Análise e Comparação dos Algoritmos RRT* e Dijkstra para Planejamento de Caminhos em Terrenos Acidentados ⋆

Douglas Viana * **Israel Amaral** * **Adriano Rezende** *
Héctor Azpúrua *,† **Gustavo Pessin** † **Gustavo Freitas** *

* *Escola de Engenharia, Universidade Federal de Minas Gerais*
† *Instituto Tecnológico Vale (ITV)*
(douglasvc, israelfilipe, adrianomcr, gustavomfreitas)@ufmg.br;
(hector.azpurua, gustavo.pessin)@itv.org

Abstract: Path planning techniques are one of the main focuses on mobile robotics, being important in various segments that aim to develop autonomous systems. One example is the EspeleoRobô, a robotic device capable of moving through rough terrain to explore and map confined environments. Given the great variety of planners available, part of the process consists in investigating the performance of specific techniques and analysing which could be implemented in the EspeleoRobô. Thus, this article presents the analysis and comparison between path planning algorithms, the RRT* and Dijkstra, to show how both behave and how applicable they are to rough terrain. In that regard, the different terrain representations used by each algorithm are described and mobility metrics to be optimized are proposed. Simulations with the EspeleoRobô allow the comparison of estimated costs during the planning stage using the robot's mobility metrics running through virtual rough terrain. The results illustrate the behavior differences of both planning methods, indicating parameters that influence processing time and the final costs associated with the paths produced. This knowledge will aid in the implementation of dedicated algorithms for the EspeleoRobô.

Resumo: Técnicas de planejamento de caminhos são amplamente utilizadas na área da robótica móvel, com especial importância no desenvolvimento de sistemas autônomos. Um exemplo de aplicação é no EspeleoRobô, um dispositivo robótico capaz de se locomover em terrenos acidentados para explorar e mapear ambientes confinados. Com a grande variedade de planejadores disponíveis, cabe investigar o desempenho de técnicas específicas e analisar quais poderiam ser embarcadas no EspeleoRobô. Dessa maneira, este artigo visa a análise e comparação dos algoritmos de planejamento de caminhos RRT* e Dijkstra, de forma a examinar o funcionamento e a aplicabilidade das técnicas em terrenos acidentados. Neste sentido, as diferentes representações de terreno utilizadas por cada um dos algoritmos são descritas, além da proposição de métricas de mobilidade a serem otimizadas. Simulações com o EspeleoRobô possibilitam comparar os custos estimados durante a etapa de planejamento com as métricas de mobilidade do robô percorrendo um terreno virtual acidentado. Os resultados ilustram as diferenças de funcionamento dos métodos de planejamento, indicando parâmetros que influenciam no tempo de processamento e nos custos finais associados aos caminhos. Estas informações auxiliarão na implementação de algoritmos dedicados a serem embarcados no EspeleoRobô.

Keywords: Path Planning; Mobile Robotics; Rough Terrain.
Palavras-chaves: Planejamento de Caminhos; Robótica Móvel; Terrenos Acidentados.

1. INTRODUÇÃO

Algoritmos de planejamento de caminhos possuem grande importância em diversas tarefas realizadas no mundo hoje, com aplicações em robótica móvel, inteligência artificial utilizada em jogos virtuais, veículos autônomos, aplicativos de transporte de pessoas, entre outras. Esta área recebe muita atenção atualmente devido à grande demanda deste tipo de tecnologia, fazendo com que a busca por maior eficiência seja constante.

Um exemplo de aplicação é no EspeleoRobô (Fig. 1), um dispositivo robótico móvel capaz de se locomover em terrenos acidentados para explorar e mapear ambientes confinados de forma semi-autônoma (Azpúrua et al., 2021). Este robô de serviço, desenvolvido em parceria entre o Instituto Tecnológico Vale (ITV) e a Universidade Federal de Minas Gerais (UFMG), tem como objetivo o auxílio na exploração de ambientes confinados, como sistemas de tubulação, galerias de drenos de barragens, moinhos, silos,

⋆ Este trabalho foi parcialmente financiado pela Vale S.A. e Instituto Tecnológico Vale (ITV), Universidade Federal de Minas Gerais (UFMG), e o Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

assim como cavernas naturais, ambientes desconhecidos e que podem apresentar riscos aos operadores envolvidos.

Figura 1. EspeleoRobô percorrendo terrenos acidentados.

Devido às limitações de espaço, presença de obstáculos e não estruturação do ambiente, a teleoperação em locais confinados por meio de ondas de rádio ou cabo umbilical nem sempre é uma opção viável. Uma possível solução corresponde ao desenvolvimento de estratégias que permitam a operação autônoma de dispositivos robóticos.

Dentre essas estratégias está incluído o planejamento de caminhos, buscando otimizar diferentes métricas de forma a maximizar a mobilidade do robô em um determinado terreno. Neste contexto, uma importante etapa no desenvolvimento de um sistema de navegação autônomo consiste em investigar diferentes técnicas de planejamento de caminho, definindo possíveis algoritmos a serem implementados no EspeleoRobô.

Os estudos apresentados neste artigo são uma extensão dos trabalhos (Santos et al., 2018, 2019), que propuseram a utilização do algoritmo Dijkstra para o planejamento de caminhos do EspeleoRobô. Neste artigo, o Dijkstra é comparado com outro algoritmo planejador, o RRT* (*Rapidly-exploring Random Trees*), escolhido devido a sua forma de funcionamento e potencial de escalabilidade em grandes terrenos. O RRT* faz uso de amostras aleatórias coletadas do terreno para construir uma estrutura tipo árvore, possuindo assim aspectos estocásticos que influenciam nos resultados obtidos. De forma a garantir a aleatoriedade da amostragem, é necessário que a representação do ambiente seja contínua, sendo que o caminho gerado depende diretamente do número de amostras. Já o Dijkstra tem característica determinística e discreta, onde o terreno é representado por um grafo e o caminho de menor custo é encontrado pela análise dos nós de origem e de destino e os demais disponíveis, teoricamente gerando sempre resultados ótimos.

Os algoritmos planejadores são avaliados considerando um terreno acidentado, buscando minimizar diferentes métricas relacionadas à mobilidade do robô, incluindo distância percorrida, variação de altura e inclinação do terreno, além de uma combinação de múltiplas métricas otimizadas conforme pesos atribuídos pelo operador. Os caminhos ótimos são verificados por meio de simulações, possibilitando comparar os custos estimados pelos planejadores com as métricas de mobilidade do robô ao realizar diferentes percursos num modelo virtual do terreno. Os resultados ilustram as diferenças de funcionamento dos métodos de planejamento, indicando a influência do número de iterações e discretização do terreno no tempo de processamento e custos finais obtidos. Estas informações auxiliarão na implementação de algoritmos dedicados a serem embarcados no EspeleoRobô, fazendo parte do sistema de navegação autônoma do robô.

2. REPRESENTAÇÃO DO TERRENO

Um dos principais problemas abordados na área de robótica é o planejamento de um movimento livre de colisão que leve o robô de uma posição inicial a uma final em um ambiente contendo obstáculos. O planejamento de caminhos possui vários desafios e etapas que devem ser superados. Duas importantes etapas são a representação do ambiente e o planejamento de um caminho factível a ser seguido pelo robô por meio de alguma técnica, como por exemplo, os algoritmos de busca em grafo. Nesse caso, mais uma etapa é necessária: a geração de um grafo que represente o ambiente por meio do mapa obtido na etapa de representação.

Três aspectos importantes devem ser considerados ao se representar um ambiente: o modelo deve ser compacto o suficiente de modo que o custo computacional necessário para processá-lo não seja muito grande, o mapa deve ser capaz de representar o local adequadamente, e o modelo deve ser capaz de lidar com as incertezas inerentes às leituras dos sensores utilizados na percepção do ambiente (Burgard et al., 2016).

Existem várias técnicas para realizar a modelagem geométrica de um terreno (Siciliano and Khatib, 2016). Uma que apresenta um baixo custo computacional é o modelo de grades de elevação, que consiste em armazenar a informação de altura do mapa em relação à um plano de referência. A topografia do terreno é descrita por meio de uma função $h = f(x, y)$, onde x e y representam as coordenadas de um ponto no plano horizontal e h a altura do terreno naquele ponto. Devido a simplicidade da estrutura de dados, esta técnica tem sido muito utilizada em robôs móveis que operam em ambientes naturais que não possuem superfícies verticais ou tetos.

Uma abordagem mais genérica consiste na representação do terreno por uma nuvem pontos, normalmente obtida por fotogrametria ou sensores de distância como Lidar ou câmera RGBD embarcados no robô. Nesta representação, as informações são armazenadas de acordo com as coordenadas x, y, z de cada ponto. Com isso, não há restrições em relação à geometria do ambiente. Devido ao seu alto custo computacional, diversas abordagens foram implementadas para lidar com essa representação. Uma dessas técnicas é apresentada em (Hornung et al., 2013) que é o OctoMap, baseado em octree, uma estrutura em árvore em que cada nó possui conexão com outros oito nós, com estimação probabilística de ocupação para cada nó.

Outra alternativa de representação é por meio de malhas triangulares, uma estratégia bastante utilizada, sendo capaz de representar qualquer combinação de superfícies de forma compacta. Além disso, mesmo se o tamanho original da malha for grande, é possível utilizar algoritmos de simplificação para reduzir o número de vértices do mapa (Heckbert and Garland, 1999).

Neste artigo são analisados dois algoritmos de busca que necessitam de diferentes representações do terreno. O RRT* utiliza amostras aleatórias do ambiente, o que faz

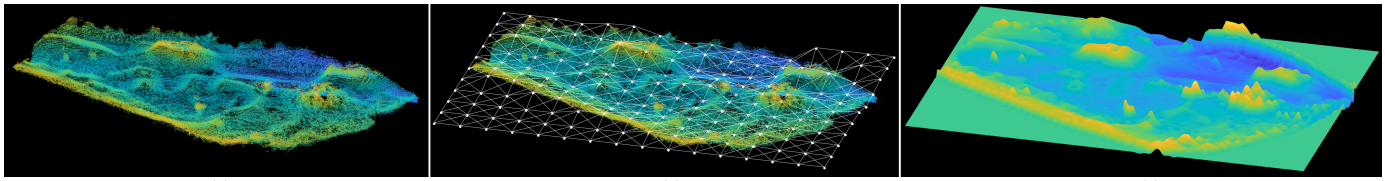

Figura 2. Diversas etapas da representação do terreno: (a) Nuvem de pontos obtida por sensores; (b) Exemplo de *grid* adquirido a partir da nuvem de pontos; (c) Superfície criada por interpolação.

com que o mesmo tenha que ser contínuo para que esta aleatoriedade seja garantida. Já o Dijkstra funciona a partir de um grafo que possui coordenadas discretas.

Ambos os algoritmos utilizam como entrada uma nuvem de pontos, discretizada em um *grid* uniforme. A Fig. 2(a) ilustra a nuvem de pontos utilizada nas simulações descritas neste artigo; a Fig. 2(b) contém o *grid* obtido.

Para o RRT*, um processo de interpolação é realizado a partir do *grid* criado, gerando uma função h que retorna a altura do terreno a partir de uma coordenada (x, y). Isso possibilita a amostragem aleatória necessária ao algoritmo, desde que os pontos escolhidos estejam dentro dos limites do ambiente representado pela nuvem de pontos. A Fig. 2(c) ilustra a superfície definida pela função h, criada a partir da mesma nuvem de pontos.

Para o algoritmo Dijkstra, o *grid* é utilizado diretamente para a criação do grafo, sendo que cada nó possui conexão com os outros nós vizinhos localizados acima, abaixo, na direita, esquerda e nas diagonais por meio de arestas. Nesta abordagem, cada nó representa um ponto do espaço e as arestas conectam dois nós adjacentes. Assim, um caminho é uma sequência de nós adjacentes que sejam acessíveis. Cada aresta está associada a um custo, que é utilizado para orientar a busca. Na Fig. 3 podem ser vistos o terreno discretizado, e o grafo equivalente com as conexões e custos entre um determinado nó e seus vizinhos.

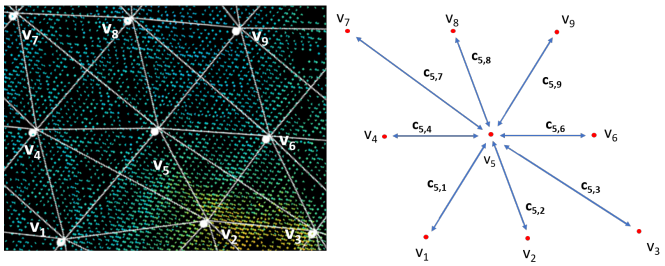

Figura 3. (a) Discretização do terreno; (b) Grafo equivalente com custos associados.

3. ALGORITMOS DE PLANEJAMENTO E MÉTRICAS APLICADAS

Após passar pela etapa de representação do terreno, é possível a aplicação dos algoritmos de planejamento já mencionados anteriormente. Nas próximas subseções são apresentados os algoritmos RRT* e Dijkstra, assim como as métricas de mobilidade adotadas.

*3.1 RRT**

O algoritmo RRT* (*Rapidly-exploring Random Trees*) é um método de busca probabilístico que realiza o planejamento por meio de amostras aleatórias agrupadas em uma estrutura do tipo árvore (Karaman and Frazzoli, 2011). Este algoritmo foi desenvolvido a partir do algoritmo RRT, que funciona de maneira similar, porém não consegue garantir a convergência dos caminhos gerados para a solução ótima, o que levou a necessidade de adaptações para otimizar o processo.

Ambos os algoritmos começam seu funcionamento a partir de pontos amostrados do terreno, que são utilizados para a construção da estrutura em árvore desde que o novo ponto esteja no espaço livre de obstáculo. No caso do RRT, a única etapa seguinte é a determinação do nó mais próximo já pertencente a árvore referente ao novo ponto obtido, que será conectado se não existir obstáculos entre eles e então o ciclo se repete. Isto acaba dificultando as chances de se encontrar o caminho ótimo.

Dessa maneira, duas novas rotinas foram adicionadas ao RRT, dando origem ao algoritmo RRT* e fazendo com que o mesmo consiga garantir o resultado ideal quando o número de amostras tende a infinito. Por isso ele é classificado como um algoritmo probabilisticamente ótimo. Mesmo que na prática o ótimo nunca seja idealmente alcançado, sabe-se que com um número grande o suficiente de amostras será possível chegar bem próximo do caminho considerado ótimo.

A primeira das rotinas mencionadas consiste na checagem da vizinhança ao redor do novo ponto escolhido, de forma que se existe algum outro vizinho com conexão de menor custo se comparado ao vizinho mais próximo, então aquele outro vizinho será o nó utilizado para estabelecer a nova conexão à árvore. Isto pode ser feito armazenando o custo da conexão de cada nó da árvore a seu respectivo nó pai.

A segunda rotina se refere a um processo de reconstrução da árvore após a adição do novo ponto. Novamente, os vizinhos do novo nó são checados para testar se a reconexão com o novo ponto não resulta em um decréscimo total do custo até aquele ponto. Se este for o caso, então novas conexões são estabelecidas e as anteriores são desfeitas. Ambas as adições que resultaram na criação do novo algoritmo RRT* fazem com que sua árvore gerada seja melhor otimizada, resultando em caminhos mais próximos do ótimo. O pseudocódigo do RRT* é apresentado no Algoritmo 1.

Inicialmente, são fornecidos ao algoritmo os pontos de começo do caminho desejado (linha 1 do Algoritmo 1). A partir disto, o programa entra em um *loop* com número de execuções n (linha 2), que começa com a amostragem de um ponto aleatório do terreno (linha 3), a procura do ponto mais próximo já existente na árvore em relação ao ponto recentemente amostrado (linha 4) e a obtenção do

Algoritmo 1: RRT*

Entradas: x_{init}, x_{final}, $height\_function$
Saídas : S

1 $S \leftarrow \{x_{init}\}$
2 **for** $i = 1, ..., n$ **do**
3 $\quad x_{rand} \leftarrow SampleFree_i$
4 $\quad x_{nearest} \leftarrow Nearest(S, x_{rand})$
5 $\quad x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
6 $\quad$ **if** $ObstacleFree(x_{nearest}, x_{new})$ **then**
7 $\quad\quad X_{near} \leftarrow Near(S, x_{new}, min(\{\gamma RRT^*(log(card(S))/card(S))^{1/d}, \eta\})$
8 $\quad\quad S \leftarrow S \cup \{x_{new}\}$
9 $\quad\quad x_{min} \leftarrow x_{nearest}; c_{min} \leftarrow Cost(x_{nearest}) + c(Line(x_{nearest}, x_{new}))$
10 $\quad\quad$ **foreach** $x_{near} \in X_{near}$ **do**
11 $\quad\quad\quad$ **if** $CollisionFree(x_{near}, x_{new})$ & $Cost(x_{near}) + c(Line(x_{near}, x_{new})) < c_{min}$ **then**
12 $\quad\quad\quad\quad x_{min} \leftarrow x_{near}; c_{min} \leftarrow Cost(x_{near}) + c(Line(x_{near}, x_{new}))$
13 $\quad\quad\quad$ **end**
14 $\quad\quad$ **end**
15 $\quad\quad$ **foreach** $x_{near} \in X_{near}$ **do**
16 $\quad\quad\quad$ **if** $CollisionFree(x_{new}, x_{near})$ & $Cost(x_{new}) + c(Line(x_{new}, x_{near})) < Cost(x_{near})$ **then**
17 $\quad\quad\quad\quad x_{parent} \leftarrow Parent(x_{near})$
18 $\quad\quad\quad$ **end**
19 $\quad\quad$ **end**
20 $\quad$ **end**
21 **end**
22 **return** S

ponto que realmente será adicionado à árvore (linha 5). Este último passo é dado através de uma função chamada *Steer*, que retorna o ponto presente na reta formada por x_{rand} e $x_{nearest}$, distante de $x_{nearest}$ por um valor previamente definido.

A seguir, após checar se a reta formada por x_{new} e $x_{nearest}$ não ultrapassa os limites de algum obstáculo (linha 6), é verificado quais pontos já existentes na árvore estão presentes dentro de uma área específica ao redor de x_{new} (linha 7). O novo ponto é adicionado ao conjunto de vértices da árvore (linha 8). Depois, é analisado qual dos pontos de X_{near} resulta na conexão de menor custo com x_{new} (linhas 9 a 14), atualizando x_{min} e c_{min}. O custo entre pontos calculado faz uso da função de altura criada a partir da nuvem de pontos anteriormente, sendo que diferentes métricas são utilizadas, como distância, altura e inclinação, assim como uma combinação das mesmas.

Com o estabelecimento da nova conexão à árvore, a mesma passa agora por um processo de *rewiring* (linhas 15 a 19), que checa se não existem caminhos mais eficientes agora que um novo ponto foi adicionado. Se este for o caso, a árvore é atualizada, assim como seus correspondentes custos e então o processo se repete. Como o número de execuções do algoritmo é escolhido previamente, a densidade da árvore e a qualidade do caminho gerado são dependentes do mesmo. Dessa maneira, quanto maior for o número n, maior a probabilidade do resultado estar mais próximo do caminho ótimo.

Na Fig. 4 está um exemplo do processo de construção da árvore empregada no RRT*. É possível perceber o aumento da densidade da árvore com o aumento do número n, assim como as diferenças nos caminhos gerados. A implementação do algoritmo possui complexidade quadrática e é feita com expansão unidirecional, de forma que a árvore cresce somente a partir do ponto inicial.

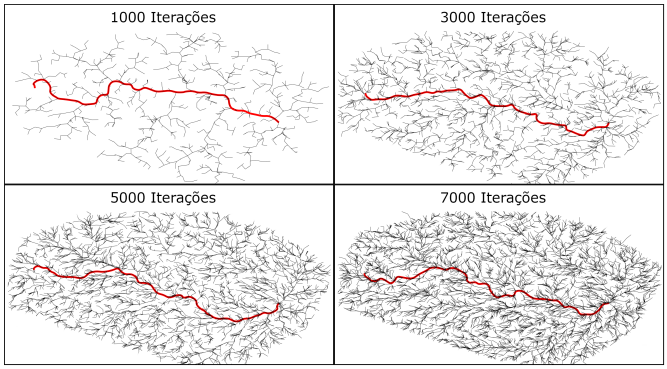

Figura 4. Progressão da árvore utilizada no algoritmo RRT.

3.2 Dijkstra

O algoritmo de Dijkstra é um método de busca em grafo comumente utilizado em tarefas que visam calcular a mínima distância entre um ponto específico e os demais. Neste trabalho, o Dijkstra é utilizado também em conjunto com diversas métricas, de forma que os custos calculados para cada uma delas é diferente.

Segundo Choset et al. (2005), o algoritmo de Dijkstra pode ser visto como uma variação do A*. Este último é um método de busca que utiliza uma função de custo $f(n) = g(n) + h(n)$, associada a diferentes métricas, como menor distância ou transversabilidade (Hart et al., 1968). Essa função avalia tanto o custo do nó inicial até o atual por meio da função $g(n)$, quanto o custo do nó atual até o nó final por meio da função $h(n)$. Dependendo da heurística $h(n)$ utilizada para estimar o custo de um nó qualquer até o objetivo, o algoritmo será completo e ótimo.

Quando não é possível estimar uma heurística admissível, o A* não garante convergência. Neste caso, uma solução é o algoritmo de Dijkstra. Ele também utiliza uma função de custo, porém considera $h(n) = 0$, de modo que $f(n) = g(n)$. Nos casos em que não há custos negativos, o algoritmo de Dijkstra irá sempre calcular o caminho de menor custo, por isso ele é classificado como um algoritmo ótimo na resolução. Este algoritmo possui como principal ideia realizar a classificação dos nós de um grafo durante a exploração do mesmo, tendo como critério a função de custo que analisa o custo do nó de origem até o nó explorado. Essa função é utilizada para classificar os vértices de um grafo durante a exploração e então armazená-los em uma estrutura do tipo fila de prioridade. O algoritmo mantém um conjunto V de nós os quais já tiveram o custo do menor caminho determinado e um conjunto O de nós não visitados. Esses nós são visitados pelo algoritmo utilizando a estimativa de menor caminho e então adicionados à V (Cormen et al., 2001). O pseudocódigo deste método é apresentado no Algoritmo 2 e seu funcionamento é ilustrado na Fig. 5.

São passadas como entradas para o algoritmo a matriz de adjacências A, onde é informado os nós acessíveis a partir da localização atual; uma matriz de custos C associados às arestas; a coordenada inicial x_{init} e a coordenada final x_{final}. Inicialmente, é determinada a função de custo de acordo com as matrizes A e C e o nó inicial $x_{initial}$ é adicionado ao conjunto O (linhas 1 e 2 do Algoritmo 2). Após

Algoritmo 2: Algoritmo de Dijkstra

Entradas: A, C, x_{init}, x_{final}
Saídas : $path$
1 $f(x) \leftarrow costFunction(A, C)$
2 $O \leftarrow O \cup \{x_{init}\}$
3 $x_{best} \leftarrow x_{init}$
4 **repeat**
5 **if** $x_{best} \notin V$ **then**
6 $V \leftarrow V \cup \{x_{best}\}$
7 **if** $x \notin V \; \forall x$ *vizinhos de* x_{best} **then**
8 $O \leftarrow O \cup \{x\}$
9 **end**
10 **end**
11 $x_{best} \leftarrow getBest(O) \; \forall \; f(x_{best}) \leq f(x), \forall \; x \in O$
12 **until** $O == \{\}$ *ou* $x_{final} \in V$;
13 **return** $path$

isso, o algoritmo entra em um *loop* onde primeiramente é verificado se o nó atual está no conjunto V, caso não esteja, ele é imediatamente adicionado à V e então todos os seus vizinhos são adicionados ao conjunto O de nós a serem visitados (linhas 6 a 10). Em seguida, é escolhido o nó de menor custo que esteja em O para ser visitado de acordo com a função de custo $f(n)$ (linha 11). Esse procedimento é repetido até que o ponto de objetivo x_{final} esteja no conjunto V ou então até O ficar vazio.

O algoritmo de Dijkstra faz uso de um *grid*. Desta forma, a qualidade dos resultados deste método está diretamente relacionada à discretização do mapa. Quanto menor a discretização, mais realista é a representação do terreno. Em contrapartida, a complexidade do algoritmo implementado é quadrática, deste modo, o tempo de execução aumenta consideravelmente de acordo com a discretização. É importante mencionar que, no caso de uma discretização muito grande, diversas características importantes do terreno podem não ser consideradas, fazendo com que o caminho retornado apresente na prática um custo maior do que o calculado.

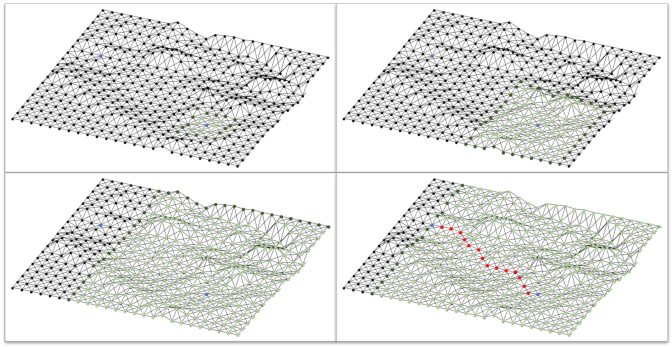

Figura 5. Progressão da busca pelo melhor caminho no algoritmo de Dijkstra.

3.3 Métricas de Mobilidade

Com o intuito de gerar caminhos de forma a maximizar a mobilidade do robô ao percorrer terrenos acidentados, diferentes métricas foram propostas. Tais métricas são utilizadas para alterar o cálculo dos custos associados aos caminhos, tanto entre nós da árvore utilizada pelo algoritmo RRT* quanto entre nós do grafo processado pelo algoritmo Dijkstra, resultando em caminhos com características consideravelmente diferentes.

Métrica de Distância Percorrida Custos calculados a partir da distância entre nós é a técnica mais comum ao se realizar o planejamento de caminhos, sendo muito utilizada em problemas onde se requer encontrar a trajetória mais curta disponível entre dois pontos. Aqui, o custo é calculado a partir da distância euclidiana entre dois pontos, como pode ser visto na expressão a seguir:

$$D_C = \sqrt{(x_{p_1} - x_{p_2})^2 + (y_{p_1} - y_{p_2})^2 + (h(p_1) - h(p_2))^2}, \quad (1)$$

onde os índices p_1 e p_2 denotam os dois nós sendo analisados. Dessa forma, o custo total de um caminho formado por certo arranjo de pontos é dado pela soma das distâncias individuais entre cada segmento:

$$C_1(\tau) = \sum_{n \in \tau} D_C(n), \quad (2)$$

onde τ é o trajeto formado pelo arranjo de pontos, $\tau = \{n_{inicial}, ..., n_i, ..., n_{final}\}$. Assim, o caminho escolhido será aquele que apresenta valor de custo total C_1 mínimo.

Métrica de Variação de Altura do Terreno Esta métrica tem como objetivo produzir caminhos com baixa variação de altura, que está associada a eficiência energética ao robô. Para isto, os custos são calculados a partir da norma do gradiente da função $h(x, y)$ do terreno, de forma que áreas mais regulares do ambiente recebem menor custo e são priorizadas no planejamento. A norma do gradiente para um certo ponto do espaço é representada por $\|\nabla h(x, y)\| \equiv \|\nabla h(p)\|$ e os custos, calculados entre dois pontos, formalmente são obtidos a partir da equação:

$$H_C(p_1, p_2) = \int_{p_1}^{p_2} \|\nabla h(p)\| \|dp\|, \quad (3)$$

uma integral de linha da norma do gradiente ao longo do segmento formado por p_1 e p_2. Devido à dificuldade de se efetuar este tipo de cálculo repetidamente, em termos de eficiência de processamento, o custo é obtido a partir de uma aproximação utilizando a expressão a seguir:

$$H_C(p_1, p_2) = \|p_1 - p_2\| \frac{\|\nabla h(p_1)\| + \|\nabla h(\frac{p_1+p_2}{2})\| + \|\nabla h(p_2)\|}{3}, \quad (4)$$

onde a média das normas do gradiente dos pontos p_1, p_2 e do ponto entre eles é calculada e multiplicada pela distância entre os pontos analisados, de forma a simular a integral de linha. Assim, o custo total de uma determinada trajetória é dado pela soma das normas do gradiente de cada segmento:

$$C_2(\tau) = \sum_{n \in \tau} H_C(n). \quad (5)$$

Com isto, o caminho escolhido será aquele que apresenta valor de custo total C_2 mínimo.

Métrica de Inclinação do Terreno A métrica de inclinação do terreno visa gerar caminhos que proporcionem maior estabilidade ao robô, escolhendo regiões do ambiente com menores inclinações de forma a reduzir o risco de tombamentos. Para realizar o cálculo dos custos, primeiramente é preciso determinar a inclinação do terreno nos pontos analisados. A partir de um valor Δ previamente definido (na ordem de 10^{-3}), três pontos ao redor do nó analisado são utilizados para determinar o plano do terreno naquele local. Com isso, é calculado o ângulo de inclinação

do terreno em um determinado ponto, com respeito à direção vertical. Dessa maneira, os custos são calculados a partir da expressão a seguir:

$$S_C = \theta_{p_1} + \theta_{p_2} + \theta_{(p_1+p_2)/2} + max(\theta_{p_1}, \theta_{p_2}, \theta_{(p_1+p_2)/2}), \quad (6)$$

onde $\theta_{p_1}, \theta_{p_2}, \theta_{(p_1+p_2)/2}$ são os ângulos referentes aos pontos p_1, p_2 e o ponto entre eles, respectivamente, sendo que também está presente uma função max, que retorna o maior ângulo entre os analisados e tem como objetivo dar mais peso para partes do caminho com maior inclinação, de forma que sejam evitadas. Dessa forma, o custo total para uma trajetória é dado pela soma das inclinações de cada segmento:

$$C_3(\tau) = \sum_{n \in \tau} S_C(n). \quad (7)$$

Com isto, o caminho escolhido será aquele que apresenta valor de custo total C_3 mínimo.

Métrica Combinada (Distância, Variação de Altura e Inclinação) A métrica combinada, que utiliza as métricas previamente apresentadas, procura fornecer caminhos que possuam um compromisso ótimo entre eficiência e estabilidade, agregando os custos de distância, altura e inclinação do terreno. É possível aplicar pesos ajustáveis a cada um dos três custos separadamente, influenciando qual deles deve ser predominante. Assim, os custos são calculados pela seguinte equação:

$$C_C = \beta_1 Dc + \beta_2 Hc + \beta_3 Sc, \quad (8)$$

onde Dc, Hc e Sc são os custos das métricas de distância, altura e inclinação, e as variáveis β_1, β_2 e β_3 são os pesos ajustáveis pelo operador, tal que $\beta_1, \beta_2, \beta_3 > 0$ e $\beta_1 + \beta_2 + \beta_3 = 1$. Portanto, o custo total para uma trajetória é dado pela soma dos custos combinados de cada segmento:

$$C_4(\tau) = \sum_{n \in \tau} C_C(n). \quad (9)$$

Com isto, o caminho escolhido será aquele que apresenta valor de custo total C_4 mínimo.

4. SIMULAÇÕES E RESULTADOS EXPERIMENTAIS

Os algoritmos planejadores RRT* e Djikstra são analisados por meio de simulações. O software *Matlab* é empregado para a implementação dos algoritmos, fornecendo caminhos de acordo com as diferentes métricas. Para o Dijkstra, um script já pronto obtido do site Mathworks foi utilizado como referência; o RRT* foi implementado pelos autores deste artigo; ambos os algoritmos estão disponíveis online. [1] Em seguida, os resultados são validados utilizando um simulador do EspeleoRobô (Cid et al., 2020), desenvolvido utilizando o *CoppeliaSim* junto com o *Robot Operating System* (ROS).

Para controlar o robô no ambiente simulado, foi usada a estratégia de campos vetoriais artificiais juntamente com a técnica de *feedback linearization*. O controle por campos vetoriais é baseado na definição de uma velocidade de comando composta por duas componentes: uma convergente, responsável por comandar o robô até uma curva de referência; e uma tangente, responsável por guiar o robô ao longo do caminho. Esta foi a mesma estratégia de controle adotada em (Amaral et al., 2020).

O modelo de terreno utilizado nas simulações é o de um campo de motocross localizado em Ouro Preto, Minas Gerais. O cenário virtual no CoppeliaSim (Fig. 6) foi obtido por fotogrametria. Um vídeo resumindo os experimentos está disponível online. [2]

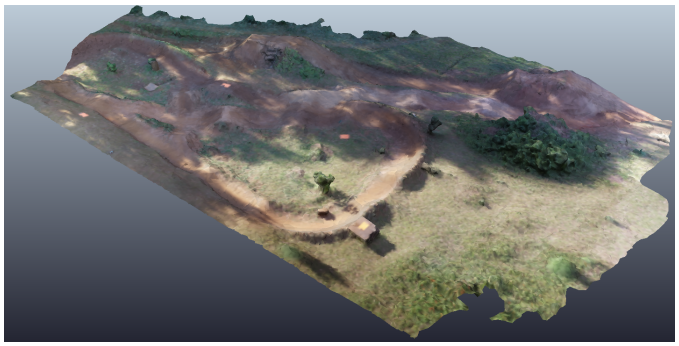

Figura 6. Cenário do campo de motocross para simulações no software CoppeliaSim.

Inicialmente, a métrica de distância foi empregada para analisar as variações de tempo de processamento e custos calculados conforme o número de iterações no caso do RRT*, e da discretização do mapa no caso do Dijkstra. Com isso, é possível estabelecer parâmetros adequados de funcionamento dos algoritmos que garantam um bom compromisso entre tempo gasto e qualidade dos resultados. Os experimentos foram realizados em um notebook com processador Intel Core i5 7300 HQ, 8GB de memória RAM, SSD de 240GB e sistema operacional Windows 10.

Para a comparação de tempo de processamento, foram realizadas cinco simulações com os mesmos parâmetros de iterações ou discretização (Fig. 7). É evidente a discrepância de tempo entre os dois algoritmos devido principalmente à custosa rotina de reconstrução de árvore do RRT*, além do fato do algoritmo implementado pelos autores em *Matlab* não estar completamente otimizado. Também é preciso evidenciar uma limitação de memória em relação ao algoritmo Dijkstra, devido às matrizes de adjacência empregadas, algo que também pode ser otimizado.

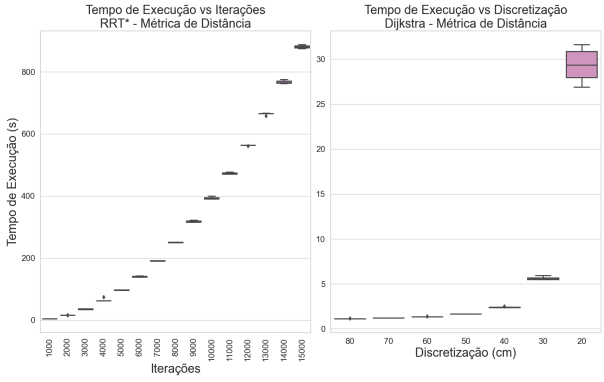

Figura 7. Comparação entre o tempo de execução dos algoritmos RRT* e Dijkstra.

[1] https://github.com/Douglas-VC/planning_algorithms

[2] https://www.youtube.com/watch?v=gp136l5lRdY

Já na Fig. 8 está a comparação dos custos de distância total calculados pelos planejadores. Novamente, os testes foram realizados cinco vezes com os mesmos parâmetros. Para o RRT*, nota-se uma curva decrescente da distância total, o que era esperado, visto que com o aumento do número de iterações o terreno é melhor amostrado, criando melhores possibilidades de caminhos. Para o Dijkstra, o oposto acontece, sendo que a distância total cresce com uma discretização menor. Isto é devido ao fato de que quanto menor a discretização, mais detalhes do terreno são obtidos, que acabam entrando no cálculo da distância total percorrida. Apesar de gerarem custos menores associados às métricas de mobilidade, discretizações maiores podem levar a modelos não representativos do terreno percorrido.

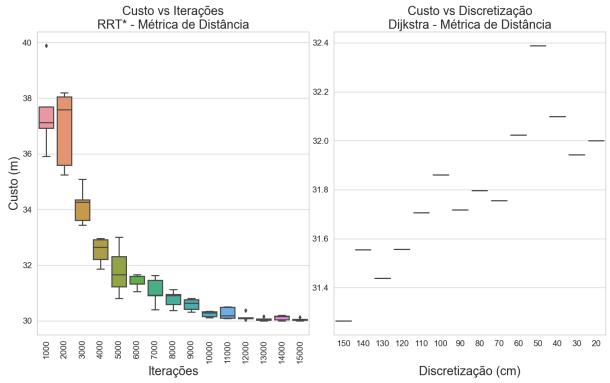

Figura 8. Comparação entre os custos de caminho dos algoritmos RRT* e Dijkstra.

O *boxplot* na Fig. 8 ilustra o comportamento probabilísticos do algoritmo RRT*, que pode apresentar grandes variações entre os caminhos calculados utilizando o mesmo número de iterações. O Dijkstra, em comparação, não apresenta este tipo de variação, sempre gerando resultados idênticos para uma mesma discretização.

Após a análise de ambos os resultados apresentados anteriormente, foi definido que para o restante das simulações seriam utilizadas 10000 iterações para o algoritmo RRT*, já que não foi observado decréscimo significativo no custo a partir deste número de iterações. Para o Dijkstra, a discretização escolhida foi de 30 cm, capaz de representar o terreno com fidelidade, além do baixo tempo de execução com respeito a discretizações menores.

Com o estabelecimento dos parâmetros utilizados, os algoritmos foram aplicados na otimização das demais métricas de mobilidade. Depois de obter todos os caminhos no *Matlab*, estes foram utilizados como referência para o EspeleoRobô no *CoppeliaSim* como forma de validação. A Fig. 9 mostra os caminhos calculados no *Matlab* (coluna da esquerda), e os percursos executados pelo robô no *CoppeliaSim* (coluna da direita).

Analisando os resultados obtidos no *Matlab* por ambos os algoritmos, nota-se a semelhança entre os caminhos gerados para cada métrica: os caminhos mais curtos tendem a se aproximar de uma reta; já os caminhos com menor variação de altura e inclinação apresentam características similares, passando por trechos mais planos do terreno; já as métricas combinadas buscam um compromisso ótimo entre as métricas. Também é possível perceber que o EspeleoRobô, utilizando o controle por campos vetoriais, conseguiu percorrer com precisão os caminhos no *CoppeliaSim*.

RRT* - Matlab / Coppelia			
Métrica	**Distância Total (m)**	**Altura Total (m)**	**Inclinação Total (graus)**
Distância	30,121/ 32,603	15,697/ 4,811	4329,90 (Ang. Máx 39,63)/ 1699,24 (Ang. Máx 39,75)
Altura	36,197/ 37,579	5,370/ 3,720	1675,27 (Ang. Máx 26,43)/ 661,48 (Ang. Máx 33,55)
Inclinação	35,680/ 36,304	6,051/ 4,077	1612,19 (Ang. Máx 26,32)/ 642,55(Ang. Máx 32,20)
Combinada	33,922/ 36,157	6,031/ 3,728	1953,91 (Ang. Máx 32,42)/ 701,90 (Ang. Máx 36,01)
Dijkstra - Matlab / Coppelia			
Métrica	**Distância Total (m)**	**Altura Total (m)**	**Inclinação Total (graus)**
Distância	31,944/ 34,628	10,318/ 4,670	3297,49 (Ang. Máx 56,36)/ 964,68 (Ang. Máx 38,64)
Altura	39,163/ 40,203	5,297/ 3,638	1522,69 (Ang. Máx 26,72)/ 657,80 (Ang. Máx 32,04)
Inclinação	40,848/ 40,446	5,578/ 3,661	1503,57 (Ang. Máx 25,81)/ 632,75 (Ang Máx 31,63)
Combinada	35,193/ 36,576	5,614/ 3,668	1904,28 (Ang Máx 26,50)/ 719,23 (Ang. Máx 32,93)

Tabela 1. Comparação entre os parâmetros referentes a cada algoritmo e diferentes métricas.

A Tabela 1 contém os custos totais de cada um dos resultados obtidos, possibilitando a comparação entre os algoritmos. Cabe observar que ao tentar otimizar métricas individuais, os planejadores obtiveram caminhos com menor custo em relação às outras métricas adotadas. A métrica com custos combinados, que procura um compromisso das demais, apresentou valores intermediários, como esperado.

Comparando os custos estimados no *Matlab* entre os dois algoritmos, o RRT* apresentou melhores resultados para a métrica de distância, enquanto o Dijkstra teve pequena vantagem com respeito às métricas de variação de altura e inclinação do terreno. Essa relação também foi observada nos resultados obtidos com o *CoppeliaSim*, com custos de distância e inclinação máxima levemente superiores. Em geral, ao analisar todos os resultados obtidos, é possível observar a conformidade entre os resultados obtidos entre os algoritmos, assim como a precisão do robô ao percorrer os caminhos no terreno virtual.

5. CONCLUSÕES

Este artigo apresentou um estudo sobre os algoritmos de planejamento de caminhos RRT* e Dijkstra, aplicados a terrenos acidentados de forma a otimizar diversas métricas de mobilidade. As análises apresentadas serão importantes para o desenvolvimento do sistema de navegação autônomo do EspeleoRobô. Os planejadores foram implementados no *Matlab*, gerando caminhos com características desejadas de acordo com a métrica adotada. Esses caminhos foram verificados com o *CoppeliaSim* em conjunto com o ROS, utilizados como referência pelo EspeleoRobô para percorrer um terreno acidentado virtual. Nestes experimentos, os tempos de execução do algoritmo de Dijkstra foram consideravelmente menores que os da implementação atual do algoritmo RRT*. Entretanto, algoritmos probabilísticos apresentam melhor desempenho em ambientes extensos onde a quantidade de nós do mapa pode inviabilizar a

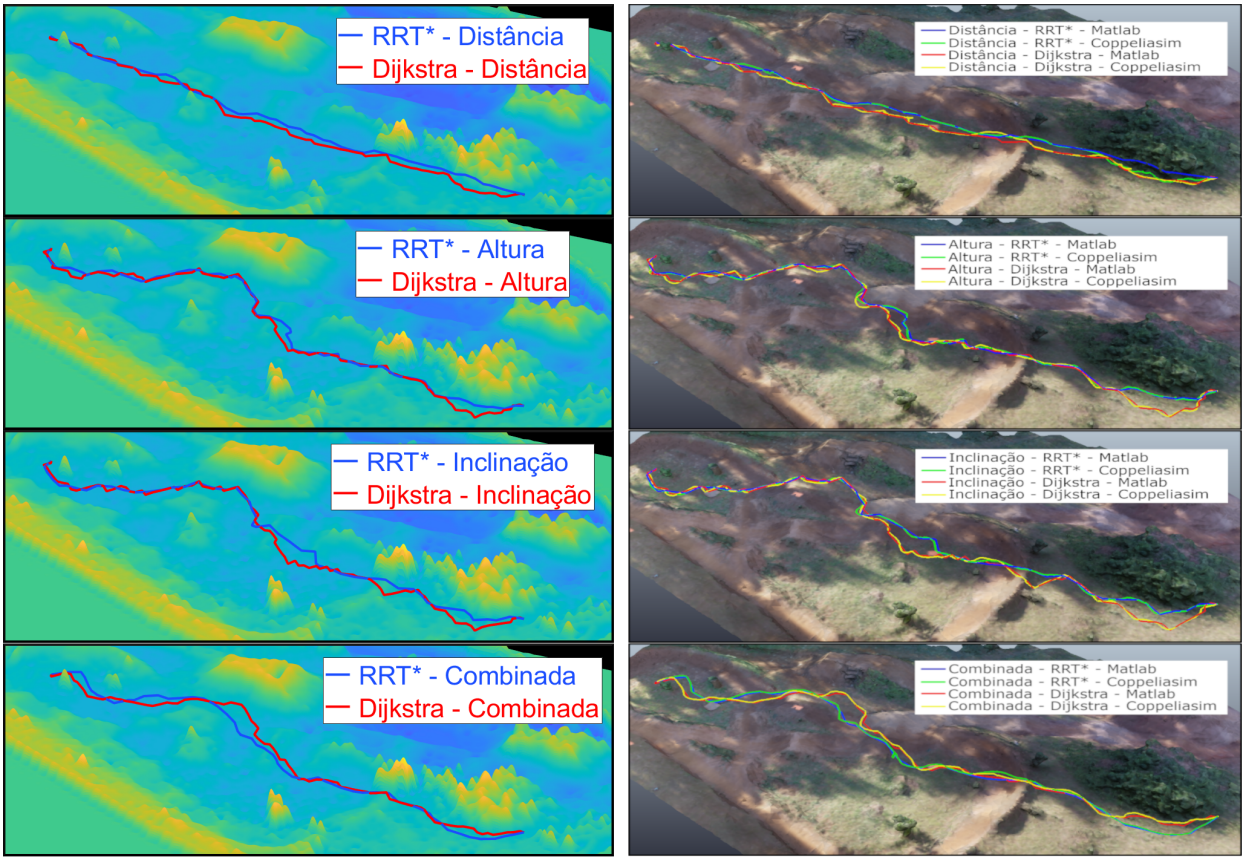

Figura 9. Comparação entre os caminhos gerados pelos algoritmos RRT* e Dijkstra no *Matlab* e percorridos no *CoppeliaSim* para cada uma das métricas de mobilidade.

utilização de métodos tradicionais de planejamento determinístico.

Trabalhos futuros incluem a implementação e otimização desses algoritmos em *Robot Operating System* (ROS), com objetivo de embarca-los no EspeleoRobô de forma a possibilitar a execução de experimentos de campo, onde serão testadas as capacidade de planejamento em ambientes reais e eficiência em termos de tempo de processamento e memória utilizada, assim como novas métricas para minimizar, por exemplo, o gasto energético. Além disso, outros algoritmos de exploração, como o A*, também serão implementados e embarcados, compondo uma biblioteca de algoritmos para planejamento de caminhos do EspeleoRobô.

REFERÊNCIAS

Amaral, I., Fany, C., Simon, D., Matos, L., Araújo, Á., Leão, L., Rezende, A., Azpúrua, H., Pessin, G., and Freitas, G. (2020). Sistema de alertas e operação assistida de um robô para a inspeção de ambientes confinados - EspeleoRobô. In *Anais do Congresso Brasileiro de Automática 2020*. sbabra.

Azpúrua, H., Rezende, A., Potje, G., da Cruz Júnior, G.P., Fernandes, R., Miranda, V., de Resende Filho, L.W., Domingues, J., Rocha, F., de Sousa, F.L.M., de Barros, L.G.D., Nascimento, E.R., Macharet, D.G., Pessin, G., and Freitas, G.M. (2021). Towards semi-autonomous robotic inspection and mapping in confined spaces with the EspeleoRobô. *J. of Intelligent & Robotic Systems*, 101(4).

Burgard, W., Hebert, M., and Bennewitz, M. (2016). World modeling. In *Springer handbook of robotics*, 1135–1152. Springer.

Choset, H.M., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S. (2005). *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

Cid, A., Nazario, M., Sathler, M., Martins, F., Domingues, J., Delunardo, M., Alves, P., Teotonio, R., Barros, L.G., Rezende, A., Miranda, V., Freitas, G., Pessin, G., and Azpurua, H. (2020). A simulated environment for the development and validation of an inspection robot for confined spaces. In *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR)*. IEEE.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. (2001). Introduction to algorithms second edition. *The Knuth-Morris-Pratt Algorithm*.

Hart, P.E., Nilsson, N.J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2), 100–107.

Heckbert, P.S. and Garland, M. (1999). Optimal triangulation and quadric-based surface simplification. *Computational Geometry*, 14(1-3), 49–65.

Hornung, A., Wurm, K.M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3), 189–206.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics research*, 30(7), 846–894.

Santos, A.S., Azpurua, H.I., Pessin, G., and Freitas, G.M. (2018). Path planning for mobile robots on rough terrain. In *2018 Latin American Robotic Symposium*. IEEE.

Santos, A.S., Azpúrua, H., Pessin, G., and Freitas, G. (2019). Planejamento de caminhos para robôs móveis em ambientes acidentados. *SBAI - Simpósio Brasileiro de Automação Inteligente*.

Siciliano, B. and Khatib, O. (2016). *Springer handbook of robotics*. springer.