (HART; NILSSON; RAPHAEL, 1968), tema de estudo deste trabalho, pois são algoritmos de implementação simples, que podem resolver uma grande gama de problemas, em especial o Algoritmo *A-Star Search*, que recebeu variações, conforme será visto adiante, resultando em algoritmos ainda mais rápidos e com uma menor quantidade de expansões de nós.

2.2 Algoritmo de Dijkstra

O Algoritmo de *Dijkstra*, (DIJKSTRA, 1959), construído por Edsger Dijkstra, é um dos outros algoritmos clássicos de planejamento de trajetória, que utiliza o conceito de grafos, para encontrar o caminho de menor custo entre 2 nós. O Algoritmo considera os custos entre os nó de origem e o nó n a ser analisado, denominado $g(n)$, de maneira encontrar um trajeto que possua o menor custo entre a origem o nó de destino, não levando em consideração o custo entre o nó atual e o nó de destino.

Desta forma, o custo para um nó no Algoritmo de Dijkstra é descrito pela Eq. 2.1:

$$f(n) = g(n) \tag{2.1}$$

2.3 A-Star Search

O *A-Star Search* foi proposto como uma maneira de encontrar o caminho com o menor custo, em um dado mapa, representado por grafos ou matrizes, cuja configuração é conhecida, ou seja, "um algoritmo geral que prescreve como usar essas informações para encontrar um caminho de custo mínimo por meio de um grafo.", em tradução livre, (HART; NILSSON; RAPHAEL, 1968).

O Algoritmo foi escolhido, tendo em vista que o objetivo do trabalho foi a busca do caminho com menor custo, no caso, a menor distância, em um mapa prédeterminado, para um robô diferencial de baixo custo, com baixo poder computacional.

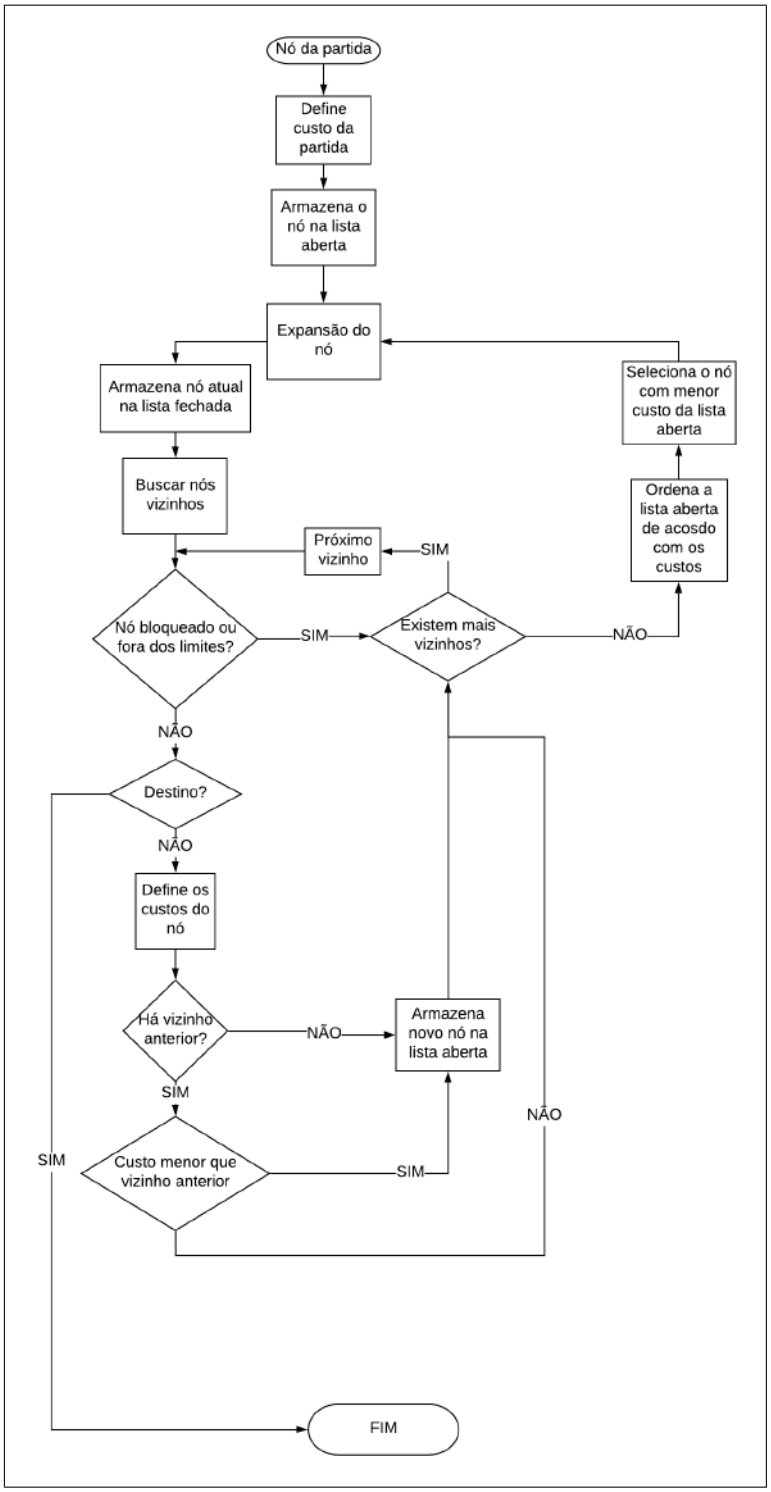
De maneira similar ao algoritmo de *Dijkstra*, (DIJKSTRA, 1959), que calcula o custo apenas da partida até o nó atual, $g(n)$, levando a um número alto de expansão de nós e uma baixa eficiência computacional, o *A-Star* utiliza uma função heurística para mensurar o custo do nó atual até o nó de destino, denominado de $h(n)$, reduzindo o número de expansões necessárias.

O seu funcionamento pode ser descrito da seguinte forma:

1. No nó inicial, são definidos os custos, como sendo $g(n) = 0$, pois a partida é o nó atual, e $h(n)$ sendo definido pela função heurística utilizada;
2. Definidos os custos do nó de partida, o nó é armazenado na lista aberta, onde são armazenados os nó, cujos vizinhos estão serão analisados;
3. O nó é então expandido, para que sejam buscados os seus vizinhos;
4. Para cada vizinho encontrado, são realizadas as verificações se o vizinho é válido, ou seja, se está dentro dos limites do mapas e se não há nenhum obstáculo no mesmo, ou se é o nó de destino;
5. São definidos então os seus custos, e feita a comparação com os outros vizinhos, a fim de encontrar aquele que possui o menor custo;
6. Encontrado o vizinho como menor custo, o nó atual, é então, armazenado em uma denominada lista fechada, onde são armazenados os nós cujos vizinhos já foram analisados;
7. Essa expansão é então realizada para todos os nós subsequentes. Caso o caminho esteja bloqueado, são então reabertos os nós anteriores, e verificados seus vizinhos;
8. Caso o próximo nó encontrado seja o nó de destino, o algoritmo é encerrado e é traçado o trajeto.

Assim, pode-se resumir o funcionamento do Algoritmo nas seguintes etapas, apresentadas na Figura 1:

Figura 1 – Fluxograma do Algoritmo *A-Star Search*



FONTE:Autoria Própria

Conforme (HART; NILSSON; RAPHAEL, 1968), para um dado nó n do grafo, a função que define o seu custo é dada pela soma dos fatores $g(n)$ e $h(n)$ conforme a Eq.

2.2

$$f(n) = g(n) + h(n) \tag{2.2}$$

A função $g(n)$ representa um somatório do deslocamento entre o nó atual e o nó de partida. A cada deslocamento entre células, é somado 1 ao valor atual de $g(n)$. A função heurística $h(n)$, como explicado no tópico a seguir, pode ser alterada de acordo com o o conhecimento que se possui do ambiente real, (HE; WANG; CAO, 2012), sendo comumente expressa pela função de distância de Manhattan e pela função de distância Euclidiana, (ZHANG et al., 2021)..

Como dito, o Algoritmo *A-Star Search*, é na verdade, um conjunto de algoritmos de planejamento de trajetória, visto que existem outras funções heurísticas, $h(n)$, a serem utilizadas, de acordo com cada caso, bem como, variações da função de custo que podem ser empregadas, com a finalidade de reduzir o número de nós a serem expandidos, diminuindo o tempo de processamento do algoritmo. Algumas dessas variações, aquelas que foram implementadas no decorrer do projeto, serão demonstradas nos tópicos a seguir.

2.3.1 Funções Heurísticas

Funções heurísticas são funções estimadoras de estados, ou sejam, que tentam quantificar o estado atual de um objeto. Para o Algoritmo *A-Star Search*, tal estado é definido como o custo do nó atual, até o nó de destino. "As funções heurísticas são estimativas do número de fronteiras que ainda precisam ser percorridas para alcançar um nó objetivo.", em tradução livre, (POHL, 1970).

As funções heurísticas utilizadas são estritamente relacionadas aos mapas e grafos aos quais o algoritmo será implementado, bem como as limitações inclusas no problema a ser resolvido. As funções comumente utilizadas, são as funções de distância de *Manhattan* e a distância Euclidiana.

2.3.1.1 Função de distância de Manhattan

A função de distância de Manhattan pressupõe que só possam ser realizados movimentos nas 4 direções, não podendo ser realizados movimentos diagonais. Assim, o custo entre o nó atual e o nó de destino, pode ser calculado utilizando o módulo da soma das diferenças entre as coordenadas dos dois pontos A e B quaisquer, de acordo com a distância de *Manhattan*, Eq. 2.3:

$$h(n) = |(A_i - B_i)| + |(A_j - B_j)| \tag{2.3}$$

Onde A_i e A_j são as coordenadas do nó A, e B_i e B_j são as coordenadas do nó B.

2.3.1.2 Função de distância Euclidiana bidimensional

A função de distância Euclidiana bidimensional, por sua vez, indica que movimentos na diagonais podem ser realizados, portanto, a distância entre o nó A e o nó B, pode ser calculada utilizando o Teorema de Pitágoras, na forma da Eq. 2.4:

$$h(n) = \sqrt{(A_i - B_i)^2 + (A_j - B_j)^2} \tag{2.4}$$

2.3.2 Variações do Algoritmo A-Star Search

2.3.2.1 Algoritmos Ponderados Simples

O Algoritmo *A-Star Search* Ponderado simples é um dos muitos algoritmos que utiliza um "parâmetro de peso que pode ser usado para controlar o equilíbrio entre a qualidade da solução e o esforço de pesquisa.", em tradução livre, (WILT; RUML, 2012).

Ponderar a função heurística, ou seja, multiplicar o valor de $h(n)$ por uma constante $w > 1$, aumentando a sua contribuição no custo, resulta numa diminuição no número de nós abertos durante a busca, acelerando o planejamento da trajetória. Assim, a função custo torna-se a Eq. 2.5:

$$f(n) = g(n) + w * h(n) \quad w \in [1, \infty] \tag{2.5}$$

Porém, em contrapartida, acarreta na possibilidade de que o caminho encontrado possa não ser o de menor custo, devido a admissibilidade do algoritmo poder ser afetada, pois, devido a redução no número de nós expandidos, o algoritmo pode encontrar o destino antes de percorrer todos os trajetos possíveis, interrompendo a busca pelo melhor trajeto, conforme mostrado em (LIKHACHEV; GORDON; THRUN, 2003).

2.3.2.2 Funções definidas em trechos

Funções definidas por trechos são descritas como funções cuja sentença depende do valor da variável independente, cada qual possuindo seu próprio subdomínio, contidos no domínio da função.

Utilizando este conceito, foram propostos os conjuntos de Equações denominadas como Convexas por Partes Ascendentes e Descendentes, (CHEN; STURTEVANT, 2021), para definir o valor dos custo de cada nó, de forma a reduzir o significativamente o números de expansões de nó e evitar a sua reabertura.

Esses conjuntos de equações, conforme mostrado nas secções a seguir, alteram dinamicamente o valor dos pesos associados a função de custo, de acordo com a região do grafo em que o nó analisado se encontra.

2.3.2.3 Piecewise Convex Upward

A modalidade *Piecewise Convex Upward (pxWU)* - Convexo por partes Ascendentes, em tradução livre, (CHEN; STURTEVANT, 2021), reduz a contribuição da função heurística na função custo, quando o algoritmo está analisando nós próximos ao nó de destino. A função custo nessa configuração é dadas pela Eq. 2.6:

$$f(n) = \begin{cases} \frac{g(n)}{K} + h(n), & se\ g(n) < \frac{K(w-1)}{K-w}h(n) \\ \frac{(g(n)+h(n))}{w}, & se\ g(n) \geq \frac{K(w-1)}{K-w}h(n) \end{cases} \tag{2.6}$$

Conforme (CHEN; STURTEVANT, 2021), foi usado um fator $K = 2w - 1$, resultando na Eq 2.9:

$$f(n) = \begin{cases} \frac{g(n)}{(2w-1)} + h(n), & se\ g(n) < (2w-1)h(n) \\ \frac{(g(n)+h(n))}{w}, & se\ g(n) \geq (2w-1) * h(n) \end{cases} \tag{2.7}$$

2.3.2.4 Piecewise Convex Downward

O algoritmo *Piecewise Convex Downward (pxWD)* - Convexo por partes Descendentes, em tradução livre, (CHEN; STURTEVANT, 2021) incrementa a contribuição da função heurística na função de custo, quando próximo ao nó de destino, tendo a função custo calculada pela Eq. 2.8.

$$f(n) = \begin{cases} g(n) + h(n), & se\ g(n) < \frac{K-w}{w-1}h(n) \\ \frac{(g(n)+Kh(n)}{w}, & se\ g(n) \geq \frac{(K-w)}{w-1}h(n) \end{cases} \tag{2.8}$$

Novamente conforme (CHEN; STURTEVANT, 2021), utilizando $K = 2w - 1$, têm-se a Eq 2.9:

$$f(n) = \begin{cases} g(n) + h(n), & se\ g(n) < (2w-1)h(n) \\ \frac{(g(n)+(2w-1)h(n))}{w}, & se\ g(n) \geq (2w-1)h(n) \end{cases} \tag{2.9}$$

Conforme será visto na secção 4, a alteração na função de custo resultou em uma diminuição do número de nós expandidos, como mostra a Figura 31, e por consequência, no tempo de execução do algoritmo.

2.4 Linguagem *Python*

Criada em 1990 por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciências da Computação da Holanda, a linguagem Python vem se destacando, principalmente nas áreas de Inteligência Artificial e Desenvolvimento *WEB*, devido a sua sintaxe clara, que facilita a compreensão dos algoritmos. "Python é uma linguagem de

altíssimo nível,orientada a objeto de tipagem dinâmica e forte, interpretada e interativa", (BORGES, 2014).

Outra característica importante da linguagem *Python*, é a grande quantidade de bibliotecas já desenvolvidas para a mesma, permitindo os mais diversos usos para a linguagem, que vão desde os cálculos mais básicos, até a implementação de algoritmos avançados de IA, como *Deep Learning*.

A versão utilizada no projeto foi a 3.8.8, lançada em 19 de fevereiro de 2021.

2.4.1 Bibliotecas Utilizadas

As bibliotecas utilizadas na implementação do trabalho foram a *Time*, a *Numpy* e *RPI.GPio*, que serão descritas a seguir.

2.4.1.1 Time

A biblioteca *Time* é uma biblioteca que disponibiliza funções relacionadas ao tempo, como a obtenção da contagem de tempo atual, até conversões entre padrões de representação de tempo.

2.4.1.2 Numpy

A biblioteca *Numeric Python (Numpy)*, é amplamente utilizada no tratamento de dados, pois possui uma grande gama de operações matemáticas implementadas, bem como suporte para cálculos com vetores e matrizes.

> "*Numpy* provém matrizes, *in-memory*, multidimensionais, homogeneamente tipados (ou seja, *single-pointed* e *strided*), na Unidade Central de Processamento (CPU), Ela funciona em dispositivos, desde dispositivos embarcados até os maiores supercomputadores do mundo, com a performance próxima a de linguagens compiladas. Pela maior parte da sua existência, *Numpy* abordou a grande maioria dos casos de uso de matrizes na computação.", em tradução livre, (HARRIS et al., 2020).

No projeto, a biblioteca foi utilizada principalmente para ler o arquivo *CSV*, com a representação matricial do ambiente, e manipular de forma eficiente a matriz, durante o processo de otimização do algoritmo.

2.4.1.3 RPi.GPIO

Biblioteca que permite o acesso e controle das *General Purpose Inputs/Outputs - (GPIOs)*, entradas e saídas de uso geral da *Raspberry Pi*. Também possui funções para o uso da Modulação *PWM*, desde que o pino utilizado seja compatível com a mesma.

Embora a biblioteca apresente um controle mais facilitado das *GPIOs*, a mesma apresenta problemas quando utilizada em aplicações onde o tempo é crucial, e, conforme será visto mais adiante, apresentou erros para o controle de velocidade dos motores.

2.5 Robôs Móveis

A robótica tem sido, nas últimas décadas, umas da áreas de pesquisa mais importantes, pois a crescente complexidade dos sistemas exige, cada vez mais, robôs com movimentações mais rápidas e precisas.

> "Um robô móvel é um agente autônomo capaz de extrair informações do ambiente e utilizar esse conhecimento do mundo para deslocar-se com segurança de modo significativo e intencional, atuando e executando tarefas."(JúNIOR, 2006).

O desenvolvimento de robôs móveis e autônomos ou semiautônomos, capazes de movimentar-se sem ou com pouca a necessidade de um condutor humano, reduzindo assim, o risco de acidentes, decorrentes de ações humanas, tem sido um desafio para os pesquisadores, pois requerem um sistema de controle robusto, que necessita da integração de vários sistemas, como o de planejamento de sua trajetória, incluindo obstáculos estáticos e móveis, o de controle de sua velocidade, para que não haja desvios no trajeto calculado, devido a alguma falha ou divergência nos motores, e os demais sistemas responsáveis pelas outras tarefas que possam vir a serem executadas pelo robô.

Na área da robótica móvel, existem 2 tipos principais de robôs, sendo eles, os Veículos Automaticamente Guiados - *Automated Guided Vehicle* (AGV), e os Robõs Móveis Autônomos - *Autonomous Mobile Robotics* (AMR).

2.5.1 Veículos Automaticamente Guiado (AGVs)

Na área da robótica móvel, os AGVs, são veículos capazes de seguir um determinado trajeto pré determinado, por meio de alguma marcação ou objeto inserido no ambiente, como um faixa ou um fio, por exemplo, onde o "sistema central de processamento do AGV emite o comando de direção e comando de velocidade", (DAS, 2016). Sua utilização vai desde a indústria, realizando o transporte de produtos e materiais internamente, (MüLLER, 2014), até entregas, ou até mesmo na área da saúde, conforme mostra (PEDAN; GREGOR; PLINTA, 2017), onde os mesmo são estudados para realizar a entrega de medicamentos, alimentação, amostras para análise, dentre outros, permitindo a alocação de profissionais para a realização de tarefas mais cruciais e menos repetitivas.

Os AGVs, contam, em geral, com sistemas de Navegação, que corrigem o percurso do veículo durante o trajeto, sistema de comando de direção, de planejamento de trajetória e o sistema de controle de tráfego, para impedir possíveis colisões.

2.5.2 Robôs Móveis Autônomos (AMRs)

Os AMRs, podem ser descritos como sendo "um sistema robótico que pode se mover de forma autônoma para fazer algumas tarefas sem qualquer interferência do ser humano", em tradução livre, (WIJAYA; ARTHAYA; SADIYOKO, 2006).

Para tal, diferentemente dos AGVs, que necessitam de uma rota pré-determinada por meios físicos, faz-se necessários algoritmo que consigam realizar o planejamento de sua trajetória, e se necessário for, realizar o sensoreamento durante o percurso, para evitar possíveis obstáculos dinâmicos e colisões, durante o percurso.

Desta forma, seu sistema de planejamento pode ser dividido em três sistemas, o sistema de planejamento global, que trata das informações necessárias para completar uma tarefa, como uma mapa do ambiente com os obstáculos fixos, o sistema de modelagem de mundo, para construir um modelo do ambiente em que o robô está inserido, por meio dos dados coletados pelo mesmo, de forma a melhorar o seu desempenho, adicionando novos dados ao mapa, e o sistema de reações locais, para evitar possíveis obstáculos dinâmicos, conforme (WIJAYA; ARTHAYA; SADIYOKO, 2006).

Os AMRs tem sido utilizados para o transporte de materiais, mercadorias e até medicamentos, para atividades que oferecem risco a vida, como "desativação de explosivos reconhecimento, vigilância, resgate, atividades militares, e algumas intervenções que incluem atividades biológicas ou contaminação química", em tradução livre, (CARDONA et al., 2020), e mais recentemente, para a higienização de ambientes, mesmo hospitalares, (BAčíK et al., 2020), que confirmam a sua eficácia na realização das tarefas designadas.

2.6 RaspBerry PI 1 Model B Rev 2

A família de computadores *Raspberry Pi* são de microcomputadores de baixo custo, desenvolvidas pela *Raspberry Pi Foundation*,(FOUNDATION, 2021), de tamanho semelhante a um cartão de crédito, com intuito de atrair jovens para o mercado de tecnologia.

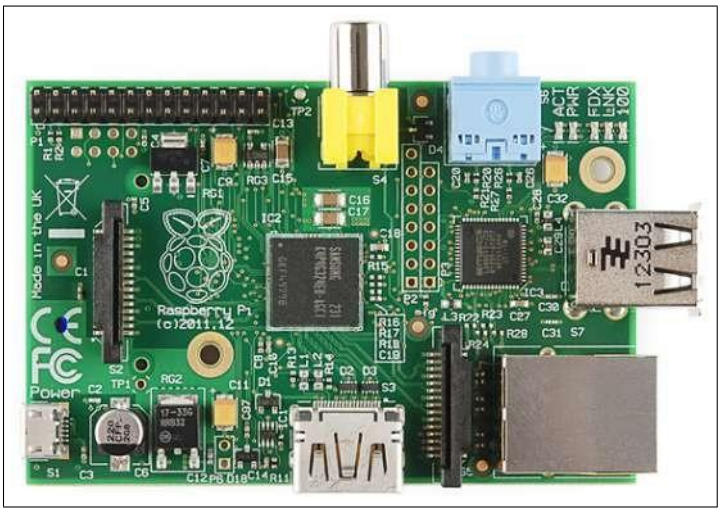
> "O Raspberry Pi foi desenvolvido como um dispositivo educacional e foi inspirado pelo sucesso do BBC Micro para o ensino de programação para uma geração. A *Raspberry Pi Foundation* estabeleceu fazer o mesmo para o mundo de hoje, aonde você não precisa saber como escrever um programa para usar um computador.", em tradução livre, (HARRINGTON, 2015).

O modelo Utilizado no trabalho foi o *Raspberry PI 1 Model B Rev 2*, Figura 2, que conta com *System on Chip (SoC)*, Broadcom BCM2835, que contém um processador ARM1176JZF-S de 700 Mhz, com arquitetura ARMv6, e uma *Graphics Process Unit - (GPU)* Broadcom VideoCore IV. A Raspberry ainda possui 512 MBs de memória *RAM*, com suporte para um cartão SD de até 2 GB e 2 portas USB 2.0.

O microcomputador conta ainda com 26 *GPIOs*, com os os pinos *Universal asynchronous receiver/transmitter* (UART), Transmissor/Receptor Universal Assíncrono, *Inter-Integrated Circuit* (I2C), Circuito Inter Integrado, *Serial Peripheral Interface* (SPI), Interface Periférica Serial, e os pinos de alimentação de 3.3 e 5V, conforme mostrado na Figura 3. No trabalho, foram utilizadas 4 *GPIOs*, para controle dos motores diferenciais.

A principal característica da *Raspberry Pi* é a possibilidade de se usar um sistema operacional. O sistema operacional utilizado no projeto, foi o *Linux*, em especial a distribuição *Raspian*, executada por meio de um cartão *Secure Digital* (SD). Esta distribuição foi criada, em especial, para o *Raspberry Pi*, com base na distribuição *Debian*, e conta com configurações que tornam a utilização do microcomputador mais simples e intuitiva.

Figura 2 – *RaspBerry PI 1 Model B Rev 2*



FONTE:(SPARKFUN, 2014)

Componente	Quantidade
ARM1176JZF-S (CPU)	1
Broadcom VideoCore IV (GPU)	1

Tabela 1 – Unidades de Processamento do Broadcom BCM2835,

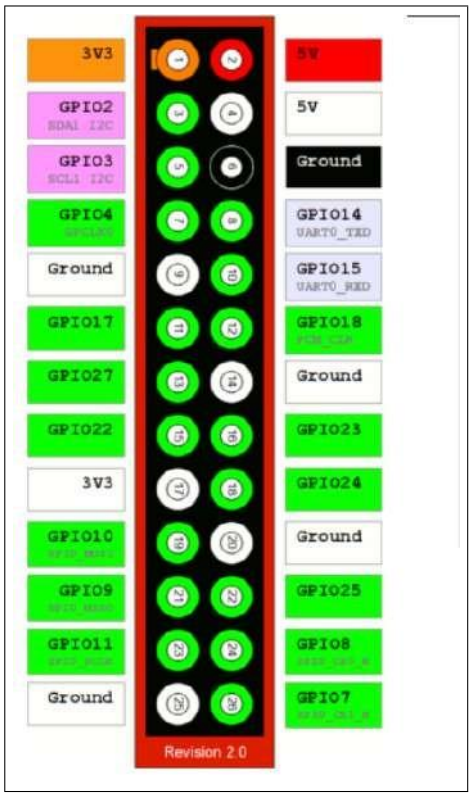
Componente	Quantidade
RAM	512 MB
Armazenamento(SD card)	2GB

Tabela 2 – Armazenamento da Raspberry PI Model B Rev 2

Portas	Quantitade
Porta Ethernet(RJ45)	1
GPIO	8
UART	2
I2C	2
SPI	5
3.3V	1
5V	2
Ground	5

Tabela 3 – Entradas e Saídas da Raspberry PI Model B Rev 2

Figura 3 – Pinos da *Raspberry Pi 1 Model B Rev 2*



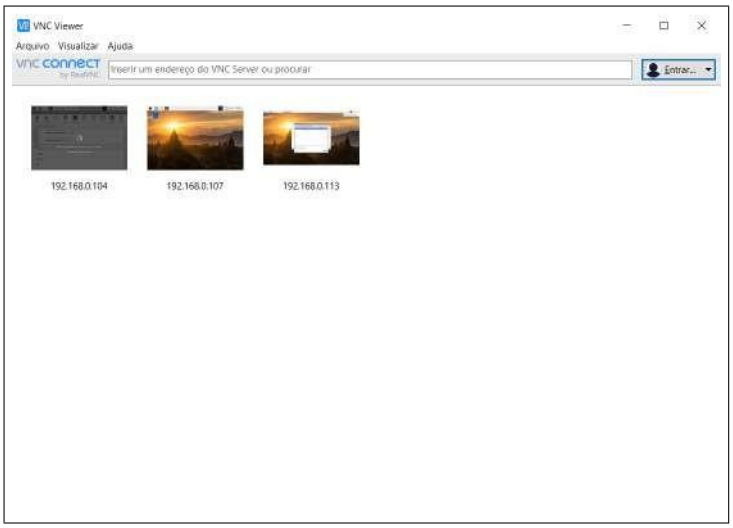
FONTE:(SOUZA, 2020)

2.7 VNC Server e VNC Viewer

O Virtual Network Computing (VNC) é um protocolo de internet que permite a conexão remota entre dispositivos, incluindo a projeção de suas interfaces gráficas. O protocolo foi utilizado para controlar a *Raspberry Pi 1* remotamente, tornando necessária apenas a conexão do módulo *Wifi* para acessar o ambiente do microcomputador.

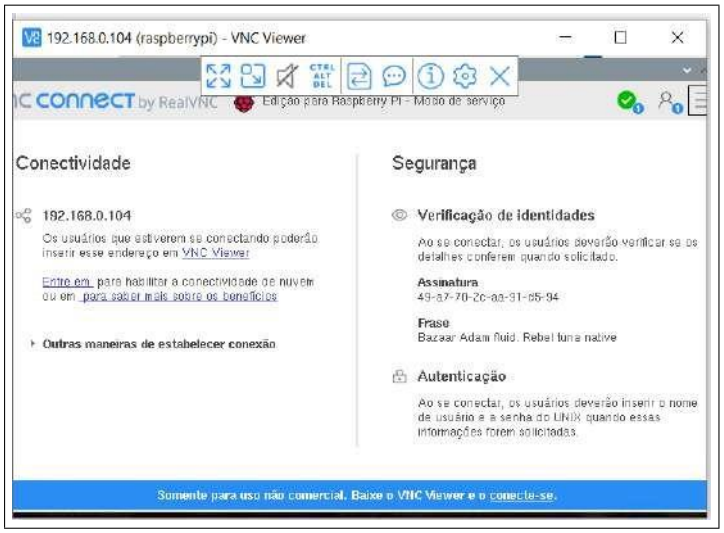
O *VNC Server* e o *VNC Viewer* foram plataformas complementares de acesso remoto utilizadas, por meio de uma conexão *Wifi*, conforme mostrado nas Figuras 4 e 5.

Figura 4 – *VNC Viewer*



FONTE:Autoria Própria

Figura 5 – *VNC Server*



FONTE:Autoria Própria

2.8 Módulo Adaptador de Antena *Wifi* ECOODA 802.IIN

O módulo *Wifi* utilizado foi o Adaptador de Antena ECOODA 802.IIN, conforme mostrado na Figura 6. O módulo conta com uma velocidade de tráfego de até 1800 Mbps e com a interface *USB* 2.0.

2.9 Motor Cc 3 A 6v Com Redução e Roda 68mm

Os motores CC(Corrente Contínua) são dispositivos capazes de converter energia elétrica CC em energia mecânica, Suas principais características são o preço baixo, em comparação a outros tipo de motores, a facilidade de controle de sua velocidade, por meio modulação PWM e sua fonte de alimentação ser de fácil acesso, como no projeto, que

Figura 6 – Módulo Adaptador de Antena *Wifi* ECOODA 802.IIN

FONTE:Autoria Própria

foram utilizadas pilhas recarregáveis.

> "Por meio das diversas combinações de enrolamentos de campo, excitados em derivação, série ou independentemente, elas podem ser projetadas de modo a apresentar uma ampla variedade de características de tensão versus corrente ou de velocidade versus conjugado, para operações dinâmicas e em regime permanente."(UMANS, 2014).

O modelo utilizado no projeto foi o Motor CC 3 A 6v Com Redução e uma Roda de 68 mm, Figura 7, cujas configurações podem ser visualizadas na Figura 8.

Figura 7 – Motor DC e Rodas utilizadas

FONTE:(ELETRO, 2021)

Figura 8 – Especificações do Motor DC e das Rodas

Especificações Redutor:
- Eixo duplo
- Tensão de Operação: 3-6V
- Redução: 1:48
- Peso: 30g
- Corrente sem carga: ≤200mA (6V) e ≤150mA (3V)
- Velocidade sem carga: 200RPM (6V) e 90RPM (3V)

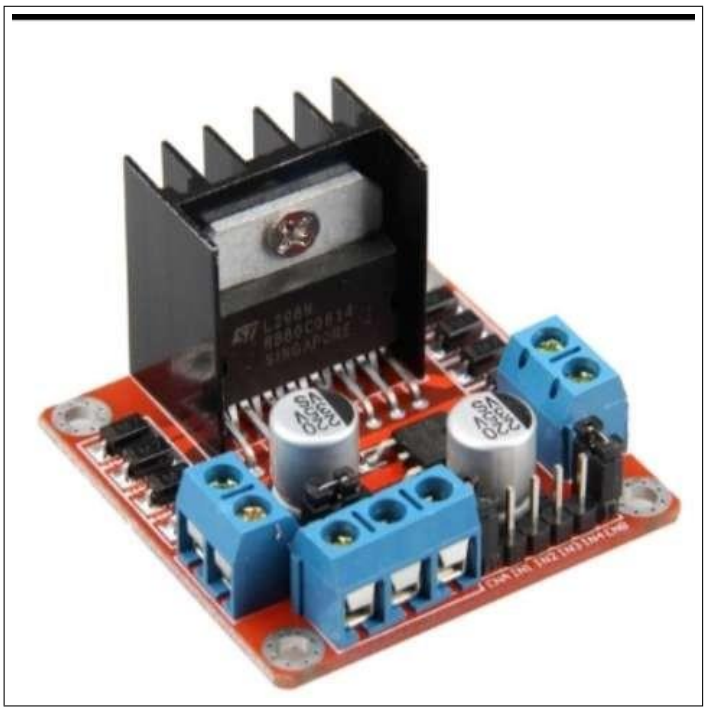
Especificações Roda:
- Diâmetro: 68mm
- Largura: 26mm
- Furo central: 5,3 x 3,66mm (Semicírculo)
- Peso: 50g

FONTE:(ELETRO, 2021)

2.10 *Driver* Motor Ponte H - L298N

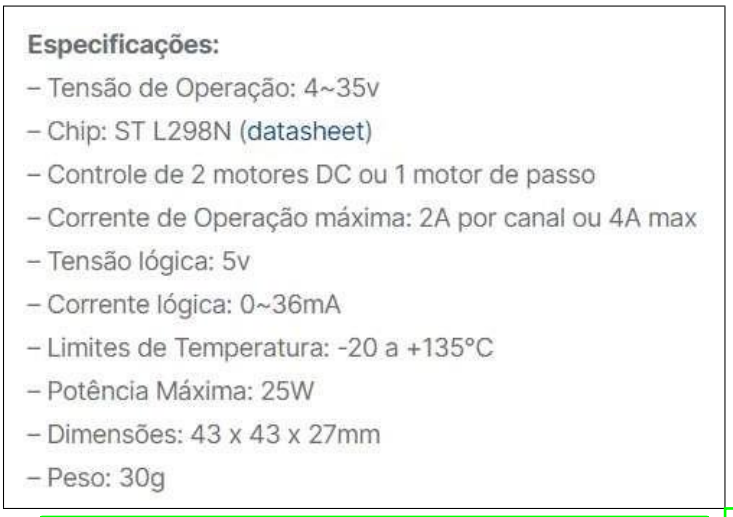
Para o acionamento dos motores, foi utilizada o *Driver* Motor Ponte H - L298N. O *Driver* é baseado na ponte H L298, que é uma ponte H completa, que suporta uma corrente de até 2A por canal, ou 4A no total e pode controlar até 2 motores CC, de maneira completa, ou seja, com 2 sentidos de rotação, e independente.

Figura 9 – *Driver* Motor Ponte H - L298N



FONTE:(FILIPEFLOP COMPONENTES ELETRÔNICOS, 2021a)

Figura 10 – Especificações do *Driver L298N*



FONTE:(FILIPEFLOP COMPONENTES ELETRôNICOS, 2021a

2.11 Modulação PWM

Uma das principais características dos motores CC é a possibilidade de se controlar a velocidade do motor a partir da sua tensão de armadura, ou seja, a tensão que o alimenta. Esse tipo de controle é denominado controle pela tensão de terminal de armadura.

> "O controle da tensão de armadura tira vantagem do fato de que, como a queda de tensão na resistência de armadura é relativamente pequena, então, em regime per- manente, uma variação na tensão de terminal de armadura de um motor em derivação será acompanhada por uma variação substancial na tensão de velocidade."(UMANS, 2014)

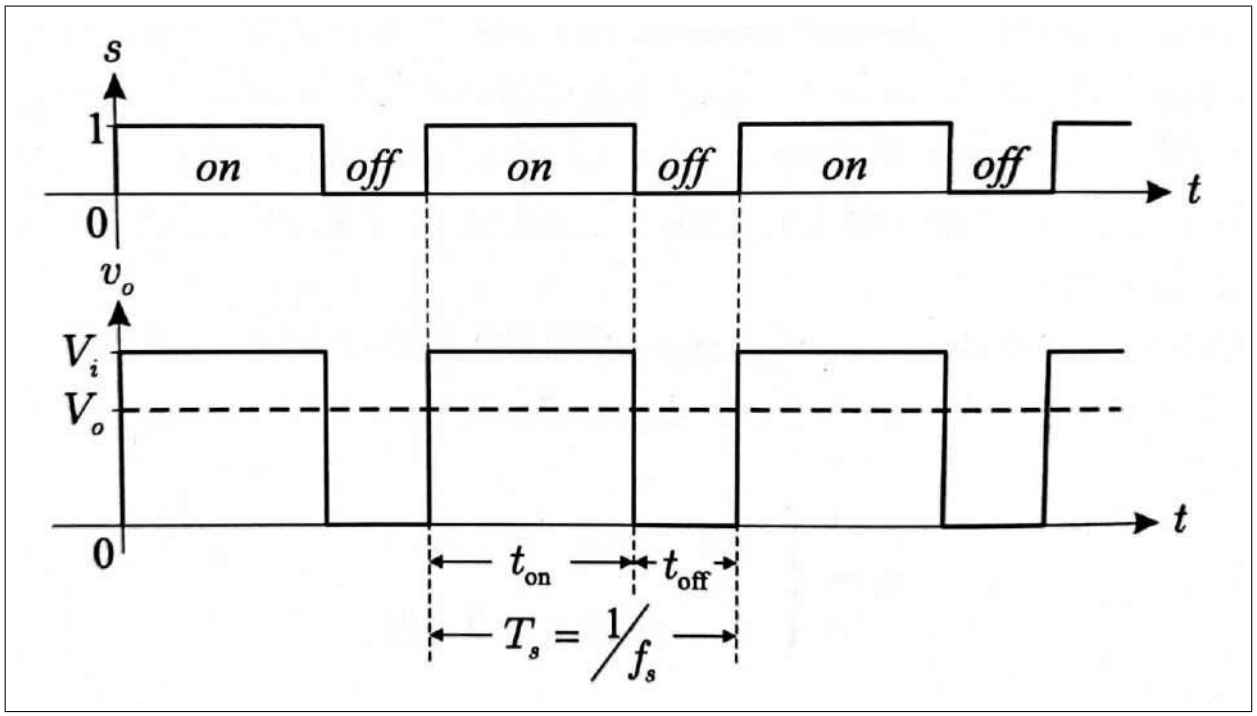
Um dos meios mais utilizados para o controle desta tensão é denominado Modulação Por Largura de Pulso - *Pulse Width Modulation(PWM)* - , que consiste numa técnica de chaveamento, onde a chave "esteja ligada(fechada) por um tempo t_{on} e desligada por um tempo t_{off} , em cada ciclo de um período T_s fixo, a forma de onda resultante da tensão de saída é um trem de pulsos retangulares de duração t_0n", (VITORINO, 2019). Conforme ilustrado na Figura 11

Desta forma, a tensão e a corrente de saída, respectivamente, V_o e I_0, tornam-se:

$$V_o = DV_i \tag{2.10}$$

$$I_o = \frac{I_i}{D} \tag{2.11}$$

Figura 11 – Representação da Modulação *PWM*



FONTE:(VITORINO, 2019)

Onde D é chamado *Duty Cycle*, ou ciclo de trabalho, e representa a relação entre t_on e e o período de chaveamento T_s, ou seja:

$$D = \frac{t_{on}}{T_s} = \frac{t_{on}}{t_{on} + t_{off}} \tag{2.12}$$

2.12 Sensor de velocidade *encoder* LM393

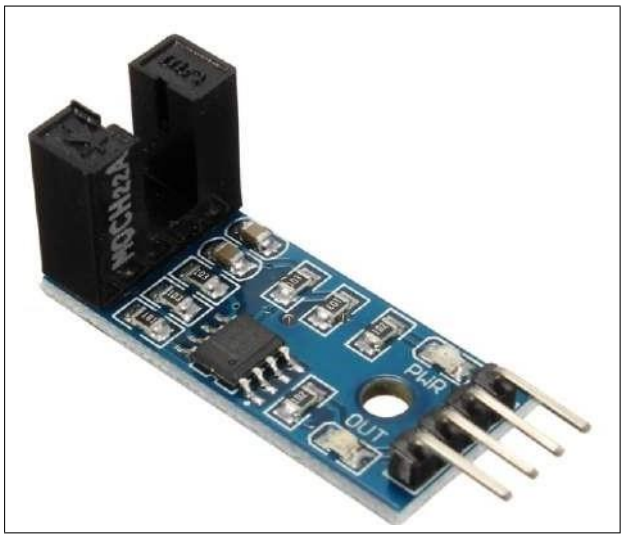
O Sensor de velocidade *encoder* LM393, Figura 12 realiza a medição da velocidade de motores, da contagem de pulsos ou a posição angular de um objeto, por meio de um receptor e um emissor. No emissor é acoplado um *Light Diode Emissor(LED)*, diodo emissor de luz, emissor de infravermelho, cujo sinal é captado quando o mesmo atravessa as ranhuras de um disco *encoder*, acoplado ao eixo do objeto que se deseja sensorear, conforme a Figura 13.

Desta forma, a velocidade em rotações por minuto(RPM), pode ser encontrada por meio da Eq. 2.13:

$$rpm = (60/pulsos\_por\_volta)/(\Delta t(s)) * pulsos \tag{2.13}$$

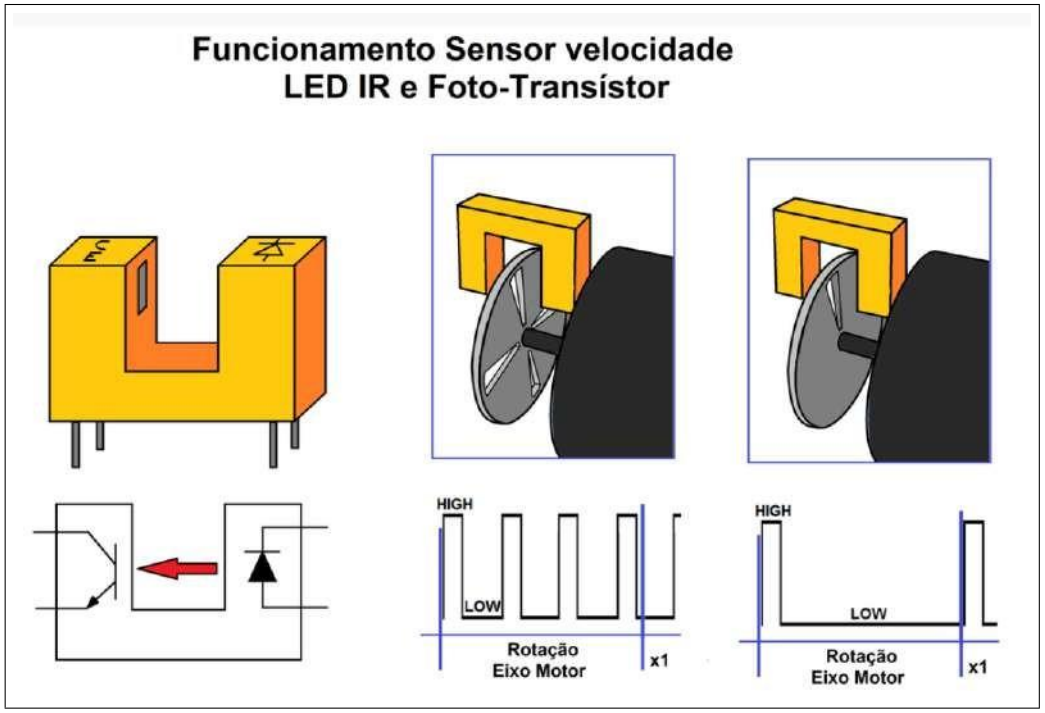
Nota-se que o valor 60 é utilizado para converter a velocidade de rotações por segundo(RPS), para RPM. O *encoder* utilizado, que possuía um total de 20 ranhuras por

Figura 12 – Sensor de velocidade encoder LM393



FONTE:(FILIPEFLOP COMPONENTES ELETRôNICOS, 2021b)

Figura 13 – Esquemático do Sistema de Medição do Sensor LM393



FONTE:(AUTODESK, 2021)

volta, o total de pulsos por volta, também seria de 20, resultando na Eq. 2.14.

$$rpm = (60/20)/(\Delta t(s)) * pulsos = 3/(\Delta t(s)) * pulsos \tag{2.14}$$

3 Metodologia

3.1 Algoritmo A-Star Search

O algoritmo do planejamento de trajetória para o robô diferencial, foi desenvolvimento inicialmente, utilizando como base o Algoritmo *A-Star Search* clássico, (HART; NILSSON; RAPHAEL, 1968). Com intuito de simplificar sua implementação, a movimentação do robô foi proposta a estar limitada a 4 direções, excluindo movimentos diagonais.

O algoritmo foi escolhido pela simplicidade de sua implementação, tendo em vista que não necessita de nenhum recurso mais avançado computacional, como redes neurais, realizando apenas cálculos razoavelmente simples para determinar a trajetória do robô.

3.1.1 Representação dos Mapas e suas células

Sendo o algoritmo *A-Star Search* construído para para buscar o caminho com menor custo num mapa cuja configuração é previamente conhecido, e a priori, sem nenhum tipo de obstáculo dinâmico, foi escolhido o formato de matriz para representar os mapas onde o algoritmo foi testado, armazenadas por meio de arquivos *"comma-separated-values"*, CSV. Para representar os nós, foram escolhidos os algarismos 0 e 1, sendo 0 para os nós livres e 1 para os nós bloqueados, Figura 14.

Os primeiros mapas, foram montados com dimensões inferiores a 10x10, e de maneira a se ter um único caminho possível, afim de verificar a eficiência do algoritmo em encontrar esse caminho, Os demais mapas de testes, foram criados com dimensões que variam entre 11x11 a 61x69, tendo sido criados com uma distribuição pseudoaleatória de 0 e 1, de forma a haver mais 0 que 1, para possibilitar a existência de caminhos e a aplicação do algoritmo.

Para os mapas usados nos testes do Algoritmo sem o protótipo, foram utilizados como partida o nó (1,1), iniciando os índices em 1, e o destino como sendo o nó mais distante da partida, sendo este, o último nós dos mapas, respectivamente os nós (8,8), (11,18), (38,36), (58,56) e (202,208), como exemplificam a Figura 15, a Figura 16 e a Figura 17, sendo a partida o retângulo verde, o destino o retângulo vermelho e os obstáculos os retângulos pretos.

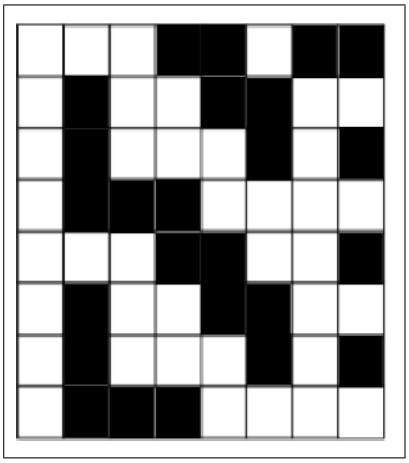
Para representar as células do mapa, cujo algoritmo processou como nós de um grafo, foi criado uma estrutura de dados no formato de classe, denominada por Nó, que foi usada para armazenar as coordenadas, linha e coluna, de cada célula, seu custo calculado e as coordenadas do nó anterior, para que se pudesse reconstruir o caminho planejado pelo algoritmo.

Figura 14 – Exemplo de Arquivo CSV

```
0;0;0;1;1;0;1;1
0;1;0;0;1;1;0;0
0;1;0;0;0;1;0;1
0;1;1;1;0;0;0;0
0;0;0;1;1;0;1;1
0;1;0;0;1;1;0;0
0;1;0;0;0;1;0;1
0;1;1;1;0;0;0;0
```

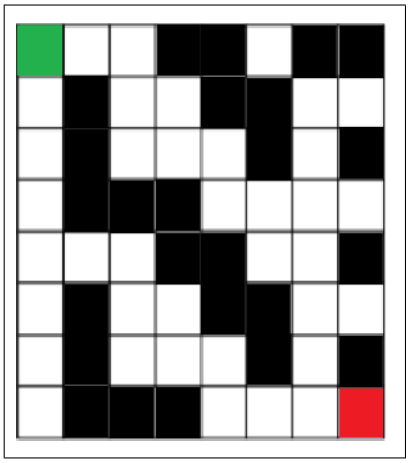
FONTE:Autoria Própria

Figura 15 – Exemplo de representação do mapa



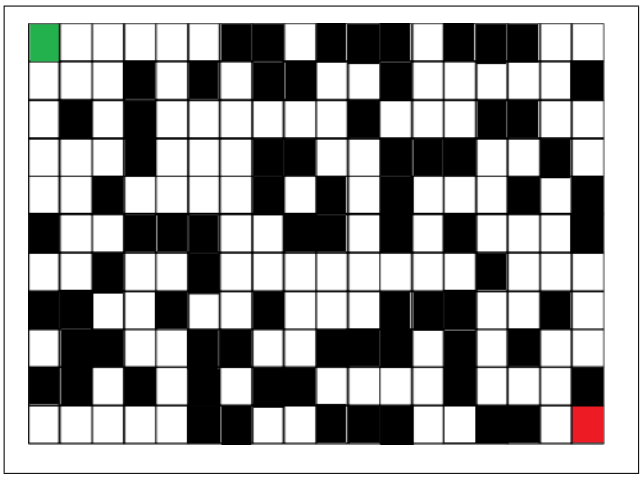
FONTE:Autoria Própria

Figura 16 – Mapa 8x8



FONTE:Autoria Própria

Figura 17 – Mapa 11x18



FONTE:Autoria Própria

3.1.2 Função de Custo Inicial

Tendo sido implementada a classe nó, foi estipulada a função de custo inicial, partindo do pressuposto que o protótipo poderia se movimentar em apenas em 4 direções. Devido a representação matricial do ambiente e a premissa do protótipo estar direcionado inicialmente para o sul, as 4 direções foi definidas como sendo. direção norte, movimento de ré, direção sul, para frente, direção leste, para esquerda, e a direção oeste, para a direita, em relação ao carrinho.

De forma a simplificar o algoritmo, cada direção foi representada por um valor inteiro, sendo 1 para a direção norte, 2 para a direção leste, 3 para a direção sul e 4 para a direção oeste.

A função de Manhattan, Eq. 2.3, foi selecionada como a função heurística inicial, $h(n)$, pois o seu cálculo é simples, necessitando de pouco poder computacional, conforme apresentado no Capítulo 2.

Inicialmente, a função custo, $f\ (n)$, foi estipulada como sendo a soma dos valores da função heurística, $h(n)$, e o custo já atribuído aquele nó, $g(n)$, conforme a Eq. 2.2. Posteriormente, essa função foi alterada com intuito de reduzir o número de expansões necessárias para se calcular o trajeto.

3.1.3 Funções de Validação

Definida a função de custo inicial, foram implementadas as funções de validação, para realizar a verificação do estado dos nós, sendo os estados definidos como bloqueados, caso o valor da célula seja 1, e como livres, caso o valor da célula fosse 0, bem como, se as células informadas como partida e destino, poderiam ser encontradas no mapa, por meio da verificação de suas coordenadas, sendo aceitas como válidas apenas se estivessem dentro dos limites das dimensões do mapa.

3.1.4 Função de busca de Vizinhos

Foi criada também uma função que realiza a busca de vizinhos em cada uma das quatro direções definidas para os movimentos, afim de encontrar as próximas células a serem analisadas, cada uma dessas células passava pelas funções de validação, conforme explicado no tópico anterior.

3.1.5 Funções de visualização da Trajetória

Afim de visualizar o caminho encontrado no mapa, foram criadas 2 funções, uma responsável pela marcação da partida, representada por P, do destino, representado por D e do caminho, representado por ž-ž, e uma função pela exibição do mapa com o trajeto encontrado, como mostrado na Figura 18.

Figura 18 – Exemplo do mapa com a marcação do trajeto

FONTE:Autoria Própria

3.1.6 Função A-Star Search

Tendo sido construídas as funções auxiliares necessárias, foi implementada a função que realiza a busca *A-Star Search*, conforme visto no Capítulo 2. Utilizando a classe nó, foram representadas as células do mapa como sendo nós de um grafo, e feita as validações em cada vizinho existente, sendo definidos os custos de locomoção para cada um deles, por meio da função de custo, e verificado qual nó teria o menor custo.

Finalizada a análise, o nó atual era armazenado numa lista, que continha o trajeto calculado, e numa lista denominada lista fechada, que continha as informações do nós já analisados, para que não preciso refazer os cálculos e as validações, no caso de reabertura de nós, em função de de algum obstrução encontrada no trajeto que está sendo calculado, e o nó de menor custo encontrado, era armazenado em outra lista, denominada lista aberta, para que fosse realizada a sua expansão, até que se identifica-se que o um dos vizinhos do nó atual era o destino desejado, encerrando a busca.

Como cada nó do caminho guarda o seu nó anterior, o caminho pode ser reconstruído do nó de destino, até a partida, apenas realizando o desempacotamento, através dos índices do nó anterior armazenados em cada nó do trajeto.

Também foi colocado um número máximo de iterações, caso o algoritmo não consiga identificar nenhum caminho entre a partida e o destino. Por fim, é realizada a marcação e é mostrada a representação do mapa.

3.1.7 Função Principal

Por fim, na função principal, era carregado o mapa, por meio da biblioteca *Numpy*, a célula de partida e a célula de destino, bem como efetuada as verificações das duas, e chamada a função de busca *A-Star Search*.

Implementado o algoritmo inicial, foram realizados testes nos mapas anteriormente citados, conforme será visto no Capítulo 4, no qual se constataram que, embora o algoritmo conseguisse encontrar caminho com o menor custo, eram realizadas um números excessivo de expansões, que, dependendo do tamanho do mapa, chegavam a ultrapassar 100 mil iterações no mapa de dimensão 58x56, gerando uma latência considerável no código, superior a 922.78654 segundos, conforme pode ser visualizado na Tabela 9, Afim de diminuir esse período de latência, foram implementadas algumas variações do Algoritmo *A-Star Search*, utilizando como base, as funções já implementadas.

3.1.8 Implementação das variações do Algoritmo A-Star Search

Portanto, para fins de diminuir o tempo de busca do algoritmo, foram implementadas algumas de suas versões com ponderamento, como pode ser visto a seguir e demonstrados no Capítulo 2, que resultaram numa redução significativa no número nós expandidos.

3.1.8.1 Algoritmo A-Star Ponderado simples

Após alguns testes no Algoritmo A-Star simples, e baseando-se no algoritmo A-Star Ponderado, (WILT; RUML, 2012), utilizou-se a Eq. 2.5, como a função de custos.

Para definir o valor de w, foram realizados teste nos mapas já citados, nos quais foram testados os valores $w \in [1, 1.5, 2, 4, 6, 8, 10, 20]$ e foi estabelecido o valor $w = 10$, devido a não haver diferenças significativas entre $w = 10$ e $w = 20$, no número de expansões para pequenos mapas, como visto nas Tabelas 4 e na Tabela 5, resultando na Eq. 3.1.

$$f = g(n) + 10 * h(n) \tag{3.1}$$

3.1.8.2 Algoritmo A-Star Search pxWU

Afim de melhorar a performance do algoritmo, foram implementados 2 variações, *pxWU* e pxWD, (CHEN; STURTEVANT, 2021).

Utilizado a Eq. 2.7 com $w = 10$, obteve-se a função na forma da Eq. 3.2:

$$f(x) = \begin{cases} g(n) + h(n), & se\ g(n) < (2 * 10 - 1) * h(n) \\ \frac{(g(n)+(2*10-1)*h(n))}{10}, & C.C \end{cases} \tag{3.2}$$

3.1.8.3 Algoritmo A-Star Search pxWD

De maneira análoga, utilizando a Eq.2.9, com $w = 10$, foi implementada a função custo dessa modalidade, como sendo a Eq. 3.3:

$$f(x) = \begin{cases} g(n) + h(n), & se\ g(n) < h(n) \\ \frac{(g(n)+(2*10-1)*h(n))}{10}, & C.C \end{cases} \tag{3.3}$$

Conforme será visto no Capítulo 4, o Algoritmo *A-Star Search - pxWD*, obteve. em média, os melhores resultados nos teste, pois resultou em um menor número de expansões, conforme mostram as Tabelas 7 a 10, para o encontro do trajeto, e foi selecionado como o algoritmo a ser usado na construção do software do protótipo.

3.2 Escolha do Protótipo

O protótipo foi escolhido de maneira a ser mantido um custo de operação relativamente baixo e acessível, para que o mesmo possa vir a ser replicado em diferentes situações. Portanto, foi escolhido um robô diferencial, cujo chassi pode ser visualizado nas Figuras 19 e 20. O mesmo foi composto por 2 Motores CC Com Redução e Roda de 68mm de diâmetro, cujo acionamento foi realizado pelo *Driver* L298N.

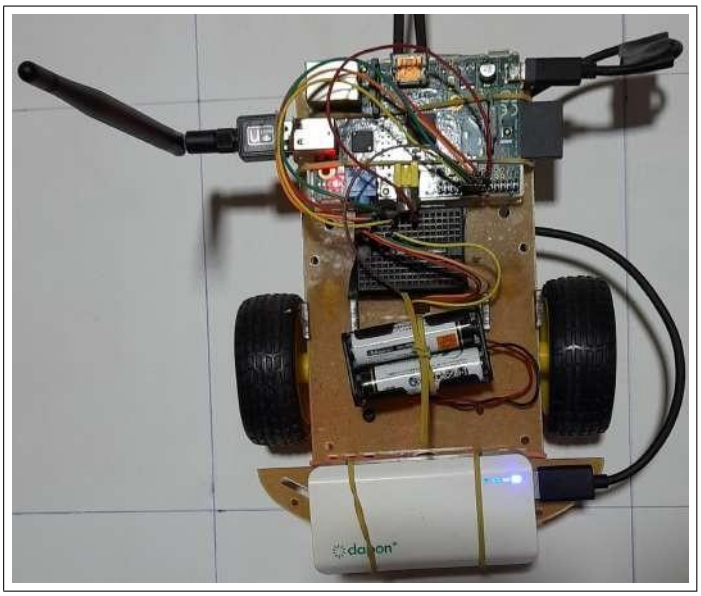
O Algoritmo de busca *A-Star Search* e o de controle dos motores, foram executados por meio de uma *RaspBerry PI 1 Model B Rev 2*, fixada ao chassi e conectada, através de *jumpers*, a uma *protoboard*, de maneira a minimizar o desgaste de suas portas.

A alimentação da *Raspberry PI 1* foi feita por meio de um *Power bank*, Figura 21, modelo danpo Solo2, capaz de fornecer, com uma capacidade de 5200 mAh, que também foi fixado ao chassi do protótipo. Por sua vez, a fonte de alimentação dos Motores CC foram 4 pilhas recarregáveis, Figura 22, acopladas ao chassi por meio de um soquete.

3.2.1 Acesso remoto a Raspberry PI

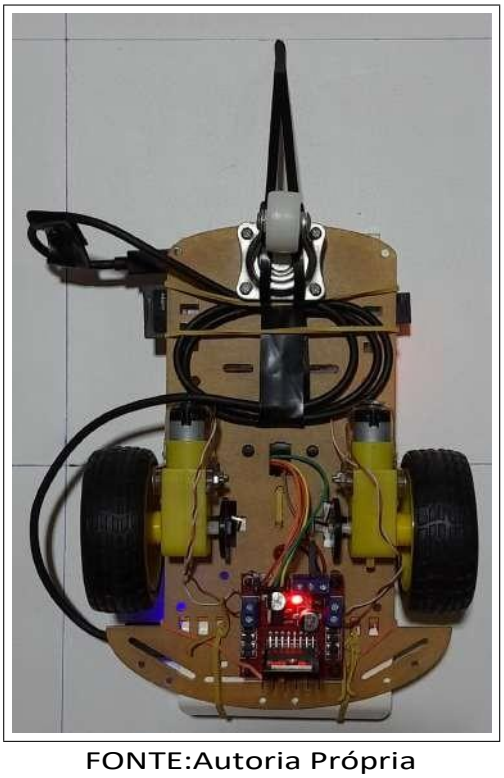
De forma a torna possível o controle da *Raspberry PI 1* de maneira remota, via *WIFI*, foram utilizados os Programas *VNC Viewer*, no computador, e *VNC Server*, na

Figura 19 – Vista Superior do Protótipo



FONTE:Autoria Própria

Figura 20 – Vista Inferior do Protótipo



FONTE:Autoria Própria

Raspberry PI. Os mesmos permitem o acesso a *Raspberry PI*, como mostrado na Figura 23, bem como, a utilização do teclado e do *mouse* do computador na placa. Para a conexão da placa a rede, foi utilizado o Módulo Adaptador de Antena *Wifi* ECOODA 802.IIN, conectado a entrada *USB*.

Figura 21 – *Powerbank* danpo Solo2

FONTE:Autoria Própria

Figura 22 – Pilha Recarregável

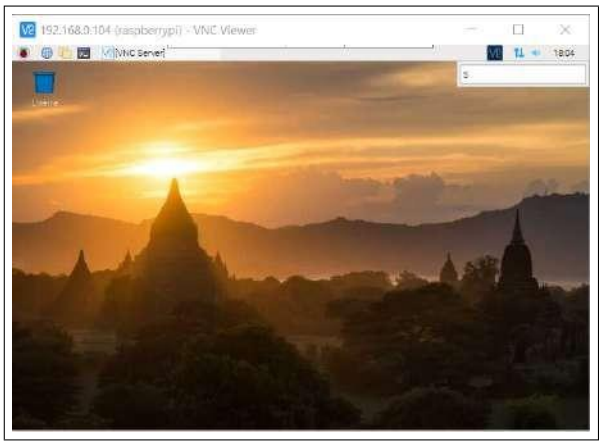
FONTE:Autoria Própria

3.2.1.1 Controle dos Motores Diferenciais

Para os motores, foram estipulados os pinos 16, *GPIO23* e 18, *GPIO24*, ré e frente motor 1, e 13, *GPIO27* e 15, *GPIO22*, ré e frente do motor 2, configurados a partir da biblioteca *RPi.GPIO*, no modo *BOARD*, onde a numeração dos pinos segue a numeração impressa no microcomputador, conforme mostra a Figura 3, através da função *RPI.GPIO.setup()*.

Assim, de acordo com o caminho encontrado, os motores eram inicialmente ativados em sua velocidade máxima, através da função *RPi.GPIO.output*, onde seria ativado o respectivo pino para o acionamento de cada motor, sendo neles colocados 5V, ou o valor lógico de *RPi.GPIO.HIGH*, conforme estabelecido na biblioteca e acionando o *driver*

Figura 23 – *Acessor Remoto a Raspberry PI*



FONTE:Autoria Própria

L298N, que acionava o respectivo motor.

Assumiu-se que a frente do robô seria os seus motores, ficando como traseira a roda-livre, que foi presa, para reduzir o risco que a mesma venha a interferir na direção que o robô deve seguir e foi estabelecido durante os testes, que a posição inicial do robô é sempre voltada para o sul.

Estabelecidos essas imposições, os motores eram mantidos acionados por meio da função *time.sleep()*, da biblioteca *Time*, cujo tempo de espera foi sendo definido, inicialmente, por meio de testes.

No teste prático, foi montado um pequeno mapa, conforme visto na Figura 24. Desta forma, os movimentos foram definidos, de maneira retilínea, seja para o sul ou para a frente, acionando os pinos 18 e 15, ou para norte, movimento de ré, acionando os pinos 16 e 13, por um determinado tempo em milissegundos, que foi o suficiente para o robô se locomover, e para os movimentos laterais, devido a características do motores, foi necessário um tempo diferente, mesmo quando o mapa tinha as mesmas dimensões.

Este tempo foi sendo estipulado conforme os teste práticos foram realizados, e devido as fontes de alimentação utilizadas, que descarregavam durante os teste, o que gerava perda de potência por parte dos motores, tornando necessários tempo maiores ou menores, em função do estado da fonte.

3.2.2 Função de Controle de Velocidade

Na tentativa de corrigir a divergência na velocidade de rotação dos motores, foi implementada uma função que controlaria a velocidade dos mesmos, por meio da Modulação de Largura de Pulso, Modulação *PWM*. Para isso, inicialmente, foram configurados os pinos 13 e 16, frente dos motores 1 e 2, respectivamente, como os pinos do *PWM*.

A medição de ambos os motores era realizada pelos sensores óticos *encoders LM393*,

que detinha entre o receptor e o emissor, um disco acoplado ao eixo de cada motor, que continha pequenos furos, responsáveis pela propagação da luz infravermelha que era captada pelo receptor do sensor. Cada vez que o receptor recebia um pulso, a variável de contagem de pulsos de cada motor era incrementada. Esse pulsos eram detectados pela função *RPi.GPIO.add_event_detect()*, que estava configurada para detectar o *RPi.GPIO.RISING*, que seria a boda de subida do pulso. Essa contagem era realizada a cada 0.5 segundos, e o cálculo do rpm era dado pele Eq. 2.14, com o período de tempo $\Delta t(s)$, de 0.5 segundos

Medidos os valores de rpm para cada motor, era então calculada a diferença de velocidade e essa diferença era aplicada no motor com maior rotação, reduzindo a sua velocidade, afim de se evitar danos forçado uma rotação maior no motor mais lento.

O controle era feito por meio da alteração do *DutyCycle* do motor, com a função *ChangeDutyCycle()*, que recebe o valor desejado *Dutycycle*, cujo padrão foi tomado como sendo 80%, e a diferença de velocidade retirada do *Dutycycle* do motor mais rápido.

Embora esse controle tenha resultado numa aproximação razoável dos rpms dos motores, foi observado que o mesmo interferia no tempo que os motores precisariam ficar ativos, devido ao período entre as medições, resultando num período de ativação maior que o desejado.

Porém, foi notado que essa estratégia de controle, devido ao uso de interrupções, resultava em um atraso considerável ao algoritmo principal de otimização, resultando em acionamentos dos motores além do tempo necessário para as movimentações. Portanto, a fim de reduzir estes atrasos e simplificar a implementação, foi escolhido um método de controle mais simples, acionando os motores por um determinado período de tempo, que foi selecionado através de testes como o protótipo nos mapas montados, e os encoders foram removidos do protótipo.

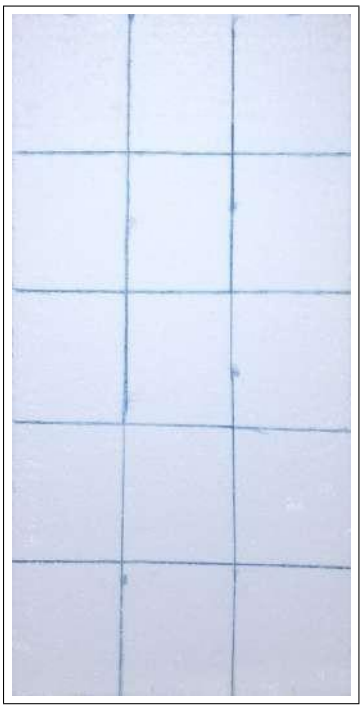
3.3 Montagem dos mapas de testes do protótipo

O mapa de testes foi inicialmente construído pelo autor,com uma folha de isopor, dividindo-a em retângulos com dimensões semelhantes ao protótipo, aproximadamente 20 cm x 16 cm, numa matriz 3x5, com uma área total de 0,48 m^2, conforme mostrado na Figura 24. Embora tenha sido útil nas etapas iniciais, foi notado que o mesmo se desgastou com facilidade, devido ao atritos com as rodas do motor.

O segundo mapa foi montado numa estrutura de madeira, que foi dividida numa matriz 5x5, cujos quadrados tinham dimensões 20 cm x 20 cm, totalizando uma área de 1 m^2, como mostrado na Figura 25. Como a superfície é mais lisa, foi notado uma melhor adequação do protótipo.

Cada retângulo nos mapas montados, foi utilizado para representar um nó do grafo,

Figura 24 – Mapa montado em Isopor



FONTE:Autoria Própria

Figura 25 – Mapa montado em Madeira

FONTE:Autoria Própria

ou uma célula do arquivos *csv*, de forma que o trajeto escolhido pudesse ser visualizado nos mesmos. Ambos os mapas foram testados com configurações diferentes, como representado nas Figura 27, 28, 29, 30 . Na configuração da 27, foram colocados 5 obstáculos no mapa, contendo 2 trajetos viáveis. Os retângulos pretos representam os obstáculos colocados, o retângulo verde representa o ponto de partida e o retângulo vermelho representa o destino

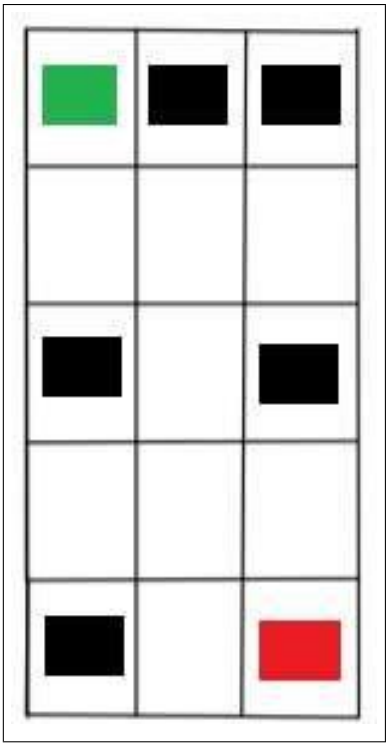
Figura 26 – Mapa montado em madeira com protótipo

FONTE:Autoria Própria

selecionado.

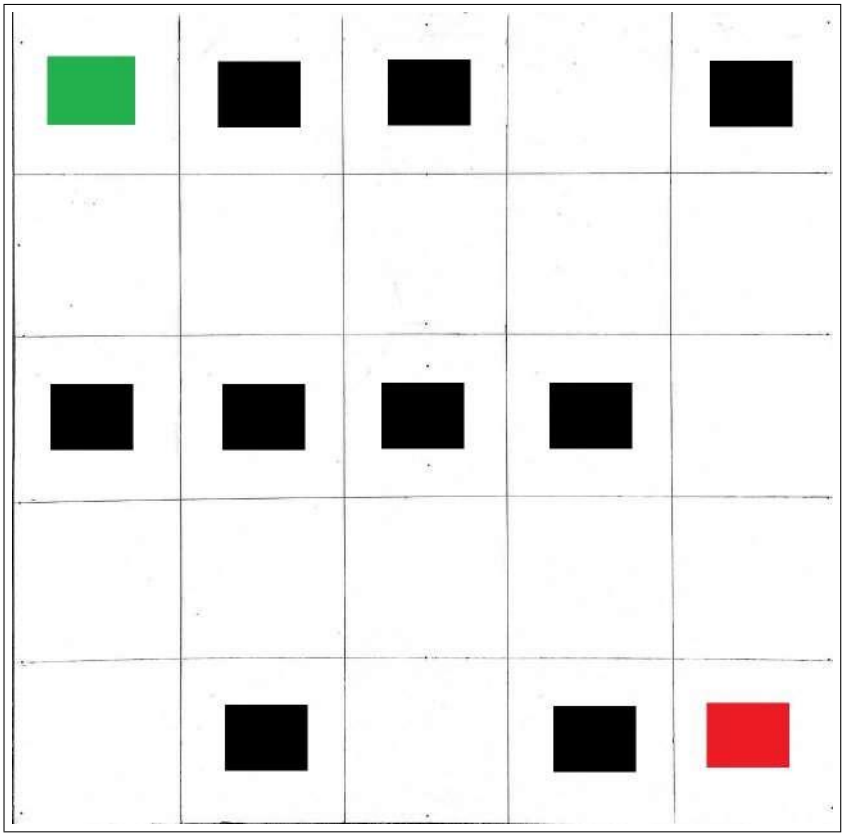
Para fins de resultados mais precisos, os testes finais foram realizados por meio dos mapas do tabuleiro de madeira, em 5 configurações diferentes, mostradas nas Figuras 28 a 30. O mapa 1, Figura 28, foi composto de 9 obstáculos. e só continha 1 trajeto viável. O mapa 2, Figura 29, continha 7 obstáculos, e apenas 1 trajeto viável. O mapa 3,Figura 30 por sua vez, continha 8 obstáculos, e 2 trajetos viáveis.

Figura 27 – Representação do Mapa 3x3 em Isopor



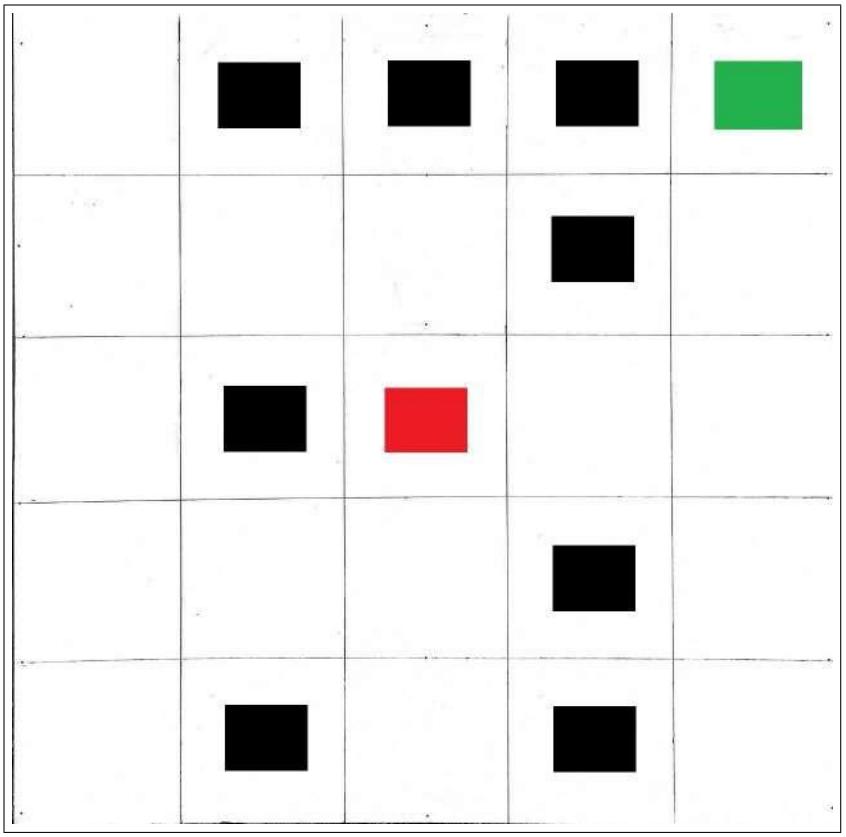
FONTE:Autoria Própria

Figura 28 – Representação do Mapa 5x5 na Configuração 1



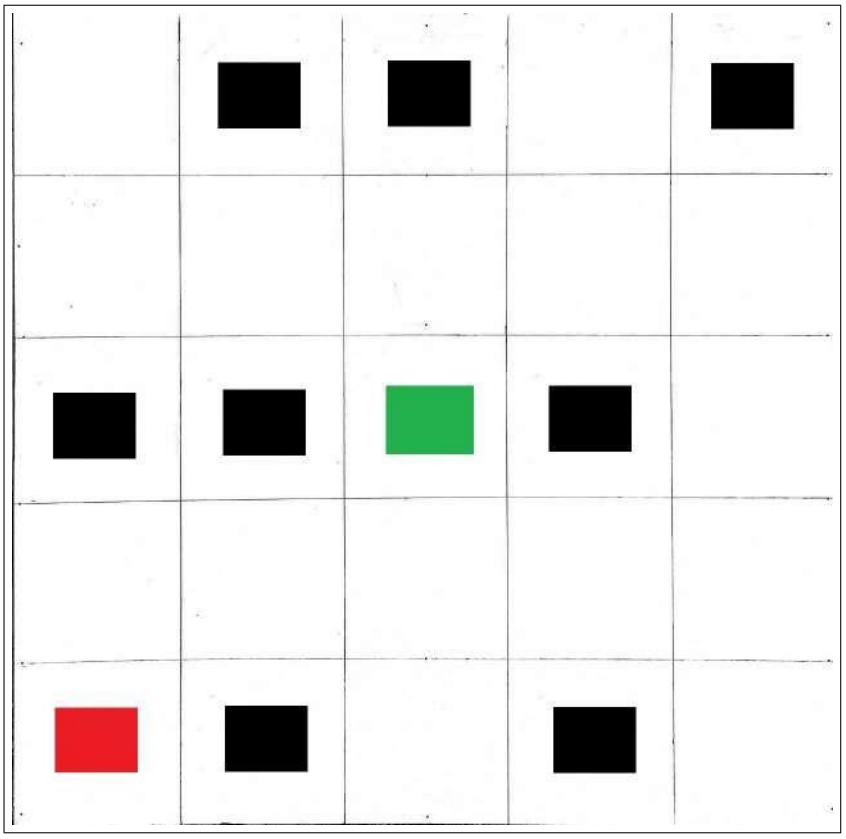
FONTE:Autoria Própria

Figura 29 – Representação do Mapa 5x5 na Configuração 2



FONTE:Autoria Própria

Figura 30 – Representação do Mapa 5x5 na Configuração 3



FONTE:Autoria Própria

4 RESULTADOS

4.1 Algoritmos de Teste

Conforme apresentado no Capítulo 3, a fase inicial dos testes foram feitas por meio dos mapas criados em isopor e madeira, cujos resultados obtidos serão apresentados a seguir.

4.1.1 Seleção dos Pesos

Para a escolha dos pesos, foram utilizados o Mapa da Figura 16 e o Mapa da Figura 17. Nestas configurações, foram avaliados os valores de $w = [1, 1.5, 2, 4, 6, 8, 10, 20]$, e, embora o valor de $w = 20$, tenha apresentado o melhor resultado, com um menor no número de expansões, ao custo de nenhum ou pouco aumento no número de nós percorridos durante o trajeto calculado, como mostram a Tabela 4 e a Tabela 5.Para os testes com o protótipo, o valor de $w = 10$ apresentou um resultado satisfatório, pois para pequenos mapas, como o de testes, os valores de $w = 10$ e $w = 20$, não apresentam uma diferença significativa no número de expansões.

Peso(w)	Expansões	Passos	Tempo De Execução(s)
1	69	15	0.00100
1.5	68	15	0.00100
2	19	15	0.00000
4	18	15	0.00000
6	18	15	0.00000
8	18	15	0.00000
10	18	15	0.00000
15	18	15	0.00000
20	18	15	0,00000

Tabela 4 – Resultados obtidos com o *A-Star* Ponderado no mapa 8x8

4.1.2 Mapa 8x8

De acordo com os dados apresentados na Tabela 6, nesta configuração, o Algoritmo *A-Star Search Ponderado* apresentou um menor número de expansões, seguido do *A-Star Search Ponderado pxWD*. Observa-se, no entanto, que os 4 tipos de algoritmos implementados conseguiram encontrar o trajeto com menor custo, como mostra a Figura 31.

A Figura 31 mostra que o número de nós expandidos e re-expandidos, retângulos cinzas, diminuiu de modo considerável, entre os Algoritmo *A-Star Básico*, Figura 31a,

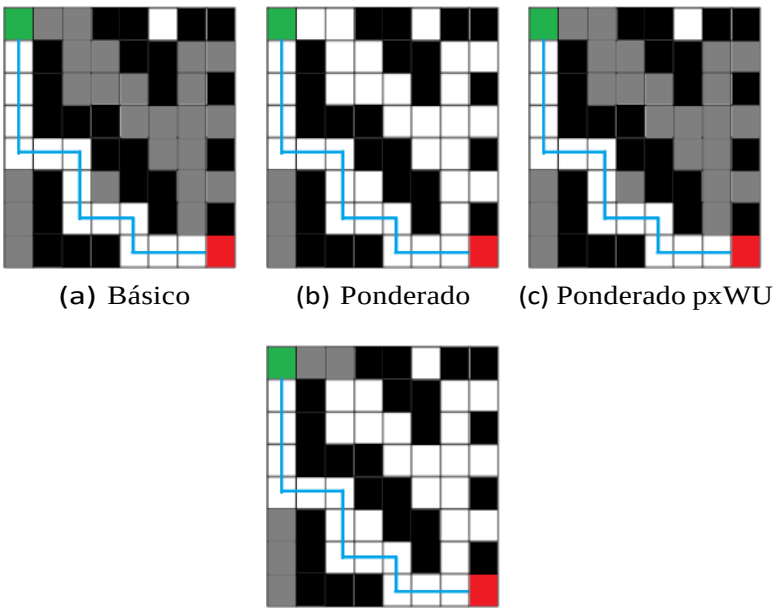
Peso(w)	Expansões	Passos	Tempo De Execução(s)
1	696	34	0.03167
1.5	694	34	0.02928
2	97	34	0.00375
4	76	36	0.00000
6	68	36	0.00000
8	62	36	0.00000
10	59	36	0.00000
15	57	42	0.00100
20	56	42	0.00000

Tabela 5 – Resultados obtidos com o *A-Star* Ponderado no mapa 11x18

com a as variações implementadas do algoritmo, Ponderado, Figura 31b, Figura *pxWU*, Figura 31c e Figura *pxWD*, Figura 31d, respectivamente, embora para esta configuração, mapa 8x8, não tenha havido mudança na trajetória encontrada, mesmo havendo mais de 1 caminho possível.

Algoritmo	Expansões	Tempo De Execução(s)
Clássico	69	0.00199
Ponderado Simples	18	0.00000
pxWU	41	0.00100
pxWD	20	0.00100

Tabela 6 – Resultados com obtidos com o mapa 8x8



(a) Básico (b) Ponderado (c) Ponderado pxWU

(d) Ponderado pxWD

Figura 31 – Comparação entre os tipos de Algoritmos *A-Star Search* implementados

4.1.3 Resultados Obtidos com os outros Mapas

Para o mapa 11x18, foi notado que a variação *pxWD*, apresentou um número menor de expansões, comportamento que será propagado, inclusive com o aumento na diferença do número de expansões, nos mapa 39x36, 58x56, de tal modo, de tal modo, que o mesmo foi escolhido para dar prosseguimento aos testes com o protótipo. O mapa 202x208, por sua vez, excedeu o limite máximo de expansões permitidas em todas as variações do algoritmo, apresentando um tempo de execução considerável.

Algoritmo	Expansões	Tempo De Execução(s)
Clássico	696	0.03324
Ponderado Simples	73	0.00000
pxWU	107	0.00000
pxWD	64	0.00000

Tabela 7 – Resultados com obtidos com o mapa 11x18

Algoritmo	Expansões	Tempo De Execução(s)
Clássico	>100000	>1073.09448
Ponderado Simples	8340	3.36325
pxWU	24330	31.87573
pxWD	191	0.01231

Tabela 8 – Resultados com obtidos com o mapa 39x36

Algoritmo	Expansões	Tempo De Execução(s)
Clássico	>100000	>922.78654
Ponderado Simples	909	0.04728
pxWU	>100000	>1039.61143
pxWD	271	0.00802

Tabela 9 – Resultados com obtidos com o mapa 58x56

Algoritmo	Expansões	Tempo De Execução(s)
Clássico	100000	> 516.93637
Ponderado Simples	>100000	>763.67453
pxWU	>100000	> 1066.28336
pxWD	>100000	>998.74843

Tabela 10 – Resultados com obtidos com o mapa 202x208

Levando em consideração estes resultados dos testes, o Algoritmo escolhido para o protótipo foi o pxWD, que apresentou um número de expansões de nós consideravelmente baixo, em comparação com os outros métodos.

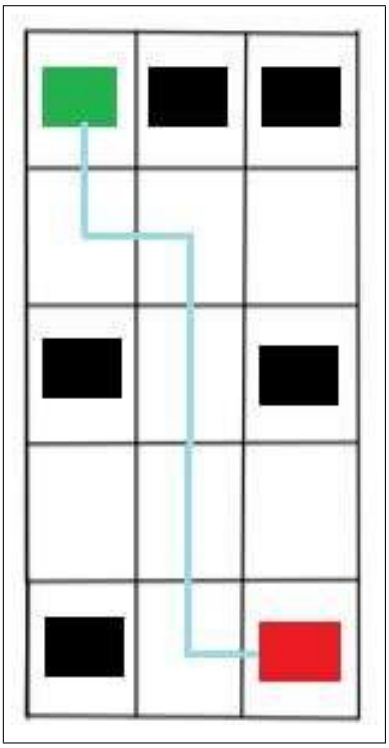
4.2 Resultados obtidos com o protótipo

Conforme da demonstrado no Capítulo 3, foram montadas configurações distintas de mapas para os testes, cujos resultados obtidos podem ser observados nas Tabelas 11 a 14.

4.2.1 Mapa 3x5

Com o Mapa 3x5 em isopor, Figura 27 , durante a etapa inicial do projeto, foi encontrado o trajeto, que percorreu 7 células Figura 32, num tempo de 19 segundos, como mostrado na Tabela 11.

Figura 32 – Representação do Trajeto encontrado para o Mapa 3x5



FONTE:Autoria Própria

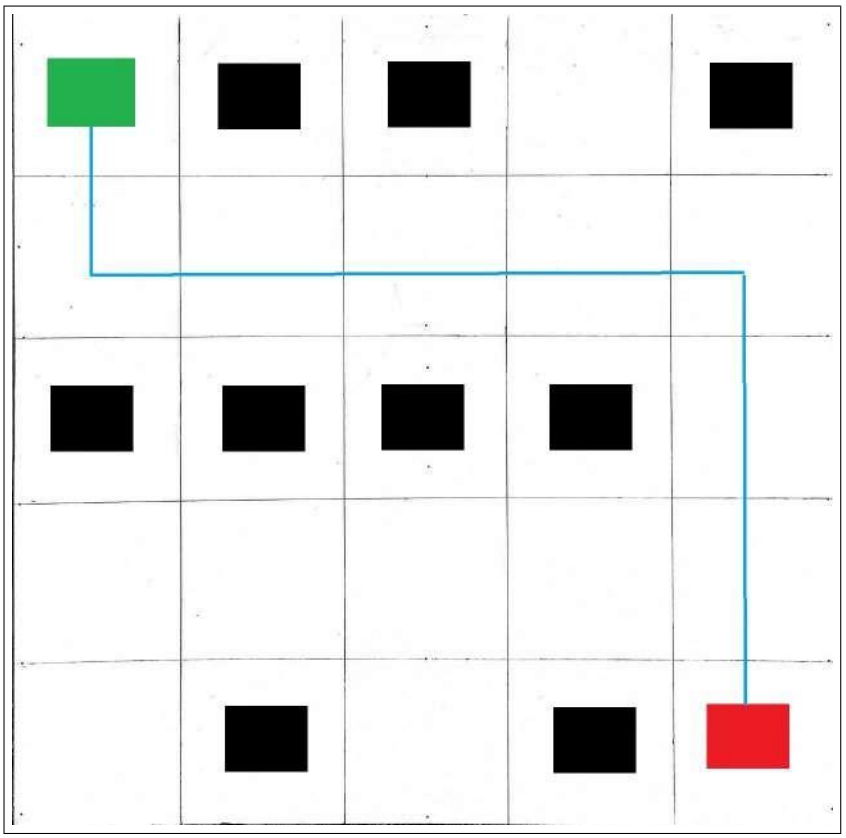
Teste	Tempo De percurso(s)
1	19

Tabela 11 – Resultado obtido com o Mapa 3x5

4.2.2 Mapa 5x5 na Configuração 1

Como mostrado na Figura 33, o trajeto encontrado para a configuração 1, linha em azul, percorria um total de 9 células, incluindo partida e destino, e o protótipo o percorreu com tempo médio de 12,2 segundos, Tabela 12.

Figura 33 – Representação do Trajeto encontrado para o Mapa 5x5 na Configuração 1



FONTE:Autoria Própria

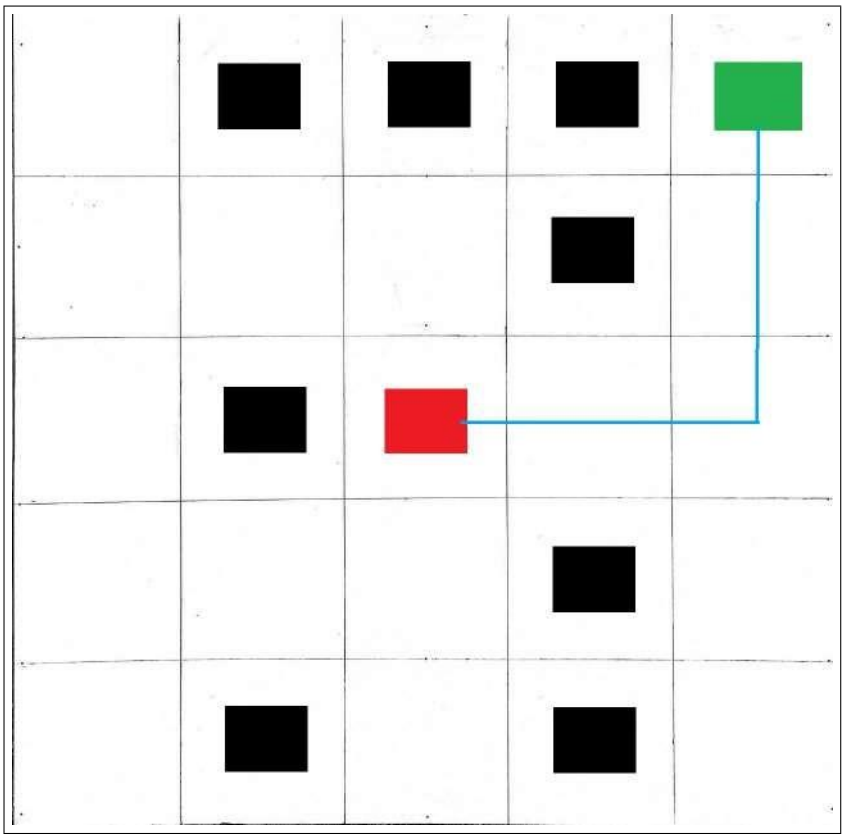
Teste	Tempo De percurso(s)
1	13
2	13
3	12
4	12
5	12
6	12
7	12
8	12
9	12
10	12
Média	12,2

Tabela 12 – Resultados obtidos com o Mapa 5x5 na configuração 1

4.2.3 Mapa 5x5 na Configuração 2

A configuração 2 continha menos obstáculos que o mapa 1 e um trajeto mais curto. O tempo médio do percurso do protótipo foi 6.4 segundos, com um total de 5 células percorridas, como mostram a Figura 35 e a Tabela 13.

Figura 34 – Representação do Trajeto encontrado para o Mapa 5x5 na Configuração 2



FONTE:Autoria Própria

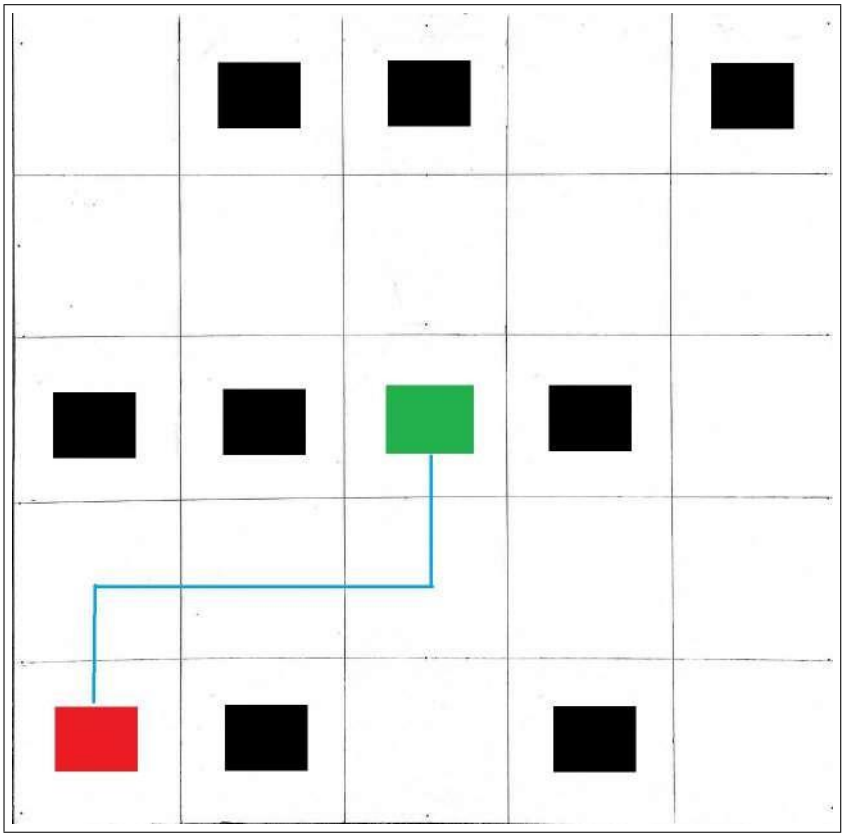
Teste	Tempo De percurso(s)
1	7
2	7
3	6
4	6
5	7
6	6
7	6
8	6
9	6
10	6
Média	6,4

Tabela 13 – Resultados obtidos com o Mapa 5x5 na configuração 2

4.2.4 Mapa 5x5 na Configuração 3

Para a configuração 3, observa-se que haviam 2 trajetos possíveis, tendo o Algoritmo selecionado o percurso com menor custo, Figura 35, conforme foi proposto, atravessando um total de 5 células em um tempo médio de 8,1 segundos. Tabela 14.

Figura 35 – Representação do Trajeto encontrado para o Mapa 5x5 na Configuração 3



FONTE:Autoria Própria

TESTE	TEMPO DE PERCURSO(S)
1	8
2	8
3	8
4	8
5	8
6	9
7	8
8	8
9	8
10	8
Média	8,1

Tabela 14 – Resultados obtidos com o Mapa 5x5 na configuração 3

5 Considerações Finais

A proposta do projeto foi criar um robô, que a partir dos dados dos mapas, informados por meio de um arquivo *csv*, pudesse realizar o planejamento de sua trajetória, levando em consideração o caminho com menor custo, com um baixo custo monetário e computacional.

O Algoritmo construído para protótipo conseguiu então, alcançar tais objetivos de maneira bastante satisfatória, sendo eficiente em buscar o trajeto de menor custo nos mapas utilizados, mesmo quando haviam mais de um trajeto possível, bem como realizar o acionamento e controle dos motores, para a movimentação do protótipo nas direções estabelecidas, mesmo quando foi necessário realizar curvas. Vale salientar, que o algoritmo utilizou abordagens recentes para a efetuação dos cálculos, como os Algoritmos A-Star pxWU e PxWD, (CHEN; STURTEVANT, 2021), cuja proposta foi publicadas no ano de 2021.

Em comparação aos robôs existentes desta categoria, o protótipo apresenta a vantagem de não precisar de nenhum tipo de equipamento instalado no ambiente, como os robôs seguidores de faixa existentes, como exemplos, os desenvolvidos nos projetos (LOPES, 2017) e (AMORIM, 2011), que precisam que seu trajeto seja demarcado fisicamente por meio de faixas no piso.

Para o uso em mapas maiores, seria necessário o auxílio de técnicas que aprimorassem a análise do mapa, como a como a *Hierarchical A-Star*, (WANG et al., 2014), que subdivide o mapa em mapas menores, realizando a análise individual dos mesmos e traçando o trajeto para o mapa original.

A implementação de uma função de controle de velocidade, semelhante a que foi tentada no projeto, também poderia auxiliar na correção dos erros associados a diferença de velocidade dos motores, de maneira a mantê-las próximas e minimizar os erros de divergência de posições.

Um chassi mais robusto poderia influenciar positivamente o desempenho do protótipo, principalmente em decorrência de possíveis desvios de trajetórias resultantes das rodas livres e da precisão dos motores, devidos a erros de construção ou devidos a desgastes entres os mesmos, que resultam em pequenas divergências de velocidades que vão se acumulando, bem como, de uma fonte de alimentação mais robusta, que suporte por um tempo maior a potência dos motores.

Outro fator, seria levar em consideração a inserção de obstáculos dinâmicos, cujo processamento poderia ser feito através de sensores acoplados ao chassi do protótipo ou de Algoritmos de processamento de imagem, de forma a, que quando fosse captado um novo

obstáculo, ou a movimentação de algum existente, o mesmo fosse inserido ou atualizado no mapa, e um novo trajeto fosse planejado.

REFERÊNCIAS

AMORIM, A. F. *ROBÔ SEGUIDOR DE LINHA AUTÔNOMO UTILIZANDO O CONTROLADOR PROPORCIONAL-DERIVATIVO EM UMA PLATAFORMA DE HARDWARE/SOFTWARE LIVRE*. Monografia (TCC) Ð UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA - UESB, VITÓRIA DA CONQUISTA, Brasil, 2011.

AUTODESK. *Arduino Tutorial - Blynk Motor Speed Control ESP8266*. 2021. Disponível em: <https://www.instructables.com/Arduino-Tutorial-Blynk-Motor-Speed-Control-ESP8266/>.

BAčíK, J. et al. PhollowerÐthe universal autonomous mobile robot for industry and civil environments with covid-19 germicide addon meeting safety requirements. *Applied Sciences*, v. 10, n. 21, 2020. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/10/21/7682>.

BORGES, L. E. *Python para desenvolvedores: aborda Python 3.3*. [S.l.]: Novatec Editora, 2014.

CARDONA, M. et al. Mobile robots application against covid-19 pandemic. In: *2020 IEEE ANDESCON*. [S.l.: s.n.], 2020. p. 1ś5.

CHEN, J.; STURTEVANT, N. R. Necessary and sufficient conditions for avoiding reopenings in best first suboptimal search with general bounding functions. *Proceedings of the AAAI Conference on Artificial Intelligence, 35(5), 3688-3696*, 2021. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/16485>.

DAS, S. Design and methodology of automated guided vehicle. *IOSR journal of mechanical and civil engineering*, 04 2016.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik. 1: 269–271*, 1959. Disponível em: <http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf>.

ELETRO, A. *MOTOR DC 3-6V COM CAIXA DE REDUÇÃO E EIXO DUPLO + RODA 68MM*. 2021. Disponível em: <https://www.arduoeletro.com/motor-dc-3-6v-com-caixa-de-reducao-e-eixo-duplo-roda-68mm>.

FILIPEFLOP COMPONENTES ELETRôNICOS. *Driver Motor Ponte H L298n*. 2021. Disponível em: <https://www.filipeflop.com/produto/driver-motor-ponte-h-l298n/>.

FILIPEFLOP COMPONENTES ELETRôNICOS. *Sensor de Velocidade Encoder LM393*. 2021. Disponível em: <https://www.filipeflop.com/produto/sensor-de-velocidade-encoder/>.

FOUNDATION, R. P. *About us*. 2021. Disponível em: <https://www.raspberrypi.org/about/>.

HARRINGTON, W. *Learning Raspbian*. Birmingham, UK: Packet Publishing, 2015.

HARRIS, C. et al. Array programming with numpy. *Nature 585*, v. 8, p. 357ś362, 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *Fifth Annual Symposium on Combinatorial Search*, 1968. Disponível em: <https://ieeexplore.ieee.org/document/4082128>.

HE, X.; WANG, Y.; CAO, Y. Researching on ai path-finding algorithm in the game development. In: . [S.l.: s.n.], 2012. p. 484ś486. ISBN 978-1-4673-2465-6.

JúNIOR, J. P. e Thais Castro e A. A robótica móvel como instrumento de apoio à aprendizagem de computação. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)*, v. 1, n. 1, 2006. Disponível em: <http://br-ie.org/pub/index.php/sbie/article/view/510> .

LIKHACHEV, M.; GORDON, G.; THRUN, S. Ara*: Anytime a* with provable bounds on sub-optimality. In: . [s.n.], 2003. v. 16. Disponível em:<https://papers.nips.cc/paper/2003/file/ee8fe9093fbbb687bef15a38facc44d2-Paper.pdf> .

LOPES, W. A. *PROJETO E IMPLEMENTAÇÃO DE ROBÔ AUTÔNOMO SEGUIDOR DE LINHA*. Monografia (TCC) Ð Universidade Tecnológica Federal do Paraná - UTFPR, PATO BRANCO, Brasil, 2017.

MüLLER, L. *Mais de 100 mil robôs autônomos em atividade nos galpões da empresa*. 2014. Disponível em: <https://www.tecmundo.com.br/mercado/131327-amazon-tem-100-mil-robos-autonomos-galpoes-video.htm> .

PEDAN, M.; GREGOR, M.; PLINTA, D. Implementation of automated guided vehicle system in healthcare facility. *Procedia Engineering*, v. 192, p. 665ś670, 2017. ISSN 1877-7058. 12th international scientific conference of young scientists on sustainable, modern and safe transport. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877705817326619> .

POHL, I. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, v. 1, n. 3, p. 193ś204, 1970. ISSN 0004-3702. Disponível em:<https://www.sciencedirect.com/science/article/pii/000437027090007X> .

SOUZA, F. Como usar a gpio da raspberry pi usando python. *Medium*, 2020. Disponível em: <https://medium.com/vacatronics/este-C3A9-um-tutorial-sobre-como-usar-a-gpio-da-raspberry-pi-usando-a-biblioteca-python-e3b5bd5c890c> .

SPARKFUN. *Raspberry Pi - Model B*. 2014. Disponível em: <https://www.sparkfun.com/products/retired/11546>.

UMANS, S. D. *Máquinas elétricas de Fitzgerald e Kingsley*. 7ª ed. ed. Porto Alegre: AMGH, 2014.

VITORINO, M. A. *Eletrônica de Potência - Fundamentos, Conceitos e Aplicações*. Curitiba, PA, BR: Appris, 2019.

WANG, H. et al. Has: Hierarchical a-star algorithm for big map navigation in special areas. In: *2014 5th International Conference on Digital Home*. [S.l.: s.n.], 2014. p. 222ś225.

WIJAYA, R.; ARTHAYA, B.; SADIYOKO, A. Design of sensory and movement system hardware for maze mapping autonomous mobile robot (amr). In: . [S.l.: s.n.], 2006.

WILT, C.; RUML, W. When does weighted a* fail? *IEEE TRANSACTIONS OF SYSTEMS SCIENCE AND CYBERNETICS*, 2012. Disponível em: <https://www.aaai.org/ocs/index.php/SOCS/SOCS12/paper/viewFile/5413/5187> .

YAO, J.; BINBIN, Z.; QINGDA, Z. The optimization of a* algorithm in the practical path finding application. 05 2009. Disponível em:<https://ieeexplore.ieee.org/abstract/document/5319519> .

ZHANG, J. et al. Autonomous land vehicle path planning algorithm based on improved heuristic function of a-star. *International Journal of Advanced Robotic Systems*, v. 18, n. 5, 2021. Disponível em: <https://doi.org/10.1177/17298814211042730>.

ZHANG, Z.; ZHAO, Z. A multiple mobile robots path planning algorithm based on a-star and dijkstra algorithm. *International Journal of Smart Home*, v. 8, p. 75ś86, 05 2014.

APÊNDICES

APÊNDICE A – Algoritmo Desenvolvido

```
"""
Created on Thu Jul 29 14:24:20 2021

@author: Matheus Lucas de Lucena Pereira
"""
import numpy as np
import RPi.GPIO as GPIO
import time

# Peso
w=10

# vetor de posições para vizinhos (norte, leste, sul, oeste)
delta = [(-1,0),(0,-1),(1,0),(0,1)]

# cria objeto nó
class No(object):

    def __init__(self, i=None, j=None, anterior=None, f=None):

        # índices do nó
        self.i = i
        self.j = j

        # nó anterior
        self.anterior = anterior

        # custo
        self.f = f



# --Declaração do GPIO
GPIO.setmode(GPIO.BOARD)

#FRENTE POWERBANK
```

```
37 #Motores
38 Motor1A = 16 #motor 1 RÉ
39 Motor1B = 18 #motor 1 FRENTE
40 Motor2A = 13 #motor 2 RÉ
41 Motor2B = 15 #motor 2 FRENTE
42
43 GPIO.setup(Motor1A,GPIO.OUT)
44 GPIO.setup(Motor1B,GPIO.OUT)
45 GPIO.setup(Motor2A,GPIO.OUT)
46 GPIO.setup(Motor2B,GPIO.OUT)
47
48 # -- Fim da declaração dos GPIOs
49
50 # Função de parada do motor
51
52 #Função de parada dos motores
53 def stop():
54     GPIO.output(Motor1A,GPIO.LOW)
55     GPIO.output(Motor1B,GPIO.LOW)
56     GPIO.output(Motor2A,GPIO.LOW)
57     GPIO.output(Motor2B,GPIO.LOW)
58     time.sleep(0.5)
59
60     return
61
62 # -- Funções para A-Star Search
63
64 # Função heurística
65 def Heuristica(atual, destino):
66     h = 0.0
67     h    =    abs(atual.i-destino.i)+abs(destino.j-atual.j)
68     return h
69
70 # Verifica se as coordenadas estão dentro do mapa
71 def VerificaLimites(mapa,row, col,):
72
73     return row>=0 and row<len(mapa) and col>=0 and col<len(mapa[0])
74
75
```

```
76  # Verifica se a célula está vazia(==0)
77  def CelulaVazia(mapa, row, col):
78
79      # blocked == 1
80      # Unblocked == 0
81      return mapa[row][col] == 0
82
83  # Busca Vizinhos
84  def BuscarVizinhos(no, mapa):
85      vizinhos = list()
86      for posicao in delta:
87          novo_i= no.i+posicao[0]
88          novo_j=no.j+posicao[1]
89          if(VerificaLimites(mapa, novo_i, novo_j)
90              and CelulaVazia(mapa, novo_i, novo_j)):
91              vizinhos.append(No(novo_i,novo_j))
92
93      if vizinhos: return vizinhos
94      else:
95          print("Não há vizinhos")
96          return
97
98  # Verifica se o proximo vizinho é o destino
99  def EncontraDestino(dest, row, col):
100
101     return row == dest.i and col == dest.j
102
103 # Marca o caminho(Caminho: 4, Partida: 2, Destino: 3)
104 def MarcarCaminho( mapa, caminho, inicio, dest):
105
106     no = caminho[-1]
107     while( no.anterior != None):
108         mapa[no.anterior.i][no.anterior.j]=4
109         no=no.anterior
110
111     mapa[inicio.i][inicio.j]= 2
112     mapa[dest.i][dest.j] = 3
113     return
114
```

```
115 # Para ordenar o vetor de fronteira
116 def ComparaCusto(no):
117     return no.f
118 
119 
120 # Converte os valores para marcar o caminho
121 def ConverteParaSimbolo(e):
122     if e==1: return '1'
123     elif e==2: return "P"
124     elif e==3: return "D"
125     elif e==4: return "-"
126     else: return "0"
127 
128 # Colocar as marcações no mapa
129 def ImprimirMapa(mapa):
130     for line in mapa:
131         for num in line:
132             print(ConverteParaSimbolo(num), end="")
133 
134         print("\n")
135     return
136 
137 # Função de busca A* search
138 def a_star(inicio=None,destino=None, mapa=None, file=None):
139     fronteira = list()
140     atual = No()
141     caminho = list()
142     custo_g=0
143     mapa_2=np.copy(mapa)
144 
145     # Define o custo da partida
146     inicio.f=Heuristica(inicio, destino)
147 
148     fronteira.append(inicio)
149     listafechada=list()
150 
151     count=0
152     while(not len(fronteira)==0):
153 
```

```
154         atual = fronteira[-1]
155         del fronteira[-1]
156         caminho.append(atual)
157 
158         if((atual.i==destino.i and atual.j==destino.j) or count==100000):
159             break
160 
161         # Realiza a busca de vizinhos
162         vizinhos = BuscarVizinhos(atual, mapa_2)
163         custo_g+=1
164 
165         # Atualiza a lista fechada
166         listafechada.append((atual.i,atual.j))
167 
168         # Se Houverem vizinhos
169         if(vizinhos):
170 
171             Custo_g+=1
172             for vizinho in vizinhos:
173 
174                 h = Heuristica(vizinho, destino)
175 
176                 #clássico
177                 #custo = custo_g+h
178 
179                 #Ponderado
180                 #custo =    custo_g+10*h
181 
182                 #puWU
183                 #custo = custo_g / (2*w - 1) +h if  custo_g < (2*w - 1)
                    ↪  *h else custo_g + h / w
184 
185                 #puWD
186                 custo =  custo_g +h if  custo_g < h
187                     else (custo_g + h*(2*w-1)) / w
188 
189                 # Verifica se aquele nó está na lista fechada
190                 try:
191                     it = listafechada.index((vizinho.i,vizinho.j))
```

```
192
193                 except:
194                     it = -1
195
196                 if  (it ==  -1 or custo < caminho[it].f):
197                     vizinho.f = custo
198                     vizinho.anterior = No(i=atual.i,j=atual.j,
199                         anterior=atual.anterior,f=atual.f)
200
201                     if (it!= -1): del caminho[it]
202
203                     fronteira.append(vizinho)
204
205         fronteira.sort(key=ComparaCusto, reverse=True)
206         count+=1
207
208     # Marca o caminho no mapa
209     MarcarCaminho(mapa_2,caminho,inicio,destino)
210     ImprimirMapa(mapa_2)
211     return mapa_2,caminho
212
213 # -- Final das funções para A-Star Search
214
215 # Função para movimentos do carrinho
216 def ListaMovimentos(caminho,destino):
217     no = caminho[-1]
218     caminho_carrinho = list()
219     move_carrinho = list()
220
221     # tempo de cada movimento
222
223     # ir pra frente
224     tempo_frente = 0.75
225
226     #vira a esquerda
227     tempo_esquerda = 0.65
228
229     #vira a direita
230     tempo_direita = 0.7
```

```
231
232     #Movimento lateral
233     tempo_lateral =  0.7
234
235     tempo_recuo = 0.35
236
237     # Desempacotando o caminho
238     while( no.anterior != None):
239         caminho_carrinho.append((no.anterior.i,no.anterior.j))
240         no=no.anterior
241
242     # Reverte o vetor caminho
243     caminho_carrinho=caminho_carrinho[::-1]
244
245     # Inclui destino no caminho
246     caminho_carrinho.append((destino.i,destino.j))
247
248     # Lista de movimentos do carrinho
249     for x in range(len(caminho_carrinho)-1):
250         atual = caminho_carrinho[x]
251         prox = caminho_carrinho[x+1]
252
253         # Norte -> direção = (-1,0)
254         # Leste -> direção = (0,1)
255         # Sul -> direção = (1,0)
256         # Oeste -> direção = (-1,0)
257         direcao = (prox[0]-atual[0], prox[1]-atual[1])
258
259         #  Norte = 1, Leste = 2, Sul = 3, Oeste = 4
260         if direcao == (-1,0):
261             move_carrinho.append(1)
262
263         elif direcao == (0,1):
264             move_carrinho.append(2)
265
266         elif direcao == (1,0):
267             move_carrinho.append(3)
268
269         else:
```

```
270             move_carrinho.append(4)
271
272     #print("Carrinho")
273     #print(move_carrinho)
274     anterior = 0
275     try:
276
277         # Estipulando que o carrinho começa em
278         #P, virado para o sul(FRENTE POWERBANK)
279         for move in move_carrinho:
280
281             # 1->2: Norte->Leste(direita)
282             # 2->3: Leste->Sul(direita)
283             # 3->4: sul->oeste(direita)
284             if  (move == 2 and anterior == 1)or (move == 3 and anterior
                ↪  == 2) or (move==4 and anterior==3):
285
286                 #Vira a direita
287                 GPIO.output(Motor1A,GPIO.HIGH)
288                 GPIO.output(Motor1B,GPIO.LOW)
289                 GPIO.output(Motor2A,GPIO.HIGH)
290                 GPIO.output(Motor2B,GPIO.LOW)
291                 time.sleep(tempo_recuo)
292                 stop()
293
294                 GPIO.output(Motor1A,GPIO.LOW)
295                 GPIO.output(Motor1B,GPIO.LOW)
296                 GPIO.output(Motor2A,GPIO.LOW)
297                 GPIO.output(Motor2B,GPIO.HIGH)
298                 time.sleep(tempo_direita)
299                 stop()
300
301             # 3->2: Sul->leste(esquerda)
302             # 2->1: Leste->Norte(esquerda)
303             # 4->3: oeste->sul(esquerda)
304             elif (move == 2 and anterior == 3) or (move == 1 and anterior
                ↪  == 2) or(move==3 and anterior==4):
305
306                 #Vira a esquerda
```

```
307                 GPIO.output(Motor1A,GPIO.HIGH)
308                 GPIO.output(Motor1B,GPIO.LOW)
309                 GPIO.output(Motor2A,GPIO.HIGH)
310                 GPIO.output(Motor2B,GPIO.LOW)
311                 time.sleep(tempo_recuo)
312                 stop()
313 
314                 GPIO.output(Motor1A,GPIO.LOW)
315                 GPIO.output(Motor1B,GPIO.HIGH)
316                 GPIO.output(Motor2A,GPIO.LOW)
317                 GPIO.output(Motor2B,GPIO.LOW)
318                 time.sleep(tempo_esquerda)
319                 stop()
320 
321             #em frente
322             GPIO.output(Motor1A,GPIO.LOW)
323             GPIO.output(Motor1B,GPIO.HIGH)
324             GPIO.output(Motor2A,GPIO.LOW)
325             GPIO.output(Motor2B,GPIO.HIGH)
326 
327             if move==1 or move ==3:
328                 tempo=tempo_frente
329 
330             else:
331                 tempo=tempo_lateral
332             time.sleep(tempo)
333 
334             stop()
335 
336             anterior = move
337 
338         GPIO.cleanup()
339 
340     except:
341         GPIO.cleanup()
342 
343     return
344 # -- Final das funções de movimento do carrinho
345 def main():
```

```
346
347     file = ['/home/pi/Documents/tcc_matheus/
        ↪  A_Star_Search-main/src/python/mapa_5x5.csv']
348
349     # Carrega o arquivo
350     my_data = np.genfromtxt(str(file), delimiter=';', dtype=int)
351     mapa=np.array(my_data)
352
353     inicio = No(0,0)
354
355     destino = No(len(mapa)-1,len(mapa[0])-1,f=0.0)
356     # Validação da Partida
357     if(not CelulaVazia(mapa,inicio.i,inicio.j) and VerificaLimites(mapa,
        ↪  inicio.i, inicio.j)):
358         print("Partida não é válida!")
359         return
360
361     # Validação do destino
362     if(not CelulaVazia(mapa,destino.i,destino.j) and
        ↪  VerificaLimites(mapa, destino.i, destino.j)):
363         print("Destino não é válido!")
364         return
365
366     mapa,caminho= a_star(inicio,destino, mapa,file=file)
367     ListaMovimentos(caminho, destino)
368     return
369
370 if __name__ == "__main__":
371     main()
```

SERVIÇO PÚBLICO FEDERAL
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
CURSO SUPERIOR DE BACHARELADO EM ENGENHARIA ELÉTRICA

ATA 27/2022 - CCSBEE/UA3/UA/DDE/DG/JP/REITORIA/IFPB

Coordenação do Curso Superior de Bacharelado

em Engenharia Elétrica
CCSBEE-JP

ATA DE APRESENTAÇÃO PÚBLICA E AVALIAÇÃO DE TRABALHO DE CONCLUSÃO DE CURSO

ATA Nº: (Nº / ANO)	247/2022

Às quinze horas e trinta minutos do dia sete do mêsde fevereiro do ano de dois mil e vinte e dois, por meio de webconferência utilizando a plataforma Google Meet no âmbito do Campus João Pessoa, foi realizada a Apresentação Pública e Avaliação do Trabalho de Conclusão de Curso intitulado"IMPLEMENTAÇÃO DE ALGORITMO DE PLANEJAMENTO DE TRAJETÓRIA COM ROBÔ DIFERENCIAL", do aluno MATHEUS LUCAS DE LUCENA PEREIRA, requisito obrigatório para conclusão do CURSO DE BACHARELADO EM ENGENHARIA ELÉTRICA, com os membros da Banca Examinadora Patric Lacouth da Silva, DR. (Orientador, IFPB), LINCOLN MACHADO DE ARAÚJO, DR. (Examinador, IFPB) e MARCELO MAGALHÃES ÁVILA PAZ, DR. (Examinador, IFPB). Após a apresentação e as considerações da Banca Examinadora, o trabalho foi considerado APROVADO, com nota 100 sendo esta composta pela média aritmética das seguintes avaliações parciais:

Texto:	Apresentação:	Defesa oral:
100	100	100

Eu, Patric Lacouth da Silva, DR. (Orientador, IFPB), lavrei a presente Ata, que segue assinada por mim e pelos demais membros da Banca Examinadora.

Observações:

Documento assinado eletronicamente por:

- Clleumar da Siilva Moreiira, COORDENADOR DE CURSO - FUC1 - CCSBEE-JP, em 24/03/2022 14:46:57.
- Patriic Lacouth da Siilva, PROFESSOR ENS BASIICO TECN TECNOLOGIICO, em 24/03/2022 18:22:04.
- Liincolln Machado de Araujjo, PROFESSOR ENS BASIICO TECN TECNOLOGIICO, em 24/03/2022 23:29:59.
- Marcello Magalhaes Aviilla Paz, PROFESSOR ENS BASIICO TECN TECNOLOGIICO, em 06/04/2022 08:53:04.

Este documento foi emitido pelo SUAP em 24/03/2022. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 276809
Código de Autenticação: e671e128bb

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus João Pessoa
Av. Primeiro de Maio, 720, Jaguaribe, CEP 58015-435, João Pessoa (PB)
CNPJ: 10.783.898/0002-56 - Telefone: (83) 3612.1200

Documento Digitalizado Ostensivo (Público)

Entrega do TCC

Assunto: Entrega do TCC
Assinado por: Matheus Pereira
Tipo do Documento: Dissertação
Situação: Finalizado
Nível de Acesso: Ostensivo (Público)
Tipo do Conferência: Cópia Simples

Documento assinado eletronicamente por:

- **Matheus Lucas de Lucena Pereira, ALUNO (20162610015) DE BACHARELADO EM ENGENHARIA ELÉTRICA - JOÃO PESSOA, em 12/09/2022 18:24:26.**

Este documento foi armazenado no SUAP em 12/09/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 622305
Código de Autenticação: 7b10882568