ROBÔ MÓVEL AUTÔNOMO DE BUSCA

Roberto Müller Marinheiro (IC) e Calebe de Paula Bianchini (Orientador)

Apoio: PIVITI Mackenzie

RESUMO

A tecnologia tomou o mundo, esta é e sempre foi uma ferramenta essencial para a construção de um futuro mais próspero, seguro e justo, a automação nos meios de produção e locomoção é uma grande oportunidade de evolução dos seres humanos. Os carros modernos possuem diversos sistema inteligentes para melhorar a experiência de seus motoristas como sistemas de suspensão, carros com GPS embutido e conexão à internet já são comuns no mercado, o próximo passo, que inclusive, já está sendo dado, são os veículos autônomos, carros que se locomovem de uma localização a outra autonomamente, sem a necessidade de qualquer tipo de intervenção do “motorista”, que no caso, passa a ser apenas um passageiro. Em um certo momento, apenas veículos autônomos estarão nas ruas, isto nos trará uma série de benefícios, principalmente na segurança do trânsito, impedindo acidentes, fazendo as viagens menos estressantes, uma vez que estes veículos podem prever e impedir defeitos, e as falhas humanas serão extintas. Além destes benefícios, também podemos mencionar que a implantação dos veículos autônomos acarretará numa diminuição drástica do trânsito de automóveis, consequência de a locomoção dos veículos autônomos ter velocidade comum (não haverá diferenças de velocidade entre dois veículos em um mesmo ponto) em conjunto com os avanços na segurança do trânsito. O objetivo do presente artigo é apresentar uma análise de algoritmos e heurísticas para a criação da inteligência destes veículos, uma vez que existem diversos problemas complexos na concepção e no desenvolvimento dos mesmos.

Palavras-chave: Veículos Autônomos. Visibilidade. Planejamento de Rota.

ABSTRACT

Technology has taken over the world, this is and always has been an essential tool for building a more prosperous, secure and fair future, automation in the means of production and locomotion is a great opportunity for human evolution. Modern cars have several intelligent systems to improve the experience of their drivers such as suspension systems, cars with built-in GPS and internet connection are already common in the market, the next step, which is already being taken, are autonomous vehicles, cars that move from one location to another autonomously, without the need for any kind of intervention by the “driver”, which in this case becomes just a passenger. At a time, only autonomous vehicles will be on the streets, this will bring us many benefits, especially in traffic safety, preventing accidents, making travel less stressful, as these vehicles can predict and prevent defects, and human failures. will be extinguished. In addition to these benefits, we can also mention that the deployment of autonomous vehicles will result in a drastic reduction in car traffic, as a consequence of the autonomous vehicles locomotion having common speed (there will be no speed differences between two vehicles at the same point) together with the advances in traffic safety. The aim of this paper is to present an analysis of algorithms and heuristics for the intelligence creation of these vehicles, since there are several complex problems in their conception and development.

Keywords: Autonomous Vehicles. Visibility. Path Planning.

1. INTRODUÇÃO

Os veículos autônomos trarão uma evolução fantástica para nossa sociedade, é necessário que este tema seja dia após dia mais pesquisado e verificado, juntamente com seus problemas complexos para que, seja possível, avançar ainda mais neste tema, e o fato de já existirem veículos autônomos sendo distribuídos ao redor do globo só reforça esta necessidade.

Destes veículos autônomos, tomamos como objeto de pesquisa os robôs móveis de busca, ou muitas vezes nomeados como robôs móveis de inspeção. Tais robôs tem a função de buscar por um ou mais objetos em um ambiente, este ambiente pode ser conhecido, parcialmente conhecido ou totalmente desconhecido pelos robôs, aspecto que transforma por completo os tipos de problema enfrentados no decorrer do desenvolvimento dos mesmos, devido ás tarefas básicas de movimento e percepção são dificultadas.

As duas tarefas mais complexas no desenvolvimento destes robôs são a implementação da inteligência do robô, ou seja, suas decisões, e o desenvolvimento do sistema de controle, na tentativa de manter o mundo digital (o que o robô “sabe”) concordando com o mundo físico, utilizando os sensores e toda a parte de “percepção de mundo físico”.

No primeiro desafio, ou seja, na parte da inteligência, é necessária a implementação de algoritmos eficientes e qualificados para cada tarefa, pois o processamento, quando tratando de um robô móvel, é essencial para seu desempenho, não só pela energia utilizada, mas também pelo fato de que a cada segundo “perdido” no processamento de algum método, o robô pode perder algum tipo de informação e isto pode causar uma grande falha em todo o sistema.

Vale mencionar também que as placas utilizadas para a criação destes robôs não são tão potentes, devido a diferença de custo entre placas de alto processamento e de baixo processamento.

Já no segundo desafio, é essencial criar formas de combinar os mundos digital e físico, de tal maneira que o robô “saiba” onde ele está localizado e não se perca no decorrer de um percurso, e é claro, mesmo que a inteligência do robô seja funcional e eficiente, sem um sistema de controle adequado ela se torna inútil, impedindo que o robô faça qualquer tarefa.

Este projeto de pesquisa tem como objetivo estruturar e desenvolver a inteligência de um robô móvel de busca juntamente com uma solução para o sistema de controle do mesmo, submetendo uma proposta com implementações de diversos algoritmos selecionados e modificados para este propósito.

2. REFERENCIAL TEÓRICO

2.0 NAVEGAÇÃO

Um dos grandes desafios na área dos robôs móveis é a navegação. Para uma boa navegação são necessários quatro elementos básicos: percepção, o robô deve interpretar seus sensores, extraindo os dados; localização, o robô deve determinar sua posição no ambiente; cognição, o robô deve decidir como agir para atingir seus objetivos; controle de movimento, o robô deve ter bom controle sobre seus motores para percorrer as rotas definidas (SIEGWART, R e R. NOURBAKHSH, 2004).

Os grandes problemas da navegação são consequências de ruídos e/ou erros nos sensores responsáveis pela percepção dos robôs, uma vez que estes são limitados e não exatos. Tudo em ambientes não controlados podem causar complicações na percepção, devido a infinidade de variáveis existentes nos mesmos.

Assim como seus sensores, outros componentes dos robôs móveis podem sofrer falhas, como por exemplo as rotações das rodas, teremos sempre: uma porcentagem de erro entre a soma de rotações registrada e a rotação realizada; uma porcentagem de erro entre as rotações propriamente ditas, entre a (s) roda (s) de uma lateral e a (s) roda (s) de outra, e uma porcentagem de erro causado por deslizamentos das rodas (SIEGWART, R e R. NOURBAKHSH, 2004).

Para realizar a navegação de maneira adequada, são desenvolvidos diversos modelos de estimativa de erros que são aplicados diretamente no sistema de controle do robô, verificando constantemente o funcionamento dos quatro elementos básicos para a navegação.

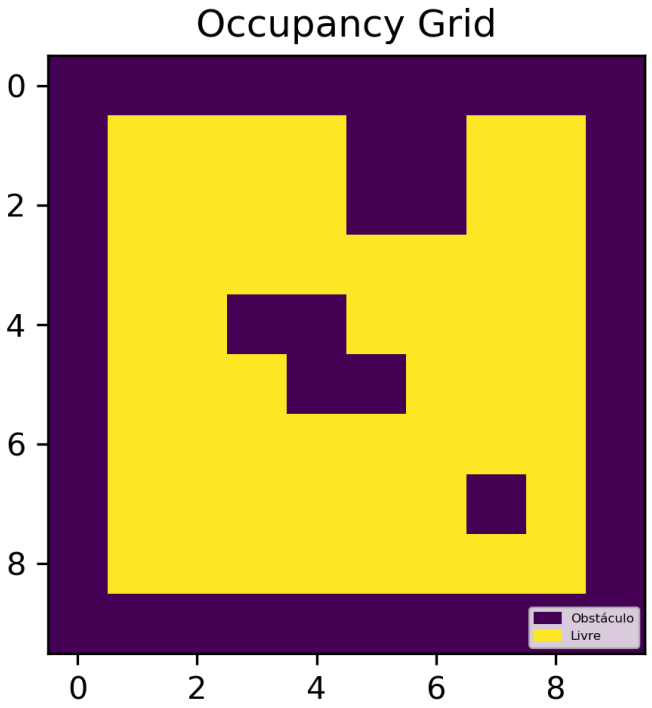
2.1 MAPEAMENTO

A primeira definição necessária para um robô ter a habilidade de se locomover em um ambiente é o mapeamento, que combinada à localização, gera uma certa “percepção” do mundo real. Existem diversas maneiras de fazer um mapeamento, nesta seção trataremos das mais comuns.

2.1.1 OCCUPANCY GRID

Neste tipo de representação, o mapa é definido por uma matriz bidimensional que abstrai o mundo real para o mundo digital em cada uma de suas coordenadas. Uma das abordagens mais comuns é a utilização de 0’s em pontos livres, e 1’s em obstáculos/paredes.

Figura 1. Simples representação de mapa utilizando occupancy grid, no caso, um binário, uma vez que temos apenas pontos livres (amarelo) e obstáculos (roxo).

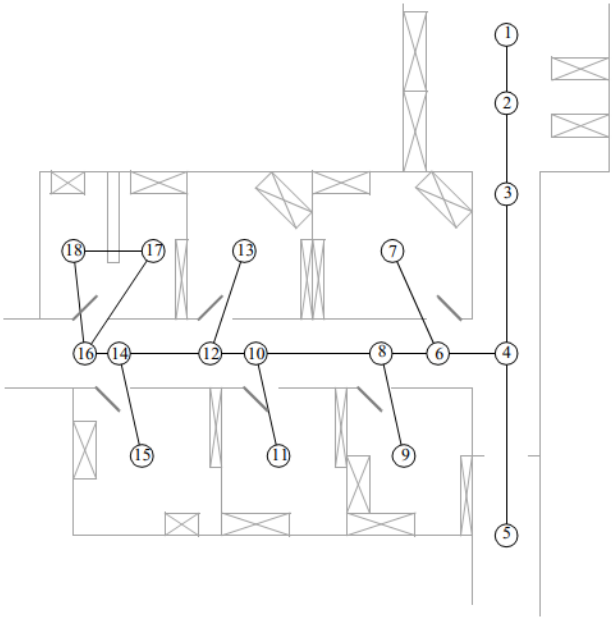


Fonte: Autor.

2.1.2 GRAFOS

O ambiente também pode ser representado por um grafo, que é essencialmente, um conjunto de nós que possuem ligações entre si, tais ligações indicam os caminhos possíveis de um nó *S* a um nó *G*.

Figura 2. Mapa representado por grafos.

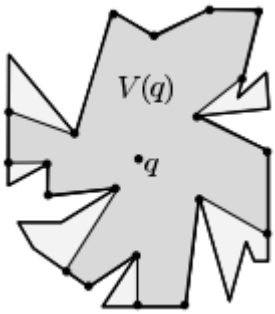


Fonte: (SIEGWART, R e R. NOURBAKHSH, 2004).

2.2 VISIBILIDADE

A determinação de visibilidade é uma área importante da computação gráfica muito utilizada no desenvolvimento e na robótica. Possui como função determinar quais áreas são visíveis por um observador em um ambiente. Para possibilitar essa determinação, é utilizada a própria definição do fenômeno visibilidade, que consiste em: 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.

Figura 3. Visibilidade de um ponto *q* em um polígono simples, representada pela cor cinza escuro, área *V(q)*, e a parte não visível representada pela cor cinza claro.



Fonte: (GHOSH, 2007).

2.2.1 O PROBLEMA DA GALERIA DE ARTE

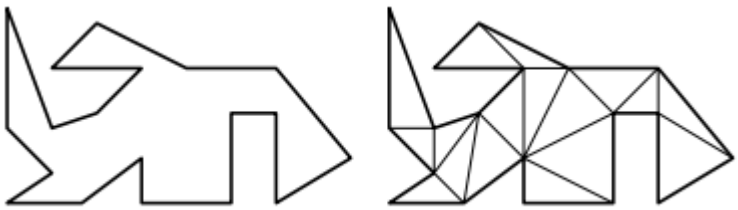
Um dos problemas de visibilidade mais conhecidos no mundo, o problema da galeria de arte tem como origem o problema real de se vigiar toda a área de uma galeria de arte com o menor número de guardas possível.

O problema tem diversas variações, mas no problema original, a galeria é tratada como um polígono simples: uma forma geométrica delimitada por segmentos de linhas retas que não cruzam umas às outras - não tem paredes internas - e que não contém “buracos” (polígonos dentro de polígonos) e os guardas tem campo de visão de 360º.

O teorema da galeria de arte gira em torno de três métodos: triangularização, 3-coloração e seleção de vértices, descritos mais detalhadamente.

A triangularização é um tipo de decomposição que divide o polígono em *N* triângulos dentro do mesmo. De acordo com o teorema da triangularização, todo polígono pode ser triangularizado, então a primeira etapa do teorema da galeria de arte é triangularizar a área da galeria.

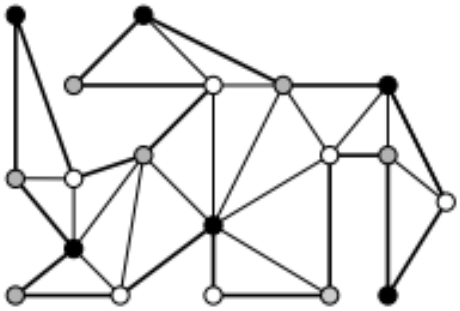
Figura 4. Triangularização de polígonos.



Fonte: (DE BERG, CHEOG, VAN KREVELD e OVERMARS, 2008).

A partir dos triângulos é possível aplicar a técnica de 3-coloração, que consiste em utilizar três cores para colorir cada um dos vértices dos triângulos, de tal modo que nenhum triângulo contenha vértices de cores iguais.

Figura 5. 3-coloração de um polígono triangularizado.



Fonte: (DE BERG, CHEOG, VAN KREVELD e OVERMARS, 2008).

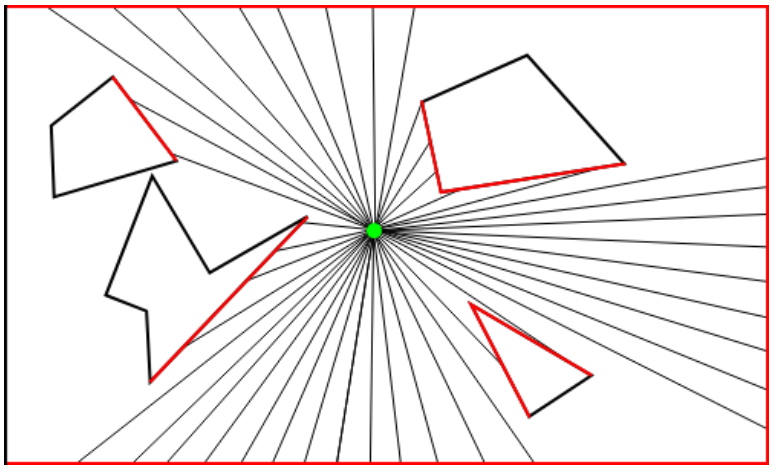
Para finalizar, os vértices da cor que menos apareceu em toda a coloração são selecionados, e estes vértices são os pontos de guarda da galeria. Conforme a figura 4, os guardas seriam posicionados nos vértices de cor branca ou de cor preta, uma vez que ambas as cores aparecem com a mesma quantidade: 6, enquanto existem 7 vértices com a cor cinza.

2.2.2 RAY CASTING

Uma abordagem para a determinação de visibilidade é o Ray Casting, 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, ou seja, para todas as direções, sendo que, os raios continuam a avançar até encontrar obstáculos.

Figura 6. Simulação do algoritmo Ray Casting.



Fonte: https://legends2k.github.io/2d-fov/design.html

2.3 PLANEJAMENTO DE ROTA

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 S e uma configuração objetivo G, evitando colisão com obstáculos conhecidos. 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.

Nesta seção, serão descritos métodos diversos de planejamento de rotas que foram estudados.

2.3.1 BUG

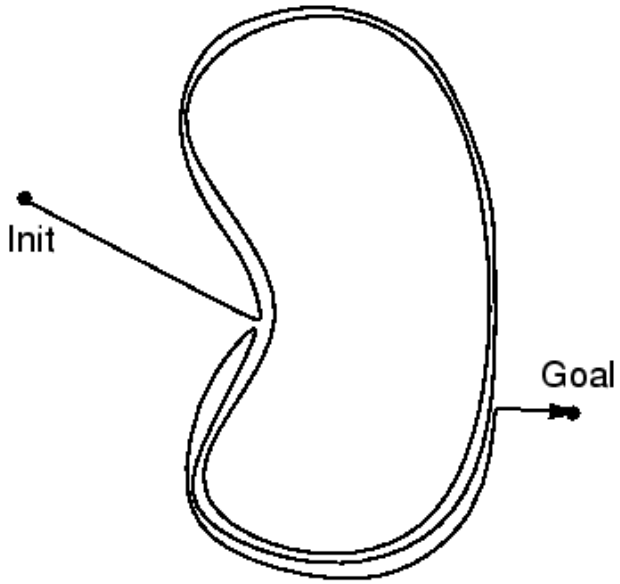
Na classe mais simples das soluções de planejamento de rotas, se encontram os algoritmos do tipo Bug (inseto) que, como o nome sugere, possui heurísticas pouco complexas e sem inteligência.

2.2.1.1 BUG 1

O primeiro dos algoritmos Bug é o Bug 1, que se movimenta em direção ao objetivo, e ao encontrar um obstáculo, faz-se o contorno do mesmo até retornar ao ponto de início, a partir deste segue a extensão do obstáculo até chegar ao ponto mais próximo do objetivo, e

neste ponto, se movimenta diretamente para o objetivo. Se existir outro obstáculo no caminho, o mesmo procedimento é executado.

Figura 7. Simulação do algoritmo Bug 1 (início = *Init*, objetivo = *Goal*).

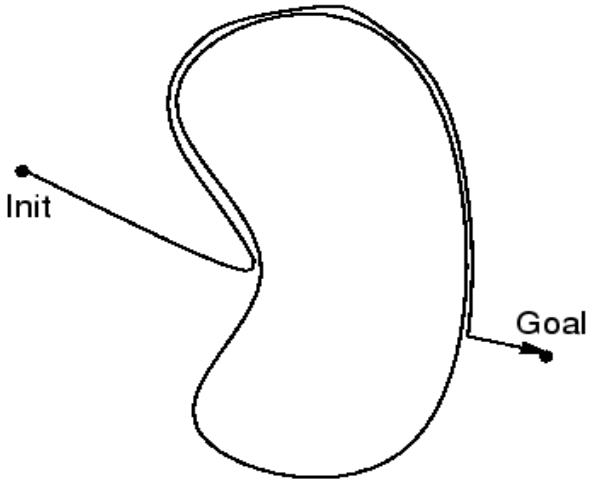


Fonte: (LAVALLE, 2006).

2.3.1.2 BUG 2

No algoritmo Bug 2, o movimento sempre é baseado na tentativa de seguir a extensão de um segmento de reta direto do ponto inicial ao objetivo. Se um obstáculo é encontrado, a extensão do mesmo é percorrida (para um dos lados direito e esquerdo, sendo definido pela implementação) até encontrar o segmento de reta direto, e quando o encontra, segue até chegar ao objetivo.

Figura 8. Simulação do algoritmo Bug 2 (início = Init, objetivo = Goal).



Fonte: (LAVALLE, 2006).

2.3.2 PLANEJAMENTO DE ROTAS DE CUSTO MÍNIMO

Nesta seção, serão apresentados algoritmos de planejamento de 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 Occupancy Grids.

2.3.2.1 DIJKSTRA

Um dos clássicos da computação, o algoritmo de caminhos mínimos de Dijkstra, calculando os caminhos mínimos do nó inicial para todos os demais nós.

Inicialmente, todos os vértices do grafo têm uma distância infinita, com exceção do vértice inicial que tem, intuitivamente, distância zero. Todos os vértices são também representados como “abertos”, pois, ainda não foi encontrado um caminho de distância mínima do ponto inicial até cada um deles.

Posteriormente, o vértice com menor distância em relação aos outros e que se encontra em “aberto” é selecionado, então este é “fechado” e são recalculados a partir dele, todas as distâncias aos vértices ainda “abertos”, simplesmente fazendo a soma da distância do vértice selecionado as distâncias dos outros vértices vizinhos. Nos casos das novas distâncias calculadas serem inferiores às atuais, faz-se a substituição destes valores.

De forma análoga, o próximo passo é novamente selecionar o vértice ainda “aberto” com a menor distância, fechá-lo, e a partir deste, recalcular as distâncias. Repete-se o último passo até que todos os vértices tenham sido “fechados”. Como resultado, obtemos os caminhos mínimos entre o vértice inicial e os outros vértices.

2.3.1 A*

Um dos algoritmos mais conhecidos no universo da computação é o A*, ou A Star, um algoritmo considerado como inteligência artificial que calcula caminhos mínimos entre dois pontos.

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).

3. METODOLOGIA

A placa escolhida para a aplicação dos testes do robô foi a placa open-source Arduino, devido a sua acessibilidade e sua considerável baixa complexidade, se comparada a outros tipos de placas. Esta escolha pode acarreta em algumas limitações tanto de memória como de processamento, em razão do Arduino não ser uma placa de processamento, mas sim um micro controlador.

A linguagem de programação selecionada para a implementação dos algoritmos selecionados foi a linguagem C, pelo fato da mesma trabalhar com Arduino, ser veloz e dinâmica, pois como necessitamos de eficiência e de velocidade, tal linguagem faz sentido e se encaixa perfeitamente no problema.

Primeiramente, o mapa foi definido como uma matriz bidimensional (occupancy grid) em um simples arquivo de texto que seria armazenado na memória do robô, juntamente com seu ponto inicial. Tal mapa é lido e armazenado num vetor contido numa estrutura nomeada *Mapa*.

Figura 9. Arquivo de entrada definindo o mapa, primeira linha: largura; segunda linha: comprimento; seguintes linhas: pontos do mapa e última linha: posição inicial do robô.

```
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
```

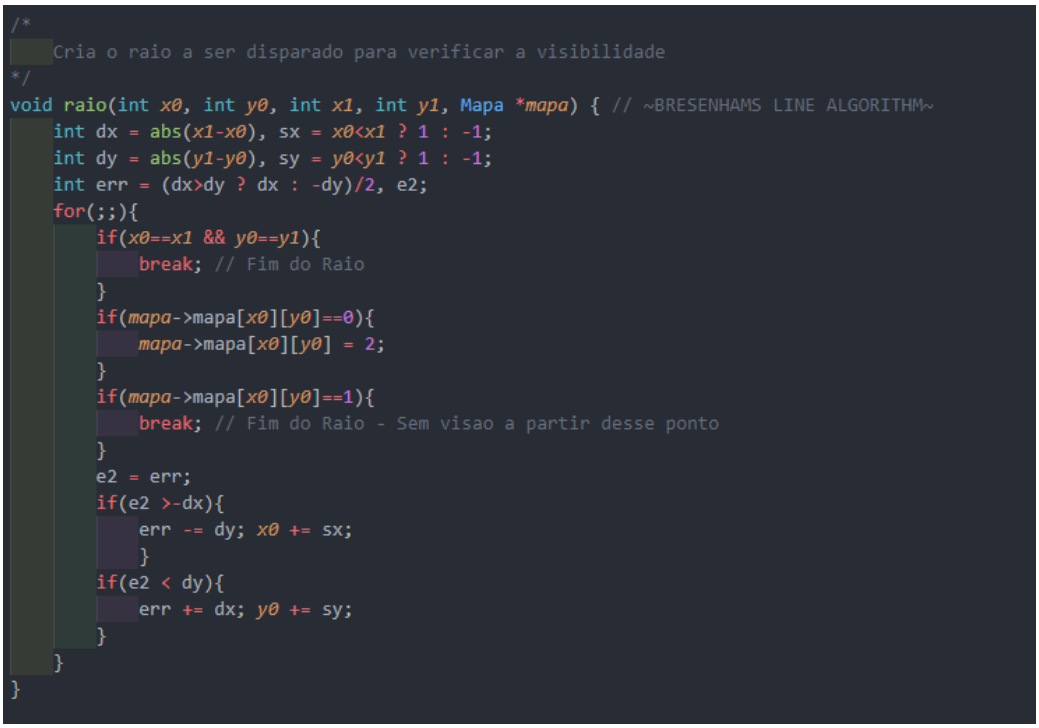
Fonte: Autor.

Uma vez que o mapa está definido, o grande problema para o robô é encontrar um objeto especificado, seria simples visitar todos os pontos do mapa e verificar cada um deles, mas isto de forma alguma seria uma inteligência, mas sim um método de força bruta.

Portanto, o teorema da galeria de arte foi estudado, pois este define pontos de guarda para obtenção de total visibilidade, mas este não foi capaz de satisfazer os propósitos do robô devido às limitações da galeria, que necessariamente deveria ser definida por um polígono simples e sem buracos (polígonos dentro de outro polígono), o que não condiz com a necessidade, pois o objetivo é que o robô seja capaz de definir os pontos em qualquer tipo de mapa.

Como cálculo de visibilidade, foi utilizado o algoritmo de Ray Casting, que funciona disparando raios para todas as direções do mapa e alterando seus valores. Como a representação do mapa é feita com uma matriz, foi necessária a implementação do algoritmo de Bresenham para criação dos raios, possibilitando determinar quais os pontos numa matriz de base quadriculada estão contidos em cada raio criado (FLANAGAN, 2007).

Figura 10. Ilustração da implementação do algoritmo de Bresenham.



```
/*
    Cria o raio a ser disparado para verificar a visibilidade
*/
void raio(int x0, int y0, int x1, int y1, Mapa *mapa) { // ~BRESENHAMS LINE ALGORITHM~
    int dx = abs(x1-x0), sx = x0<x1 ? 1 : -1;
    int dy = abs(y1-y0), sy = y0<y1 ? 1 : -1;
    int err = (dx>dy ? dx : -dy)/2, e2;
    for(;;){
        if(x0==x1 && y0==y1){
            break; // Fim do Raio
        }
        if(mapa->mapa[x0][y0]==0){
            mapa->mapa[x0][y0] = 2;
        }
        if(mapa->mapa[x0][y0]==1){
            break; // Fim do Raio - Sem visao a partir desse ponto
        }
        e2 = err;
        if(e2 >-dx){
            err -= dy; x0 += sx;
            }
        if(e2 < dy){
            err += dx; y0 += sy;
        }
    }
}
```

Fonte: Autor.

Uma vez que é possível calcular a visibilidade de qualquer ponto no mapa, é necessário definir os pontos de guarda. Para tanto, foi criada uma heurística própria que funciona da seguinte maneira:

I. Faz-se o cálculo de visibilidade do ponto inicial do robô, configurando os pontos visíveis como “visível”, e o adiciona na lista de pontos de guarda.

II. Enquanto houver algum ponto no mapa que não estiver configurado como “visível”: busca por um ponto não visível e o adiciona na lista de guardas.

III. Retorna.

Nesta implementação, é utilizada a estrutura “*visibilidade”*, responsável por armazenar os pontos de guarda num vetor nomeada por “*pontos”*, pela variável de controle “*completoVisivel”* que indica se o mapa está completamente visível ou não e a *quantidade* de pontos necessárias para total visibilidade.

Figura 11. Implementação da função de processamento de visibilidade.

```
int processamentoVisibilidade(Mapa *mapa, Visibilidade *visibilidade){
    int i, j;
    Ponto proximoPonto = mapa->inicio;
    while(visibilidade->completoVisivel==0){
        visibilidade->pontos[visibilidade->quantidade] = proximoPonto;
        visibilidade->quantidade++;
        // DOIS LOOPS: UM PARA ATIRAR NO SENTIDO VERTICAL, E OUTRO NO HORIZONTAL
        for(i=0; i<=mapa->largura; i++){ // SENTIDO VERTICAL
            raio(proximoPonto.x, proximoPonto.y, 0, i, mapa); // CIMA -> 0, I (ITERAR ATE LARGURA)
            raio(proximoPonto.x, proximoPonto.y, mapa->altura, i, mapa); // BAIXO -> ALTURA, I (ITERAR ATE LARGURA)
        }
        for(j=0; j<=mapa->altura; j++){ // SENTIDO HORIZONTAL
            raio(proximoPonto.x, proximoPonto.y, j, mapa->largura, mapa); // DIREITA -> J, LARGURA (ITERAR ATE ATLURA)
            raio(proximoPonto.x, proximoPonto.y, j, 0, mapa); // ESQUERDA -> J, 0 (ITERAR ATE ALTURA)
        }
        proximoPonto = catchNext(mapa);
        if(proximoPonto.x == -1) visibilidade->completoVisivel = 1;
    }
    return 0;
}
```

Fonte: Autor.

Uma vez que o robô tem o conhecimento dos pontos necessários para obter total visibilidade do mapa, a próxima tarefa é a de planejar as rotas de maneira eficiente, para isto, foi implementado o algoritmo A*, planejando rotas mínimas de forma rápida. Para tal, também foi essencial o desenvolvimento de uma estrutura de árvore binária máxima, conhecida como Max Heap, pois em muitos iterações do A* existe um método de selecionar o nó com menor custo *f,* e buscar por este nó a cada iteração perderia toda a eficiência do algoritmo. Portanto, a Max Heap faz com que sua lista sempre esteja ordenada, limitando todo aquele processamento de busca à uma simples tarefa de remoção da Max Heap.

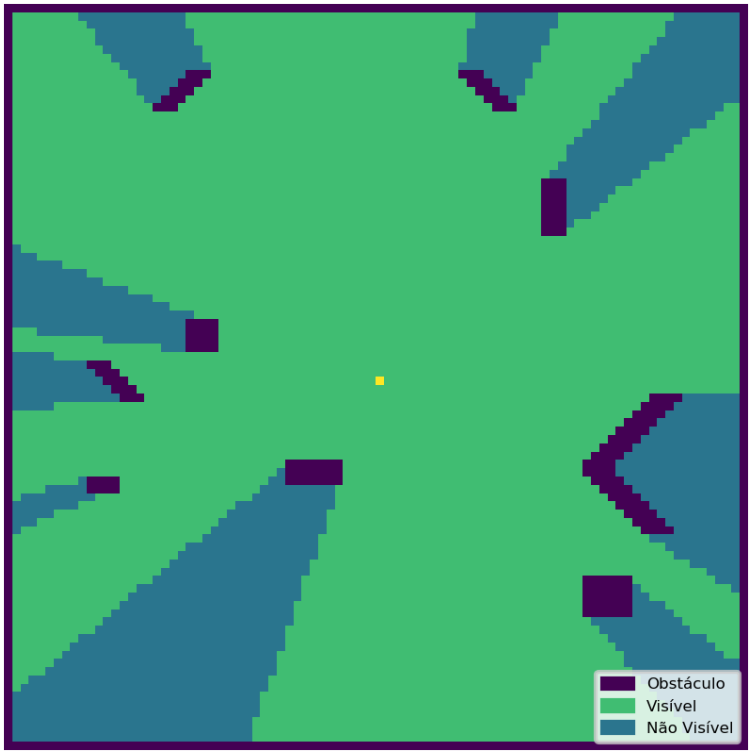
O planejamento com o algoritmo A* é eficiente, porém, também foi verificado que era necessário executar o planejamento do ponto inicial até todos os outros pontos de guarda, para verificar o caminho mais curto/o ponto de guarda mais próximo (em relação a distância do caminho deste e não da distância exata entre os pontos) do mesmo, e depois partir deste ponto mais próximo ao ponto mais próximo deste, etc. Como solução deste problema, foi implementada uma função, *initListaPath()*, de geração de caminhos mínimos, que tem como resultado um caminho que passa por todos os pontos de guarda.

4. RESULTADO E DISCUSSÃO

Todo o código implementado é bem estruturado, organizado, e acima de tudo eficiente, contando com uma série de estruturas que auxiliam na execução das tarefas e métodos.

Os algoritmos selecionados para visibilidade, combinação de Ray Shooting com Bresenham’s Line se mostraram excepcionais em calcular a visibilidade, mostrando pouquíssimas regiões com falhas.

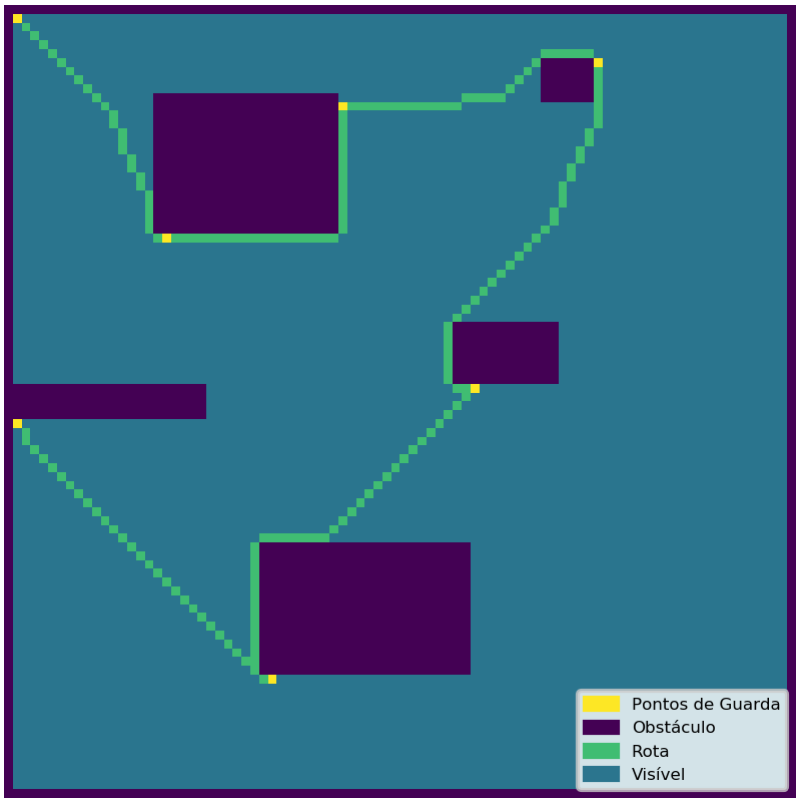
Figura 12. Resultado da implementação da função de processamento de visibilidade.



Fonte: Autor.

A soma dos algoritmos de visibilidade, definição de guardas, A* e a função *initListaPath()* se provaram uma solução viável para a inteligência do robô móvel de busca.

Figura 13: resultado da implementação da inteligência de planejamento do robô.



Fonte: Autor.

Em contrapartida destes bons resultados, os problemas de implementação enfrentados e o tempo utilizado no desenvolvimento da inteligência do robô impossibilitaram a implementação da inteligência no Arduino e a realização dos testes online, o que, por consequência, fez com que a parte do sistema de controle do robô não fosse verificada.

5. CONSIDERAÇÕES FINAIS

No término deste projeto, foi verificado que a inteligência desenvolvida é válida para robôs móveis de busca, como trabalho futuro, é necessário verificar se a inteligência apresentada será válida quando testada no Arduino e, obviamente, desenvolver melhor o robô e seu sistema de controle.

6. REFERÊNCIAS

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

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

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

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

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.

Contatos: robertom@marinheiro.com.br e calebe.bianchini@mackenzie.br.