Trabalho Prático 1

Arthur da Costa Vangasse * **Gabriel Gomes** **
João Felipe Ribeiro Baião ***

* *Departamento de Engenharia Elétrica, Universidade Federal de Minas Gerais, MG, (e-mail: vangasse@ufmg.br).*
** *Departamento de Engenharia Elétrica, Universidade Federal de Minas Gerais, MG, (e-mail: ggomes1015@ufmg.br).*
*** *Departamento de Mecânica, Universidade Federal de Minas Gerais, MG, (e-mail: baiaojfr@ufmg.br).*

Resumo:
Este trabalho apresenta o desenvolvimento do segundo trabalho prático da disciplina de Planejamento de Movimento de Robôs, do Programa de Pós Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais ministrada no semestre 2022.2. Neste, são utilizados os pacotes desenvolvidos para o robô móvel *Turtlebot3* em *ROS2* para simulação física e validação dos algoritmos implementados. Os algoritmos compreendidos foram escritos em *Python*, sendo eles: A*, *Probabilistic Roadmaps* (PRM) e *Rapidly Exploring Random Tree* (RRT).

1. MAPEAMENTO

O ambiente modelado representa uma área de $100m^2$ compreendendo três obstáculos cercados por um limite quadrado de $10m \times 10m$. Na Fig. 1, a imagem rotulada *Original* é extraída diretamente do modelo 3D exibindo o plano $x-y$ visto de cima. A imagem rotulada *Treated Map* é resultado de uma filtragem, estabelecendo obstáculos em preto e espaço navegável em branco, sucedido de um processo de aumento dos obstáculos em função do raio aproximado do robô, de $30cm$. Por fim, a imagem *Subsampled* é uma imagem de $53px \times 53px$ resultado de um processo de subamostragem visando reduzir a carga computacional dos algoritmos.

Figura 1. Construção do mapa.

2. ALGORITMO A*

O algoritmo A* consiste em uma estratégia de busca em grafos. Como entrada, o algoritmo recebe um grafo formado pelo nó de saída, objetivo e o *roadmap*. Esse grafo deve ser ponderado com as distâncias euclidianas respectivas a cada uma das arestas.

São criadas duas estruturas de armazenamento: a fila, que é um objeto da biblioteca *heapq* do *Python* hea (2022) e um dicionário de visitados - ambos contendo os nomes dos nós, um custo associado e o nome do nó "pai". Ademais, é decidida a heurística que permitirá ordenar os elementos adicionados à fila. A heurística escolhida é a distância euclidiana entre o nó referente e o objetivo. Dessa forma, calcula-se a estimativa do custo de seguir o caminho por determinado nó a partir de: *custo estimado do nó = heurística + custo para chegar ao nó*. Para o dicionário de visitados, o custo associado é apenas o custo para chegar ao nó.

A busca inicia-se pelo nó de partida, que é adicionado à fila (excepcionalmente para o nó de partida, o nó "pai"é o próprio nó de partida). Sendo o único nó na fila, o nó de partida é o primeiro a ser retirado e são adicionados os vizinhos à fila. Para cada iteração do algoritmo, o elemento de menor custo estimado é retirado da fila.

Ao ser retirado da fila, o nó é adicionado ao dicionário de visitados. Sempre adiciona-se como nó "pai", aquele que confere o menor custo associado, dessa forma, nós que foram visitados podem não aparecer no resultado final, caso seja encontrado um caminho menos custoso que não passe por eles.

Caso um nó vizinho já tenha sido adicionado à fila, compara-se o custo estimado armazenado e o atual. Caso o atual seja menor, significa que foi encontrado um caminho melhor para esse nó. Sendo assim, o armazenado anteriormente é substituído na fila.

A partir do momento que o nó de objetivo é adicionado aos visitados, a cada iteração é feita uma checagem se o caminho encontrado é o melhor possível. Para isso compara-se o custo desse caminho ao custo estimado do próximo nó da fila. Caso o caminho encontrado tenha custo menor, significa que este já é o melhor caminho, então, o algorítimo retorna a sequência de nós correspondente.

3. PROBABILISTIC ROADMAP METHOD (PRM)

O PRM é um algoritmo de planejamento na robótica, ele determina um caminho entre uma configuração inicial

⋆ Reconhecimento do suporte financeiro deve vir nesta nota de rodapé.

do robô e uma configuração final. É caracterizado por possuir duas fases: aprendizado e consulta. Na fase de aprendizado, um *roadmap* é construído através do espaço livre disponível para o robô se locomover. E a fase de consulta consiste em conectar as configurações de início e fim desejadas ao *roadmap*, para que um planejador local possa encontrar os caminhos livres entre essas duas configurações.

A construção do *roadmap* foi realizada através da distribuição uniforme de pontos no espaço de trabalho. O algoritmo gera um ponto aleatório e verifica se há colisão com algum obstáculo do espaço de trabalho, se houver, esse ponto é descartado e um novo ponto é gerado para verificação, caso não tenha colisão este ponto é adicionado ao *roadmap*. Esse procedimento é realizado até que uma determinada quantidade de pontos preestabelecida para o *roadmap* seja alcançada.

A próxima etapa foi conectar os pontos do *roadmap* que são próximos uns dos outros. Esse procedimento foi efetuado com auxílio da biblioteca *KDtree* do *Python* kdt (2022), que procura rapidamente os vizinhos mais próximos de qualquer ponto e sua distância. Os pontos gerados foram usados para a criação da *kd-tree*, que através de um ponto informado encontra quais são os seus pontos vizinhos mais próximos. Cada um dos pontos pertencentes ao *roadmap* foi utilizado para encontrar os seus vizinhos mais próximos e conectá-los através de uma linha reta. A linha foi traçada apenas entre o ponto e os vizinhos que não possuem obstáculos que bloqueiam o caminho entre eles.

A construção do *roadmap* foi realizada com assistência da biblioteca *networkx* net (2022), que armazena os pontos e suas conexões em forma de grafo, sendo que os pontos se tornam os nós do grafo e suas conexões as arestas ponderadas pelo valor da distância. A configuração inicial e final do robô são adicionadas ao *roadmap* no final, após verificar se existe colisão com os obstáculos e se a conexão com algum dos pontos está livre. A forma utilizada para encontrar o caminho entre a posição inicial e final foi através do Algoritmo A* 2.

A Figura 2 mostra o processo de criação do *roadmap* pelo PRM. É possível observar que o grafo criado não conecta todos os pontos por causa de obstáculos que atrapalham a formação dos caminhos, e isso ocasiona em pontos isolados. Esses pontos podem estar isolados de maneira individual ou com junto de outros pontos isolados formando um grafo desconexo.

4. RAPIDLY EXPLORING RANDOM TREE (RRT)

O algoritmo RRT, em termos gerais, consiste na geração de árvores que podem partir de configurações iniciais e finais do sistema, no intuito de explorar o ambiente e convergir em um caminho que ligue os dois pontos. O método utiliza uma série de ferramentas as quais serão expostas a seguir, esclarecendo o funcionamento do algoritmo. Ressalta-se que sendo caracterizado como *single query*, o algoritmo necessita ser executado a cada vez que um novo objetivo é definido, diferentemente do PRM. As etapas implementadas são:

- Sorteio de um ponto do mapa;

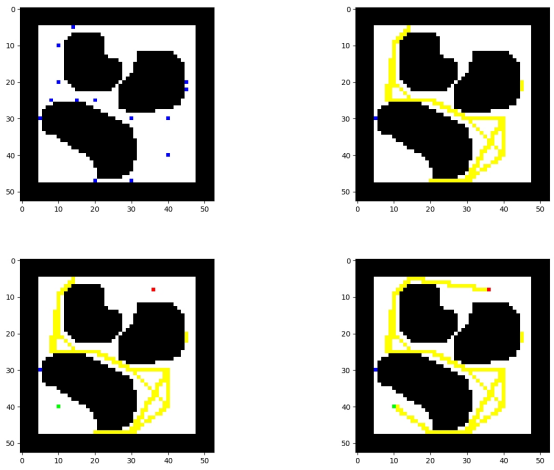

Figura 2. Caminho computado pelo PRM.

- Extensão da árvore em direção ao ponto sorteado;
- Tentativa de integração das árvores.

Em primeiro lugar, será abordada a implementação baseada na expansão de uma única árvore de raiz na configuração inicial do sistema e em seguida serão tratadas as adaptações necessárias para a implementação da expansão da segunda árvore.

4.1 Árvore Única

De início, um grafo $\boldsymbol{G}$ é representado por um único nó contendo a configuração inicial do sistema. A partir disto, o primeiro passo para implementação do algoritmo é o sorteio de um ponto no mapa $\overline{p}$, o qual indica a direção para onde a árvore deve crescer. O sorteio, foi realizado de maneira aleatória por uma distribuição uniforme limitada às dimensões do mapa sub-amostrado.

Em seguida, todo o grafo é percorrido à medida que a distância de cada um dos nós ao ponto sorteado $\overline{p}$ é inferida. Ao fim da consulta, o ponto mais próximo $p* \in \boldsymbol{G}$, com relação a $\overline{p}$ é obtido. A partir de $p*$, em direção a $\overline{p}$ a uma distância $\delta = 3px$ um novo nó q é considerado. Testa-se então a existência de obstáculos entre p^* e q, incorporando q ao grafo caso não hajam obstáculos e consequentemente estejam conectados os nós. A abordagem se mostrou demasiadamente exclusiva ao restringir δ, podendo ocasionar situações em que a exploração fique presa, impossibilitada de encontrar caminhos estreitos após incorporar pontos próximos a estruturas como funis, o que poderia ser tratado inserindo pontos interiores a um raio de tamanho δ antes de identificar uma colisão. Vale ressaltar que estruturas de dados otimizadas foram cogitadas, porém a estrutura e parâmetros em constante mudança do problema não permitiram o uso do recurso.

E assim, ao inserir um novo nó ao grafo, testa-se a distância deste à configuração alvo, caso a mesma esteja dentro de um raio $r = 4px$, não havendo obstáculos impedindo a conexão, é inserida ao grafo. O raio para inserção do nó objetivo poderia ter sido ignorado, conectando-o a qualquer momento em que uma linha reta navegável fosse identificada. E para finalmente gerar o caminho, a

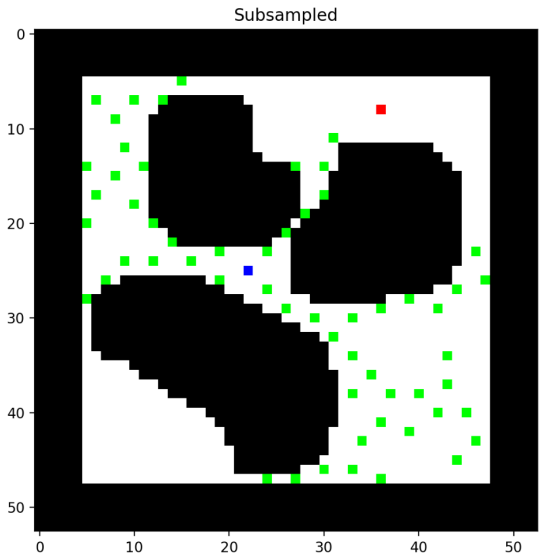

Figura 3. Execução do RRT de árvore única

estrutura de árvore requer somente que sejam percorridos os nós a partir do nó final em direção aos seus antecessores.

A Fig. 3, ilustra a execução do algoritmo tendo estabelecido o ponto azul como configuração inicial e o ponto vermelho como objetivo. Os pontos em verde, são nós da árvore produzida, a qual encontra o caminho ao passar por entre os dois obstáculos na porção superior do mapa. Curiosamente, a porção inferior-esquerda do mapa é pouco explorada, devido aos caminhos estreitos que se fizeram disponíveis, o que poderia ser tratado por um método de sorteio no entorno dos obstáculos.

4.2 Par de Árvores

Adaptando o algoritmo anterior, para expandir uma segunda árvore, o mesmo processo ocorre de maneira análoga e independente de modo que a segunda árvore tenha como raiz a configuração objetivo.

A primeira etapa acrescentada acontece após a inserção de um nó p a uma das árvores, que chamaremos $\boldsymbol{G}$. Esta consiste em percorrer a árvore oposta H em busca de um nó q à distância de até $3px$ do último nó p adicionado à $\boldsymbol{G}$, caso p e q possam ser conectados, não havendo obstáculos entre eles, é satisfeita a condição de parada.

A partir daí, os caminhos que ligam p e q às suas raízes são combinados para que configurações inicial e final sejam ligadas. Fig. 4 exibe resultados do algoritmo com a configuração inicial em azul e os nós expandidos de sua árvore em verde, o objetivo é marcado em vermelho ao canto esquerdo inferior e os nós da árvore nele enraizados em rosa. Ao centro, pode-se observar onde um pixel rosa encontra um verde, conectando as árvores.

4.3 Desempenho

Para observar o desempenho da implementação, foram simulados 1000 cenários do RRT com árvores duplas nas configurações inicial e final vistas na Fig. 4, medidos os tempos de computação desde o início ao momento em que as árvores se encontram. A Fig. 5 resume esses resultados

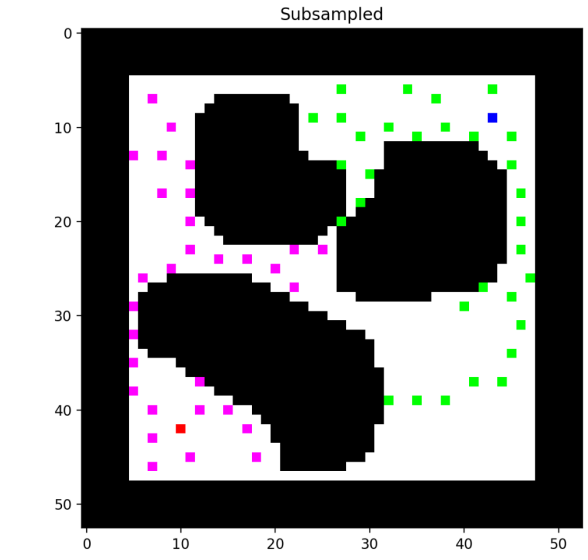

Figura 4. Execução do RRT de árvore dupla

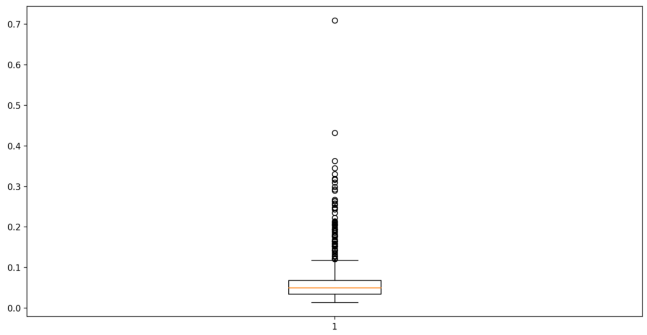

Figura 5. Boxplot do tempo de computação das mil execuções do RRT de árvore dupla.

em um *boxplot*, ilustrando o tempo médio em torno dos $.05s$. Os *outliers* observados correspondem a cenários em que árvores de raízes em espaços confinados não conseguem explorar novos espaços.

REFERÊNCIAS

(2022). Heapq. URL `https://docs.python.org/3/library/heapq.html`.

(2022). Kd-tree. URL `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html`.

(2022). Networkx. URL `https://networkx.org/`.