Universidade Federal de Minas Gerais

Escola de Engenharia

EEE935 - Planejamento de Movimento de Robôs

Trabalho Prático 2

Felipe Bartelt de Assis Pessoa 2016026841

29 de janeiro de 2022

Sumário

1 Introdução

O trabalho visa o estudo de algoritmos de planejamento, baseados em amostragem, pesquisa em grafos, além de algoritmos capazes de realizar a cobertura de um ambiente e gerar as deformações por retração do espaço topológico associado ao espaço livre do ambiente. Tentou-se ser o mais fiel possível às descrições de Choset et al. [1] quanto à esses algoritmos, entretanto algumas premissas e modificações foram necessárias, de forma que o foco da explicação quanto à implementação está nas modificações e não no algoritmo original.

Dessa forma, na Subseção 2.1 é apresentado o algoritmos A*, junto dos resultados obtidos; na Subseção 2.2 é apresentado o algoritmo de construção do diagrama de Voronoi generanlizado (GVD) baseado em sensores, assim como seus resultados; na Subseção 2.3 é apresentado o algoritmo de cobertura de ambiente, baseado na decomposição trapezoidal do mesmo, junto dos resultados obtidos; na Subseção 2.4 é apresentado o algoritmos RRT, seus resultados e uma comparação com os resultados obtidos pelo A*;

2 Simulações

O robô utilizado em todas as simulações apresenta 1440 sensores laser com alcance de 2 m, distribuídos igualmente em 360°. O robô é quadrado com tamanho 0.5 m. Utilizou-se o ROS-Noetic para as implementações em Python, assim como o simulador StageRos. Para a simulação nesse ambiente 2D, definiu-se diversos arquivo `.world`, onde são definidas as características do ambiente, tais como, obstáculos, robôs e outros objetos. Os mapas foram criados manualmente por meio do *software* GIMP.

O robô utilizado é não-holonômico e, assim, utilizou-se um controlador com *feedback linearization*, que recebe a velocidade instantânea, em componentes (x, y), a ser seguida e retorna seu *twist* resultante. O parâmetro, comumente denominado apenas por d no *feedback linearization* foi tomado como 0.1, uma vez que esse valor corresponde a 1/5 do comprimento do robô.

2.1 A*

A implementação tomada parte do pressuposto de que existe um *grid* do ambiente em questão. Tal que esse *grid* tenha valores 1 para representar os obstáculos e valores 0 para representar o espaço livre. É também adotado que o robô seja puntiforme, ou seja, o *grid* deve ter todos os obstáculos expandidos de um tamanho próximo ao maior tamanho do robô, de tal forma que colisões sejam evitadas. O mapa utilizado é apresentado na Figura 1 e tem tamanho de $(16, 16)$m, com resolução de 28 px/m.

Tendo-se o *grid*, as configurações inicial e objetivo do planejamento, o algoritmo cria um grafo iterativamente, nos quais os nodos são os pixels do *grid* e seus vizinhos são dados pela conectividade de 4 pontos, de tal forma que o custo de percorrer uma aresta que liga dois nodos é sempre 1. O custo de colisão com um obstáculo igual a 1000, de forma a evitar que o algoritmo gere um caminho que atravesse obstáculos. Utilizando uma heurística euclidiana, o algoritmo gera uma lista de prioridades, utilizando *breadth-first search*, e uma lista de nodos visitados, conforme descrito por Choset et al. [1]. A partir de ambas as listas, o algoritmo gera uma lista de pixels a serem percorridos pelo robô, o caminho planejado. Convertendo-se as coordenadas em pixels para metros, é possível controlar o robô para que convirja à configuração objetivo.

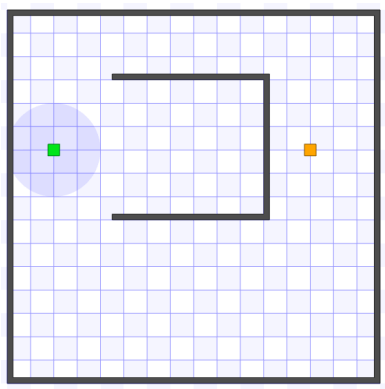
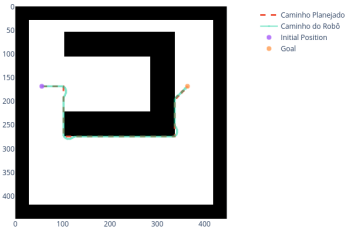


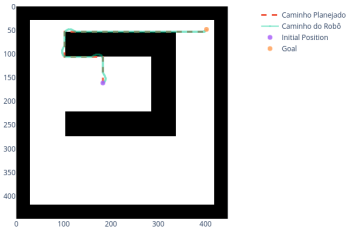
Figura 1: Mapa utilizado para simulação do algoritmo A* no Stage

2.1.1 Resultados

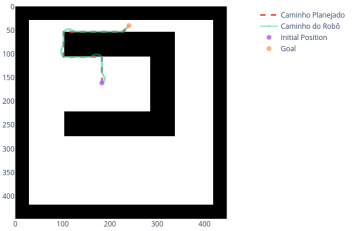
A implementação foi deveras satisfatória, uma vez que o algoritmo A* é capaz de encontrar, se houver, o caminho ótimo de uma configuração inicial até uma configuração objetivo, como mostra a Figura 2. Além disso, nota-se a importância de se considerar um *grid* com os obstáculos expandidos, visto que o A* tende a gerar um caminho rente aos obstáculos, como pode ser visto nos três exemplos da Figura 2. Ao se comparar os tempos de execução, notou-se que o planejamento referente à Figura 2a tomou o maior tempo, 41 s, o que se supõe ser devido ao maior volume de espaço livre entre a configuração inicial e objetivo, que gera um grafo com mais nodos e, portanto, com maior custo computacional.



(a) q_g: $(-6, 2)$m ou $(56, 168)$px. q_g: $(5, 2)$m ou $(364, 168)$px



(b) q_0: $(-1.46, 2.25)$m ou $(183, 161)$px. q_g: $(6.36, 6.29)$m ou $(402, 48)$px



(c) q_g: $(-6, 2)$m ou $(56, 168)$px. q_g: $(5, 2)$m ou $(364, 168)$px

(d) q_g: $(-6, 2)$m ou $(56, 168)$px. q_g: $(5, 2)$m ou $(364, 168)$px

Figura 2: Planejamentos gerados pelo RRT para três configurações inciais (q_0), mostrados em ciano, e de objetivo (q_g), em vermelho, diferentes. Nodos filhos gerados para a árvore da configuração inicial (T_{init}) são mostrados em roxo e nodos filhos gerados a árvore da configuração do objetivo (T_{goal}) são mostrados em laranja. O caminho planejado é mostrado por setas em azul.

2.2 Diagrama de Voronoi Generalizado

O mapa utilizado é apresentado na Figura 3 e apresenta tamanho de $(64, 40.5)$m, com resolução de 13 px.

Para a implementação do algoritmo capaz de gerar o diagrama de Voronoi generalizado (GVD), por meio de sensores, buscou-se duas alternativas. A primeira se baseia na existência de mínimos locais em planejamentos feitos por meio de funções potenciais simples. Dessa forma,

adotou-se potenciais repulsivos, inversamente proporcionais à distância do robô ao obstáculo, de tal forma que quando a resultante é nula, o robô se encontr em um ponto equidistante a dois obstáculos, dado que o robô somente enxerga esses dois. Uma vez que não foi possível obter uma forma de se verificar o caso de o robô estar em um *meet point*, o algoritmo não é completo e falha sempre que o robô encontra 3 ou mais obstáculos.

Na segunda abordagem, tomou-se três estados de movimento para o robô. Caso o robô somente "enxergue" um obstáculo, ele se move na direção oposta ao vetor normal do obstáculo, visando então encontrar a equidistância com um segundo obstáculo. A todo momento, calcula-se as menores distâncias à cada obstáculo visto e, dessa forma, é possível estimar quantas dessas distâncias são próximas. Caso o número de distâncias próximas seja exatamente dois, o algoritmo toma o vetor que liga o ponto mais próximo do obstáculo O_1 ao ponto mais próximo do obstáculo O_2, esse vetor é rotacionado em 90° e passa a ser a velocidade de referência para o robô, sendo análogo à uma tangente simultânea de dois obstáculos. O terceiro estado ocorre quando o número de distâncias similares é igual ou superior a três. Nesse caso, o algoritmo toma um processo análogo ao segundo estado, porém escolhendo aleatoriamente o par de obstáculos que estão equidistantes ao robô. O ponto onde isso ocorre é um *meet point*, dessa forma, armazena-se esse ponto, assim como o vetor resultante seguido pelo robô, de forma a evitar que, caso haja outro vetor tangente, o robô dê preferência para um vetor ainda não utilizado, ou seja, o robô siga para uma parte do GVD ainda não explorada.

2.2.1 Resultados

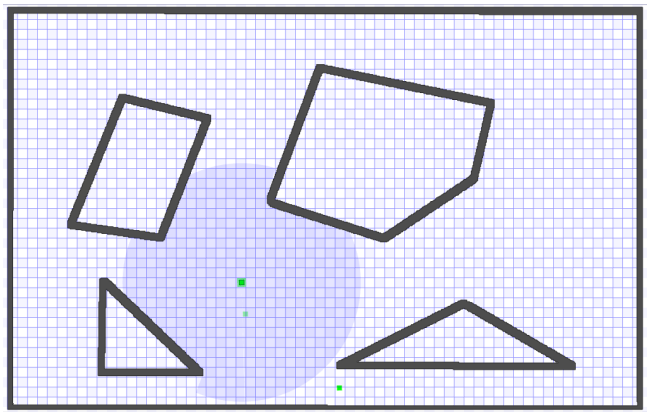


Figura 3: Mapa utilizado para simulação do algoritmo GVD no Stage

O GVD gerado pela primeira abordagem proposta é mostrado na Figura 4, na qual percebe-se uma grande similaridade com o GVD esperado, entretanto, uma vez que não foi possível fazer um tratamento dos *meet points*, o algoritmo não é completo e não é capaz de gerar um GVD conexo.

Para a segunda abordagem, encontrou-se outro problema. Uma vez que havia uma descontinuidade nas medições do sensor via Stage, várias vezes o algoritmo é prejudicado, encontrando um obstáculo a mais do que realmente existe e concluindo haver então dois obstáculos equidistantes quando na verdade são o mesmo obstáculo. O algoritmo também não foi finalizado, faltando corrigir erros do terceiro estado. Entretanto, nota-se pela Figura 5 um desempenho pouco mais satisfatório, o trajeto seguido pelo robô conseguiu mapear partes do GVD de forma satisfatória, faltando então apenas ser capaz de torná-lo conexo.

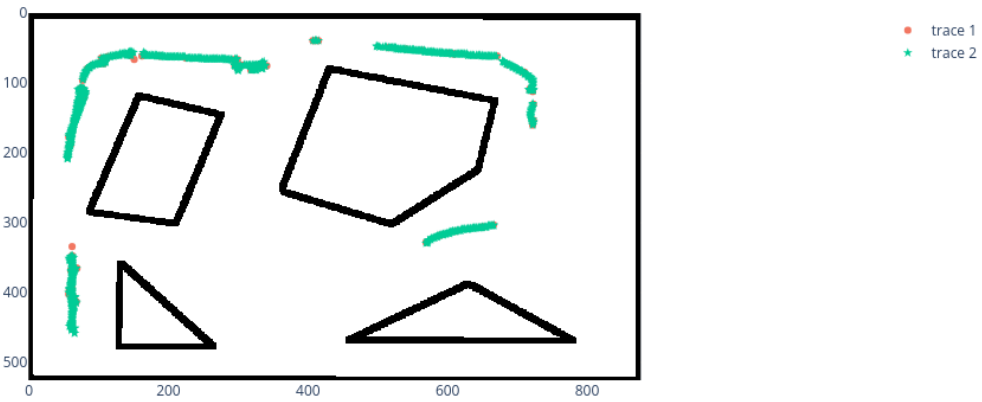


Figura 4: GVD gerado pela primeira abordagem de algoritmo

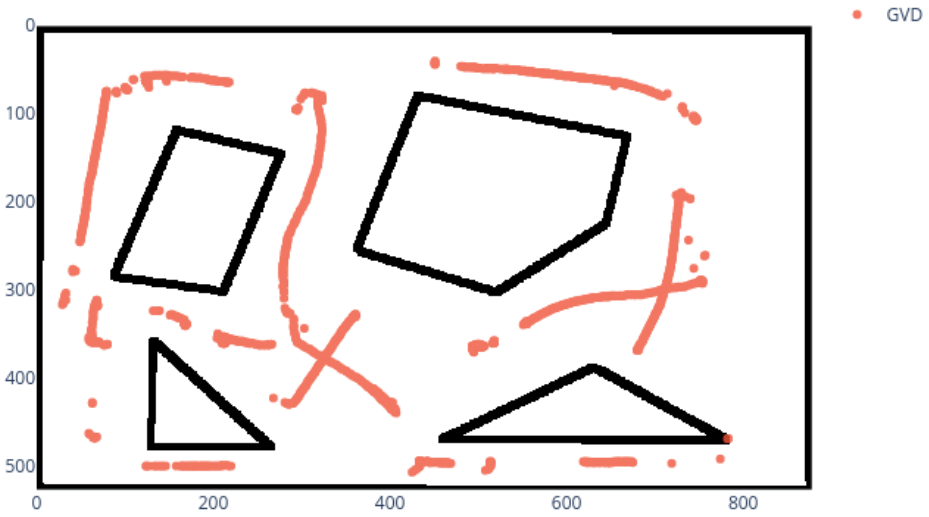
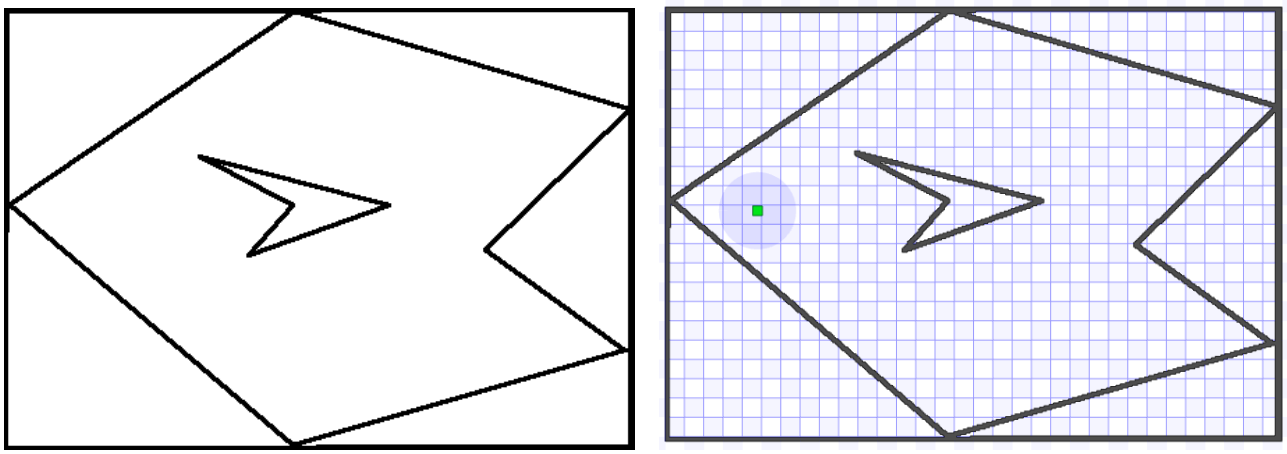


Figura 5: GVD gerado pela segunda abordagem de algoritmo

2.3 Decomposição Trapezoidal e Cobertura do Ambiente

Inicialmente, desenhou-se um mapa para a decomposição trapezoidal, Figura 6a no *software* GIMP, baseando-se nos exemplos fornecidos por Choset et al. [1], obtendo-se assim um ambiente para simulação conforme a Figura 6b, utilizando um tamanho de $(32, 22.5)$m, com resolução de 14 px/m. Dessa forma, fez-se a decomposição trapezoidal do ambiente de forma manual, que pode ser vista na Figura 7.



(a) Mapa utilizado

(b) Visualização do mapa pelo Stage.

Figura 6: Mapa desenhado e sua visualização no simulador Stage.

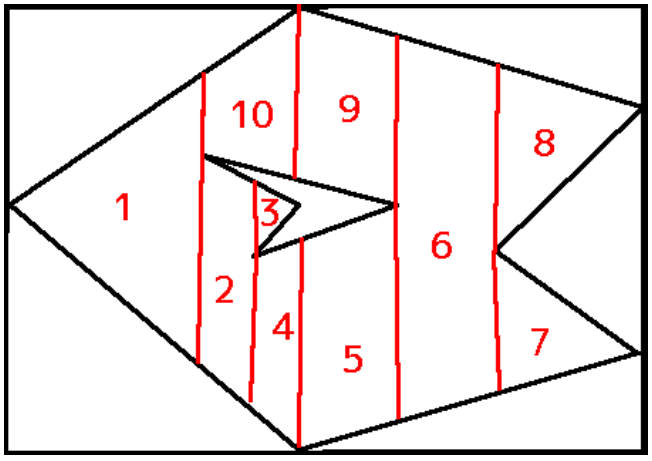


Figura 7: Decomposição trapezoidal considerada.

Com a decomposição feita, criou-se um grafo para a representação do mapa. Cada nodo do grafo, correspondentes às células da decomposição, contém suas extremidades à esquerda e à direita, assim como seus vizinhos. Ao adicionar um vizinho a um nodo, o grafo automaticamente determina se esse vizinho está à esquerda ou à direita, baseando-se em suas extremidades, dividindo os vizinhos em dois subconjuntos: vizinhos à esquerda e vizinhos à direita. Esses subconjuntos são ordenados, de forma crescente, com base nas extremidades esquerdas. Assim, é possível gerar um caminho que passe por todas as células ao se fazer uma busca gulosa, utilizando *breadth search*, priorizando-se células cuja extremidade esquerda é mais próxima da célula atual, tal que o caminho contenha todas as células.

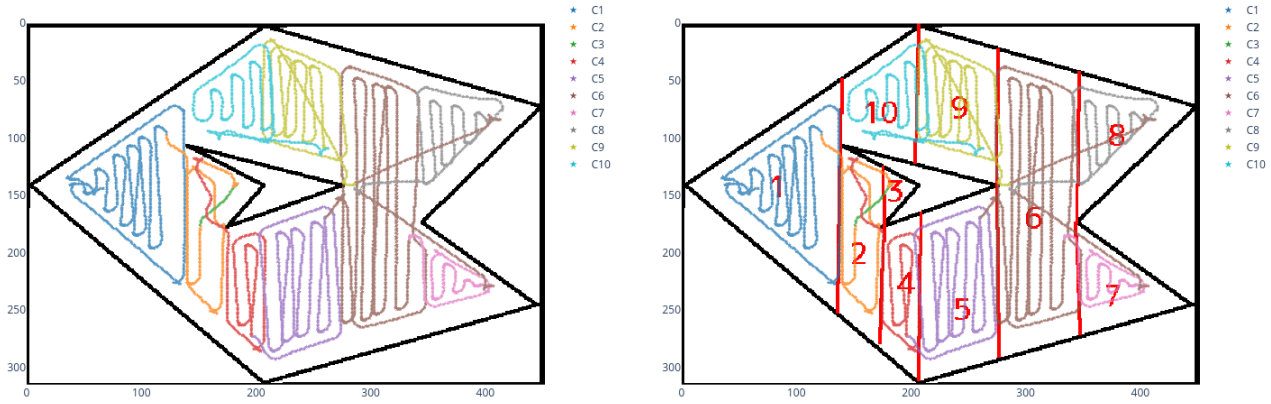
Uma vez que a decomposição trapezoidal é manual, o algoritmo de cobertura assume a existência de um arquivo `.pickle`, que contém uma lista dos nodos do grafo gerado. Dessa forma, baseado na posição inicial do robô, o algoritmo gera a ordem de células a serem visitadas e com base nela, realiza a cobertura do ambiente.

A cobertura do ambiente pode ser descrita por meio de três estados: mudança de célula, seguimento de bordo e cobertura. No estado de mudança de célula, o robô se dirige à extremidade esquerda da próxima célula a ser explorada e, quando atingida, troca-se para o estado de seguimento de bordo. Nesse segundo estado, o robô realiza o mesmo comportamento de *boundary following* do *Tangent Bug* e, ao completar uma volta completa no perímetro da célula, entra no estado de cobertura. No estado de cobertura, o robô inicialmente segue para baixo até encontrar um obstáculo, então move L metros para a direita, onde L é o tamanho do robô (0.5 m no caso simulado), após, o robô move para cima até encontrar um obstáculo e novamente se move para à direita, o processo então é repetido até o robô alcançar a extremidade direita da célula, quando então retorna ao estado de mudança de célula. Caso o robô necessite passar por uma célula já explorada, então ele se dirige à sua extremidade esquerda, sem reexplorá-la e passa diretamente para a célula seguinte do planejamento. O algoritmo se encerra assim que a última célula é totalmente explorada, não havendo necessídade de retornar à célula inicial.

A motivação para o estado de seguimento de bordo foi a tentativa de garantir a cobertura completa da área de uma célula. Uma vez que o robô, no estado de cobertura, se dirige à próxima célula assim que chega próximo o suficiente da extremidade direita da célula atual, seria possível existir pontos, não explorados, na reta vertical que contém essa extremidade. Ao tomar o estado de seguimento de bordo, conforme descrito, garantir-se-ia que todos os pontos contidos nessa reta já teriam sido visitados.

2.3.1 Resultados

O comportamento supracitado pode ser facilmente visualizado por meio da Figura 8a e Figura 8b, nas quais é explicitada a célula em exploração no momento em que o robô passa por cada ponto. A Figura 8b permite também que se visualize a cobertura dentro da decomposição trapezoidal feita. Em ambas, nota-se os bordos das células percorridos, movimentos em ziguezague (estado de cobertura) e os trajetos de uma célula a outra. Além disso, nota-se o comportamento de evitar a reexploração de uma célula ao se observar o trajeto da célula C_7 a C_6, que instantaneamente se altera para um trajeto de C_6 a C_8, o mesmo ocorre no trajeto de C_8 a C_9.



(a) Caminho percorrido pelo centro do robô.

(b) Caminho percorrido pelo centro do robô, com indicação da decomposição trapezoidal.

Figura 8: Caminho percorrido pelo centro do robô durante a cobertura do ambiente. Os marcadores indicam a célula em exploração, quando o robô percorreu os pontos.

A Figura 8 somente traz informações quanto ao trajeto seguido pelo centro do robô, não obtendo-se uma referência quanto à área coberta durante o trajeto. Dessa forma, expandiu-se

cada ponto percorrido pelo centro do robô em 4 px em cada direção, o que equivale à metade do tamanho do robô, aproximadamente. Obteve-se assim a área total coberta pelo robô, de forma aproximada, apresentada na Figura 9. Nota-se que há áreas não cobertas pelo robô, entretanto correspondem a somente 0.6 m de largura, no máximo, equivalente a pouco mais que a largura do robô. Isso se deve ao fato do algoritmo não saber que existem mais pontos não explorados em uma célula assim que alcança sua extremidade direita, no estado de cobertura, e assim, ignora pontos abaixo ou acima da mesma. A correção mais simples para esse problema necessitaria do conhecimento de todos os vértices das células.

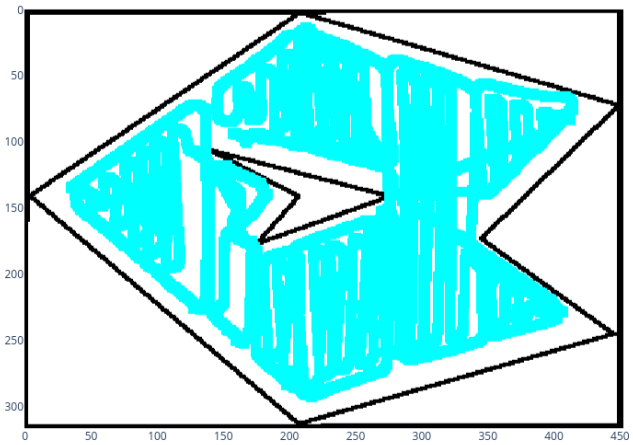


Figura 9: Área total explorada pelo robô. Aproximada por meio da expansão do caminho percorrido pelo centro do robô.

2.4 RRT

A implementação tomada parte do pressuposto de que existe um *grid* do ambiente em questão. Tal que esse *grid* tenha valores 1 para representar os obstáculos e valores 0 para representar o espaço livre. É também adotado que o robô seja puntiforme, ou seja, o *grid* deve ter todos os obstáculos expandidos de um tamanho próximo ao maior tamanho do robô, de tal forma que colisões sejam evitadas. O mapa utilizado foi o mesmo para o algoritmo A*, Figura 1, assim como seu grid correspondente.

Inicialmente, criam-se duas árvores (RRT), uma com a raiz da configuração inicial T_{init} e outra com raiz sendo a configuração do objetivo T_{goal}. Uma das árvores T_1 é selecionada e gera-se uma configuração aleatória com distribuição uniforme q_{rand} no espaço livre. Obtendo-se o ponto aleatório, procura-se, na árvore T_1, o nodo com configuração mais próxima a q_{rand}. A partir desse ponto, q_{nearest}, toma-se um passo de tamanho L na direção de q_{rand}, obtendo-se um novo ponto q_{new}. Se o caminho de q_{nearest} até q_{new} não colidir com nenhum obstáculo, q_{new} é adicionado aos filhos de q_{nearest}. Então, encontra-se na outra árvore, T_2, o nodo cuja configuração é a mais próxima de q_{new} e, caso o caminho de q_{new} até essa configuração não resulte em qualquer colisão, as árvores são mescladas e o algoritmo é finalizado. Caso contrário, inverte-se a ordem das árvores ($T_1 = T_2,\ T_2 = T_1$) e repete-se o procedimento anterior, expandindo-se alternadamente ambas as árvores, até que as árvores sejam mescladas ou o número máximo de iterações seja atingido.

Para a geração da configuração aleatória, amostrou-se pixels uniformemente em todo o *grid*. Caso esse ponto não repouse no espaço livre, um novo ponto é gerado até que se encontre um ponto no espaço livre ou o máximo de iterações (1000) seja atingido. O número máximo de iterações adotado para a convergência do planejamento foi de 15 000.

A forma adotada para checar colisões no caminho entre configurações q_1 e q_2, foi a realização de uma regressão linear sobre as coordenadas, em metros, de ambas as configurações. Por meio da equação de reta obtida via regressão, geram-se 50 pontos entre ambas as configurações, as coordenadas em metros desses pontos são convertidas para pixels. Assim, verifica-se, pixel a pixel, se alguma das posições no *grid* representa um obstáculo.

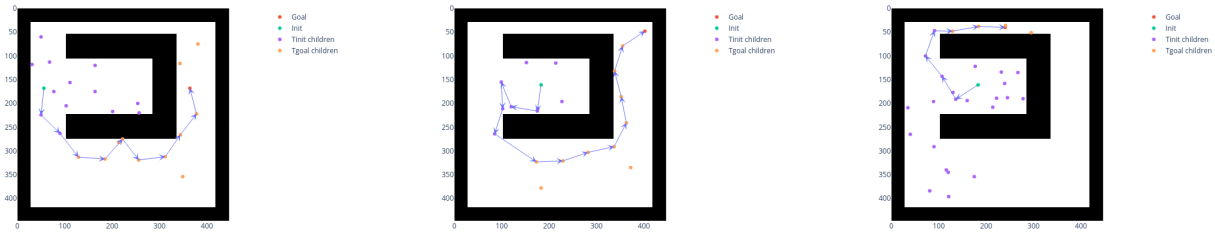
O passo adotado para gerar uma nova configuração q_{new}, selecionado de forma empírica, foi uma constante de 2 m, que corresponde à $1/8$ da largura do ambiente. Caso q_{new} não repouse no espaço livre, essa configuração é descartada e prossegue-se o funcionamento do algoritmo.

2.4.1 Resultados

As particularidades adotadas mostraram-se bastante satisfatórias. Nota-se, pela Figura 10a e Figura 10b, que o planejador não gera o caminho ótimo por meio dos nodos, apenas encontra o primeiro caminho válido. Isso não é uma desvantagem, visto que é bastante simples reduzir o custo desse trajeto uma vez encontrado um trajeto válido. É possível notar a consequência de se descartar q_{new} por meio da Figura 10c, tem-se muito mais nodos na árvore T_{init} do que em T_{goal}, uma vez que a primeira tem mais espaço livre ao seu redor. Ainda, constata-se o óbvio, quanto mais complexo é o trajeto, isso é, quanto menor é o espaço livre entre a configuração inicial e final, mais nodos são necessários para a convergência do algoritmo, como indica a Figura 10c.

Por meio da Figura 10 ainda é possível notar que o planejamento gerado pelo RRT não converge à mínimos locais, visto que todas as situações propostas tendem a induzir a convergência à um mínimo local.

Uma vez que as configurações dos exemplos utilizados no RRT foram os mesmos adotados para o algoritmo A*, é possível comparar o comportamento de ambos. Em média, o caminho gerado pelo RRT, para todas essas configurações, é menor que 1 s, sendo bastante mais eficaz que o A*, cuja média é de 20 s. Entretanto, conforme esperado, o RRT não gera caminhos ótimos, ao contrário do A*, isso fica bastante claro ao se comparar os caminhos planejados na Figura 2b e Figura 10b.



(a) q_g: $(-6, 2)$m ou $(56, 168)$px. q_g: $(5, 2)$m ou $(364, 168)$px

(b) q_0: $(-1.46, 2.25)$m ou $(183, 161)$px. q_g: $(6.36, 6.29)$m ou $(402, 48)$px

(c) q_0: $(-1.46, 2.25)$m ou $(183, 161)$px. q_g: $(0.57, 6.57)$m ou $(240, 40)$px

Figura 10: Planejamentos gerados pelo A* para três configurações inciais (q_0), mostrados em ciano, e de objetivo (q_g), em vermelho, diferentes. O caminho planejado é mostrado em tracejados vermelhos, enquanto o caminho real do robô é mostrado em verde, as posições inicial e objetivo são mostradas em roxo e laranja, respectivamente.

3 Conclusão

A maioria dos algoritmos foram satisfatórios. Tornou-se possível analisar os diversos comportamentos dos algoritmos, tanto desse quanto do trabalho anterior, para o mesmo mapa e assim compreender perfeitamente as vantagens e desvantagens de cada um. Observa-se que um algoritmo de cobertura não é simples, exigindo grande conhecimento prévio do ambiente para que seja possível realizar a cobertura perfeitamente. Quanto ao algoritmo GVD, espera-se ser possível implementar perfeitamente a segunda abordagem proposta, buscando-se uma forma de contornar a descontinuidade dos sensores e corrigir os problemas encontrados para a implementação do terceiro estado de movimento, conjectura-se ser possível construir o GVD completo.

Referências

[1] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, May 2005. [Online]. Available: https://www.scholars.northwestern.edu/en/publications/principles-of-robot-motion-theory-algorithms-and-implementations