Pontifícia Universidade Católica
do Rio de Janeiro

Arnaldo Roberto Rady Peron

Comparação de Técnicas de Planejamento de Trajetórias para Robôs Móveis Autônomos

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção de grau de Mestre em Engenharia Mecânica pelo Programa de Pós-Graduação em Engenharia Mecânica, do Departamento de Engenharia Mecânica da PUC-Rio.

Orientador: Prof. Marco Antonio Meggiolaro

Rio de Janeiro
Julho 2020

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Arnaldo Roberto Rady Peron

Comparação de Técnicas de Planejamento de Trajetórias para Robôs Móveis Autônomos

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Engenharia Mecânica da PUC-Rio. Aprovada pela Comissão Examinadora abaixo:

Prof. Marco Antonio Meggiolaro
Orientador
Departamento de Engenharia Mecânica – PUC-Rio

Prof. Ivan Fabio Mota de Menezes
Departamento de Engenharia Mecânica – PUC-Rio

Prof. Mauro Speranza Neto
Departamento de Engenharia Mecânica – PUC-Rio

Rio de Janeiro, 30 de Julho de 2020

Arnaldo Roberto Rady Peron

Graduou-se em Engenharia Mecânica com ênfase em Mecatrônica pela Pontifícia Universidade Católica de Minas Gerais (2004), especializando-se em Mecatrônica pela POLI – UFRJ (2016). Possui longa experiência na indústria, atuando com projetos mecânicos, controle e instrumentação. Suas áreas de interesse abrangem robótica, sistemas mecatrônicos e projetos mecânicos.

Peron, Arnaldo Roberto Rady

Comparação de técnicas de planejamento de trajetórias para robôs móveis autônomos / Arnaldo Roberto Rady Peron ; orientador: Marco Antonio Meggiolaro. – 2020.

166 f. : il. color. ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Mecânica, 2020.

Inclui bibliografia

1. Engenharia Mecânica – Teses. 2. Planejamento de trajetórias. 3. Robótica móvel. 4. Robôs autônomos. 5. Simulação. I. Meggiolaro, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Mecânica. III. Título.

CDD: 621

À memória de meu tio Paulo Rady e minha avó Arlinda Neves Rady.

Agradecimentos

Agradeço a Deus, pela oportunidade e por ter sido meu suporte.

Aos meus pais, Mário e Sara, pela educação e exemplo.

À minha noiva, Nicole Rousseau, pelo enorme apoio, incentivo e paciência.

Ao professor Marco Antonio Meggiolaro, por ter tão gentilmente aceitado me orientar e por todo o apoio e dedicação a mim concedidos durante esse período.

Aos professores Ivan Menezes e Mauro Speranza, por aceitarem o convite de compor a banca e por toda a ajuda ao longo do mestrado.

A todos os demais professores de quem tive a honra de ser aluno: Marcelo Dreux, Rubens Sampaio, Arthur Braga, Carlos Almeida, Hans Weber e José Freire.

Aos funcionários do Departamento de Engenharia Mecânica.

Aos colegas Diego Rosa, João Virgolino e Felipe Lopes, cujos auxílios na elaboração deste trabalho foram essenciais.

Aos demais colegas do LabRob, com quem tive o prazer de conviver. Todos me ajudaram de alguma forma: Fischer, Gustavo, Anna Rafaela, Junior, Rafael e Vivian.

A todos os colegas com quem cursei disciplinas, especialmente Bárbara Lavour, Thiago Moreira e Ricardo Takahashi, que tanto me ajudaram e também Ricardo Fernandes, que nunca me faltou com uma palavra amiga.

À pequena Selina, por estar sempre ao meu lado, mesmo nas longas noites de estudo e trabalho.

Ao CNPq e à PUC-Rio, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

Resumo

Peron, Arnaldo Roberto Rady; Meggiolaro, Marco Antonio. **Comparação de Técnicas de Planejamento de Trajetórias para Robôs Móveis Autônomos**. Rio de Janeiro, 2020. 166p. Dissertação de Mestrado – Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro.

A robótica móvel tem desempenhado um papel cada vez maior na sociedade moderna. Para cumprir suas tarefas, usualmente é necessário que os robôs móveis sejam capazes de navegar autonomamente pelo ambiente, partindo de um determinado local e chegando a um objetivo, evitando colisões. Uma etapa importante desse processo envolve a decisão dos caminhos a serem percorridos pelo robô ao longo de um mapa conhecido, que é executada por um programa chamado planejador de trajetórias. Muitos planejadores estão disponíveis na literatura, cada um com suas particularidades e aplicações. Esta dissertação apresenta um estudo comparativo entre cinco planejadores de trajetórias: Algoritmo de Dijkstra, *Dynamic Programming*, *Probabilistic Roadmaps* (PRM), *Rapidly-exploring Random Trees* (RRT) e *Hybrid A**, no qual são avaliados os tempos de processamento, os números de passos e os comprimentos das trajetórias resultantes. O estudo é executado por meio de simulações no ambiente MATLAB em três mapas diferentes, com objetivos diversos: o primeiro avalia a capacidade de se planejar uma trajetória em um ambiente complexo com múltiplos caminhos; o segundo testa a capacidade do planejador de evitar maiores obstáculos e lidar com passagens estreitas; e o último mapa, composto por longas retas e curvas abertas, tem o objetivo de avaliar o comportamento dos planejadores em trajetórias mais extensas. Cada planejador é submetido a séries de simulações em cada mapa, sendo que em cada uma alteram-se também suas características, para uma análise de sensitividade. Os melhores resultados de cada planejador para cada critério permitem identificar as principais vantagens e desvantagens de cada um.

Palavras-Chave

Planejamento de Trajetórias; Robótica Móvel; Robôs Autônomos; Simulação.

Abstract

Peron, Arnaldo Roberto Rady; Meggiolaro, Marco Antonio. **Comparison of Path Planning Techniques for Autonomous Mobile Robots**. Rio de Janeiro, 2020. 166p. Dissertação de Mestrado – Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro.

Mobile robotics has been playing an increasing role in modern society. To accomplish their tasks, these robots are usually required to autonomously navigate along the environment, starting from a certain location and reaching a goal while avoiding collisions. An important step in this process is finding a trajectory within a known map that the robot should follow, which is calculated by a program called path planner. There are many planners described in the literature, each with its own particularities and applications. This work presents a comparative study among five path planners: Dijkstra's Algorithm, Dynamic Programming, Probabilistic Roadmaps (PRM), Rapidly-exploring Random Trees (RRT) and Hybrid A*. The evaluations are based on computer processing times, number of required steps, and resulting trajectory lengths. The study is carried out through simulations in the MATLAB environment on three different maps: the first assesses the ability to plan a trajectory in a complex environment and with multiple paths; the second map evaluates the behavior in the presence of larger obstacles and narrow passages; and the third is composed of long straight lines and open curves, aimed to assess the behavior of the planners along longer trajectories. Each planner is subjected to a series of simulations on each map. Moreover, the characteristics of each planner are changed in each map for a sensitivity analysis. The best results for each planner and criterion are then used to compare the advantages and disadvantages of the studied approaches.

Keywords

Path Planning; Mobile Robotics; Autonomous Robots; Simulation.

Sumário

Lista de Figuras

Lista de Abreviações

ACO *Ant Colony Optimization*

AG Algoritmo Genético

APF *Artificial Potential Field*

AUV *Unmanned Underwater Vehicle*

DWA *Dynamic Window Approach*

FCE *Free Configuration Eigen-Spaces*

ISO *International Organization for Standardization*

LLL *Large Label Last*

PRM *Probabilistic Roadmaps*

PSO *Particle Swarm Optimization*

RNA Redes Neurais Artificiais

RRT *Rapidly-exploring Random Trees*

SA *Simulated Annealing*

SLAM *Simultaneous Localization And Mapping*

SLF *Small Label First*

SUAV *Small Unmanned Aerial Vehicles*

tp tempo de processamento

UGV *Unmanned Ground Vehicle*

UCAV *Unmanned Combat Aerial Vehicle*

VANT Veículos Aéreos Não Tripulados

Eu sou o Senhor teu Deus, que te tirei da terra do Egito, da casa da servidão;
Não terás outros deuses diante de mim.

Deuteronômio 5; 6,7.

1
Introdução

1.1
Motivação

Os robôs, em suas várias aplicações, exercem um papel importante na sociedade, sobretudo nos meios industriais, onde se tornaram populares por serem flexíveis, precisos e não compartilhar dos mesmos requisitos de conforto, segurança e descanso que os seres humanos [1]. O termo “robô” foi cunhado no teatro pelo autor tcheco Karel Capec, em 1922, para a peça “Robôs Universais de Rossum” (em tradução livre) e remete à palavra *rabota* que quer dizer servo [1]. Voltando às aplicações reais, a ISO (*International Organization for Standardization*), por meio de sua norma 10218, define um robô como "uma máquina manipuladora com vários graus de liberdade controlada automaticamente, reprogramável, multifuncional, que pode ter base fixa ou móvel para utilização em aplicações de automação industrial" [2].

A ciência que estuda os robôs e suas aplicações é chamada de robótica. O termo foi criado pelo autor de ficção científica Isaac Asimov em 1950, em sua série de crônicas que foram compiladas no livro “Eu, Robô”, na qual ele estabeleceu as três leis da robótica, que ainda são utilizadas como parâmetros éticos por engenheiros e pesquisadores [3].

Sendo uma ciência que vem se desenvolvendo desde 1946, quando George Devol criou seu dispositivo de reprodução chamado de “controlador magnético”, os robôs não se limitam às bases fixas. A robótica móvel é por si só um campo de estudos vasto [1], que começou em 1951 quando William Grey criou o que pode ser considerado o primeiro robô móvel da história, o *Turtle*: um sistema eletromecânico que simulava os movimentos de uma tartaruga, que é apresentado na Figura 1.1(a) [4]. Há uma variedade de robôs móveis, mesmo dentre os terrestres, que podem ter rodas ou pernas em números variados em ambos os casos, como os UGVs (*Unmanned Ground Vehicle*) e ainda os aéreos como os *drones* e VANTs (Veículos Aéreos Não Tripulados) ou aquáticos como os AUVs (*Unmanned Underwater Vehicle*). Exemplos deste tipo de veículos podem ser encontrados na Figuras 1.1.

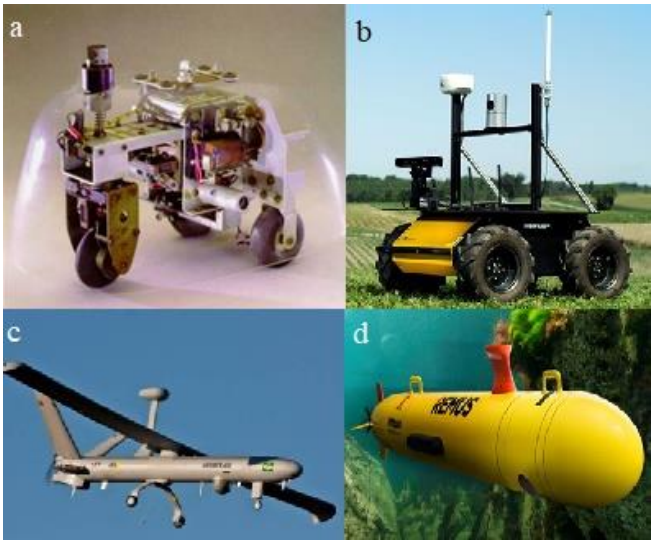


Figura 0.1 - (a) Sistema eletromecânico *Turtle*; (b) UGV Husky, da Clearpath Robotics; (c) VANT da Força Aérea Brasileira; (d) UAV Remus 100, da Hydroid.

Um dos objetivos mais nobres da robótica é a criação de robôs autônomos, que tenham como "entrada" descrições de tarefas de alto nível (que especifiquem "o que fazer" e não "como fazer") e as executem sem intervenção humana. O desenvolvimento destas tecnologias envolve questões de raciocínio, percepção e controle, o que levanta muitos problemas. Um deles é o planejamento de movimento, que pode ser descrito como a maneira na qual o robô executa seus movimentos para atingir seu objetivo. Sendo que, por definição, um robô realiza tarefas movendo-se no mundo real, um robô autônomo precisa ter a capacidade de planejar seu próprio movimento [5].

No caso dos robôs móveis, a habilidade mais importante é a de navegação. O robô deve ser capaz de se mover ambientes conhecidos ou desconhecidos, baseando-se nos dados obtidos por seus sensores, para alcançar seus objetivos. O robô deve, então, confiar em seus demais aspectos, como a percepção (uso de sensores para obter dados do ambiente), localização (conhecimento de sua posição e configuração), cognição (capacidade de decidir o que fazer) e controle de movimento (capacidade de calcular as forças de entrada dos atuadores a fim de executar a trajetória desejada). Uma vez que, de forma geral, o caminho que o robô móvel precisa percorrer de sua posição inicial até o objetivo final não é direto, faz-se necessária a aplicação de técnicas de planejamento de trajetórias. A navegação de robôs móveis divide-se nas seguintes tarefas:

- Criação de um modelo do "mundo" (ambiente) na forma de um mapa.

- Cálculo de uma trajetória sem colisões desde uma posição inicial até uma posição de destino.
- Movimento ao longo da trajetória calculada, evitando colisões com obstáculos [6].

Há muitas técnicas de planejamento de trajetórias que se aplicam à robótica móvel, com abordagens, características e resultados variados. Seu conhecimento e as soluções delas provenientes é de suma importância para o projeto de robôs móveis visto que algumas podem demandar maiores capacidades de processamento que outras, assim como há aquelas podem se adequar mais a um determinado ambiente. Há ainda o caso em que existe a necessidade de se implementar alguma técnica de planejamento em conjunto com alguma outra técnica relacionada à outra área de estudo da robótica móvel, como por exemplo o SLAM (*Simultaneous Localization and Mapping*), uma técnica auto localização e mapeamento que é fundamental para que o robô tenha autonomia completa [85].

No próximo capítulo são apresentadas técnicas estado da arte para planejamento de trajetórias, assim como outros estudos comparativos entre elas.

1.2 Objetivos

O objetivo deste trabalho é desenvolver um estudo comparativo entre cinco planejadores de trajetórias, selecionados como os mais conhecidos ou utilizados na atualidade em robótica móvel - Algoritmo de Dijkstra, *Dynamic Programming*, PRM, RRT e *Hybrid A** - avaliando três parâmetros: tempo de processamento, comprimento de trajetória e quantidade de passos, em três ambientes virtuais distintos. O intuito é fornecer ao leitor dados para que se tenha condições de se avaliar qual planejador se enquadra melhor em suas necessidades, levando em conta os aspectos avaliados e os ambientes. Os estudos são conduzidos por meio de simulações no ambiente MATLAB, utilizando planejadores já desenvolvidos para a sua sintaxe.

Um gráfico em [11], de 2018, revela que na base de dados *Engineering Village* a quantidade de trabalhos envolvendo planejadores de trajetórias clássicos é muito inferior aos classificados como de inteligência artificial. Todavia, segundo Patle et al. (2019), em [9], os planejadores clássicos são utilizados como base para

os planejadores híbridos e consultando a lista de trabalhos publicados pelo Kavraki Lab, da *Rice University* (EUA) [73], um dos principais laboratórios de pesquisa de planejamento de trajetórias no mundo, observa-se que os planejadores clássicos ainda estão sendo desenvolvidos, sobretudo os baseados em amostragem, focando em otimização.

Logo, há ainda necessidade de estudos envolvendo planejadores clássicos que tenham sido desenvolvidos para plataformas modernas como a *Motion Planning Toolbox* do MATLAB, que utiliza o RRT e *Hybrid A**.

Há também o objetivo se contribuir com a pesquisa científica, apresentando um estudo que envolva o planejador *Hybrid A** pois, até o fechamento deste trabalho, não foi identificado na base de dados "Periódicos CAPES" algum estudo comparativo que o envolvesse. Este trabalho também é o primeiro, até o momento de sua conclusão, a executar um estudo comparativo entre os cinco planejadores de trajetórias mencionados.

A seguir, é apresentada a estrutura da dissertação.

1.3 Estrutura da Dissertação

Esta dissertação é dividida em 6 capítulos conforme a seguinte estrutura:

- O **Capítulo 2** apresenta uma revisão bibliográfica com planejadores estado da arte, assim como outros estudos comparativos entre planejadores.
- O **Capítulo 3** apresenta os fundamentos teóricos dos planejadores de trajetórias que serão estudados, seus algoritmos e princípios de operação.
- O **Capítulo 4** apresenta o sistema no qual as simulações foram executadas, os programas utilizados, metodologia, ambientes virtuais e equipamentos.
- O **Capitulo 5** apresenta os resultados obtidos nas simulações no primeiro mapa, Labirinto e um estudo comparativo entre eles.
- O **Capítulo 6** apresenta os resultados obtidos nas simulações no segundo mapa, Labirinto Estreito e um segundo estudo comparativo.
- O **Capítulo 7** apresenta os resultados obtidos nas simulações no terceiro mapa, Ziguezague e seu respectivo estudo comparativo.
- O **Capítulo 8** apresenta as conclusões finais e sugestões para trabalhos futuros.

2
Revisão Bibliográfica

A robótica trata da automação de sistemas mecânicos que possuem sensores, atuadores e recursos de computação. Ela necessita de algoritmos capazes de converter instruções de tarefas de alto nível, fornecidas por seres humanos, em descrições de baixo nível (para o robô) de como se locomover. Este tipo de problema é abordado pelo planejamento de trajetória, ou planejamento de movimento, como também é conhecido. Uma forma de se entendê-lo é imaginar o "problema do carregador de piano", no qual deseja-se mover um piano de cauda entre cômodos de uma casa. Os carregadores devem, a cada momento, planejar as posições do piano para que o mesmo possa ser deslocado sem colisões. Da mesma forma, o planejamento de movimento de um robô geralmente ignora a dinâmica e demais restrições diferenciais, concentrando-se principalmente nas translações e rotações necessárias para mover o "piano" (ou robô) [7].

Mesmo antes do advento de robôs móveis, já se estudava planejamento de trajetórias na robótica devido à sua aplicação nos manipuladores industriais. A formulação do problema de planejamento de trajetórias para um manipulador de seis graus de liberdade é muito mais complexa do que formular o mesmo tipo de problema para um robô móvel diferencial operando em um ambiente plano. Por isso, devido ao menor número de graus de liberdade, os algoritmos de planejamento de trajetórias utilizados pelos robôs móveis tendem a ser mais simples que os aplicados aos manipuladores. Também deve ser considerado que os manipuladores industriais, geralmente, operam na maior velocidade possível, a fim de elevar o rendimento da linha de produção em que atuam. Logo, a dinâmica, e não apenas a cinemática, também é significativa, complicando ainda mais o planejamento e a execução da trajetória. Entretanto, muitos robôs móveis da atualidade operam em velocidades relativamente baixas, de modo que a dinâmica raramente é considerada durante o planejamento de trajetórias, simplificando a formulação do problema [8].

Planejamento de trajetórias é o tópico mais estudado no campo da robótica móvel e inúmeras técnicas têm sido desenvolvidas em todo o mundo. Estas técnicas podem ser enquadradas em duas categorias: navegação global, que diz respeito à capacidade do robô em identificar a posição dos elementos em relação ao seu eixo de referência e mover-se em direção ao objetivo, exigindo informação prévia do

ambiente, posições dos obstáculos e objetivos; e navegação local, que trabalha com a identificação das condições dinâmicas do ambiente e não exige informações prévias sobre ele [9].

Diferentes trabalhos abordando planejamento de trajetórias dividem as técnicas em outras categorias. Patle et al. (2019) em [9], por exemplo, classificam as técnicas em abordagem clássicas ou reativas. Mohd e Mohanta (2018), por sua vez, as classificam como abordagem clássica e abordagem por inteligência artificial ou heurística [10]. Já Zhang, Lin e Chen (2018) em [11], mantém a classificação tradicional e acrescenta subdivisões nas quais as técnicas se enquadram segundo o esquema abaixo:

- Planejamento Global de Trajetórias
    - Modelagem Ambiental
        - Abordagem por Espaço Configurado
            - Grafo de Visibilidade
            - Diagrama de Voronoi
            - Grafo de Tangente
        - Abordagem por Espaço Livre
        - Abordagem por Decomposição Celular
        - Método do Mapa de Rotas Probabilístico
    - Busca por Trajetória
        - Abordagem Heurística
            - Algoritmo de Dijkstra
            - Algoritmo A*
            - Algoritmo D*
        - *Dynamic Programming*
        - Algoritmos Probabilísticos e de Inteligência Artificial
            - Redes Neurais Artificiais
            - Algoritmos Genéticos
            - Otimização de Colônia de Formigas
            - Otimização de Enxame de Partículas
            - Recozimento Simulado
- Planejamento Local de Trajetórias
    - Busca por Trajetória

- Campo Potencial Artificial
- Método da Decomposição de Comportamento
- Método de Aprendizagem Baseado em Casos
- Algoritmo de Rolagem de Janelas
- Algoritmos de Inteligência Artificial ou Probabilísticos (as mesmas técnicas aplicadas ao planejamento global)

Vale notar que os algoritmos de inteligência computacional aparecem tanto na categoria de planejadores globais quanto locais, pois apresentam critérios que atendem ambas categorias.

A seguir, são apresentados os algoritmos de planejamento supracitados.

2.1 Planejadores Globais

A primeira subdivisão, a modelagem ambiental, é uma técnica que auxilia na compreensão das variáveis ambientais, reduzindo planejamento e cálculos desnecessário. Na abordagem por espaço estruturado, o robô é reduzido a um ponto e os obstáculos são dimensionados de forma a compensar esta redução. Assim, o robô pode se mover livremente entre os obstáculos evitando colisões [11]. As três técnicas referentes a esta abordagem são vistas a seguir.

O método do Grafo de Visibilidade é um dos mais antigos para planejamento de trajetórias e Latombe [5], em 1991, o classifica dentro de uma abordagem de mapa de rotas. O método utiliza polígonos para representar obstáculos, cujos vértices se conectam por linhas, desde que estejam visíveis e estes também se conectem aos pontos inicial e final. Dado este conjunto de linhas, o algoritmo pode selecionar um caminho entre o ponto inicial e o desejado [5]. A Figura 2.1 apresenta um exemplo de grafo de visibilidade.

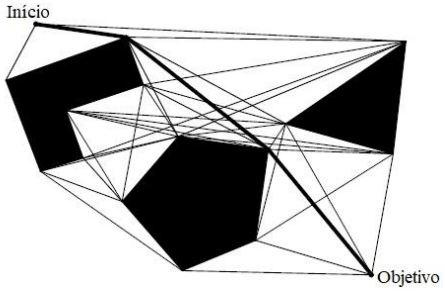


Figura 0.1 - Grafo de Visibilidade.

O Diagrama de Voronoi, apresentado na Figura 2.2, foi desenvolvido por Canny e Donald [13] em 1988. Esta técnica consiste em construir linhas a partir de todos os pontos que sejam equidistantes de dois ou mais obstáculos, incluindo os limites do mapa. Sendo q_i e q_o as posições inicial e objetivo respectivamente, as mesmas são mapeadas para dentro do diagrama, assumindo as nomenclaturas q'_i e q'_o, traçando, a partir de cada ponto q_i e q_o, uma linha ao longo da qual sua distância aos limites dos obstáculos aumente mais rapidamente [8].

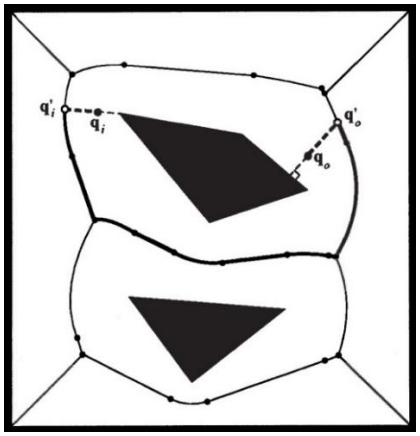


Figura 0.2 - Diagrama de Voronoi.

No método do grafo de tangentes para navegação de robôs móveis, conforme proposto por Liu e Arimoto [14] em 1991, os nós, formados pelos vértices das figuras, representam pontos tangentes nos limites dos obstáculos e as arestas representam tangentes comuns dos obstáculos, sem colisões, ou segmentos convexos dos limites entre os pontos tangentes. A Figura 2.3 apresenta um modelo de grafo de tangentes.

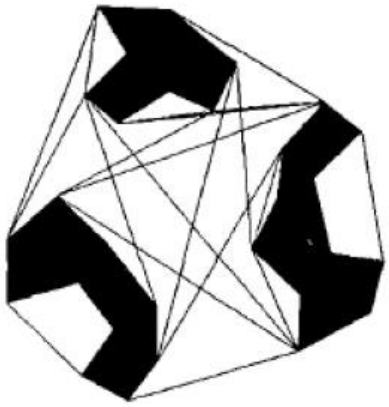


Figura 0.3 - Grafo de Tangentes.

A segunda abordagem do subgrupo da modelagem ambiental é a dos espaços livres, que foi desenvolvida por Habib e Asama [15] em 1991. A técnica se baseia no conceito de elos livres para construir um espaço livre entre os obstáculos do ambiente, enquadrado nos termos de uma região convexa livre. Gera-se então um grafo chamado MAKLINK, responsável por criar trajetórias livre de colisões. O grafo, por usa vez, utiliza os pontos médios dos elos livres comuns que se encontram na região convexa livre como pontos de passagem. A Figura 2.4 ilustra uma trajetória livre de colisões baseada em um grafo MAKLINK.

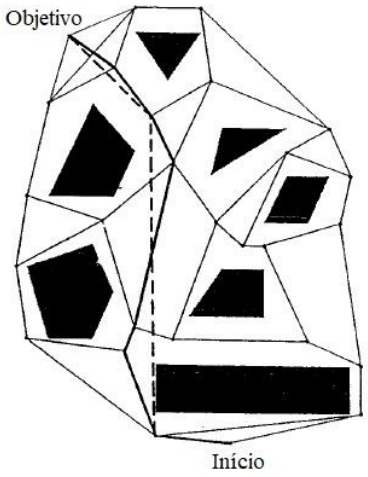
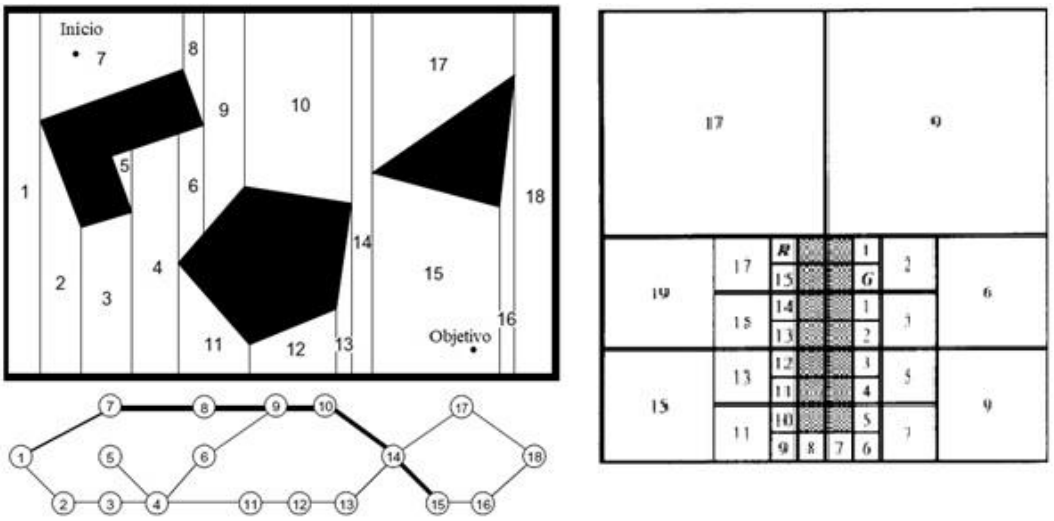


Figura 0.4 - Grafo MAKLINK.

A abordagem por decomposição celular, apresentada por Brooks e Lozano-Perez [16] em 1985, propõe a divisão, ou decomposição, do espaço livre em regiões na forma de grades, chamadas células, as diferenciando das regiões ocupadas. O algoritmo procura por células adjacentes e constrói um grafo que as conecte em sequência, identificando em quais células estão os pontos inicial e objetivo. A partir deste grafo, um algoritmo de busca pode estabelecer um caminho por entre as células como, por exemplo, passando pelos pontos médios das fronteiras ou seguindo uma parede em linha reta [8]. A decomposição celular pode ser exata - na qual o espaço resultante da divisão do espaço livre é igual ao original - ou aproximada, na qual as células possuem forma pré-determinada e quando alguma delas contém obstáculos, se subdivide em quatro outras menores, formando uma estrutura chamada *quadtree* [11]. A Figura 2.5 apresenta exemplos de grafos de decomposição celular exata e aproximada respectivamente.



(a) (b)

Figura 0.5 - Método da decomposição celular: (a) exata e (b) aproximada.

O método do mapa de rotas probabilístico ou PRM (*Probabilistic Roadmaps)*, conforme estabelecido por Kavraki et al [17] em 1996, propõe a construção de um grafo de rotas não direcionado a partir de amostragem aleatória, tal que R = (N, E), onde "N" são os nós obtidos e "E" são as arestas que ligam esses nós. Sendo *i* o ponto inicial e *o* o objetivo, o planejamento da trajetória é obtido pesquisando a sequência de arestas que conectam diretamente *i* a *o* no grafo de rotas não direcionado [11]. Esta metodologia é vista com maiores detalhes no Capítulo 2. A Figura 2.6 apresenta um exemplo de PRM.

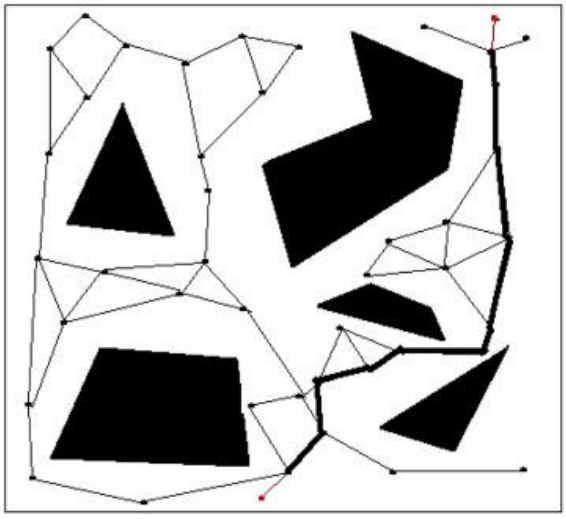


Figura 0.6 - Mapa de Rotas Probabilístico.

Outra categoria de planejador baseado em amostragem é o *Rapidly-exploring Random Trees* (RRT), apresentado por LaValle [18] em 1998, representado pela imagem da Figura 2.7. Neste planejador, os nós vão sendo incrementados a partir da posição inicial até atingir a posição final, ou vice-versa, se conectando na forma de ramos de uma árvore, da seguinte forma: um ponto é selecionado aleatoriamente no espaço configurado e verifica-se se ele se encontra em um espaço livre. Caso positivo, ocorre uma tentativa de conectá-lo ao vértice mais próximo da árvore, obedecendo-se o seguinte critério: se a distância entre os nós for menor ou igual a um determinado valor, ele se conecta à árvore, senão, um novo nó, localizados entre eles, é selecionado e este sim é conectado à árvore [19].

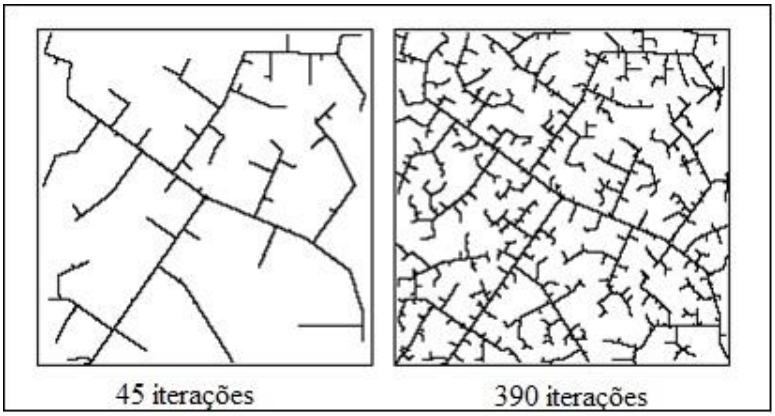


Figura 0.7 - *Rapidly-exploring Random Trees*.

A segunda subdivisão dos planejadores globais são os algoritmos de busca por trajetórias, que também se dividem em abordagem heurística e inteligência artificial. Três exemplos clássicos da abordagem heurística são os algoritmos de Dijkstra, A* (*A star*) e D* (*D star*). O Algoritmo de Dijkstra, proposto por E.W. Dijkstra [20] em 1959 é, segundo Pettie e Ramachandran [86], um tipo de *Minimum Spanning Tree* (MST) e visa resolver o problema da trajetória mais curta em um grafo direcionado que possui arestas, formadas por um par ordenado de dois vértices. Tais arestas possuem valores descritos por uma função de peso. O algoritmo mantém dois conjuntos de vértices chamados A e B, no qual A é o conjunto inicial e se encontra vazio. A cada iteração, um vértice em B é movido para A, de forma que o vértice selecionado deve garantir que a soma dos pesos de todas as arestas desde o ponto inicial até o ponto final seja minimizada [11].

O algoritmo A* foi desenvolvido com base no algoritmo Dijkstra por Hart et al. [21] em 1968 e utiliza nós com pesos que são constantemente atualizados. Iniciando em um nó específico, os valores ponderados dos "nós filhos" são atualizados e aquele que possuir o menor valor passa a ser o nó atual, até que todos os nós sejam atravessados. O algoritmo A* estabelece uma função de avaliação $f(n)= g(n) + h(n)$, onde $g(n)$ representa o custo real do nó inicial para o nó n, e $h(n)$ representa o custo estimado da trajetória ótima desde o nó n até o nó de destino, que geralmente é a distância euclidiana entre eles. Quando o valor de $g(n)$ é constante, o valor de $f(n)$ é afetado principalmente pelo valor de $h(n)$. Quando o nó está próximo ao nó de destino, o valor de $h(n)$ é pequeno e consequentemente, o valor de $f(n)$ também é relativamente pequeno. Dessa forma, a busca pela trajetória mais curta vai sempre em direção ao nó de objetivo. O algoritmo A* leva em consideração os pontos iniciais e objetivo do robô móvel, assim como informações colhidas ao logo da trajetória [11]. Em 2012, Petereit, Emter e Frey [76] apresentaram uma nova abordagem chamada *Hybrid A**, que foi desenvolvida para um robô não holonômico (que possui restrições de locomoção, como um automóvel) de aplicação ao ar livre, em ambientes externos cuja maior parte seja desconhecida, com o objetivo de criar trajetórias próximas a ótimas. Esta variação do A* é vista com maiores detalhes no Capítulo 3.

O algoritmo D* foi proposto por Stentz [22] em 1994 como uma alternativa ao A*. Enquanto este é mais utilizado para ambientes estáticos, o D* leva em consideração informações ambientais, sendo mais dinâmico. Este algoritmo formula o problema no espaço, como uma série de estados representado a direção da posição do robô [11]. Outros algoritmos foram gerados a partir do D* como o *field* D* [23] e Theta* [24].

Uma outra estratégia é a utilização de *Dynamic Programming* para o planejamento de trajetórias de robôs móveis, conforme foi estabelecido por Suh e Shin [25] em 1987. Nela, os vários nós de um mapa possuem determinados valores, que podem ser dados em função da distância entre eles. Partindo de um nó inicial até um objetivo, esses valores vão se somando e se atualizando até que todos dos nós tenham sido contemplados. Dessa forma, ao final, é possível encontrar o caminho mais "valioso" até o objetivo. O algoritmo de Dijkstra e a abordagem por *Dynamic Programming* são vistos com mais detalhes no Capítulo 3.

2.1.1
Planejadores Locais de Trajetórias

Os planejadores locais são todos considerados como de busca por trajetórias e se subdividem em cinco categorias que são vistas a seguir.

Khatib [26] em 1986 apresentou o método do Campo Potencial Artificial, reconhecido na literatura pela sigla APF (referente a *Artificial Potential Field*), para navegação robôs móveis. O método se baseia na ideia de campos potenciais da Física [11], sendo que o objetivo e os obstáculos agem como superfícies carregadas e o potencial total cria a força imaginária no robô. Essa força imaginária atrai o robô em direção ao objetivo e mantém longe de um obstáculo [9], conforme Figura 2.8.

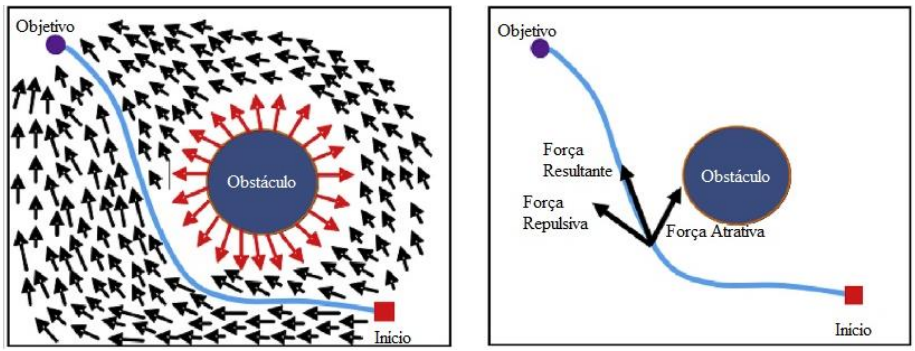


Figura 0.8 - Campo Potencial Artificial.

O método de decomposição comportamental divide o problema da navegação em problemas menores e relativamente independentes como prevenção de colisões, rastreamento, orientação de objetivos entre outros. Eles são chamados de primitivos comportamentais e são unidades completas de controle de movimento com sensores, atuadores e navegação própria, que se coordenam para executar as tarefas. Alguns trabalhos de destaque foram os de Whitbrook, Aickelin e Garibaldi [27] que utilizaram uma rede idiotípica de sistema imunológico artificial junto com *reinforcement learning* planejamento de comportamento; de Huq, Mann e Gosine [28], que propuseram uma abordagem que combina esquema de motor com modulação comportamental dependente de contexto *fuzzy*; de Fernandez-Leon et al. [29], que demonstraram que comportamentos de expansão na robótica evolutiva são eficientes; de Shi, Wang e Yang [30] que apresentaram um método local para prevenção de colisões, combinando método de curvatura de feixe com predição de

colisão e de Toibero et al. [31], que apresentaram uma abordagem de controle por comutação para o estacionamento de robô móvel não-holonômico [11].

No método de aprendizado baseado em casos, o robô necessita de um banco de dados de casos preexistentes. Mediante um novo problema, o robô busca informações relevantes e compara os resultados da pesquisa a fim de encontrar uma solução semelhante ao problema corrente. Marefat e Britanik. [32] desenvolveram um sistema de planejamento de processos que utiliza treinamento para acumular experiência. Weng et al. [33] apresentaram um método inteligente de planejamento de trajetórias, no qual casos típicos foram combinados com o conhecimento prévio da rede viária, reduzindo o espaço de busca, acelerando o processo de pesquisa e satisfazendo preferências por determinadas rotas.

A última abordagem do planejamento local é o método das janelas rolantes, no qual o robô móvel obtém informações do ambiente e as utiliza para estabelecer uma "janela", através da qual são realizados cálculos recursivos para o planejamento da trajetória. Este cálculo é contínuo de forma que a cada nova etapa, objetivos intermediários são estabelecidos por método heurístico e implementados em tempo real na janela corrente. Com o movimento da janela rolante, os objetivos intermediários são atualizados pelas informações obtidas, até que o planejamento da trajetória esteja concluído [11]. Fox, Burgard e Thrum [34] apresentaram em 1997 a abordagem por janelas dinâmicas ou DWA (*Dynamic Window Approach*) que leva em consideração a cinemática do robô, selecionando um "espaço" de todas as velocidades lineares e angulares possíveis, conforme a Figura 2.9. Baseado no DWA, Chou, Lian e Wang [35] propuseram a DWA* que aplica uma técnica de análise de região para filtrar comandos inadequados e usa o algoritmo A* para determinar o comando ótimo que pode levar o robô ao melhor resultado.

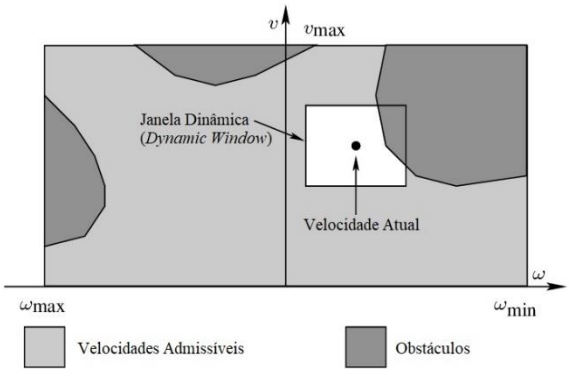


Figura 0.9 - Abordagem por Janelas Dinâmicas ou DWA.

2.1.2
Algoritmos Probabilísticos e de Inteligência Artificial

Alguns algoritmos probabilísticos e também de inteligência artificial são utilizados para planejamento de trajetórias de robôs móveis. Alguns exemplos proeminentes são vistos a seguir.

As Redes Neurais Artificiais (RNA) descrevem as restrições entre o robô e o ambiente, definindo sua localização em termos de energia. Divide-se a trajetória em vários pontos aos quais correspondem diferentes níveis de energia, que é função de sua posição. Assim como uma corrente elétrica, o robô se move na direção da diminuição da energia, de forma que o trajeto final será aquele que possuir a menor energia total. Embora a trajetória obtida seja livre de obstáculos, geralmente não é a mais curta ou ótima. Um desafio para a abordagem por RNA, sobretudo para robôs móveis, é um ambiente de trabalho não estruturado cuja descrição matemática seja difícil. Além disso, uma estrutura complexa e grande também dificulta a configuração dos pesos da rede neural [11]. Martin e Del Pobil [36] aplicaram a RNA para planejamento de trajetórias de robôs em 1994 e discutiram como as redes podem contribuir para aumentar o desempenho dos planejadores. Outros trabalhos relevantes incluem os de Mulder e Mastebroek [37] de1998, Li, Zhang e Li [38] de 2006, que propuseram um método híbrido de RNA com *Q-learning* para planejamento de trajetórias e Gonzalez et al. [39] de 2016. A Figura 2.10 apresenta uma estrutura de RNA para direção de robôs móveis.

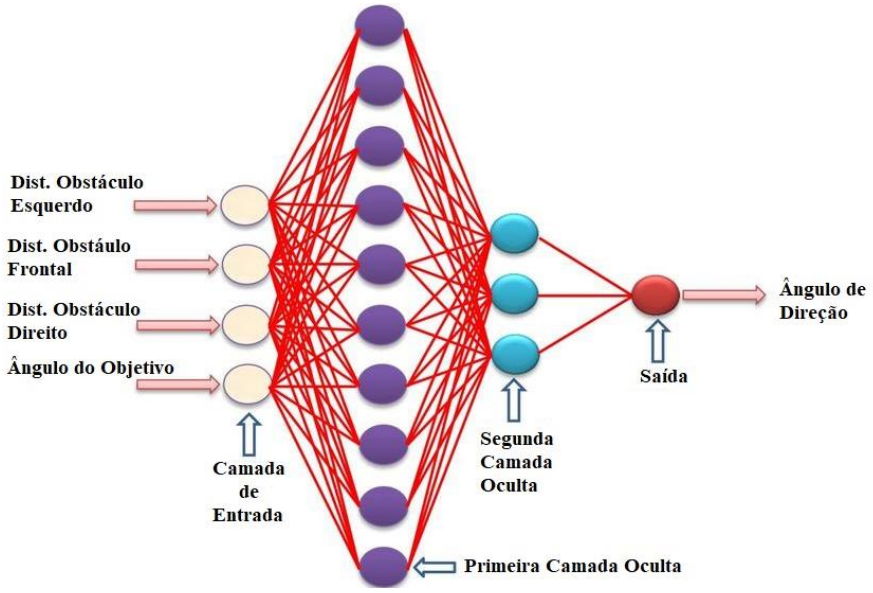


Figura 0.10 - Estrutura de RNA para direção de robôs móveis.

Algoritmos genéticos (AG) foram propostos por Holland [40] em 1975 em sua tese *Adaptation in natural and artificial systems*. O Algoritmo funciona como uma população, formada por cromossomos nos quais todas as soluções possíveis do problema são codificadas e com os quais são realizadas operações de cruzamento, mutação e seleção. Uma vez gerada uma população inicial, um valor de adequação é atribuído a cada indivíduo, determinando quais deles serão selecionados para as próximas operações. O AG apresenta como vantagem a simplicidade, a robustez, a capacidade e eficiência de pesquisa. No entanto, pode ocorrer convergência prematura pois, conforme a solução se aproxima da ótima, a taxa de convergência diminui. Uma forma de se aplicar AG no planejamento de trajetórias é utilizar um grafo cujos nós sofram mutações. Técnica de mutação avançada pode ser utilizada para identificar os nós livres próximos a um nó mutante ao invés de seleciona-los um por um e ainda identificar um nó de acordo à sua adequação à trajetória [87]. Zhao, Ansari e Hou [41], em 1992, aplicaram AG ao planejamento de trajetórias de um manipulador móvel; Pehlivanoglu, Baysal, Hacioglu [42], propuseram em 2007, um AG vibracional para o planejamento de trajetórias para VANTS; Tsai, Huang e Chan [43], em 2007, desenvolveram o PEGA (*Parallel Elite Genetic Algorithm*) que é aplicado à navegação de robôs autônomos; Tuncer e Yildirim [44] propuseram, em 2012, um AG melhorado para planejamento dinâmico de trajetórias; Qu, Xing e Alexander [45], em 2013, propuseram um AG, também melhorado, com uma estratégia co-evolucionária como solução para o planejamento da trajetória global para vários robôs móveis; Liu, Liang e Xian [46] desenvolveram, em 2014, um AG personalizado para o planejamento ótimo de trajetórias; e Shorakaei, Vahdani, Imani et al. [47], em 2016, aplicaram AG paralelo para planejamento de trajetórias ótimas cooperativas de VANTs.

O algoritmo de otimização por colônias de formigas ou ACO foi proposto por Marco Dorigo [48] em 1992, em sua tese de Doutorado. O algoritmo funciona de forma análoga à sua contraparte na natureza, na qual cada formiga de uma colônia, quando sai em busca por alimento, libera uma secreção chamada feromônio, que, entre outras, tem a função de demarcar trilhas e promover comunicação entre indivíduos. Da mesma forma que libera feromônios ao longo do caminho, uma formiga também percebe os que são deixados pelas outras. Quando a presença de feromônio em um determinado caminho é maior do que em outro, a

colônia de formigas se move espontaneamente para este, tornando-o o caminho preferencial, fazendo com que os demais deixem de existir [11]. Na robótica móvel o ACO é utilizado para otimização na identificação de trajetórias mais curtas. O robô libera "feromônios" que acabam por se concentrar nos locais que pertencem ao menor caminho. Wen e Cai [49], em 2006, modificaram o algoritmo ACO para otimizar trajetórias de robôs móveis. Liu, Mao e Yu [50] apresentaram, também em 2006, a navegação de múltiplos robôs móveis através da ACO. Tan, Huan e Sloman [51] apresentaram, em 2007, uma aplicação de ACO para planejamento de trajetórias de robôs móveis em tempo real. Wang e Xiong [52], em 2009, utilizaram ACO em pesquisas de planejamento global de trajetórias de AUVs. Zhao e Fu [53], em 2012, utilizaram uma ACO aprimorada no planejamento de trajetórias, demonstrando que esta converge rapidamente, mesmo em ambientes complexos. You, Liu e Liu [54], em 2016, propuseram um sistema caótico de colônias de formigas aplicado ao planejamento de trajetórias de robôs móveis. A Figura 2.11 representa o funcionamento de uma ACO.

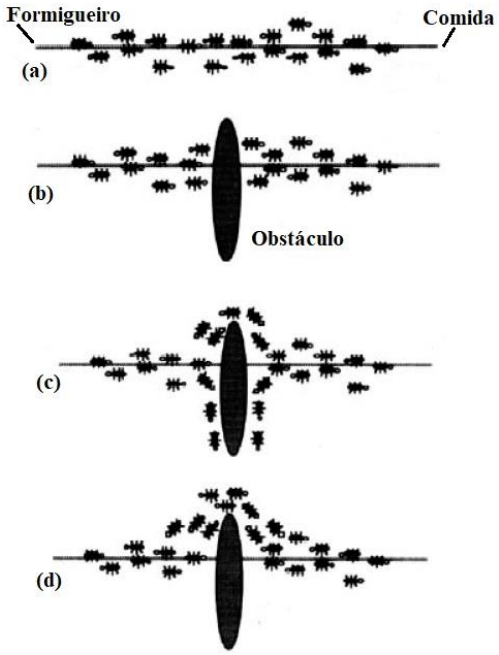


Figura 0.11 - Representação de uma colônia de formigas.

Outra técnica inspirada na natureza é a otimização por enxame de partículas ou PSO (*Particle Swarm Optimization*), proposto por Eberhart e Kennedy [55] em 1995, que emula o comportamento de grupos de aves. Um bando de pássaros, quando está buscando alimento, não exige um líder pois os indivíduos seguem

aquele que se encontra mais próximo à comida [9]. O algoritmo PSO consiste em um grupo de partículas na qual cada uma representa uma solução aleatória potencial e busca o melhor resultado através de iterações. Há uma avaliação qualitativa da solução através de um "valor de aptidão", o qual é comparado com um "valor ótimo atual" [11]. O PSO pode ser aplicado à robótica móvel para otimização de técnicas de planejamento de trajetórias ou para controle de múltiplos robôs (*robot swarm*), por exemplo. Atyabi, Phon-Amnuaisuk e Ho [56] desenvolveram em 2010, o PSO de área estendida chamado de AEPSO (*Area Extension PSO*) para um "enxame" de robôs móveis, que foi utilizado para busca e resgate de sobreviventes e desarme de bombas. Gong, Zhang e Zhang [57], em 2011, propuseram um outro método global de planejamento de trajetórias baseado em PSO com múltiplos objetivos, para aplicações com robôs móveis navegando em ambientes com fontes de perigo. Em 2013, Zhang, Wu e Wang [58] apresentaram uma abordagem de PSO Caótico adaptativo, para planejamento de trajetórias de UCAVs (*Unmanned Combat Aerial Vehicle*). Tang, Li e Jiang [59] apresentaram, em 2014, um método de SLAM de usando um filtro de partículas e utilizando PSO para minimizar cálculos e manter características de convergência mais estáveis. Liu, Li, Xie et al. [60], em 2014, desenvolveram tecnologias de planejamento de trajetórias para ambientes radioativos, baseados em PSO.

O último método mencionado é o *Simulated Annealing* (SA), que foi proposto por Kirkpatrick, Gelatt e Vecchi [61] em 1983, baseado no algoritmo *Metropolis Monte Carlo Simulation*, para encontrar o nível de energia mínimo ou máximo para um determinado sistema. O termo *annealing* refere-se ao método metalúrgico de recozimento, no qual o metal é aquecido a uma alta temperatura e resfriado lentamente a fim de remover tensões internas [62]. O algoritmo SA, por sua vez, faz analogia da temperatura com estados de energia. Iniciando a simulação com uma "temperatura alta", gera-se uma quantidade elevada de estados. Por meio de interações entre estes estados, pode-se determinar se a mudança de um estado para outro faz o "nível de energia" aumentar ou diminuir e dessa forma, verificar se a mudança de estado foi positiva ou negativa. Este processo se repete até que se atinja a "temperatura mínima" determinada, buscando sempre a solução ótima. Este processo pode ser aplicado ao "problema do caixeiro viajante" por exemplo, no qual deseja-se encontrar a menor rota em um mapa e que é base para algumas soluções no planejamento de trajetórias. Martinez-Alfaro e Flugrad [63], em 1994, aplicaram

o SA à navegação de robôs móveis a fim de encontrar uma trajetória ótima sem colisões em ambientes bi e tridimensionais. Vougioukas, Ippolito e Cugini [64] propuseram, em 1996, um algoritmo SA acelerado aplicado ao planejamento de trajetórias. Miao e Tian [65], em 2008, propuseram uma abordagem da SA para encontrar a trajetória ótima, ou quase ótima, de robôs móveis em ambientes dinâmicos. Chou [66] em 2011, também utilizou o SA para fins de planejamento de trajetórias de robôs autônomos de maneira eficaz. Miau e Tian [67] em 2013, desenvolveram uma abordagem aprimorada de SA, como solução para o planejamento dinâmico de trajetórias de robôs. Behnck et al. [68] em 2015, desenvolveram um algoritmo SA modificado, voltado para o planejamento de trajetórias de SUAVs (*Small Unmanned Aerial Vehicles*).

Assim, dada a grande variedade de planejadores de trajetórias é natural que haja também um certo número de trabalhos visando tecer comparações entre eles, sendo que alguns desses são vistos no tópico seguinte.

2.2
Estudos Comparativos entre Planejadores de Trajetórias

O presente tópico visa expor alguns outros estudos comparativos planejadores de trajetórias, que comtemplam ao menos um dos algoritmos que são estudados neste trabalho.

Começando em 2008, Sathyaraj, Jain, Finn et al. [69] realizaram um estudo comparativo de algoritmos de planejamento de trajetórias de múltiplos VANTs, através de simulações no programa de computação matemática MATLAB. A motivação do trabalho foi o uso de VANTs em conjunto para detectar alvos e mantê-los ao alcance de seus sensores, formando uma rede de comunicação entre eles. Dentre os vários algoritmos capazes de realizar pesquisa e busca de objetivos, foram estudados os de Dijkstra, de Bellman Ford (que é base para *Dynamic Programming*), de Floyd-Warshall e A*. Tais algoritmos foram aplicados a vários ambientes e seus desempenhos comparados através de grafos de nós, representando uma rede de comunicação. As simulações se deram de forma que um caminho de comunicação pudesse ser estabelecido e monitorado. A quantidade de pesquisa de cada algoritmo e o tempo de computação foram incrementados conforme a

quantidade de nós aumentava. Os resultados demostraram que o algoritmo A* teve um desempenho superior aos demais, encontrando o menor caminho.

Glabowski, Musznicki, Nowak et al. [70], em 2013, realizaram uma avaliação da eficiência de algoritmos capazes de calcular a menor trajetória entre nós. Foram avaliados 12 algoritmos, tendo como critério o tempo de execução em três tipos de grafos: customizado; multiestágio, dividido em 5 estágios de dez vértices cujas distâncias eram geradas aleatoriamente dentro de um intervalo definido; e aleatório, que foi gerado por meio de *loops*, com 5 arestas se originando de cada vértice. Os algoritmos testados foram divididos em dois grupos: a) Problema do Menor Caminho com Fonte Única, focado em algoritmos para determinar o menor caminho entre um único vértice inicial e os demais vértices do grafo e b) Problema do Menor Caminho entre Todos os Pares, que, como o nome sugere, reuniu algoritmos para determinar qual o menor caminho entre todos os pares de vértices do grafo. O primeiro grupo reuniu os seguintes algoritmos: 1) Algoritmo Genético, 2) Algoritmo de Dijkstra, 3) Algoritmo de Dijkstra usando fila (no conceito de informática), 4) Algoritmo de Dial, 5) Algoritmo de Bellman-Ford, 6) Algoritmo D'Esopo-Pape, 7) SLF (*Small Label First*), 8) LLL (*Large Label Last*) e 9) SLF/LLL. As do segundo grupo são: 1) Algoritmo de duplicação (*doubling algorithm*), 2) Algoritmo de Floyd-Warshall e 3) Algoritmo de Johnson. Os testes foram realizados através de um ambiente desenvolvido em linguagem C#, no qual cada um dos algoritmos foi executado 100 vezes em cada um dos grafos. Do primeiro grupo, o SFL foi o que teve o melhor resultado, seguido de perto do LLL e os algoritmos de Bellman-Ford e D'Esposo-Pape tiveram o pior resultado. Dentre os do segundo grupo, o algoritmo de Johnson apresentou o melhor resultado em um mapa e o de Floyd-Warshall apresentou o melhor resultado em dois mapas.

Em 2015 Zaheer, Jayaraju e Gulrez [71], realizaram um estudo comparativo envolvendo cinco algoritmos de planejamento de trajetórias para robôs autônomos em ambientes desorganizado, com o objetivo de comparar o desempenho de quatro planejadores clássicos – RRT, PRM, APF e A* - com um novo algoritmo proposto no trabalho chamado FCE (*Free Configuration Eigen-Spaces*). Os parâmetros avaliados foram: tempo de computação para se encontrar uma solução, o comprimento de trajetória e curvaturas realizadas pelo robô móvel. Os planejadores foram implementados no programa de simulação de robôs de fonte aberta *Player/Stage*, que permite a simulação de vários modelos de robôs em um ambiente

2D, inclusive em tempo real. Foi simulado um robô diferencial em dois cenários e análises adicionais foram realizadas no software de computação matemática MATLAB. Os resultados mostraram que o planejador PRM teve melhor desempenho nos critérios de distância percorrida e curvaturas, mas o RRT foi mais rápido que o PRM. O A* gerou um caminho ótimo, porém a um custo computacional elevado e muito próximo aos obstáculos. O algoritmo APF exigiu menos custos computacionais, porém o comprimento da trajetória gerada depende dos parâmetros de ajuste. E o algoritmo proposto, o FCE, apresentou valores de comprimento de trajetória e curvaturas maiores que os dos demais planejadores, mas tendo como ponto favorável o fato de gerar trajetórias suaves e manter uma boa distância dos obstáculos.

Khanmirza, Haghbeigi, Nazarahari et al. [72] realizaram um estudo comparativo em 2017, entre sete planejadores de trajetórias de robôs móveis determinísticos e probabilísticos, simulando 19 ambientes de diferentes complexidades, no MATLAB. Foram estudados: a) Algoritmos Determinísticos: A*, Algoritmo de Dijkstra e Grafo de Visibilidade; b) Algoritmos baseados em amostragem (probabilísticos): PRM, L-PRM (*Lazy* PRM), RRT e B-RRT (RRT bidirecional). Os parâmetros avaliados foram: i) comprimento da trajetória, ii) suavidade da trajetória, iii) tempo de execução, iv) taxa de sucesso em cada ambiente. Os parâmetros dos planejadores probabilísticos ainda foram avaliados acerca de sua eficiência sob diferentes condições de trabalho. Esses algoritmos forma submetidos a 50 testes separadamente e os valores médios e os desvios padrões foram utilizados nas comparações. Foi também utilizado um operador de melhoria que identificou e apagou nós intermediários em alguns casos, conectando outros diretamente. Os resultados obtidos foram que o Grafo de Visibilidade teve desempenho melhor que o Algoritmo de Dijkstra e A*em encontrar o caminho mais curto, mas tempo de execução bastante longo, sobretudo nos ambientes com obstáculos com formato irregular, pois o algoritmo utiliza os vértices dos obstáculos para construir o grafo. Ainda em relação ao Algoritmo de Dijkstra e A*, o tempo de execução depende do número de nós utilizados, que neste estudo, utilizou-se a divisão do espaço em grades (*gridmap*) e sua função heurística. Aumentando o número de nós, o desempenho dos planejadores em relação ao comprimento e à suavidade da trajetória melhoraram, mas o tempo de execução também aumentou. Por outro lado, o operador de melhoria conseguiu reduzir ainda mais o comprimento

da trajetória e aumentar a suavidade. O estudo afirma categoricamente, embasado nos resultados apresentados que, utilizar A* e algoritmo de Dijkstra juntamente com um operador de melhoria é a melhor escolha para planejamento de trajetórias em ambientes 2D. Do lado dos algoritmos probabilísticos, o que obteve a maior taxa de sucessos (geração de trajetórias) foi o B-RRT, mas o que apresentou os caminhos mais curtos e suaves foi o PRM. O B-RRT, aliás, foi o planejador que apresentou o menor tempo de execução entre todos. O estudo chama a atenção para o fato de que estes dois algoritmos são fortemente afetados pela quantidade de nós aleatórios gerados no ambiente. Os planejadores L-PRM e RRT foram os que obtiveram os piores resultados.

O próximo capítulo aborda os fundamentos teóricos acerca do planejamento de trajetórias e dos algoritmos de interesse deste estudo.

3
Fundamentos Teóricos

Este capítulo detalha as cinco técnicas de planejamento de trajetória que são objeto de estudo deste trabalho: Algoritmo de Dijkstra, *Dynamic Programming*, *Probabilistic Roadmaps* – PRM, *Rapidly-exploring Random Trees* – RRT e *Hybrid A**. São mostradas a estrutura dos algoritmos, seus princípios de funcionamento e aplicações. Mas antes são apresentados alguns conceitos importantes.

3.1
Conceitos Básicos Comuns ao Planejamento de Trajetórias

LaValle [7] em 2006, elenca alguns componentes básicos comuns a todos planejadores.

O primeiro deles é o conceito de *Estado* que, no contexto de planejamento de trajetórias de robôs móveis, significa posição e orientação do robô. A posição pode ser na forma de coordenadas cartesianas e orientação, na forma de um ângulo em relação a uma referência. Este conceito envolve também o de *Espaço de Estados*, que é o espaço que engloba todos os estados possíveis, que pode ser discreto ou contínuo. Um espaço de estados pode ter configurações diferentes acerca de dimensões e movimentos. Logo, o mesmo deve ser definido para a formulação do problema de movimento.

O *Tempo* também um uma variável importante pois os problemas de planejamento de trajetórias envolvem uma sequência de ações que são executadas através do tempo, que pode ser explícito, como no caso de um robô percorrendo uma rota durante um determinado período, ou implícito, no caso de ações que precisam ser executadas em uma determinada sequência que deve ser mantida independente do tempo de execução.

As *Ações* que são geradas pelo plano para promover mudança de estado, também faz parte deste grupo. Em alguns casos, elas podem ser determinadas por equações discretas ou contínuas no tempo, mas em outros, podem ser naturalmente determinadas.

Os problemas de planejamento de trajetórias se iniciam em um determinado local que pode ser definido em termos de um *Estado Inicial* e envolve chegar a um

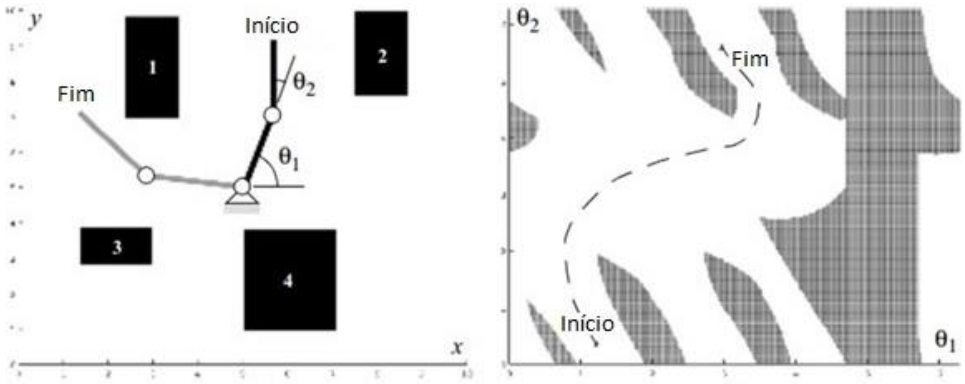
Estado Objetivo. Quando se trata de um problema de planejamento puro, alguns planejadores não levam em consideração a orientação e trabalham então somente com as coordenadas inicial e objetivo.

A trajetória a ser gerada pelo planejador pode obedecer a dois critérios: o primeiro é o da *viabilidade*, no qual o algoritmo planeja uma trajetória viável até o objetivo, independentemente de ser a mais eficiente; e o segundo é o da *otimalidade*, no qual o planejador traça uma trajetória que seja não apenas viável, mas que também tenha sua performance otimizada.

3.1.1 Espaço Configurado

O planejamento de trajetórias, tanto de manipuladores quanto de robôs móveis, em sua maioria, ocorre no que é convencionalmente chamado de *espaço configurado*. Siegwart e Nourbakhsh [8], em 2004, ofereceram uma descrição simples e didática do espaço configurado. Supondo um robô com k graus de liberdade (com $k \in R$), que pode ser um manipulador, um robô móvel, um VANT etc, qualquer estado ou configuração do robô pode ser expresso em termos de k da seguinte forma: $q_1,..., q_k$, onde q é uma configuração do robô. Por exemplo, seja um manipulador plano, como o da Figura 3.1 (a), com dois elos e dois graus de liberdade, cuja posição do atuador final, na ponta do robô, seja dada pelos ângulos θ_1 do primeiro elo e θ_2 do segundo. Então no exemplo dado, tem-se: $k = 2$ e $q_1 = \theta_1$ e $q_2 = \theta_2$. De forma mais ampla os valores de k podem representar um ponto p em um espaço de dimensões k. A este espaço é dado o nome de *Espaço Configurado C*, do robô.

A Figura 3.1 apresenta um manipulador com dois graus de liberdade, como o do exemplo dado, trabalhando em um espaço com 4 obstáculos e que necessita executar um movimento levando seu atuador de um ponto inicial a um ponto final, evitando os obstáculos. Sendo o espaço configurado C é dado pelos ângulos θ_1 e θ_2 dos elos do manipulador, pode-se definir dois subespaços de C: o *espaço configurado dos obstáculos* C_{obs}, definido como os locais em que os elos podem colidir com os obstáculos e o *espaço configurado livre* C_{free} no qual o robô pode se movimentar livremente sem riscos, dado por $C_{free} = C - C_{obs}$ [8].



(a) (b)

Figura 0.1 - (a) Manipulador com dois graus de liberdade em um espaço de trabalho com obstáculos. (b) Espaço configurado livre, das coordenadas das juntas θ_1 e θ_2, mostrando a trajetória entre os pontos inicial e final.

No caso dos robôs móveis terrestres, para fins de planejamento de trajetórias, é usual assumir que os mesmos sejam holonômicos (cujos graus de liberdade sejam iguais ao do espaço), especialmente os diferenciais, por sua capacidade de rotacionar em torno de seu eixo. Neste caso, seu estado é representado por três dimensões: suas as coordenadas no plano cartesiano *x* e y, tomadas em um ponto de referência (que pode ser o centro geométrico, por exemplo) e um ângulo θ em relação a um referencial. Entretanto, a fim de simplificar ainda mais, assume-se que o robô é um ponto no espaço, de forma que o espaço configurado possa ser reduzido a um plano cartesiano 2D. Este plano, por sua vez, deve representar o ambiente físico real e, uma vez que as dimensões do robô foram reduzidas a um ponto, todos os demais componentes no mapa que constituem os obstáculos e limites, devem ser "inflados" na mesma medida do raio de um círculo que circunscreve o robô [8]. Esta simplificação é importante para a aplicação das técnicas de planejamento de trajetórias que são vistas a seguir, pois em todas o robô será tratado como um ponto.

3.1.2 Grafos e Árvores

Grafos e árvores são conceitos muito utilizados e fundamentais para compreender os planejadores de trajetórias. Lynch e Park [74] (2017) apresentam estes conceitos de maneira eficiente.

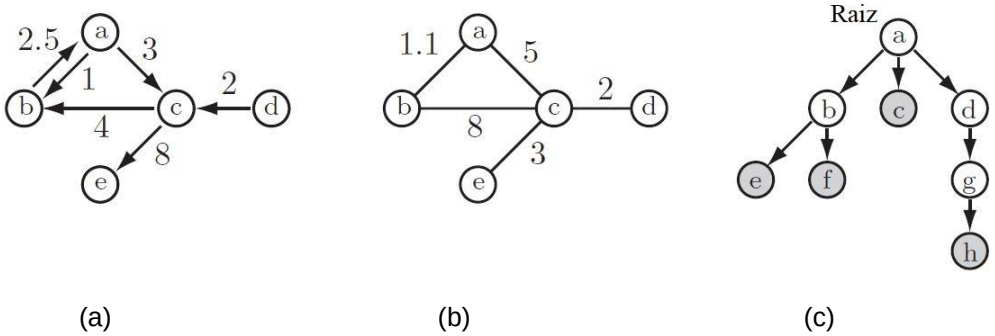
Grafos são conjuntos de *N* nós e *E* arestas, nos quais, cada aresta "*e*" (do inglês *edge*) conecta um par de nós. O espaço configurado C e o espaço de estados podem ser, por vezes, representados na forma de um grafo. Neste caso, um nó "*n*" representa um estado e a aresta que liga este nó a um outro, representa a capacidade de se mudar do primeiro estado para o segundo, sem que haja colisões ou violação das restrições. Os grafos são classificados em:

- Direcionado ou dígrafo, no qual as arestas permitem movimento em apenas uma direção, partindo de um nó n_1 para um nó n_2. Este par de nós pode ter duas arestas entre eles, permitindo assim movimentos em direções opostas.
- Não direcionado, onde cada aresta é bidirecional, permitindo que o robô trafegue tanto do nó n_1 para o n_2, como vice-versa.
- Ponderado, no qual a cada aresta é atribuído um custo associado à sua travessia.
- Não-Ponderado, onde todas as arestas possuem o mesmo custo.

Uma *árvore*, no contexto de planejamento de trajetórias, é um grafo direcionado no qual cada nó possui apenas uma aresta conectando-o a um nó *pai*. Há ainda o nó *raiz*, que é o único a não ter um *pai* e também há os nós *folhas*, que não possuem *filhos*. Outra característica das árvores é a ausência de ciclos, de forma que os nós se conectam apenas uma vez.

Uma maneira de se trabalhar com os grafos é na forma matricial. Um grafo de *N* nós pode ser representado como uma matriz $A \in R^{N \times N}$ cujos elementos a_{ij} são os custos das arestas que conectam o nó *i* ao nó *j*. Valores nulos ou negativos indicam a ausência de aresta. Os grafos e as árvores também podem ser representados na forma de uma lista de nós, cada qual com suas devidas conexões.

A Figura 3.2 apresenta três exemplos: o primeiro, no item (a), é um grafo direcionado ponderado, no qual os nós são representados pelas letras circunscritas, as arestas pelas setas e os pesos pelos números próximos às setas; o segundo, no item (b), é um grafo não direcionado ponderado, no qual os nós são representados da mesma forma e as arestas na forma de linhas com seus respectivos pesos; e por último, no item (c) tem-se a representação de uma árvore, onde o nó "a" é a raiz e os nós sombreados são as folhas.



(a) (b) (c)

Figura 0.2 - (a) Grafo direcionado ponderado. (b) Grafo não direcionado ponderado. (c) Árvore com raiz no nó “a” e nós folhas sombreadas.

3.1.3
Algumas Propriedades dos Planejadores

Uma vez apresentado o espaço configurado C e os subespaços C_{obs} e C_{free}, bem como os grafos, alguns outros conceitos podem ser melhor compreendidos. Segundo Lynch e Park [74], os planejadores podem ainda apresentar algumas outras distinções, conforme é visto a seguir.

Planejadores de múltiplas consultas, utilizam a representação do C_{free} repetidas vezes na busca de uma solução para o planejamento de trajetórias. Por este motivo, para ambientes estáticos, é interessante para eficiência do processo, investir um certo tempo construindo uma estrutura de dados que represente o C_{free} com precisão. Por outro lado, os *planejadores de consulta única*, executam cada planejamento de trajetória a partir do zero.

Um planejador do tipo “*anytime*” possui a característica de seguir buscando a melhor trajetória, mesmo depois que uma já se tenha solução. Seu nome provém do fato de que ele pode ser parado a qualquer momento (*anytime* em inglês), mediante uma condição, que pode ser, por exemplo, um intervalo de tempo ou a satisfação de algum critério.

Um planejador de trajetórias é tido como *completo* se, desde que exista uma solução, ele for seguramente capaz de encontrá-la em um determinado intervalo de tempo. Um planejador que possui *resolução completa* é aquele capaz de encontrar uma solução, caso exista, na resolução de uma representação discreta do problema. Por exemplo, a resolução de uma representação em grade do C_{free}. Por outro lado, o planejador é *probabilisticamente completo* se a probabilidade de ele encontrar uma solução existente tende a 1, conforme o tempo tende ao infinito.

Complexidade computacional trata do tempo requerido pelo planejador para executar uma trajetória ou da quantidade de memória necessária. Ambos parâmetros são medidos com base na descrição do problema, como a dimensão do espaço *C* ou o número de vértices na representação desse espaço, por exemplo. A complexidade computacional também pode ser expressa em termos de caso médio ou o pior caso.

3.2
Dynamic Programming

Dynamic Programming além de poder ser utilizada como ferramenta para encontrar o menor caminho, é também base para a criação do Algoritmo de Dijkstra. Por este motivo, a presente seção explica seu funcionamento detalhado.

Os elementos que compõem o *Dynamic Programming* são baseados em sistemas de controle, discretos, conforme a Figura 3.3.

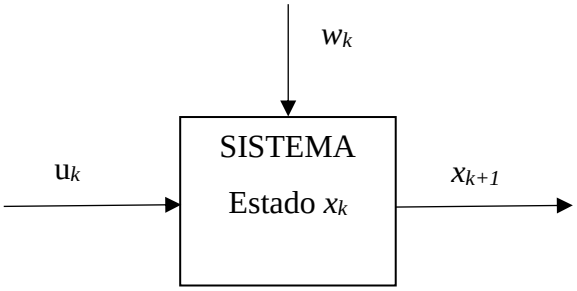


Figura 0.3 - Sistema de controle discreto.

Define-se então a dinâmica do sistema:

$$x_{k+1} = f_k(x_k, u_k, w_k), x_k \in S_k, k = 0, \dots, N, u_k \in \mathcal{U}_k(x_k), k = 0, \dots, N-1$$

Eq. 3.1

onde:

- k = índice de tempo discreto, ou estágio;
- N = horizonte de tempo dado;
- $x_k \in S_k$ = vetor de estado do sistema no tempo k. S_k é o conjunto de estados admissíveis, que podem ser função tanto do estado quanto do tempo.

- $u_k \in \mathcal{U}_k(x_k)$ = vetor entrada de controle (*input*) no tempo *k*. $\mathcal{U}_k(x_k)$ é o conjunto admissível de entradas de controle, que pode ser função tanto do tempo quanto do estado.
- f_k (,) = função de captura da evolução do sistema no tempo *k*.
- w_k = vetor de distúrbio no tempo *k*, sendo este uma variável aleatória e condicionalmente independente.

  E também se define a função de custo aditivo com valor escalar:

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k\,(x_k, u_k, w_k) \qquad \text{Eq. 3.2}$$

onde:

- $g_N(x_N)$ = custo terminal;
- $g_k(x_k, u_k, w_k)$ = custo do estágio;
- $\sum_{k=0}^{N-1} g_k\,(x_k, u_k, w_k)$ = custo acumulado.

Segundo [75] para implementar *Dynamic Programming* na solução do problema do caminho mais curto considera-se um grafo definido por um espaço de vértices $\mathcal{V}$ e um espaço de arestas ponderadas $\mathcal{C}$, tal que:

$$\mathcal{C} := \{(i, j, c_{i,j}) \in \mathcal{V} \times \mathcal{V} \times \mathbb{R} \cup \{\infty\} \,|\, i, j \in \mathcal{V}\} \qquad \text{Eq. 3.3}$$

onde:

- $c_{i,j}$ é a distância ou custo do vértice, ou nó, *i* para o vértice *j*.

A Figura 3.4 oferece uma boa visualização deste espaço. Trata-se de um grafo direcionado com arestas ponderadas.

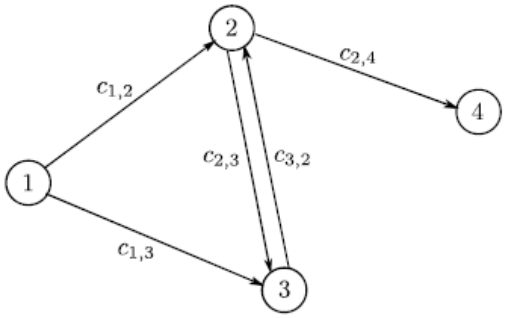


Figura 0.4 - Grafo com nós e arestas ponderadas representando distâncias.

Uma trajetória Q é então definida como uma lista de q nós ordenados, de forma que cada elemento de Q está em $\mathcal{V}$:

$$Q := (i_1, \dots, i_q) \qquad \text{Eq. 3.4}$$

Sendo S $\in \mathcal{V}$ um nó no qual se inicia uma trajetória e T $\in \mathcal{V}$ o nó onde essa trajetória termina, o conjunto das trajetórias iniciadas em qualquer S e terminadas em qualquer T é dada por $\mathbb{Q}_{S,T}$.

O comprimento da trajetória de Q, J_Q, é então, a soma dos comprimentos individuais dos segmentos que constituem esta trajetória:

$$J_Q = \sum_{h=1}^{q-1} c_{i_h, i_{h+1}} \qquad \text{Eq. 3.5}$$

O objetivo é encontrar uma trajetória Q^* que possua a menor trajetória entre S $\in \mathcal{V}$ e T $\in \mathcal{V}$. Assim:

$$Q^* = \underset{Q \in Q_{S,T}}{\arg min} J_Q \qquad \text{Eq. 3.6}$$

Para que o problema descrito tenha validade, é necessário satisfazer a seguinte suposição, que assegura que não há ciclos negativos no grafo:

$$Para\ todo\ i\ \in \mathcal{V}\ e\ para\ todo\ Q\ \in\ \mathbb{Q}_{i,j}, J_Q \geq 0$$

Logo, para todo $i \in \mathcal{V}$, $c_{i,j} \geq 0$.

Uma vez que se tem um determinado número de nós (vértices) e arestas o problema pode ser encarado como um sistema de estados finitos e determinísticos, DFS (*Deterministic Finite State*). Definindo como $|\mathcal{V}|$ o número de elementos de $\mathcal{V}$ e assumindo que $c_{i,i} = 0$ para todo $i \in \mathcal{V}$, ou seja, quando não há movimento, o horizonte de tempo N é o número de estágios do DFS. Ou seja: N $:= |\mathcal{V}| - 1$.

Definindo então:

O custo terminal: $J_N(T) = g_N(T) = 0$

Assim: $J_{N-1}(i) = c_{i,T} \ \forall\, i \in \mathcal{V}\backslash\{T\}$

De acordo com [75], para se aplicar *Dynamic Programming* diretamente sobre o problema do caminho mais curto, deve-se levar em conta que este é um problema simétrico: o menor caminho de S para T é igual ao menor caminho de T para S. O algoritmo deve ser aplicado então sobre este problema "reverso" de caminho mais curto.

Para isso, define-se um elemento auxiliar:

$$\tilde{c}_{j,i} := c_{i,j} \quad \forall \left(i, j, c_{i,j}\right) \in \mathcal{C} \qquad \text{Eq. 3.7}$$

O elemento $\tilde{c}_{j,i}$ representa as arestas ponderadas do problema reverso.

Sendo o peso das arestas representações das distâncias entre os nós, o objetivo agora é encontrar o caminho de T para S, de menor custo. Ou seja, o caminho mais curto.

Aplicando *Dynamic Programming* sobre este problema tem-se:

$$J_{N-1}(j) = \tilde{c}_{j,i} = c_{i,j}, \qquad \forall\, j \in \mathcal{V}\backslash\{S\} \qquad \text{Eq. 3.8}$$

$$\begin{aligned} J_k(j) &= \min_{i \in \mathcal{V}\backslash\{S\}} \left(\tilde{c}_{j,i} + J_{K+1}(i)\right), \forall\, j \in \mathcal{V}\backslash\{S\}, \qquad k = N-2, \dots, 1 \\ &= \min_{i \in \mathcal{V}\backslash\{S\}} \left(c_{i,j} + J_{K+1}(i)\right), \forall\, j \in \mathcal{V}\backslash\{S\}, \qquad k = N-2, \dots, 1 \end{aligned} \qquad \text{Eq. 3.9}$$

$$J_0(T) = \min_{i \in \mathcal{V}\backslash\{S\}} \left(c_{i,T} + J_1(i)\right) \qquad \text{Eq. 3.10}$$

Seja $l := N - K$ e $J_l^F := J_{N-l}$, as equações 3.8 a 3.10 podem ser reescritas da seguinte forma:

$$J_l^F(j) = c_{S,j}, \qquad \forall\, j \in \mathcal{V}\backslash\{S\} \qquad \text{Eq. 3.11}$$

$$J_l^F(j) = \min_{i \in \mathcal{V}\backslash\{S\}} \left(c_{i,j} + J_{l-1}^F(i)\right), \forall\, j \in \mathcal{V}\backslash\{S\}, \qquad l = 2, \dots, N-1 \qquad \text{Eq. 3.12}$$

$$J_N^F(T) = \min_{i \in \mathcal{V}\backslash\{S\}} \left(c_{i,T} + J_{N-1}^F(i)\right) \qquad \text{Eq. 3.13}$$

O termo $J_l^F(j)$ é o *custo ótimo para chegar* ao nó *j*, partindo do nó S, executando *l* movimentos.

A aplicação das equações 3.12 e 3.13 revela os caminhos mais curtos entre todos os nós e consequentemente a trajetória mais curta de desde o nó inicial ao nó de objetivo.

3.3 Algoritmo de Dijkstra

O Algoritmo de Dijkstra é uma forma especial de *Dynamic Programming*. É um algoritmo de busca sistêmica, capaz de gerar trajetórias ótimas, mais curtas, em um grafo com fonte única.

Seu princípio de funcionamento, segundo LaValle [7], é o seguinte: supondo um plano discreto representado como um grafo cujas arestas $e \in E$, possui, associado a si, um custo $l(e)$, definido como o custo para aplicar a ação. Utilizando a representação de espaço de estado, $l(e)$ pode ser escrito da forma $l(x, u)$, que é o custo da aplicação da ação u do estado x. Assim sendo, o custo total de uma trajetória é a soma dos custos de suas arestas, desde o estado inicial até o objetivo.

Uma fila de prioridade Q é classificada de acordo com a função *custo por vir*, definida como $C: X \rightarrow [0, \infty]$. A cada estado x é associado um valor $C^*(x)$, que é chamado de *custo por vir ótimo* a partir do estado inicial x_I. $C^*(x)$ é obtido somando-se todos os custos $l(e)$ das arestas, em todos os caminhos possíveis de x_I a x, e pegando o valor do caminho que produz o menor custo acumulativo. Caso não se saiba se o custo é o ótimo, ele é referenciado apenas como $C(x)$.

O custo por vir é calculado incrementalmente durante a execução do algoritmo de busca apresentado no final da seção. Iniciando o algoritmo com $C^*(x_I) = 0$, a cada vez que se gera um determinado estado x', é calculado um custo $C(x') = C^*(x) + l(e)$, onde e é a aresta de x a x'. Desta forma $C(x')$ torna-se o melhor custo por vir até então. Todavia, o mesmo não pode ainda ser classificado como ótimo (e por isso ainda não é chamado de C^*). Caso já exista um estado x' na fila Q, pode ocorrer que o novo caminho entre x e x' seja mais eficiente. Se for o caso, o valor do custo por vir $C(x')$ deverá ser reduzido e a fila Q deverá ser reordenada de forma a comtemplar esta mudança.

Quando um estado *x* atinge seu custo por vir *C*(*x*) mais baixo, ele assume o topo da fila *Q*. Então, o estado *x* é removido da fila através do comando *Q.PegarPrimeiro()*, de forma que o mesmo passa a ser um *estado morto* e não pode ser alcançado a um custo mais baixo do que aquele *C*(*x*). Assumindo que todo o estado morto tem seu *custo por vir ótimo* determinado corretamente, estes valores não podem mais sofrer alterações. Assim, para o primeiro elemento *x*, de *Q*, o valor do custo por vir *C*(*x*) deve ser ótimo, pois qualquer outro caminho que possua um menor custo total próprio, teria que trafegar por algum outro estado em *Q*. Contudo, os custos de todos os demais estados já são mais altos. Todos os caminhos que passam apenas pelos estados mortos já foram considerados na produção de *C*(*x*). Assim, *C*(*x*) passa a ser *C**(*x*).

Algoritmo 1 - BUSCA DIRETA

```
1   Q.Inserir (x_I) e marcar x_I como visitado
2   enquanto Q não estiver vazio, fazer
3       x ← Q.PegarPrimeiro()
4       se x ∈ X_G
5               retornar SUCESSO
6       paratodo u ∈ U(x)
7               x' ← f(x, u)
8               se x' não foi visitado
9                   Marcar x' como visitado
10                  Q.Inserir(x')
11              senão
12                  Resolver x' duplicado
13  retornar FRACASSO
```

3.4 Probabilistic Roadmaps – PRM

O *Probabilistic Roadmaps* (PRM), como seu próprio nome em inglês já diz, é um algoritmo probabilístico, que se utiliza de amostragem de nós para criar um grafo não direcionado representando o C_{free}, chamado mapa de rotas (*roadmaps*). O mapa de rotas permite que um robô trafegue de um nó a outro, em ambas as

direções, através das arestas do grafo. Por este motivo, o planejador é mais aplicado a problemas cinéticos, pois pode encontrar caminhos entre quaisquer pontos no espaço com precisão, evitando obstáculos [74].

A metodologia é composta por duas fases: uma de aprendizado e outra de consulta (*query*). A primeira é a de construção do mapa de rotas conforme descrito, e na segunda, dadas duas configurações quaisquer, sendo uma inicial e a outra de objetivo, são conectadas a dois nós do mapa de rotas, sobre o qual é realizada uma busca por caminhos que interliguem estes dois nós [17].

O PRM pode ser aplicado a praticamente qualquer tipo de robô holonômico, devido à sua simplicidade e facilidade de implementação. Para isso, é necessário configurar alguns parâmetros cujos valores são dependentes das próprias configurações do robô, assim como do ambiente. Um desses parâmetros pode ser, por exemplo, o tempo de duração da fase de aprendizado [17].

O mapa de rotas necessita que um planejador local realize buscas entre seus caminhos e construa uma trajetória entre os nós. Segundo [74], o algoritmo A* é comumente utilizado para este fim.

Uma característica importante do PRM é manter a resolução mesmo com a expansão do espaço configurado *C*, de forma que a área de ação do planejador local não se reduz exponencialmente com o crescimento do espaço, como ocorre com os planejadores que utilizam representações do espaço em grades, para criar seus mapas de rotas. Isto conta muito em favor da eficiência do PRM [74].

O Algoritmo 2 abaixo apresenta a construção de um mapa de rotas *R* com *N* nós, cujo nó inicial é q_i.

Algoritmo 2 - Algoritmo PRM de construção de mapas

```
1  para i = 1, ..., N fazer
2      q_i ⟵ amostra do C_free
3      adicionar q_i a R
4  fim para
5  para i = 1, ..., N fazer
6      𝒩(q_i) ⟵ k vizinhos mais próximos de q_i
7      para cada q ∈ 𝒩(q_i) fazer
8          se houver um caminho local livre de colisões de q a q_i e não houver
```

uma aresta conectando q a q_i **então**

9 adicionar uma aresta de q a q_i ao mapa de rota R

10 **fim se**

11 **fim para**

12 **fim para**

13 **retornar** R

A Figura 3.5 apresenta um mapa de rotas construído em um espaço $C = \mathbb{R}^2$ e número de nós conectados ao nó amostral q_i, $k = 3$ (linha 6 do algoritmo). O número de conexões de um nó, entretanto, pode ser superior a três, visto que o mesmo pode ser próximo a vários outros nós.

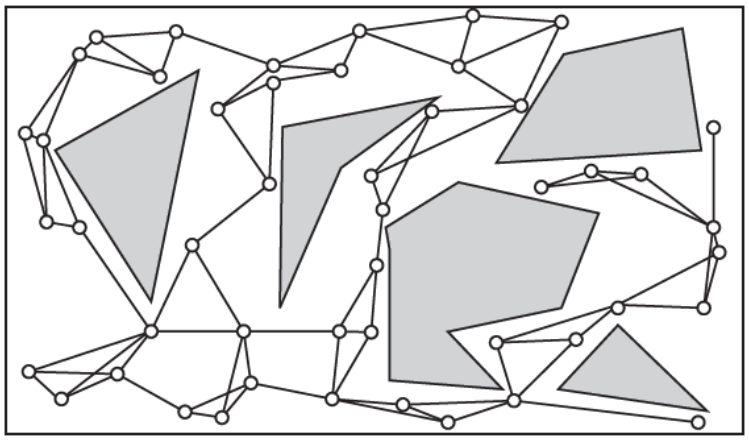


Figura 0.5 - Mapa de rotas probabilístico.

3.5 Rapidly-Exploring Random Trees – RRT

O algoritmo *Rapidly-exploring Random Trees* (RRT), diferentemente do PRM, constrói árvores, conforme descrito em 3.1.2.

O conceito da RRT é a de uma estrutura de dados aleatória capaz de atender a uma série de problemas de planejamento de trajetórias, tendo sido desenvolvida principalmente para lidar com restrições não holonômicas, incluindo dinâmicas e também com um número de graus de liberdade elevados. Uma RRT se expande iterativamente por meio de comandos que proporcionam uma conexão direcionada a nós selecionados aleatoriamente [18]. Partindo de um estado inicial x_{inicio} o planejador procura por um caminho sem colisões até uma região de objetivo $\mathcal{X}_{obj}$.

Para aplicações cinemáticas o estado x é apenas uma configuração q em C_{free} (coordenadas cartesianas e orientação) e para aplicações dinâmicas, além da configuração *q*, o estado inclui velocidades [74].

O Algoritmo 3 apresenta uma RRT básica para a criação de uma árvore T (do inglês *tree*), a partir de um espaço amostral $\mathcal{X}$, iniciando de um nó x_{inicio}.

Algoritmo 3 – Algoritmo RRT

```
1   Iniciar busca da árvore T em x_inicio
2   Enquanto T for menor que o tamanho máximo da árvore faça
3     x_amostra ◄— amostra de X
4     x_perto ◄— nó de T mais próximo de x_amostra
5     empregar planejamento local para traçar uma rota entre x_perto e x_novo em
      direção a x_amostra
6     se a rota for livre de colisões então
7       Adicionar x_novo a T com uma aresta partindo de x_perto para x_novo
8       se x_novo estiver em X_obj então
9         retornar SUCESSO e a rota para x_novo
10      fim se
11    fim para
12  fim enquanto
13  retornar FALHA
```

Algumas considerações acerca do algoritmo 3 [74]: para implementações em problemas simplesmente cinéticos, a amostragem de $x_{amostra}$ na terceira linha é aleatória, a partir de uma distribuição quase uniforme em $\mathcal{X}$, mas com uma pequena tendência a buscar estados que estejam em $\mathcal{X}_{obj}$. Na linha 4, o nó da árvore T mais próximo de $x_{amostra}$, x_{perto}, é aquele que apresenta a menor distância euclidiana a ele. Na linha 5, ao ser escolhido $x_{amostra}$ e verificado qual é o nó da árvore mais próximo x_{perto}, um planejador local, geralmente um bem simples, traça uma trajetória entre eles (que pode ser uma linha reta). Caso haja um outro nó entre esses dois, ou seja, ainda mais próximo a x_{perto} ele recebe o nome de x_{novo} e, se satisfizer a condição da linha 6, então ele é acrescentado à árvore. Mas, caso não

haja nenhum outro nó entre x_{perto} e $x_{amostra}$, este último será o mais próximo. Neste caso $x_{amostra}$ torna-se x_{novo}.

A Figura 3.6 apresenta uma sequência de criação de uma árvore T por uma RRT. Na primeira imagem, (a), tem-te o início da criação, indicando qual o x_{inicio}. Em (b) tem-se a escolha de um $x_{amostra}$ aleatório e em (c) a inclusão de um x_{novo} à árvore.

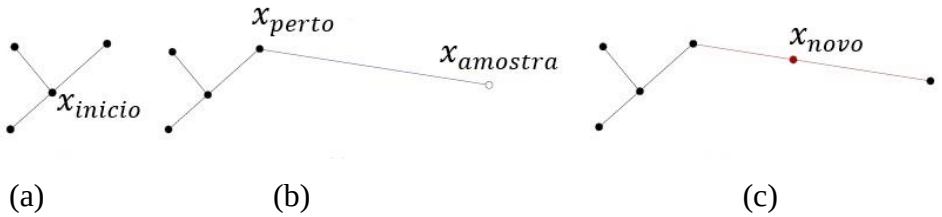


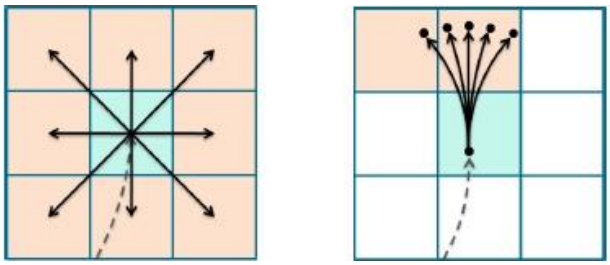
Figura 0.6 - Sequência de construção de uma árvore T por RRR.

3.6 Hybrid A*

O algoritmo *Hybrid A** foi criado como uma extensão do A* tradicional, com o fim de atender a veículos com restrições não holonômicas. O uso mais comum do A* é em espaços de estado discretos. Tomando como exemplo um espaço de nove grades conectadas como o da Figura 3.7(a), para um veículo seguir em qualquer direção que não seja avançar ou regredir em linha reta, ele necessitará executar uma rotação. Isso não é possível para veículos não holonômicos. Diferentemente do seu antecessor, o *Hybrid A** realiza uma busca contínua no espaço, através de um conjunto de pequenas curvas pré-computadas chamadas de primitivos de movimento, que são capazes de determinar quais estados são possíveis para o robô alcançar, para assim construir uma árvore. Um recurso do qual o algoritmo se utiliza é guardar as posições dos primitivos, que são contínuos, juntamente com as posições discretas das células que eles ocupam. A Figura 3.7(b) ilustra um nó do *Hybrid A** com seus primitivos de movimento [76].

Em se tratando de um movimento contínuo, o estado do veículo também deve ser representado de forma contínua por sua posição *x* (coordenadas em um plano cartesiano) e sua orientação θ. Entretanto, o *Hybrid A** necessita que o espaço esteja discretizado para poder pesquisa-lo. Uma vez que a posição *x* é dada por pares de coordenadas, o movimento de translação já é discretizado pela representação do mapa. Quanto a θ, assume-se de que ele e seu equivalente discreto,

$\tilde{\theta}$ (o acento “til” significa variável discreta) são iguais, pois como a quantidade de primitivos é limitada, os mesmos sempre terminarão com uma orientação que se encaixa no esquema de discretização. Para pesquisa do espaço discretizado, o algoritmo utiliza a posição discreta do robô, dada por $\tilde{x}$. Apesar desse parâmetro ser suficiente para fins de pesquisa ambiental, as coordenadas da posição contínua, *x*, também são armazenadas para cada célula visitada e são utilizadas como base, ou raiz, para pesquisa na vizinhança, que é chamada de expansão nodal. Esta dupla natureza do algoritmo, discreta e contínua, garante a viabilidade da trajetória gerada e dá nome ao algoritmo. Isso não seria possível se apenas os centros de cada célula fossem considerados nas pesquisas [76].



(a) (b)

Figura 0.7 - (a) Representação do comportamento do algoritmo A* tradicional. (b) Representação do comportamento do Algoritmo *Hybrid A*.*

Além desses parâmetros relacionados à posição, um nó *n* do *Hybrid A** também é definido por alguns outros parâmetros conforme a Equação 3.14.

$$n = (\tilde{x}, \tilde{\theta}, x, g, f, n_p) \qquad \text{Eq. 3.14}$$

onde:

- $\tilde{x}$ é posição discreta do robô.
- $\tilde{\theta}$ é a orientação discreta do robô.
- *x* é a posição contínua do robô.
- *g* é o custo acumulado ao longo da trajetória.
- f é uma chave de consulta prioritária que contém a soma dos custos reais *g* e dos custos estimados *h* para atingir o objetivo.
- n_p é um ponteiro que retorna ao nó gerador (pai), utilizado para construção da trajetória.

Os primitivos de movimentos são a menor entidade da trajetória e seus arcos devem satisfazer três condições: 1) l ser maior ou no mínimo, suficiente para transpassar a célula em que se encontra. Sendo ζ o tamanho da célula: $l > \sqrt{2} \, . \, \zeta$. 2) A curva é limitada pelo ângulo máximo de esterçamento das rodas do robô: α_{max}. 3) A mudança de direção δ deve ser um múltiplo do tamanho do passo de discretização da dimensão da orientação do espaço de pesquisa [76].

Uma vez satisfeitas as condições acima, o ângulo de esterçamento α pode ser calculado segundo a Equação 3.15

$$\alpha(l, \delta) = \arctan\left(\frac{2b}{l} \, . sen \frac{\delta}{2}\right) \qquad \text{Eq. 3.15}$$

onde:

- b = base da roda do robô.

Há então um conjunto de primitivos de movimento válidos $\mu(\theta, \delta)$, composto por todos os ângulos de mudança de direção que atendam à condição: $\alpha(l, \delta) \leq \alpha_{max}$.

Assim como o A* tradicional, o *Hybrid A** separa os nós em dois conjuntos: o primeiro é o conjunto aberto *O* (do inglês *open*), contendo os nós que se avizinham aos que já foram expandidos durante a pesquisa; e o conjunto fechado *C* (do inglês *closed*), contendo os nós que já tenham sido processados [76].

O algoritmo 4 apresenta o algoritmo básico do *Hybrid A**, no qual um veículo se encontra em um mapa discreto *m*, composto por três camadas: m_o, que é a camada com informações se a célula possui obstáculos ou não; m_s, com informações acerca da qualidade do terreno e m_h que contém informações da altura da célula (considerando mapas 2.5D). A posição corrente do veículo é dada por x_s e θ_s e o nó corrente é n_s. O objetivo encontra-se na região *G*, que é o conjunto das localidades *x*, dentro de um raio *r*, em torno do ponto de parada especificado ω.

Algoritmo 4 – Algoritmo *Hybrid A**

1 **proceder** PLANEJAR_TRAJETÓRIA (m, μ, x_s, θ_s, G)

2 $n_s \longleftarrow (\tilde{x}_s, \tilde{\theta}_s, x_s, 0, h(x_s, G), -)$

3 $O \longleftarrow \{n_s\}$

```
4     C ⟵ ∅
5     enquanto O ≠ ∅ faça
6       n ⟵ nó com o menor valor f em O
7       O ⟵ O\{n}
8       C ⟵ C ∪{n}
9       se n_x ∈ G então
10        retornar trajetória reconstruída começando em n.
11      senão
12        ATUALIZAR_VIZINHANÇA(m, μ, O, C, n)
13      fim se
14    fim enquanto
15    retornar nenhum caminho encontrado
16  fim proceder
17  proceder ATUALIZAR_VIZINHANÇA(m, μ, O, C, n)
18    Para todo δ fazer
19      n' ⟵ estado seguinte de n usando μ(n_θ, δ)
20      se n' ∉ C então
21        se m_o(n'_x̃) = obstáculo então
22          C ⟵ C ∪ {n'}
23        senão se ∃ n ∈ O : n_x̃ = n'_x̃ então
24          Computar novos custos g'
25          Se g' < g valor do nó existente em O então
26                Substituir o nó existente em O com n'
27            fim se
28          senão
29              O ⟵ O ∪ {n'}
30            fim se
31          fim se
32      fim enquanto
33  fim
```

No próximo capítulo, é apresentada a metodologia aplicada à realização das simulações.

4
Metodologia

Este capítulo apresenta a implementação dos cinco planejadores estudados, assim como suas estruturas no ambiente MATLAB. É mostrado a forma de se obter cada parâmetro avaliado e também os cálculos executados. O estudo é conduzido por meio de ambientes virtuais na forma de três mapas, nos quais as simulações foram realizadas.

4.1
Mapas

Foram idealizados três mapas para a realização das simulações, sendo que cada um foca em um determinado aspecto que se busca avaliar nos planejadores. Todos possuem 25 x 25 unidades de comprimento, tendo sido criados sobre um ambiente dividido em grades de tamanho 1 x 1 unidades de comprimento. A Figura 4.1 apresenta os três mapas, com as estrelas verdes indicando os pontos iniciais e as amarelas os pontos de objetivo.

(a) (b) (c)

Figura 0.1 - Mapas das simulações: (a) Labirinto, (b) Labirinto Estreito e (c) Ziguezague.

O primeiro mapa, chamado de “Labirinto” busca avaliar a capacidade dos planejadores em traçar uma trajetória em um ambiente complexo e com múltiplos caminhos. O segundo mapa, denominado “Labirinto Estreito”, repete os obstáculos do primeiro, porém, inflando os mesmos de forma a estreitar os caminhos, para se avaliar a capacidade dos planejadores em traçar trajetórias em um ambiente mais desafiador. Nesses dois mapas, os pontos inicial e de objetivo são (2,2) e (23,23),

respectivamente. O terceiro mapa, denominado “Ziguezague”, tem o objetivo de averiguar o comportamento dos planejadores em um caminho longo e no qual se necessite inverter a direção e fazer curvas fechadas. Os pontos inicial e de objetivo neste mapa são (2,2) e (3, 23).

4.2 MATLAB

O MATLAB, abreviatura de *Matrix Laboratory*, é um pacote de *software* desenvolvido pela *The Math Works Inc* para cálculos matemáticos interativos. Pode ser entendido como uma linguagem de computação dedicada ao processamento de dados na forma de matrizes numéricas. O programa conta com uma grande variedade de funções e ferramentas para geração de gráficos e visualizações [77].

Um recurso útil do MATLAB são as *Toolboxes*, definidas como um conjunto de funções dedicadas a um tema específico, que podem ser desenvolvidas tanto pela *Math Works* quanto por outras empresas ou indivíduos, comercializadas ou até distribuídas gratuitamente [78], como por exemplo a *Robotic Toolbox*, desenvolvida por Peter Corke. Dois dos planejadores estudados, o RRT e o *Hybrid A**, fazem parte da *Navigation Toolbox*, que contém algoritmos e ferramentas de análises para o desenvolvimento de planejamento de movimento e sistemas de navegação [79].

4.3 Procedimento de Simulação

Os planejadores de trajetória implementados no MATLAB possuem alguns parâmetros que podem ser editáveis. A simulações consistem em, primeiramente, executar uma série dez simulações na configuração *default* do planejador, sendo que em cada simulação individual, mede-se o tempo de processamento, anota-se a quantidade de passos e calcula-se o comprimento da trajetória. Ao final de cada uma guarda-se a imagem da trajetória gerada. Após esta primeira série, altera-se um ou mais parâmetros, dependendo do planejador e executa-se mais dez simulações, medindo os mesmos parâmetros e também guardando imagens das simulações. Repete-se este procedimento até que se obtenha uma quantidade de dados que

permita uma análise satisfatória ou até que uma determinada configuração se mostre superior às demais.

A partir dos dados coletados, constroem-se tabelas e gráficos, a fim de comparar os comportamentos dos planejadores nas diferentes configurações. Gera-se, então, gráficos para as médias de cada série de dez simulações, quando for pertinente, assim como de cada simulação individual, também quando for pertinente, dos parâmetros medidos: tempos de processamento, quantidades de passos e comprimentos das trajetórias. Caso as curvas apontem alguma correspondência entre dois parâmetros, gera-se um outro gráfico de um parâmetro versus o outro a fim de se avaliar tal correspondência. Por exemplo, caso se verifique que o tempo de processamento cresce à medida que a quantidade de passos cresce, pode-se verificar se há correspondência entre esses parâmetros através de um gráfico de tempo de processamento versus quantidade de passos.

Os planejadores baseados em Algoritmo de Dijkstra, *Dynamic Programming* e PRM sempre apresentam uma trajetória diferente a cada simulação. Logo, o tempo de processamento, o número de passos e o comprimento da trajetória serão diferentes para cada simulação individual. Ou seja, em uma série de dez simulações, com os mesmos parâmetros, serão geradas dez trajetórias diferentes. Em contrapartida, RRT e *Hybrid A** apresentam sempre a mesma trajetória em cada série, se as configurações dos planejadores não forem modificadas. Neste caso, o único parâmetro que varia entre as simulações é o tempo de processamento de cada uma delas. Por este motivo, também não se faz necessário guardar uma imagem de cada trajetória gerada para estes dois planejadores. Uma imagem que represente uma série já é suficiente, tomando o devido cuidado de se verificar se as trajetórias geradas são iguais.

Os tempos de processamento são obtidos através da função “tic toc” do MATLAB, que mede em segundos o tempo de execução de um comando. Essa função é inserida em partes estratégicas dos programas de forma a medir os tempos de execução de cada função envolvida na tarefa de geração das trajetórias.

A quantidade de passos, que é o número de nós que constituem a trajetória, são fornecidas diretamente pelos programas, através de uma função que gera uma matriz com todos os pontos, ou nós, do plano cartesiano que fazem parte do trajeto. Mais adiante é mostrado como obter este parâmetro para cada planejador.

O comprimento da trajetória é obtido por meio da distância entre os pontos que compõem as trajetórias. Uma vez que os programas fornecem tais pontos, pode-se calcular a distância entre eles de acordo com desenho da trajetória. Os comprimentos das trajetórias geradas pelos planejadores Algoritmo de Dijkstra, *Dynamic Programming*, PRM e RRT são calculadas por uma mesma equação, mas as geradas por *Hybrid A**, devido à característica contínua de tais trajetórias, são calculadas por um algoritmo, conforme é visto mais adiante.

4.4 Algoritmo de Dijkstra e Dynamic Programming

Os algoritmos de Dijkstra e *Dynamic Programming* utilizados nas simulações foram desenvolvidos por Balcitar [80] em 2020, e fazem parte de um mesmo programa que gera os mapas, cria nós aleatoriamente em suas áreas livres e traça as trajetórias. Tanto o Algoritmo de Dijkstra quanto *Dynamic Programming* são funções chamadas dentro de seu corpo principal, de forma que pode-se utilizar ambos os algoritmos ao mesmo tempo. Entretanto, a fim de se obter distribuições de nós distintas, optou-se por utilizar um algoritmo por vez nas simulações.

Apesar de ser um programa de fonte aberta, buscou-se preservar ao máximo suas características originais. Para as simulações deste trabalho, foram modificados: os mapas, pois o programa originalmente possuía um mapa apenas, completamente diferente dos três que foram desenvolvidos; os pontos inicial e de objetivo, pois os mesmos precisaram ser adaptados aos novos mapas; e, por fim, as quantidades de nós também foram modificadas no código fonte, a cada série de simulações.

Além do corpo principal, chamado "*main*", as seguintes funções compõem o programa:

- *map_definition*: função na qual os componentes dos mapas são definidos.
- *generate_node*: responsável por gerar os nós nas áreas livres dos mapas.
- *generate_undirected_graph*: interliga os nós, gerando um grafo não direcionado, que é utilizado no planejamento das trajetórias.
- *addstartendpoint2ungraph*: adiciona os pontos inicial e final ao grafo.
- *astar2*: algoritmo A* (desativado neste estudo).
- *dijkstra*: algoritmo de Dijkstra.
- *dynamicpathplanning*: algoritmo de *Dynamic Programming*.

- *sp_dp*: função recursiva do *Dynamic Programming* para resolver o problema do caminho mais curto.
- *drawRoute*: desenha o grafo de nós e a trajetória.
- *costcal:* calcula o "custo" ou distância entre dois nós.
- *pathcost*: calcula o "custo", ou comprimento da trajetória.

O mapa construído pelo programa é constituído por polígonos em um plano cartesiano cujos vértices são definidos por pontos do tipo (*x*,*y*). Tomando como exemplo a aresta inferior do mapa Labirinto, Figura 4.1(a), então definida como sendo o "objeto 1" do mapa, seus vértices se encontram nas posições (0,0), (25,0), (25,1) e (0,1). Para se desenhar este objeto no mapa define-se os vértices no código fonte da função "*map_definition*" da seguinte forma:

map.pgx{1}=[0 25 25 0 0];

map.pgy{1}=[0 0 1 1 0];

Onde *map.pgx{1}* é o conjunto das coordenadas, no eixo "*x*", dos vértices do objeto 1 e *map.pgy{1}* é o conjunto das coordenadas no eixo "*y*" dos mesmos pontos. Salienta-se de que é necessário "fechar" o polígono ao se definir os pontos das coordenadas. Assim, a primeira e a última posição de cada conjunto se repetem.

Tradicionalmente o Algoritmo de Dijkstra utiliza ambientes discretizados na forma de *gridmaps*, para planejamento de trajetórias de robôs móveis. Mas o programa deste estudo se baseia em amostragem, gerando nós aleatoriamente nas áreas livres do mapa e estabelecendo conexões entre eles para, então, aplicar o algoritmo sobre esses nós. Essa abordagem não é nova, já tendo sido apresentada em [70]. O interessante dela, é que não há necessidade de se discretizar todo o ambiente a fim de se gerar a trajetória, o que poderia acarretar em custos computacionais mais elevados. Sendo então de natureza estocástica, cada simulação gera uma trajetória diferente, com parâmetros distintos.

Uma vez que o programa permite a variação do número de nós como parâmetro ajustável, optou-se por variar apenas este parâmetro para cada série de dez simulações.

O tempo de processamento, é obtido através da medição dos tempos de execução das funções envolvidas diretamente na geração dos nós, do grafo e da trajetória: *generate_node*, *generate_undirected_graph* e *dijkstra*, para o Algoritmo de Dijkstra ou *dynamicpathplanning* para *Dynamic Programming*. Essas três

medidas são somadas para constituir o tempo de processamento de cada simulação individual. A Figura 3.2 mostra o local na tela onde as tomadas de tempo são coletadas.

A quantidade de passos é fornecida através do parâmetro "*dijkstra_route*" para o Algoritmo de Dijkstra, mostrado na Figura 4.2 e "*dpundirected_route*" para o *Dynamic Programming*, conforme Figura 4.3. Nos exemplos das figuras, a simulação com o Algoritmo de Dijkstra deu 8 passos e *Dynamic Programming*, 7.

O comprimento da trajetória é medido coletando os nós que constituem os passos diretamente dos parâmetros "*dijkstra_route*" e "*dpundirected_route*" para o Algoritmo de Dijkstra e *Dynamic Programming* respectivamente. Clicando sobre estes arquivos eles fornecem os valores dos nós que constituem os passos, conforme a Figura 4.4.

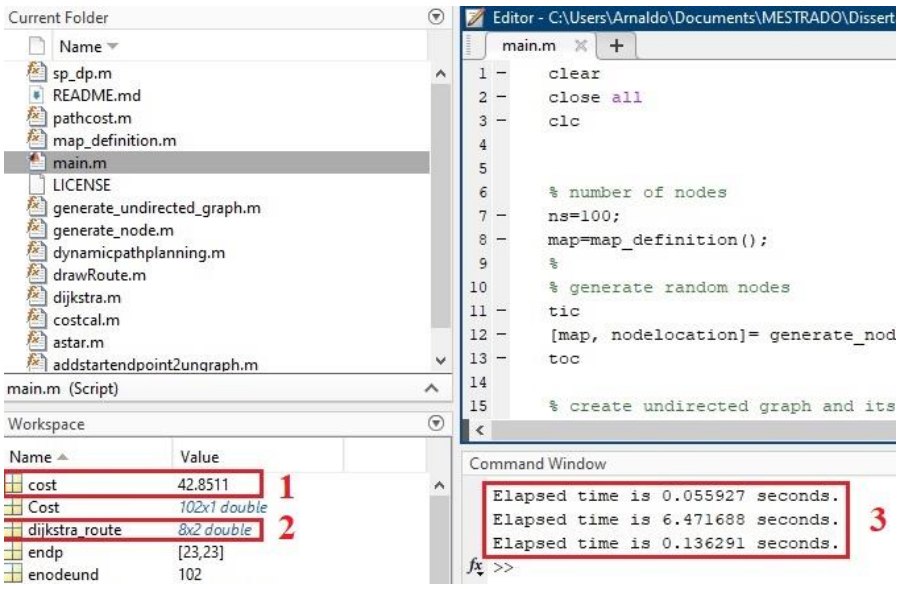


Figura 0.2 - Espaço de trabalho do MATLAB com dados acerca da simulação: 1) comprimento da trajetória, 2) matriz com os passos, 3) tempos de execução.

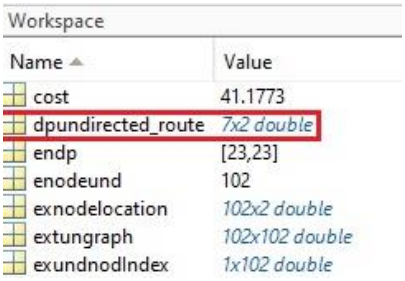


Figura 0.3 - Indicação da matriz contendo os passos da simulação com *Dynamic Programming.*

dijkstra_route

8x2 double

	1	2
1	2	2
2	17.4067	2.3455
3	20.0337	5.6961
4	17.1949	8.9807
5	17.0841	13.6648
6	21.2678	14.0140
7	23.4957	16.1388
8	23	23

Figura 0.4 - Nós que constituem os passos de uma trajetória, sendo a coluna 1, os valores do eixo x e a coluna 2, os valores do eixo y.

De posse das coordenadas dos passos, é possível calcular o comprimento da trajetória pela Equação 4.1, abaixo:

$$d = \sum_{i=2}^{p} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \qquad \text{Eq. 4.1}$$

onde:

- d = comprimento da trajetória em unidades de comprimento;
- p = quantidade de passos;
- x_i = coordenada no eixo x do nó.
- x_{i-1} = coordenada no eixo x do nó anterior.
- y_i = coordenada no eixo y do nó.
- y_{i-1} = coordenada no eixo y do nó anterior.

O somatório para o cálculo do comprimento se inicia no segundo nó da trajetória pois a primeira distância se dá entre os nós 1 e 2.

O programa também calcula o comprimento das trajetórias através das funções *costcal* e *pathcost* automaticamente, mas ainda assim o cálculo manual é executado forma de conferência.

A seguir é vista a metodologia de simulações do *Probabilistic Roadmaps*.

4.5
PRM – Probabilistic Roadmaps

O programa com planejador PRM utilizado neste estudo é um objeto do próprio MATLAB chamado "*mobileRobotPRM*", capaz de criar um mapa de rotas entre os espaços vazios de um ambiente 2D, como os mapas apresentados anteriormente, utilizando a técnica de PRM [81].

O planejador funciona definindo-se, primeiramente, um mapa que represente o ambiente do robô, na forma de um "mapa de ocupação binária". Este, por sua vez, se trata de uma matriz com valores binários no qual "1" (um), ou verdadeiro, indica a presença de obstáculos e "0" (zero), ou falso, indica espaço livre. Em seguida, o planejador gera um determinado número de nós, que é especificado pelo usuário, entre os espaços vazios do mapa. A partir de então, estabelece-se as conexões entre os nós formando o mapa de rotas. A conexão entre dois nós quaisquer deve respeitar dois critérios: que não haja obstáculos entre eles e que os mesmos se encontrarem a uma determinada distância, que é o delta, definido pelo usuário. Assim, a partir do mapa de rotas formado o planejador traça a trajetória entre os pontos inicial e final definidos.

Um programa de MATLAB foi desenvolvido para gerar os mapas de ocupação binária, configurar o número de nós e o delta – que são as variáveis de entrada ajustáveis - e ainda, chamar a função "*mobileRobotPRM*" e gerar as imagens com as trajetórias. A fim de evitar falhas na geração de trajetórias, utilizou-se o recurso "*findpath*", que verifica se a tentativa de traçar uma rota foi bem-sucedida. O recurso foi inserido em um *loop* no qual a cada falha na geração de uma trajetória, há um incremento de dez nós à simulação, até que haja sucesso. Cabe aqui uma observação de que simulações preliminares mostraram não haver formação de trajetórias com menos de 50 nós, que foi tomado como *default*.

O procedimento de simulações segue então o roteiro estabelecido iniciando o planejador com 50 nós e com delta igual a 5 unidades de comprimento. A cada série de dez simulações o delta é incrementado em cinco, até que se tenha quatro séries, com delta igual a 10, 15, 20 e 25.

A medição do tempo de processamento é executada sobre a função "*mobileRobotPRM*" que gera os nós, os mapas de rotas e ainda traça a trajetória e também dentro dos *loops* de verificação e incremento de nós.

A quantidade de passos é dada pelo parâmetro "*path*", que é a matriz dos nós gerados na trajetória e pode ser conferida diretamente no espaço de trabalho do MATLAB, conforme a Figura 4.5.

O comprimento da trajetória é obtido coletando os valores das coordenadas *x* e *y* dos passos, presentes no parâmetro *path*, mostrado nas Figuras 4.5 e 4.6 e aplicando a Equação 4.1.

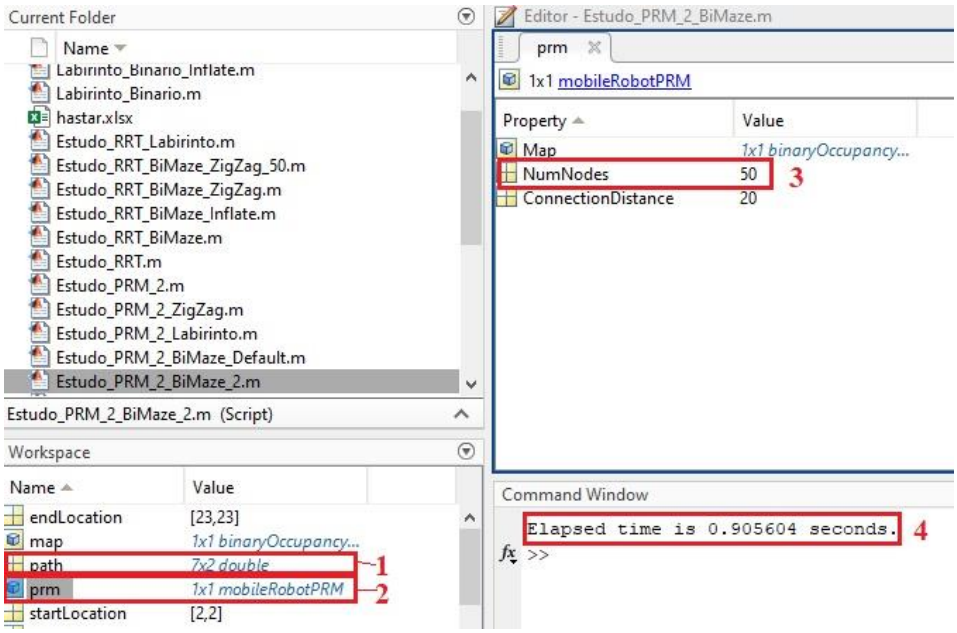
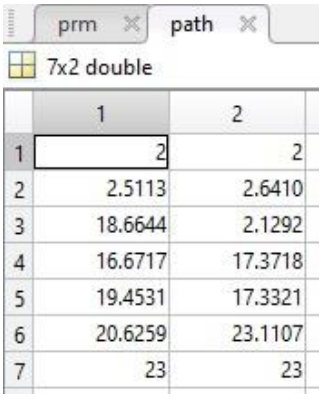


Figura 0.5 - Tela do MATLAB com dados acerca das simulações com PRM: 1) Matriz *path* que contém os passos; 2) arquivo do "mobileRobotPRM" contendo 3) número de nós e delta; 4) tempo computado.



	1	2
1	2	2
2	2.5113	2.6410
3	18.6644	2.1292
4	16.6717	17.3718
5	19.4531	17.3321
6	20.6259	23.1107
7	23	23

Figura 0.6 - Nós que constituem os passos de uma trajetória do PRM, sendo a coluna 1, os valores do eixo *x* e a coluna 2, os valores do eixo *y*.

No próximo tópico é apresentado a metodologia do RRT.

4.6
RRT – Rapidly-exploring Random Trees

O planejador RRT utilizado nas simulações é o "*plannerRRT*" um objeto que faz parte da *Navigation Toolbox* do MATLAB. Ele trabalha com espaço de estados para executar a expansão dos ramos da árvore e criar a trajetória, sendo necessário selecionar e validar este espaço de estado [82].

O planejador possui algumas propriedades editáveis, mas procurou-se mantê-la o mais próximo possível da configuração padrão:

- *StateSpace*: é o Espaço de Estados do planejador, que delimita suas fronteiras de atuação e os ângulos limites de rotação. O estado é caracterizado por posição e direção. O espaço de estados escolhido foi o "*stateSpaceSE2*" que é composto por vetores com três parâmetros (x, y, θ), sendo x e y coordenadas cartesianas e θ o ângulo de orientação e calcula a distância por meio da fórmula Euclidiana e utiliza interpolação linear para calcular translação e rotação do estado.
- *StateValidator*: é um validador do espaço de estados, que checa se um determinado estado pertence a um espaço válido (desocupado) do mapa. Pode ser utilizado para se criar um *costmap* para o veículo, que pode ser interpretado o quanto ele deve se manter afastado de obstáculos. O validador utilizado foi o "*validatorOccupancyMap*" que avalia se os estados são válidos ou não a cada 0,01 unidades de comprimento.
- *MaxNumTreeNodes*: número máximo de nós na árvore. Por *default* este valor é $1x10^4$.
- *MaxIterations*: número máximo de iterações. Por *default* este valor é $1x10^4$.
- *MaxConnectionDistance*: comprimento máximo de uma conexão entre dois nós da árvore. É o delta.
- *GoalReachedFcn*: função que avalia se o objetivo foi alcançado ou não.
- *GoalBias*: probabilidade do planejador de escolher o "estado de objetivo" durante o processo de amostragem dos estados, no espaço de estados. Por *default* este valor é 0,05 e não foi alterado para nenhuma simulação.

Novamente, um programa de MATLAB também foi desenvolvido para a execução das simulações com RRT, para criação dos mapas de ocupação binária e validação de seus espaços de estados, ajuste dos parâmetros de entrada, seleção dos

pontos inicial e objetivo, acionar a função "*plannerRRT"* e gerar as imagens das trajetórias.

As simulações são executadas com os valores *default* apresentados e com o delta ajustado em 0,5, que foi o valor no qual houve formação de trajetórias de forma consistente nos testes primários. A cada série de dez simulações, o delta é incrementado adquirindo os seguintes valores: 1, 2, 3, 5, 7, 10, 15, 20 e 25 unidades de comprimento. Estes valores foram escolhidos na busca por comportamentos diversos até o limite de 25 unidades de comprimento, que é o tamanho das arestas dos mapas.

O tempo de processamento é medido sobre a função "*plannerRRT*", responsável por gerar os nós, a árvore e planejar a trajetória. O mesmo é exibido na janela de comandos do MATLAB conforme Figura 4.7.

A quantidade de passos é dada diretamente pelo parâmetro "*pthObj*" na área de trabalho do MATLAB (Figura 4.7). Os passos, no caso desse planejador, são os estados da trajetória.

O comprimento da trajetória é obtido coletando-se os valores das coordenadas *x* e *y* dos estados, na matriz "*states*" do parâmetro "*pthObj*", mostrado nas Figuras 4.7 e 4.8 e aplicando a Equação 4.1 utilizando esses valores.

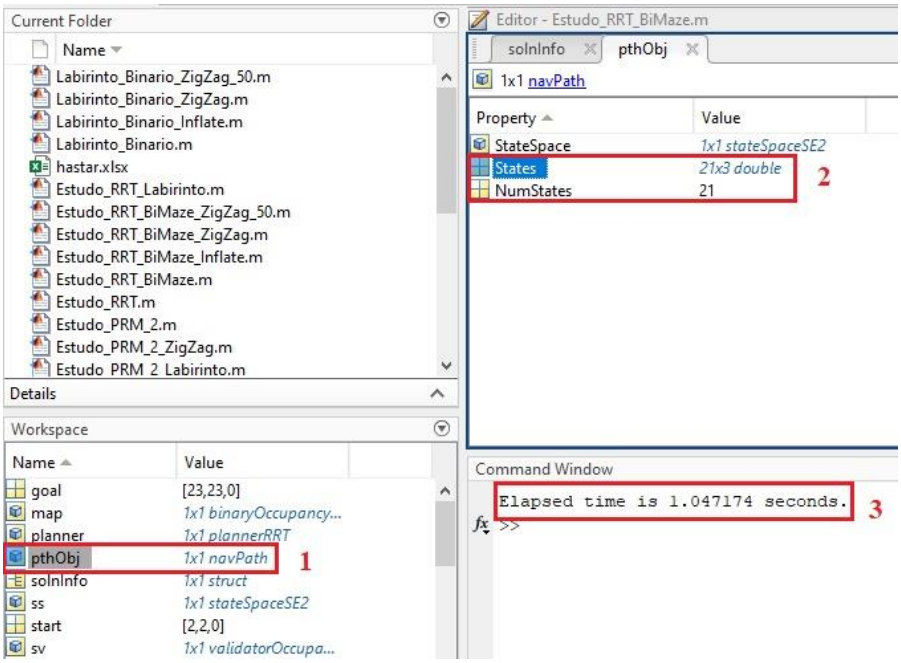
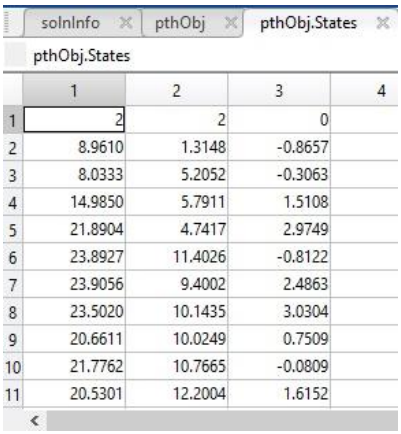


Figura 0.7 - Tela do MATLAB com dados das simulações com RRT: 1) "pthObj", função que contém o espaço de estados e os estados (passos); 2) matrizes contendo o número de estados e os estados em si; 3) tempo computado.



solnInfo | pthObj | pthObj.States

pthObj.States

	1	2	3	4
1	2	2	0	
2	8.9610	1.3148	-0.8657	
3	8.0333	5.2052	-0.3063	
4	14.9850	5.7911	1.5108	
5	21.8904	4.7417	2.9749	
6	23.8927	11.4026	-0.8122	
7	23.9056	9.4002	2.4863	
8	23.5020	10.1435	3.0304	
9	20.6611	10.0249	0.7509	
10	21.7762	10.7665	-0.0809	
11	20.5301	12.2004	1.6152	

Figura 0.8 - Estados de uma trajetória do RRT, sendo as colunas 1 e 2 as coordenadas nos eixos x e y e a coluna 3, o ângulo de orientação.

4.7
Hybrid A*

O planejador *Hybrid A** utilizado é o "*plannerHybridAStar*", um objeto da *Navigation Toolbox* do MATLAB, capaz de gerar trajetórias suaves em ambientes 2D. O planejador é focado em veículos não-holonômicos, como os automóveis e trabalha em mapas discretizados (*gridmaps*) [83].

As propriedades ajustáveis do planejador são as seguintes:

- *StateValidator*: o *Hybrid A** também trabalha com espaço de estados e necessita de um validador de estados, da mesma forma que o RRT. O espaço de estados utilizado pelo planejador é o "SE(2)" e o mesmo foi validado em cima do mapa de ocupação. A outra opção era validar em cima de um mapa de custo do veículo. Como o trabalho não aborda nenhum robô específico, escolheu-se a primeira opção.
- *MotionPrimitiveLength*: comprimento dos primitivos a serem gerados. Seu valor *default* é dado pela Equação 4.2, abaixo, e não pode exceder ¼ (um quarto) do comprimento da circunferência do raio mínimo de curvatura.

$$L_{prim} = \lceil \sqrt{2} \, . \, D_{cel} \rceil \qquad \text{Eq. 4.2}$$

onde:

- L_{prim} = comprimento do primitivo;
- D_{cel} = dimensão da aresta da célula do *gridmap*.

- *MinTurningRadius*: é o raio mínimo de curvatura do veículo, representado pela curvatura do primitivo. Seu valor é dado pela Equação 4.3:

$$R_{min} = \frac{(2\ .\ L_{prim})}{\pi} \qquad \text{Eq. 4.3}$$

onde:

  - R_{mim} = raio mínimo de curvatura;

- *NumMotionPrimitives*: número de primitivos por nó. Por *default* este valor é 5.
- *ForwardCost*: é um multiplicador de custo de avanço. O aumento de seu valor acarreta em penalizar o movimento de avanço e seu valor *default* é 1.
- *ReverseCost*: é um multiplicador de custo de deslocamento na direção reversa. O aumento de seu valor acarreta em penalizar o movimento em direção oposta ao objetivo. Seu valor *default* é 3.
- *DirectionSwitchingCost*: é um custo adicional para se mudar a direção do deslocamento. Por *default* seu valor é 0 (zero).
- *AnalyticExpansionInterval*: é o intervalo para tentativa de expansão analítica a partir do nó com o menor custo disponível. Por *default* este valor é igual a 5. O *Hybrid A** expande seus primitivos sempre a partir do nó com o menor custo (mais próximo do objetivo) da seguinte forma [83]: primeiro, o planejador expande os nós, dependendo da quantidade de primitivos especificada, da direção e da validade. Este ciclo se repete até que o valor do *AnalyticExpansionInterval* seja alcançado. Ou seja, se o parâmetro estive ajustado com seu valor *default* de 5, e o número de primitivos também for 5, o *Hybrid A** vai expandir um nó com cinco primitivos, dentro de um espaço válido (vazio) e na direção que for conveniente, 5 vezes. Depois, o planejador tenta executar uma expansão analítica para alcançar a pose de objetivo a partir da árvore, usando o modelo de Reeds-Shepp [84]. Se a tentativa falhar, o ciclo se repete. Logo, para o exemplo dado, a cada 5 expansões de nós, o planejador faz uma tentativa de alcançar o objetivo.

- *InterpolationDistance*: é a distância entre poses (posição e direção) interpoladas na trajetória. Seu valor *default* é 1 e, na prática, significa o tamanho do passo da trajetória ou o quanto ela avança de um nó para outro.

Assim como para as simulações com os dois planejadores anteriores, foi necessário desenvolver um programa para as simulações com *Hybrid A**, no qual, além de chamar o *plannerHybridAStar*, os mapas foram construídos e validados, as configurações de entrada ajustados e os pontos inicial e final, determinados.

O tempo de processamento é medido sobre a função *"refpath*", que chama o objeto *plannerHybridAStar* e quantidade de passos é obtida diretamente do parâmetro "*NumStates*", que faz parte da função *refpath*. Estes valores são exibidos na janela de comandos do MATLAB, conforme a Figura 4.9.

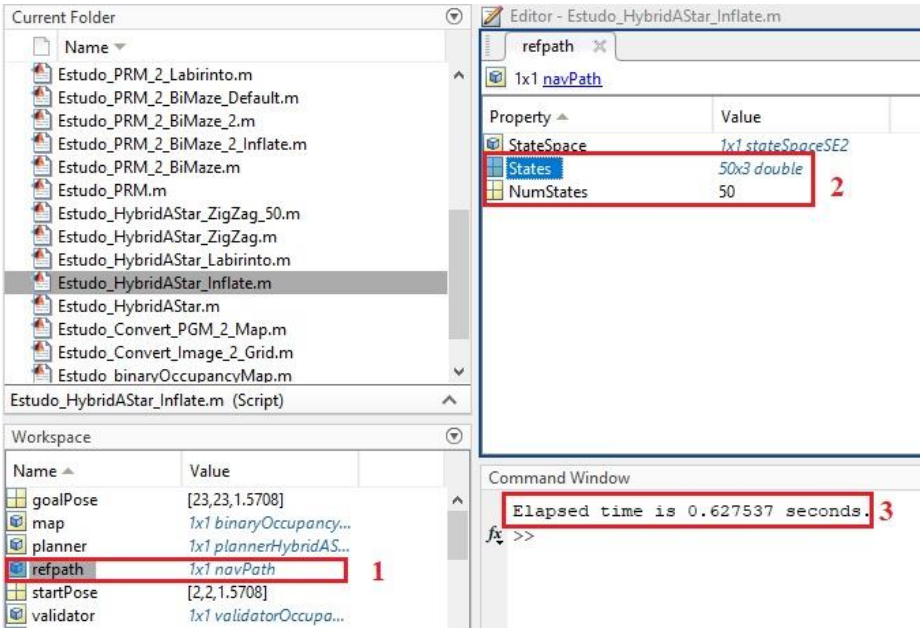


Figura 0.9 - Tela do MATLAB com dados acerca das simulações com *Hybrid A**: 1) "refpath", função que contém o estado de espaços e os estados (passos); 2) matrizes contendo o número de estados e os estados; 3) Tempo de processamento.

Uma vez que o intervalo de expansão analítica do planejador *Hybrid A** se dá segundo o modelo de Reeds-Shepp, o cálculo da trajetória também deve se dar segundo este modelo, pois a aplicação da Equação 4.1 só é possível aos trechos em que a trajetória é uma linha reta.

O modelo de movimento de Reeds-Shepp foi introduzido por Reeds e Shepp [84] em 1990 e é uma variação do modelo de movimento de Dubins, introduzido em 1957, no qual o movimento se dá em uma série curvas interligadas

chamada *geodésica*. Os movimentos da geodésica são compostos sempre por três elementos do tipo C, para curvas ou S para retas, na forma de CCC ou CSC e podem ser descritos por seis "palavras" compostas pelos seguintes caracteres: *l*(*left*), indicando movimento em sentido anti-horário; *r*(*right*), indicando movimento para em sentido horário e *s*(*straight*), indicando movimento em linha reta. São elas: *lrl, lsl, lsr, rlr, rsr* e *rsl*. Os movimentos circulares ocorrem em torno de um círculo de raio unitário e sempre inferiores a 2π (circunferência completa). São utilizadas notações subscritas para indicar os comprimentos dos arcos e segmentos na forma de $l_t s_u r_v$ e sobrescritos "+" ou "-" para indicar se o movimento é de avanço ou de retorno. Enquanto o modelo de Dubins admite apenas movimentos de avanço, o modelo de Reeds-Shepp admite movimentos de retorno também. A Figura 3.10 ilustra quatro exemplos de movimentos de Reeds-Shepp.

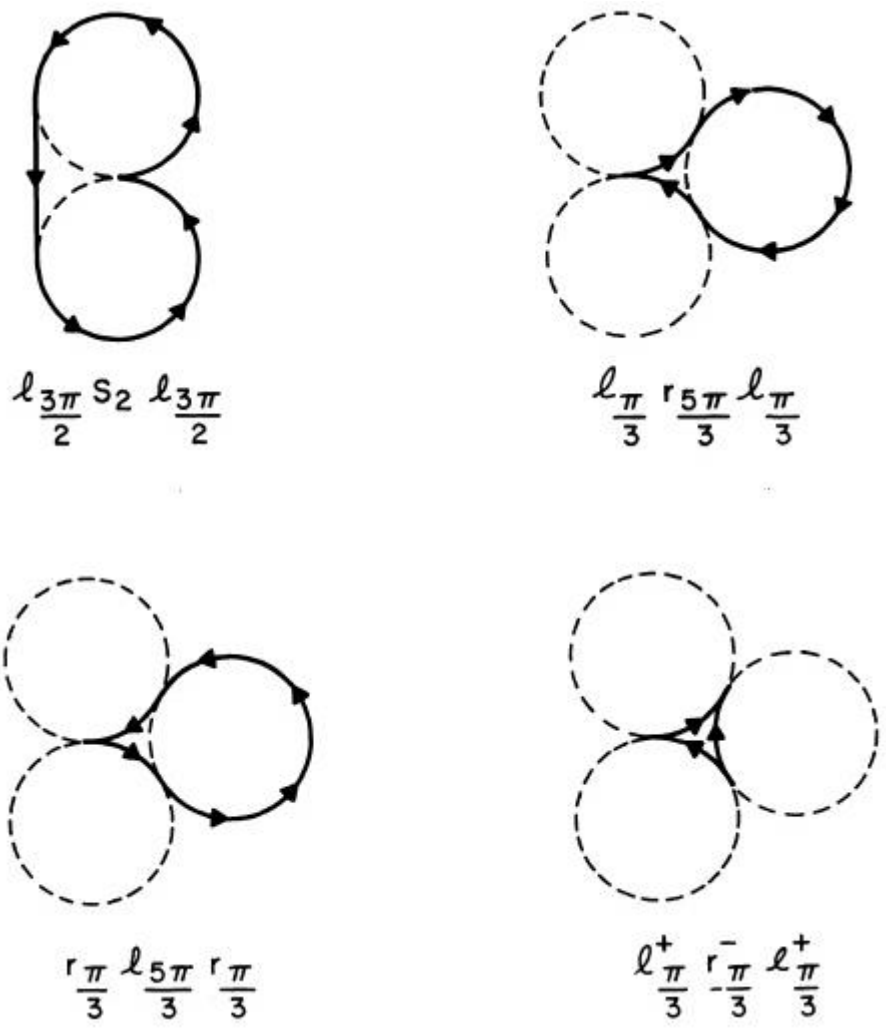
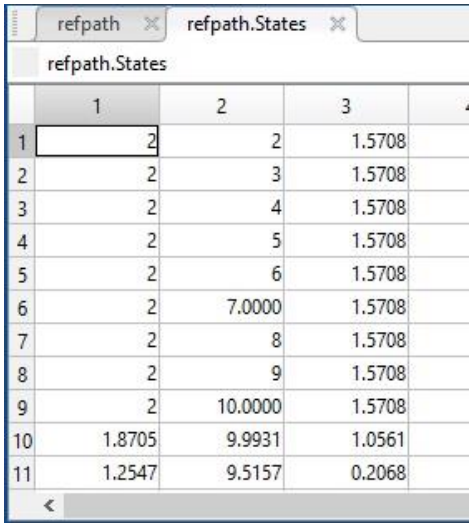


Figura 0.10: Exemplos de curvas de Reeds-Shepp.

Assim, o comprimento da trajetória é calculado a partir das coordenadas dos estados obtidos do parâmetro "*States*", conforme as Figuras 4.9 e 4.11, tendo como base o modelo de Reeds-Shepp.



refpath | refpath.States

refpath.States

	1	2	3
1	2	2	1.5708
2	2	3	1.5708
3	2	4	1.5708
4	2	5	1.5708
5	2	6	1.5708
6	2	7.0000	1.5708
7	2	8	1.5708
8	2	9	1.5708
9	2	10.0000	1.5708
10	1.8705	9.9931	1.0561
11	1.2547	9.5157	0.2068

Figura 0.11 - Estados de uma trajetória do *Hybrid A**, sendo as colunas 1 e 2 as coordenadas nos eixos x e y e a coluna 3, o ângulo de orientação.

A partir dessas coordenadas, o cálculo do comprimento é executado segundo o algoritmo a seguir:

- Definindo:
  - p = quantidade de passos.
  - r = raio mínimo de curvatura.
  - d = distância entre dois nós.
  - θ = ângulo entre dois nós.
  - li = comprimento entre dois nós.
  - L = Comprimento da trajetória.

Algoritmo 5 – Cálculo do Comprimento da Trajetória do *Hybrid A**

1 **carregar** tabela *refpath.States*
2 p ⟵ número de linhas de *refpath.States*
3 L ⟵ 0
4 *x(p)* ⟵ coluna 1 de *refpath.States*
5 *y(p)* ⟵ coluna 1 de *refpath.States*

6 **para** i = 2,..., p **fazer**

7 $d = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$

8 $\theta = 2\,.\,arcsen(\frac{d}{2r})$

9 **se** $\theta = \pi$ **então**

10 $li \longleftarrow d$

11 **senão**

12 $li \longleftarrow \theta\,.\,r$

13 **fim se**

14 $L = L + li$

15 **fim para**

16 **Retornar** L

4.8 Hardware e Sistema

Foram utilizados nas simulações um computador tipo *notebook* com as seguintes configurações:

- Processador: Intel® Core™ i5-5200U CPU @ 2.20GHz
- Memória instalada (RAM): 8,00 GB
- Tipo de sistema: Sistema Operacional Windows 10 Home de 64 bits, processador com base em x64.

A versão do MATLAB utilizada foi:

- R2019b Update 3 64bit

No próximo capítulo são apresentados os resultados das simulações.

5
Simulações no Mapa 1 - Labirinto

O presente capítulo apresenta os resultados das simulações realizadas com os planejadores Algoritmo de Dijkstra, *Dynamic Programming*, PRM, RRT e *Hybrid A**, no primeiro mapa, Labirinto, seguindo a metodologia apresentada no capítulo anterior, da seguinte forma: primeiro são mostrados os resultados de cada planejador no mapa Labirinto, seguido de uma análise comparativa entre eles.

5.1
Simulações com Algoritmo de Dijkstra no Labirinto

A primeira série de simulações computadas foi realizada com o parâmetro ajustado de 80 nós, pois uma quantidade menor de nós não foi suficiente para gerar resultados de forma consistente. Mesmo assim, das dez simulações realizadas, uma falhou, não gerando trajetórias. A partir de 90 nós, houve sucesso em todas as simulações. Os números de nós foram incrementos de dez em dez até o número de 150 nós. A Tabela 5-1 a seguir apresenta os resultados das simulações.

TABELA 0-1 – SIMULAÇÕES COM ALGORITMO DE DIJKSTRA NO LABIRINTO

Série	*nós*	$\overline{tp}$(s)	$\overline{passos}$	$\bar{d}$(u.c.)	Obs.
1	80	3,721	8,333	41,759	9 resultados
2	90	3,958	8,200	41,470	
3	100	4,854	9,200	40,471	
4	110	5,763	8,600	41,563	
5	120	6,863	9,400	40,570	
6	130	8,940	9,600	39,384	
7	140	9,048	9,600	39,700	
8	150	10,826	9,400	40,172	

Legenda:

nós = Número de nós que ocupam a área livre do mapa.

$\overline{tp}$ = Tempo de processamento médio, medido em segundos.

$\overline{passos}$ = Número médio de nós utilizados para conectar os pontos inicial e objetivo.

$\bar{d}$ = Comprimento médio das trajetórias, medido em unidades de comprimento (u.c.).

Os valores presentes na Tabela 5-1 são novamente apresentados de forma gráfica na Figura 5.1. Analisando o gráfico dos tempos médios de processamento, *tp(s)*, constata-se que o aumento do tempo acompanha o aumento do número de

nós, mas não de maneira proporcional. Observando-se por exemplo as sequências de 90 a 120 nós, o tempo médio cresce em uma proporção de aproximadamente 1 segundo a cada 10 nós acrescentados. Mas esse tempo salta para 2,077 segundos entre as sequências com 120 e 130 nós. O mesmo se observa entre as sequências com 140 e 150 nós, que tem um incremento de tempo de 1,778 segundos, enquanto que o incremento entre as sequências de 130 e 140 nós é de apenas 0,108 segundos. Em contrapartida, observando o gráfico dos comprimentos médios de trajetória *d(u.c.)*, verifica-se que não há diferenças significativas entre tais comprimentos. Da mesma forma, o gráfico da quantidade média de passos também mostra que este parâmetro não apresenta grandes variações, ficando entre uma média mínima de 8,2 e máxima de 9,7 passos, indicando que, independentemente de quantos nós forem gerados, a quantidade de conexões necessárias para ir do ponto inicial ao objetivo não varia muito, mesmo para trajetórias diferentes.

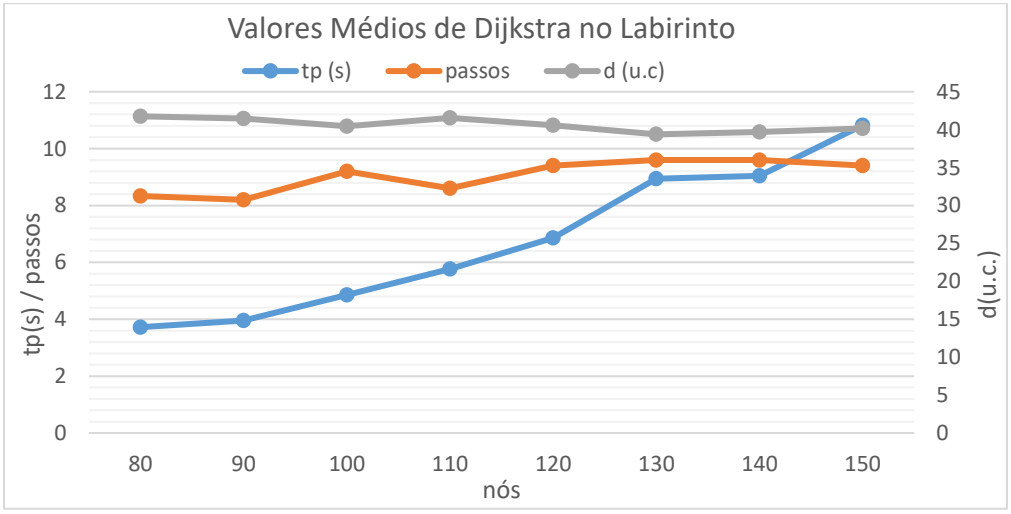


Figura 0.1 - Valores médios de tempo de processamento, quantidade de passos e comprimentos de trajetórias das simulações do Algoritmo de Dijkstra no Labirinto.

Uma vez que, devido à natureza aleatória da distribuição de nós, cada trajetória gerada é diferente das demais, faz-se necessário analisar mais de perto cada uma das simulações individuais. Assim, as Figuras 5.2, 5.3 e 5.4 apresentam os valores dos tempos de processamento, quantidade de passos e comprimento de cada trajetória individualmente.

A Figura 5.2, que apresenta os tempos de processamento, corrobora com a informação passada pelo gráfico da Figura 5.1, de que o tempo de processamento possui relação com o número de nós. Salvo as exceções nas quais há saltos dos

tempos, formando picos, os tempos de processamento de cada série aumentam conforme o número de nós também aumenta. Há um patamar nos tempos de processamento da série de 130 nós, na sétima e oitavas simulações, justificando o porquê das médias de tempo entre as séries de 130 e 140 nós apresentarem variações relativamente menores.

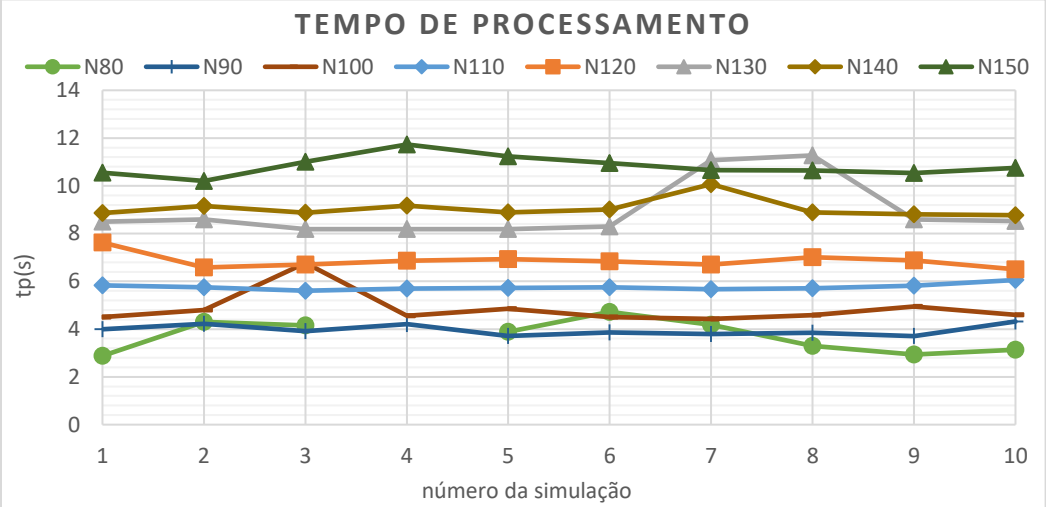


Figura 0.2 - Tempo de processamento de cada simulação do Algoritmo de Dijkstra no Labirinto.

Os gráficos das Figuras 5.3 e 5.4 demonstram a aleatoriedade da distribuição de nós, refletida nos passos e a consistência dos tamanhos das trajetórias que se concentram praticamente dentro de uma única faixa de valores. Estes gráficos também evidenciam que não existe relação entre número de nós com os passos ou os comprimentos das trajetórias.

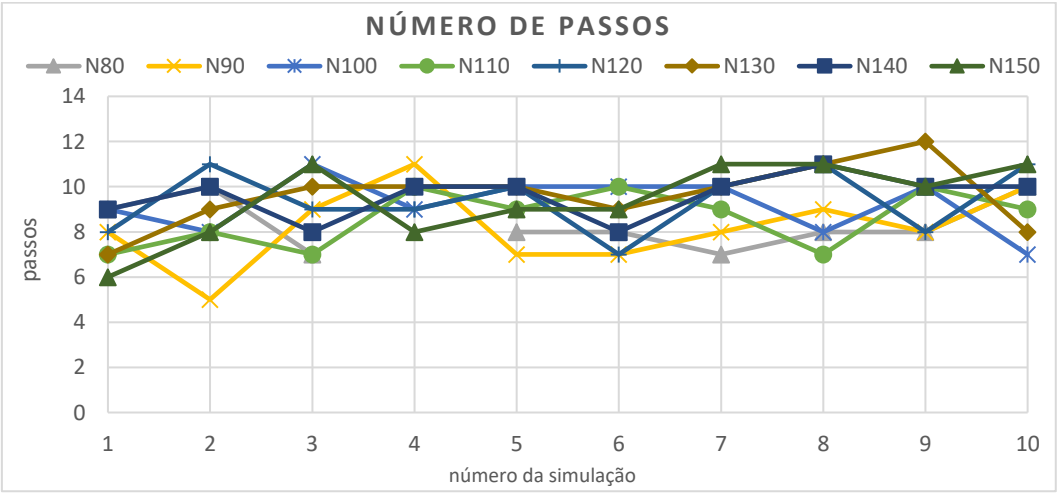


Figura 0.3 - Quantidade de passos de cada simulação do Algoritmo de Dijkstra no Labirinto.

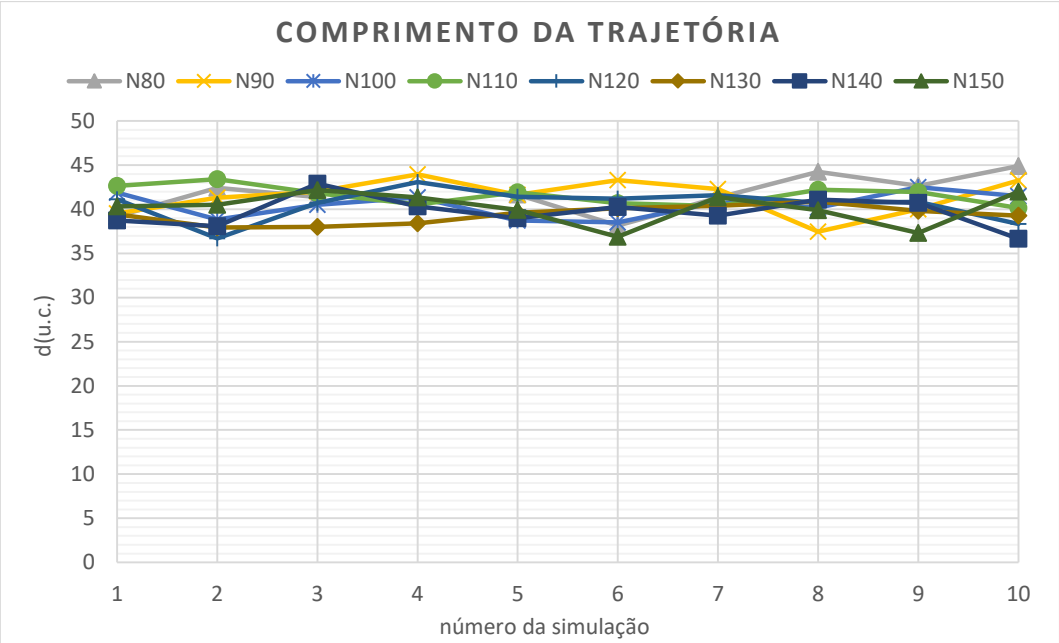
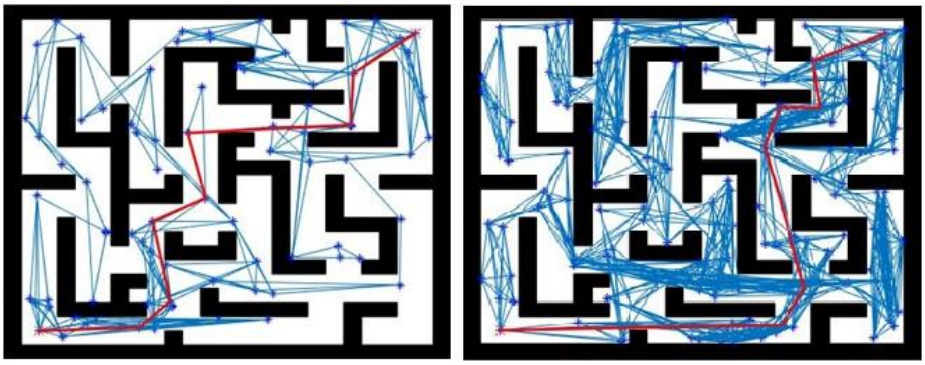


Figura 0.4 - Comprimento de trajetória de cada simulação do Algoritmo de Dijkstra no Labirinto.

Tendo como critério somente o tempo de processamento, que representa o custo computacional para se gerar a trajetória, a segunda série de simulações, com noventa nós, apresentou o melhor desempenho, dentre as simulações com 100% de aproveitamento. Caso o critério seja pelo tamanho da trajetória, a série com 130 nós, foi a que apresentou o menor tamanho médio de trajetória. Este critério, entretanto, só se justifica se as dimensões do robô forem muito menores que as da trajetória. Por exemplo, supondo um robô de limpeza doméstica de geometria circular com um metro de diâmetro e a unidade de comprimento do mapa em metros, a escolha do número de nós em função do tamanho da trajetória não se justifica, pois, a diferença entre o tempo de processamento médio entre as séries de simulações 2 (90 nós) e 6 (130 nós) é de 4,982 segundos, enquanto que a diferença de tamanhos médios de trajetória é de 2,086 metros. Ou seja, seria um custo computacional adicional de 55,73% em troca de um benefício relativamente pequeno. Se a unidade de comprimento for em quilómetros, tal critério poderia se justificar, pois, uma diferença de mais de dois quilómetros, para um robô de dimensões mil vezes menor, é significativa.

A Figura 5.5 apresenta as duas trajetórias geradas no menor e no maior tempo de processamento: a primeira simulação da série com 80 nós e a quarta simulação da série com 150 nós.



(a) (b)

Figura 0.5 - Simulações do Algoritmo de Dijkstra no Labirinto: (a) a primeira com 80 nós e (b) a quarta com 150 nós.

5.2
Simulações com Dynamic Programming no Labirinto

O algoritmo de *Dynamic Programming* utilizado nas simulações divide o mesmo programa que o Algoritmo de Dijkstra, apesar dos planejadores funcionarem distintamente.

Assim como ocorreu com as simulações com o Algoritmo de Dijkstra, não houve resultados consistentes com menos de 80 nós. No caso, esta primeira série de simulações gerou oito resultados. A partir da segunda série, com 90 nós, todas as simulações foram bem-sucedidas. O número de nós entre as séries foi incrementado de 10 em 10, até que se completasse 150 nós. Os valores médios de tempo de processamento, quantidade média de passos e comprimento médio das trajetórias são apresentadas na Tabela 5-2.

TABELA 0-2 – SIMULAÇÕES COM DYNAMIC PROGRAMMING NO LABIRINTO

Sequencial	*nós*	$\overline{tp}$(s)	$\overline{passos}$	$\bar{d}$(u. c.)	Obs.
1	80	3,075	8,500	45,854	8 resultados
2	90	3,871	9,900	47,814	
3	100	4,866	10,100	47,584	
4	110	6,550	8,900	46,306	
5	120	7,671	9,400	44,239	
6	130	8,459	8,600	43,206	
7	140	9,098	7,700	43,957	
8	150	10,832	8,800	43,251	

Legenda:

nós = Número de nós que ocupam a área livre do mapa.

$\overline{tp}$ = Tempo de processamento médio, medido em segundos.

$\overline{passos}$ = Número médio de nós utilizados para conectar os pontos inicial e objetivo.

$\bar{d}$ = Comprimento médio das trajetórias, medido em unidades de comprimento (u.c.).

A Figura 5.6 apresenta os gráficos do tempo médio de processamento, *tp(s)*, que também aumenta conforme o número de nós; da quantidade de passos, que apresenta uma diferença máxima de 2,3 passos médios; e do comprimento da trajetória *d(u.c.)*, mostrando que os comprimentos médios das trajetórias possuem uma pequena variação dentro de um faixa com diferença máxima de 3,575 unidades de comprimento. Assim como nas simulações com Algoritmo de Dijkstra, a quantidade de passos e o comprimento da trajetória demonstram não ter relação com o número de nós.

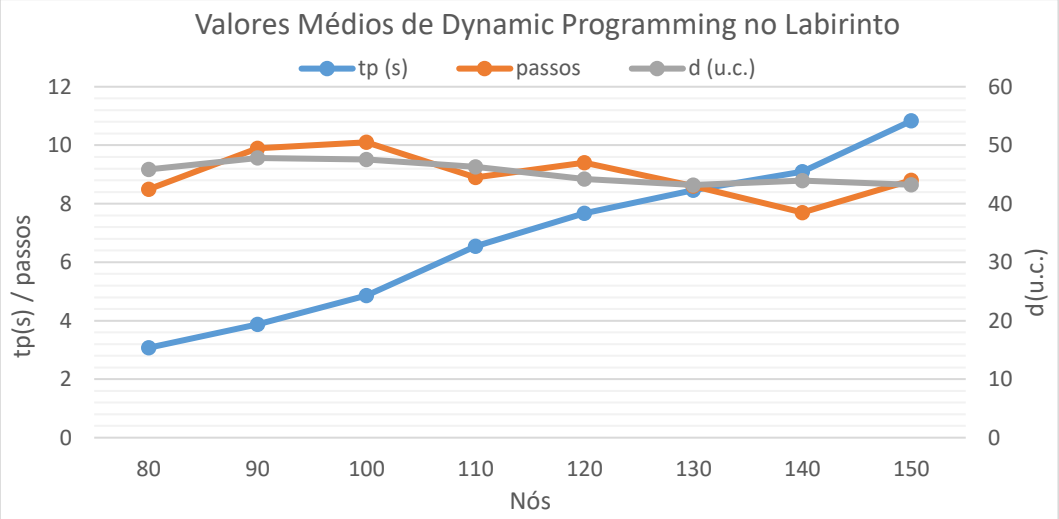


Figura 0.6 - Valores médios de tempo de processamento, quantidade de passos e comprimento de trajetórias das simulações do *Dynamic Programming* no Labirinto.

As Figuras 5.7, 5.8 e 5.9 apresentam os valores individuais do tempo de processamento, quantidade de passos e comprimento da trajetória de cada simulação. O gráfico da Figura 5.7 também mostra que há relação entre o tempo de processamento e o número de nós. Ocorrem apenas três exceções nas quais uma simulação com um número menor de nós superou o tempo de execução de uma com mais nós: a quarta simulação com 100 nós, a décima com 120 nós e a quinta com 140 nós. No caso das simulações com 100 nós e 140 nós, houve dois picos, mas no caso da simulação de 120 nós, houve uma oscilação ascendente nos tempos de processamento nas quarta e quinta simulações e os tempos de processamento das

simulações com 120, 130 e 140 nós são muito próximos. Observando o gráfico das quantidades de passos, Figura 5.8, nota-se que em algumas simulações, o parâmetro apresenta maiores oscilações. Isso ocorre porque o mapa Labirinto apresenta várias opções de rotas e a distribuição de nós no mapa é aleatória. Também é interessante notar como algumas simulações como, por exemplo, a de 100 e 150 nós apresentam comportamentos mais estáveis. A Figura 5.9 demonstra que os tamanhos de trajetórias se concentram dentro de uma faixa cuja amplitude aproximada é de 20 unidades de comprimento. Uma curiosidade mostrada no gráfico é que tanto a maior quanto a menor trajetórias ocorrem na sexta e nona simulação com 90 nós, respectivamente.

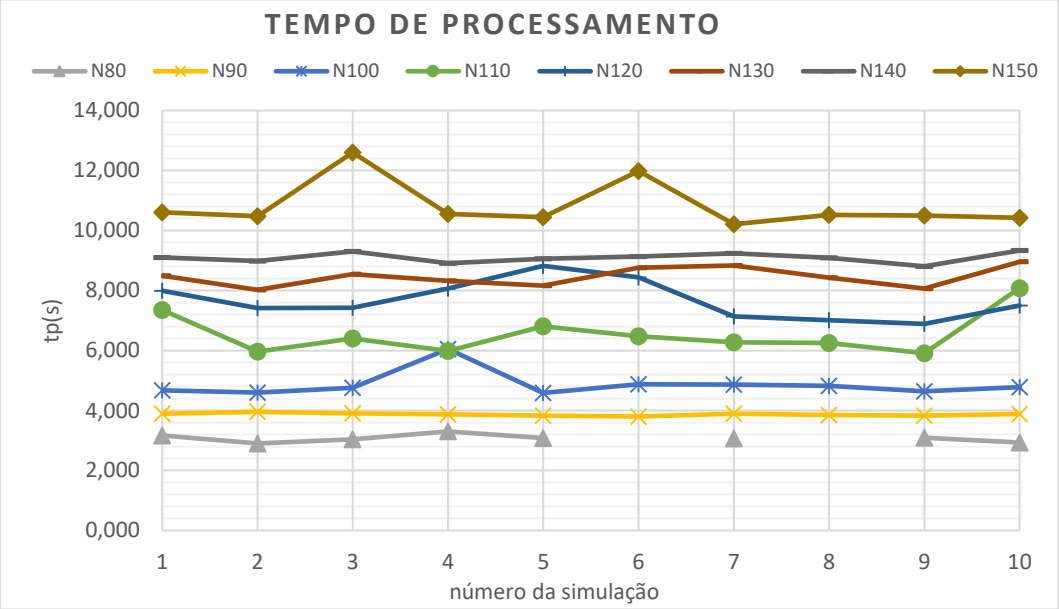


Figura 0.7 - Tempo de processamento de cada simulação do *Dynamic Programming* no Labirinto.

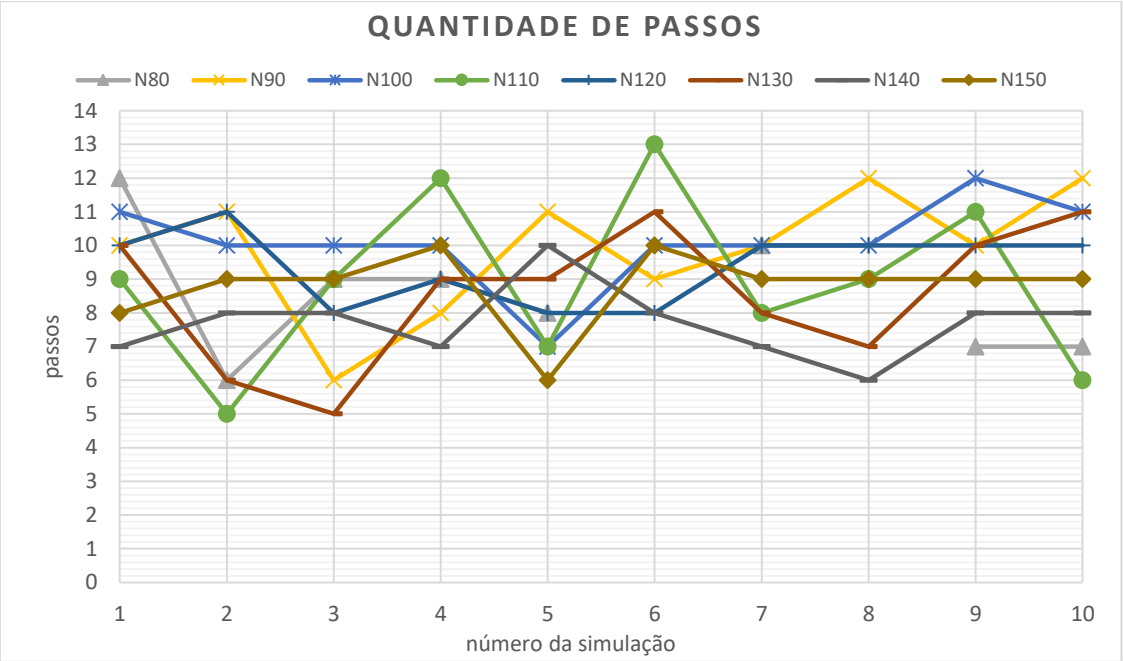


Figura 0.8 - Quantidade de passos de cada simulação do *Dynamic Programming* no Labirinto.

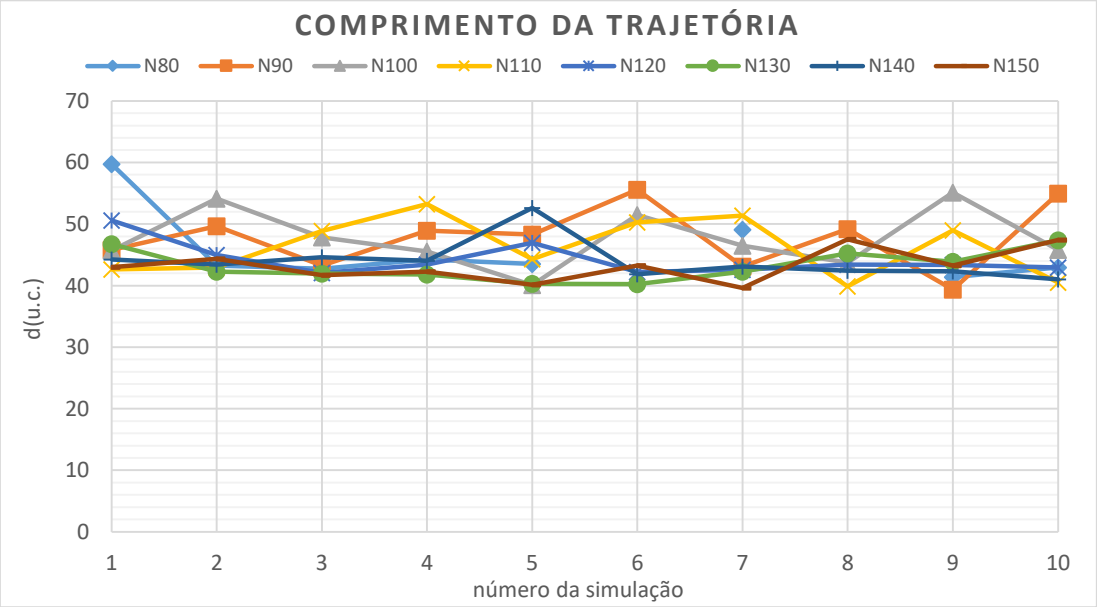
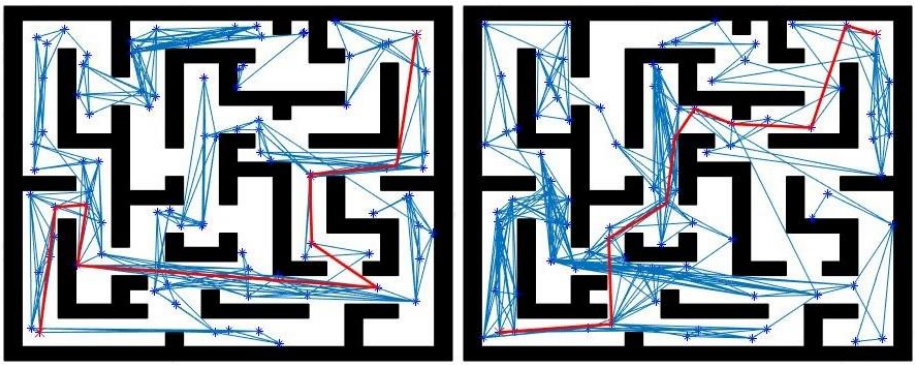


Figura 0.9 - Comprimento de trajetória de cada simulação do *Dynamic Programming* no Labirinto.

A Figura 5.10 apresenta a sexta e a nona simulação com 90 nós, que são respectivamente a maior e a menor trajetórias geradas.



(a) (b)

Figura 0.10 - Simulações do *Dynamic Programming* com 90 nós: (a) sexta e (b) nona simulação.

5.3
Simulações com PRM no Labirinto

Nas simulações com o PRM, o parâmetro variável é o "delta", que é a distância máxima entre dois nós que compõem um mapa de rotas. Seu valor *default*, que foi utilizado como entrada para a primeira série de simulações, é de 10 unidades de comprimento. A cada nova série de simulações o delta foi incrementado em 5 unidades de comprimento, até que se atingir o valor de 25 u.c, que é a dimensão das arestas do mapa. No total, foram realizadas quarenta simulações. Para as presentes simulações, o número de nós foi ajustado para iniciar em 50 nós e caso não se completasse a trajetória, mais dez nós são adicionados automaticamente.

A Tabela 5-3 apresenta dos valores médios do número de nós, tempo de processamento, quantidade de passos e comprimento de trajetórias.

TABELA 0-3 – SIMULAÇÕES COM PRM NO LABIRINTO

Série	Δ	$\overline{nós}$	$\overline{tp}(s)$	$\overline{passos}$	$\bar{d}(u.c.)$
1	10	62	0,057	10,80	44,302
2	15	66	0,060	10,20	44,783
3	20	59	0,113	10,60	47,176
4	25	53	0,094	10,90	48,818

Legenda:

Δ = delta.

$\overline{nós}$ = Número de nós médios gerados durante as simulações.

$\overline{tp}$ = Tempo de processamento médio, medido em segundos.

$\overline{passos}$ = Número médio de nós utilizados para conectar os pontos inicial e objetivo.

$\bar{d}$ = Comprimento médio das trajetórias, medidos em unidades de comprimento (u.c.).

A Figura 5.11 apresenta os gráficos dos valores médios do tempo de processamento, quantidade de passos e comprimento da trajetória. De acordo com os gráficos o único parâmetro que aparenta ter relação com o delta é o comprimento médio das trajetórias.

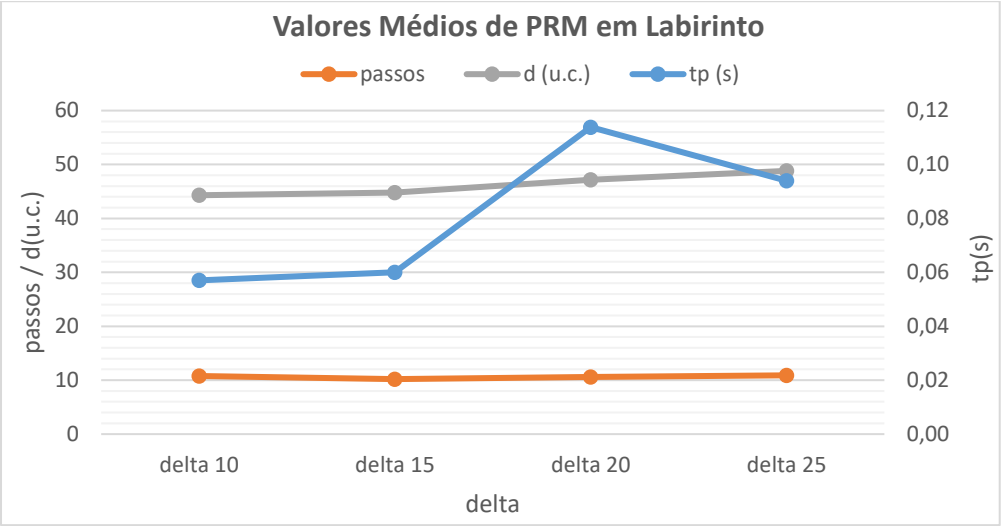


Figura 0.11 - Valores médios de tempo de processamento, quantidade de passos e comprimento de trajetórias das simulações do PRM no Labirinto.

As Figuras 5.12 a 5.14 apresentam os valores, para cada simulação, do tempo de processamento, quantidade de passos e comprimento da trajetória.

Pelo gráfico da Figura 5.12, o tempo de processamento das simulações com os menores deltas, 10 e 15, são muito próximas e abaixo dos tempos das outras duas séries com deltas maiores. Não se pode afirmar, contudo, de que este parâmetro é determinado pelo valor do delta pois os tempos de processamento da série com delta 20 são superiores aos da série com delta 25 e apresenta maiores oscilações também. Vale ressaltar que os tempos de processamento das simulações com PRM são muito pequenos, menos de um segundo, e podem ser afetados por algum processo interno do computador que estiver rodando a simulação.

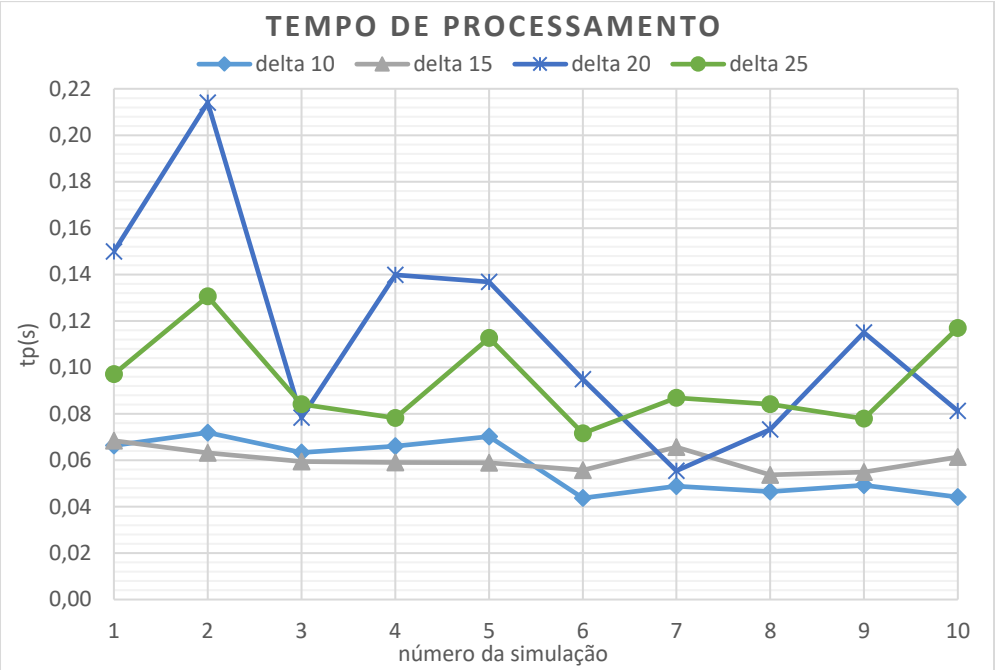


Figura 0.12 - Tempo de processamento de cada simulação do PRM no Labirinto.

A quantidade de passos e o comprimento da trajetória, apresentadas nas Figuras 5.13 e 5.14 respectivamente, se concentram dentro de uma mesma faixa de valores, independentemente do delta. No caso dos passos, um pico na última simulação determinou que a série com delta 25 tivesse a maior média. Já para os comprimentos da trajetória, as simulações com deltas 20 e 25 em atingiram maiores picos, sendo que neste último, permaneceu por três simulações consecutivas com comprimentos de trajetórias acima das demais, ocasionando sua maior média. Algo a se observar tanto no o gráfico da Figura 5.11, série *d*(*u.c.*) quanto o da Figura 5.14 é a proximidade entre os comprimentos das simulações com deltas 10 e 15.

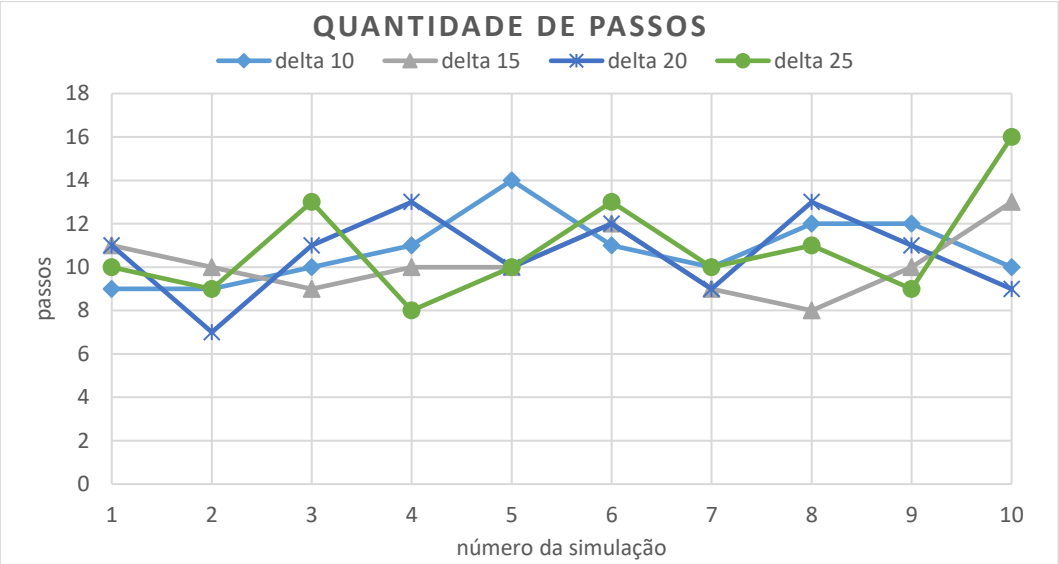


Figura 0.13 - Quantidade de passos de cada simulação do PRM no Labirinto.

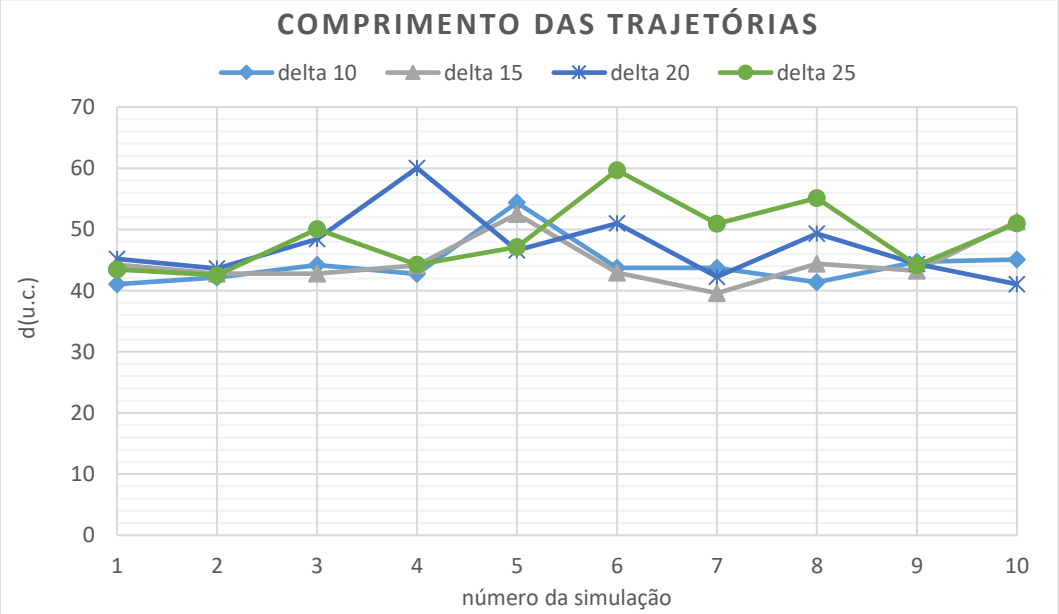


Figura 0.14 - Comprimento de trajetória de cada simulação do PRM no Labirinto.

A Figura 5.15 apresenta o gráfico do comprimento de trajetória versus quantidade de passos, evidenciando que existe uma relação entre esses dois parâmetros. Uma vez que o planejador PRM sempre tentará traçar a trajetória com a quantidade inicial de nós e somente se houver falha este número é incrementado, é razoável supor que o aumento do número de nós acarreta também o aumento da quantidade de passos. Da mesma forma há então a possibilidade de o comprimento da trajetória também ser afetado por este fenômeno. Diante disto e do

comportamento semelhante do comprimento e da quantidade de passos, viu-se por bem avaliar a relação entre estes parâmetros, o que acabou se confirmando.

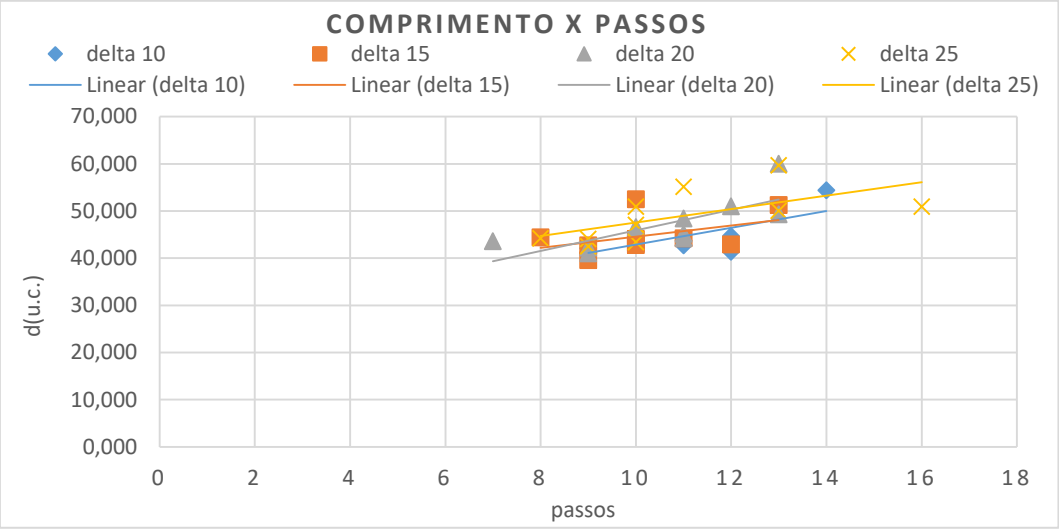
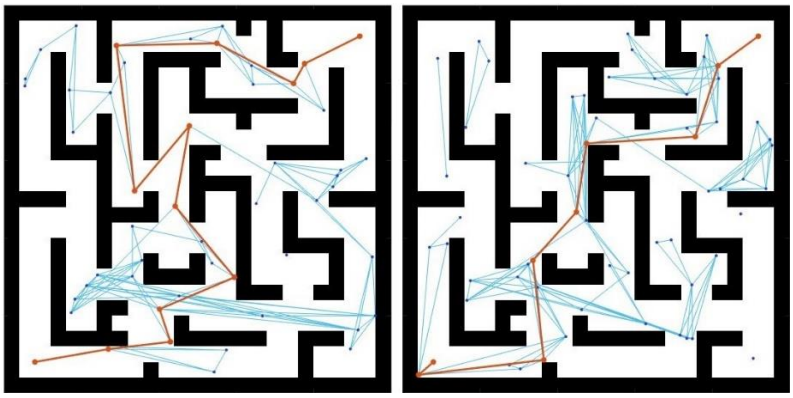


Figura 0.15 - Comprimento de trajetória versus número de nós do PRM no Labirinto.

A seguir, na Figura 5.16 são apresentadas a maior trajetória gerada com o PRM no mapa Labirinto, na quarta simulação da série com delta 20 e a menor trajetória, gerada na sétima simulação da série com delta 15.



(a) (b)

Figura 0.16 - Simulações do PRM no Labirinto: (a) quarta, com delta 20 e (b) sétima, com delta 15.

5.4
Simulações com RRT no Labirinto

O planejador RRT do MATLAB utilizado nas simulações, permite alterar o número máximo de nós e de iterações, assim como o delta, que é a distância máxima

entre os nós. Tanto o primeiro parâmetro, quanto o segundo, são, por *default*, ajustados para 10.000 (dez mil). Estes valores foram mantidos em todos as simulações, pois, conforme é mostrado na Tabela 5-4, os números de nós e iterações geradas nas simulações foram muito inferiores a estes. O valor inicial do delta foi de 0,5 unidades de comprimento, pois não foi possível gerar trajetórias seu o valor *default* de 0,1 e seu incremento se deu conforme a Tabela 5-4, até o valor de 25 unidades de comprimento.

Diferentemente dos três planejadores já vistos, o RRT sempre gera a mesma rota para um mesmo valor de delta em um mesmo mapa. Logo, cada simulação de uma mesma série terá o mesmo número de nós, comprimento de trajetórias iguais e a mesma quantidade de passos. Somente o tempo de processamento difere de uma para outra, sendo o único parâmetro apresentado na forma de média.

TABELA 0-4 – SIMULAÇÕES COM RRT NO LABIRINTO

Seq.	*max nós*	*max iter*	Δ	*nós*	*iter*	$\overline{tp}(s)$	*passos*	*d(u.c.)*
1	1 x 10^4	1 x 10^4	0,5	822	1554	0,719	95	46,071
2	1 x 10^4	1 x 10^4	1	165	359	0,131	42	40,223
3	1 x 10^4	1 x 10^4	2	60	139	0,050	22	39,267
4	1 x 10^4	1 x 10^4	3	74	247	0,081	16	42,217
5	1 x 10^4	1 x 10^4	5	77	331	0,091	13	45,150
6	1 x 10^4	1 x 10^4	7	274	835	0,299	21	61,539
7	1 x 10^4	1 x 10^4	10	263	835	0,274	17	45,738
8	1 x 10^4	1 x 10^4	15	195	652	0,216	15	52,603
9	1 x 10^4	1 x 10^4	20	195	652	0,226	15	52,603
10	1 x 10^4	1 x 10^4	25	195	652	0,206	15	52,603

Legenda:

max nós = número máximo de nós.

max iter = número máximo de iterações.

Δ = delta.

nós = número de nós gerados durante as simulações.

iter = iterações geradas durante as simulações.

$\overline{tp}$ = tempo de processamento médio, medido em segundos.

passos = número médio de nós utilizados para conectar os pontos inicial e objetivo.

d = Comprimento da trajetória, medido em unidades de comprimento (u.c.).

A Figura 5.17 apresenta os gráficos de tempo de processamento médio, *tp*(*s*), quantidade de passos e comprimento das trajetórias, *d(u.c.)* gerados a partir

dos valores apresentados na Tabela 5-4. Nota-se que há semelhança entre as curvas de tempo e quantidade de passo. Tal relação é melhor analisada mais adiante. Nenhum dos três parâmetros, entretanto, indica qualquer relação com o delta. Percebe-se, todavia, que um delta muito pequeno, no caso 0,5, acarreta em valores maiores, passos e tempo de processamento. Isso ocorre porque quanto menor a distância máxima que dois nós podem se conectar, maior será o número de nós necessários para se completar uma trajetória. Da mesma forma, quanto maiores forem os números de nós, maiores serão as conexões e, consequentemente, maior será o tempo necessário para se construir a árvore.

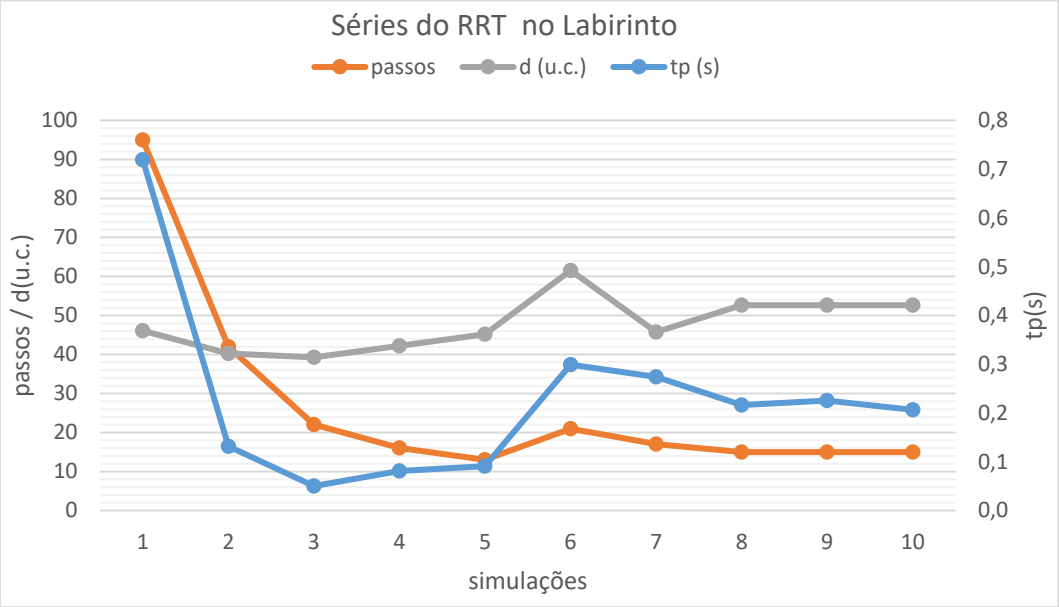


Figura 0.17 - Tempo de processamento médio, quantidade de passos e comprimento de trajetória das séries de simulações do RRT no mapa Labirinto.

A Figura 5.18 apresenta os tempos de processamento individuais de cada simulação. Os tempos são coerentes com as médias apresentadas no gráfico da Figura 5.17 e apresentam baixa amplitude, sobretudo nas simulações com os menores tempo. A exceção fica por conta da série com delta 0,5 que, além de apresentar valores destoantes das demais, também apresentas oscilações com amplitudes maiores.

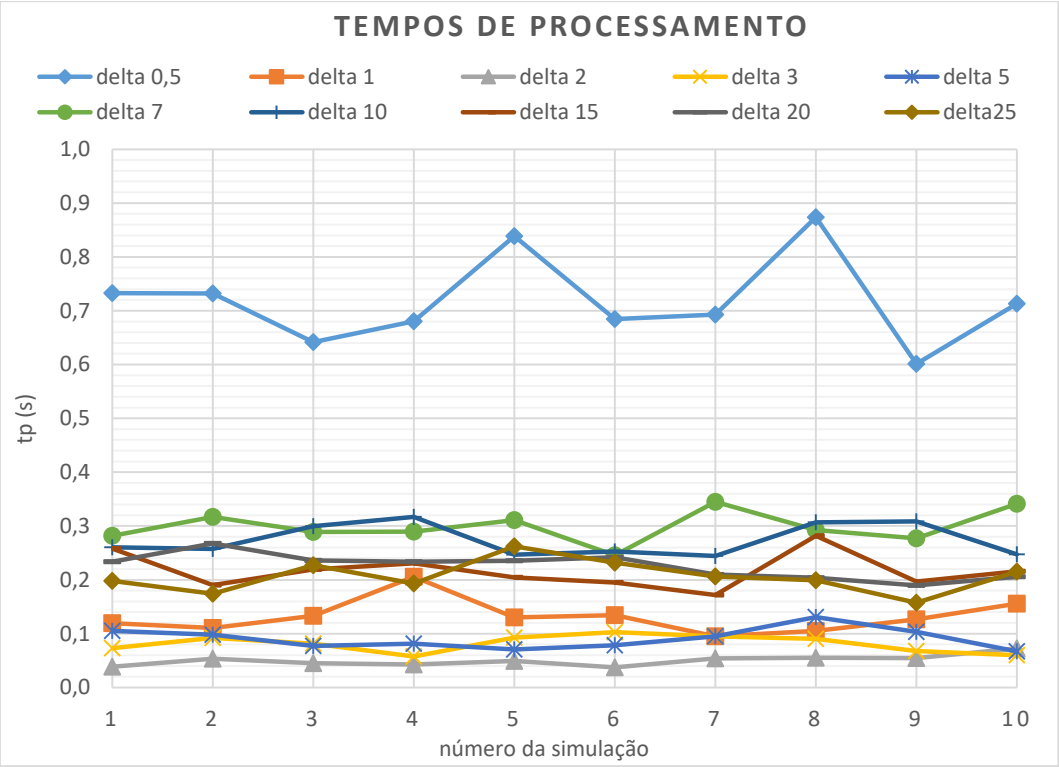


Figura 0.18 - Tempo de processamento de cada simulação individual com RRT no mapa

A Figura 5.19 apresenta o gráfico dos tempos médios de processamento versus as quantidades de passos. Apesar da semelhança entre suas curvas da Figura 5.17, o gráfico da Figura 5.19 demonstra não haver relação entre estes parâmetros.

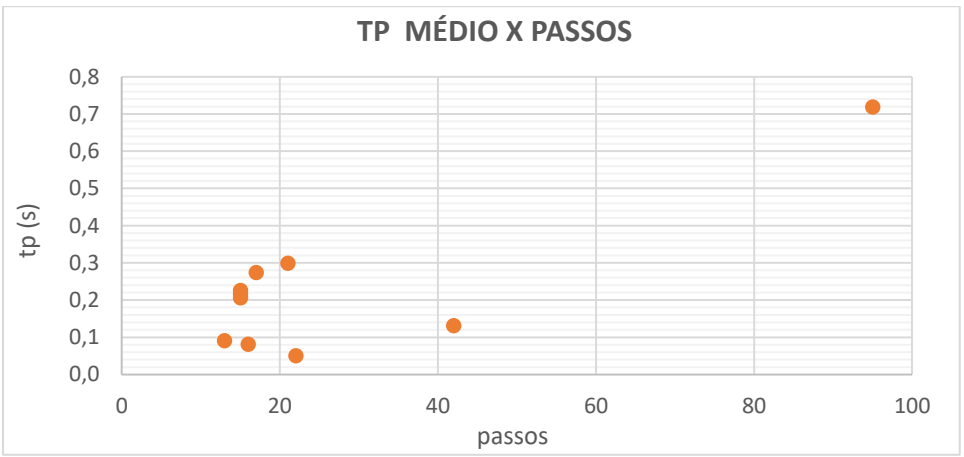
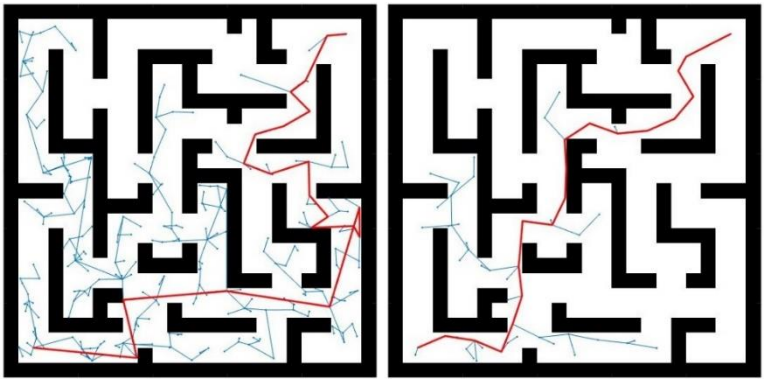


Figura 0.19 - Tempos médios de processamento versus quantidade de passos nas simulações do RRT no Labirinto.

A seguir, é apresentada na Figura 5.20, a trajetória gerada com delta igual a 7, que é a maior do RRT no mapa Labirinto e a menor trajetória, gerada na série com delta 2.



(a) (b)

Figura 0.20 - Simulações do RRT no Labirinto: (a) com delta 7 e (b) com delta 2.

5.5
Simulações com Hybrid A* no Labirinto

Conforme mencionado no capítulo anterior, o planejador *Hybrid A** do MATLAB possui muitos parâmetros editáveis. Foram realizadas dez séries de simulações sendo que a primeira série foi executada com os valores *default* do *Hybrid A**. A cada nova série, um ou mais desses parâmetros foram sutilmente modificados conforme a Tabela 5-5. Ao final, na última simulação, ajustaram-se todos os parâmetros que causaram um impacto positivo nas simulações. Os parâmetros diferentes do *default* de cada simulação, estão marcados de amarelo. Vale mencionar que assim como o RRT, uma trajetória criada com o *Hybrid A** com os mesmos parâmetros, sempre vai se repetir. Logo, em uma série de dez repetições, o único parâmetro diferente em cada uma delas é o tempo de processamento.

TABELA 0-5 – SIMULAÇÕES COM HYBRID A* NO LABIRINTO

Ser	*Lprim*	*Rmin*	*Nprim*	*FC*	*RC*	*DC*	*ID*	*AEI*	$\overline{tp}(s)$	*passos*	*d(u.c.)*
1	2	2	5	1	3	0	1	5	1,099	54	53,714
2	3	2	5	1	3	0	1	5	0,474	64	63,546
3	2	2	7	1	3	0	1	5	0,853	44	44,162
4	3	2	7	1	3	0	1	5	0,526	55	54,379
5	2	2	5	1	4	0	1	5	1,036	56	54,297
6	2	2	5	1	3	1	1	5	0,986	54	53,968
7	2	2	5	1	3	0	2	5	0,897	27	64,884
8	2	2	5	1	3	0	1	4	0,992	54	53,634
9	4	3	5	1	3	0	1	5	0,859	87	85,771
10	3	2	7	1	3	1	2	5	0,568	28	53,993

Legenda:

Lprim = comprimento dos primitivos.

Rprim = raio mínimo de curvatura.

Nprim = número máximo de primitivos por nó.

FC = custo de avanço (do inglês *forward cost*).

RC = custo de retorno (do inglês *reverse cost*).

DC = custo de mudança de direção (do inglês *direction switching cost*).

ID = distância de interpolação (do inglês *interpolation distance*).

AEI = intervalo de expansão analítica (do inglês *analytic expansion interval*).

$\overline{tp}$ = tempo de processamento médio, medido em segundos.

passos = número médio de nós utilizados para conectar os pontos inicial e objetivo.

d = Comprimento da trajetória, medido em unidades de comprimento (u.c.).

A Figura 5.21 a seguir, apresenta os gráficos dos tempos de processamento médios, quantidade de passos e comprimento da trajetória de casa série presente na Tabela 5-5. Destaca-se a nona série de simulações, apresentando tanto o maior número de passos, como também o maior comprimento de trajetória. Este parâmetro, aliás, parece acompanhar a quantidade de passos. A relação entre eles é vista com maiores detalhes mais adiante.

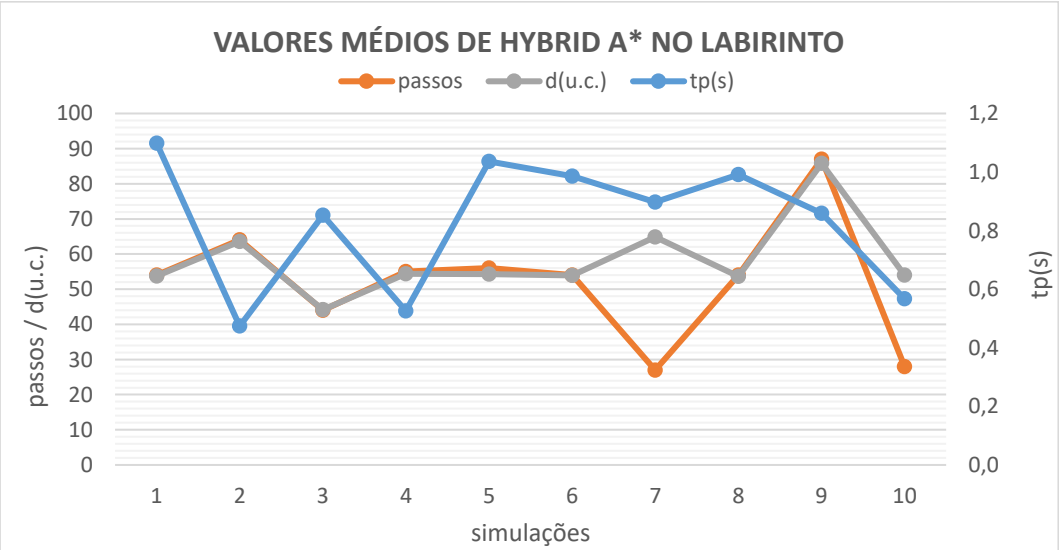


Figura 0.21 - Tempo de processamento médio, quantidade de passos e comprimento das trajetórias das simulações do *Hybrid A** no Labirinto.

O gráfico da Figura 5.22, abaixo, mostra os tempos de processamento de cada simulação individual. O mesmo confirma a correspondência entre as médias apresentadas na Figura 5.21, pois, os tempos, em sua maioria, apresentam uma

baixa amplitude, com exceção de dois picos apenas na segunda simulação da sexta série e terceira simulação da oitava série.

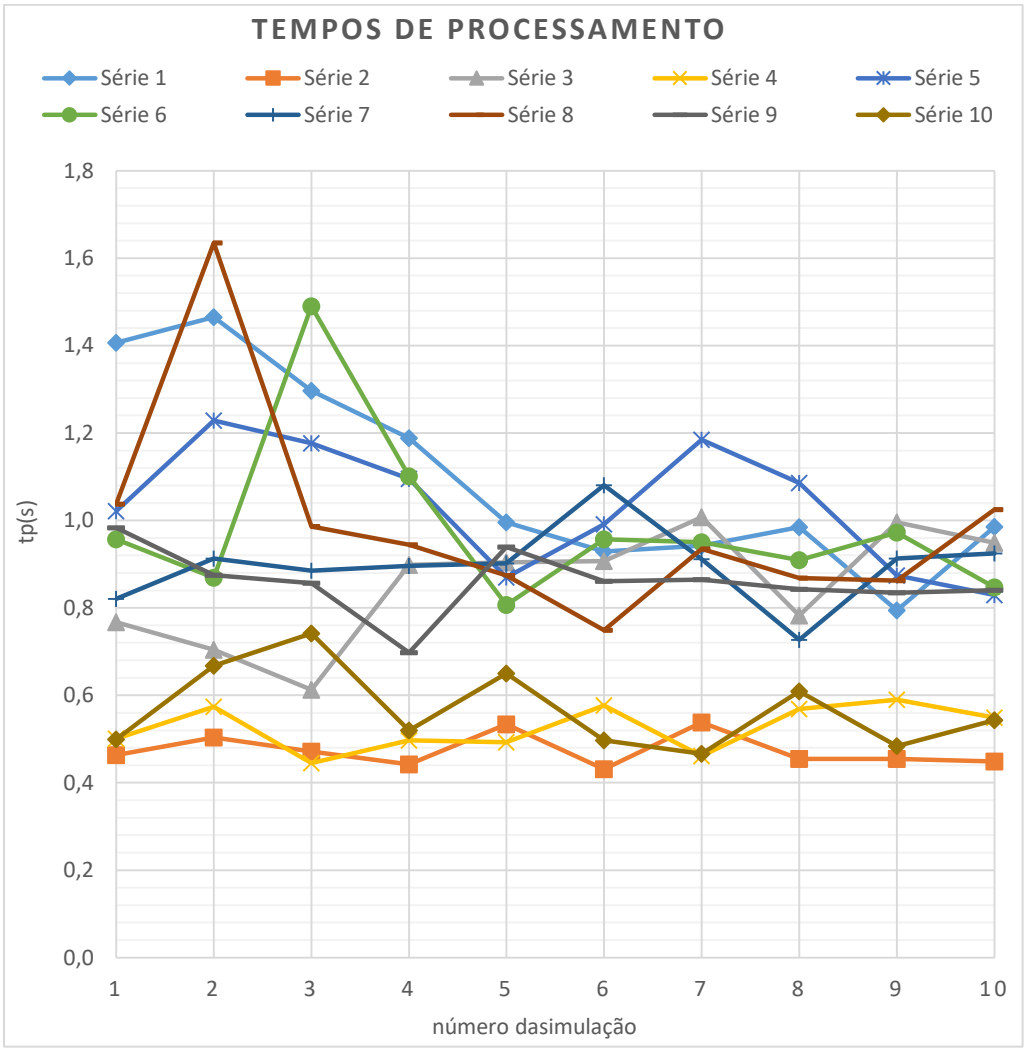


Figura 0.22 - Tempo de processamento de cada simulação com *Hybrid A** no Labirinto.

Os parâmetros de quantidades de passos e comprimento da trajetória, demonstraram, por suas curvas na Figura 5.21, possuírem uma relação entre si. O gráfico da Figura 5.23, a seguir procura averiguar esta relação, que demonstra ser verdadeira. Há duas exceções, que são da sétima e décima série, representados pelos dois pontos mais à esquerda. Essas séries possuem em comum o parâmetro "distância de interpolação" igual a 2. Este parâmetro pode ser interpretado como o comprimento aproximado de cada passo da trajetória e, por *default*, seu valor é 1.

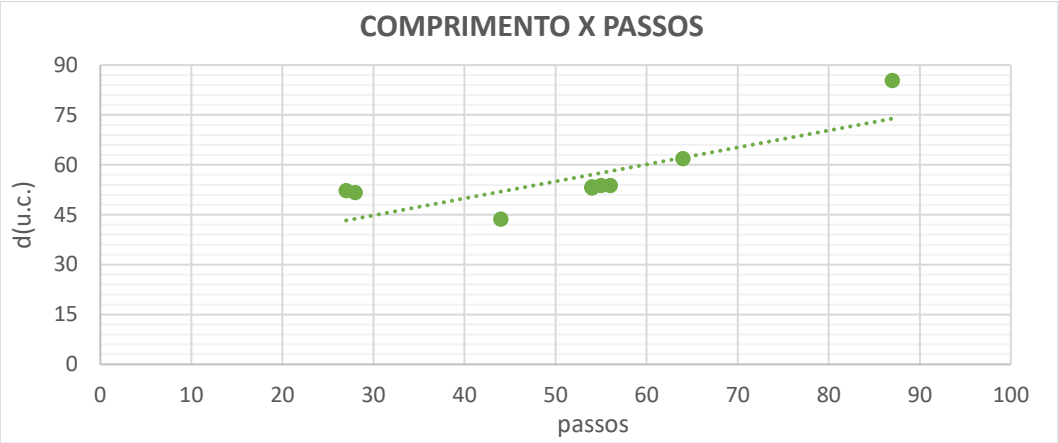
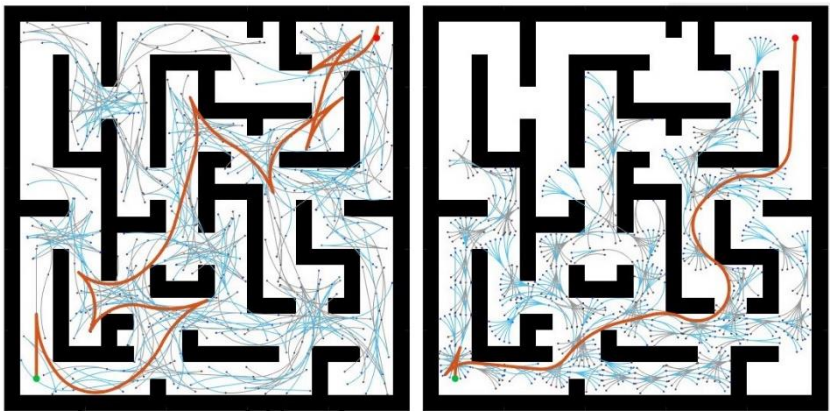


Figura 0.23 - Comprimento de trajetória versus quantidade de passos das simulações do *Hybrid A** no Labirinto.

A seguir é apresentada, na Figura 5.24, a maior e a menor trajetória geradas nas simulações com *Hybrid A** no mapa Labirinto, que correspondem à nona e terceira simulações respectivamente.



(a) (b)

Figura 0.24 - Simulações de *Hybrid A** no Labirinto: (a) nona série e (b) terceira série.

5.6
Comparações Entre as Simulações no Labirinto

Concluindo as simulações no mapa Labirinto, procede-se então à comparação dos parâmetros medidos.

A comparação entre os tempos de processamento e comprimento das trajetórias é bastante lógica pois são parâmetros quantificáveis e que realmente podem qualificar os planejadores. Já o número de passos não é um parâmetro tão óbvio, pois é de se imaginar que não importa se um robô vai de um lugar a outo

dando um ou dois passos. Porém, este parâmetro pode ser importante para determinadas situações ou até mesmo uma restrição. Por exemplo, um robô que se movimenta em trilhos pode ter seus passos limitados por restrições físicas em seu ambiente; ou um robô com rodas que seja muito pesado, cujo acionamento do motor quando em repouso gaste muito mais energia do que quando em movimento, se cada passo representar uma parada do motor, pode ser preferível, do ponto de vista do uso da energia, tomar uma rota mais longa, mas que dê menos passos. Outro exemplo é o robô de quatro rodas sem esterçamento, que faz curvas aplicando velocidades de rotação diferentes às rodas. Caso os passos da trajetória indiquem mudança de direção, pode ser que seja desejável dar menos passos, a fim de se evitar o desgaste das rodas.

O método de comparação consiste em: para cada planejador, selecionar a série de simulações que teve o melhor desempenho em cada um os parâmetros – tempo de processamento, quantidade de passos e comprimento das trajetórias – e executar as comparações quantitativas numéricas.

5.6.1
Comparação dos Tempos de Processamento

A Tabela 5-6 apresenta as séries selecionadas que tiveram o melhor desempenho no parâmetro tempo de processamento.

TABELA 0-6 – SIMULAÇÕES NO LABIRINTO COM OS MENORES TEMPOS DE PROCESSAMENTO

Planejador	*Série*	$\overline{tp}\ (s)$	*passos*	*d(u.c.)*
Dijkstra	1	**3,721**	8,333[1]	41,759[1]
Dyn. Prog.	1	**3,075**	8,500[1]	45,854[1]
PRM	1	**0,057**	10,80[1]	44,302[1]
RRT	3	**0,050**	22	39,267
Hybrid A*	2	**0,474**	64	63,546

[1] valores médios

A Figura 5.25, a seguir apresenta os dados de tempo de processamento listados na Tabela 5-6.

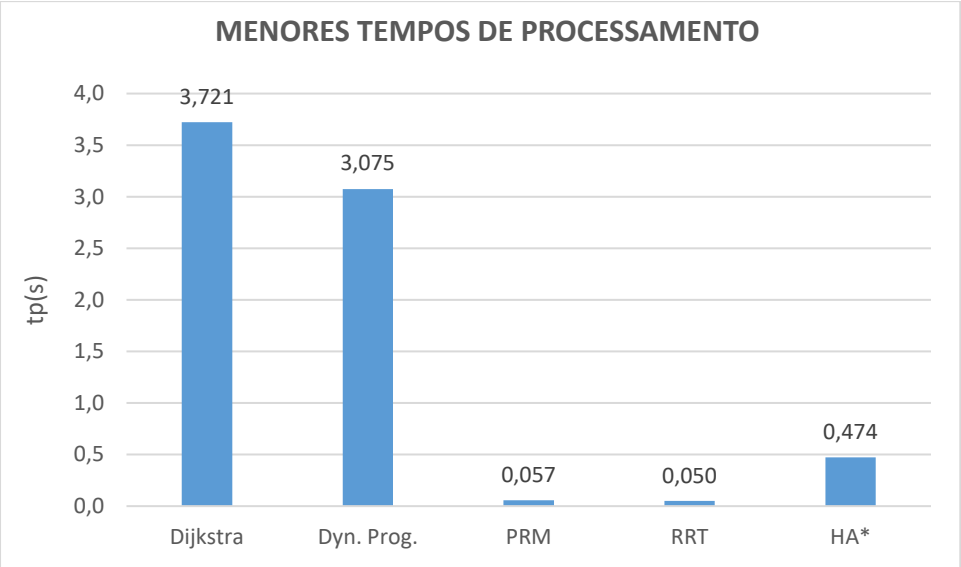


Figura 0.25 - Menores tempos de processamento médios nas simulações no Labirinto.

O planejador RRT foi o que gerou trajetórias com o menor tempo de processamento médio em sua terceira série de simulações. Dentro do conjunto de melhores tempos, foi também o que apresentou a menor trajetória.

Pelo gráfico da Figura 5.25, fica fácil enxergar a grande disparidade entre os tempos de processamento dos planejadores Algoritmo de Dijkstra e *Dynamic Programming* em relação aos demais. Ainda pesa contra estes o fato de seus rendimentos não terem sido 100% nas séries selecionadas: a primeira série do Algoritmo de Dijkstra gerou trajetórias em nove de dez tentativas e *Dynamic Programming*, em oito apenas. Em contrapartida, esses planejadores foram os que apresentaram as menores quantidades de passos. Então, em um contexto em que se busque uma quantidade pequenas de passos, dentro de um conjunto de menores tempos, esses planejadores podem ser uma opção. Mas deve-se levar em conta que existe a possibilidade de eles não conseguirem gerar uma trajetória logo na primeira tentativa. Por outro lado, há o planejador PRM, que apresenta um tempo de processamento 65,28 vezes menor que do Algoritmo de Dijkstra e uma média de apenas 2,5 passos a mais.

5.6.2
Comparação das Quantidades de Passos

A Tabela 5-7 a seguir apresenta as séries de cada planejador que geraram as menores quantidades de passos para as simulações no mapa Labirinto.

TABELA 0-7 – SIMULAÇÕES NO LABIRINTO COM AS MENORES QUANTIDADES DE PASSOS

Planejador	*Série*	*passos*	$\overline{tp}(s)$	*d(u.c.)*
Dijkstra	2	**8,200**[1]	3,958	41,470[1]
Dyn. Prog.	7	**7,700**[1]	9,098	43,957[1]
PRM	2	**10,20**[1]	0,060	44,783[1]
RRT	5	**13**	0,091	45,150
Hybrid A*	7	**27**	0,897	64,884

[1] valores médios

A Figura 5.26 apresenta os valores das quantidades de passos da Tabela 5-7 de forma gráfica, mostrando que o planejador *Dynamic Programming* foi o que obteve o melhor resultado neste quesito, em sua sétima série de simulações. Além da menor quantidade de passos, o *Dynamic Programming* também apresentou o segundo menor comprimento de trajetória. O ponto fraco continua sendo o tempo de processamento, que é 2,23 vezes maior que o do segundo colocado, o Algoritmo de Dijkstra, que também possui o menor comprimento de trajetória, nesta avaliação. Novamente o planejador PRM apresenta uma quantidade de passos próxima à do *Dynamic Programming*, com apenas 2,5 passos a mais, mas com um tempo de processamento muito menor.

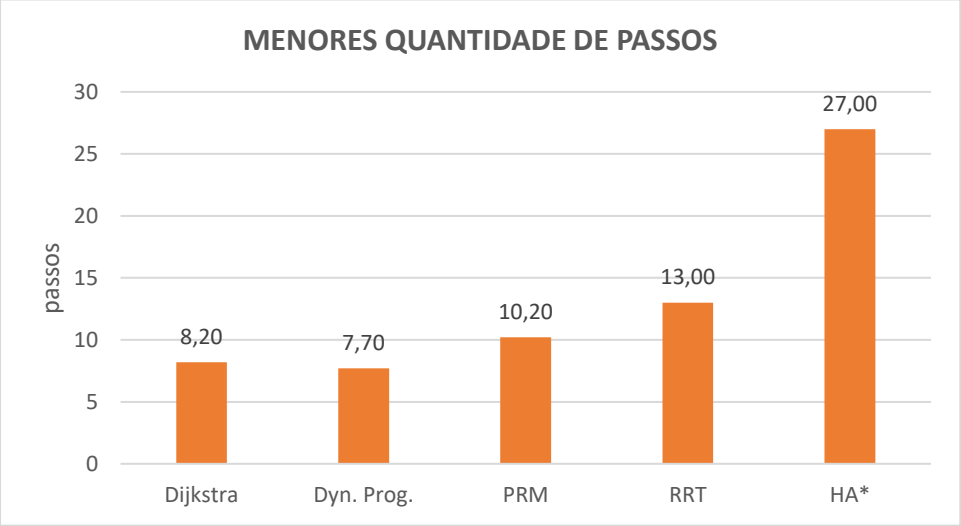


Figura 0.26 - Menores quantidades de passos das simulações no Labirinto.

A maior quantidade passos obtida pelo *Hybrid A** se explica pela trajetória curvilínea traçada pelo planejador, pois, para executar cada curva são necessários alguns passos, como pode ser verificado na matriz *repath.sates* gerada pelo MATLAB, conforme Figura 4.11.

5.6.3
Comparação dos Comprimentos das Trajetórias

A Tabela 5-8 abaixo apresenta as séries de cada planejador cujos comprimentos de trajetórias foram os menores, nas simulações no mapa Labirinto.

TABELA 0-8 – SIMULAÇÕES NO LABIRINTO COM OS MENORES COMPRIMENTOS DE TRAJETÓRIA

Planejador	*Série*	*d(u.c.)*	$\overline{tp}(s)$	*passos*
Dijkstra	6	**39,384**[1]	8,940	9,600[1]
Dyn. Prog.	6	**43,206**[1]	8,459	8,600[1]
PRM	1	**44,302**[1]	0,057	10,80[1]
RRT	3	**39,267**	0,050	22
Hybrid A*	3	**43,674**	0,853	44

[1] valores médios

A Figura 5.27, a seguir, apresenta o gráfico dos comprimentos de trajetória *d(u.c.)*, da Tabela 5-8.

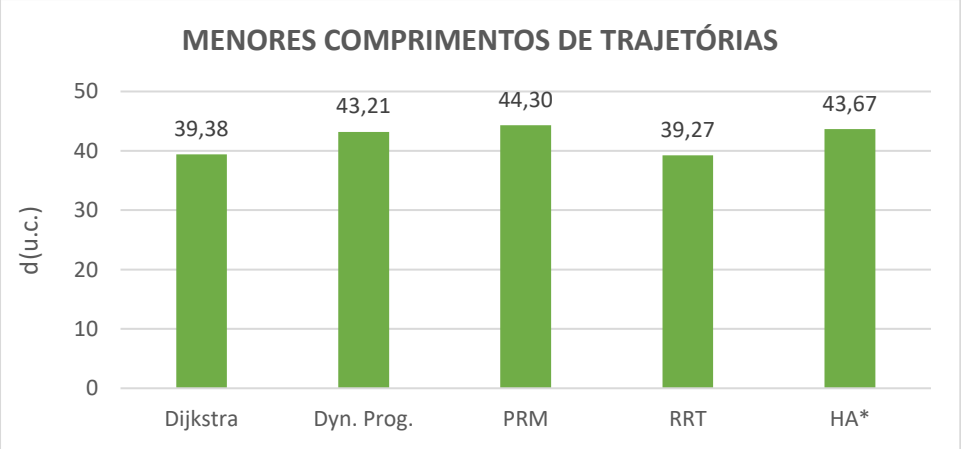


Figura 0.27 - Menores comprimentos de trajetórias das simulações no Labirinto.

O planejador RRT possui o menor comprimento de trajetória dentre todas as séries de simulações. Uma vez que este planejador sempre executa a mesma trajetória para um mesmo delta, o valor de 39,384 unidades de comprimento é também o menor dentre todas as suas simulações individualmente. O mesmo vale para o comprimento de 43,674 unidades, do planejador *Hybrid A**. Ainda sobre o RRT, a terceira série de simulações é também a que apresenta o menor tempo de processamento médio, conforme visto anteriormente. Apenas o número de passos

desse planejador, dentro deste conjunto, aparece como o “penúltimo colocado”, mas ainda assim, com a metade dos passos do último.

Os menores comprimentos de trajetórias são mais próximos entre si do que os demais parâmetros, sobretudo o tempo de processamento. A maior diferença, por exemplo, que é para o comprimento gerado pelo PRM, é de apenas 5,035 unidades de comprimento. Isso ocorre porque cada planejador foi capaz de encontrar o caminho mais adequado para si. Observa-se que o caminho mais curto do *Hybrid A** foi diferente dos demais planejadores, devido à sua trajetória mais contínua.

6
Simulações no Mapa 2 – Labirinto Estreito

6.1
Simulações com Algoritmo de Dijkstra no Labirinto Estreito

O procedimento de testes descrito no capítulo anterior foi aplicado às simulações com Algoritmo de Dijkstra no mapa Labirinto Estreito.

O algoritmo encontrou mais dificuldades para gerar trajetórias neste mapa, sendo que a primeira foi gerada com100 nós. Mesmo assim, em dez tentativas, apenas uma simulação foi bem-sucedida. Somente a partir de 200 nós é que se obteve uma quantidade significativa de 8 resultados.

Viu-se por bem então adotar uma nova estratégia: incrementar em 50 o número de nós e, caso houvesse uma sequência de 100% de aproveitamento em duas sequências seguidas, dar "um passo para trás" e executar mais uma série de dez simulações, com um número de nós intermediário entre as duas sequências. A Tabela 6-1 mostra a aplicação desta estratégia, assim como os valores médios de tempo de processamento, quantidade de passos e comprimento das trajetórias. Faz-se necessário relatar que não foi possível avançar com as simulações com valores acima de 300 nós por limitações do *hardware*.

TABELA 0-1 – SIMULAÇÕES COM ALGORITMO DE DIJKSTRA NO LABIRINTO ESTREITO

Série	*nós*	$\overline{tp}$(s)	$\overline{passos}$	$\bar{d}$(u. c.)	*Obs.*
1	200	20,352	13,429	46,517	8 resultados
2	250	33,475	13,600	46,279	
3	270	37,594	13,556	45,560	9 resultados
4	300	48,578	14,200	44,116	

Legenda:

nós = Número de nós que ocupam a área livre do mapa.

$\overline{tp}$ = Tempo de processamento médio, medido em segundos.

$\overline{passos}$ = Número médio de nós utilizados para conectar os pontos inicial e objetivo.

$\bar{d}$ = Comprimento médio das trajetórias, medido em unidades de comprimento (u.c.).

A Figura 6.1 apresenta os gráficos obtidos a partir dos valores da Tabela 6-1. O gráfico de *tp*(*s*) indica que os tempos médios de processamento crescem conforme aumenta-se os números de nós das simulações, da mesma forma que ocorreu nas simulações no mapa Labirinto. Este crescimento não é exatamente

proporcional pois cada simulação individual tem um tempo próprio de processamento. As quantidades médias de passos, apresentadas na Figura 6.1 também não demonstram ter relação com o número de nós. Já os tamanhos das trajetórias, diminuem conforme o número de nós aumenta, mas de forma muito pequena. A diferença entre o maior e menor valor dessas médias de tamanho é de 2,401 unidades de comprimento.

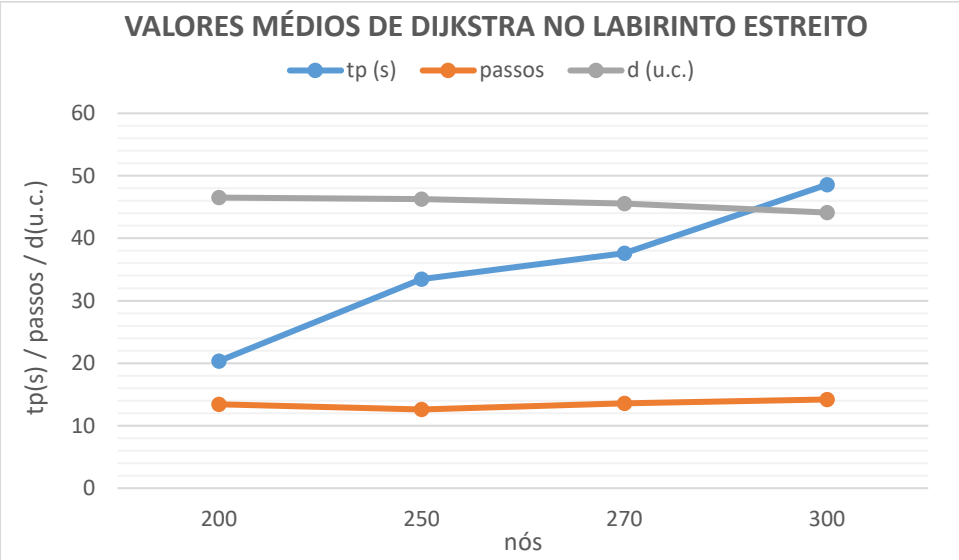


Figura 0.1 - Valores médios de tempo de processamento, quantidade de passos e comprimento das trajetórias das simulações do Algoritmo de Dijkstra no Labirinto Estreito.

As Figuras 6.2 a 6.4 apresentam os valores de tempo de processamento, quantidade de passos e comprimento da trajetória de cada simulação individualmente. A Figura 6.2 evidencia que, assim como para o mapa anterior e já apontado pelo gráfico da Figura 6.1, os tempos de processamento possuem a tendência de acompanhar o número de nós da simulação. Há apenas um pico na oitava simulação com 250 nós, destoante dessa tendência. Pelos gráficos das Figuras 6.3 e 6.4, tal pico não acarretou em uma quantidade maior de passos ou mesmo comprimento de trajetória na mesma simulação. Ainda sobre os gráficos dessas figuras, os mesmos não apresentam evidências de relações entre o número de nós e as quantidades de passos ou comprimentos de trajetórias. A única relação aparente entre os passos e o tamanho da trajetória ocorre na oitava simulação da série de 270 nós, na qual há picos tanto no gráfico de passos, da Figura 6.3, quanto de tamanho de trajetória, da Figura 6.4.

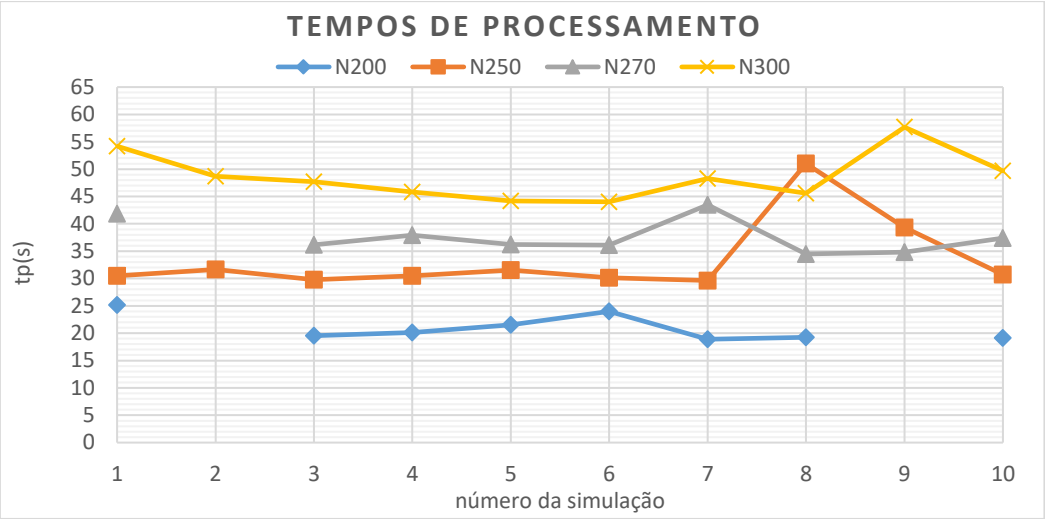


Figura 0.2 - Tempo de processamento de cada simulação de Dijkstra no Labirinto Estreito.

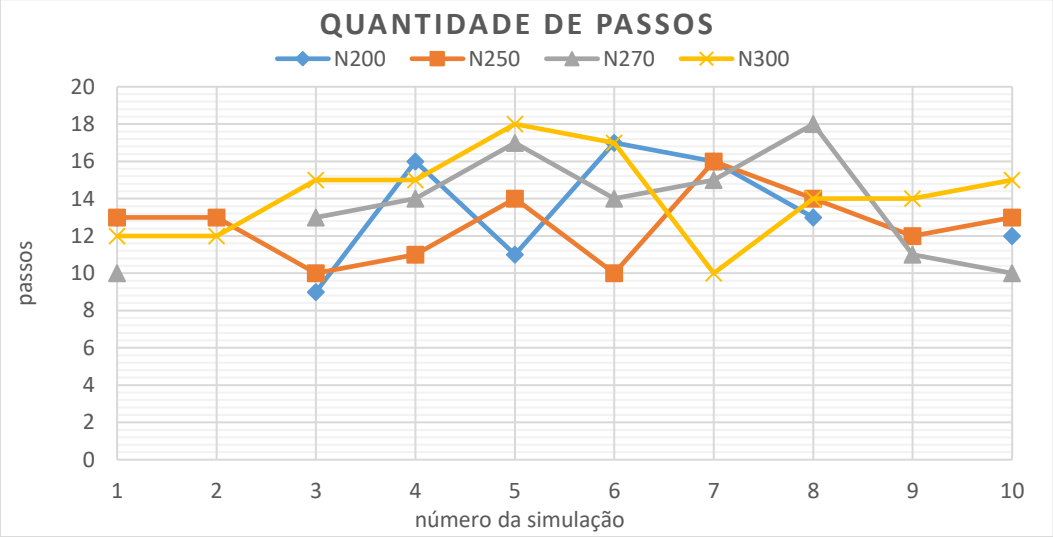


Figura 0.3 - Quantidade de passos de cada simulação de Dijkstra no Labirinto Estreito.

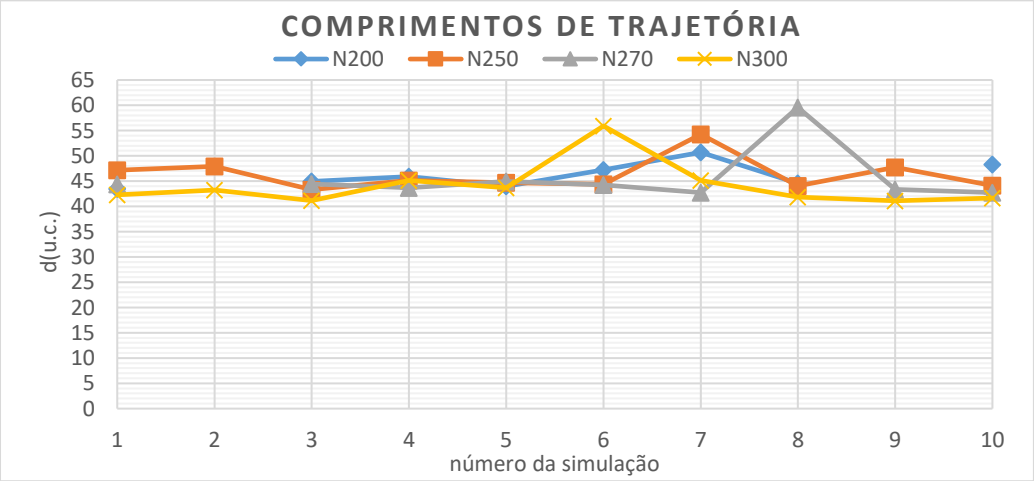
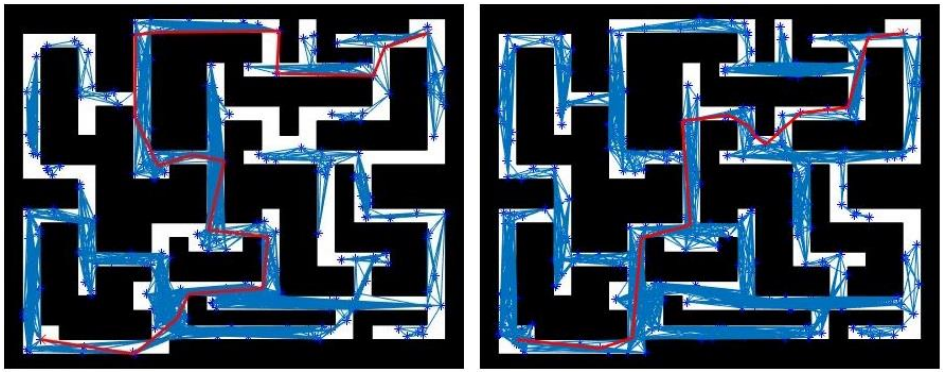


Figura 0.4 - Comprimento de trajetória cada simulação de Dijkstra no Labirinto Estreito.

A Figura 6.5 apresenta a maior e a menor trajetórias geradas com o Algoritmo de Dijkstra no Labirinto Estreito, que foram na oitava simulação com 270 nós e na nona simulação com 300 nós, respectivamente. Para a maior trajetória, nota-se como a falta de conexão do nó de coordenadas (8, 9) com o de coordenadas (11, 10) e a falta de um nó intermediário entre os nós (13, 18) e (11, 15), prejudicou a formação de uma trajetória mais direta, forçando o planejador a adotar desvios, acarretando em um comprimento maior.



(a) (b)

Figura 0.5 - Simulações do Algoritmo de Dijkstra no Labirinto Estreito: (a) oitava, com 270 nós e (b) nona, com 300 nós.

6.2
Simulações com Dynamic Programming no Labirinto Estreito

O planejador *Dynamic Programming* também encontrou dificuldades para gerar trajetórias no mapa Labirinto Estreito, sendo que a primeira trajetória foi gerada com 90 nós. Então adotou-se a mesma estratégia de incrementar o número de nós em 50 a cada série de simulações, pois incrementos de 10 nós não provocaram melhoras na geração de resultados. Apenas a série de simulações com 300 nós gerou resultados nas dez simulações e, por isso, a fim de haver dados em quantidade razoável, foi feita uma concessão às séries de simulações com 150, 200 e 250 nós, para que os valores computados durante as simulações fossem considerados nas comparações, pois não foi possível executar mais simulações com um número maior de nós. A Tabela 6-2 apresenta os tempos de processamento, quantidade de passos e número de trajetórias das séries de simulações do *Dynamic Programming* no mapa Labirinto Estreito.

TABELA 0-2 – SIMULAÇÕES COM DYNAMIC PROGRAMMING NO LABIRINTO ESTREITO

Série	*nós*	$\overline{tp}$(s)	$\overline{passos}$	$\bar{d}$(u. c.)	Obs.
1	150	13,581	11,571	51,412	7 resultados
2	200	27,145	11,667	51,152	6 resultados
3	250	32,755	11,167	48,251	6 resultados
4	300	48,342	12,100	49,306	

Legenda:

nós = Número de nós que ocupam a área livre do mapa.

$\overline{tp}$ = Tempo de processamento médio, medido em segundos.

$\overline{passos}$ = Número médio de nós utilizados para conectar os pontos inicial e objetivo.

$\bar{d}$ = Comprimento médio das trajetórias, medido em unidades de comprimento (u.c.).

A Figura 6.6 apresenta os valores da Tabela 6-2 de forma gráfica. Nota-se que o tempo de processamento, *tp* (*s*), aumenta com o incremento do número de nós, o que é natural para este planejador, pois além do tempo necessário para se gerar uma maior quantidade de nós, quanto maior este número, maior será a quantidade de conexões efetuadas entre eles antes de se traçar a trajetória propriamente. Já a quantidade de passos e o comprimento da trajetória apresentam amplitudes muito pequenas entre seus valores máximos e mínimos.

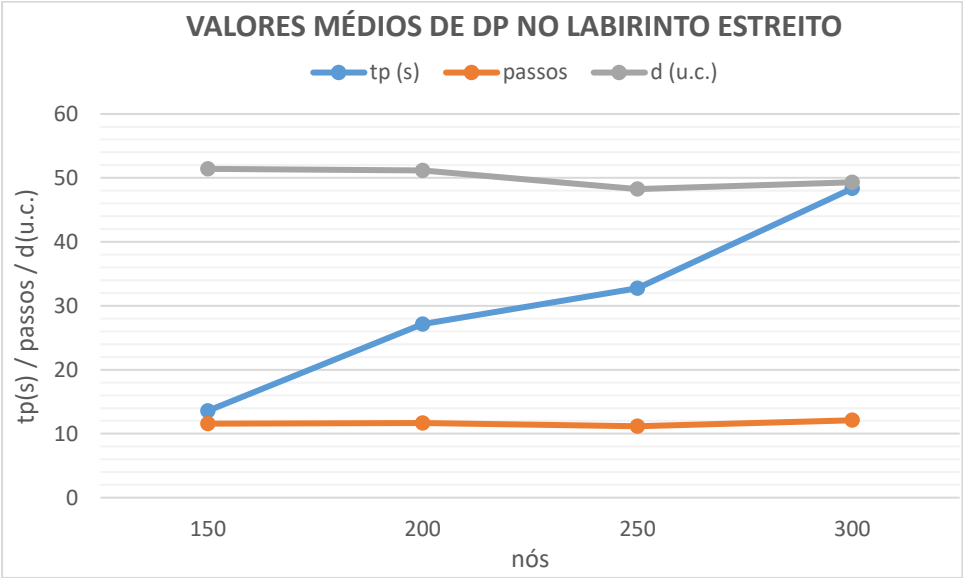


Figura 0.6 - Valores médios de tempos de processamento, quantidade de passos e comprimento de trajetória das simulações do *Dynamic Programming* no Labirinto Estreito.

Os resultados numéricos das simulações com o *Dynamic Programming* refletem a dificuldade maior que o planejador teve em gerar trajetórias para o mapa Labirinto Estreito. Enquanto que para o mapa Labirinto, o maior número de nós foi

150, para o labirinto estreito, esse foi o número mínimo de nós para se obter uma quantidade razoável de resultados. Mesmo para esse número de nós, o tempo de processamento, a quantidade de passos e o comprimento da trajetória nas simulações no mapa Labirinto Estreito são superiores aos valores computados nas simulações no mapa Labirinto.

A seguir, são apresentados os gráficos dos tempos de processamento, quantidades de passos e comprimentos das trajetórias de cada simulação individualmente, nas Figuras 6.7 a 6.9, respectivamente. A Figura 6.7 confirma a tendência do tempo de processamento aumentar com o número de nós. As simulações aparecem bem separadas por séries, havendo apenas interseções entre simulações da segunda e terceira série, de 200 e 250 nós, sendo que essas são as que apresentam os tempos de processamento médios mais próximos. As Figuras 6.8 e 6.9 também confirmam a proximidade entre os valores das quantidades de passos e comprimento das trajetórias, mostrando que, nos dois casos, os valores computados nas simulações individuais oscilam todos dentro de uma mesma faixa, independentemente dos números de nós. Observa-se uma certa semelhança entre as curvas das quantidades de passos e comprimentos das trajetórias, sobretudo na série de 300 nós. Esta relação entre os dois parâmetros é vista com detalhes a seguir.

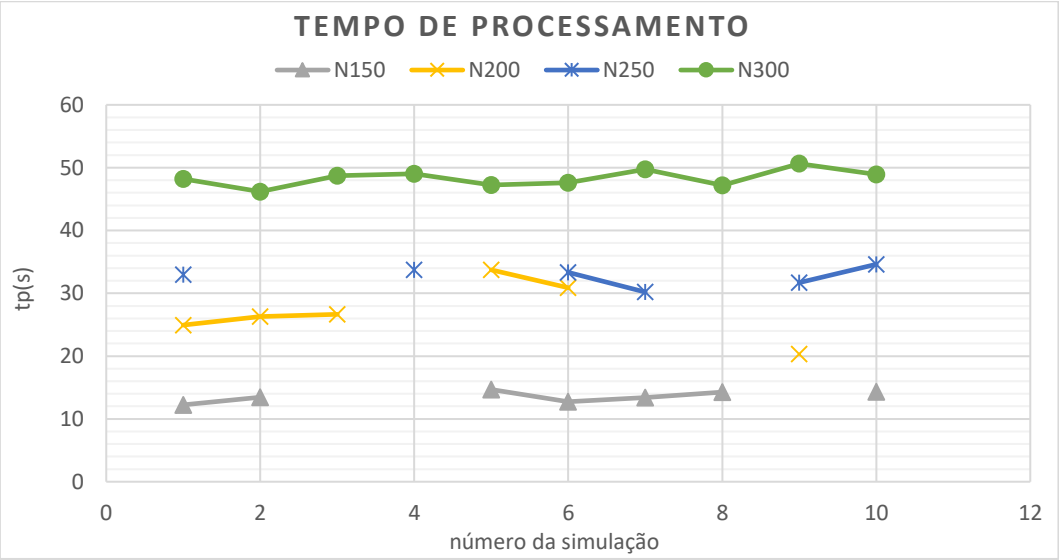


Figura 0.7 - Tempos de processamento de cada simulação do *Dynamic Programming* no Labirinto Estreito.

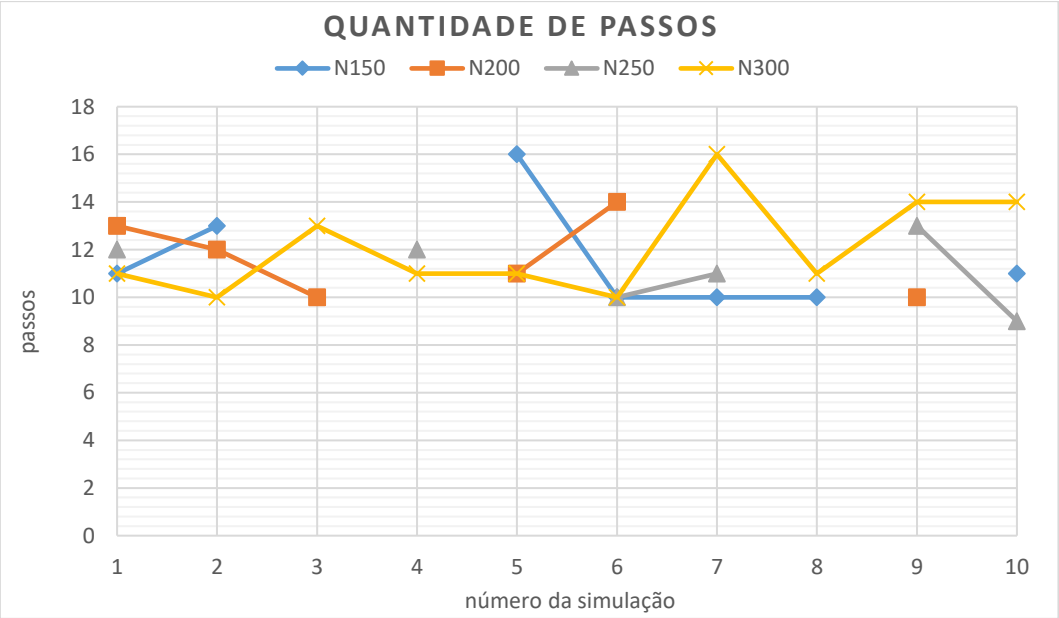


Figura 0.8 - Quantidade de passos de cada simulação do *Dynamic Programming* no Labirinto Estreito.

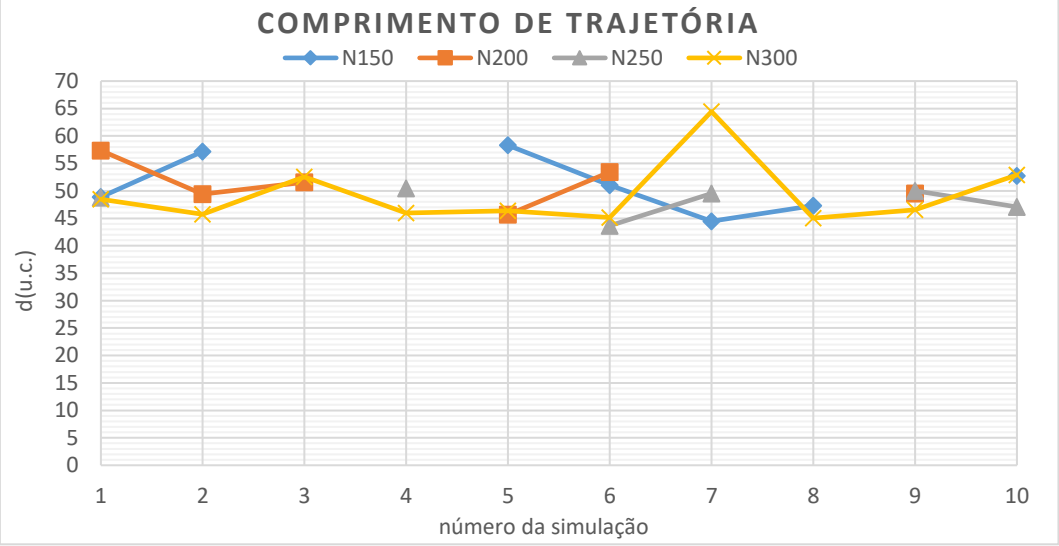


Figura 0.9 - Comprimento de trajetória de cada simulação do *Dynamic Programming* no Labirinto Estreito.

A Figura 6.10 apresenta o gráfico dos comprimentos de trajetória versus a quantidade de passos, revelando que há uma tendência dos comprimentos das trajetórias acompanhar a quantidade de passos. Este comportamento se explica pelo desafio de traçar as trajetórias em um mapa com caminhos tão estreitos. A rota até o objetivo necessita fazer muitos desvios visto a dificuldade para se estabelecer conexões entre dois nós. Logo, uma trajetória com desvios acaba acarretando uma quantidade de passos maior, assim como acaba por ser tornar mais longa.

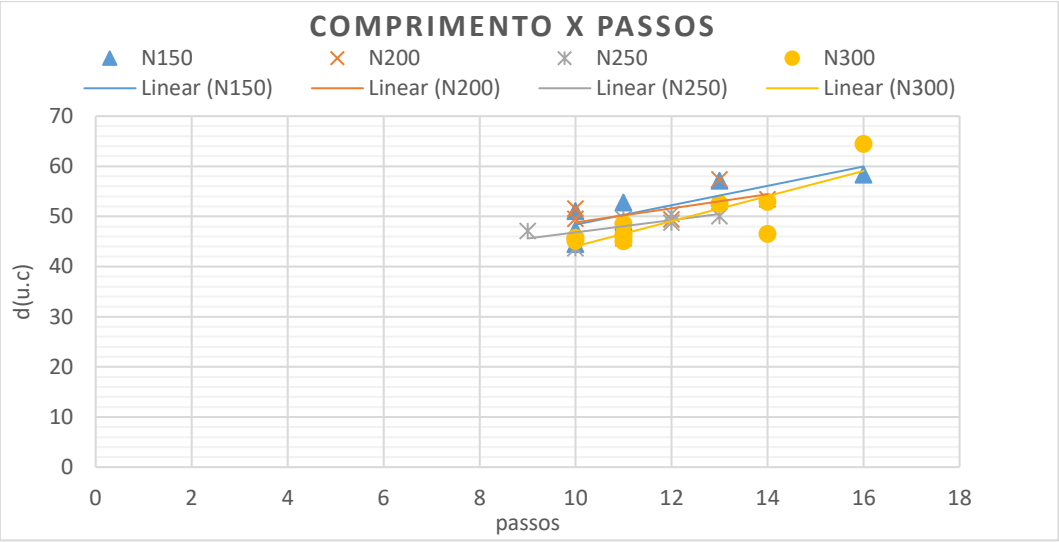
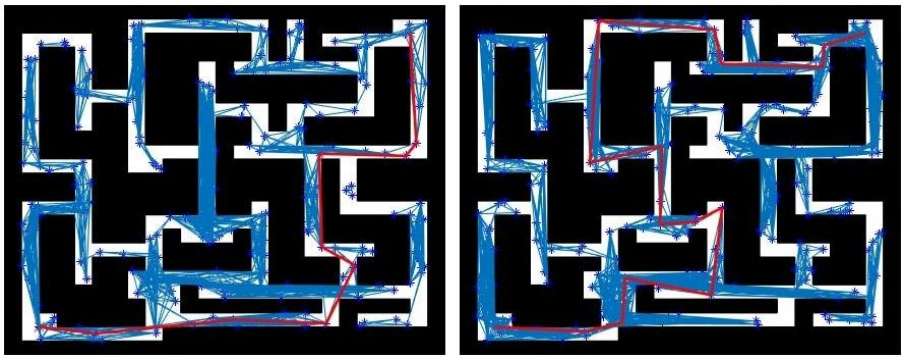


Figura 0.10 - Comprimento de trajetória versus quantidade de passos do *Dynamic Programming* no Labirinto Estreito.

A seguir, na Figura 6.11 são mostradas a maior e a menor trajetória gerada pelo *Dynamic Programming* no mapa Labirinto Estreito, que são respectivamente a sexta simulação com 250 nós e a sétima simulação com 300 nós.



(a) (b)

Figura 0.11 - Simulações do *Dynamic Programming* no Labirinto Estreito: (a) sexta, com 250 nós e (b) sétima, com 300 nós.

6.3
Simulações com PRM no Labirinto Estreito

As simulações no mapa Labirinto Estreito seguem o mesmo esquema que no mapa Labirinto. São quatro séries de dez simulações, sendo que a cada série o delta é incrementando em 5.

A Tabela 6-3 apresenta os números médios de nós, os tempos de processamento médios, a quantidade média de passos e o comprimento médio das trajetórias resultantes das simulações com o PRM no mapa Labirinto Estreito.

TABELA 0-3 – SIMULAÇÕES COM PRM NO LABIRINTO ESTREITO

Série	Δ	$\overline{nós}$	$\overline{tp}(s)$	$\overline{passos}$	$\bar{d}(u.c.)$
1	10	137	0,078	15,00	50,925
2	15	141	0,072	16,30	50,939
3	20	123	0,097	14,50	50,779
4	25	137	0,163	16,00	50,556

Legenda:

Δ = delta.

$\overline{nós}$ = Número de nós médios gerados durante as simulações.

$\overline{tp}$ = Tempo de processamento médio, medido em segundos.

$\overline{passos}$ = Número médio de nós utilizados para conectar os pontos inicial e objetivo.

$\bar{d}$ = Comprimento médio das trajetórias, medidos em unidades de comprimento (u.c.).

A Figura 6.12, a seguir, apresenta os gráficos dos parâmetros da Tabela 6-3. Pela curva dos tempos de processamento médios, *tp(s)*, parece que, a princípio, não há relação entre este parâmetro com o delta, visto que há uma redução dos tempos da série com delta 10 para a série com delta 15. Mas, a partir desta série, os tempos passam aumentar, acompanhando o delta. Já os comprimentos das trajetórias apresentam valores muito próximo, demonstrando que, independentemente do delta, o planejador foi capaz de encontrar trajetórias equivalentes. No mais, os valores médios de comprimento de trajetória e quantidades de passos apresentam variação muito baixa em suas respectivas curvas.

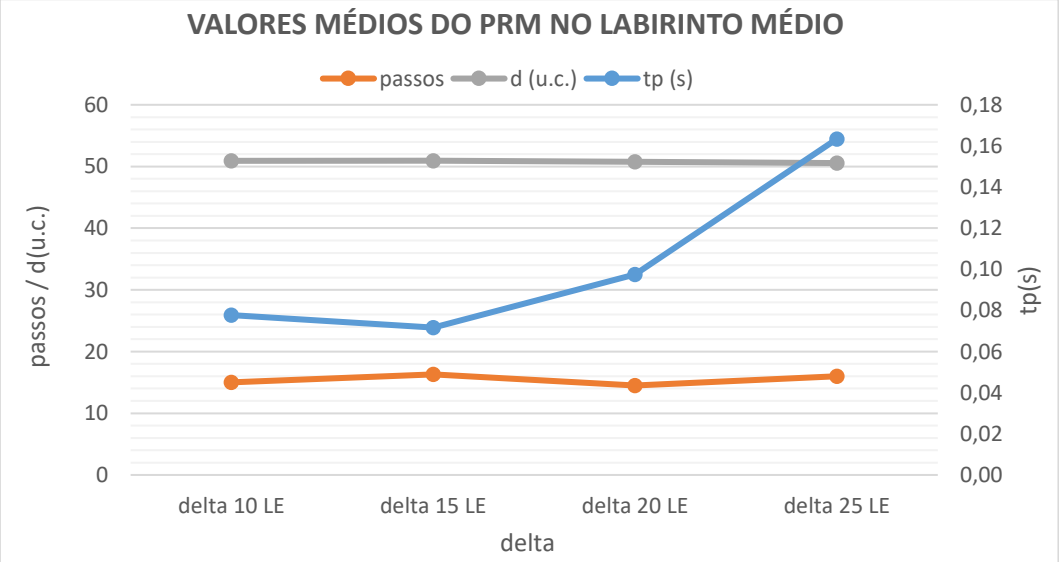


Figura 0.12 - Valores médios de tempo de processamento, quantidade de passos e comprimento das trajetórias das simulações do PRM no Labirinto Estreito.

A seguir são mostrados os valores de cada simulação individual dos tempos de processamento, quantidades de passos e comprimentos das trajetórias, nas Figuras 6.13 a 6.15 respectivamente. A Figura 6.13 demonstra que os tempos de processamento individuais seguem o mesmo padrão das médias. O pico da terceira simulação da série de delta 25, foi responsável pela média da série apresentar um salto de 0,66 segundos em relação à série com delta 20. Esta última havia apresentado uma média de tempo de apenas 0,025s acima da anterior. A Figura 6.14 demonstra o equilíbrio entre as quantidades de passos de cada série de simulações e que estas independem do valor do delta. Os dois picos na sexta e sétima simulações da quarta e segunda séries respectivamente (delta 25 e delta 15) foram os fatores determinantes para que as mesmas apresentassem as maiores médias. Os comprimentos de cada trajetória individual, da Figura 6.15, se encontram quase todos dentro de uma mesma faixa de valores, com a única exceção do pico da terceira simulação de delta 10, corroborando com a curva dos valores médios.

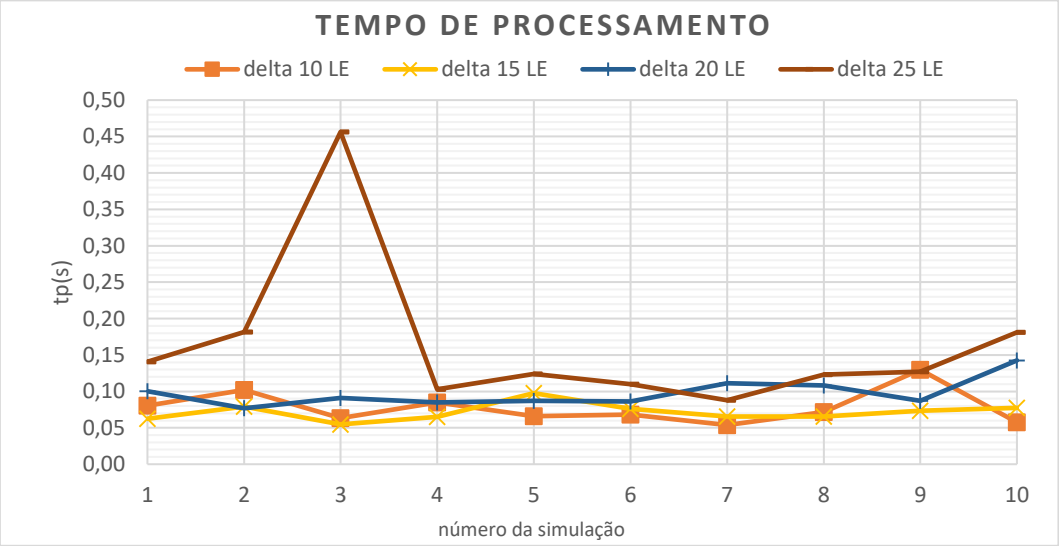


Figura 0.13 - Tempo de processamento de cada simulação do PRM no Labirinto Estreito.

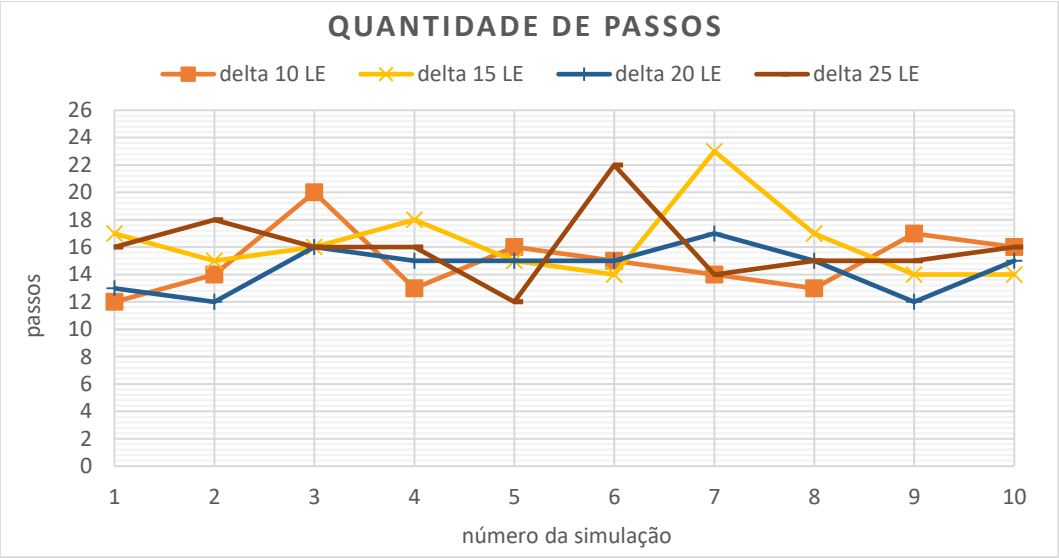


Figura 0.14 - Quantidade de passos de cada simulação do PRM no mapa Labirinto Estreito.

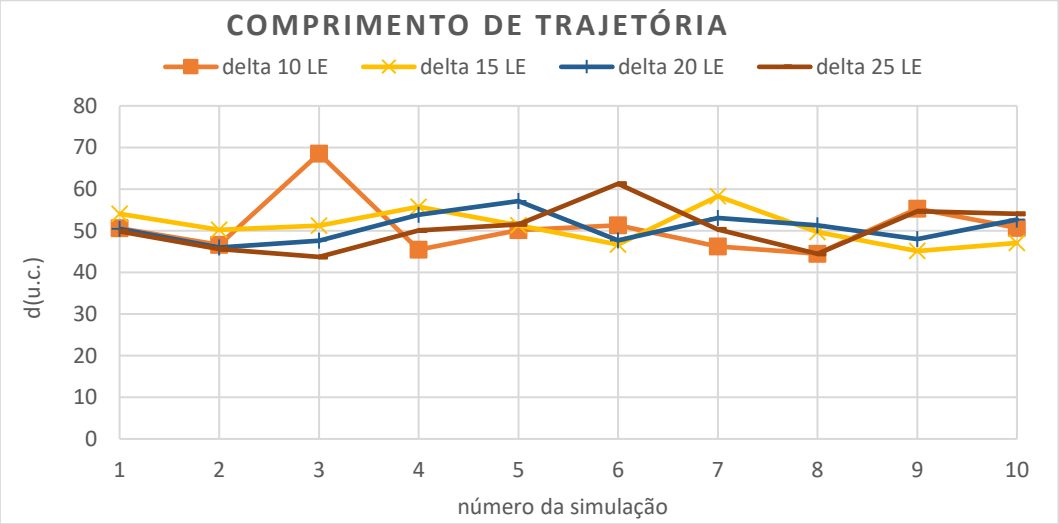


Figura 0.15 - Comprimento das trajetórias de cada simulação do PRM no Labirinto Estreito.

Dada a tendência natural, para este planejador, do comprimento de trajetórias acompanhar a quantidade de passos, estudou-se novamente esta relação por meio do gráfico da Figura 6.16, que confirma tal tendência, sobretudo nas duas primeiras séries de simulações, com delta 10 e delta 15, mesmo que um parâmetro não seja a causa do crescimento do outro, mas encontrem suas variações em uma mesma raiz, que é o loop de acréscimo de nós.

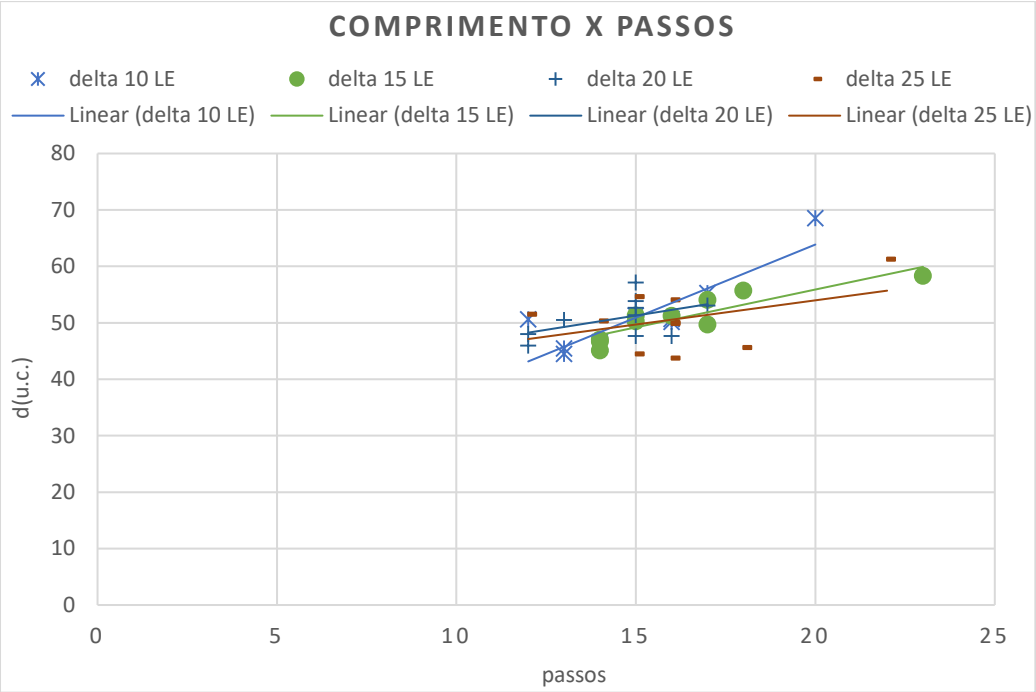
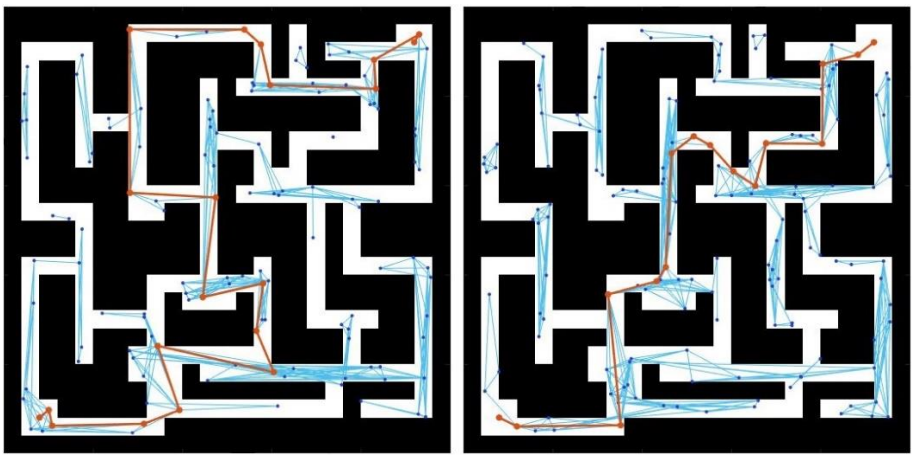


Figura 0.16 - Comprimento de trajetória versus quantidade de passos das simulações do PRM no Labirinto Estreito.

A seguir, a Figura 6.17 apresenta a maior e a menor trajetórias geradas nas simulações com PRM. São elas: a terceira simulação da primeira série, com delta 10 e a terceira simulação da quarta série, com delta 25.



(a) (b)

Figura 0.17 - Simulações com PRM no Labirinto Estreito: (a) terceira, com delta 10 e (b) terceira, com delta 25.

6.4
Simulações com RRT no Mapa Labirinto Estreito

A mesma estratégia de simulação adotada para o mapa Labirinto foi repetida para o mapa Labirinto Estreito, com os mesmos valores de delta, inclusive. Isso porque o planejador mostrou, para este mapa, capacidade de gerar trajetórias consistentemente a partir de um delta igual a 0,5 e sem alterar a capacidade máxima de nós e iterações.

A Tabela 6-4 apresenta o número de nós, as iterações, os tempos de processamento médios, a quantidade de passos e os comprimentos das trajetórias geradas em cada série de simulações. Lembrando que, desses parâmetros, apenas o tempo é médio, porque, para um mesmo delta, o planejador sempre repete a mesma trajetória, com as mesmas características, com exceção do tempo de processamento.

TABELA 0-4 – SIMULAÇÕES COM RRT NO LABIRINTO ESTREITO

Série.	*max nós*	*max iter*	Δ	*nós*	*iter*	$\overline{tp}(s)$	*passos*	*d(u.c.)*
1	1×10^4	1×10^4	0,5	1214	4152	2,403	99	47,993
2	1×10^4	1×10^4	1	198	1565	0,571	50	47,763
3	1×10^4	1×10^4	2	159	743	0,288	26	47,044
4	1×10^4	1×10^4	3	418	2557	1,000	22	51,190
5	1×10^4	1×10^4	5	408	2557	0,861	22	53,874
6	1×10^4	1×10^4	7	998	4432	2,039	27	47,152
7	1×10^4	1×10^4	10	938	4432	2,022	27	48,667
8	1×10^4	1×10^4	15	938	4432	1,751	27	48,667
9	1×10^4	1×10^4	20	938	4432	1,904	27	48,667
10	1×10^4	1×10^4	25	938	4432	2,028	27	48,667

Legenda:

max nós = número máximo de nós.

max iter = número máximo de iterações.

Δ = delta.

nós = número de nós gerados durante as simulações.

iter = iterações geradas durante as simulações.

$\overline{tp}$ = tempo de processamento médio, medido em segundos.

passos = número médio de nós utilizados para conectar os pontos inicial e objetivo.

d = Comprimento da trajetória, medido em unidades de comprimento (u.c.).

A Figura 6.18 apresenta os gráficos dos tempos médios de processamento, quantidades de passos e comprimentos das trajetórias das séries da Tabela 6-4, cuja

análise expõe alguns dados interessantes: a quantidade de passos da sexta à décima séries (delta 7 a delta 25) é a mesma. Da mesma forma, os comprimentos das trajetórias da sétima à décima série (delta 10 a delta 25) também são iguais. Mas, uma informação importante que não está explícita nos gráficos é que as trajetórias da sétima, oitava, nona e décima séries, são exatamente iguais. Ou seja, a partir de delta igual a 10, já não faz mais diferença se aumentar o valor deste parâmetro, pois o planejador já não encontra mais espaço dentro do mapa para gerar trajetórias diferentes.

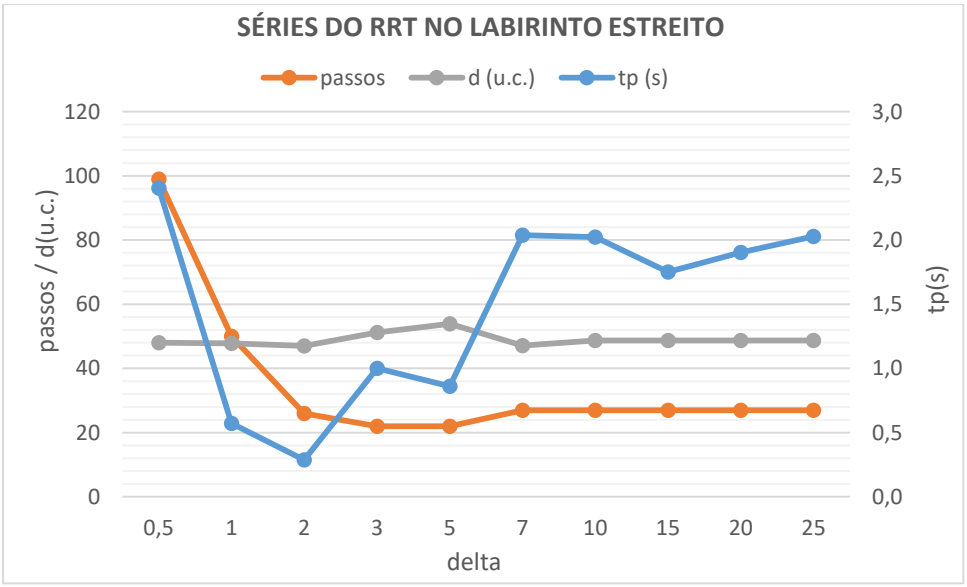


Figura 0.18 - Tempos de processamento médio, quantidade de passos e comprimento das trajetórias das séries do RRT no Labirinto Estreito.

Os tempos de cada simulação individual são apresentados na Figura 6.19. As séries, em relação ao tempo de processamento, se dividem em dois grupos, cada um ocupando uma faixa: uma que vai de 0,265 a 1,166 segundos, formado pela segunda a quinta séries de simulações (deltas 1 a 5) e outra que vai de 1,141 a 2,646 segundos, englobando as demais séries. Destaca-se o gráfico da série de delta 0,5, pois, além de estar no grupo que reúne as séries com os maiores deltas, apesar de apresentar o menor, é também a séries que possui: o maior tempo de processamento, o maior número de nós e a maior quantidade de passos.

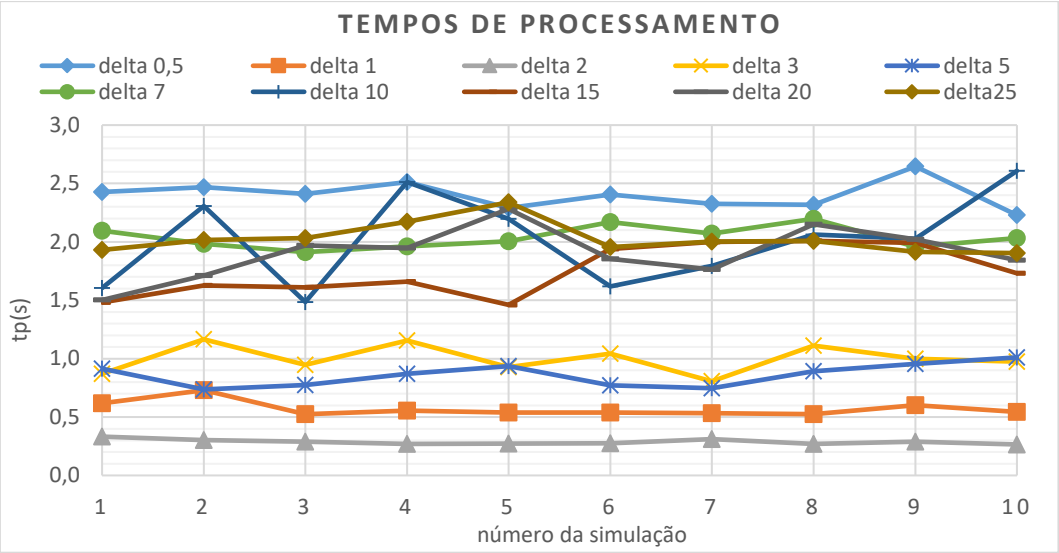
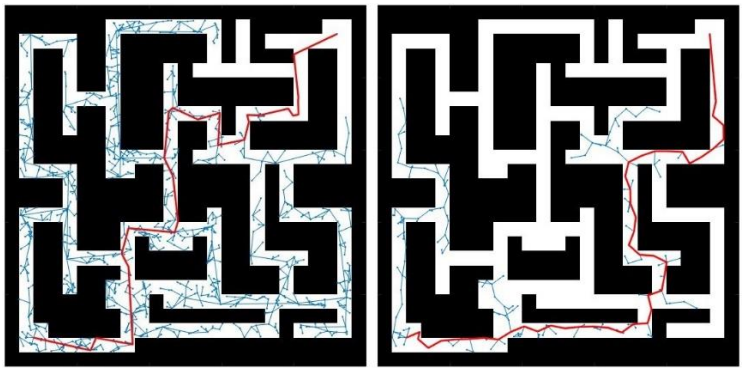


Figura 0.19 - Tempos de processamento de cada simulação com o RRT no Labirinto Estreito.

A Figura 6.20 apresenta a trajetórias gerada a partir da sétima série, com delta 10, que são iguais e a trajetória gerada na primeira simulação, com delta 0,5.



(a) (b)

Figura 0.20 - Simulações do RRT no Labirinto Estreito: (a) com delta 10 e (b) com delta 0,5.

6.5
Simulações com Hybrid A* no Labirinto Estreito

As simulações com *Hybrid A** no mapa Labirinto Estreito necessitaram de uma estratégia diferente da utilizada no mapa anterior, pois, dada a restrição de espaço, o planejador não foi capaz de gerar trajetórias com os mesmos comprimentos do primitivo e raios de curvaturas. A pesquisa então focou-se mais em descobrir quais os limites dos tamanhos de primitivos seriam suportados pelo mapa e, consequentemente, os raios de curvaturas. O tamanho mínimo do comprimento do primitivo suportado para este mapa foi de 1,41422 unidades de

comprimento e o valor máximo foi de 1,8. Acima deste, não houve geração de trajetórias

A Tabela 6-5 a seguir lista os resultados dos parâmetros avaliados. Os valores com a marcação amarela, indicam quais parâmetros foram modificados em relação à primeira série. Lembrando que, assim como o RRT, as trajetórias geradas nas dez simulações de uma série são iguais. O único parâmetro diferente é o tempo de processamento e por isso, este é o único que aparece como média.

TABELA 0-5 – SIMULAÇÕES COM HYBRID A* NO LABIRINTO ESTREITO

Sr	*Lprim*	*Rmin*	*Nprim*	*FC*	*RC*	*DC*	*ID*	*AEI*	$\overline{tp}(s)$	*passos*	*d(u.c.)*
1	1,414	1	5	1	3	0	1	5	0,644	51	47,679
2	1,414	1	7	1	3	0	1	5	0,772	49	47,042
3	1,414	1	5	1	3	0	2	5	0,602	26	47,952
4	1,414	1	9	1	3	0	1	5	1,192	47	44,825
5	1,414	1	5	1	3	0	1	4	0,413	50	46,869
6	1,500	1	5	1	3	0	1	5	1,197	72	69,554
7	1,500	1	7	1	3	0	1	5	1,213	48	46,318
8	1,600	2	5	1	3	0	1	5	0,937	60	55,895
9	1,700	2	5	1	3	0	1	5	0,790	55	51,695
10	1,800	2	5	1	3	0	1	5	0,769	59	57,026

Legenda:

Lprim = comprimento dos primitivos.

Rprim = raio mínimo de curvatura.

Nprim = número máximo de primitivos por nó.

FC = custo de avanço (do inglês *forward cost*).

RC = custo de retorno (do inglês *reverse cost*).

DC = custo de mudança de direção (do inglês *direction switching cost*).

ID = distância de interpolação (do inglês *interpolation distance*).

AEI = intervalo de expansão analítica (do inglês *analytic expansion interval*).

$\overline{tp}$ = tempo de processamento médio, medido em segundos.

passos = número médio de nós utilizados para conectar os pontos inicial e objetivo.

d = Comprimento da trajetória, medido em unidades de comprimento (u.c.).

Os valores de tempo de processamento, quantidade de passos e comprimento de trajetórias da Tabela 6-5 são apresentados de forma gráfica na Figura 6.21, destacando-se o tempo de processamento da quinta série de simulações como o menor. Essa simulação teve como característica, o intervalo de expansão analítica reduzido de cinco para quatro. Na prática, este parâmetro significa quantas vezes o planejador vai gerar novos nós – com primitivos – para tentar alcançar o

objetivo. Por exemplo, se o parâmetro estiver ajustado para cinco, que é seu *default*, ele vai gerar cinco novos "ramos" e tentar alcançar o objetivo através de uma expansão analítica usando o modelo de Reeds-Shepp. Se a tentativa falhar o ciclo se reinicia. Modificando o parâmetro para quatro, o planejador gera quatro novos nós a cada tentativa de se alcançar o objetivo.

Todavia, o tempo de processamento menor não se traduziu em menos passos ou menor comprimento de trajetória. Sobre os gráficos desses parâmetros, é notável a semelhança entre eles, com exceção da terceira série, que tem como característica a distância de interpolação (ID) igual a 2. Ou seja, o comprimento do passo das simulações dessa série é igual a 2 unidades de comprimento, enquanto que as das demais é igual a 1.

Na sétima simulação, o comprimento do primitivo é mantido em 1,5 e o número de primitivos por nós é aumentado para sete, reduzindo tanto a quantidade de passos, quanto o comprimento da trajetória, em relação à série anterior. É interessante notar como que um incremento pequeno no tamanho do primitivo, de 1,141 na primeira série, para 1,5 na sexta, mantendo todos os demais parâmetros iguais, gerou um acréscimo em todos os parâmetros de saída (em relação à primeira série): tempo de processamento, quantidade de passos e comprimento da trajetória. E também como que o acréscimo do máximo de primitivos em dois, acarreta em uma melhora na performance para todos os parâmetros.

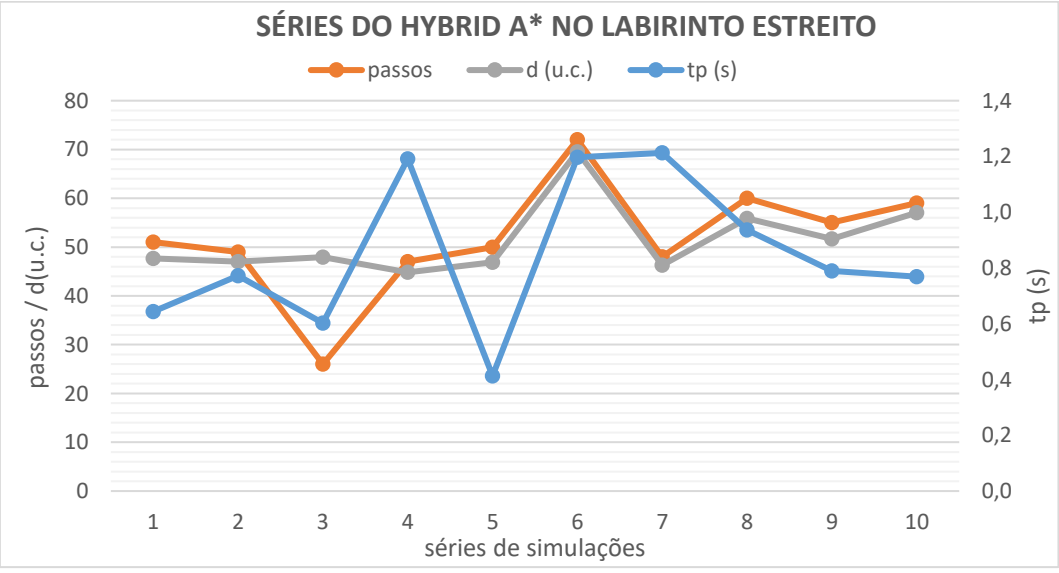


Figura 0.21 - Valores médios de tempo de processamento, quantidade de passos e comprimento de trajetória das séries de simulações do *Hybrid A** no Labirinto Estreito.

A Figura 6.22 apresenta os tempos de processamento de cada simulação individualmente, confirmando que as médias dos tempos de processamento de cada série, de fato refletem. Destaque para as séries de maior tempo médio de processamento, que se encontram também na parte superior do gráfico, intercalando suas posições. E também para a nona série, cuja curva é descendente em sua maior parte.

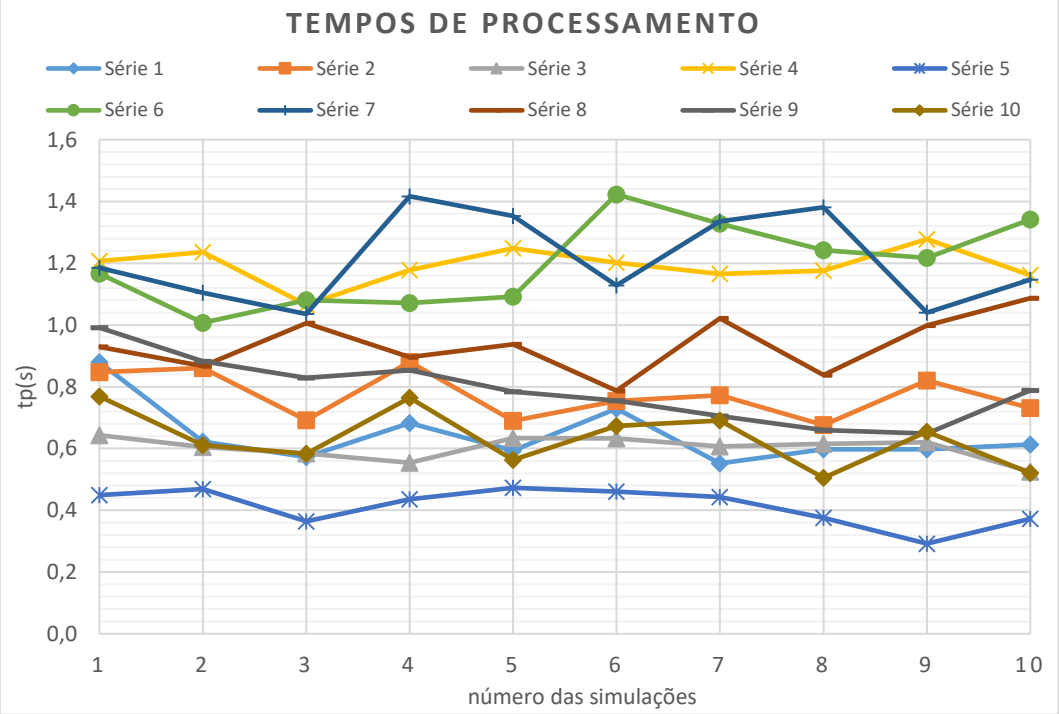


Figura 0.22 - Tempo de processamento de cada simulação do *Hybrid A** no Labirinto Estreito.

Novamente, as curvas do comprimento das trajetórias e da quantidade de passos se assemelham. O gráfico da Figura 6.23, dos comprimentos versus a quantidade de passos indica que um parâmetro acompanha o outro. O motivo de tal proximidade é que, conforme mencionado, os nós, que são também os passos, fazem parte das pequenas curvas que compõem a trajetória. Diferentemente de outros navegadores, o *Hybrid A** não executa um "passo largo" indo de um nó a outro que se encontra mais distante. Ele vai seguindo pelas curvas formadas pelos primitivos, cada um com pequenos passos. Logo, quanto maior a trajetória, maior a quantidade de passos. Nota-se que as curvas dos comprimentos e passos somente não se aproximam nas simulações nas quais a distância de interpolação é diferente, como no caso da terceira série.

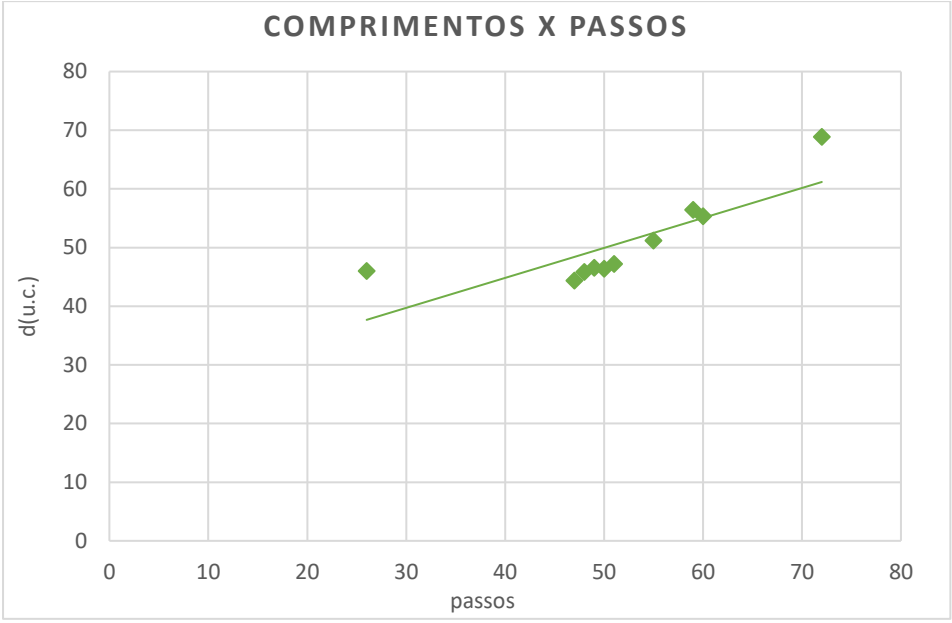
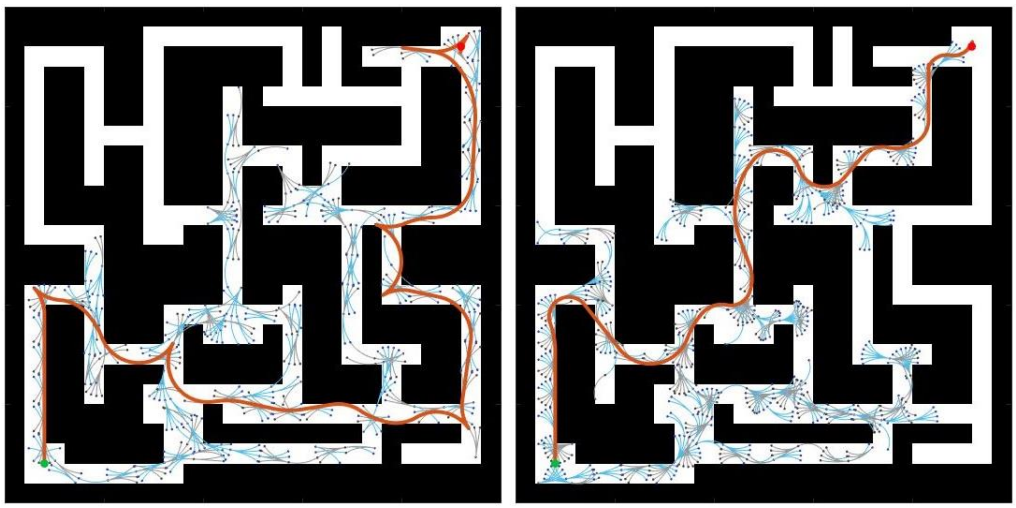


Figura 0.23: Comprimento de trajetória versus quantidade de passos do *Hybrid A** no Labirinto Estreiro.

A Figura 6.24 apresenta a maior e a menor trajetórias geradas nas simulações com o *Hybrid A**, sendo a primeira da sexta série e a segunda da sétima série. Nota-se que, ao traçar curvas próximas a quinas, o planejador não respeita muito bem os limites do mapa. Isso pode ser visto nas duas figuras, na posição (4,10). Esta é uma informação importante que o usuário deve levar em consideração ao utilizar o planejador e ajustar os limites do mapa de maneira adequada.



(a) (b)

Figura 0.24 - Simulações do *Hybrid A** no Labirinto Estreito: (a) na sexta série e (b) na quarta série.

6.6
Comparações das Simulações no Labirinto Estreito

O método de comparação entre os planejadores é o mesmo, selecionando a série de simulações de cada planejador que teve o melhor desempenho em cada um dos parâmetros. A comparação é realizada com os valores médios para evitar que eventuais fatores aleatórios, como um processo interno do computador, aos quais simulações individuais possam estar submetidas, afetem as análises.

6.5.1
Comparação dos Tempos de Processamento

A Tabela 6-6 apresenta as séries cujos tempos de processamento foram os menores de cada planejador, no mapa Labirinto Estreito.

TABELA 0-6 – SIMULAÇÕES NO LABIRINTO ESTREITO COM OS MENORES TEMPOS DE PROCESSAMENTO

Planejador	*Série*	$\overline{tp}(s)$	*passos*	*d(u.c.)*
Dijkstra	1	**20,352**	13,429[1]	46,517[1]
Dyn. Prog.	1	**13,581**	11,571[1]	51,412[1]
PRM	2	**0,072**	16,30[1]	50,939[1]
RRT	3	**0,288**	26	47,044
Hybrid A*	5	**0,413**	50	46,869

[1] valores médios

A Figura 6.25 apresenta o gráfico dos tempos de processamento médios, listados na Tabela 6-6. No mapa Labirinto Estreito, o planejador PRM obteve o melhor desempenho em sua segunda série. Em contrapartida, dentro deste conjunto, o planejador apresenta o segundo maior comprimento de trajetória, apesar deste parâmetro não apresentar grande diferença relativa entre os planejadores. Todavia, é importante ressaltar a grande superioridade do PRM em relação aos demais no quesito dos tempos de processamento.

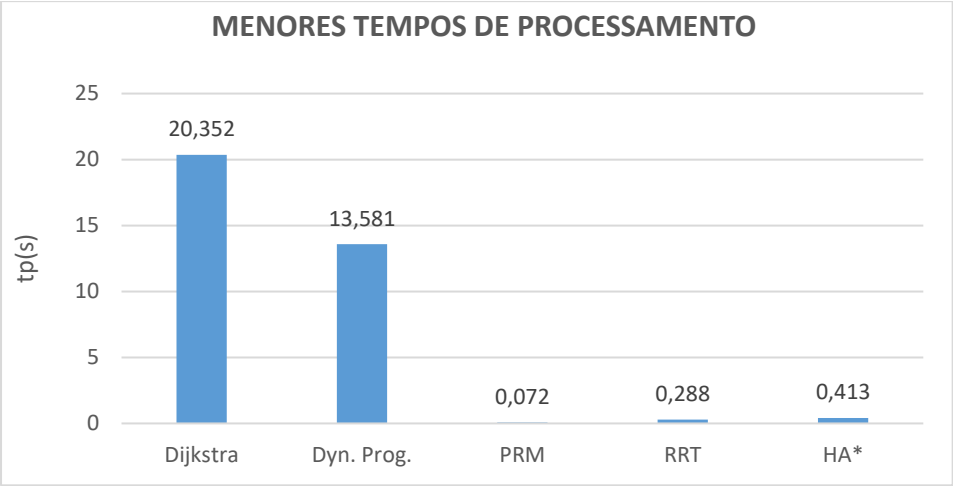


Figura 0.25 - Menores tempos médios de processamento das simulações no Labirinto Estreito.

As diferenças entre os tempos de processamento médios nas simulações no mapa Labirinto Estreito são ainda maiores do que as calculadas na comparação dos tempos do mapa Labirinto. Isso é natural, visto a dificuldade que os planejadores encontraram para gerar as trajetórias, sobretudo Algoritmo de Dijkstra e *Dynamic Programming*. Isso pode ser constatado na quantidade maior de nós requeridos por todos os planejadores, ou, no caso do *Hybrid A**, na incapacidade de gerar trajetórias com o comprimento do primitivo maior que 1,4122 unidades de comprimento. Assim sendo, a maior diferença para o tempo de processamento do PRM foi em relação ao Algoritmo de Dijkstra, cujo melhor tempo ficou 20,280 segundos acima do melhor tempo do PRM.

6.6.2
Comparação das Quantidades de Passos

A Tabela 6-7 apresenta as séries de cada planejador que geraram as menores quantidades de passos nas simulações no mapa Labirinto Estreito. Deve-se fazer uma ressalva quanto à série escolhida para representar o planejador RRT. Duas séries de simulações, a quarta e a quinta, apresentaram a mesma quantidade de passos. Foi adotado então, como critério de desempate o tempo de processamento. Como a quinta série obteve o menor tempo, foi escolhida como representante. Curiosamente, esta série também apresentou a maior trajetória de todas as simulações do RRT.

TABELA 0-7 – SIMULAÇÕES NO LABIRINTO ESTREITO COM AS MENORES QUANTIDADES DE PASSOS

Planejador	*Série*	*passos*	$\overline{tp}(s)$	*d(u.c.)*
Dijkstra	2	**12,600**[1]	33,475	46,279[1]
Dyn. Prog.	3	**11,167**[1]	33,755	48,251[1]
PRM	4	**14,500**[1]	0,097	50,779[1]
RRT	5	**22**	0,861	53,873
Hybrid A*	3	**26**	0,602	47,952

[1] valores médios

A Figura 6.26 apresenta o gráfico das menores quantidades de passos listadas na Tabela 6-7. Mais uma vez, o planejador *Dynamic Programming* apresentou a menor média deste quesito. Em contrapartida, o tempo de processamento da desse planejador é o maior, considerando este conjunto de dados. Comparando o desempenho geral do *Dynamic Programming* aos demais planejadores, tendo a quantidade de passos como principal critério, o PRM se apresenta como uma opção interessante, pois apresenta uma quantidade média de apenas 3,333 passos a mais e com um pequeno acréscimo de 2,528 unidades de comprimento, mas com um tempo de processamento 347,990 vezes menor. Ou seja, a troca seria um pouco de passos e comprimento a mais, por uma velocidade de processamento muito superior.

Deve-se levar em conta também que, em relação ao RRT e ao *Hybrid A**, os valores não se aplicam à médias, mas à menor quantidade absoluta de passos geradas pelo planejador nas simulações no mapa Labirinto Estreito.

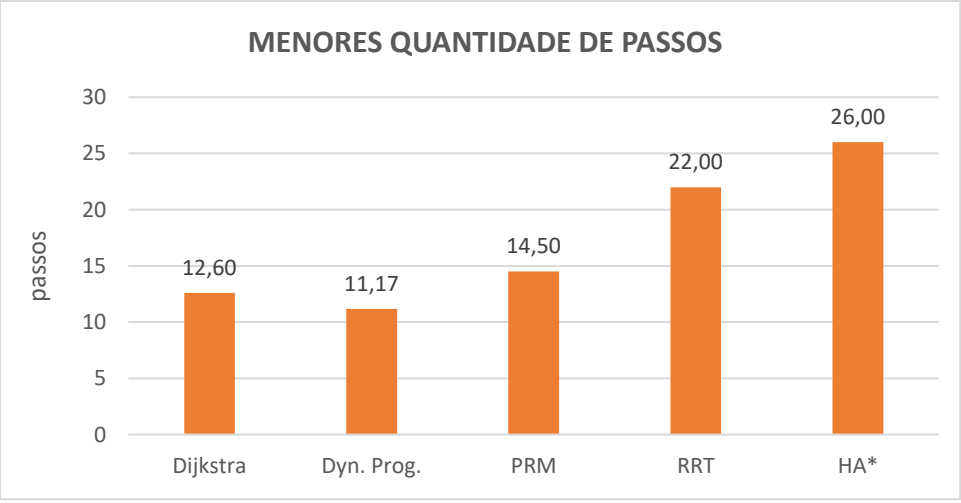


Figura 0.26 - Menores quantidades de passos das simulações no Labirinto Estreito.

6.6.3
Comparação dos Comprimentos das Trajetórias

A Tabela 6-8 abaixo apresenta as séries de simulações, no mapa Labirinto Estreito, cujos comprimentos de trajetórias foram os menores.

TABELA 0-8 – SIMULAÇÕES NO LABIRINTO ESTREITO COM OS MENORES COMPRIMENTOS DE TRAJETÓRIA

Planejador	*Série*	*d(u.c.)*	$\overline{tp}(s)$	*passos*
Dijkstra	4	**44,116**[1]	48,578	14,200[1]
Dyn. Prog.	3	**48,251**[1]	37,755	8,600[1]
PRM	4	**50,556**[1]	0,163	16,00[1]
RRT	3	**47,044**	0,288	26
Hybrid A*	4	**44,825**	1,192	47

[1] valores médios

A seguir, a Figura 6.27 apresenta o gráfico dos menores comprimentos de trajetória, presentes na Tabela 6-8. O Algoritmo de Dijkstra foi o planejador que apresentou a série com o menor comprimento médio. Deve-se levar em conta que este valor foi obtido em sua quarta série de simulações que, no caso do mapa Labirinto Estreito, teve 100% de aproveitamento em todas as tentativas. Contra o planejador pesa o tempo de processamento da série, que não é só a maior do próprio planejador, como é a maior dentre todas as séries listadas. Levando em consideração somente o comprimento da trajetória, o planejador *Hybrid A** é uma alternativa, pois seu menor comprimento de trajetória é apenas 0,709 unidades de comprimento maior que a do Algoritmo de Dijkstra e seu tempo de processamento é 40,753 vezes menor. A desvantagem do *Hybrid A** é sua quantidade de passos, que é superior à demais. Entretanto, este algoritmo não faz mudanças bruscas, buscando sempre curvas suaves. Então, caso haja preocupação com paradas e acionamentos de motores ou desgaste de rodas, a alternativa segue válida.

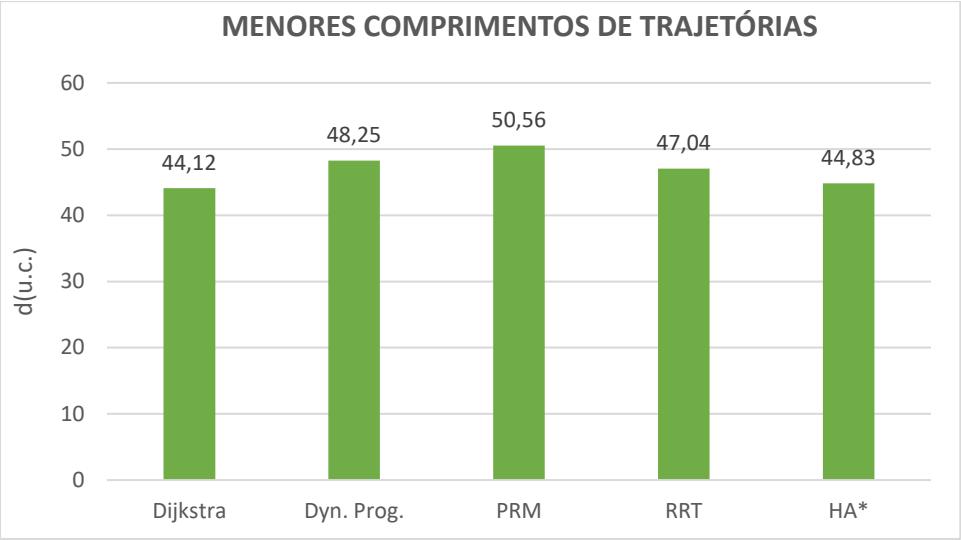
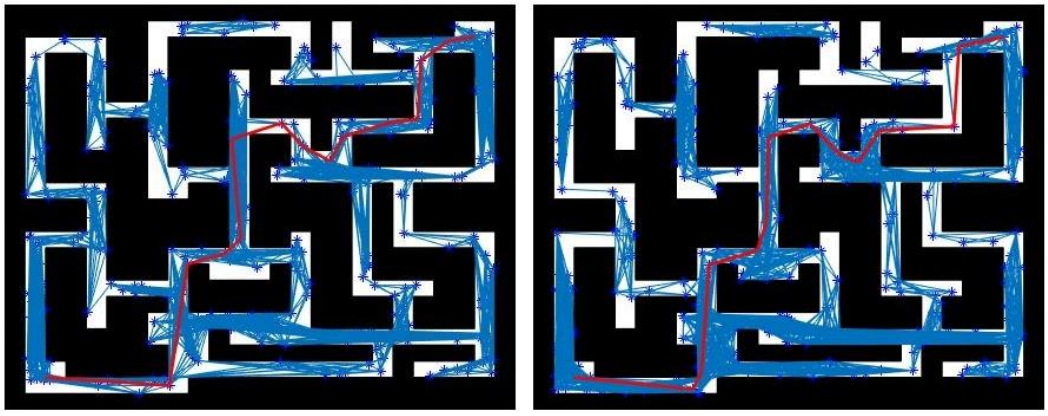


Figura 0.27 - Menores comprimentos de trajetórias das simulações no Labirinto Estreito.

No caso dos planejadores Algoritmo de Dijkstra, RRT e *Hybrid A**, as menores trajetórias geradas apresentam semelhanças na maioria das simulações. Segue, como exemplo, a Figura 6.28 com as trajetórias geradas na terceira e oitava simulação da quarta série do Algoritmo de Dijkstra no Labirinto Estreito, que são muito parecidas. Das dez trajetórias de compõem a série, seis delas também se assemelham a estas, passando pelo mesmo caminho.

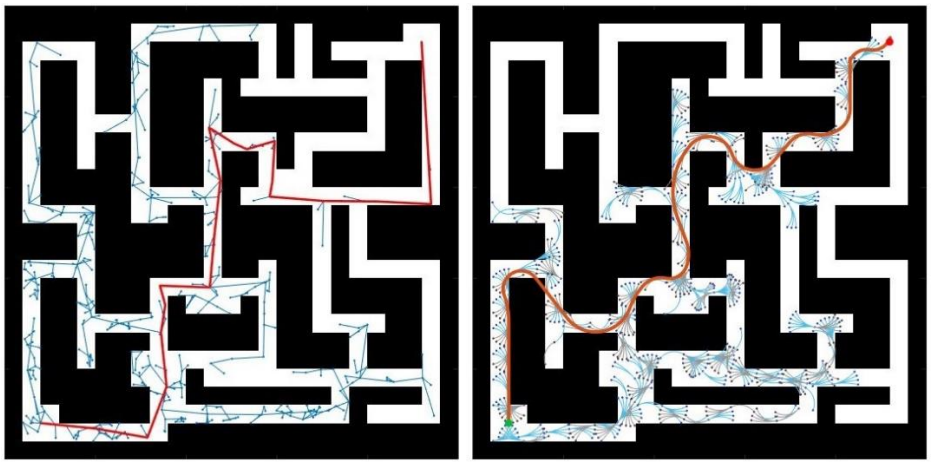


(a) (b)

Figura 0.28 - Terceira (a) e oitava (b) trajetórias da quarta série de simulações do Algoritmo de Dijkstra no Labirinto Estreito.

E segue também a Figura 6.29, que mostra as trajetórias geradas na terceira série do RRT e quarta série do *Hybrid A** que são as que geraram as menores trajetórias desses planejadores. Nota-se que a trajetória gerada pelo *Hybrid A** (b), passa pelo mesmo caminho que as trajetórias do Algoritmo de Dijkstra da Figura

6.28. A trajetória do RRT (a) também é parecida com as demais e segue pelo mesmo caminho por quase toda a trajetória, sofrendo um desvio na parte final, próximo ao ponto (15, 15), fazendo com que essa trajetória fosse um pouco maior que a do Algoritmo de Dijkstra e do *Hybrid A**.



(a) (b)

Figura 0.29 - Trajetórias geradas na (a) terceira série do RRT e (b) quarta série do *Hybrid A**.

7
Simulações no Mapa 3 – Ziguezague

7.1
Simulações com o Algoritmo de Dijkstra no Ziguezague

Nas simulações com o Algoritmo de Dijkstra com o mapa Ziguezague, não foi possível gerar trajetórias em simulações com menos de 70 nós, sendo que só houve resultados consistentes a partir de 110 nós. A partir de 150 nós houve 100% de aproveitamento em todas as séries de simulações, que foram até a marca de 200 nós. Dessa forma, foram quatorze séries de simulações com resultados, sendo quatro com nove resultados e seis com dez resultados.

Os valores médios dos tempos de processamento, número de passos e tamanhos das trajetórias das séries de simulações encontram-se na Tabela 7-1.

TABELA 0-1 – SIMULAÇÕES COM ALGORITMO DE DIJKSTRA NO ZIGUEZAGUE

Série	*nós*	$\overline{tp}$(s)	$\overline{passos}$	$\bar{d}$(u. c.)	Obs.
1	110	6,353	12,778	128,417	9 resultados
2	120	7,375	13,556	126,763	9 resultados
3	130	8,893	13,889	125,330	9 resultados
4	140	10,334	13,556	124,865	9 resultados
5	150	11,665	13,500	125,645	
6	160	15,443	13,500	125,335	
7	170	19,384	14,000	124,081	
8	180	23,190	14,300	123,712	
9	190	23,711	13,700	124,481	
10	200	26,695	13,500	123,114	

Legenda:

nós = Número de nós que ocupam a área livre do mapa.

$\overline{tp}$ = Tempo de processamento médio, medido em segundos.

$\overline{passos}$ = Número médio de nós utilizados para conectar os pontos inicial e objetivo.

$\bar{d}$ = Comprimento médio das trajetórias, medido em unidades de comprimento (u.c.).

A Figura 7.1, a seguir, apresenta os gráficos de tempo de processamento, quantidade de passos e comprimento de trajetória das simulações. Novamente verifica-se que o tempo médio de processamento sobe conforme aumenta-se o número de nós das simulações e que a quantidade média de passos mantem-se dentro de uma faixa com pouca variação, tendo uma diferença máxima de 1,522 passos. Da mesma forma, os tamanhos médios das trajetórias também se

concentram próximos aos valores de 123, 124 e 125 unidades de comprimento, sendo que as duas primeiras séries, com os menores números de nós e 9 resultados apresentam os maiores valores de 128,417 e 126,763 unidades de comprimento.

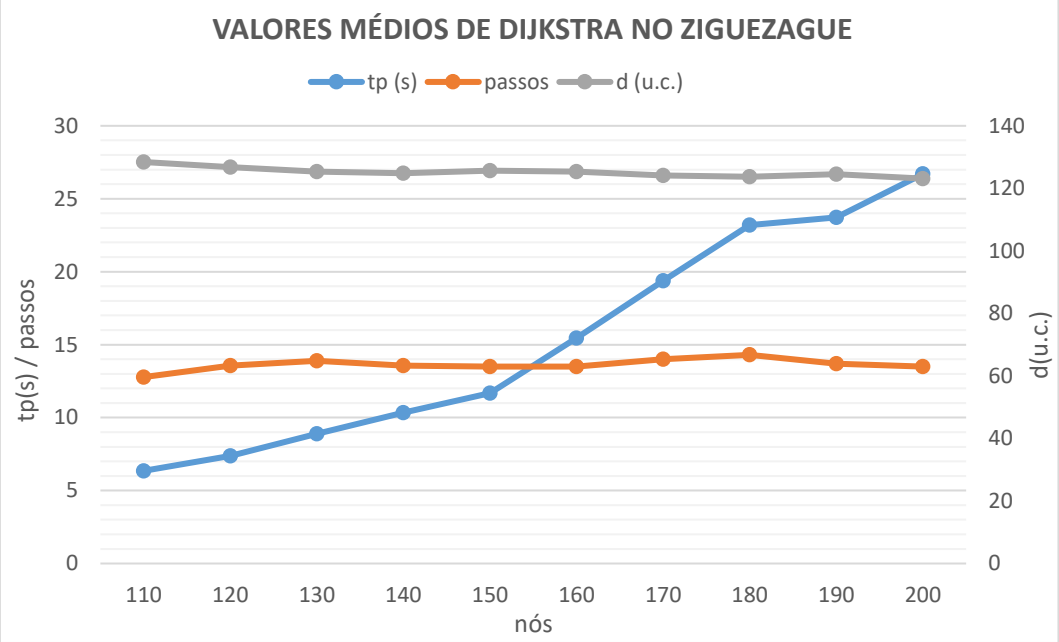


Figura 0.1 - Valores médios do tempo de processamento, quantidade de passos e comprimento de trajetória das simulações com Dijkstra no mapa Ziguezague.

As Figuras 7.2, 7.3 e 7.4 apresentam os valores de tempo de processamento, quantidade de passos e comprimento da trajetória, respectivamente, de cada simulação individual.

O gráfico da Figura 7.2 confirma a tendência apresentada no gráfico da Figura 7.1, de que o número de nós da simulação é o fator mais determinante para o tempo de processamento, mas não é o único. Os tempos das simulações com 190 nós foram atípicos se comparados ao demais, devido às grandes oscilações. Na série com 200 nós houve uma queda dos tempos das três últimas simulações. Tais fenômenos podem se dar por diversos fatores, tais como processos internos do computador, por exemplo. Por isso optou-se por trabalhar com as médias a fim de mitigar esses eventos.

Os gráficos das Figuras 7.3 e 7.4 confirmam as poucas variações nos parâmetros de quantidades de passos e comprimento da trajetória, ficando a maior parte desses valores concentrados dentro de uma faixa. Esse comportamento é natural para este mapa em que só há uma opção de caminho. As poucas variações

de passos e comprimento se dá principalmente pela aleatoriedade em que os nós são posicionados na área livre do mapa.

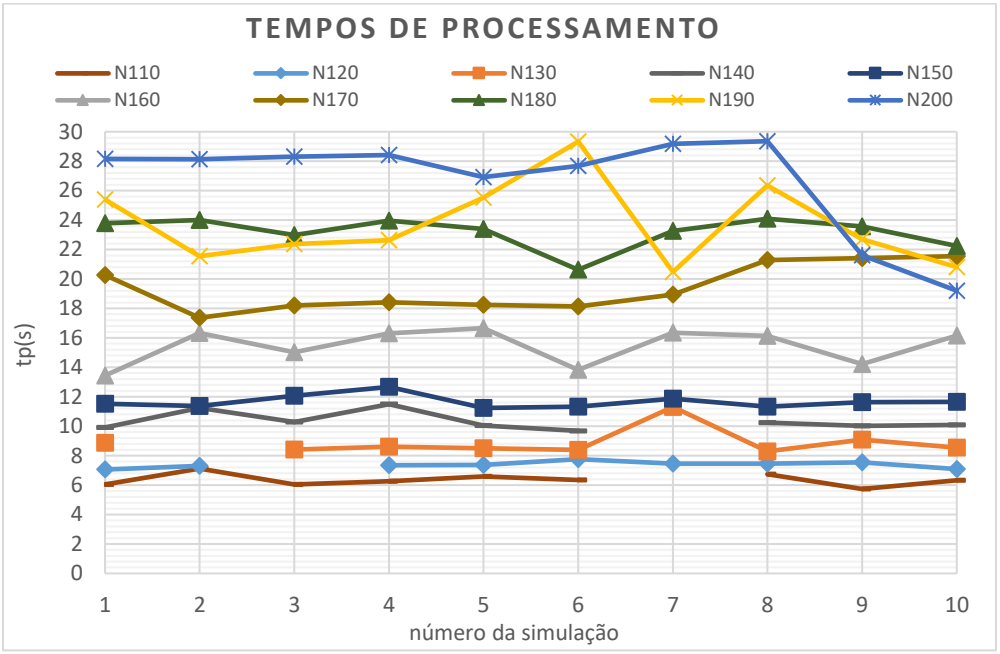


Figura 0.2 - Tempo de processamento de cada simulação do Algoritmo de Dijkstra no Ziguezague.

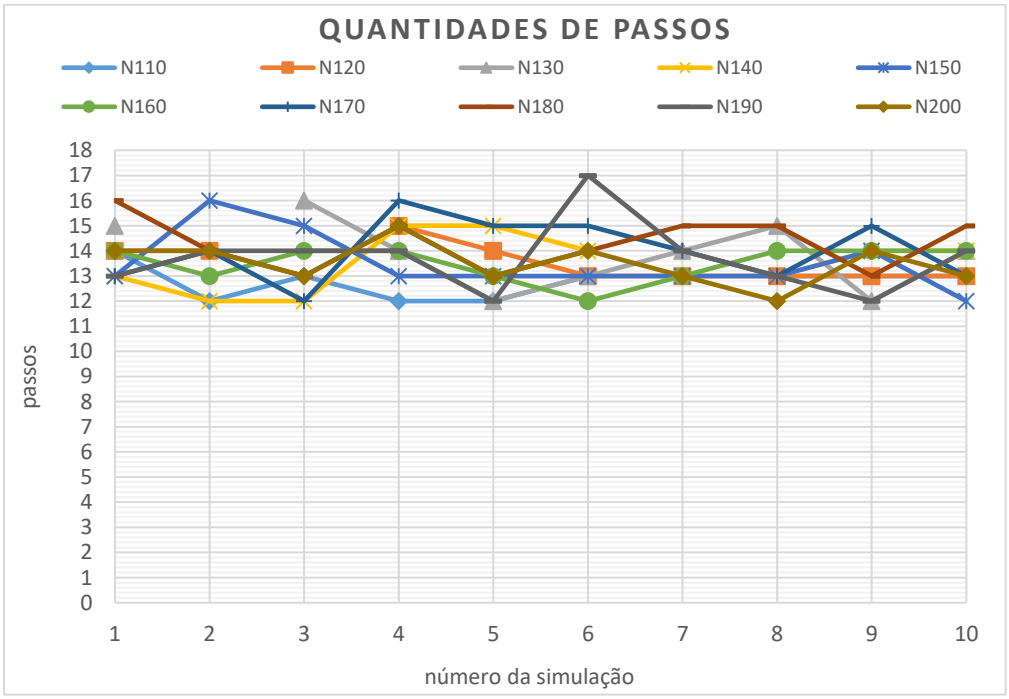


Figura 0.3 - Número de passos de cada simulação com Dijkstra no mapa Ziguezague.

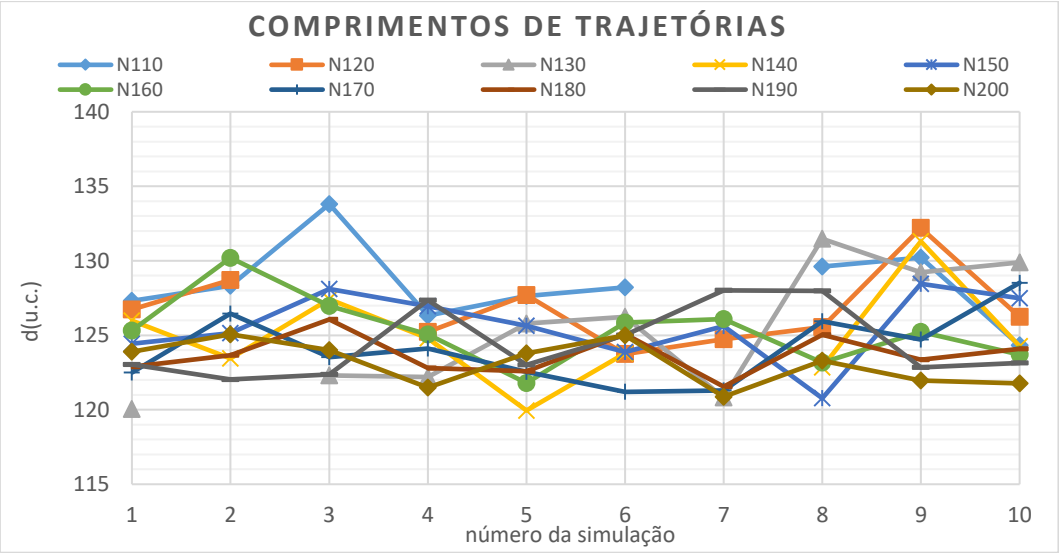
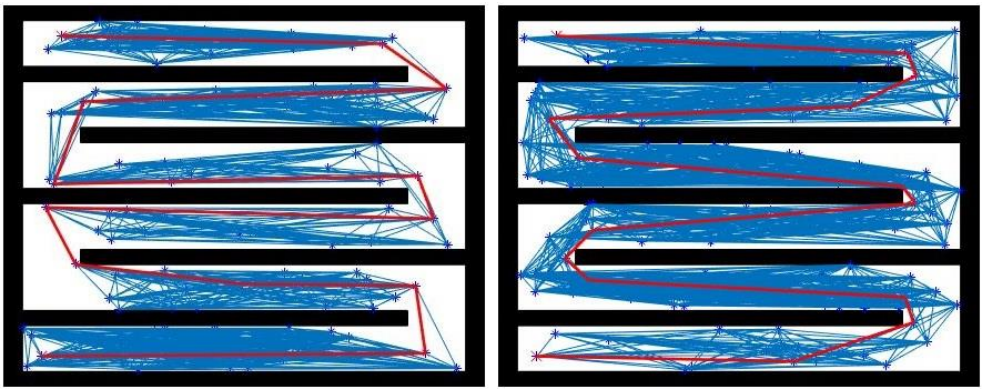


Figura 0.4 - Comprimento de trajetória de cada simulação do Algoritmo de Dijkstra no Ziguezague.

A Figura 7.5, a seguir, apresenta a maior trajetória dentre todas as simulações, gerada na terceira simulação com 110 nós, e também a menor, da quinta simulação com 140 nós. Observa-se que a quantidade de nós nas curvas do mapa é um fator importante para o comprimento da trajetória. Enquanto que na simulação da Figura 7.5(a), a trajetória utiliza somente dois nós para mudar de direção, necessitando se afastar das arestas, na simulação da Figura 7.5(b), as curvas são feitas mais próximas das mesmas, utilizando três nós, em quatro das cinco áreas de inversão de direção. A falta de pelo menos um nó nas áreas de mudança de direção foram o motivo de algumas simulações não terem gerado trajetória.



(a) (b)

Figura 0.5 - Simulações do Algoritmo de Dijkstra no Ziguezague: (a) terceira, com 110 nós e (b) quinta, com 140 nós.

7.2
Simulações com Dynamic Programming no Ziguezague

As simulações com *Dynamic Programming* também começaram a apresentar resultados em quantidade significativa a partir de 100 nós. A partir de 120 nós, os resultados foram se alternando entre 9 e 10 resultados conforme a Tabela 7-2, que também apresenta os valores médios dos tempos de processamento, quantidade de passos e comprimento das trajetórias de todas as séries de simulações. Optou-se por simular 11 séries, para compensar a segunda série, que teve um rendimento de 80%.

TABELA 0-2 – SIMULAÇÕES COM DYNAMIC PROGRAMMING NO ZIGUEZAGUE

Série	*nós*	$\overline{tp}(s)$	$\overline{passos}$	$\bar{d}(\text{u.c.})$	Obs.
1	100	4,934	12,889	131,518	9 resultados
2	110	6,055	12,750	132,298	8 resultados
3	120	9,375	12,900	132,150	
4	130	9,120	12,889	131,469	9 resultados
5	140	10,017	13,100	132,060	
6	150	11,995	13,400	130,460	
7	160	16,901	13,444	129,611	9 resultados
8	170	18,005	13,000	129,320	9 resultados
9	180	21,758	12,600	129,549	
10	190	23,960	12,900	129,676	
11	200	26,825	12,900	131,557	

Legenda:

nós = Número de nós que ocupam a área livre do mapa.

$\overline{tp}$ = Tempo de processamento médio, medido em segundos.

$\overline{passos}$ = Número médio de nós utilizados para conectar os pontos inicial e objetivo.

$\bar{d}$ = Comprimento médio das trajetórias, medido em unidades de comprimento (u.c.).

A Figura 7.6 a seguir, apresentam os gráficos dos tempos de processamento, quantidades de passos e comprimentos de trajetórias, no quais pode-se observar a relação entre o aumento dos números de nós e o tempo de processamento, como é característico do planejador. Mas, novamente, não há evidências de influência do número de nós sobre a quantidade de passos ou o comprimento da trajetória. Estes dois parâmetros, aliás, apresentam baixa variação em suas médias, o que é natural, visto que o presente mapa não oferece mais do que uma opção de rota e as diferenças entre as trajetórias são definidas por detalhes.

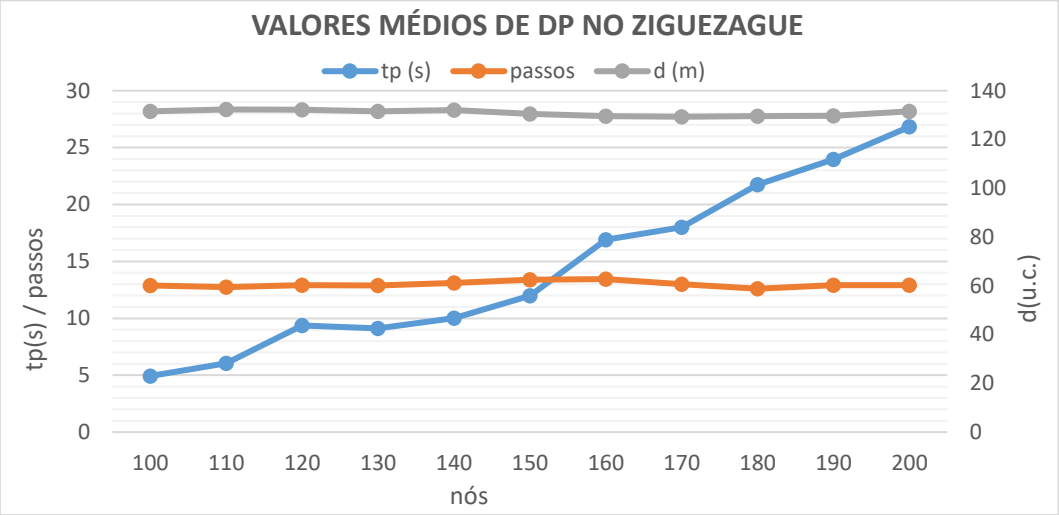


Figura 0.6 - Valores médios de tempo de processamento, quantidade de passos e comprimento de trajetórias das simulações do *Dynamic Programming* no Ziguezague.

As Figuras 7.7 a 7.9 apresentam os tempos de processamento, quantidades de passos e comprimentos das trajetórias de cada simulação individual. Na Figura 7.7, os tempos de cada simulação refletem também as médias apresentadas na Figura 7.6. Percebe-se muita proximidade e alternância de posição das séries com 100 a 150 nós, assim como um distanciamento maior entre as séries e baixa oscilação das séries de 160 a 200 nós. O gráfico da Figura 7.8 demonstra a proximidade das quantidades de passos de cada simulação, apontada pela curva das médias. Todas as séries se concentram dentro da mesma faixa, que vai de 12 a 15 passos. Pode-se dizer então que estes são os valores mínimos exigidos e máximos necessários para se completar uma trajetória no mapa Ziguezague com o planejador *Dynamic Programming*. Assim como as demais, a Figura 7.9 também reflete o equilíbrio entre os comprimentos de trajetórias, apontado pelas médias na Figura 7.6. Todas as simulações encontram-se dentro da mesma faixa, que vai de 120 a 140 unidades de comprimento, aproximadamente. Como o caminho a ser seguido é um só, a aleatoriedade na qual os nós são posicionados determinam essas pequenas variações nos comprimentos.

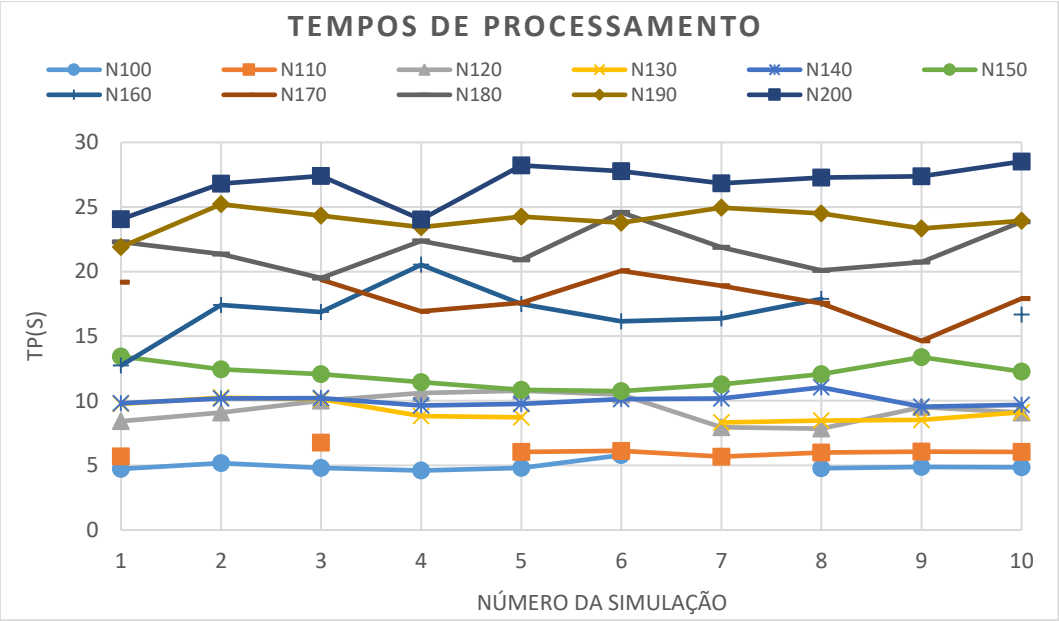


Figura 0.7 - Tempos de processamento de cada simulação do *Dynamic Programming*, no Ziguezague.

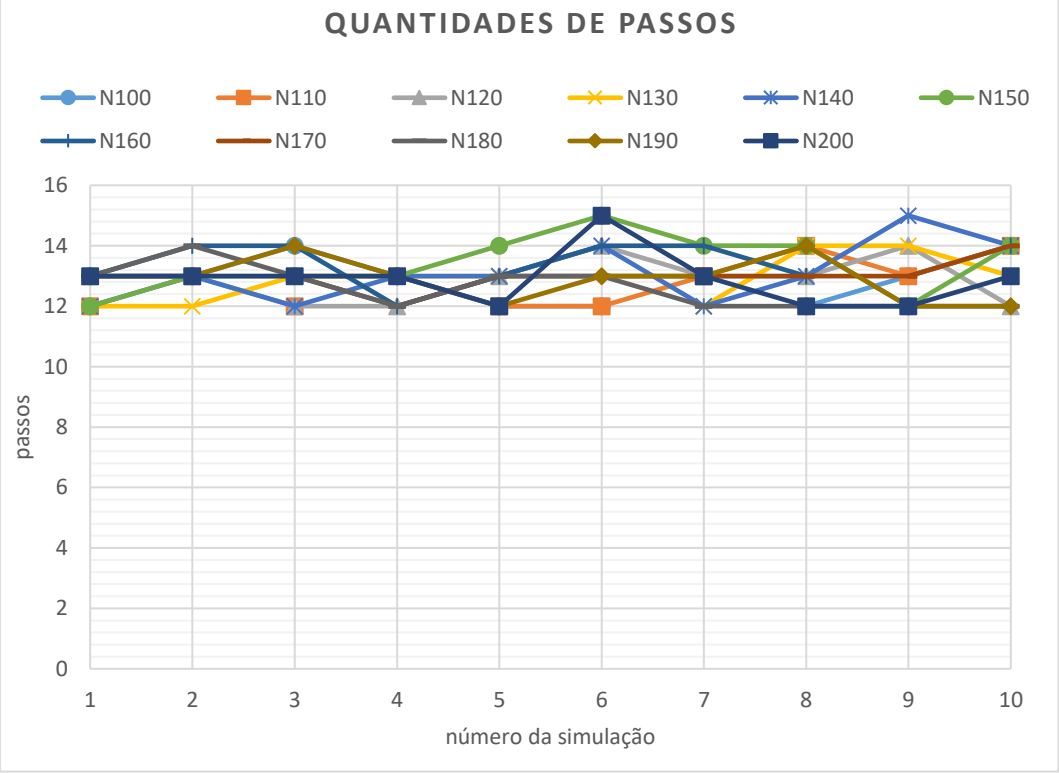


Figura 0.8 - Quantidades de passos de cada simulação do *Dynamic Programming* no Ziguezague.

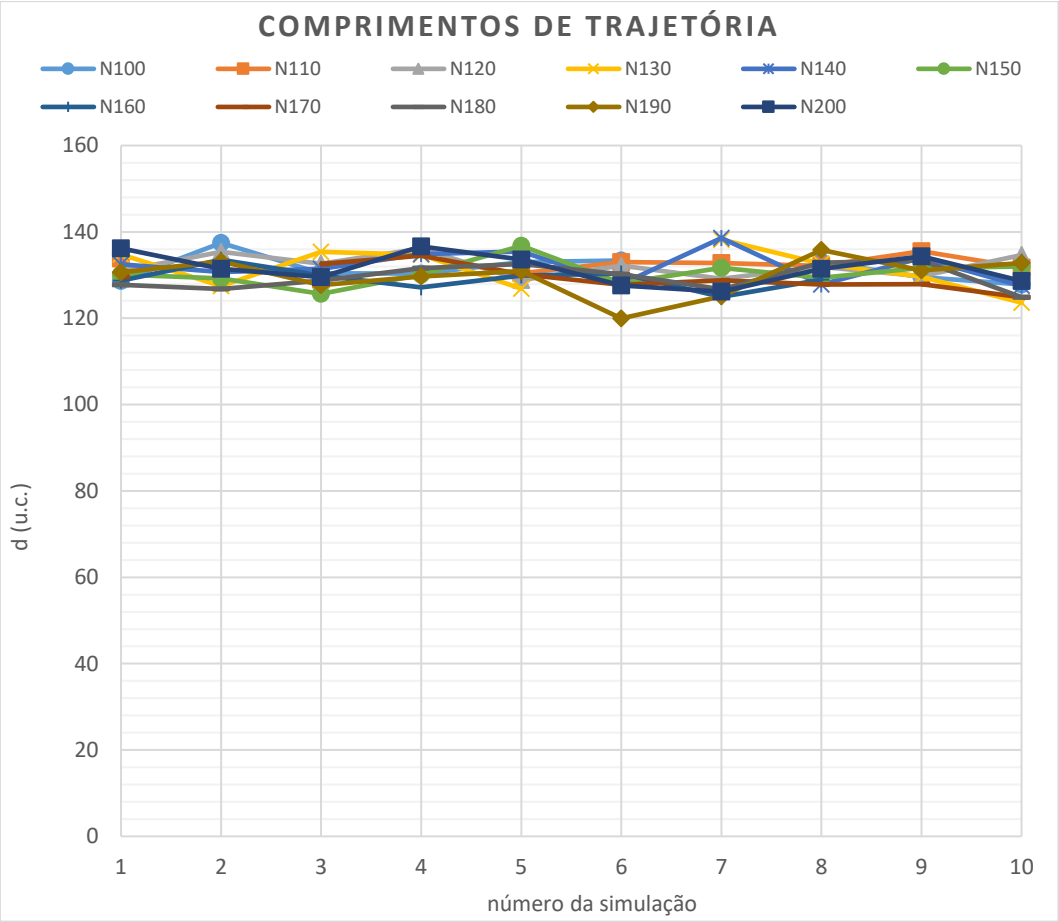
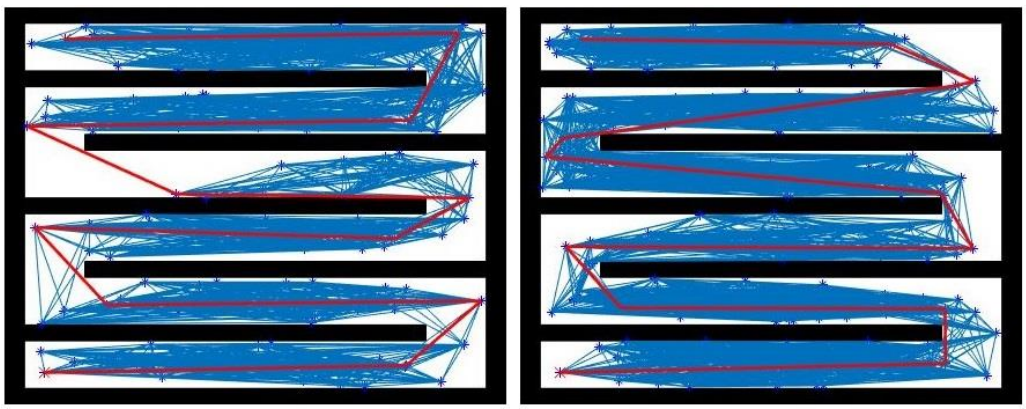


Figura 0.9 - Comprimentos das trajetórias de cada simulação do *Dynamic Programming* no Ziguezague.

Figura 7.10 apresenta a sétima simulação da série com 140 nós, que é a maior trajetória gerada pelo *Dynamic Programming* no mapa Ziguezague e a sexta simulação da série com 190 nós, que é a menor trajetória gerada.



(a) (b)

Figura 0.10 - Simulações do *Dynamic Programming* no Ziguezague: (a) Sétima, com 140 nós e (b) sexta, com 190 nós.

7.3
Simulações com PRM no Ziguezague

As simulações do PRM com o mapa Ziguezague seguiram o mesmo procedimento das simulações anteriores, divididas em quatro séries de dez simulações, iniciando com delta igual a 10 e incrementando em 5 a cada série. O planejador não enfrentou dificuldades neste mapa, sendo que os resultados numéricos se encontram na Tabela 7-3.

TABELA 0-3 – SIMULAÇÕES COM PRM NO ZIGUEZAGUE

Série	Δ	$\overline{nós}$	$\overline{tp}(s)$	$\overline{passos}$	$\bar{d}(u.c.)$
1	10	75	0,055	23,70	131,480
2	15	73	0,076	19,80	132,215
3	20	78	0,121	16,10	130,617
4	25	74	0,213	15,60	130,418

Legenda:

Δ = delta.

$\overline{nós}$ = Número de nós médios gerados durante as simulações.

$\overline{tp}$ = Tempo de processamento médio, medido em segundos.

$\overline{passos}$ = Número médio de nós utilizados para conectar os pontos inicial e objetivo.

$\bar{d}$ = Comprimento médio das trajetórias, medidos em unidades de comprimento (u.c.).

A Figura 7.11 a seguir apresenta os gráficos das médias dos tempos de processamento, quantidades de passos e comprimentos das trajetórias geradas nas simulações do PRM no mapa Ziguezague. O gráfico *tp*(*s*) indica haver uma relação entre o valor do delta e o tempo de processamento. Segundo a Tabela 7-3, este fato não está relacionado com a quantidade de nós. Já a curva "*passos*" indica existir uma a relação inversa da quantidade média de passos em relação ao valor do delta. O entendimento desta relação é intuitivo pois o mapa Ziguezague se caracteriza por grandes retas e, quanto maior o delta, mais distantes os nós conseguem se conectar. Logo, quanto maior o delta, menos passos são necessários para percorrer uma determinada distância. Os comprimentos médios das trajetórias, representados na série *d*(*u.c.*), apresentam valores muito próximos, demonstrando que, independentemente do tempo de processamento e da quantidade de passos, os valores dos deltas foram adequados. Sendo que o mapa Ziguezague oferece apenas um caminho, é natural que as médias que as trajetórias geradas apresentem valores médios próximos.

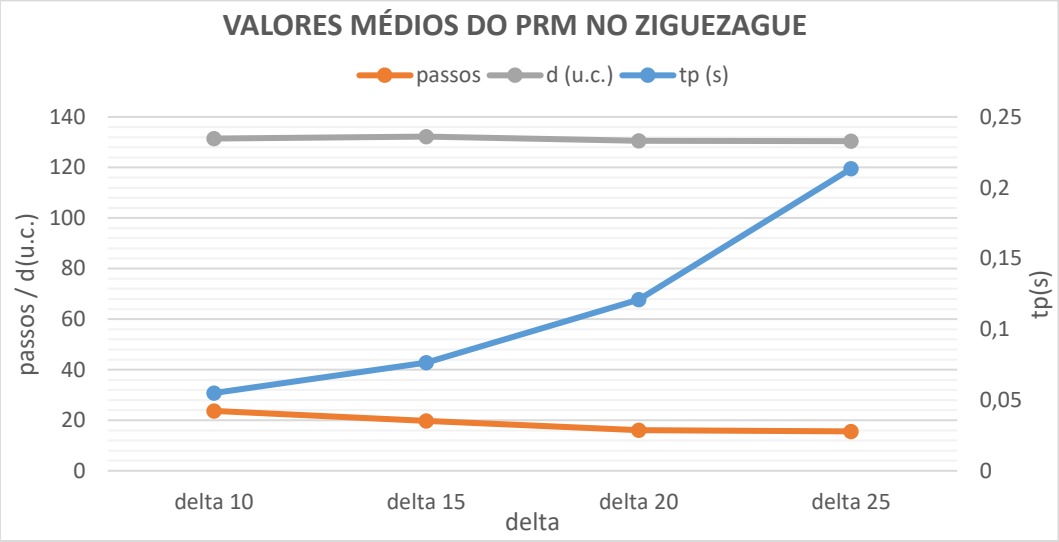


Figura 0.11: Valores médios de tempos de processamento, quantidade de passos e comprimento de trajetórias das simulações do PRM no Ziguezague.

A seguir, nas Figuras 7.12 a 7.14 são vistos, respectivamente, os tempos de processamento, quantidades de passos e comprimentos de trajetórias de cada simulação individual.

O gráfico dos tempos de processamento da Figura 7.12 revela uma proximidade maior entre os tempos das simulações individuais do que a curva das médias. Destacam-se os dois picos da quarta série, logo na primeira e na terceira simulações. Há um pico também na primeira simulação da terceira série, enquanto que a primeira e segunda séries não apresentam oscilações bruscas. Esses picos podem explicar porque os tempos médios de processamento da Figura 7.11 aumentam desproporcionalmente de uma série para outra: da primeira série para a segunda, ambas sem picos, o aumento foi de 27,632%. Da segunda para a terceira, que tem um pico, o aumento foi de 37,190% e, da terceira para a quarta série, com dois picos relativamente grandes, o aumento foi de 43,194%. Os gráficos da Figura 7.13, mostram que as médias presentes no gráfico da Figura 7.11 refletem bem as quantidades de passos das simulações individuais, com duas primeiras séries bem divididas em suas faixas de valores e as duas últimas se entrelaçando. A Figura 7.13 não apresenta nenhuma novidade quanto ao comprimento das trajetórias, pois individualmente todas ocupam a mesma faixa, que vai de 125,526 a 137,842 unidades de comprimento.

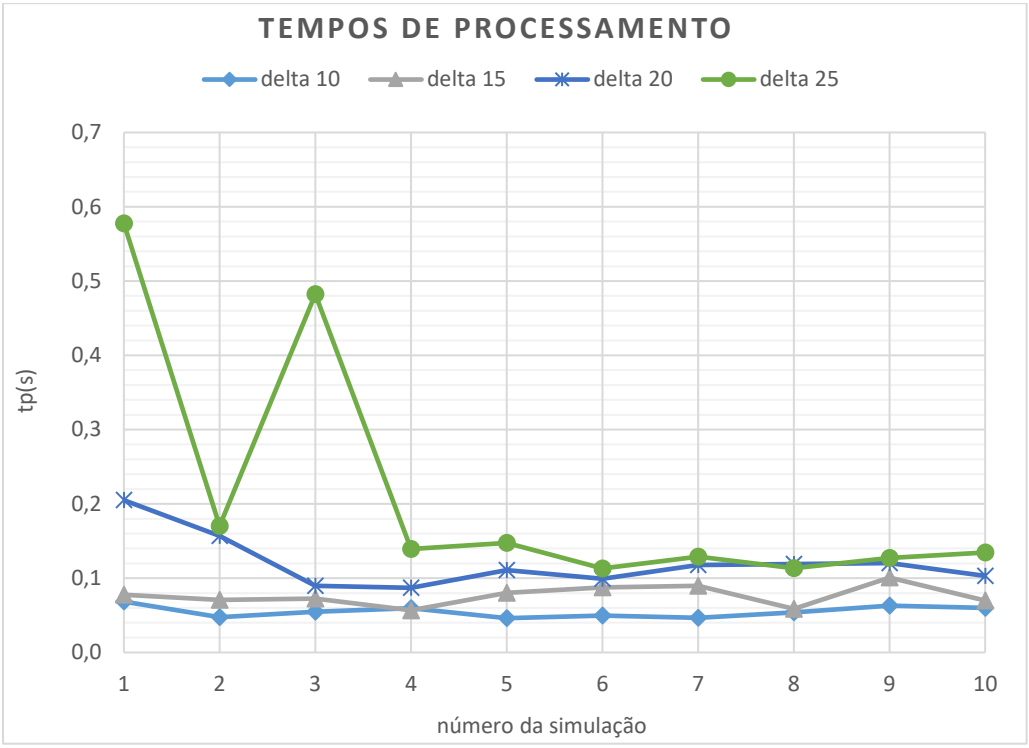


Figura 0.12: Tempos de processamento de cada simulação do PRM no Ziguezague.

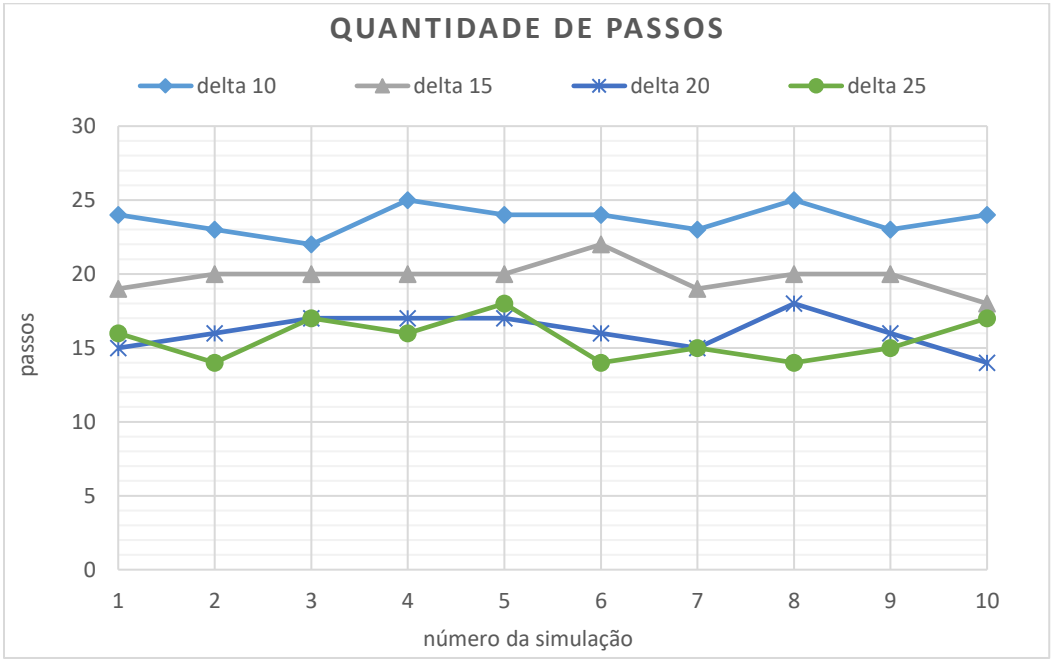


Figura 0.13: Quantidade de passos de cada simulação do PRM no Ziguezague.

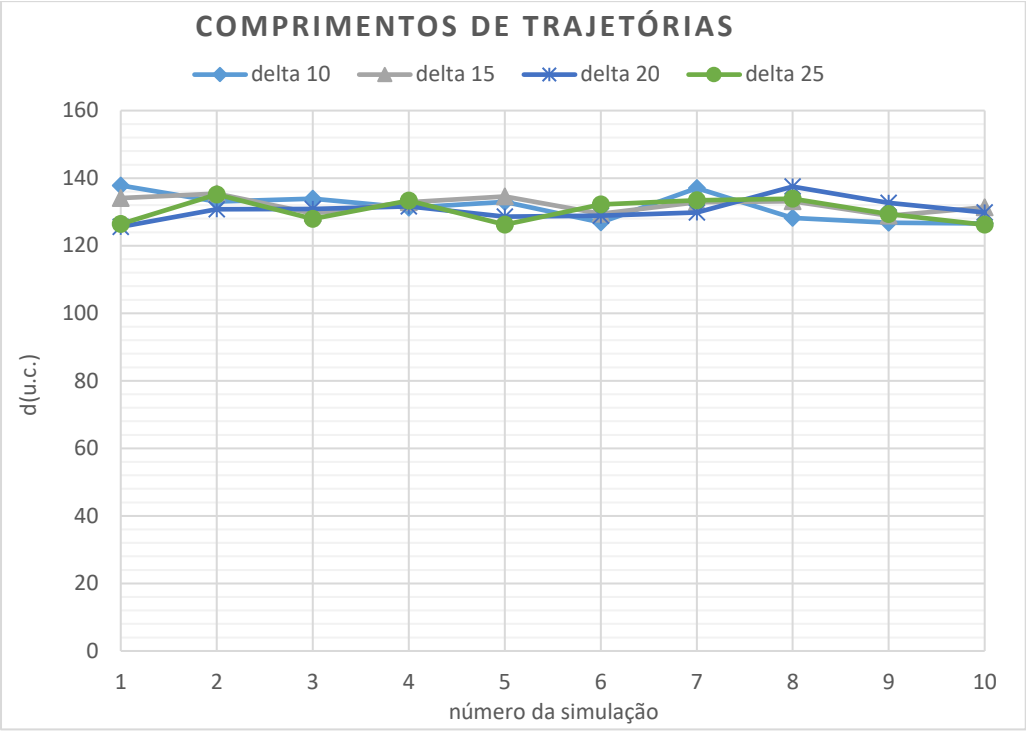
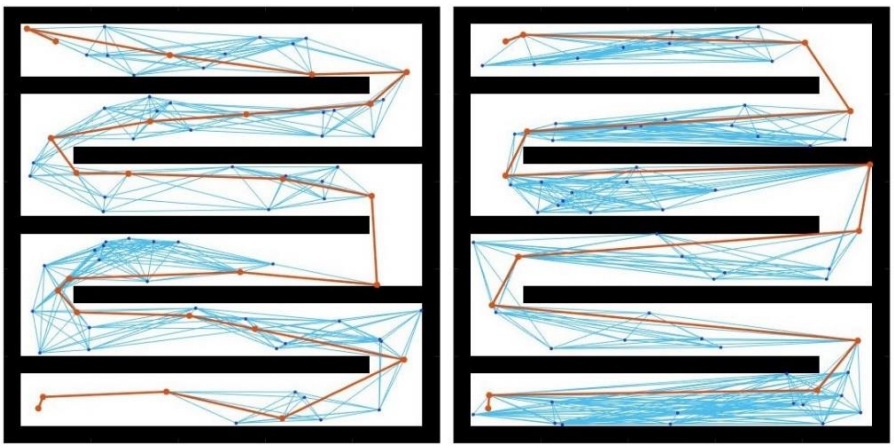


Figura 0.14: Comprimento de trajetória de cada simulação do PRM no Ziguezague.

A seguir, na Figura 7.15, são apresentadas as trajetórias geradas na quarta simulação da primeira série do PRM no mapa Ziguezague, com delta 10 e também a segunda simulação da quarta série, com delta 25. Essas simulações se caracterizam por terem sido as primeiras a apresentarem, respectivamente, a menor e a maior quantidade de passos das simulações: 14 e 25 passos.



(a) (b)

Figura 0.15 - Simulações do PRM no Ziguezague: (a) quarta, da primeira série e (b) segunda, da quarta série

7.4
Simulações com RRT no Ziguezague

As simulações com o RRT no mapa Ziguezague ocorreram de forma diferente das anteriores pois não foi possível formar uma trajetória com os valores iniciais do planejador: número máximo de nós e iterações de $1x10^4$ e delta igual a 0,5. O primeiro delta foi então de 1 unidade de comprimento, recebendo incrementos até atingir o valor de 25 u.c. Executou-se uma série dobrando a quantidade máxima de nós e de iterações e voltando com o delta igual a 0,5, que gerou resultados, mas com um custo computacional muito elevado, chegando a travar o computador algumas vezes. Essa abordagem foi então descartada e optou-se por modificar o número de máximo de nós e iterações, mas mantendo o delta igual a 3, pois foi o que obteve os melhores resultados numéricos de tempo de processamento e comprimento de trajetórias. Os resultados dessas simulações podem ser conferidos na Tabela 7-4.

TABELA 0-4 – SIMULAÇÕES COM RRT NO ZIGUEZAGUE

Série	*max nós*	*max iter*	Δ	*nós*	*iter*	$\overline{tp}(s)$	*passos*	*d(u.c.)*
1	1×10^4	1×10^4	1	3516	9218	7,205	164	152,724
2	1×10^4	1×10^4	2	1937	5514	3,001	90	151,202
3	1×10^4	1×10^4	3	1846	4478	2,515	75	146,575
4	1×10^4	1×10^4	5	2241	5176	2,854	73	152,406
5	1×10^4	1×10^4	7	2285	5176	2,800	71	147,877
6	1×10^4	1×10^4	10	2280	5176	2,995	71	148,768
7	1×10^4	1×10^4	15	2279	5176	3,012	72	148,145
8	1×10^4	1×10^4	20	2278	5176	3,022	73	149,002
9	1×10^4	1×10^4	25	2278	5176	2,998	73	149,002
10	2×10^4	2×10^4	3	1844	4483	2,693	69	140,380
11	1×10^5	1×10^5	3	1808	4456	2,407	75	145,302

Legenda:

max nós = número máximo de nós.

max iter = número máximo de iterações.

Δ = delta.

nós = número de nós gerados durante as simulações.

iter = iterações geradas durante as simulações.

$\overline{tp}$ = tempo de processamento médio, medido em segundos.

passos = número médio de nós utilizados para conectar os pontos inicial e objetivo.

d = Comprimento da trajetória, medido em unidades de comprimento (u.c.).

A Figura 7.16 apresenta os tempos de processamento médios, as quantidades de passos e os comprimentos de trajetórias das séries de simulações do RRT no mapa Ziguezague. A curva *tp*(*s*) dos tempos de processamento, mostra como um valor de delta relativamente baixo, igual a 1, impacta na construção da trajetória. Isso se reflete também no gráfico dos passos. A dobra do delta de um para dois fez com que o tempo de processamento caísse 58,348% e a quantidade de passos, 45,122%. Esses parâmetros, sobretudo o tempo de processamento, são diretamente afetados pelo número de iterações. No caso da primeira série, ele é 67,174% maior que o da segunda. Os demais tempos de processamento não apresentam grandes variações, assim como as quantidades de passos. Os comprimentos de trajetórias não apresentam grandes variações de uma série para outra, sendo o destaque para a décima série que completou a trajetória com 140,380 nós. Tal fato, contudo, não se deve ao aumento da quantidade máxima de nós e iterações atribuídos a esta série pois, segundo a Tabela 7-4, o número de nós gerados não foi superior as das outras séries, tampouco o número de iterações. As menores trajetórias foram geradas nas séries 3, 10 e 11 que também apresentaram os menores tempos médios de processamento. Essas séries têm em comum o delta igual a três. Outro ponto a se notar é a semelhança entre as curvas de tempo médio e passos. Esta relação é vista com maiores detalhes mais adiante.

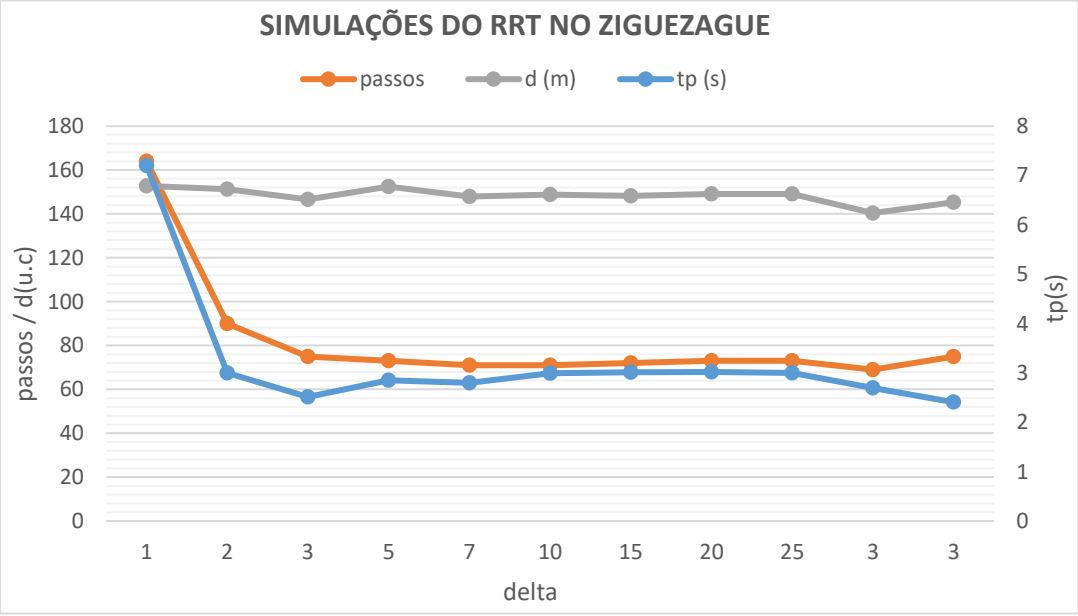


Figura 0.16 - Tempos de processamento médios, quantidade de passos e comprimento de trajetórias das simulações do RRT no Ziguezague.

A Figura 7.17 apresenta os valores de tempo de processamento de cada simulação individual, refletindo, de certa forma, o gráfico das médias dos tempos e, ao mesmo tempo mostrando um equilíbrio entre os tempos de processamento de cada série, com exceção da primeira, cujos valores estão acima das demais.

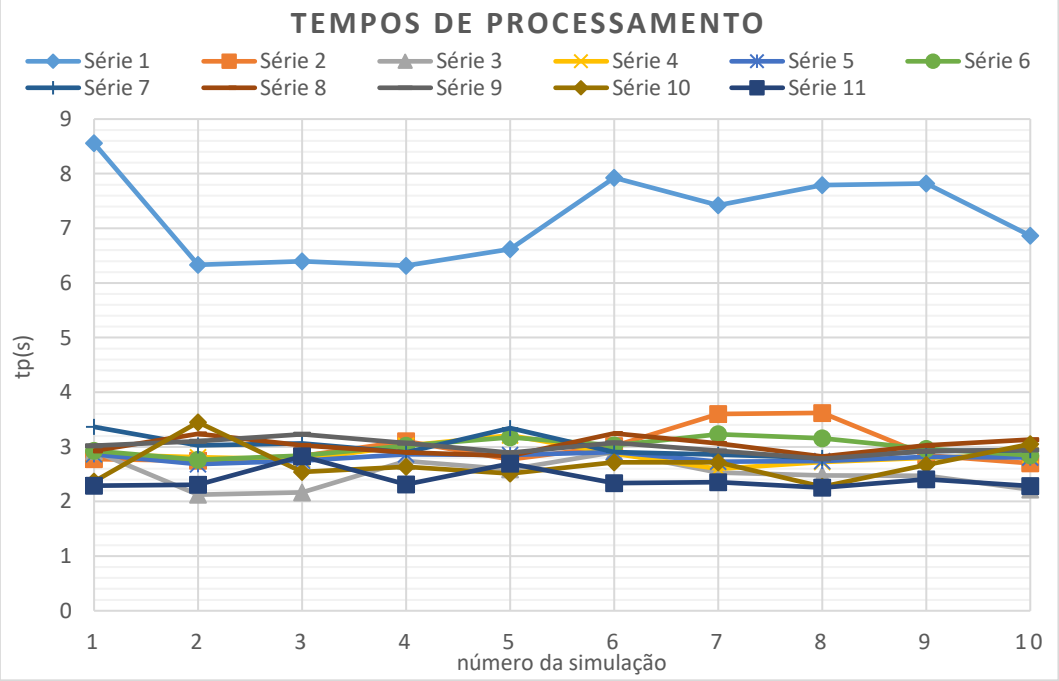


Figura 0.17 - Tempos de processamento de cada simulação do RRT no Ziguezague.

Voltando à Figura 7.16, as curvas dos tempos de processamento médio e das quantidades médias de passos, além de se assemelharem, evidenciam um equilíbrio desses parâmetros a partir da terceira série de simulações. Buscando verificar esta relação, traçou-se o gráfico da Figura 4.18, que mostra o tempo versus os passos. Nota-se que há uma concentração de pontos na região que corresponde a 70 passos e entre 2,5 a 3,0 segundos, caracterizando o equilíbrio evidenciado nas curvas da Figura 7.17 e os pontos destoantes referentes à segunda e primeira séries. Por ambos os gráficos, o que se conclui é que o fator determinante para os comportamentos de ambos os parâmetros é o delta. Um valor pequeno de delta acarreta na necessidade de geração de mais nós e iterações, levando à uma quantidade maior de passos para se atingir o objetivo e também a um maior tempo de processamento.

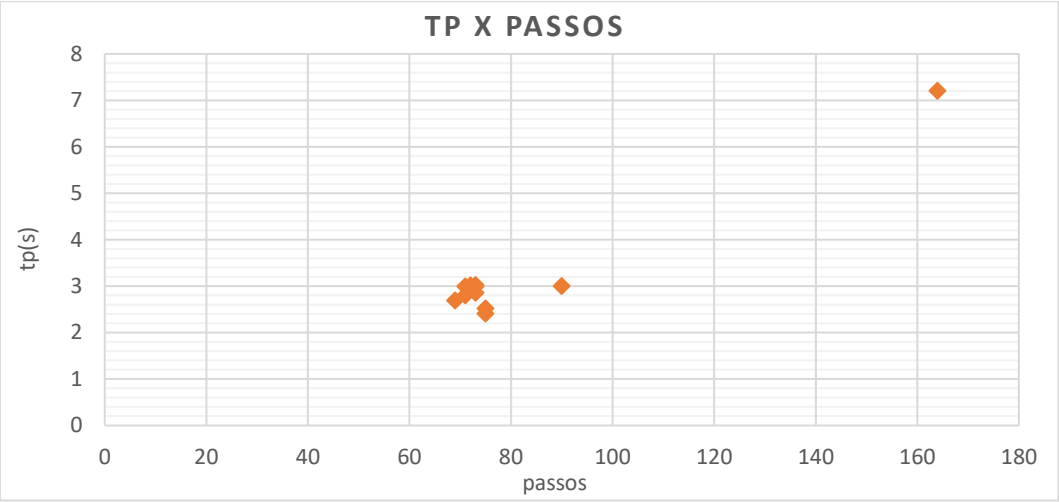
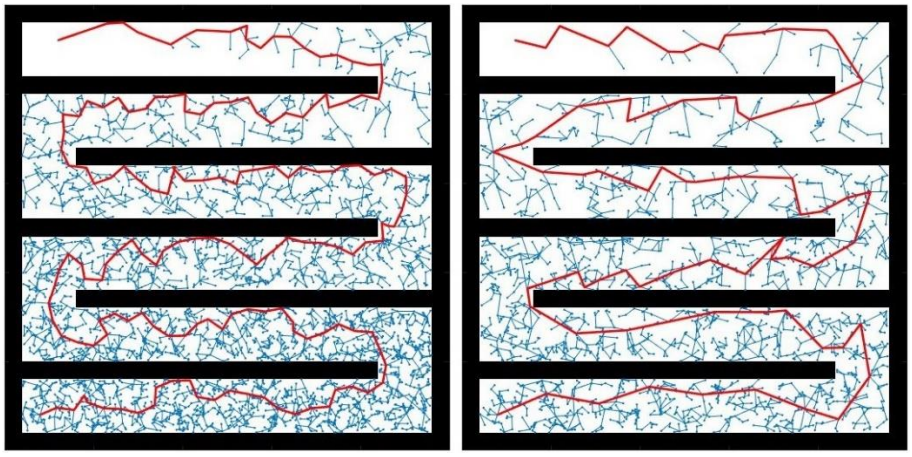


Figura 0.18 - Tempos de processamento médios versus quantidades de passos nas simulações do RRT no Ziguezague.

A seguir são vistas, na Figura 7.19, a maior trajetória gerada pelo RRT no mapa Ziguezague, que é a da primeira série de simulações e a menor trajetória, que é a da décima série.



(a) (b)

Figura 0.19 - Simulações do RRT no Ziguezague: (a) primeira série e (b) décima série.

7.5
Simulações com Hybrid A* no Ziguezague.

As simulações com o planejador *Hybrid A** no mapa Ziguezague seguiram o procedimento normal de simulações, alterando sutilmente os valores dos parâmetros, avaliando os impactos causados e, ao final, concatenando as modificações que causaram melhorias nos parâmetros de saída. Algumas

configurações do planejador prejudicaram a capacidade do planejador em gerar trajetórias. A primeira delas foi o aumento do custo de retorno de seu valor *default*, 3, para 4, resultando na falha para alcançar o objetivo. Além disso, o planejador não respeitou os obstáculos do mapa. Outras configurações cujos resultados foram negativos foram ajustar o comprimento do primitivo para 4 e o raio mínimo para 3, ou o primeiro parâmetro para 5 e o segundo para 4 ou ajustar o comprimento do primitivo para 3 e o número de primitivos por nó, para 3 também. Todas as demais simulações e seus resultados podem ser conferidas na Tabela 7-5.

TABELA 0-5 – SIMULAÇÕES COM HYBRID A* NO ZIGUEZAGUE

Ser	*Lprim*	*Rmin*	*Nprim*	*FC*	*RC*	*DC*	*ID*	*AEI*	$\overline{tp}(s)$	*passos*	*d(u.c.)*
1	2	2	5	1	3	0	1	5	1,131	134	134,116
2	3	2	5	1	3	0	1	5	0,426	128	128,555
3	2	2	7	1	3	0	1	5	1,037	131	131,978
4	3	2	7	1	3	0	1	5	0,822	133	133,414
5	2	2	5	1	3	1	1	5	1,029	134	134,116
6	2	2	5	1	3	0	2	5	1,085	67	137,622
7	2	2	5	1	3	0	1	4	1,129	134	134,116
8	2	2	3	1	3	0	1	5	0,515	133	133,616
9	2	2	5	1	3	1	2	5	1,138	67	137,622
10	3	2	5	1	3	1	1	5	0,414	128	128,555

Legenda:

Lprim = comprimento dos primitivos.

Rprim = raio mínimo de curvatura.

Nprim = número máximo de primitivos por nó.

FC = custo de avanço (do inglês *forward cost*).

RC = custo de retorno (do inglês *reverse cost*).

DC = custo de mudança de direção (do inglês *direction switching cost*).

ID = distância de interpolação (do inglês *interpolation distance*).

AEI = intervalo de expansão analítica (do inglês *analytic expansion interval*).

$\overline{tp}$ = tempo de processamento médio, medido em segundos.

passos = número médio de nós utilizados para conectar os pontos inicial e objetivo.

d = Comprimento da trajetória, medido em unidades de comprimento (u.c.).

A Figura 7.20 a seguir apresenta os gráficos dos tempos médios de processamento, das quantidades de passos e dos comprimentos das trajetórias das séries de simulações do *Hybrid A** no mapa Ziguezague.

A curva dos tempos médios varia bastante de uma série para outra, mostrando quais alterações de parâmetros ocasionaram os maiores efeitos sobre o

tempo de processamento, como, por exemplo, o aumento do comprimento do primitivo de 2, na primeira série, para 3 na segunda. Apenas essa alteração provocou uma redução de 62,334% no tempo de processamento. Nota-se também como que a soma de dois fatores que, isoladamente geraram impacto positivo, não se traduz em resultados melhores. Por exemplo: na quinta série, a alteração do custo de mudança de direção de zero para um, provocou uma pequena melhora no tempo de processamento. Na sexta série, o incremento da distância de interpolação em mais 1, também causou apenas uma leve melhora no tempo de processamento. Em contrapartida, na nona série em que esses dois parâmetros foram modificados da mesma forma, o impacto foi negativo para o tempo de processamento. Mas o contrário também ocorre, como na décima série, na qual a alteração do comprimento do primitivo e do custo de mudança de direção geraram uma redução de tempo bastante significativa, de 63,395% em relação ao tempo da primeira série, que utiliza os valores *default* do planejador.

O gráfico das quantidades de passos mostra a pequena oscilação do parâmetro, que fica na faixa entre 128 e 134 passos. As exceções são a sexta e nona séries, que possuem 67 passos ambas, a metade da quantidade que o planejador gera em sua configuração padrão. Tal redução se dá pela distância de interpolação igual a 2, comum à ambas as séries. Esse parâmetro significa a distância máxima que a trajetória poderá avançar entre dois nós. Logo, um valor mais elevado significa poder cruzar uma determinada distância com menos passos.

A série *d*(*u.c.*) indica que há bastante proximidade entre os comprimentos de trajetória pois o mapa é de caminho único. Como o planejador trabalha com valores pré-estabelecidos do comprimento dos primitivos e raio mínimo de curvatura, alguns valores de comprimento de trajetória se repetem. A menor trajetória ocorre em duas séries: na segunda e na décima, que possuem o comprimento do primitivo ajustado em 3 e 5 primitivos por nó. A quarta série que também possui comprimento do primitivo igual a 3, mas número de primitivos por nó igual a 7, não repetiu o mesmo comprimento de trajetória das outras duas.

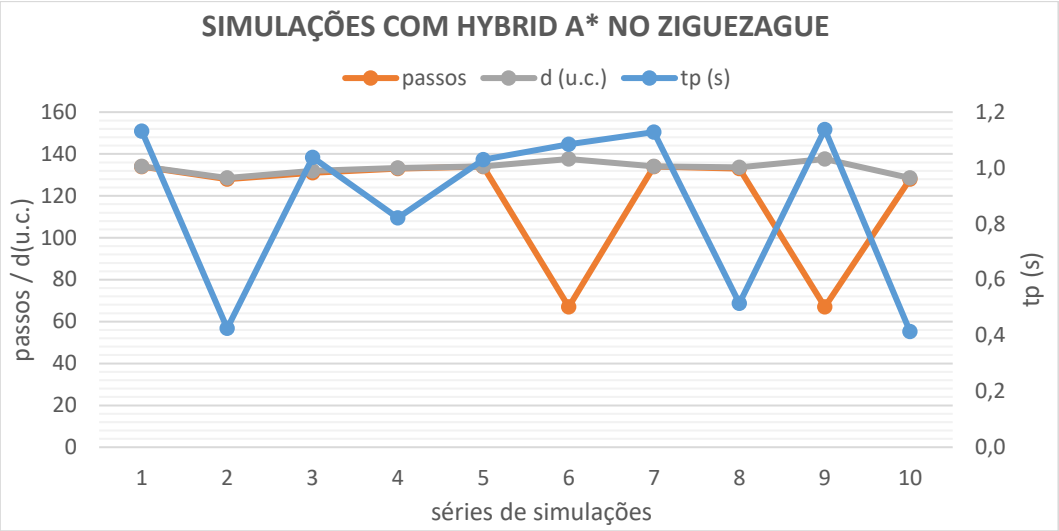


Figura 0.20 - Tempos de processamento médios, quantidade de passos e comprimentos de trajetória das simulações do *Hybrid A** no Ziguezague.

A Figura 7.21, a seguir, apresenta os tempos de processamento de cada simulação individualmente. A curva dos tempos médios reflete bem como foram os tempos das simulações individuais, tendo como destaque um aumento na nona série a partir da quinta simulação, cujo pico ocorreu na nona. Esta série acabou sendo a que apresentou o maior tempo médio de processamento.

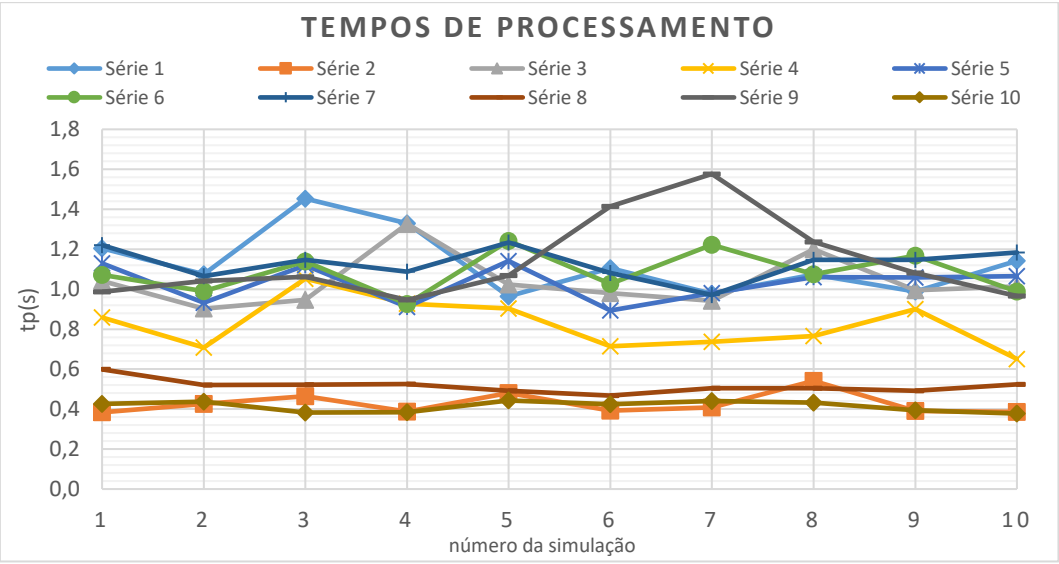
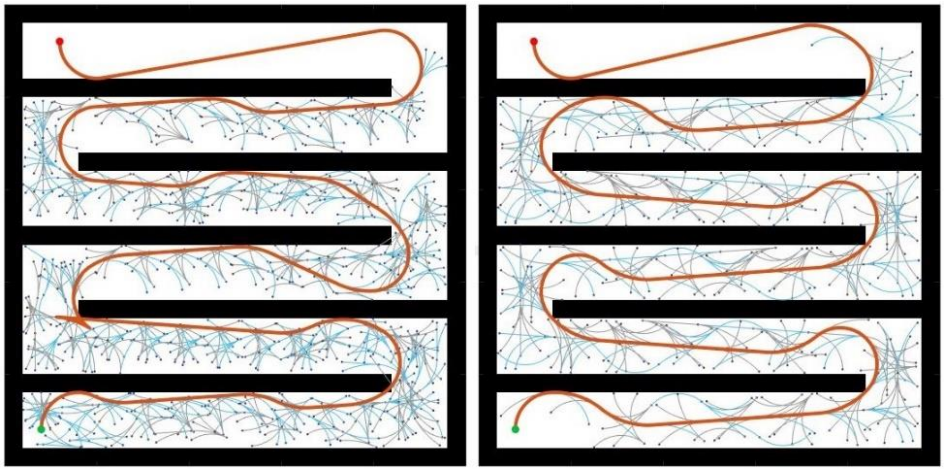


Figura 0.21 - Tempos de processamento de cada simulação do *Hybrid A** no Ziguezague.

A Figura 7.22 apresenta a maior trajetória gerada nas simulações com *Hybrid A**, cuja primeira ocorrência foi na sexta série e a menor trajetória, que foi gerada na segunda série.



(a) (b)

Figura 0.22 - Simulações do *Hybrid A** no Ziguezague: (a) sexta série e (b) segunda série.

7.6
Comparações das simulações no Ziguezague.

7.6.1
Comparação dos Tempos de Processamento

A Tabela 7-6 apresenta as séries que obtiveram os menores tempos de processamento de cada planejador, no mapa Ziguezague.

TABELA 0-6 – SIMULAÇÕES NO ZIGUEZAGUE COM OS MENORES TEMPOS DE PROCESSAMENTO

Planejador	*Série*	$\overline{tp}(s)$	*passos*	*d(u.c.)*
Dijkstra	1	**6,353**	12,778[1]	128,417[1]
Dyn. Prog.	1	**4,934**	12,889[1]	132,298[1]
PRM	1	**0,055**	23,700[1]	131,480[1]
RRT	11	**2,407**	75	145,302
Hybrid A*	10	**0,414**	128	128,555

[1] valores médios

A Figura 7.23 apresenta o gráfico dos menores tempos de processamento médios, conforme a Tabela 7-6. Mais uma vez o planejador PRM obteve o menor tempo de processamento, muito à frente ao segundo colocado, o *Hybrid A**. Seu comprimento de trajetória, dentro deste conjunto, é o terceiro melhor, estando próximo ao menor - que foi o do Algoritmo de Dijkstra - e sua quantidade de passos também foi a terceira melhor, apesar de ser o dobro da melhor, que também foi a

do Algoritmo de Dijkstra. Contudo, o desempenho do tempo de processamento muito superior aos demais, em muito compensa esses outros parâmetros.

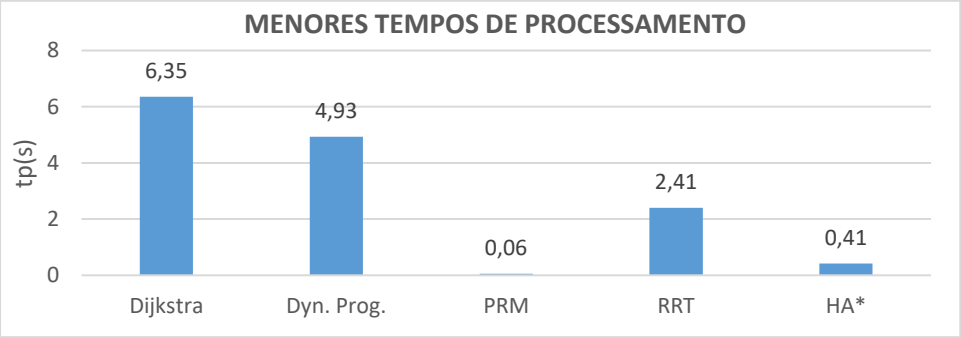


Figura 0.23 - Menores tempos médios de processamento das simulações no Ziguezague.

7.6.2
Comparação das Quantidades de Passos

As menores quantidades de passos de cada planejador, geradas nas simulações do mapa Ziguezague, estão presentes na Tabela 7-7. O planejador *Hybrid A** apresentou duas séries com a mesma menor quantidade de passos, conforme discutido na Seção 4.5.5. Logo, o critério de desempate, de se selecionar a série com o menor tempo de processamento, foi mais uma vez utilizado.

TABELA 0-7 – SIMULAÇÕES NO ZIGUEZAGUE COM AS MENORES QUANTIDADES DE PASSOS

Planejador	*Série*	*passos*	$\overline{tp}(s)$	*d(u.c.)*
Dijkstra	1	**12,778**[1]	6,353	128,417[1]
Dyn. Prog.	9	**12,600**[1]	21,758	129,549[1]
PRM	4	**15,600**[1]	0,213	130,418[1]
RRT	10	**69**	2,693	140,380
Hybrid A*	6	**67**	1,085	137,622

[1] valores médios

A Figura 7.24 apresenta o gráfico das menores quantidades de passos, conforme os valores da Tabela 7-7. O planejador *Dynamic Programming* apresentou a menor quantidade média de passos novamente. Mas, assim como ocorreu para os outros dois mapas, seu tempo de processamento, dentro do conjunto das menores quantidades de passos, é o maior. O Algoritmo de Dijkstra também possui uma quantidade média de passos muito próxima à do *Dynamic*

Programming, com uma diferença de apena 0,178 passo médios, com um tempo médio de processamento 3,425 vezes menor. O comprimento médio de trajetória do Algoritmo de Dijkstra também é menor que o do *Dynamic Programming* em 1,132 unidades de comprimento. Por esses motivos, o planejador Algoritmo de Dijkstra pode ser considerado superior ao *Dynamic Programming*, quando o principal parâmetro avaliado é a quantidade de passos.

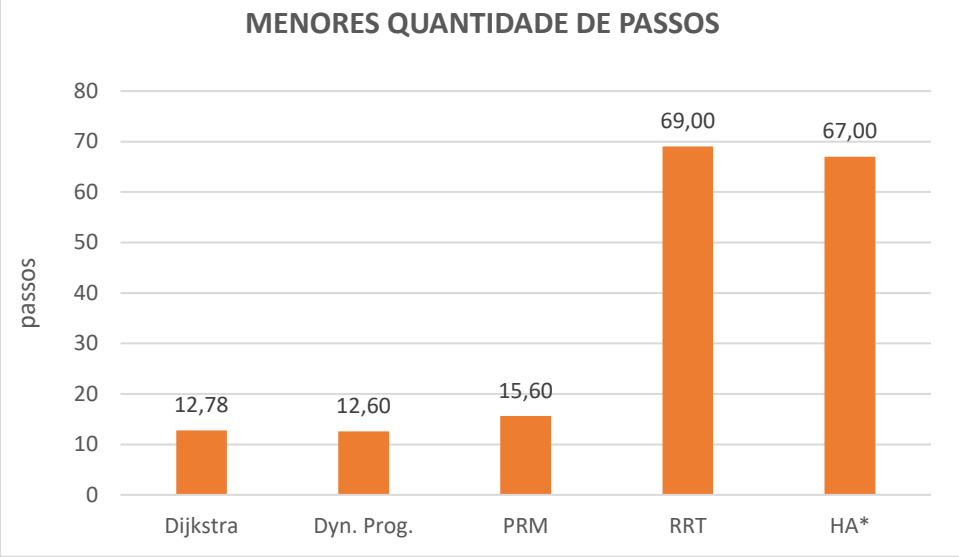


Figura 0.24 - Menores quantidades de passos das simulações no Ziguezague.

O gráfico ainda reforça a proximidade ente as quantidades de passo do *Dynamic Programming* e Algoritmo de Dijkstra e salienta o distanciamento dos algoritmos que crescem suas estruturas na forma de ramos, como o RRT e o *Hybrid A**, para os demais, no que diz respeito à quantidade de passos. Isso ocorre devido ao próprio princípio de funcionamento do planejador que acrescenta seus nós a partir de um ponto fixo e a distâncias estabelecidas. Logo, ele não pode dar "saltos" de um nó para outro em uma trajetória longa. Essa é a vantagem sobretudo do *Dynamic Programming* e do Algoritmo de Dijkstra pois sua distribuição randômica de nós propicia justamente esses "saltos" entre nós, percorrendo longas distâncias.

7.6.3
Comparação dos Comprimentos das Trajetórias

A Tabela 7-8 abaixo apresenta as séries de simulações, no mapa Ziguezague, cujos comprimentos de trajetórias foram os menores. Para o planejador *Hybrid A**

houve necessidade de se aplicar novamente a regra do desempate pois tanto a segunda quanto a décima séries apresentaram os mesmos menores comprimentos de trajetória, mas a última processou suas trajetórias em tempos menores, em média.

TABELA 0-8 – SIMULAÇÕES NO ZIGUEZAGUE COM OS MENORES COMPRIMENTOS DE TRAJETÓRIA

Planejador	*Série*	*d(u.c.)*	$\overline{tp}(s)$	*passos*
Dijkstra	10	**123,114**[1]	26,695	13,500[1]
Dyn. Prog.	8	**129,320**[1]	18,005	13,000[1]
PRM	4	**130,418**[1]	0,213	15,600[1]
RRT	10	**140,380**	2,693	69
Hybrid A*	10	**128,555**	0,414	128

[1] valores médios

A Figura 7.25 apresenta o gráfico dos menores comprimentos de trajetória conforme Tabela 7-8. O Algoritmo de Dijkstra foi, novamente, o que apresentou o menor comprimento médio de trajetória. Ou seja, tanto para um mapa com caminho mais longo quanto para um com vias mais restritas, como o Labirinto Estreito, o planejador foi o que teve melhor desempenho quanto ao comprimento. A desvantagem do planejador continua sendo seu tempo de processamento superior aos demais, frente ao fato de que proporcionalmente, as diferenças entre os comprimentos de trajetórias são menores do que entre os outros dois parâmetros avaliados, o tempo de processamento e a quantidade de passos.

As diferenças entre os comprimentos para o mapa Ziguezague são relativamente pequenas. Dentre as menores trajetórias, a maior delas foi a do RRT, com apenas 17,266 unidades de comprimento a mais que a trajetória do Algoritmo de Dijkstra.

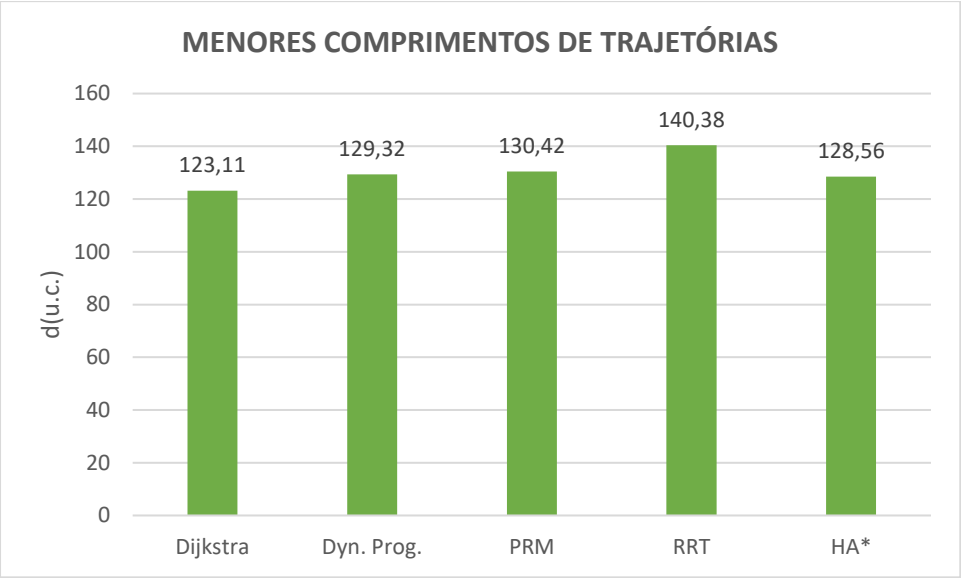


Figura 0.25 - Menores comprimentos de trajetórias das simulações no Ziguezague.

A seguir são apresentadas as conclusões desse trabalho.

8
Conclusões e Trabalhos Futuros

Neste trabalho, foram realizados estudos comparativos entre cinco planejadores de trajetórias - Algoritmo de Dijkstra, *Dynamic Programming*, PRM, RRT e *Hybrid A** - por meio do ambiente MATLAB. O objetivo foi avaliar o desempenho desses planejadores em três critérios: tempo de processamento, quantidade de passos e comprimento da trajetória. Para isso, foram criados três mapas distintos: um chamado de Labirinto, que é desordenado e apresenta várias opções de caminhos; outro foi o Labirinto Estreito, que possui a mesma topologia do Labirinto, mas com suas paredes infladas; e o terceiro foi chamado de Ziguezague, pois apresentava um circuito longo na forma de ziguezague.

Cada mapa avaliou aspectos diferentes dos planejadores. O mapa Labirinto avaliou suas capacidades em gerar trajetórias em um ambiente desorganizado e com múltiplas opções de caminhos. Um dos objetivos era verificar se cada planejador demonstraria maior tendência em ir por um caminho do que por outro. Já o propósito do mapa Labirinto Estreito foi avaliar a capacidade de se gerar trajetórias em um ambiente de caminhos estreitos, cujo movimento do robô é limitado. O intuito do mapa Ziguezague foi verificar o desempenho dos planejadores em um caminho longo, assim como a capacidade de fazer curvas fechadas.

Os planejadores PRM, RRT e *Hybrid A** são parte das bibliotecas e *toolboxes* do MATLAB, enquanto que o Algoritmo de Dijkstra e *Dynamic Programming* foram desenvolvidos por um terceiro, porém disponibilizados oficialmente pela *Math Works*®, a desenvolvedora do MATLAB.

Dentre os parâmetros avaliados, o tempo de processamento e o comprimento da trajetória são os mais intuitivamente compreendidos, pois estão diretamente conectados ao desempenho do planejador. No caso do tempo de processamento, é uma medida de sua complexidade computacional. A quantidade de passos é mais relevante para robôs que se locomovem em linha reta e que necessitam interromper seu movimento e girar seu corpo para poder mudar de direção. Se essa interrupção de movimento não for suave, ela pode ocasionar esforços nos componentes mecânicos do robô. E ainda, a cada vez que o motor para e é acionado novamente há um consumo maior de energia, que pode ser significativo, dependendo do peso do robô ou da carga que ele carrega. Há também os robôs com rodas que executam

movimento de giro derrapando algumas rodas. Quando se trabalha com robôs que possuem estas características, é interessante conhecer seu comportamento em relação aos passos para questões relacionadas a manutenção, torque no eixo do motor e desgaste de peças como, por exemplo, borracha das rodas.

A comparação de desempenho entre os planejadores foi feita da seguinte forma: para cada mapa foram selecionadas as séries de simulações que apresentaram os melhores desempenhos para cada um dos três parâmetros selecionados, que foram então comparados entre si.

De modo geral, pode-se dizer que em relação ao tempo de processamento, os planejadores Algoritmo de Dijkstra e *Dynamic Programming* apresentaram desempenhos muito inferiores aos demais. Isso se deve ao fato desses planejadores buscarem conexões com o maior número de nós possível, independentemente de restrições de distância entre nós, como ocorre com o PRM, por exemplo e, a partir dessas conexões, estabelecer qual a menor trajetória possível. Neste quesito, o PRM apresentou os melhores resultados em dois mapas, Labirinto Estreito e Ziguezague. Mas, mesmo assim, no mapa Labirinto, seu tempo foi apenas 7ms a mais que o primeiro lugar, o RRT. O bom desempenho do PRM neste caso se deve ao processo de construção do mapa de rotas ser direcionado pela limitação da distância entre os nós estabelecida pelo delta, o que otimiza o tempo de processamento. Outro fator importante para o planejador utilizado é o fato da simulação se iniciar com 50 nós e nos casos de necessidade, incrementar este número em mais 10 nós. O RRT também apresentou um bom desempenho em relação ao tempo de processamento, sobretudo nos mapas que apresentaram restrições, Labirinto e Labirinto Estreito, também devido à expansão de sua árvore ser direcionada buscando a região de objetivo e limitada pela distância imposta pelo delta. Especificamente para o mapa Labirinto, o delta de 2 unidades de comprimento se mostrou a ideal, promovendo o melhor tempo médio para o RRT. Tal desempenho, entretanto, não se repetiu no mapa Ziguezague pois a necessidade de se expandir a árvore a partir da raiz gerou uma quantidade muito grande de nós e iterações e, consequentemente, acarretando em um maior tempo de processamento. Já o planejador *Hybrid A** apresentou algo interessante: os melhores tempos de processamento de cada mapa são bastante próximos, apesar das configurações de cada simulação serem distintos. O algoritmo A* no qual o planejador se baseia, leva em consideração o nó de objetivo e

discretiza o ambiente para fazer a expansão direcionada, contribuindo para melhoria do tempo de processamento.

Em relação ao comprimento das trajetórias, o Algoritmo de Dijkstra obteve os melhores resultados nos mapas Labirinto Estreito e Ziguezague. No mapa Labirinto, ficou muito próximo ao comprimento do RRT, que foi o que obteve o menor comprimento. O Algoritmo de Dijkstra é realmente voltado à busca da trajetória ótima, pois, conforme explicitado, ele procura interligar o maior número de nós, de todas as formas possíveis e analisar cada conexão a fim de encontrar tal trajetória. A melhor trajetória do RRT no mapa labirinto ocorreu na mesma série com que o planejador apresentou o melhor tempo de processamento, reforçando a tese de que o delta de 2 unidades de comprimento é a ideal para este mapa. Deve-se levar em conta também que os comprimentos do Algoritmo de Dijkstra utilizados para fins de comparação são valores médios de dez simulações de uma série. Enquanto que os valores de comprimento do RRT utilizados nas comparações correspondem ao comprimento de uma simulação, pois, o planejador repete sempre a mesma trajetória se os parâmetros não forem alterados. Olhando para cada simulação individualmente, o Algoritmo de Dijkstra apresentou a menor trajetória de todas, na décima simulação com 140 nós, com o comprimento de 36,667 unidades de comprimento.

O planejador *Dynamic Programming* obteve os melhores resultados no critério das quantidades de passos nos três mapas. Ainda sobre este parâmetro, observando as trajetórias geradas, conclui-se que não faz sentido avaliar o planejador *Hybrid A** segundo este critério pois suas trajetórias são curvilíneas, não acarretando em mudanças bruscas de direção que justifiquem paradas ou derrapagens.

A Tabela 8-1 apresenta os melhores resultados de cada planejador para cada um dos mapas, lembrando que *tp* é o tempo de processamento e *d* é o comprimento das trajetórias.

Supondo então outros tipos de mapas e situações, como por exemplo uma casa, com móveis dispostos pelos cômodos. Uma boa opção para planejamento global de trajetórias para um robô doméstico seria o RRT, pois foi o que apresentou os melhores desempenhos quanto a tempo de processamento e comprimento de trajetórias no mapa Labirinto, que possui um certo espaço para manobras, mas também muitos obstáculos. Em um outro ambiente, como o de uma indústria, no

qual os espaços para circulação são geralmente restritos, o PRM se apresenta como uma boa opção pois, conforme visto nas simulações no mapa Labirinto Estreito, seu tempo de processamento foi muito inferior aos demais, o que em muito compensa a trajetória mais longa gerada por ele, na média. Outros ambientes nos quais a aplicação da robótica móvel é cada vez maior, como os galpões logísticos e os hospitais, levando em consideração que os corredores são geralmente longos, o PRM também se mostra uma opção interessante devido a seu desempenho no mapa Ziguezague, novamente com um tempo de processamento muito menos que os demais, compensando o fato de não apresentar a menor trajetória. Todavia, a escolha de outros planejadores também se justifica para situações específicas. Por exemplo, para o caso do planejamento de trajetórias de um automóvel autônomo, a escolha do *Hybrid A** se aplica, pois, suas trajetórias são específicas para atenderem a veículos não-holonômicos. Ou no caso de um comprimento da trajetória menor ser algo imprescindível, pode-se optar pelo Algoritmo de Dijkstra, pois é certo que este apresentará uma trajetória ótima. Se o requerido fora mínima quantidade de passos, seja para evitar paradas ou desgaste de rodas ou qualquer outro motivo, aí então a melhor opção é o *Dynamic Programming*, cujo desempenho neste quesito foi o melhor para todos os mapas.

Para trabalhos futuros sugere-se implementar os mesmos planejadores em uma simulação que emule um robô real, verificando os mesmos parâmetros e outros adicionais, como, por exemplo, a suavidade da trajetória. Para isto, pode-se utilizar a plataforma Gazebo, que é voltada para este fim. Ela trabalha com modelos de robôs em formato URDF (*Unified Robot Description Format*), que permite representar as restrições cinemáticas e dinâmicas de um robô, assim como sua descrição física tanto visual quanto de contato [88]. A plataforma também é compatível com ROS (*Robot Operational System*), um *framework* computacional de fonte aberta com a qual é possível criar aplicações para robótica através de ferramentas integradas [88]. Os planejadores podem ser implementados na *stack* de navegação do ROS através das linguagens de programação *Python* e C++.

Posteriormente, pode-se utilizar as mesmas plataformas desenvolvidas pera a simulação em um robô real através do ROS e submetê-lo a testes. Basta que o controlador do robô seja compatível com o sistema operacional *Linux*. Por fim, comparar os resultados das simulações com os experimentos.

TABELA 0-1 – MELHORES RESULTADOS DE CADA PLANEJADOR

	Labirinto			*Labirinto Estreito*			*Ziguezague*		
Planejador	$\overline{tp}(s)$	*passos*	*d(u.c.)*	$\overline{tp}(s)$	*passos*	*d(u.c.)*	$\overline{tp}(s)$	*passos*	*d(u.c.)*
Dijkstra	3,721	8,20[1]	39,384[1]	20,352	12,600[1]	44,116[1]	6,353	12,778[1]	123,114[1]
Dyn. Prog.	3,075	7,70[1]	43,206[1]	13,581	11,167[1]	48,251[1]	4,934	12,600[1]	129,320[1]
PRM	0,057	10,20[1]	44,302[1]	0,072	14,500[1]	50,556[1]	0,055	15,600[1]	130,418[1]
RRT	0,050	13,00	39,267	0,288	22,000	47,044	2,407	69,00	140,380
Hybrid A*	0,474	27,00	43,674	0,413	26,000	44,825	0,414	67,00	128,555

[1] valores médios

Bibliografia

1 NIKU, S. B**. Introdução à Robótica**: Análise, Controle, Aplicações. 2ª ed. Rio de Janeiro-RJ, LTC – Livros Técnicos e Científicos, 2013.

2 INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 10218-1** – Robots and Robotic Devices – Safety Requirements for industrial robots – Part 1: Robots.

3 ASIMOV, I. **Eu, Robô**. 1ª ed. São Paulo, Aleph, 2014.

4 ORTIGOZA, R. S.; ARANDA, M. M.; ORTIGOZA, G. S.; GUZMÁN, V. M. H.; VILCHIS, M. A. M.; GONZÁLEZ, G. S.; LOZADA, J. C. H.; CARBAJAL, M. O. **Wheeled Mobile Robots: A Review**. In: IEEE Latin America Transactions, 2012, v. 10, n. 6, p. 2209-2217.

5 LATOMBE, J.C. **Robot Motion Planning**. 2. ed. New York: Springer Science + Business Media. 1991.

6 RUBIO, F.; VALERO, F.; LLOPIS-ALBERT, C. **A review of mobile robots:** Concepts, methods, theoretical framework, and applications**.** International Journal of Advanced Robotic Systems. Sage Pub. March-April 2019, p. 1–22.

7 LAVALLE, S. M. **Planning Algorithms**. Cambridge University Press. Available for downloading at http://planning.cs.uiuc.edu/. 2006.

8 SIEGWART, R., NOURBAKHSH, I. R. **Introduction to Autonomous Mobile Robots.** 1. ed. Cambridge: MIT Press, 2004.

9 PATLE, B.K.; BABU, G.; PANDEY, A.; PARHI, D.R.K.; JAGADEESH, A. In: **A review: On path planning strategies for navigation of mobile robot**. Elsevier. Defense Technology, 2019, n.15, p. 582 - 606.

10 MOHD, N.C.; MOHANTAB, J.C. **Methodology for Path Planning and Optimization of Mobile Robots: A Review**. In: International Conference on Robotics and Smart Manufacturing, 2018**.** Procedia Computer Science, 2018, n. 133, p. 141–152.

11 ZHANG, H.; LIN, W.; CHEN, A. **Path Planning for the Mobile Robot: A Review**. Symmetry, 2018, n. 10, p. 450.

12 ASANO, T; ASANO, T.; GUIBAS, l.; HERSHBERGER, j; IMAI, H. **Visibility-Polygon Search and Euclidean Shortest Paths**. In: 26TH ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER

SCIENCE – SFCS ´85, 1985. **Proceedings of the 26th Annual Symposium on Foundations of Computer Science.** IEEE**,** 1985, pp. 155-164.

13 CANNY, J.F.; DONALD, B.R. **Simplified Voronoi Diagrams**. In: THIRD ANNUAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY – SCG´87, 1987, Waterloo, ON, Canada. **Proceedings of the third annual symposium on Computational Geometry**. 1987, pp. 153-161.

14 LIU, Y.; ARIMOTO, S**. Proposal of Tangent Graph and Extended Tangent Graph for Path Planning of Mobile Robots**. In: 1991 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION – ICRA'91, 9–11 April 1991, Sacramento, CA, USA. **Proceedings of the 1991 IEEE International Conference on Robotics and Automation - ICRA'91**. 1991, pp. 312–317.

15 HABIB, M.K.; ASAMA, H. **Efficient Method to Generate Collision Free Paths For an Autonomous Mobile Robot Based on New Free Space Structuring Approach**. In: IEEE/RSJ International Workshop on Intelligent Robots and Systems - IROS'91, 3–5 November 1991, Osaka, Japan. **Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems - IROS'91**. 1991, pp. 563–567.

16 BROOKS, R. A.; LOZANO-PEREZ, T. **A Subdivision Algorithm in Configuration Space for Findpath with Rotation**. In: IEEE Transactions on Systems, Man and Cybernetics. 1985, v.15, n. 2, pp. 224-233.

17 KAVRAKI, L.E.; SVESTKA, P.; LATOMBE, J.; OVERMARS, M.H. **Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces**. In: IEEE Transactions in Robotics and Automation. 1996, v.12, n. 4, pp. 566–580.

18 LAVALLE, S. M. **Rapidly-exploring Random Trees: A New Tool for Path Planning**. Department of Computer Science. Iowa State University. Technical Report – TR'98. 1998, n.11.

19 MOHAMED, E.; MILAN, S. **Sampling-Based Robot Motion Planning: A Review**. In: IEEE Access, 2014, v.2, pp.56–77.

20 DIJKSTRA, E.W. **A Note on Two Problems in Connection with Graphs**. Numerische Mathematik. 1959, 4, pp. 269–271.

21 HART, P.E.; NILSSON, N.J.; RAPHAEL, B. **A Formal Basis for the Heuristic Determination of Minimum Cost Paths**. In: IEEE Transactions of Systems Science and Cybernetics. **1968**, v. 4, n. 2, pp. 100–107.

22 STENTZ, A. **Optimal and Efficient Path Planning for Partially-Known Environments**. In: 1994 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION – ICRA'94, 8–13 May 1994, San Diego, CA, USA. **Proceedings of the 1994 IEEE International Conference on Robotics and Automation – ICRA'94**. 1994. pp. 3310-3317.

23 FERGUSON, D.; STENTZ, A. **Using Interpolation to Improve Path Planning: The Field D* Algorithm**. In: Journal of Field Robotics. Willey InterScience. 2006, 23, v. 2, pp. 79–101.

24 DANIEL, K.; NASH, A.; KOENIG, S.; FELNER, A. **Theta*: Any-Angle Path Planning on Grids**. Journal of Artificial Intelligence Research. 2010, 39, pp. 533–579.

25 SUH, S.H.; SHIN, K. G. **A Variational Dynamic Programming Approach to Robot Path Planning With a Distance-Safety Criterion**. Robot Systems Division Technical Report. College of Engineering. The University of Michigan. 1987, n. 11.

26 KHATIB, O. **Real-Time Obstacle Avoidance for Manipulators and Mobile Robots**. IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION – ICRA'85, 1985, St. Louis, MO, USA. **Proceedings of the IEEE International Conference on Robotics and Automation – ICRA'85**. 1985, pp. 500-505.

27 WHITBROOK, A.M.; AICKELIN, U.; GARIBALDI, J.M. **Idiotypic Immune Networks in Mobile-Robot Control**. In: IEEE Transactions in Systems, Man and Cybernetics - Part B: Cybernetics. 2007, v. 37, n. 6, pp. 1581–1598.

28 HUQ, R.; MANN, G.K.I.; GOSINE, R.G. **Mobile Robot Navigation Using Motor Schema and Fuzzy Context Dependent Behavior Modulation**. In: Applied Soft Computing Journal. Elsevier. 2008, v. 8, pp. 422–436.

29 FERNANDEZ-LEON, J.A.; ACOSTA, G.G.; MAYOSKY, M.A. **Behavioral Control through Evolutionary Neurocontrollers for**

Autonomous Mobile Robot Navigation. In: Robotics and Autonomous Systems. Elsevier. 2009, v. 57, pp. 411–419.

30 SHI, C.X.; WANG, Y.Q.; YANG, J.Y. **A Local Obstacle Avoidance Method for Mobile Robots in Partially Known Environment**. In: Robotics and Autonomous Systems. Elsevier. 2010, v. 58, pp. 425–434.

31 TOIBERO, J.M.; ROBERTI, F.; CARELLI, R.; FIORINI, P. **Switching Control Approach for Stable Navigation of Mobile Robots in Unknown Environments**. In: Robotics and Computer Integrated Manufacturing. Elsevier. 2011, v. 27, pp. 558-568.

32 MAREFAT, M.; BRITANIK, J. **Case-Based Process Planning Using an Object-Oriented Model Representation**. In: Robotics and Computer Integrated Manufacturing. Elsevier Pergamon. 1997, v. 13, n. 3, pp. 229-251.

33 WENG, M.; WEI, X.; QU, R.; CAI, Z. **A Path Planning Algorithm Based on Typical Case Reasoning**. In: Geo-Spatial Information Science. Taylor&Francis. 2008, v. 12. pp. 66-71.

34 FOX, D.; BURGARD, W.; THRUN, S. **The Dynamic Window Approach to Collision Avoidance**. In: IEEE Robotics & Automation Magazine. 1997, v. 4, n. 1, pp. 23-33.

35 CHOU, C.; LIAN, F.; WANG, C. **Characterizing Indoor Environment for Robot Navigation Using Velocity Space Approach with Region Analysis and Look-Ahead Verification**. In: IEEE Transactions on Instrumentation and Measurement. 2011, v. 60, n. 2, pp. 442-451.

36 MARTIN, P.; DEL POBIL, A.P. **Application of Artificial Neural Networks to the Robot Path Planning Problem**. In: NINTH INTERNATIONAL CONFERENCE ON APPLICATIONS OF ARTIFICIAL INTELLIGENCE IN ENGINEERING – ICAAIE'94, Pennsylvania, PA, USA. 19–21 July 1994. **Proceedings of the Ninth International Conference on Applications of Artificial Intelligence in Engineering – ICAAIE'94**. 1994, pp. 73–80.

37 MULDER, E.; MASTEBROEK, H.A.K. **Construction of an Interactive and Competitive Artificial Neural Network for the Solution of Path Planning Problems**. In: 6TH EUROPEAN SYMPOSIUM ON

ARTIFICIAL NEURAL NETWORKS – ESANN'98, Bruges, Belgium, 22–24 April 1998. **Proceedings of the 6th European Symposium on Artificial Neural Networks – ESANN'98**. 1998, pp. 407–412.

38 LI, C.H.; ZHANG, J.Y.; LI, Y.B. **Application of Artificial Neural Network Based On Q-Learning for Mobile Robot Path Planning**. In: 2006 IEEE INTERNATIONAL CONFERENCE ON INFORMATION ACQUISITION – ICIA'06. Weihai, China, 20–23 August 2006. **Proceedings of the 2006 IEEE International Conference on Information Acquisition**. 2006, pp.978–982.

39 GONZALEZ, A.F.C.; VEGA, J.I.H.; SANTOS, C.H.; GORHAM, D.G.P. **A Method to Verify a Path Planning by a Back-Propagation Artificial Neural Network**. In: 10TH LATIN AMERICAN WORKSHOP ON LOGIC/LANGUAGES, ALGORITHMS AND NEW METHODS OF REASONING. Puebla, Mexico, 15 August 2016. **Proceedings of the 10th Latin American Workshop on Logic/Languages, Algorithms and New Methods of Reasoning**. Puebla, Mexico, 15 August 2016; pp. 98–105.

40 HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Ann Arbor, MI: University of Michigan Press. 1975.

41 ZHAO, M.; ANSARI, N.; HOU, E.S.H. **Mobile Manipulator Path Planning by a Genetic Algorithm**. In: 1992 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS – ICIRS'92. Raleigh, NC, USA, July 7-10, 1992. **Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems – ICIRS'92**. 1992, pp. 681-688.

42 PEHLIVANOGLU, Y.V.; BAYSAL, O.; HACIOGLU, A. **Path Planning for Autonomous UAV via Vibrational Genetic Algorithm**. In: Aircraft Engineering and Aerospace Technology: An International Journal. Emerald Group Publishing. 2007, 79, pp. 352–359.

43 TSAI, C.; HUANG, H.; CHAN, C. **Parallel Elite Genetic Algorithm and Its Application to Global Path Planning for Autonomous Robot Navigation**. In: IEEE Transactions on Industrial Electronics. IEEE. 2011, v. 58, n.10, pp. 4813–4821.

44 TUNCER, A.; YILDIRIM, M. **Dynamic Path Planning Of Mobile Robots with Improved Genetic Algorithm**. In: Computers and Electrical Engineering. Elsevier. 2012, v. 38, n. 6, pp.1564–1572.

45 QU, H.; XING, K.; ALEXANDER, T. **An Improved Genetic Algorithm with Co-Evolutionary Strategy for Global Path Planning of Multiple Mobile Robots**. In: Neurocomputing. Elsevier. 2013, v. 120, pp. 509–517.

46 LIU, F.; LIANG, S.; XIAN, X.D. **Optimal Path Planning for Mobile Robot Using Tailored Genetic Algorithm**. In: IAES TELKOMNIKA Indonesian Journal of Electrical Engineering. 2014, v. 12, n. 1, pp. 1–9.

47 SHORAKAEI, H.; VAHDANI, M.; IMANI, B.; GHOLAMI, A. **Optimal Cooperative Path Planning Of Unmanned Aerial Vehicles by a Parallel Genetic Algorithm**. In: Robotica. Cambridge University Press. 2016, v. 34, pp. 823–836.

48 DORIGO, M. **Optimization, Learning and Natural Algorithms**. Ph.D Tesis. Politecnico di Milano. 1992.

49 WEN, Z.Q.; CAI, Z.X. **Global Path Planning Approach Based on Ant Colony Optimization Algorithm**. In: Journal of Central South University of Technology. Springer. 2006, 13, pp. 707–712.

50 LIU, S.; MAO, L.; YU, J. **Path Planning Based on Ant Colony Algorithm and Distributed Local Navigation for Multi-Robot Systems**. In: 2006 IEEE INTERNATIONAL CONFERENCE OF MECHATRONICS AND AUTOMATION - ICMA'06. Luoyang, China. 2006. **Proceedings of the 2006 IEEE International Conference of Mechatronics and Automation**. 2006, pp. 1733-1738.

51 TAN, G. Z.; HUAN, H.; SLOMAN, A. **Ant Colony System Algorithm for Real Time Globally Optimal Path Planning of Mobile Robots**. In: Acta Automatica Sinica. Elsevier. 2007, v. 33, n.3, pp. 279-285.

52 WANG, H.J.; XIONG, W. **Research on Global Path Planning Based on Ant Colony Optimization for AUV**. In: Journal of Marine Science and Application. **2009**, 8, pp. 58–64.

53 ZHAO, J.; FU, X. **Improved Ant Colony Optimization Algorithm and its Application on Path Planning of Mobile Robot**. In: Journal of Computers. Academy Publisher. 2012, v. 7, n. 8, pp. 2055–2062.

54 YOU, X.; LIU, K.; LIU, S. **A Chaotic Ant Colony System for Path Planning of Mobile Robot**. In: International Journal of Hybrid Information Technology. 2016, v. 9, n. 1, pp. 329-338.

55 EBERHART, R.; KENNEDY, J. **A New Optimizer Using Particle Swarm Theory**. In: SIXTH INTERNATIONAL SYMPOSIUM ON MICRO MACHINE AND HUMAN SCIENCE - MHS'95. 1995, Nagoya, Japan. **Proceedings of the Sixth International Symposium on Micro Machine and Human Science - MHS'95**. 1995, pp. 39-43.

56 ATYABI, A.; PHON-AMNUAISUK, S.; HO, C.K. **Applying Area Extension PSO in Robotic Swarm**. In: Journal of Intelligent and Robotic Systems. 2010, v. 58, pp. 253–285.

57 GONG, D.W.; ZHANG, J.H.; ZHANG, Y. **Multi-objective Particle Swarm Optimization for Robot Path Planning in Environment with Danger Sources**. In: Journal of Computers. Academy Publisher. 2011, v. 6, n. 8, pp. 1554-1561.

58 ZHANG, Y.D.; WU, L.N.; WANG, S.H. **UCAV Path Planning by Fitness-scaling Adaptive Chaotic Particle Swarm Optimization**. In: Mathematical Problems in Engineering. Hindawi. 2013, v. 2013.

59 TANG X, LI L, JIANG B. **Mobile robot SLAM method based on multi-agent particle swarm optimized particle filter**. In: The Journal of China Universities of Posts and Telecommunications. 2014, v. 21, n. 6, pp. 78-86.

60 LIU, Y.K.; LI, M.K.; XIE, C.L.; PENG, M.; XIE, F. **Path-Planning Research in Radioactive Environment Based on Particle Swarm Algorithm**. In: Progress in Nuclear Energy. Elsevier. 2014, v. 74, pp. 184–192.

61 KIRKPATRICK, S.; GELATT, C.D.; VECCHI, M.P. **Optimization by Simulated Annealing.** In: Decision Science. 1983, v. 220, n. 4598, pp. 498-516.

62 ALMEIDA, B.S.; PINTO, P.E.D; SORIANO, R.R.A. **A Técnica "*Simulated Annealing*" Aplicada aos Problemas de Percurso do Cavalo e Damas Pacíficas**. In: Cadernos do IME: Série Informática. UERJ. 2005, v. 19.

63 MARTINEZ-ALFARO, H.; FLUGRAD, D.R. **Collision-Free Path Planning for Mobile Robots and/or AGVs Using Simulated Annealing**. In: 1994 IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS. 1994, San Antonio, TX, USA. **Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics**. 1994, pp. 270-275.

64 VOUGIOUKAS, S.; IPPOLITO, M.; CUGINI, U. **Path Planning Based on Accelerated Simulated Annealing**. In: 1996 RESEARCH WORKSHOP ON ERNET—EUROPEAN ROBOTICS NETWORK. 1996, Darmstadt, Germany. **Proceedings of the 1996 Research Workshop on ERNET—European Robotics Network**. 1996, pp. 259–268.

65 MIAO, H.; TIAN, Y. **Robot Path Planning in Dynamic Environments Using a Simulated Annealing Based Approach**. In: 10TH INTERNATIONAL CONFERENCE ON CONTROL, AUTOMATION, ROBOTICS AND VISION. 2008, Hanoi, Vietnam. **Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision**. 2008, pp. 1253-1258.

66 CHOU, M.C. **Path-Planning For an Autonomous Robot Using a Simulating Annealing**. In: Journal of Information and Optimization Science. 2011, v. 32, pp. 297–314.

67 MIAO, H.; TIAN, Y.C. **Dynamic Robot Path Planning Using an Enhanced Simulated Annealing Approach**. In: Applied Mathematics and Computation. 2013, v. 222, pp. 420–437.

68 BEHNCK, L.P.; DOERING, D.; PEREIRA, C.E.; RETTBERG, A**. A Modified Simulated Annealing Algorithm for SUAVs Path Planning**. In: IFAC-Papers on Line. Elsevier. 2015, v. 48, pp. 63–68.

69 SATHYARAJ, B.M; JAIN, L.C; FINN, A.; DRAKE. S. **Multiple UAVs Path Planning Algorithms: A Comparative Study**. In: Fuzzy Optimization and Decision Making. Springer Science + Business. 2008, v. 7, pp. 257–267.

70 GLABOWSKI, M.; MUSZNICKI, B.; NOWAK, P.; ZWIERZYKOWSKI, P. **Efficiency Evaluation of Shortest Path Algorithms**. In: THE NINTH ADVANCED INTERNATIONAL CONFERENCE ON

TELECOMMUNICATIONS – AICT'13. 2013, Rome, Italy. **Proceedings of the Ninth Advanced International Conference on Telecommunications – AICT'13**. 2013. pp. 154-160.

71 ZAHEER, S.; JAYARAJU, M.; GULREZ, T. **Performance analysis of path planning techniques for autonomous mobile robots**. In: 2015 IEEE INTERNATIONAL CONFERENCE ON ELECTRICAL, COMPUTER AND COMMUNICATION TECHNOLOGIES – ICECCT'15. pp. 1-5. DOI: 1-5. 10.1109/ICECCT.2015.7226205.

72 KHANMIRZA, E.; HAGHBEIGI, M.; NAZARAHARI, M.; DOOSTIE, S. **A Comparative Study of Deterministic and Probabilistic Mobile Robot Path Planning Algorithms**. In: 5TH RSI INTERNATIONAL CONFERENCE ON ROBOTICS AND MECHATRONICS – ICROM'17. 2017, Tehran, Iran . **Proceedings of 5th RSI International Conference on Robotics and Mechatronics – ICRoM'17**. 2017. pp. 534-539, doi: 10.1109/ICRoM.2017.8466197.

73 http://www.kavrakilab.org/publications.html. Acesso em 05/07/2020.

74 LYNCH, K.M.; PARK, F.C. **Modern Robotics. Mechanics, Planning, and Control**. 1st ed. Cambridge University Press. 2017.

75 D'Andrea, R. **Dynamic Programming and Optimal Control**. Class Notes ETH Zurich. Disponível em: http://www.idsc.ethz.ch/education/lectures/optimal-control.html.

76 PETEREIT, J.; EMTER, T.; FREY, C. W.; KOPFSTEDT, T.; BEUTEL, A. **Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments**. 7th German Conference on Robotics - *ROBOTIK 2012*. 2012, pp. 1-6.

77 MAGRAB, E.B.; AZARM, S.; BALACHANDRAN, B.; DUNCAN, J.H.; HEROLD, K.E.; WALSH, G.C. **An Engineer's Guide to MATLAB. With Applications from Mechanical, Aerospece, Electrical, Civil, and Biological Systems Engineering.** 3th ed. Prentice Hall. 2011.

78 CORKE, PETER. **Robotics, Vision and Control. Fundamental Algorithms in MATLAB**. 2th ed. Springer. 2017.

79 https://www.mathworks.com/help/nav/index.html?s_tid=CRUX_lftnav. Acesso em 18/02/2020.

80 BALCILAR, M. **RobotPathPlanning**. https://github.com/muhammetbalcilar/RobotPathPlanning. Acesso em 18/02/2020

81 https://www.mathworks.com/help/robotics/ref/mobilerobotprm.html. Acesso em 18/02/2020.

82 https://www.mathworks.com/help/nav/ref/plannerrrt.html#mw_0edfa51a-6e3d-4e8e-8dff-f3e320402d86. Acesso em 18/02/2020.

83 https://www.mathworks.com/help/nav/ref/plannerhybridastar.html?s_tid=srchtitle. Acesso em 18/02/2020.

84 REEDS, J. A.; SHEPP, L.A. **Optimal Paths for a Car that Goes Both Forwards and Backwards**. In: Pacific Journal of Mathematics. v. 145, n. 2, pp. 367 – 393, 1990.

85 SOARES, J.C.V. **Graph Optimization and Probabilistic SLAM of Mobile Robots using an RGB-D Sensor.** 2018. 109 f. Dissertação de Mestrado – Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2018.

86 PETTIE, S., RAMACHANDRAN, V. **An Optimal Minimum Spanning Tree Algorithm**. In: Journal of the ACM. 2002, v.49, n.1, pp. 16-34.

87 TUNCER, A., MEHMET, Y. **Dynamic Path Planning of Mobile Robots with Improved Genetic Algorithm**. In: Computers & Electrical Engineering. 2002, v. 38, n. 6, pp. 1564-1572.

88 JOSEPH, L. **Learning Robotics Using Python**. Design, Simulate, Program, and Prototype an Autonomous Mobile Robot using ROS, OpenCV, PCL and Python. 2nd ed. Birmingham-UK, Packt Publishing, 2018.