Occupancy Grid

M.T. Tristão
Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Belo Horizonte, Brasil
marcotuliopin@ufmg.br

I. Introdução

Neste trabalho, implementamos o algoritmo de Mapeamento de Grade de Ocupação utilizando um robô com sensores de distância. Os programas foram criados com base nos materiais da disciplina e em literatura especializada. Para testar o algoritmo, utilizamos dois diferentes cenários, cada um apresentando um nível distinto de complexidade - um estático e outro dinâmico.

O algoritmo de Mapeamento de Grade de Ocupação implementado utiliza a decomposição em grid, onde o ambiente é dividido em células que são marcadas como ocupadas ou livres, com base nas leituras dos sensores do robô. O objetivo é construir um mapa do ambiente que pode ser usado para navegação e planejamento de trajetórias.

Este relatório organiza a documentação do trabalho. Cada tópico será detalhado em uma seção distinta, de acordo com o enunciado. Também serão apresentadas outras informações consideradas relevantes, como descrição de partes da implementação e como executar o código. As duas últimas seções compõem os comentários finais e as referências bibliográficas.

II. Execução

Abra o simulador CoppeliaSim na cena escolhida. Após isso, execute o notebook occupancy_grid.ipynb até a célula "Execution" e a simulação deve começar a executar. Caso não ocorra o início automático, clique no botão de iniciar a execução no simulador.

III. Occupancy Grid

O algoritmo de Mapeamento de Grade de Ocupação é comumente usado em robótica para construir mapas do ambiente. Este algoritmo usa leituras de sensores para estimar a ocupação de cada célula em uma grade que representa o ambiente. Esse algoritmo foi extraído do livro "Probabilistic Robotics" [1], e seu funcionamento está descrito nas figuras 1 e 2, também retiradas da mesma referência.

A função $occupancy\_grid\_mapping$ atualiza a grade de ocupação com base nas leituras atuais do sensor. Ela recebe como entrada a grade de ocupação anterior, o conjunto de células no campo perceptivo do robô, a pose atual do robô, as leituras atuais do sensor, os ângulos dos sensores, o tamanho da grade e o tamanho de cada célula na grade. A função retorna a grade de ocupação atualizada.

Algorithm occupancy_grid_mapping($\{l_{t-1,i}\}, x_t, z_t$)**:**
  for all cells $\mathbf{m}_i$ do
    if $\mathbf{m}_i$ in perceptual field of z_t then
      $l_{t,i} = l_{t-1,i} + \mathbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) - l_0$
    else
      $l_{t,i} = l_{t-1,i}$
    endif
  endfor
  return $\{l_{t,i}\}$

Fig. 1. Algoritmo de Occupancy Grid.

Algorithm inverse_range_sensor_model(i, x_t, z_t)**:**
  Let x_i, y_i be the center-of-mass of $\mathbf{m}_i$
  $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
  $\phi = \text{atan2}(y_i - y, x_i - x) - \theta$
  $k = \text{argmin}_j \, |\phi - \theta_{j,\text{sens}}|$
  if $r > \min(z_{\max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{k,\text{sens}}| > \beta/2$ then
    return l_0
  if $z_t^k < z_{\max}$ and $|r - z_{\max}| < \alpha/2$
    return l_{occ}
  if $r \leq z_t^k$
    return l_{free}
  endif

Fig. 2. Função utilizada no Occupancy Grid.

Para cada célula no campo perceptivo, a função calcula o novo valor de log-odds para a célula usando a função $inverse\_range\_sensor\_model$ e atualiza o valor da célula na grade de ocupação.

A função $inverse\_range\_sensor\_model$ calcula o modelo de sensor de alcance inverso para uma célula. Ela recebe como entrada os índices da célula, a pose atual do robô, as leituras atuais do sensor (z), os ângulos dos sensores, o tamanho da grade e o tamanho de cada célula na grade. A função retorna o valor de log-odds para a célula.

IV. Navegação

A estratégia de navegação usada no código envolve fazer o robô girar 360º para obter uma visão completa do ambiente e, em seguida, mover-se em direção ao próximo ponto na lista de destinos. Quando o robô chega a um destino, ele gira novamente 360º antes de prosseguir para o próximo destino. O giro serve para que o robô capte o máximo de informações possíveis da posição antes de avançar para a seguinte.

O código mantém uma fila de pontos de destino que o robô deve visitar. A cada iteração do loop principal, o código verifica se o robô chegou ao destino atual. Se sim, ele remove o destino da fila e faz o robô girar 360º para atualizar o mapa do ambiente antes de prosseguir para o próximo destino.

A navegação entre pontos utiliza o algoritmo de campos potenciais desenvolvido previamente na disciplina. Note que o robô continua captando informações e atualizando o mapa conforme navega para o alvo.

V. Discretização da Cena

Criamos um grid de tamanho 10x10 com células de tamanho variável de acordo com o teste que fizemos. Para obter o índice da célula em uma grade para um determinado ponto, usamos a fórmula: $\lceil (1.0/cell\_size) * x \rceil - 1$, e aplicamos a mesma fórmula à componente y. Essa fórmula divide a coordenada do ponto pelo tamanho da célula, arredonda para cima e subtrai 1. Por fim, adicionamos metade do tamanho da grade às coordenadas x e y.

Para encontramos as coordenadas de um ponto a partir do índice de uma célula subtraímos metade do tamanho da grade das coordenadas x e y. Finalmente, multiplica as coordenadas x e y pelo tamanho da célula para obter as coordenadas finais do ponto. Dessa forma, cada célula terá somente um ponto de correspondência no mapa. Portanto, discretizar o mapa causa perda da capacidade de especificar um ponto dentro de uma célula.

Por fim, para encontramos as células cruzadas por um laser, implementamos o algoritmo definido em [2]. O loop principal do algoritmo é exposto na figura 4. Perceba que esse algoritmo, diferente da Linha de Bresenham, detecta todas as células cruzadas pelo laser. Uma demonstração está na figura 7.

VI. Testes

A. Tamanho da Célula

Mostramos o mapa resultante para três tamanhos distintos de células: 0.05, 0.1 e 0.5, respectivamente nas figuras 5, 6 e 8. Percebemos que o mapa com células de tamanho 0.5 não foi um bom resultado. Nele, não foi possível captar a forma dos obstáculos. Além disso, alguns obstáculos tiveram seu tamanho excessivamente superestimado, enquanto outros não foram captados. O mapa com células de tamanho 0.1 é um resultado melhor, pois já é bem representativo do cenário, porém não consegue captar seus detalhes como o mapa de células de tamanho 0.05, que detecta os contornos até mesmo dos menores obstáculos, como os pés das cadeiras.

B. Cena Dinâmica

Testamos o programa em uma cena dinâmico, com um agente que caminha pela cena além do robô. O algoritmo se provou resistente a esse cenário. Como muitas leituras são feitas consecutivamente, mesmo que algumas leituras detectem o agente caminhando por uma célula como um obstáculo, o agregado das chances em todas as leituras irá convergir para um veredito da célula estar livre. Por isso, não houve alteração no resultado do mapa criado pelo robô.

C. Começo em Posições Distintas

Testamos variar a posição de início do robô, para isso executamos o programa para duas posições iniciais distintas em cada mapa. Decidimos por não incluir visualizações adicionais para cada caminho porque não houve mudança no mapa final. Ou seja, o resultado independe da posição de início do robô. No entanto, mostramos nas figuras 9 e 10 os caminhos percorridos pelo robô. Perceba que o resultado deve logicamente ser o mesmo, uma vez que aproximadamente os mesmos pontos são visitados ao fim do processo.

D. Ruído

Como o robô consegue muitos dados da cena, tanto pelo trajeto definido, quanto pelo giro de 360º, entre outros motivos, os mapas gerados foram robustos aos ruídos testados. As figuras 5, 6 e 8 mostram mapas com ruído de 0 a tamanho da célula em cada leitura do laser, enquanto o mapa da figura 11 colocou ruído de 0 a 3 vezes o tamanho da célula em cada leitura. Ainda assim, os mapas mantiveram uma grande qualidade.

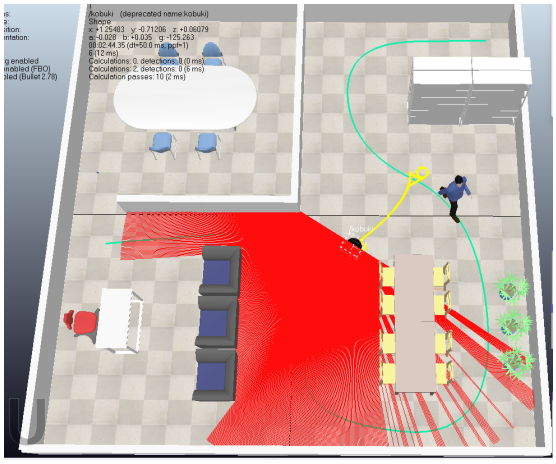

Fig. 3. Cenário usado para os testes.

```
loop{
    if(tMaxX < tMaxY) {
        tMaxX= tMaxX + tDeltaX;
        X= X + stepX;
    } else   {
        tMaxY= tMaxY + tDeltaY;
        Y= Y + stepY;
    }
    NextVoxel(X,Y);
}
```

Fig. 4. Algoritmo de detecção de células atravessadas pelo laser.

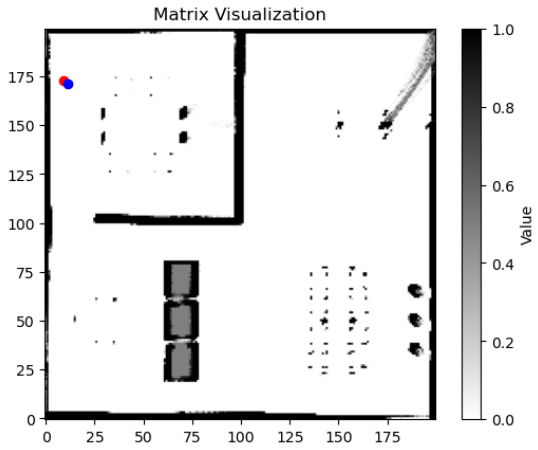

Fig. 5. Mapa com células de tamanho 0.05.

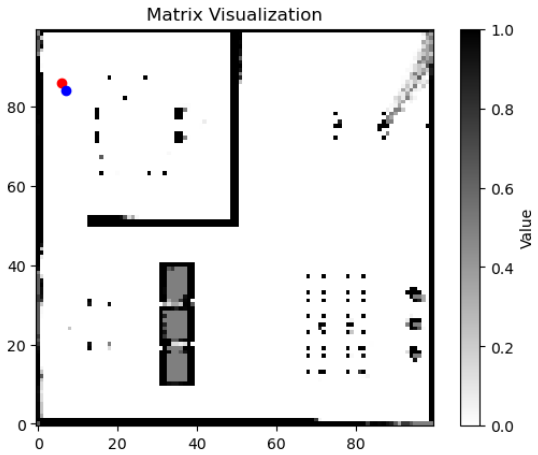

Fig. 6. Mapa com células de tamanho 0.1.

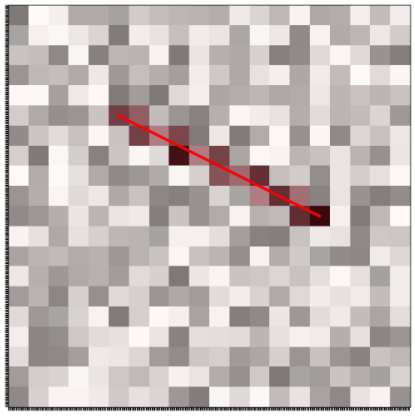

Fig. 7. Detecção de células cruzadas por um laser.

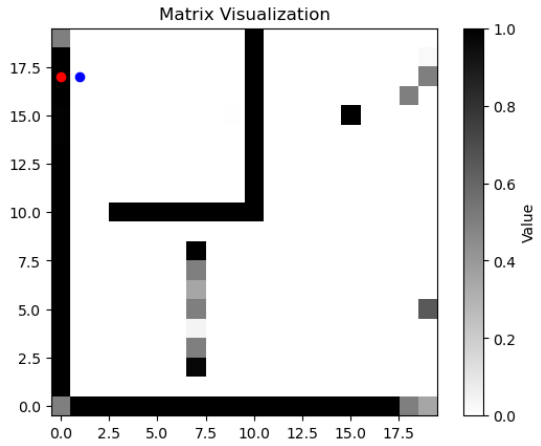

Fig. 8. Mapa com células de tamanho 0.5.

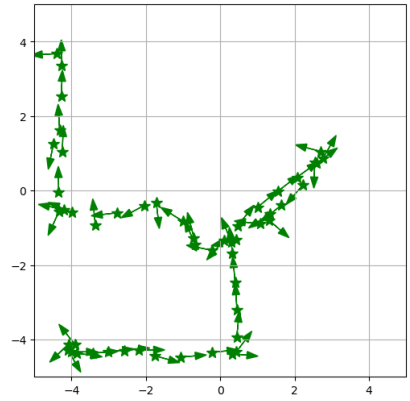

Fig. 9. Primeiro caminho percorrido pelo robô.

VII. Conclusões

O algoritmo foco do trabalho - Occupancy Grid - se mostrou muito eficaz em criar mapas tanto em regiões estáticas quanto em regiões esparsamente populadas. No entanto, a necessidade de discretizar o ambiente torna o processo custoso computacionalmente, principalmente para ambientes maiores. Além disso, se perde a capacidade de especificar rotas precisas para o robô, uma vez que cada célula corresponde a somente um ponto.

A respeito do processo de navegação, a abordagem de realizar uma rotação de 360º é interessante para captar o máximo de informação possível de um ambiente. Com isso, uma rota relativamente pequena conseguiu capturar o ambiente em quase sua totalidade. No entanto, a definição de um caminho fixo para o robô, apesar de funcionar para o escopo do trabalho (uma vez que se trata somente de mapeamento, e não SLAM), não é aplicável em demais cenários.

Por fim, apesar dos gráficos incrementais gerados não terem uma boa qualidade, provavelmente devido a uma falha no processo de gerar a visualização, os mapas gerados corresponderam satisfatoriamente ao ambiente em todos os cenários examinados.

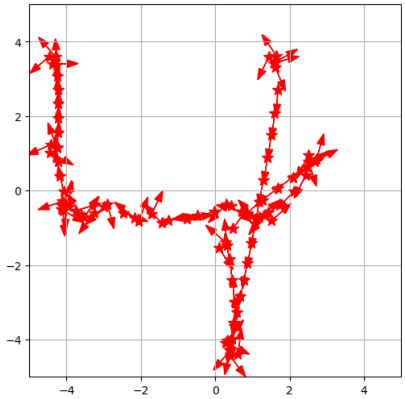

Fig. 10. Segundo caminho percorrido pelo robô.

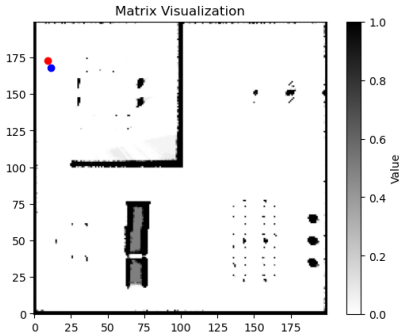

Fig. 11. Mapa com aumento do ruído.

REFERENCES

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

[2] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," *Proceedings of EuroGraphics*, vol. 87, 08 1987.

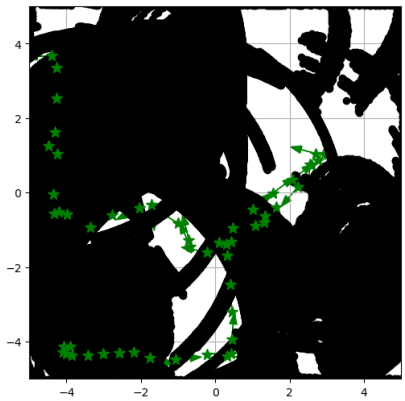

Fig. 12. Visualização incremental do laser.