Robô Móvel Semi-Autônomo de Busca

Bruna Buium, Roberto M. Marinheiro, Professor Dr. Calebe de Paula Bianchini

[1]Faculdade de Computação e Informática – Universidade Presbiteriana Mackenzie
São Paulo – SP – Brasil

brunabuium@hotmail.com, robertom@marinheiro.com.br,

calebe.bianchini@mackenzie.br

Abstract. *Machines' capability to reproduce human abilities is the new source of technological advances. One of the most discussed approaches are autonomous vehicles, machines that can move from place to place without human intervention and in a safer way, since they can even "predict"accidents. The objective of this research is to present algorithms for the development of a mobile searcher robot, qualifying it to autonomously search an object in an environment previously known by the robot. To this end, algorithms that fit the case were selected and implemented. Such elected algorithms proved being able to construct the robot's intelligence efficiently and insightfully.*

Resumo. *A capacidade concebida às máquinas de reproduzir habilidades humanas é a nova fonte dos avanços tecnológicos. Uma das abordagens mais discutidas são os veículos autônomos, máquinas capazes de se locomoverem sem a intervenção de seres humanos e de uma maneira mais segura, uma vez que estas podem até "prever"acidentes. O objetivo da pesquisa é apresentar algoritmos para o desenvolvimento de um robô móvel de busca, qualificando o para buscar, de forma autônoma, um objeto em um ambiente previamente conhecido pelo robô. Para tal, foram selecionados e implementados algoritmos que se encaixassem no caso. Tais algoritmos eleitos se mostraram capazes de construir a inteligência do robô de maneira eficiente e perspicaz.*

1. Introdução

Os avanços na área da robótica decolaram nos últimos anos, porém, seu estudo vêm sendo desenvolvido há muitos séculos. A data do surgimento dos primeiros autômatos, ancestrais dos atuais robôs, é incerta, tendo como grande referência a primeira máquina de calcular, criada por Pascal em 1642. A partir deste, inúmeros autômatos foram desenvolvidos, entretanto, o termo "robô"só começou a ser empregado a partir de 1923, sendo usado pela primeira vez por Karel Capek, em sua peça "A fábrica de robôs", onde os robôs são definidos como seres obedientes e responsáveis por realizar todo trabalho físico para os seres humanos (DUDEK, 2000).

Os primeiros robôs eram na verdade autômatos, executando as mesmas tarefas de modo repetitivo. Estes deram origem aos atuais braços manipuladores, largamente adotados em indústrias. Mais recentemente, surgiram os robôs móveis, com a habilidade de se locomover de modo guiado, semi-autônomo ou totalmente autônomo (DUDEK, 2000).

Este trabalho tem como objetivo apresentar os resultados da construção de um robô móvel semi-autônomo que, a partir da entrada de um mapa totalmente conhecido e de seu ponto inicial, é capaz de encontrar um alvo. Nas seções seguintes serão apresentados os modelos utilizados para o desenvolvimento do robô.

2. Mapeamento

A primeira definição necessária para um robô ter a habilidade de se locomover em um ambiente é o mapeamento, que combinado à localização, gera uma "percepção" do mundo real. O mapeamento consiste na abstração de um ambiente real, em um ambiente digital, ou seja, na criação de um mapa propriamente dito, do local especificado. Existem diversas maneiras de fazer o mapeamento, nesta seção são tratadas duas delas.

A grade ocupacional, que consiste na definição de um mapa através de uma matriz bidimensional, onde cada uma de suas coordenadas representa uma porção do espaço real. Uma das abordagens mais comuns é a utilização do binário, onde 0's representam pontos livres, e 1's representam obstáculos ou paredes.

E o mapeamento através de um grafo, que é essencialmente, um conjunto de nós que possuem ligações entre si, tais ligações indicam os caminhos possíveis de um nó a outro.

3. Algoritmos de Visibilidade

O fenômeno visibilidade consiste na seguinte afirmação: dois pontos são mutuamente visíveis se, e somente se, o segmento de reta que os conecta não for obstruído por qualquer parede e ou obstáculo. Portanto, a determinação da visibilidade tem como função definir as áreas visíveis e não visíveis do dado ambiente, à partir de um determinado ponto.

3.1. Galeria de Arte

O problema da galeria de arte tem como origem o problema de se vigiar toda a área de uma galeria de arte com o menor número de guardas possível. A galeria pode ser determinada como um polígono simples, ou seja, uma forma geométrica delimitada por segmentos de linhas retas que não cruzam umas às outras, não tem paredes internas, e que não contém polígonos dentro de polígonos caracterizados como buracos. Portanto, os pontos guardas tem campo de visão de 360º. (DE BERG, CHEOG, VAN KREVELD e OVERMARS, 2008).

Em algumas das variações do problema da galeria de arte, pode-se encontrar polígonos com "buracos", ou, polígonos que contém outros polígonos dentro dele.

A determinação dos pontos de guardas ocorre através da triangulação e a 3-coloração do ambiente que serão detalhados posteriormente.

3.1.1. Triangulação

A triangulação do polígono é realizada através da decomposição do mesmo em diversos triângulos. Os triângulos serão formados a partir do desenho de uma diagonal, segmento de reta, que junta dois vértices que são mutualmente visíveis.

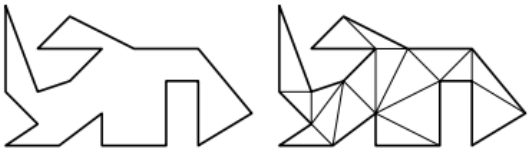


Figura 1. Exemplo de polígono antes e depois da triangulação

Conforme visto na Figura 1, há várias possibilidades de ligar vértices que não necessariamente formam triângulos. Pensando dessa maneira, é possível notar que uma triangulação não é única. Portanto, determina-se um nó a cada triangulo formado e caso esses triângulos possuírem diagonais em comum esses nós serão ligados formando, então, um grafo. A partir dessas informações, é possível chegar a algumas conclusões:

1- Toda triangulação de polígonos simples com n vértices usam n-3 diagonais e formam n-2 triângulos;

2- Toda triangulação de polígonos com b buracos com n vértices usam n + 3b – 3 diagonais e formam n + 2b – 2 triângulos;

3- O grafo formado a partir da triangulação de um polígono simples forma uma árvore;

4- O grafo formado a partir da triangulação de um polígono com buracos deve conter um ciclo.

Considerando que o algoritmo de triangulação é um algoritmo de desenvolvimento altamente complexo, são conhecidos alguns algoritmos de triangulação, com os critérios para resolução do problema da galeria de arte, dentre eles estão o y-monotone criado por Lee e Preparata e provado que funciona em polígonos com buraco por Asano et al., e a técnica de plane sweep criada por Mehlhorn e provado que funciona em polígonos com buraco por Ghosh and Mount.

3.1.2. 3-coloração

Após a finalização do processo de triangulação e determinados os triângulos, é aplicado a técnica de coloração dos vértices. A 3-coloração dos vértices é um processo de colorir os vértices dos triângulos formados, de tal maneira que nenhum triângulo contenha vértices de cores iguais, conforme Figura 2.

3.2. Ray Shooting

Procurando definir pontos para visibilidade total de um mapa, o método Ray Shooting é totalmente voltado a computação gráfica, onde raios, segmentos de reta, são disparados do ponto de início aos pontos desejados verificando a área visível.

Este algoritmo pode ser aplicado de diversas maneiras dependendo da representação do ambiente, no caso de ambientes definidos como polígonos, onde temos os vértices do ambiente e os vértices dos obstáculos, é possível disparar os raios apenas nestes vértices formando triângulos que definem as áreas visíveis. Já no caso de ambientes representados por grids, é necessário disparar os raios em todo o contorno do mapa,

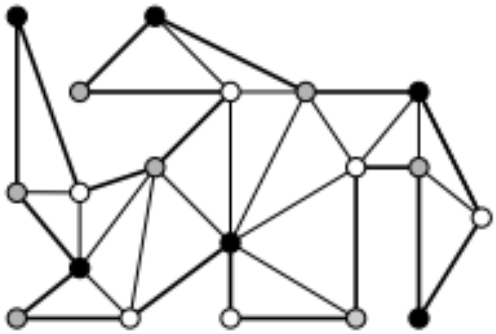


Figura 2. Polígono após a execução da triangulação e 3-coloração

ou seja, para todas as direções, sendo que, os raios continuam a avançar até encontrar obstáculos como pode ser visto na Figura 3.

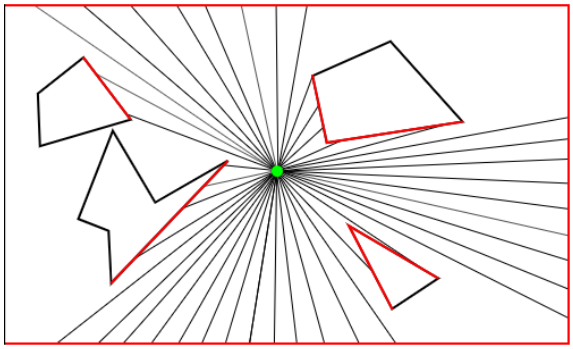


Figura 3. Execução do Ray Shooting a partir de um ponto

3.2.1. Algoritmo de Bresenham

O algoritmo de Bresenham consiste em traçar linhas, em ambientes ”digitais”, ou seja, não analógicos, pois os valores são finitos. Essa linha será desenhada entre dois pontos dadas as suas coordenadas.

O algoritmo se move a partir da linha horizontal (x) e sempre acresce uma unidade e então verifica se precisa ou não aumentar também a linha vertical (y). Ou seja, a próxima coordenada será (x+1, y) ou (x+1, y+1). Considerando que o algoritmo sempre parte do canto inferior esquerdo para o superior direto.

4. Algoritmos de Planejamento de Rotas

Uma das funções mais essenciais para qualquer tipo de robô móvel é o planejamento de rota, este problema é generalizado em produzir uma movimentação contínua que conecte uma configuração inicial e uma configuração objetivo, evitando colisão com obstáculos. (LAVALLE, 2006) O robô e os obstáculos são representados em um ambiente 2D ou 3D, enquanto a movimentação é representada por uma rota no espaço de configuração.

O conceito de espaço de configuração é definido pela combinação de todas as

configurações possíveis do robô no ambiente. Dentro deste espaço de configuração existem importantes representações:

- Espaço livre: são as configurações que evitam colisões com obstáculos;

- Espaço alvo: subespaço de espaço livre que descreve onde é desejado que o robô se mova;

- Espaço de obstáculo: é o espaço onde o robô não pode se mover.

Quando é falado sobre planejamento de rotas, ou path planning termo comumente utilizado na robótica, muitas vezes, esse termo é confundido com planejamento de movimentação. (LAVALLE, 2006) A principal diferenças entre eles é que o planejamento de rotas determina a rota que um objeto deve realizar e o planejamento de movimento determina quais movimentos devem ser projetados para realizar a rota determinada.

O planejamento de rotas é usado para resolver problemas em diferentes campos, desde o planejamento simples de rotas espaciais até a seleção de uma sequência de ações apropriada necessária para atingir um determinado objetivo. Como o ambiente nem sempre é conhecido com antecedência, esse tipo de planejamento geralmente se limita aos ambientes projetados com antecedência e aos ambientes que podemos descrever com precisão antes do processo de planejamento. O planejamento do caminho pode ser usado em ambientes totalmente conhecidos ou parcialmente conhecidos, bem como em ambientes totalmente desconhecidos, onde as informações detectadas definem o movimento desejado do robô.

O estado ou configuração fornece uma possível posição do robô no ambiente e pode ser representado como um ponto no espaço de configuração que inclui todos os estados possíveis do robô. O robô pode passar de um estado para outro implementando ações diferentes. Um caminho adequado é, portanto, descrito como uma sequência de ações que guiam o robô desde a configuração inicial (estado) através de algumas configurações intermediárias até a configuração do objetivo. Qual ação é escolhida no estado atual e qual será no próximo depende do algoritmo de planejamento de caminho usado e dos critérios usados. O algoritmo escolhe o próximo estado mais adequado do conjunto de todos os estados possíveis que podem ser visitados a partir do estado atual. Essa decisão é tomada de acordo com a função de alguns critérios, geralmente definida com uma das medidas de distância, como a menor distância euclidiana ao estado objetivo.

4.1. Algoritmos não complexos

Os algoritmos de Bug foram desenvolvidos por Lumelsky & Stepanov no ano de 1987. Esses códigos são os mais antigos e utilizados para planejamento de rotas com sensores sendo os mais provados e garantidos na época que foram desenvolvidos. Esses algoritmos assumem que o robô é um ponto que opera no plano com um sensor de contato para detectar obstáculos.

O algoritmo Bug1 consiste em ir em direção ao ponto destino em linha reta até encontrar um obstáculo identificado a partir de um sensor ou no melhor caso o destino. Ao encontrar o obstáculo o robô deve:

1- Procurar qual é o próximo ponto mais próximo para chegar ao destino. Para isso ele vai fazer a volta completa no obstáculo e quando identificar esse ponto deve percorrer

até esse ponto.

2- Continuar a andar em linha reta até encontrar o objeto. Caso encontre outro obstáculo voltar ao passo 1. (LUMELSKY e STEPANOV, 1987)

Já o algoritmo Bug2 consiste em traçar uma linha (m) entre o ponto inicial e o destino e seguir em linha reta até encontrar um obstáculo identificado a partir de um sensor ou no melhor caso o destino desejado. Ao encontrar o obstáculo o robô deve:

1- Percorrer para o seu lado esquerdo e procurar qual é o próximo ponto que faz parte da linha m. Para isso ele vai fazer a volta no obstáculo apenas até encontrar esse ponto.

2- Continuar a andar em linha reta até encontrar o objeto. Caso encontre outro obstáculo voltar ao passo 1. (LUMELSKY e STEPANOV, 1987)

4.2. Algoritmos de Busca

Os algoritmos de busca tem as mais diversas aplicações, foi muito estudado na área da teoria dos grafos. Para o planejamento de rotas eles irão explorar e manipular diferentes estruturas de dados para conseguir encontrar, de maneira eficiente, o melhor caminho entre dois pontos.

4.2.1. Algoritmo de Busca em Largura

O algoritmo de busca em largura, ou Breadth First Search (BFS) como é chamado em inglês, procura o alvo através da expansão das possibilidades de movimento do robô. Por exemplo, se durante o percurso houver um obstáculo ao lado direito do robô, ele vai poder andar em uma unidade para qualquer direção, com exceção da direita, pois identificou que aquele espaço não está disponível e também não pode revisitar a posição de onde ele veio. A estrutura da expansão fica parecida com uma árvore que expande através da verificação de todas as possibilidades onde não há colisão e são adicionadas a uma estrutura de fila para verificação de quais são as próximas possibilidades que entrarão na fila até ser encontrada a posição desejada.

Para esse tipo de algoritmo, a pesquisa será concluída desde que o fator da ramificação seja finito, isso significa que sempre será retornada uma solução se ela existir. No sentido de planejar o caminho, o BFS sempre o encontrará a partir da posição inicial até a meta, desde que exista um caminho real que possa ser encontrado.

O BFS só é ideal se o custo do caminho for o mesmo para cada direção. De acordo com Russel (1995, p. 74) ”Para essa solução é necessário considerar o tempo levado e a quantidade de memória disponível para completar a busca”. Podemos ver com isso que, para um espaço de trabalho muito grande, onde a meta está profundamente dentro do espaço de trabalho, o número de nós pode expandir-se exponencialmente e exige um requisito de memória muito grande.

4.2.2. Algoritmo de Busca em Profundidade

O algoritmo de busca em profundidade, mais conhecido pelo seu nome em inglês Depth First Search (DFS), se assemelha bastante com o BFS. Além disso, ele também procura o alvo através da expansão das possibilidades de movimento do robô; porém, o algoritmo ganha profundidade mais rápido que o BFS. Por exemplo, se durante o percurso houver um obstáculo ao lado direito do robô, ele vai poder andar em uma unidade para qualquer direção, com exceção da direita, pois identificou que aquele espaço não está disponível e também não pode revisitar a posição de onde ele veio. A partir desse ponto, a estrutura é modificada: enquanto o algorítimo do BFS usa uma estrutura de fila, o DFS vai usar a de pilha, mais conhecida como *"first in, last out"*, para a criação de uma árvore.

A estrutura da expansão fica parecida com uma árvore, que expande através da verificação de todas as possibilidades onde não há colisão e são adicionadas ao topo da pilha. As possibilidades são inseridas na pilha conforme elas vão sendo conhecidas. Uma vez encontrado o destino, deve-se fazer o rastreamento de volta ao início. Sendo assim denominado como algoritmo de busca em profundidade.

A vantagem de usar o algoritmo de DFS segundo Russell (1995, p. 77) "requere um uso muito modesto de memória, uma vez que só memoriza um caminho do início ao fim."Se existem duas possibilidades de caminhos e o objetivo é encontrado no primeiro nó, não será necessário expandir a segunda possibilidade. Já a desvantagem é que o DFS precisa realizar o caminho de volta, e quando há uma possibilidade de caminho, mesmo que o destino não esteja no final, esse deve ser testado antes.

4.3. Algoritmos Modernos

Os algoritmos de planejamento de rotas modernos procuram sempre encontrar rotas de custo mínimo. Para estes algoritmos, utilizaremos o conceito de grafos para suas explicações, mas os algoritmos podem ser adaptados para qualquer representação de mapa, inclusive para grade ocupacional.

4.3.1. Decomposição de Células

O objetivo do planejamento de rotas baseado na decomposição de células é a criação de um grafo conexo onde são traçadas linhas verticais a partir de todos os vértices do polígono. Após essa linha ser tracejada, serão formados diversos trapézios. Então, deve-se marcar o centro e traçar o caminho até o centro do próximo trapézio. Dessa forma, teremos formado o caminho entre um ponto de inicio e o destino objetivado.

As vantagens de usar um algoritmo desse tipo são que é computacionalmente mais simples, mais estável, simples de implementar. (SWINGLER and FERRARI, 2010)

4.3.2. Exploração Rápida da Árvore Randômica

A ideia da exploração rápida da árvore randômica, ou Rapidly exploring Random Tree (RRT) como é reconhecido em inglês, é construir uma árvore de pesquisa de forma incremental que melhore a resolução, mas não precisa definir explicitamente um parâmetro

de resolução. No maior número de iterações, a árvore, densamente, cobre o espaço inteiro. Assim, possui propriedades semelhantes às curvas de preenchimento de espaço, porém, em vez de um caminho longo, existem caminhos mais curtos e organizados em uma árvore. Uma sequência densa de amostras é usada como um guia no processo incremental na construção da árvore. (KUFFNER e LAVALLE, 2000)

O algoritmo consiste em, a partir de um ponto aleatório que será definido, se conectar com o ponto mais próximo que pertence a árvore. Exemplificado da Figura 4.

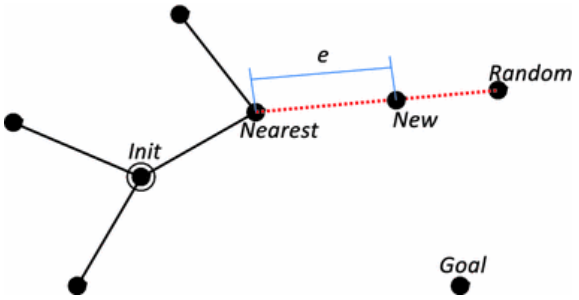


Figura 4. Mecanismo de expansão de uma RRT

A vantagem de usar essa solução é que há um balanceamento entre a busca e a exploração e também é um método fácil de ser implementado. Em contrapartida ele tem métricas sensíveis e tem uma taxa de convergência desconhecida.

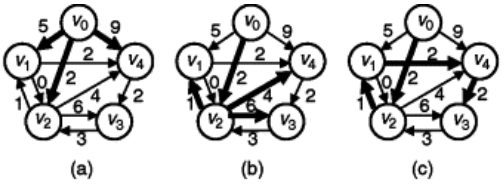
4.3.3. Djikstra

A ideia surgiu em 1956, sua criação finalizada no ano de 1959 e a publicada em 1960, o algoritmos de Dijkstra foi desenvolvido originalmente para calcular a menor distância entre as cidades de Rotterdam to Groningen na Holanda. Foram então selecionadas 64 cidades no mapa pra realizar esse cálculo possível. (DJIKSTRA, 1959)

O algoritmo de Dijkstra resolve o problema do caminho mais curto para qualquer grafo direcionado sendo que cada vértice contém pesos não negativos. Ele pode manipular grafos que contém ciclos, grafos que possuem na lista duas vezes um mesmo vértice não seguidas, porém, os pesos negativos farão com que esse algoritmo produza resultados incorretos.

O Djikstra mantém uma fila de prioridade usada para armazenar os vértices não processados com suas estimativas de caminho mais curto como valores-chave. Em seguida, extrai repetidamente o vértice que tem o valor mínimo da fila de prioridade e verifica todas as arestas incidentes do vértice escolhido para qualquer vértice na fila. Depois que um vértice é extraído da fila de prioridade e todos as arestas nele concluídos, o algoritmo tratará esse vértice como processado e não o tocará novamente. O algoritmo de Dijkstra é finalizado quando a fila de prioridade está vazia ou quando todos os vértices são examinados exatamente uma vez.

A Figura 5 ilustra a execução do algoritmo de Dijkstra em um grafo direcionado com custos não negativos e o grafo também contém um ciclo.



vertex	Predecessors					Shortest-Path Estimates				
	v_0	v_1	v_2	v_3	v_4	v_0	v_1	v_2	v_3	v_4
non	NIL	NIL	NIL	NIL	NIL	0	∞	∞	∞	∞
v_0	NIL	v_0	v_0	NIL	v_0	0	5	2	∞	9
v_2	NIL	v_2	v_0	v_2	v_2	0	3	2	8	6
v_1	NIL	v_2	v_0	v_2	v_1	0	3	2	8	5
v_4	NIL	v_2	v_0	v_4	v_1	0	3	2	7	5
v_3	NIL	v_2	v_0	v_4	v_1	0	3	2	7	5

Figura 5. Exemplo de execução do algoritmo Djikstra

4.4. Inteligência Artificial

Na inteligência artificial, o planejamento de rotas significa uma pesquisa por uma sequência de ações lógicas que transformam um estado inicial do robô em uma meta desejada. Esse planejamento pode incluir muitas ideias teóricas de decisão, como processos de decisão de Markov, informações de estado imperfeitos, métodos de aprendizagem ou equilíbrio teórico dos jogos.

4.4.1. A*

O algoritmo de busca A* consiste em usar uma heurística para encontrar a melhor solução, completa e uma variante do melhor algoritmo. Ele garante que será encontrada a melhor solução possível e nesse caso o algoritmo que encontrará o menor caminho entre o ponto anterior e o destino. Um algoritmo ser completo significa que se existe uma possibilidade de solução para o problema, esse algoritmo a encontrará.

A grande evolução deste algoritmo em relação ao anterior, Dijkstra, está no número de nós expandidos, uma vez que o algoritmo de Dijkstra expande para todos os nós vizinhos, o algoritmo A* faz uso de uma função heurística para priorizar nós específicos.

Ele prioriza os nós através de uma combinação de g(n) que é a distância entre o ponto atual ao ponto inicial, com a função h(n) que é a distância estimada do ponto atual ao ponto objetivo, dado matematicamente pela seguinte equação:

$$f(n) = g(n) + h(n)$$

O algoritmo é iniciado com a declaração das variáveis e de duas listas, essenciais para o mesmo. A primeira lista, comumente nomeada de aberta, contém os nós ou posições que já foram expandidos, enquanto a segunda lista, normalmente nomeada de fechada contém os nós ou posições que já foram expandidas. A lista aberta é inicializada com o ponto inicial, e a lista fechada é inicializada vazia.

Seguimos tais métodos até a lista aberta estiver vazia (o que implicará na falha de criar um caminho, ou seja, não existe caminho do ponto inicial ao ponto objetivo): retirar de aberta o vértice N com f mínimo e colocá-lo em fechada, se N for o objetivo o algoritmo retorna o caminho mais curto (utilizando de backtracking, seguindo os pontos

com menor custo f), caso não seja, gera os filhos de N, e para cada filho n', ignora este se ele está na lista fechada, se ele não está na lista aberta, o adiciona e se este já estava na lista aberta e define o ponto atual como parente de n', seguindo, verifica se o caminho para este vértice é melhor, usando o custo G como medida, se for o caso, recalcula custo G e F e altera o parente de n' para o ponto atual (ZANCHIN, 2018).

5. Detalhamento do Projeto

Para implementação da pesquisa foram usadas duas diretrizes, uma lógica e outra física. Nas seções seguintes serão tratadas separadamente cada uma delas através da retomada de conceitos explicados ao longo da pesquisa e como eles foram utilizados para a criação e a implementação do robô semi-autônomo, bem como os resultados obtidos através dos testes realizados de cada módulo.

5.1. Diretriz lógica

Para a diretriz lógica, a linguagem C foi utilizada por conta de sua velocidade e de sua compatibilidade com microprocessadores Arduino, que detalharemos mais à frente.

Foram implementados dois módulos base para o robô: visibilidade e planejamento de rota, onde o primeiro é responsável por toda a geração dos pontos de guarda necessários e o segundo pelo planejamento das rotas entre os mesmos.

Iniciando a rotina principal, é feita a leitura de um arquivo de texto que contém os parâmetros de entrada necessários para toda a diretriz lógica. O arquivo é composto pelas dimensões do mapa: altura e largura, e em seguida, é descrito o mapeamento do ambiente utilizando a técnica de grade ocupacional. Este mapeamento foi selecionado pois é a maneira mais simples de fazer a abstração do ambiente, sendo apenas necessário verificar o local e descrevê-lo na forma de um panorama, este aspecto pode também ser visualizado na Figura 6.

Na leitura do arquivo texto, ainda é recebida a posição e a direção inicial que robô se encontra no mapa. Essas informações são lidas e armazenadas na função initMapa, mostrada na Figura 7, e então são utilizadas para execução do algoritmo de visibilidade, porém, a direção do robô será apenas explorada mais a frente.

Visibilidade total do mapa contido no arquivo texto é obtida, através da seleção e implementação do algoritmo de visibilidade Ray Shooting. Em conjunto com a maneira de representação do mapa, o Ray Shooting é menos complexo para implementação e mais eficaz para solução, de ambientes que contém obstáculos, que o outro algoritmo estudado.

Ao iniciar o Ray Shooting, primeiramente, é executada a função initVisibilidade para calcular a visibilidade a partir da emissão de raios, criados pelo algoritmo de Bresenham, do ponto inicial para todas as direções em 360○.

Pode-se dizer, então, que a função de visibilidade traz um resultado de onde é possível obter visão e onde não através da execução do algoritmo. Se realizada uma representação gráfica de um mapa que contém obstáculos, conforme a Figura 8, pode-se concluir que os pontos em azul representam a porção não visível do ambiente a partir do ponto inicial, que é representado pelo ponto em amarelo. Nesse caso o algoritmo determinará novos pontos de guarda para obter a visão completa do mapa.

```
30
30
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
1 3
```

Figura 6. Arquivo exemplo utilizado como entrada para execução do algoritmo

Os pontos de guardas, conforme mencionado anteriormente, é uma lista composta pelo ponto inicial mais os pontos que devem ser visitados. Estes são determinados através da execução do Ray Shooting. A emissão dos raios verificam qual é a porção visível do mapa e então é ele percorrido sequencialmente desde o início até ser encontrado o próximo ponto não visível. Este ponto será adicionado à lista de guardas. Isso é realizado até que o mapa completo seja visível.

No melhor caso o mapa é completamente visível a partir do ponto inicial. Já em outros casos, o mapa deve ser inteiramente percorrido até que não possua nenhum ponto não visível. Uma vez que o mapa inteiro torna-se visível, o algoritmo é finalizado. Na Figura 9 é exemplificado um mapa onde são mostrados os pontos de guarda e suas respectivas visibilidades.

Todos os pontos determinados pelo algoritmo, que compõem a lista de guardas, são utilizados para ação do planejamento de rota e traçar o melhor caminho entre os pontos.

De todos os métodos de planejamento de rotas estudados, para a implementação da pesquisa, foi escolhido o A* por dois motivos: a uma redução de espaços que devem ser visitados por causa da utilização de uma heurística de distância e por possuir uma resposta mais rápida que o Djikstra e de menor custo comparado aos algoritmos simples que foram estudados.

No projeto mais especificamente, o algoritmo A* utiliza a heurística euclidiana, $\sqrt{(pi - pd)^2 - (pi - pd)^2}$ onde *pi* é o ponto inicial e *pd* é o ponto destino. Essa heurística é utilizada para conseguir encontrar o melhor caminho entre dois pontos. A seleção dos mesmos é iniciada no ponto de partida determinado no initMapa e continuada pela lista

```c
int main(){
    Mapa *mapa = initMapa();

    Visibilidade *visibilidade = initVisibilidade();

    processamentoVisibilidade(mapa, visibilidade);

    ListaPath * listaPath = initListaPath(visibilidade, mapa);

    Robot *robot = initRobot();

    percorrePath(robot, mapa, listaPath);

    liberaMapa(mapa);
    liberaVisibilidade(visibilidade);
    liberaListaPath(listaPath);
    liberaRobot(robot);

    return 0;
}
```

Figura 7. Seção principal do algoritmo onde ocorre a execução das funções descritas ao longo da pesquisa

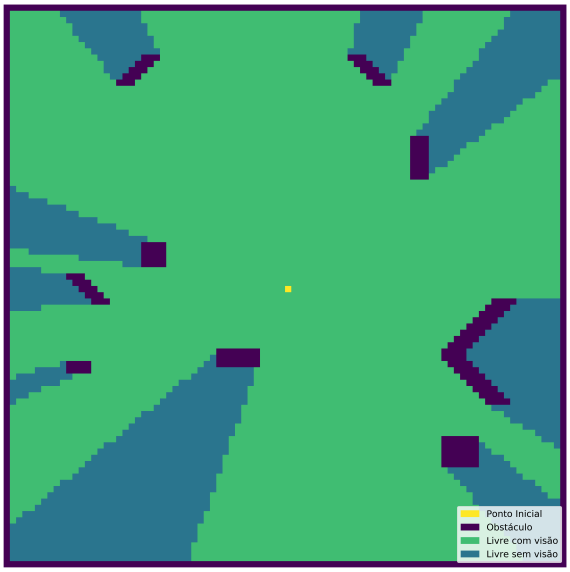


Figura 8. Ray Shooting com auxilio do Algoritmo de Bresenham

de pontos de guarda gerada pelo algoritmo de visibilidade conforme visto anteriormente.

A inicialização da estrutura do caminho foi realizada através da função initListaPath, conforme indicado na Figura 7, que recebe como parâmetro o resultado da visibilidade e o mapa. Na função, é determinado o tamanho da lista e quais são as possíveis rotas mínimas que o robô pode realizar de forma rápida.

Na mesma função é utilizado o A*. Para tal, também foi essencial o desenvolvimento de uma estrutura de árvore binária máxima, conhecida como heap, um tipo de estrutura de dados baseada em uma árvore completa, pois em muitas iterações do A* existe um método de selecionar o nó com menor custo, e buscar por este nó a cada iteração perderia toda a eficiência do algoritmo. Portanto, a heap faz com que esta lista sempre esteja ordenada, limitando todo aquele processamento de busca à uma simples tarefa de remoção do ponto da heap.

O planejamento com o algoritmo A* é eficiente, porém, também foi verificado

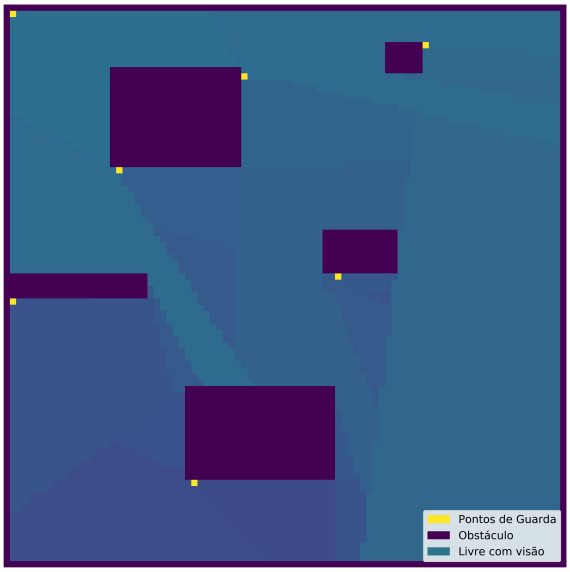


Figura 9. Resultado do algoritmo de visibilidade destacando os pontos de guarda e suas respectivas visibilidades

que é necessário executar o planejamento do ponto inicial até todos os outros pontos de guarda, para verificar o caminho mais curto o ponto de guarda mais próximo, em relação a distância do caminho deste e não da distância exata entre os pontos, do mesmo, e depois partir deste ponto mais próximo ao ponto mais próximo deste, etc.

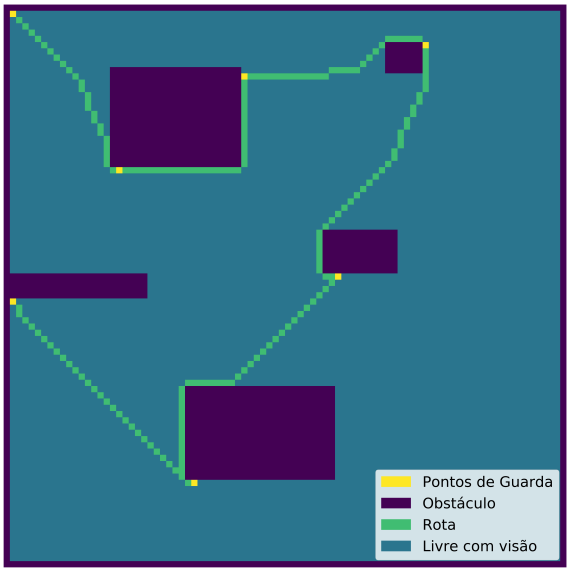


Figura 10. Pontos de guarda e resultado da execução do algoritmo A*.

Na sequencia o algoritmo volta para a execução do initListaPath que recebe um caminho encontrado no A*, e verifica, através de comparações, se o custo do caminho encontrado é menor que o custo do caminho armazenado anteriormente. Caso o custo seja mais baixo, o novo caminho encontrado será o melhor naquele momento e o valor do menor custo é atualizado. O processo se repetirá até todos os possíveis caminhos serem encontrados. O resultado do algoritmo de planejamento de rotas pode ser visualizado na

Figura 10.

5.2. Diretriz física

Já no módulo físico da pesquisa, será introduzido e explicado a função de cada componente utilizado para composição do robô, uma imagem do modelo físico e como é feito o controle do mesmo.

O robô é sustentado por uma estrutura de acrílico, onde são acopladas duas rodas com motores DC e uma roda de apoio e sem tração, conforme mostrado na Figura 11. Na parte de cima do robô, encontra-se a placa micro-controladora, uma bateria de 9V, Arduíno 2560, juntamente com seus módulos, listados a seguir:

- Chassi 2WD de acrílico, acoplado com duas rodas, dois motores DC e uma roda de apoio sem tração.

- Arduino MEGA 2560.

- Ponte H L298N.

- Módulo Micro SD.

- Micro SD 16 GB.

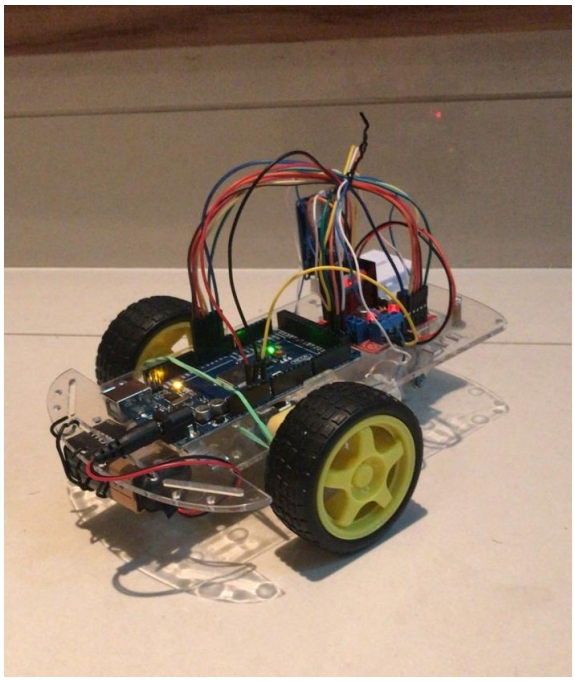


Figura 11. Estrutura física do robô

O controle e processamento do robô faz uso do Arduino MEGA 2560, uma placa microcontroladora com 256KB de memória flash, 54 pinos de entrada/saídas digitais e 16 entradas analógicas. A seleção desta placa é devido a sua capacidade de armazenamento elevada e seu número de pinos, tais características o difere e o torna superior em relação à outras distribuições de Arduíno.

A movimentação é toda executada pela Ponte H L298N, uma placa para controle de motores que possui dois canais e permite controlar a velocidade e sentido de rotação

dos dois motores ao mesmo tempo, diminuindo os ruídos e trazendo precisão ao acionar os mesmos.

Para o armazenamento dos arquivos de entrada, foi selecionado um cartão Micro SD de 16GB, que é lido pelo Arduino através do módulo Micro SD, um módulo específico para a leitura desse tipo de cartão.

5.3. Combinação das diretrizes lógica e física

Para a combinação da diretriz física e lógica do sistema, todas as funções descritas acima foram carregadas na memória do Arduíno. Ao iniciar o robô, é feita a leitura do arquivo de entrada carregado no cartão Micro SD e então, as informações são armazenadas em estruturas locais. Feita a leitura, o robô faz toda a rotina citada na diretriz lógica, tendo como resultado os pontos de guarda e a lista de caminhos entre os mesmos. E então, a função para percorrer os caminhos da lista foi desenvolvida com o propósito de gerar a menor quantidade de ruído possível. Pra tal, a movimentação é feita com dois tipos de ação: deslocamento para frente e rotação, descritas na sequência.

O deslocamento para frente é executado de tal forma: os motores são ligados, é chamada uma função nativa da linguagem do Arduino: delay(x) que pausa a execução e após o valor x de tempo (em milissegundos) continua a rotina do programa. Retomando a execução, os motores são desligados. Esta lógica gera o deslocamento. A quantidade de tempo necessária para fazer o deslocamento para frente em uma determinada distância, foi definida de maneira empírica, testando o tempo de delay e a quantidade de movimento que este tempo gerava.

Para as rotações, foram definidos oito números inteiros, uma para cada direção possível do robô em relação ao mapa, sendo elas: 0: Noroeste; 1: Norte; 2: Nordeste; 3: Leste; 4: Sudeste; 5: Sul; 6: Sudoeste e 7: Oeste. Com estas direções, foi desenvolvida uma função que, dado um ponto inicial, uma direção inicial e um ponto final, calcula o sentido (horário ou anti horário) e a quantidade mínima de rotações, sendo cada unidade de rotação igual a 45º, necessárias para: do ponto inicial, com a direção inicial, rotacionar e estar na direção correta para seguir em frente e chegar ao ponto final, ou seja: de frente para o ponto final. Vale mencionar que a direção pode estar previamente correta, neste caso, não há rotação.

A ação das rotações consiste em acionar somente um motor, no caso do motor direito, a rotação será anti horária, e no caso do motor esquerdo, a rotação será horária, utilizando a função delay() para auxiliar na quantidade de rotação. Assim como a definição da quantidade de movimento para uma determinada distância, foi utilizado o método empírico para fazer a definição da quantidade de movimento necessária para a execução de uma unidade de rotação (45º).

Para o robô se movimentar através dos caminhos gerados, a função para percorrer os caminhos seleciona ponto a ponto dos mesmos, e executa a tomada de ação necessária pelo robô. Então, dadas a posição e direção inicial do robô, essa função seleciona o primeiro ponto de guarda do primeiro dos caminhos, calcula a rotação necessária do robô e a executa, acionando um de seus motores, após fazer esta rotação, o robô se movimenta uma unidade para frente, chegando ao primeiro ponto de guarda, este processo se repete até o robô percorrer todos os pontos guarda de cada caminho da lista de caminhos gerada. Unindo as diretrizes lógica e física, compondo o sistema do robô.

6. Conclusão

Após a realização de testes, foi concluído que todo o código implementado é bem estruturado, organizado, e funcional, contando com uma série de estruturas que auxiliam na execução das tarefas e métodos.

O algoritmo de geração de pontos de guarda através do cálculo de visibilidade se mostrou excepcional, dada a estrutura de seu código, para todos e quaisquer tipos de mapa, ele é capaz de retornar os pontos de guarda de tal forma que todas as áreas do mapa terão visibilidade, sem exceção, uma vez que o algoritmo não para de adicionar guardas até todas as áreas serem visíveis. A combinação deste algoritmo com o algoritmo de A* se provou uma solução viável para a inteligência do robô móvel de busca, uma vez que o retorno da diretriz lógica do projeto atinge os objetivos propostos pelo robô.

Entretanto, o hardware selecionado não foi capaz de atingir os resultados desejados. Mais especificamente: as rodas do chassi do robô são de plástico, o que faz com que o robô deslize muito facilmente; a estrutura do chassi em conjunto com a terceira roda sem tração acabam gerando muito ruído em razão de não serem capazes de executar as rotações corretamente; e os motores acoplados tornam as movimentações do robô imprecisas.

No término desta pesquisa, foi verificado que a inteligência desenvolvida é válida para robôs móveis de busca, porém, apesar do hardware funcionar, o robô é impedido de atingir resultados ótimos dados os problemas citados acima.

Como trabalho futuro, é possível melhorar o hardware utilizando um motor mais preciso, rodas de borracha e um chassi com apenas duas rodas, melhorando a movimentação do robô, outras ações seriam a instalação de uma câmera no circuito do robô e a implementação de procedimentos capazes de identificar objetos nas imagens, provenientes da câmera, capacitando o robô a fazer a busca.

7. Referências Bibliográficas

Breadth-First and Depth-First Search for Path Planning. Disponível em: http://www.daslhub.org/unlv/wiki/lib/exe/fetch.php?media=dylanw:bfsdfstutorial.pdf. Acesso em: 15 de outubro de 2019.

BUKHOI, Iksan. ZOOL, Hilmi I. Detection of kidnapped robot problem in Monte Carlo localization based on the natural displacement of the robot. International Journal of Advanced Robotic Systems, Julho de 2017.

Burgard, Wolfram, Stachniss, Cyrill, Bennewitz, Maren, Arras, Kai. Introduction to Mobile Robotics. University Frieburg.

CHUNG-Yang, Huang., CHAO-YUE, Lai., KWANG-TING, Cheng. Electronic Design Automation. Capítulo 4. 2009.

DE BERG, M; CHEOG, O; VAN KREVELD, M; OVERMARS, M. Computational Geometry: Algorithms and Applications. 3. ed. Springer-Verlag Santa Clara, 2008.

DUCHOŇ, František et al. Path planning with modified a star algorithm for a mobile robot. Procedia Engineering, v. 96, p. 59-69, 2014.

DUDEK, G. and Jenkin, M. Computational Principles of Mobile Robo-

tics.Cambridge University Press, Cambridge, UK. 2000

FLANAGAN, C. The Bresenham Line-Drawing Algorithm. Disponível em: https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html. Acesso em: 5 março 2019.

GHOSH, S. Visibility Algorithms in the plane. Cambridge: Cambridge University Press, 2007

HAVRAN, Vlastimil. Heuristic ray shooting algorithms. 2000. Tese de Doutorado. Ph. d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague.

JAISWAL, Percy. Let's be A* — Learn and Code a Path Planning algorithm to fly a Drone. Disponível em: https://towardsdatascience.com/lets-be-a-learn-and-code-a-path-planning-algorithm-to-fly-a-drone-4d5b566fa1ae. Accesso em: 18 de outubro de 2019.

Klančar, Gregor. Škrjanc, Igor. Wheeled Mobile Robotics. 2017

LAVALLE S. PLANNING ALGORITHMS. Cambridge: Cambridge University Press, 2006.

LAVALLE, Steven M. KUFFNER, James J. Rapidly-Exploring Random Trees: Progress and Prospects, 2000.

'LUMELSKY, VLADIMIR J. and STEPANOV, ALEXANDER A. . Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, Vol. AC-31, No. 11, NOV. 1986.

MISA, Thomas J. An Interview with Edsger W. Dijkstra. Communications of the ACM, 2010.

O'ROURKE, Joseph. Art Gallery Theorems and Algorithms.Department of Computer Science, John Hopkins University, 1987.

PAIVA, Ely Carneiro de. Veículos robóticos semi-autônomos. 30 de setembro de 2002. Disponível em: http://www.bv.fapesp.br/pt/auxilios/3843/veiculosroboticos-semi-autonomos/. Acesso em: 22 de outubro de 2019

RUSSELL, Stuart J. NORVIG, Peter. Artificial Intelligence: A Modern Approach, 1995.

S. H. Tang, W. Khaksar, N. B. Ismail and M. K. A. Ariffin. A Review on Robot Motion Planning Approaches. Pertanika Journal of Science & Technology. Vol. 20, Jan 2012.

SIEGWART, R; R. NOURBAKHSH. Introduction to Autonomous Mobile Robots. Massachusetts: Bradford Company, 2004.

SWINGLER, Ashleigh. FERRARI, Silvia. A Cell Decomposition Approach to Cooperative Path Planning and Collision Avoidance via Disjunctive Programming, 2010.

ZANCHIN, B. ANÁLISE DO ALGORITMO A* (A ESTRELA) NO PLANEJAMENTO DE ROTAS DE VEÍCULOS AUTÔNOMOS. Disponível em:

http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/10221/1/PG_COELE_2018_1_03.pdf. Acesso em: 10 julho 2019.