UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA - PPGI

Uma heurística simplificada para funções custo de planejadores da família A*

JEFFERSON BARBOSA BELO DA SILVA

João Pessoa, Paraíba, Brasil
Agosto, 2015

JEFFERSON BARBOSA BELO DA SILVA

Uma heurística simplificada para funções custo de planejadores da família A*

Dissertação de Mestrado apresentada à Coordenação do Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba, como requisito parcial para obtenção do grau de Mestre em Informática

Área de concentração: Sistemas de computação
Linha de pesquisa: Computação distribuída
Orientador: Prof. Dr. Clauirton de A. Siebra
Co-orientador: Prof. Dr. Tiago P. Nascimento

João Pessoa, Paraíba, Brasil
Agosto, 2015

S586u Silva, Jefferson Barbosa Belo da.
Uma heurística simplificada para funções custo de planejadores da família A* / Jefferson Barbosa Belo da Silva.- João Pessoa, 2015.
81f. : il.
Orientador: Clauirton de A. Siebra
Coorientador: Tiago P. Nascimento
Dissertação (Mestrado) - UFPB/CI
1. Informática. 2. Computação distribuída. 3. Função custo. 4. Planejamento de caminho. 5. Geração de trajetória.

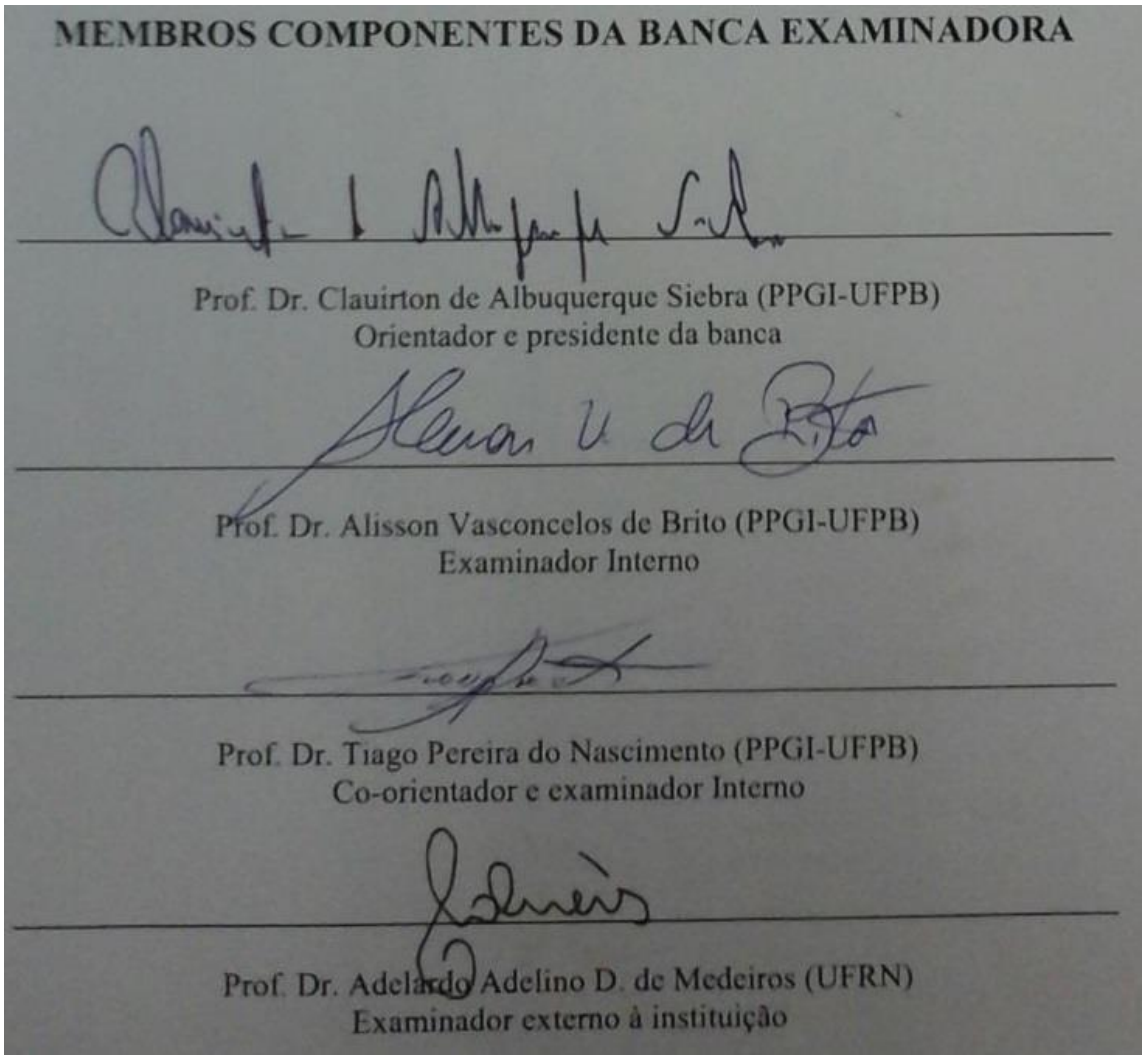
UFPB/BC CDU: 004(043)

JEFFERSON BARBOSA BELO DA SILVA

UMA HEURÍSTICA SIMPLIFICADA PARA FUNÇÕES CUSTO DE PLANEJADORES DA FAMÍLIA A*

Dissertação de Mestrado apresentada à Coordenação do Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como requisito parcial para obtenção do grau de Mestre em Informática

Data da aprovação: 21/08/2015



Dedico esse trabalho à minha esposa, Márcia Belo, pelo seu amor e por sua expressiva devoção, por compreender a minha ausência durante horas de estudos e leituras, mesmo estando no mesmo cômodo, e, sobretudo, por sempre acreditar em mim, mais do que eu mesmo. Desde o processo de seleção até o término dessa dissertação, seu encorajamento foi determinante para que eu acreditasse ser possível obter êxito nessa pós-graduação. Sem você, meu amor, nenhuma das páginas seguintes estariam escritas. Eu te amo.

"Descobrir consiste em olhar para o que todo mundo está vendo e pensar uma coisa diferente".

(Roger Von Oech)

Agradecimentos

- Primeiramente, devoto meus agradecimentos a Deus por ter me proporcionado saúde, força, inteligência e paciência para conseguir concluir mais essa etapa em minha vida, a Ele dedico meus maiores feitos.

- À minha esposa Márcia Belo, pela sua compreensão, amor e incentivo durante a realização dessa caminhada.

- À toda minha família, em especial, a minha mãe Terezinha, por estar sempre me incentivando a continuar e nunca desistir de um sonho. Obrigado por tudo.

- Ao Prof. Dr. Clauirton de Albuquerque Siebra, por ter aceitado minha proposta de trabalho e pela excepcional orientação ao longo do mesmo;

- Ao Prof. Dr. Tiago Pereira do Nascimento, pela sugestão da proposta de trabalho e pela brilhante co-orientação no decorrer deste;

- Aos amigos Vinicius de Andrade e Ewerton Lopes que direta e indiretamente me ajudaram com contribuições teóricas, sugestões e algumas gargalhadas;

- À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pela concessão da bolsa durante todo o período de mestrado;

- E não menos importante, a todos os ausentes, presentes, colegas, amigos que direta ou indiretamente me ajudaram ou influenciaram nos caminhos a serem tomados nessa jornada.

Resumo

Uma das principais questões relacionados ao tema da robótica móvel é descobrir a maneira mais eficiente para realizar a navegação, de um ponto a outro no ambiente, com máxima segurança e despendendo a menor quantidade de tempo e de recursos computacionais possível. À vista disso, o presente trabalho se motiva a desenvolver uma melhoria heurística que possa ser aplicável às funções custo dos principais algoritmos baseados na família A* e que propõe utilizar, de forma mais eficiente, os recursos computacionais disponíveis, aperfeiçoando assim, os resultados obtidos através dos principais algoritmos de buscas aplicados à robótica móvel. A mesma tem o objetivo de minimizar a quantidade de colisões, a duração do trajeto, bem como o tempo de processamento através da minimização da importância da variável *g(n)* - responsável em armazenar informações subutilizadas do passado dos algoritmos. Para visualizar os efeitos dessa modificação, um levantamento das melhores estratégias de busca em ambiente estático e dinâmico foi realizado e, através deste, foram analisados os quatro melhores e mais atuais algoritmos destacados pela literatura técnica especializada. Algumas comparações foram efetuadas considerando ambientes estáticos e altamente dinâmico com diferentes direções de busca e parâmetros que visavam mensurar a qualidade das trajetórias geradas. Em seguida, esses foram novamente analisados com suas respectivas funções custo modificada. Os resultados da comparação demonstraram que o algoritmo R*, com direção de busca direta, é o mais eficiente para diferentes espaços e pesquisas. No entanto, a modificação em suas respectivas funções custo proporcionou uma melhora significativa nos resultados conquistados pelos algoritmos originais. Em ambientes estáticos, esta modificação se mostrou mais eficaz para problemas grandes e complexos, os que são efetivamente utilizados por robôs reais. Em ambientes altamente dinâmicos, a mesma apresentou uma redução considerável no tempo de planejamento e no número de iterações para localizar o objetivo, bem como reduziu a utilização de memória o que, consequentemente, tornou os robôs mais ágeis e habilidosos.

Palavras chaves: Função custo, planejamento de caminho, geração de trajetória.

Abstract

One of the main issues related to the mobile robotics area is to find the most efficient way to perform the navigation from one point to another over environments, considering maximum safety and spending as less as possible time and computer resources. From this perspective, the aim of this work was to specify improved heuristics that could be applicable to cost functions of key A based algorithms and use, more efficiently, the available computational resources. In this way, our approach aimed at minimizing the amount of collisions, the length of paths, and the processing time by minimizing the importance of g(n) term, which accounts for storing information from past steps of A* family algorithms. To show the effects of this modification, a survey of the best search strategies in dynamic and static environments was carried out and, after that, we analyzed the four best and latest algorithms, according to the specialized literature. Some comparisons have been made considering static and highly dynamic environments with different directions and search parameters to measure the quality of generated paths. Then, these algorithms were again analyzed with their cost functions modified according to our approach. The results of the comparison show that the R* algorithm, with forward search, is the most efficient for different spaces and searches. However, the change in their respective cost functions provided a significant improvement in the already excellent results achieved by the algorithms. In static environments, this modification showed up to be more effective for large and complex problems, which are commonly used for real robots. In highly dynamic environments, the cost function modification provided a considerable reduction in the time of planning and number of iterations to find the goal, as well as reductions in the memory utilization.*

Keywords: *Cost function, path planning, trajectory generation.*

Lista de Abreviaturas e Siglas ou abreviaturas

2D	Espaços compostos por 2 dimensões (x,y).
3D	Espaço compostos por 3 dimensões (x,y,θ).
a.C	Antes de Cristo.
$C_{espaço}$	Espaço de trabalho de um ambiente robótico.
C_{Livre}	Conjunto de espaços não ocupados por obstáculos.
$C_{obstáculo}$	Conjunto de espaços ocupados por obstáculos.
D.V.	Diagrama de *Voronoi:* Método de planejamento de caminho.
E.S.A	Agência Espacial Europeia.
F(n)	Variável que armazena o custo de um estado.
FEUP	Faculdade de Engenharia da Universidade do *Porto*.
g(n)	Valor do custo do caminho percorrido desde a posição inicial até o nó atual n.
h(n)	Função heurística que estimar o custo do caminho do nó atual n até o nó destino.
K	Limite de Iteração do algoritmo IDA*.
LaSER	Laboratório de Aplicações em Sistemas Embarcados e Robótica.
MB	Unidade de medida de informação equivalente a 2^{20} bytes.
ms	Unidade de medida de tempo equivalente a 10^{-3} segundos.
N.A.S.A	Administração Nacional da Aeronáutica e do espaço.
RAM	Memória computacional de acesso aleatório.
S.B.P.L	Laboratório de planejamento baseado em buscas.
SimTwo	Simulador dinâmico desenvolvido pelo professor Paulo Costa.
S_{inicio}	Posição inicial.
$S_{meta,}$	Posição objetivo.
UFPB	Universidade Federal da Paraíba.
V.G.	Grafo de visibilidade: Método de planejamento de caminho.
ε	Índice de otimalidade.

Lista de Figuras

Lista de Tabelas

Lista de Apêndices

Sumário

1. Introdução

O objetivo desse capítulo é fornecer uma visão geral e introdutória do problema estudado, apresentando a evolução da robótica e de sua aplicabilidade, bem como revelar como esse progresso moldou a ciência no intuito de fornecer soluções arrojadas e práticas para utilização de robôs autônomos nas mais diversas tarefas cotidianas. Também é exposta a motivação do trabalho, o problema de pesquisa e a organização do mesmo.

1.1. Contextualização

A robótica é uma tecnologia utilizada para auxiliar ou substituir o homem nas mais diversas atividades, executando ações que o ser humano não é capaz de executar, tais como mapear ambientes que possuem um alto grau de periculosidade, investigações marítimas, resgates em meios a incêndios, desarmamento de bombas, exploração de áreas com contaminação radioativa ou gases tóxicos, bem como realizar tarefas morosas e repetitivas como as atividades domésticas, industriais ou de vigilância.

Inicialmente, os robôs eram utilizados apenas na realização de tarefas repetitivas, como na automação industrial. Neste ambiente, os mesmos permanecem fixados em posições específicas na linha de montagem, movimentando apenas seus braços a uma grande velocidade e com elevada precisão a fim de realizar tarefas recorrentes. No entanto, os autômatos atrelados a esse domínio apresentam fundamental desvantagem no que diz respeito à falta de mobilidade. Com a evolução tecnológica e o auxílio de planejadores de caminhos, esses passaram a ser utilizados nas mais diversas áreas, como medicina de precisão, na área de entretenimento, serviços domésticos, exploração e etc. Para atender e apoiar a essas e outras aplicabilidades, surgiram as pesquisas para o desenvolvimento de robôs móveis autônomos, que devem ser capazes de atuar em ambientes reais e reagirem a situações desconhecidas de forma inteligente [1].

Entre as diversas aplicações da robótica móvel, pode-se citar o robô desenvolvido pela universidade *Carnegie Mellon*, denominado *Groundhog*, que trabalha autonomamente na exploração e mapeamento de minas abandonadas, enfrentando eminente risco de desabamento e, em muitos casos, gases tóxicos [2]. Também é possível citar o robô *Sojourner*, da Administração Nacional da Aeronáutica e do Espaço - NASA, que explorou e enviou fotos e outras muitas informações do planeta Marte para a

Terra [3] e o robô *Philae*, da Agência Espacial Europeia - ESA, que embarcou na nave espacial *Rosetta* e recentemente tornou-se o primeiro robô a posar na superfície de um cometa. Esse deverá realizar um estudo pioneiro sobre a composição física desses corpos celestes, visando entender a origem da vida e da água na Terra [4].

Na área de automação de serviços domésticos, a robótica móvel demonstra possuir grande potencial científico e vem atraindo a atenção de diversas empresas ligadas ao ramo tecnológico, além de receber cada vez mais investimentos para construção e aperfeiçoamento de produtos que executam autonomamente atividades ligadas ao lar. Dentre vários, destaca-se o aspirador de pó *Roomba* da *IRobot* [5], e o robô cortador de grama RS-630 da *Robomow* [6]. Ambos apresentam grande sucesso comercial na área de limpeza residencial e manutenção de gramados, respectivamente. Nos exemplos apresentados, a liberdade de movimento e autonomia dos robôs são conquistadas graças à diferentes técnicas de planejamento de caminhos que concedem aos mesmos a possibilidade de navegar pelo ambiente com segurança e rapidez. Essas técnicas serão apresentadas com mais detalhes no capítulo 2 dessa dissertação.

No campo do entretenimento, aplicações como jogos e/ou simulações virtuais [7], e animações realizadas por computador [8], utilizam de diferentes técnicas de planejamento de caminho para que toda idealização e execução dos movimentos de seus personagens ocorra de maneira natural dentro do cenário virtual.

Sendo assim, para um robô móvel conseguir atuar com liberdade e eficiência em diferentes ambientes, o mesmo deve ser capaz de obter e usufruir de informações sobre o local, possibilitando aferir sua posição atual e a de um obstáculo ou meta no ambiente, reconhecendo empecilhos e respondendo imediatamente as situações que ocorram. Essas tarefas de percepção, localização e movimento são problemas fundamentais da robótica móvel [9].

Usualmente, a navegação é a principal tarefa a ser executada por um robô móvel. A mesma consiste na localização do robô e do seu objetivo no ambiente, o mapeamento dos obstáculos e das rotas existentes no mesmo, o planejamento de um caminho eficiente entre a posição inicial e o destino e por fim, a execução do movimento pelo caminho planejado. A exploração dessas áreas e dos seus vários segmentos auxiliaram o crescimento acelerado do planejamento de caminho nos últimos anos, e uma das principais questões relacionados ao tema é descobrir a maneira mais eficiente para realizar essa navegação.

1.2. Motivação e justificativa

Partindo dessa indagação, o presente trabalho se motiva a desenvolver uma melhoria heurística que possa ser aplicável às funções custo dos principais algoritmos baseados na família A* e que propõe utilizar, de forma mais eficiente, os recursos computacionais disponíveis, melhorando assim, os resultados obtidos através dos principais algoritmos de buscas aplicados à robótica móvel. Essa otimização tem o objetivo de minimizar com eficácia a quantidade de colisões, a duração do trajeto, bem como o tempo de processamento de robôs holonômicos - que não possuem restrições cinemáticas e podem locomover-se com total liberdade e em qualquer direção -, como é o caso do robô *TurtleBot2* (Figura 1).

Figura 1[1]: Robô TurtleBot2

O objetivo geral deste trabalho está relacionado à proposta e validação de um melhoramento na função custo dos algoritmos de busca da família A*, bem como à tarefa de investigação dos avanços existentes no processo de planejamento de caminho na robótica móvel atual. A metodologia a ser desenvolvida tem ainda os seguintes objetivos específicos:

- Identificar os principais algoritmos de planejamento de caminho e de busca em grafos destacados pela literatura técnica especializada;

- Analisar vantagens e comportamento heurístico dos algoritmos escolhidos e verificar possibilidade de melhorias nas estratégias utilizadas;

[1] Imagem extraída da página: "*TurtleBot 2 - Open-source robot development kit for apps on wheels*." < http://www.turtlebot.com/assets/images/turtlebot_2_lg.png > Acessado em: 10 ago 2015

- Realizar um estudo comparativo entre os planejadores selecionados e eleger abordagem mais eficiente, considerando diferentes configurações dos ambientes, distintas direções de buscas e movimentação de obstáculos;

- Implementar modificação na função custo baseada na evolução das estratégias estudadas e validar a proposta através de testes comparativos entre suas respectivas versões originais;

1.3. Formulação do problema

Um robô autônomo deve ser capaz de perceber o ambiente à sua volta, tomar decisões sobre a melhor ação a ser executada e realizá-la com o mínimo erro possível. Para auxiliá-lo nessa tarefa é muito comum a utilização de estratégias de buscas heurísticas para resolver problemas grandes e complexos em um tempo satisfatório. A função heurística consegue melhorar a eficiência do processo de localização do objetivo, destacando entre vários caminhos possíveis, o mais vantajoso. Dentre os principais procedimentos de planejamento e geração de trajetória que utilizam função heurística, destacam-se os algoritmos da família A*. Nesse contexto, a pesquisa em questão visa responder o seguinte questionamento:

Como reduzir com eficiência o tempo de processamento e os recursos computacionais utilizados pelos algoritmos da família A e aplicados a robótica móvel?*

1.4. Hipótese da Pesquisa

Atualmente, existem inúmeros algoritmos de pesquisas em grafos que utilizam diferentes heurísticas e podem ser empregados no intuito de planejar um caminho aplicáveis à robótica móvel. Estes, fazem uso de distintas estratégias e oferecem suporte à diversos tipos de ambientes - conhecidos e/ou desconhecido, estáticos ou com alta dinâmica. Contudo, a grande maioria destes algoritmos conservam em suas respectivas heurísticas a função custo clássica herdada do planejador A*, para promover ou descartar estados. Desta forma, propõe-se potencializar o desempenho destes através de uma simplificação heurística que objetiva empregar os recursos computacionais de forma mais eficiente. Sendo assim, pode-se formular a seguinte hipótese para o problema estudado:

É possível reduzir o tempo de processamento, o consumo computacional e melhorar a eficiência dos principais algoritmos da família A, através da minimização da importância da variável g(s) - responsável em armazenar, calcular e atualizar*

informações do custo da rota já realizadas pelo autômato a cada iteração, em suas respectivas funções custos.

1.5.Organização do texto

A presente dissertação está organizada em 6 partes específicas. No Capítulo 2 é apresentada a fundamentação teórica que embasa a investigação, bem como detalha os principais conceitos, terminologias e procedimentos citados no decorrer da mesma. O Capítulo 3 apresenta os principais trabalhos relacionados com a proposta, destacando as principais vantagens dos algoritmos de pesquisa em grafos ressaltados pela literatura técnica especializada. O Capítulo 4 descreve em detalhes a metodologia utilizada para realizar as experimentações. O Capítulo 5 exibe os resultados obtidos e, por fim, a conclusão e os trabalhos futuros são apresentados no decorrer da Capítulo 6.

2. Fundamentação teórica

Por meio do planejamento do caminho é possível definir, de maneira mais eficiente, a trajetória que leve o robô com segurança de uma posição inicial até uma posição objetivo almejada.

Neste capítulo, serão apresentados os métodos mais utilizados no âmbito do planejamento do caminho na robótica móvel, bem como serão detalhadas as suas finalidades. Inicialmente, alguns conceitos e terminologias serão explicados e será abordado o espaço de configuração. Em seguida, serão expostas as soluções relacionadas com o planejamento do caminho e por fim apresentadas as considerações finais.

2.1. Conceitos e terminologias

Dado que a presente proposta de pesquisa faz uso de diversos tópicos de estudo no campo da inteligência artificial e da robótica, serão adotados alguns conceitos e terminologias básicas apresentadas nesta seção. Três conceitos iniciais são:

- Planejamento do caminho (*path planning*)
- Planejamento de trajetória (*trajectory planning*)
- Planejamento de movimentos (*motion planning*)

O planejamento do caminho descreve, em termos geométricos ou matemáticos, qual rota o robô deverá seguir para que o mesmo possa se descolocar com segurança de um ponto inicial até um ponto destino, evitando obstáculos, agentes externos e/ou outros robôs.

O planejamento de trajetória representa esse caminho em função do tempo. Ou seja, indica para cada instante de tempo, qual o local exato onde o robô deverá estar posicionado.

O planejamento de movimento considera as restrições cinemáticas e dinâmicas do robô e como as mesmas podem interferir em sua trajetória.

Estes três conceitos, muitas vezes são confundidos e empregados erroneamente como sinônimos na literatura. Partindo desse ponto, serão apresentados alguns aspectos importantes que devem ser considerados para um planejamento eficiente na robótica móvel.

2.2. Considerações na escolha de um método

Quando se pretende eleger qual o método utilizar para o planejamento do caminho, vários aspectos devem ser considerados, tais como:

- A natureza da otimização pretendida, se o objetivo é otimizar o comprimento do caminho, o tempo de execução da trajetória, a energia consumida e/ou outras variações.
- Se o método é online ou off-line. Um método é caracterizado off-line se o mesmo constrói a solução baseado no modelo do ambiente, ou seja, antes de iniciar a trajetória. O método é dito online se a solução for construída à medida que o robô a executa.
- Se a complexidade computacional é muito elevada, haja visto que muitos dos métodos demonstrados teoricamente tornam-se inviáveis de serem implementados, devido a possuírem um elevado tempo de execução e/ou consumo de memória.
- Se o método é completo, completo em resolução ou probabilisticamente completo.
    - Um método é descrito como completo, se sempre conseguir localizar uma solução, caso ela exista.
    - Um método é denominado completo em resolução caso exista uma solução para uma determinada discretização do ambiente.
    - O método diz-se probabilisticamente completo se a probabilidade de localizar uma solução converge para 1 a medida que o tempo tende ao infinito.
- Se o robô é holonômico ou não holonômico.
    - Um robô é declarado holonômico quando possui vasta liberdade de movimento, podendo se mover em qualquer direção, como é o exemplo dos robôs omnidirecionais.
    - Um robô é denominado não holonômico se possuir restrições de direções em seus movimentos, como é o exemplo de um carro.
- Se os obstáculos estão parados, em movimento previsível ou movimento desconhecido.

- Se os obstáculos são deformáveis ou não.

Todos esses aspectos deverão ser considerados quando se objetiva definir a técnica a ser escolhida. Nesta dissertação o método utilizado será completo em resolução, tendo como objetivo principal a minimização do tempo de execução da trajetória. Em um primeiro momento, serão analisados métodos de planejamento *off-line*, para o estudo do planejamento de caminho em ambiente estáticos e posteriormente *online*, para a observação do planejamento em ambientes com características dinâmicas. Inicialmente os obstáculos serão estáticos e em um segundo momento dinâmicos - com movimentação desconhecida, estes também possuirão características não deformáveis. Por fim, a liberdade de movimentação adotada será a total, como vista em robôs omnidirecionais.

2.3. Espaço de configuração

Para planejar um caminho e os movimentos de um robô, faz-se necessário possuir informações do ambiente onde o mesmo está inserido e que especifique completamente a sua localização, visto que é essencial garantir que nenhuma colisão com obstáculos ocorra. Essas informações são especificadas por um conjunto mínimo de parâmetros que podem detalhar todas as configurações possíveis no sistema e é denominado de $C_{espaço}$. A dimensão do problema é dada pelo número de parâmetros que define esse espaço e é conhecida como grau de liberdade do robô. O fragmento do $C_{espaço}$ que não é ocupado por obstáculos é denominado de espaço livre e é representada por C_{Livre}. Por fim, a parte ocupada por obstáculos é representada por $C_{obstáculo}$. Esses conceitos tornaram-se muito importante na resolução de problemas de planejamento de trajetória e foram utilizados pela primeira vez por [10].

A posição e a orientação de um robô em um plano no espaço físico pode ser especificado por três parâmetros (x,y,θ) no $C_{espaço}$, com rotação, ou dois parâmetros (x,y) no $C_{espaço}$, sem rotação. Essa transformação do espaço físico para o $C_{espaço}$ faz com que o robô passe a ser tratado como um ponto, ou seja, no $C_{espaço}$ o robô diminui para um ponto e aumenta-se os obstáculos com o raio que foi subtraído do autômato. Sendo assim, a principal vantagem dessa modelagem está relacionada com o fato de ser mais fácil planejar o movimento para um único ponto do que para um corpo rígido complexo. Desse modo, o problema do planejamento de trajetória no espaço físico consiste apenas na pesquisa de um caminho livre de colisões para o ponto que representa o robô entre dois pontos pertencentes ao seu espaço de configuração.

2.3.1. Procedimentos para calcular o $C_{obstáculo}$

O $C_{obstáculo}$ é o conjunto de todas as configurações em que o robô se superpõe parcial ou totalmente ao obstáculo. De acordo com [11], existem sete métodos básicos para o cálculo de obstáculos no espaço de configuração e sua utilização é válida para qualquer tipo de robô. Em seguida são apresentados os métodos.

- **Ponto de evolução (*Point of evalution*)**

Para cada posição possível do robô é verificado se a mesma intercepta algum obstáculo no intuito de determinar se esta pertence ou não ao conjunto $C_{obstáculo}$. Este método é o mais simples de ser implementado, mas em contrapartida, o mais ineficiente.

- **Conjunto diferença de Minkowski (*Minkowski set difference*)**

O conjunto diferença de Minkowski, entre dois conjuntos A e B, são posições definidas pela seguinte expressão: $Mdiff(A,B) = \{a-b \mid a \in A, b \in B\}$. Nessa fórmula, o robô é considerado um objeto rígido e sem rotação. O $C_{obstáculo}$ é então constituído pela união dos conjuntos de diferenças de *Minkowski* entre as áreas ocupadas pelo obstáculo e o robô. Na Figura 2, a zona sombreada em cinza faz parte de $C_{obstáculo}$ definido pelo método em questão. A região cinza clara é o Mdiff(A,B) e o cinza escuro, o obstáculo. O ponto de referência *r* não pode estar dentro destas zonas, sob pena de o robô encontrar-se em colisão com o obstáculo. Este método foi implementado pela primeira vez em [12].

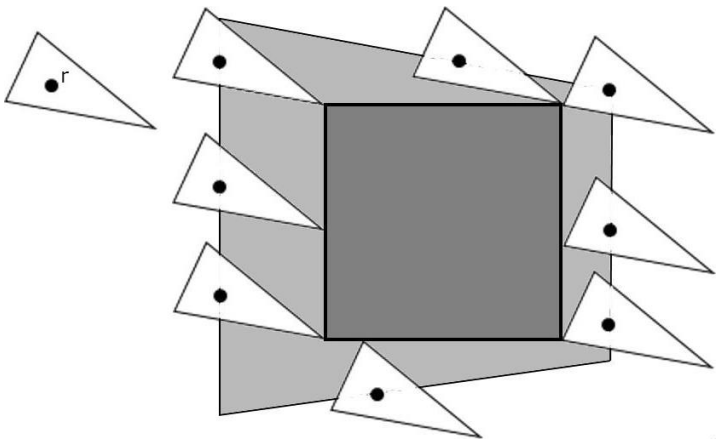


Figura 2[2]: Definição do $C_{obstáculo}$, a partir do conjunto diferença de *Minkowski*

[2] Imagem extraída e adaptada da página: "*Motion Planning HW5*" do professor *Hyun Soo Park* < http://www.andrew.cmu.edu/user/hyunsoop/HW5/Motion_Planning_HW5.htm >

- **Equações de fronteira (*Boundary equations*)**

Este método utiliza de equações matemáticas para definir quando o robô toca os obstáculos. Isso é feito através de equações de restrições que são originadas a partir dos vértices e das bordas de contato entre o robô e os obstáculos. Normalmente, estes casos servem para testar se uma configuração particular está em C_{Livre}, pois a utilização de equações de fronteira para representar todos os obstáculos presentes em $C_{obstáculos}$ são consideradas inviáveis devido a sua complexidade, especialmente para graus de liberdade superior a 3. Em [13], foi descrito o método para obter a equação de fronteira de um plano e em [14], para um ambiente em 3D.

- **Modelos (*Templates*)**

Segundo [15], o cálculo do espaço de configuração é feito de acordo com as características dos obstáculos, verificáveis no Espaço de trabalho - região no espaço físico que o robô pode alcançar através de seus movimentos. Estes, são tipicamente decompostos em formas mais simples, tais como pontos e linhas e as suas correspondências no espaço de configuração são denominas *templates*. Essas formas simplificadas são parametrizadas e a união das mesmas geram o $C_{obstáculo}$. Este método funciona bem para graus de liberdade menores que 4, devido à alta utilização de memória.

- **Método agulha (*Needle method*)**

Este método cria o $C_{obstáculo}$ fixando todos os parâmetros da configuração, à exceção de um que é variável. O valor desse parâmetro é então definido utilizando equações de fronteira para detectar as colisões. O $C_{obstáculo}$ é então representado como um conjunto de intervalos discretos e é frequentemente utilizado para gerar seções transversais de duas dimensões do espaço de configuração, para cada um dos parâmetros fixados. A utilização deste método não é viável para espaços de configuração com dimensão elevada, visto que é necessária uma grande quantidade de intervalos para os representar. O mesmo foi implementado pela primeira vez em [12].

- **Método do varrimento do volume (*Sweep volume method*)**

Este método é usado para construir áreas em que o robô pode mover-se sem colisão, conhecido como C_{Livre}. Com ele é realizado um varrimento no espaço de trabalho do robô, variando através da fixação de um dos parâmetros da configuração do robô e armazenando os outros valores de parâmetros da posição do autômato em C_{Livre}, repetindo

assim o processo para todos os parâmetros. Se o espaço varrido não contemplar nenhuma interseção com obstáculos, então o conjunto não pertence ao $C_{obstáculo}$. Segundo [11], o método se torna computacionalmente complexo quando possui grau de liberdade superior a 3. Em [16] foi demonstrado uma variação desse método denominada de busca sequencial.

- **Método baseado na matriz Jacobiana (*Jacobian-based method*)**

De acordo com [15] e [17], este método consiste num procedimento capaz de calcular tanto blocos de C_{Livre} como de $C_{obstáculos}$. A matriz Jacobiana *J* de um robô relacionará o deslocamento *dx* de um ponto do mesmo com a mudança de sua configuração *dq*. Isto é, $dx = J(q).dq$, sendo *J* função de *q*. Para o robô na configuração *q*, o máximo de $|J(q)|$ para todos os pontos do robô é designado por limite $B(q)$. Se a mínima distância entre o robô em *q* e todos os obstáculos for *D*, então a esfera centrada em *q*, com raio $D|B(q)$, estará inclusa em C_{Livre}. De modo similar, caso seja definido uma distância mínima D^- como a distância negativa[3] entre dois objetos que se sobreponham e represe a translação necessária para se separarem, pode-se calcular uma esfera com centro em *q* e raio $D^- | B(q)$ que é um $C_{obstáculo}$.

2.3.2. Métodos para planejamento do caminho

Ao longo desta seção, serão apresentados alguns dos principais métodos de planejamento de caminho, que normalmente são utilizados no contexto da robótica móvel. Segundo [18], os algoritmos clássicos de planeamento de caminho podem ser divididos em três tipos: (1) Métodos *Roadmap*; (2) Decomposição em Células e (3) Métodos de Campos Potenciais. Estes não são mutuamente exclusivos, podendo existir combinações entre eles no intuito de tirar melhor proveito de suas vantagens. Em seguida, será dada uma breve explicação de cada uma das categorias.

2.3.2.1. Mapa de Rotas (*Roadmap*)

De acordo com [19] e [20], a técnica de *roadmap* para planejamento de caminho respalda-se em minimizar as informações do ambiente a um grafo simbolizando os possíveis caminhos livres existentes, o C_{Livre}, em uma rede de curvas unidimensional. O planejamento do caminho então se sumariza a conectar os pontos inicial e final do robô

[3] Distância negativa é a denominação dada ao deslocamento realizado pelo o agente autônomo contra a orientação da trajetória.

no mapa de rotas e pesquisar neste, através de algoritmos de busca em grafos, uma passagem entre esses dois pontos. Se o mesmo existir, esse será dado pela junção de três subcaminhos: um entre o ponto inicial até algum ponto do *roadmap*, um subcaminho dentro do mesmo e um outro do mapa até o ponto destino. Vários métodos foram propostos baseados nessa ideia, em seguida serão apresentados os principais para a sua construção.

○ Grafo de Visibilidade (V.G.)

Este método é normalmente escolhido para espaços com duas dimensões (2D), em que os nós são vértices dos obstáculos e as ligações entre eles só existem se os dois vértices estiverem de frente um para o outro, isto é, se existir uma reta que conecte os vértices e que não passe por nenhum obstáculo, como visto em [20, 10]. O ponto inicial e o ponto de destino são considerados nós e esses devem estar ligados a pelo menos um outro nó para existir um caminho, como pode ser visto na Figura 3. As zonas sombreadas em cinza representam obstáculos e as linhas sólidas são as bordas do gráfico que se ligam aos vértices dos mesmos. As linhas pontilhadas conectam as configurações iniciais e finais com o *roadmap*.

Existem $O(n^2)$ ligações que podem ser construídas com o *Roadmap* no tempo $O(n^2)$, onde *n* é o número de nós. Segundo [21], uma das principais desvantagens desta técnica é que qualquer caminho gerado por ela passará muito rente aos vértices dos obstáculos, como novamente verificado na Figura 3, o que pode causar muitas colisões, caso haja qualquer erro no controlador do robô.

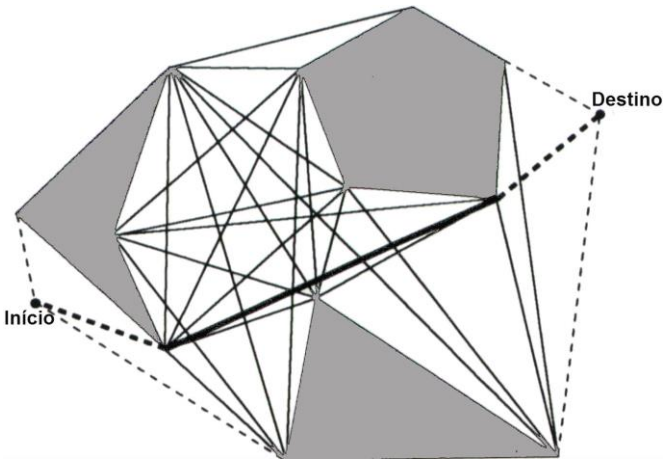


Figura 3[4]: Exemplo de uma trajetória gerada com um mapa em V.G.

---

4 Imagem extraída e adaptada da página: "*Motion Planning in Robotics*" do professor *Eric S. Roberts* < http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html >

○ Diagrama de *Voronoi* (D.V.)

O diagrama *Voronoi* (D.V.) [22] é um agrupamento de pontos equidistantes de pelo menos dois obstáculos que subdivide o espaço de configuração em seções distintas com a existência de apenas um entrave por seção. Cada aresta[5] do diagrama constitui um lugar onde os pontos são equidistantes em relação a dois locais. Diferentemente do Grafo de Visibilidade, qualquer caminho gerado pelo D.V. estará bem afastado dos vértices dos obstáculos. O mesmo é composto por curvas quando mensura as distâncias em relação às arestas dos polígonos. Uma implementação diferente foi desenvolvida em [23], contendo apenas linhas retas e que utilizava uma medida de distância distinta da euclidiana. Para localizar o caminho no D.V., deve-se ligar o ponto inicial, representando o robô, até o diagrama de *Voronoi*, em seguida localizar o melhor caminho dentro do mesmo e por fim, liga-o ao ponto destino, como visto na Figura 4. Nesta figura, as zonas sombreadas em cinza representam obstáculos do ambiente, as linhas sólidas são as arestas que por sua vez estão equidistantes a pelo menos dois obstáculos e formam assim, uma passagem segura para o robô. As linhas pontilhadas conectam as configurações iniciais e finais ao D.V.

Conforme [24], o diagrama de *Voronoi* para *n* obstáculos pode ser construído em O(n) tempo e possuir O(n) ligações. Para o cálculo do D.V, diversos trabalhos desenvolvidos apresentam diferentes soluções, como são os casos de [24, 25, 26].

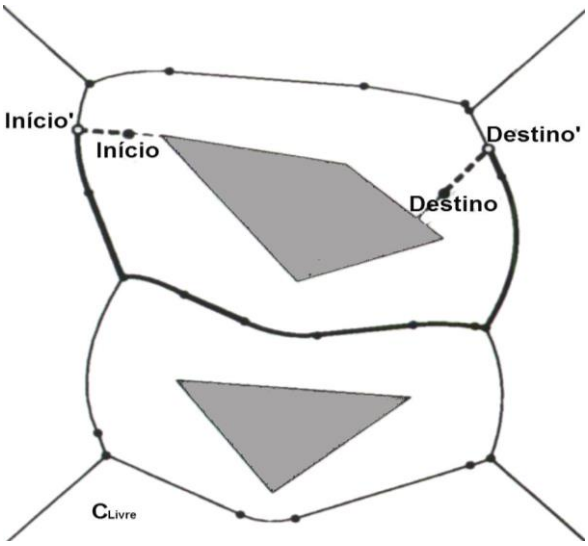


Figura 4[6]: Exemplo de uma trajetória gerada com um mapa em D.V

[5] Aresta é a denominação dada ao segmento que representa a intersecção de duas faces de um poliedro.

[6] Imagem extraída e adaptada da página: "*Motion Planning in Robotics*" do professor *Eric S. Roberts* < http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html >

- **Método da Silhueta**

Desenvolvido por [27], o método da silhueta foi projetado para gerar caminhos na fronteira do espaço livre em espaços dimensionais elevados. O mesmo baseia-se na projeção dos limites de C_{Livre} em planos de dimensão inferior – silhueta do espaço livre - acrescentando recursivamente segmentos de curvas da fronteira da projeção até reduzir a dimensão a um plano 2D.

As opções de rotas geradas são representadas por um grafo onde os seus ramos retratam os segmentos de curva e os seus nós, as extremidades desses segmentos. Os caminhos entre o ponto inicial e ponto objetivo são localizados por meio de algoritmo de pesquisa em grafos. Contudo, segundo [18], uma das desvantagens deste método reside em sua elevada dificuldade de implementação. Uma variação deste algoritmo foi sugerida e implementada em [28].

2.3.2.2. Campos potenciais

De acordo com [29], a utilização de campos potenciais no planejamento de caminho, baseia-se no uso dos potenciais elétricos da física como heurística para encontrar a trajetória. A ideia principal deste método consiste em considerar o robô como um ponto com carga positiva e massa finita, pertencente ao $C_{espaço}$, e que se desloca num espaço com obstáculos sob a influência de um campo potencial artificial. Esse campo é elaborado de forma a atrair o robô ao objetivo à medida que o repele dos obstáculos existentes. O $C_{obstáculo}$, comporta-se como partícula de carga positiva, criando um campo repulsivo, e o ponto de destino, comporta-se como uma partícula de carga negativa, atraindo o robô para si. Essa combinação de campos, em um cenário ideal, obriga o robô a deslocar-se em direção ao objetivo, evitando os obstáculos como visto na Figura 5.

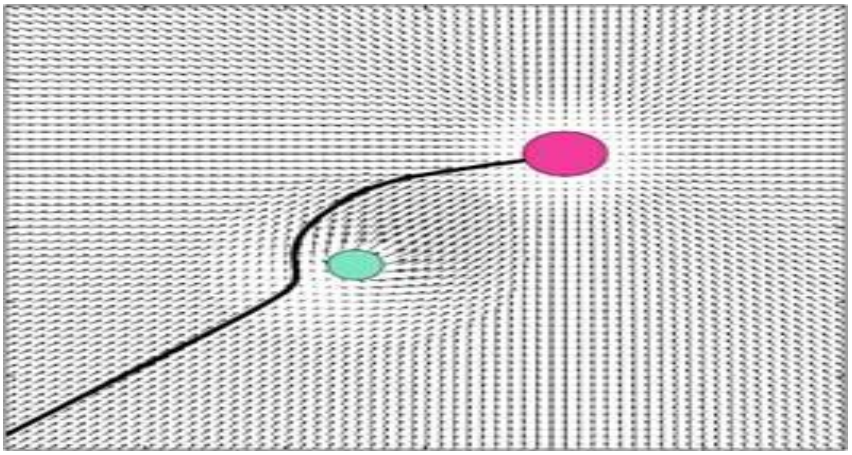


Figura 5[7]: Exemplo de uma trajetória gerada por campos potenciais

A combinação das forças de atração e repulsão geram o campo potencial artificial. Com a utilização dessas forças é possível fazer o robô desviar de obstáculos e se nortear em direção ao objetivo, contudo para se obter uma trajetória completa, faz-se necessário utilizar um algoritmo de geração de trajetória. Nesse contexto, um dos mais utilizados é o do gradiente descendente. Esse algoritmo faz com que o robô partindo de sua posição inicial, mova-se no sentido oposto do gradiente resultante (sentido da força resultante) e siga para uma nova configuração no C_{Livre}. Em seguida, é repetido o mesmo movimento até conquistar um gradiente igual a 0.

A Figura 6 apresenta um exemplo de geração de trajetória completa com campos potenciais. Na mesma, os obstáculos são representados por zonas sombreadas em cinza (a). O campo potencial atrativo é um paraboloide com ponto de mínimo localizado na posição do objetivo (b). O campo potencial repulsivo difere de zero somente a partir de uma determinada distância limite dos obstáculos. Abaixo dela, quanto mais próximo dos obstáculos, maior a influência (c). O sentido da força resultante é representado em (d). O caminho gerado por essa abordagem, definido pelo algoritmo do gradiente descendente, é apresentado em (e). Em (f), tem-se uma matriz de orientações do vetor gradiente, representando as orientações das forças induzidas pelo campo potencial.

A construção do campo potencial é dita como correta, quando existe apenas um mínimo global, coincidente com a configuração objetivo, não existindo nenhum outro

[7] Imagem extraída da página: "*Local Path Planning Using Virtual Potential Field*" do professor *Hani Safadi* < http://www.cs.mcgill.ca/~hsafad/robotics/>

mínimo local ao longo do caminho; caso contrário, segundo [30], o robô pode não conseguir chegar ao destino.

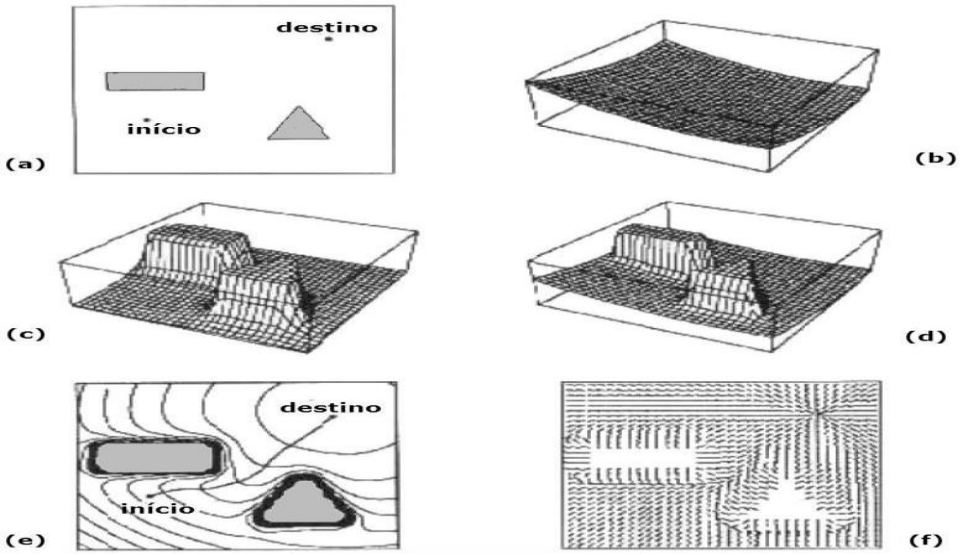


Figura 6[8]: Processo completo de geração de trajetória com campo potenciais

2.3.2.3. Decomposição por célula

Conforme [18], este método baseia-se em dividir o espaço de configuração, $C_{espaço}$, em regiões geométricas simples e não sobrepostas, denominadas células, de forma que um caminho entre quaisquer duas células possa ser facilmente obtido. Um grafo, conhecido como grafo de conectividade é então criado, representando a relação de adjacência entre as mesmas. Nessa representação, os nós do grafo representam as células que atuam como as configurações e os ramos ligam nós correspondentes a células adjacentes. Em seguida, a solução é encontrada através de algoritmos de pesquisa em grafos. Segundo [31], o resultado de um caminho é uma sequência de células denominada canal, onde este permite computar um caminho contínuo que pode ser determinado simplistamente seguindo as células livres adjacentes, desde o ponto inicial até o ponto objetivo.

Os passos desse método se resumem a: primeiramente, aloca-se o ponto inicial, representando o robô, a uma célula e o ponto destino a outra e em seguida, localiza-se, através de algoritmos de buscas em grafos, a sequência de células que ligam as duas. Este

[8] Imagem extraída de [31]

método pode ser subdividido em três tipos, decomposição em células exatas, aproximadas e modificada:

- ## Decomposição em células exatas

Segundo [18], na decomposição em células exatas, a divisão de $C_{espaço}$ é feita de forma a representar com exatidão a configuração do mundo real, ou seja, consegue-se retratar, de forma fidedigna, os espaços livres ou espaços ocupados do $C_{espaço}$ através da geometria. Para que isso aconteça, o método de decomposição tem de criar células com geometria simples para que facilmente se possa calcular o caminho de duas configurações e que facilite localizar suas células vizinhas e o caminho pretendido. Duas das decomposições mais utilizadas nessa abordagem são: polígonos convexos e trapézios.

- ### Polígonos Convexos

Na decomposição por polígonos convexos, os vértices dos polígonos são os vértices dos obstáculos e as ligações representam as células adjacentes. Os caminhos começam a ser construídos a partir dos pontos médios dos limites dessas células, como mostra a Figura 7.

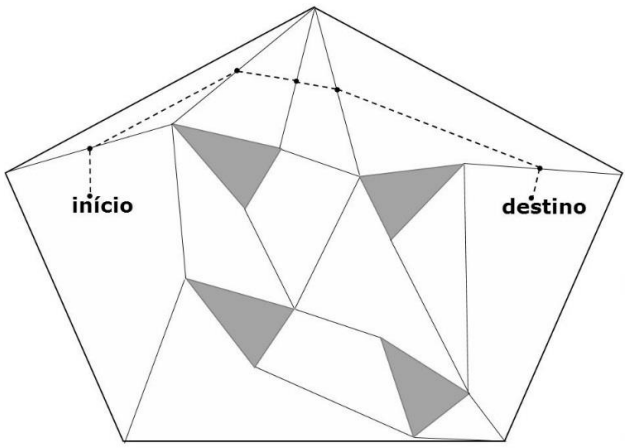


Figura 7[9]: Exemplo de uma decomposição por polígonos.

- ### Trapézios

Nessa decomposição, o C_{Livre} é subdividido em trapézios, os quais são obtidos à custa de linhas verticais desenhadas a partir dos vértices dos polígonos que

[9] Imagem adaptada de [32]

constituem o espaço obstáculo. Essas linhas representam as arestas das células. A Figura 8 mostra uma decomposição para 3 obstáculos.

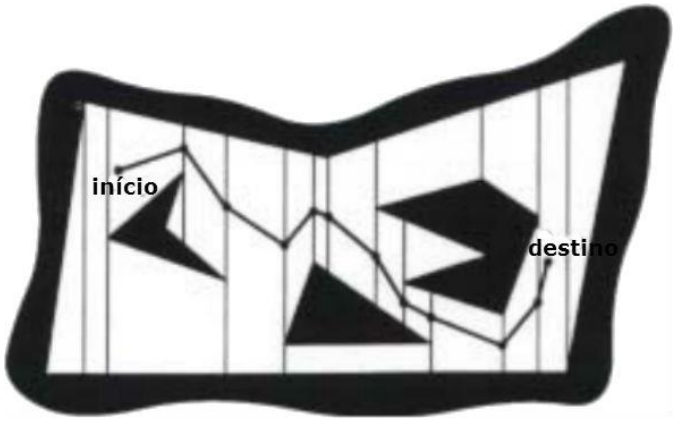


Figura 8[10] - Exemplo de uma decomposição por trapézio

- **Decomposição em células aproximadas**

A decomposição aproximada de células é uma forma de representação do $C_{espaço}$ que não retrata com exatidão o espaço real. A mesma divide o espaço de configurações em células com formas previamente definidas, como retângulos para o caso bidimensional, ou paralelepípedos, para o tridimensional, no intuito de simplificar e otimizar a construção do mesmo.

De acordo com [18], as células podem possuir três estados distintos para representar o ambiente: livres, ocupadas ou parcialmente ocupadas. Um dos problemas gerados por essa abordagem pode ser a não localização do caminho, mesmo que ele exista, pois, o tamanho das células influencia o resultado. Existem várias formas de decomposição, sendo a célula fixa e *quadtree*, as mais usadas para os espaços em 2D.

- **Célula Fixa**

É um método de simples implementação, no qual todas as células têm tamanho pré-definido e divide-se o $C_{espaço}$ em pequenos quadrados desse tamanho. Tal ação facilita os cálculos para localizar o caminho, como mostra a Figura 9.

[10] Imagem extraída e adaptada da página: "*Motion Planning in Robotics*" do professor *Eric S. Roberts* < http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html >

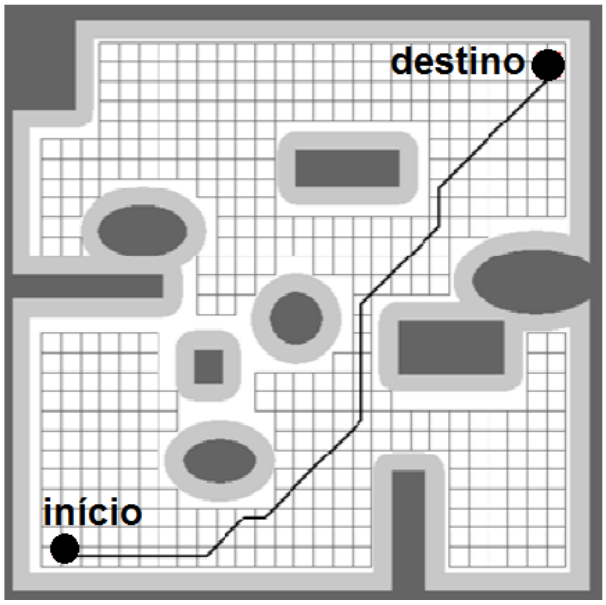


Figura 9[11] - Exemplo de uma decomposição por célula fixa

- ***Quadtree***

Essa técnica decompõe, recursivamente, o $C_{espaço}$ em 4 células quadradas idênticas, e sempre que uma célula não pertencer ao C_{Livre}, volta a dividir essas células em mais 4 até que um limite mínimo de tamanho seja alcançado. Segundo [32] e [33], as vantagens de tal decomposição é a minimização do número de células existentes que pode provocar redução no tempo de execução do algoritmo, além de acelerar a pesquisa e o endereçamento das células vizinhas. Contudo, essa solução torna-se inviável devido à alta exigência de recursos computacionais e a sua elevada complexidade de implementação. Na Figura 10, pode-se visualizar tal divisão.

[11] Imagem extraída de [32]

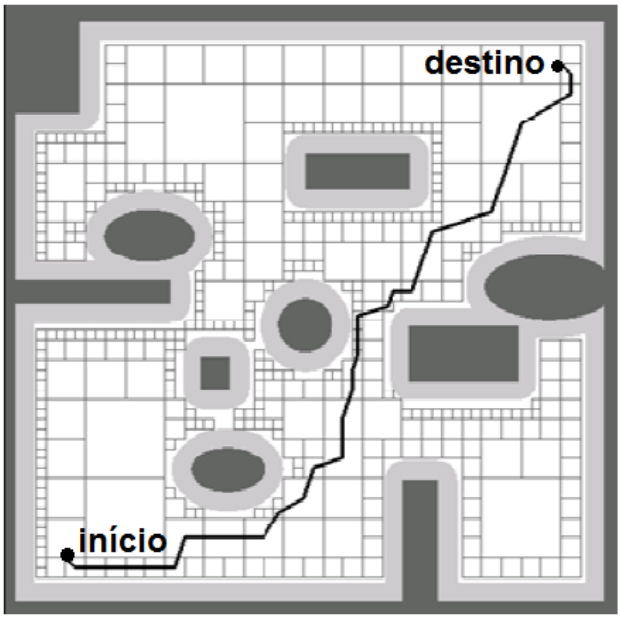


Figura 10[12] - Exemplo de uma decomposição por *Quadtree*

- **Modificação na decomposição por célula**

Em [34] é apresentado uma versão modificada da decomposição por células que incluí algumas melhorias no seu espaço de configuração, dentre essas: (1) Mensuração da importância relativa dos obstáculos em relação a aproximação com o robô – os obstáculos apenas ganham importância, crescem e são considerados, à medida que se aproximam do autômato. Os que não estão em seu campo de visão são momentaneamente desprezados. Tal ação, segundos os autores, minimizaria o custo computacional para processar essas informações. (2) Aperfeiçoamento na representação dos obstáculos com a inclusão de uma área extra de segurança - é acrescido uma área adicional nos obstáculos para evitar indesejadas colisões proveniente da aproximação rápida do robô com os mesmos, otimizando assim o deslocamento dos autômatos. (3) Consideração da direção e velocidade dos obstáculos no intuito de evitar que esses dificultem, por um longo tempo, a movimentação do robô, caso desloquem-se paralelamente e na mesma direção com o autômato. Além dessas, a modificação consideraria futuros pontos de colisões do robô com os obstáculos baseado em suas velocidades.

[12] Imagem extraída de [32]

A Figura 11 apresenta, com o auxílio do simulador *simtwo* [67], os diferentes campos de visão utilizados pelo robô, quando considerado a utilização da melhoria (1). Com o auxílio da subfigura esquerda, é possível perceber que os obstáculos que estão distantes do campo de visão do autômato possuem importância e tamanho de raio bem menores dos que estão próximos e, além disso, é possível verificar a direção dos deslocamentos dos mesmos, próximos ao agente, com base em seus formatos. Já na subfigura direita, todos os obstáculos têm importância e tamanho iguais, independentemente de estarem ou não no campo de atuação do robô - com isso, segundos os autores, são consideradas e calculadas, desnecessariamente, a movimentação de todos os obstáculos, mesmo os que não estão próximos ao agente.

Figura 11: Campo de visão dos robôs com as modificações ativada e desativada

A Tabela 1, apresentada a seguir, expõe o resumo dos principais métodos utilizados para identificação dos obstáculos e, a Tabela 2, exibe a síntese dos principais algoritmos empregados para o planejamento de caminho.

Tabela 1: Métodos utilizados para identificar $C_{obstáculo}$

Métodos	Resumo
Ponto de evolução	Segundo [11], o robô é colocado numa determinada configuração e verifica-se a interseção do mesmo com os obstáculos presentes no ambiente. Este método é o mais simples de ser implementado, contudo é o mais ineficiente.
Conjunto diferença de *Minkowski*	Implementado pela primeira vez por [12], o método define as posições do ambiente através da expressão: $Mdiff(A,B) = \{a-b \mid a \in A, b \in B\}$. Dessa forma, forma-se o conjunto de $C_{obstáculos}$ pela união dos conjuntos diferenças das áreas ocupadas pelo robô e pelos obstáculos.
Equações de fronteira	Implementado para ambientes planos em [13] e ambientes em 3D em [14], este método utiliza equações de restrições matemáticas para definir quando o robô toca ou não um obstáculo. Isso é feito a partir dos vértices e das bordas de contato entre o robô e os obstáculos.
Modelos (*Templates*)	Segundo [15], os obstáculos são decompostos em formas mais simples, tais como pontos e linhas, e as suas correspondências no espaço de configuração são parametrizadas, a união dessa geram o $C_{obstáculo}$.
Método da agulha	Conforme [12], este método fixa todos os parâmetros da configuração, à exceção de um. O valor deste parâmetro, que é variável, é então calculado utilizando as equações fronteira para detectar o $C_{obstáculo}$
Método do varrimento por volume	Segundo [11], este método identifica áreas onde o robô pode se mover sem colisões – C_{Livre}, através de um varrimento no seu espaço de trabalho. Se o espaço varrido não contemplar nenhuma interseção com obstáculos, então o conjunto não pertence ao $C_{obstáculo}$.
Método baseado na matriz jacobiana	De acordo com [15] e [17], este método pode ser utilizado para cálculo de blocos de C_{Livre} ou $C_{obstáculo}$. O mesmo, utilizando a matriz Jacobiana *J* de um robô, relaciona o deslocamento de um ponto pertencente ao mesmo com a mudança de sua configuração. Se a distância mínima obtida entre o robô e todos os obstáculos for D, então a esfera estará inclusa em C_{Livre}. Contudo, caso a distância mínima for D^-, então a esfera estará inclusa em $C_{obstáculo}$.

Tabela 2: Algoritmos de planejamento de caminho

Algoritmo	Versão	Resumo
Roadmap [19]	Grafo de visibilidade [10]	Este método é normalmente escolhido para espaços com duas dimensões em que os nós são vértices dos obstáculos e as ligações entre eles só existem se os dois vértices estiverem de frente um para o outro. Segundo [21], uma das principais desvantagens desta técnica é que qualquer caminho gerado por ela passará muito rente aos vértices dos obstáculos.
	Diagrama de *Voronoi* [22]	O diagrama Voronoi é um agrupamento de pontos equidistantes de pelo menos dois obstáculos que subdivide o espaço de configuração em seções distintas, com a existência de apenas um obstáculo por seção. Diferentemente do Grafo de Visibilidade, qualquer caminho gerado pelo D.V. estará bem afastado dos vértices dos obstáculos.
	Método da silhueta [27]	O método da silhueta foi projetado para gerar caminhos nos limites de C_{Livre} em planos de dimensão inferior – silhueta de C_{Livre}. Este método acrescenta, recursivamente, segmentos de curvas na fronteira de projeção até reduzir a dimensão a um plano 2D. Contudo, segundo [18], uma das desvantagens deste método reside em sua elevada dificuldade de implementação.
Campo potenciais [29]	A utilização de campos potenciais no planejamento de caminho, baseia-se no uso dos potenciais elétricos da física como heurística para encontrar a trajetória. O robô é considerado um ponto de carga positiva que se desloca num espaço com obstáculos sob a influência de um campo potencial artificial. O $C_{obstáculo}$ comporta-se como partícula de carga positiva, criando um campo repulsivo e o ponto de destino comporta-se como uma partícula de carga negativa, atraindo o robô para si. Essa combinação de campos, em um cenário ideal, obriga o robô a deslocar-se em direção ao objetivo. Contudo, segundo [30], a utilização dessa estratégia maximiza as chances de o robô ficar preso a mínimos locais e não alcançar o objetivo.	
Decomposição por células [18]	Exata [18]	Na decomposição em células exatas, a divisão de $C_{espaço}$ é feita de forma a representar com exatidão a configuração do mundo real, conseguindo retratar de forma fidedigna, os espaços livres ou espaços ocupados do $C_{espaço}$ através do uso da geometria. Duas das formas geométricas mais utilizadas na decomposição são: polígonos convexos e trapézios.
	Aproximadas [18]	A decomposição aproximada de células é uma forma de representação do $C_{espaço}$ que não retrata com exatidão o espaço real. A mesma divide o espaço de configurações em células com formas previamente definidas, no intuito de simplificar e otimizar a construção do $C_{espaço}$. Um dos problemas gerados por essa abordagem pode ser a não localização do caminho, mesmo que ele exista, pois, o tamanho das células influencia o resultado. Existem várias formas de decomposição, sendo a célula fixa e *quadtree*, as mais usadas para os espaços em 2D.
	Modificada [34]	Na decomposição modificada, é acrescentado modificações para melhor representar o espaço de configurações. Entre elas, a mensuração da importância de um obstáculo com base em sua distância, a inclusão de uma zona de segurança adicional, cálculos das futuras posições dos obstáculos com base em sua velocidade.

2.4.Considerações finais

Nessa subseção foram apresentados os principais procedimentos para mapear o $C_{obstáculo}$ de um ambiente. Através desses, são delimitadas as zonas ocupadas existentes no espaço de configuração, permitindo assim que o robô as evitem e possa navegar com segurança e autonomia. Tal procedimento é importante, pois fornece ao autômato informações úteis para o planejamento e alcance do seu objetivo, enquanto evita os obstáculos existentes no trajeto.

As três principais estratégias para o planejamento do caminho possuem diversas vantagens e inconveniências, que devem ser consideradas de acordo com o problema em questão que se pretende resolver. No caso específico dessa proposta, que visa o melhoramento de planejadores utilizados em ambiente com alta dinâmica, a abordagem do *Roadmap* não se mostra tão útil, haja visto que a mesma é mais recomendada para ambientes com baixa dinâmica, onde os obstáculos não se movem aleatoriamente, como no caso da mineração subterrânea ou gestão de armazéns [35] e muitas vezes, dependendo da abordagem, possui complicada implementação [20].

O método de campos potenciais também não se mostra muito adequado ao problema em questão, graças ao fato deste não ser diretamente aplicável à robótica móvel, tornando-se inapropriado a aplicações que envolvam planejamento global, como é o caso estudado. Graças a sua concepção de otimização do campo potencial, que emprega função decrescente, haverá grande possibilidade de o robô ficar preso em mínimos locais da função potencial, acarretando assim uma grande possibilidade de não alcançar o estado de objetivo, como visto em [20].

O método de decomposição por células possui uma melhor adequação para o problema estudado, pois atende com mais eficiência sistemas com alta dinâmicas e possui relativamente baixo custo computacional [20]. A forma de decomposição mais adequada e escolhida para a pesquisa é a fixa, devido à mesma requerer menos processamento para mapear o ambiente e sua implementação ser mais simples. As modificações sugeridas em [34] também serão adotadas na pesquisa, visando melhorar a representação dos obstáculos e minimizar o processamento computacional, bem como o método de varrimento por volume - *Sweep volume method*, para detecção de obstáculos. Tal procedimento foi escolhido devido a sua natureza de varrimento por iteração, a qual combina e aproveita os esforços já utilizados pelo método de decomposição, minimizando assim, o custo computacional necessário para tal atividade.

3. Trabalhos relacionados

Neste capitulo são apresentadas as principais estratégias recentes ou habitualmente empregadas na geração de trajetória e aplicadas ao planejamento de caminho na robótica móvel. Inicialmente, será detalhado o algoritmo clássico de busca e, posteriormente, serão apresentadas as mais atuais soluções destacadas pela literatura. Por fim, serão expostas as considerações finais.

3.1. Algoritmos de pesquisa em grafos

No capítulo anterior, foram descritas as principais abordagens para a identificação de obstáculos ou de configurações livres em um ambiente, bem como a apresentação dos mais conhecidos algoritmos de geração de caminho. Contudo, algumas das metodologias apresentadas não conseguem planejar sozinhas um caminho livre entre duas configurações, como é o caso da abordagem do *Roadmap* e da Decomposição por células. Essas apenas constroem um grafo que representa todas as posições livres do $C_{espaço}$ e, em razão de tal limitação, necessitam do auxílio de algoritmos de pesquisa em grafos para localizar a trajetória mais eficiente entre todas as configurações disponíveis no mesmo.

Desse modo, é possível perceber a importância de uma escolha acertada do algoritmo que irá executar tal tarefa, especialmente em ambientes dinâmicos, avaliando não apenas a capacidade do mesmo localizar ou não a solução pretendida, mas principalmente a sua eficiência na realização de tal tarefa, já que o tempo na robótica móvel é um fator crítico e não se pode ficar segundos à espera de uma solução. O presente capítulo tratará do tema em questão, apresentando as vantagens e limitações dos principais algoritmos de geração de trajetória disponíveis na literatura.

3.1.1. Investigação do estado da arte

Planejamento consiste em descobrir uma sequência de ações que interligue um estado inicial a um estado do objetivo desejado. No planejamento de caminho, estado é a denominação dada à localização do robô no espaço e o termo custo de transição representa o esforço necessário para que o robô possa se deslocar de uma posição a outra nesse ambiente. Um caminho é dito ideal, se a soma de seus custos de transição for a mínima em todos os trajetos possíveis, que partam do estado inicial, $S_{inicio,}$ até um estado meta,

S_{meta}. Um algoritmo de planeamento é dito como completo se esse localizar sempre uma trajetória, caso ela exista, em uma quantidade finita de tempo. E por fim, um algoritmo de planejamento é dito como ideal, se ele sempre encontrar um caminho ideal.

Sendo assim, um algoritmo clássico de geração de trajetória e muito utilizado com o método de decomposição por células é o A*. Esse foi descrito pela primeira vez em 1968 [36]. Este algoritmo emprega uma estimativa heurística que classifica cada nó pela estimação de melhor rota, calculada do nó inicial até o nó destino. Para isso, o mesmo utiliza da seguinte função custo:

$$F(n) = g(n) + h(n), \quad (1)$$

Onde:

g(n) = É o valor do custo do caminho percorrido desde a posição inicial até o nó atual *n*;

h(n) = É a função heurística que estimar o custo do caminho do nó atual *n* até o nó destino;

O mesmo é classificado como um algoritmo completo, garantindo assim que sempre encontrará um caminho entre a origem e o destino, caso esse exista, e também ideal, pois sempre encontrará o caminho com menor custo existente em um grafo. Sua aplicação vai desde aplicativos para localização de rotas entre duas configurações [37], resolução de problemas finitos, como resolução de quebra-cabeças, bem como a utilização em jogos que apresentam inteligência simulada [38]. O pseudocódigo do A* pode ser visto abaixo:

Pseudocódigo A* extraído de [32]

1. Adicionar o nó de partida a lista ABERTA
2. Repetir
    2.1. Escolher o n_{melhor} (nó melhor) de ABERTA tal que $f(n_{melhor}) \leq f(n)\ \forall n \in$ ABERTA
    2.2. Remover o n_{melhor} de ABERTA e adicionar à lista FECHADA
    2.3. Se n_{melhor} = nó final, terminar o algoritmo
    2.4. Para todos $n \in Star(n_{melhor})$ fazer o seguinte:
        2.4.1. Se ($n \notin$ ABERTA) e ($n \notin$ FECHADA) então adiciona o nó n a ABERTA

2.4.2. Se ($n \in$ ABERTA) então se $g(n_{melhor}) + c(n_{melhor},n) < g(n)$ então alterar o pai do nó n para n_{melhor}

2.4.3. Se ($n \in$ FECHADA) então se $g(n_{melhor}) + c(n_{melhor},n) < g(n)$ então alterar o pai do nó n para n_{melhor} e passar n para a lista ABERTA

3. Até que ABERTA esteja vazio

Onde:

ABERTA - Contém os nós que ainda não foram selecionados, mas podem vir a serem escolhidos;

FECHADA - Contém os nós já processados, isto é, nós que já saíram da lista aberta;

c(n1,n2) – É o custo computacional gasto para se deslocar do nó n1 para nó n2;

Star(n) – Representação do conjunto de nós adjacentes ao nó n;

O algoritmo A* certamente é um dos mais famosos planejadores de caminho existentes e, para algumas ocasiões específicas, ainda é um dos mais utilizados na atualidade. Contudo, ao longo do tempo, o mesmo ficou conhecido como um algoritmo computacionalmente lento, devido ao seu vasto tempo de processamento e pelos seus sérios problemas computacionais com elevada utilização de recurso de memória, já que o mesmo realiza diversos cálculos redundantes para garantir a trajetória ideal. Para tentar corrigir esses e outros problemas, no decorrer dos anos, pesquisas foram realizadas visando sempre alcançar uma solução que minimize a utilização desses recursos e que apresentem notáveis avanços. As mais destacadas na literatura são IDA* [39], MA* [40], D* [41], Focused D* [42], LPA [43], D* Lite [44], ARA* [45], Delayed D* [46], Field D* [47], AD* [48], ANA* [49], R* [50] e, recentemente, o LazyARA* [51].

O algoritmo Iterative Deepening A*, também conhecido como IDA* [39], é uma variação do algoritmo de busca clássico A* que utiliza os conceitos do algoritmo de pesquisa por aprofundamento iterativo para manter o uso de memória inferior ao do primeiro. O IDA* baseia-se em informação heurística para determinar o limite de iteração *K* que indicará até que nível o mesmo poderá descer para localizar o objetivo, diferentemente do algoritmo de aprofundamento iterativo que localiza seus objetivos transversalmente por iteração. Em cada iteração, o IDA* executará uma busca por profundidade, sob limite de custo e expandindo apenas os nós que tenham $f(n) < K$. Se o algoritmo não localizar o objetivo na iteração atual, incrementará K e realizará uma nova iteração. Esse comportamento é repetido até encontrar o objetivo ou esgotar um tempo predeterminado. Caso o algoritmo finalize sem localizar o objetivo, a solução retornada será os últimos nós visitados.

O algoritmo MA* [40], foi adaptado a partir do A* e consiste em retirar da lista aberta os nós que apresentam maior valor de *f(n)* e que, logicamente, são considerados menos promissores para localizar o objetivo; liberando assim mais espaço para novos nós. Quando é atingido um limite de nós na lista aberta e na lista fechada, o próximo nó a entrar na lista aberta faz com que o nó com o *f(n)* mais elevado seja retirado e seu antecessor receba sua informação atualizada. Em 2002, simplificaram o algoritmo MA* e utilizaram uma estrutura de dados mais ágil, resultando no SMA* [52], contudo, o conceito geral é igual ao MA*.

O algoritmo D* [41], também conhecido como A* dinâmico, quando iniciado executa a trajetória exatamente igual ao A* convencional. Contudo, quando ocorrem mudanças no ambiente, o mesmo recalcula sua nova trajetória de forma bem mais otimizada do que o algoritmo clássico, graças à grande vantagem de possuir um replanejamento aprimorado. O A*, quando necessita recalcular sua trajetória, sempre considera a posição inicial do planejamento e não a posição atual do robô, refazendo assim todos os cálculos já realizados. Já o D*, diferentemente, recalcula a mesma partindo da última posição conhecida do autômato, sem a necessidade de refazer todo o processo do zero.

O *Focused* D* [42] é uma versão melhorada do D*, o qual somente considera as informações dos estados mais relevantes e que contribuem diretamente para que o mesmo localize a trajetória almejada. Para identificar quais dos estados presentes no grafo são significativos, o *Focused* D* utiliza uma função heurística que avalia a importância de os mesmos de acordo com a variação abrupta dos esforços necessários para o robô alcançar uma nova posição que o leve em direção ao objetivo, destacando assim, as configurações que apresentem elevado aumento ou uma volumosa redução nos seus custos de transição, desde a última vez que foram inseridas na lista aberta do algoritmo.

O algoritmo *Lifelong Planning A** - (LPA) [43], também conhecido como A* incremental, une as vantagens do algoritmo *Dynamic* SWSF-FP [53] e do algoritmo de planejamento de caminho clássico A*. A princípio, o mesmo executa a trajetória analogamente ao A* convencional. Contudo, a cada incremento, reusa as informações de pesquisas anteriores para encontrar caminhos mais curtos e mais rápidos do que os dois algoritmos individualmente. Graças a essa redução, o LPA quando comparado ao A*, minimiza consideravelmente o número de nós que precisam ser examinados para localizar a melhor trajetória. Em contrapartida, o mesmo apresenta melhores resultados em ambientes estáticos.

O algoritmo D* *Lite* [44] é baseado no algoritmo de LPA. O mesmo possui a mesma estratégia de navegação do D*, mas é implementado de maneira distinta. Esse, utiliza uma heurística de busca incremental para localizar a melhor trajetória, reaproveitando as informações das pesquisas anteriores e reajustando as mesmas enquanto o robô obtém novas informações sobre o ambiente. É um algoritmo de rápido replanejamento e é mais indicado para ambientes desconhecidos e dinâmicos. O mesmo tem a característica de construir inversamente a sua trajetória, ou seja, do destino para o início e é considerado um algoritmo muito mais simples de implementar e que apresenta melhores resultados do que o D*, como visto em [54].

O algoritmo ARA* [45] é um tipo de algoritmo conhecido como *anytime* ou com otimização descontínua, o qual se concentra em gerar rapidamente resultados que não necessariamente são os ideais, otimizando-os enquanto existir tempo disponível. O ARA* utiliza em sua função heurística uma variável ponderada ε para determinar a ordem de visita das células. Quanto maior for o valor dessa variável, mais rápido o algoritmo encontrará uma solução. A solução gerada inicialmente pode ser melhorada, reduzindo o valor de ε em cada iteração e recalculando uma nova solução. A mesma será garantida ótima se esse valor chegar ao valor mínimo 1. O ARA* reutiliza as informações das pesquisas anteriores enquanto melhora os seus resultados. Existem vários outros algoritmos baseados nesse conceito, os quais podem ser conferidos em [55], [56], [57], [58] e [59].

O algoritmo *Delayed* D* [46] é uma versão modificada do D* *Lite* que, ao receber novas informações do ambiente, retarda tanto quanto possível a propagação do aumento de custos para que os estados que tiveram diminuição destes sejam processados imediatamente. Dessa forma, apenas os novos valores relevantes para as localizações imediatas da nova trajetória serão computadas, economizando recursos computacionais. Os demais estados que foram ignorados serão incluídos em uma lista de prioridades que atualizará todos os seus custos de uma vez, aproveitando uma única iteração.

Em [54] e [60] foram realizados alguns estudos que relacionam os algoritmos de reajustamento de trajetória: D*, D* *Lite*, *Focused* D* e *Delayed* D* com o algoritmo de geração de trajetória clássico A*. Em [54] foi demonstrado que, para a maioria das tarefas de navegação, o D* *lite* é mais eficiente do que o D*. Em [60], o *Delayded* D* foi considerado mais eficiente do que D* *lite* apenas para cenários onde a distância entre o ponto inicial do grafo e o destino eram grandes e as mudanças ocorridas eram aleatórias. Já em cenários em que as alterações ocorrem próxima a localização do robô, o algoritmo D* *Lite* mostrou-se bem mais eficaz do que o algoritmo A* como visto em [60]. Por fim, os casos que apresentam muitas alterações, ou as mesmas são muito próximas ao destino, esses algoritmos deixam de ser eficazes, sendo mais aconselhado refazer os cálculos do zero, como apresentado em [60].

O Field D* [47], é uma variação do D* *Lite* que utiliza interpolação linear para estimar o menor caminho entre duas células, produzindo trajetórias de baixo custo que evitam em suas rotas que o robô rotacione desnecessariamente sobre seu eixo, para que possa alcançar alguma configuração. Diferente da maioria dos planejadores que limita artificialmente o movimento do agente a um pequeno conjunto de ângulos, como por

exemplo 30º, 45º e 90º enquanto transitam entre as células. O *Field* D* calcula a trajetória do robô por meio de estimativas de custo de posições arbitrárias dentro de cada célula e não apenas em seus centros, escolhendo a menor trajetória que interligue S_{inicio} até S_{meta} e que possibilite ao robô rotacionar o menor número de vezes possível. De acordo com seus idealizadores, os caminhos produzidos são ideais e tal abordagem é mais bem recomendada para ambientes dinâmicos.

O algoritmo *Anytime Dinamic* A* [48], também conhecido como AD*, é uma combinação do algoritmo iterativo D* *Lite* [44] com o algoritmo descontínuo ARA* [45]. O mesmo tem o objetivo de ajustar a qualidade de sua solução com base no tempo de busca disponível, reutilizando em cada iteração, os esforços da pesquisa anterior. Tal algoritmo ainda evita o recálculo da trajetória do zero, quando as novas informações sobre o ambiente são recebidas, apenas atualizando e reajustando a mesma. O resultado é uma abordagem que combina os benefícios de planejadores de otimização descontínuos com planejadores incrementais para fornecer hábeis soluções aos complexos problemas de pesquisas dinâmicas.

O algoritmo *Anytime Nonparametric* A* [49], também conhecido como ANA*, utiliza técnicas de inferência estatística para maximizar seus resultados e planejar a melhor rota. Sua utilização exige que sejam satisfeitos alguns requisitos como: distribuição normal, variância homogênea e intervalos contínuos e iguais de dados [61]. O ANA* é recomendado nos casos onde não se tem informações dos parâmetros de entrada da função. O mesmo adaptativamente vai reduzindo a variável ε, adequando-se a pesquisa de forma eficiente enquanto melhora a qualidade da trajetória. O algoritmo não expande o nó com o *f(s)* mínimo, como é comum nos algoritmos da família A. Em vez disso, expande os nós com o valor de *e(s)* máximo.

O algoritmo R* [50] é um tipo de algoritmo aleatorizado, que depende menos da qualidade da função heurística. Esse evita os mínimos locais, realizando todo o planejamento do problema através uma série de pesquisas ponderadas de curto alcance, para um estado objetivo escolhido aleatoriamente. Como medida para otimizar o seu funcionamento, o próprio evita as buscas que demoram a apresentar resultados, tentando construir uma solução global, usando apenas as soluções das pesquisas localizadas de modo ágil e simplificado e que empregam a mínima quantidade de recursos possível. Para delimitar se uma busca fornece resultados de maneira simplificada, podem ser utilizadas diferentes heurísticas para estabelecer este limite, tal como o tempo de busca ou o número de expansões de estado. O mesmo, através da marcação dos estados pesquisados, sinaliza

as buscas que demoram a apresentar resultados como estados a evitar. De acordo com os autores, o R* ainda emprega com mais eficiência a utilização de memória, desalocando memória após finalizar cada uma de suas buscas.

O algoritmo LazyARA* [51], utiliza uma abordagem conhecida como Grafo de experiências para armazenar e gerenciar uma grande variedade de informações das buscas já realizadas anteriormente, evitando assim, sempre que possível, o replanejamento enquanto acelera a localização do objetivo, mesmo em cenários com desordem aleatória. O mesmo é capaz de reduzir o custo computacional reutilizando apenas partes relevantes das experiências prévia coletadas, diferenrenciando-as e aproveitando apenas as que forem significativas para a tarefa de planejamento. Além disso, o mesmo utiliza um mecanismo conhecido como atalhos para introduzir um novo nó sucessor às buscas e reduzir o número de expansões necessárias para localizar o objetivo. O mesmo ainda emprega uma busca por heurísticas ponderada que, de acordo com os autores, fornece garantias de integridade e um limiar subótimo no que diz respeito ao caminho ideal, mesmo em ambientes desafiadores.

A Tabela 3 apresenta o resumo dos algoritmos estudados, bem como suas principais vantagens.

Tabela 3: Principais algoritmos da família A*

Algoritmo	Ano	Vantagem
IDA* [39]	1985	Grande economia de memória, quando comparado ao A*.
MA* [40]	1989	Garante soluções admissíveis dentro das limitações de memória especificada.
D* [41]	1994	Repara a busca rapidamente à medida que as informações chegam e garante que a solução será ideal.
Focused D* [42]	1995	Reduzir o tempo total de replanejamento do D*, pois calcula apenas o custo dos estados relevantes a trajetória final
LPA*[43]	2001	Minimiza o número de nós para localizar a trajetória, comparado com o A*, e diminuir o custo de replanejamento da trajetória utilizando informações de pesquisas anteriores*
D* Lite[44]	2002	Mais simples de implementar e apresenta melhores resultados que o D*, sua trajetória é realizada inversamente
ARA* [45]	2003	Constrói soluções sub-ótimas rapidamente e vai melhorando-as enquanto houver tempo
Delayed D* [46]	2005	Retarda a propagação de aumentos de custos, lidando com todos os estados de uma vez. Reduzindo assim o custo computacional
Field D* [47]	2005	Utiliza de interpolação linear para estimar o custo dos caminhos exatos para cada célula e produzir trajetórias de baixo custo evitando que o robô rotacione sem necessidade
AD* [48]	2005	Une as abordagens iterativas de otimização constante com as soluções de replanejamento continuo. Ajustando a qualidade da solução com base no tempo disponível.
R* [50]	2008	Depende menos da qualidade da função heurística, evita mínimo local, minimiza a utilização de recursos computacionais, fornece garantia probabilistas de sub-otimalidade da solução global.
ANA* [49]	2011	Não requer definição de parâmetros de entrada, adapta-se a pesquisa de forma eficiente enquanto melhora a sua qualidade de trajetória.
LazyARA*[51]	2015	Acelera a localização do objetivo reutilizando apenas informações relevantes das buscas anteriores, emprega um mecanismo conhecido como atalho para reduzir o número de expansões e alcançar o objetivo com menos esforço.

3.1.2. Considerações finais

Durante este capítulo foram apresentados os principais trabalhos relacionados aos algoritmos de geração de trajetória disponíveis na literatura, desde dos anos 80 até a atualidade, bem como foi discutido conceitos técnicos relacionados aos mesmos, como a terminologia referente ao tipo e modo do planejamento adotado. Também foram descritas as principais vantagens e limitações dos algoritmos de planejamentos mais relevantes destacados pela literatura. As oportunidades de investigação descritas nesta subseção ajudaram a definir os objetivos de pesquisa e serão responsáveis pelo direcionamento do processo científico descrito no capítulo a seguir.

4. Metodologia

Desde os primórdios da civilização, a ideia de idealizar sistemas autônomos que imitem o comportamento humano acompanha as civilizações. No século IV antes de Cristo, os egípcios já fabricavam versões rudimentares de homens mecânicos ou autômatos com características de estátuas e com articulações móveis. Povos helênicos na antiguidade também arquitetavam estátuas providas de tubos falantes e a partir dos anos 500 a.C. surgiram as marionetes acionadas por cordas e polias. Hoje, a ambição da robótica visa produzir robôs independentes que aceitem descrições de serviços em alto nível e as concretizem sem a influência do operador, tais ações como as de vigilância, limpeza em geral, transporte de materiais, exploração espacial, limpeza de acidentes nucleares, mineração e/ou navegação em estradas. Apesar do desenvolvimento desses sistemas ser considerado uma tarefa altamente complexa, estas metas vêm aos poucos sendo alcançada com o emprego de diferentes técnicas de planejamento, percepção, localização e movimento [9].

Um robô móvel autônomo deve ser preparado para perceber o ambiente à sua volta, tomar decisões sobre a melhor ação a ser executada e realizá-la com o mínimo erro possível. Se a tarefa a ser concretizada for navegar de um ponto a outro do ambiente, o agente deve ser capaz de detectar obstáculos próximos, desviar-se deles e alcançar com segurança o ponto determinado como sendo seu objetivo. Uma das principais questões relacionados ao tema é descobrir a maneira mais eficiente para realizar essa navegação com máxima segurança e despendendo a menor quantidade de tempo e de recursos computacionais possível.

Sendo assim, para auxiliar o leito no entendimento do trabalho, será retomado, posteriormente, o problema norteador da pesquisa e a hipótese que visa solucioná-lo. Logo em seguida dos mesmos serão apresentadas as informações de quatro estratégias algorítmicas selecionadas para os estudos comparativos aprofundados, bem como os detalhes dos ambientes utilizados, direções de buscas empregadas e informações gerais das análises efetuadas. Por fim, serão apresentados minucias da simplificação heurística obtida e de seus resultados parciais que visam utilizar a menor quantidade de tempo e recursos computacionais possíveis.

4.1.Problema norteador da pesquisa

No capítulo anterior foram descritas as principais abordagens algorítmicas utilizadas para o planejamento de trajetórias. Em algumas dessas, o $C_{espaço}$ é reduzido a grafos: C_{Livre} – contendo todas as posições acessíveis no ambiente e o $C_{Ocupado}$ – contendo todas as configurações ocupadas no mesmo. Nessa categoria de abordagens estão o *Roadmap* e a Decomposição em células. Para localizar a melhor trajetória que interligue duas posições presente em um grafo, recorre-se a uma outra estratégia, os algoritmos de pesquisar em grafo, que visam localizar o caminho mais promissor que interligue as posições desejadas. Sendo assim, é muito importante realizar uma escolha acertada do algoritmo que vai executar essa tarefa, não só considerando a capacidade de o mesmo descobrir ou não a solução, mas também analisando a sua eficiência, pois no mundo da robótica o tempo é um fator crítico e não se pode ficar segundos à espera de uma solução.

Para resolver problemas grandes e complexos em um tempo satisfatório é muito comum utilizar a estratégia de busca heurística. A função heurística melhora a eficiência do processo de localização do objetivo, indicando entre os vários caminhos, o mais vantajoso. Dentre os principais procedimentos de buscas em grafos que utilizam função heurística, destacam-se os algoritmos da família A*.

Nesse contexto, a pesquisa em questão visa responder o seguinte questionamento: *Como reduzir o tempo de processamento e os recursos computacionais usados pelos algoritmos da família A* e aplicados à robótica móvel?*

4.2. Recapitulação da Hipótese

Como visto na seção 3.1, há inúmeros algoritmos de pesquisas em grafos que utilizam diferentes heurísticas e podem ser empregados no intuito de planejar um caminho de forma eficiente e aplicável à robótica móvel. Diante de tantas opções é sugerida uma comparação entre as principais abordagens destacadas na literatura no intuito de identificar as melhores e mais eficazes estratégias algorítmicas que atentem a ambientes conhecidos e/ou desconhecido com alta dinâmica.

Para isso, serão avaliados os algoritmos mais atuais de planejamento de caminho através de comparações entre suas particularidades, funções custos e direções de buscas. Em seguida, será verificado se esses resultados poderão ser aperfeiçoados através de ajustes em suas respectivas funções custo.

Dessa forma, pode-se então formular a seguinte hipótese de pesquisa: *É possível reduzir o tempo de processamento, o consumo computacional e melhorar a eficiência dos principais algoritmos da família A*, através da minimização da importância da variável g(s) - responsável em armazenar, calcular e atualizar informações do custo da rota já realizadas pelo autômato a cada iteração, nas respectivas funções custos dos algoritmos avaliados.*

4.3.Detalhamento Metodológico

Esta seção descreve os procedimentos metodológicos realizados para que a hipótese levantada neste projeto de pesquisa seja verificada.

4.3.1. Análise dos algoritmos selecionados

A tarefa de identificação de pesquisas para a revisão do estado da arte foi realizada durante o primeiro ano do mestrado. Nela, foram analisados os principais trabalhos relacionados ao planejamento de caminho aplicados à ambientes estáticos e dinâmicos. Porém, assume-se a necessidade de investigação constante de novos projetos relevantes a fim de acompanhar os progressos paralelos na área de pesquisa no decorrer da investigação.

Durante a revisão foram identificados os melhores e mais bem avaliados planejadores de caminhos baseados na família A* de acordo com as bibliografias técnicas especializadas, desde os anos 80 até a atualidade. Ao longo da mesma, foi observado que cada nova versão desenvolvida tinha como objetivo principal corrigir problemas

identificados nas versões anteriores, tais como alto uso de memória, lentidão, redundâncias, travamentos e etc. Dessa forma, resolveu-se selecionar as 4 versões mais atuais e consideradas completas dos planejadores disponíveis, afim de estudá-las com mais afinco e posteriormente compará-las no intuito de averiguar qual metodologia seria mais oportuna e eficiente para identificar a solução do problema proposto.

Os algoritmos selecionados foram os planejadores AD*[48], R*[50], ANA*[49] e LazyARA* [51]. Os mesmos utilizam de abordagens iterativas e descontinua, de inferência estatística, de técnicas randomizadas e grafos de experiência, respectivamente, para planejar e construir suas trajetórias. Foram utilizadas as duas possíveis direções de buscas abordadas pela literatura, a *forward*, que é a direção padrão, onde o S_{inicio} localiza-se onde o robô está situado e o S_{meta} posicionado onde o algoritmo deseja alcançar; e a direção backward: onde o S_{inicio} localiza-se no estado final e o S_{meta} será disposto na posição atual do autômato - elaborando uma pesquisa de trás para frente, como mostra a Figura 12. Os resultados comparativos entre estes algoritmos podem ser conferidos na Seção 5 que aborda os resultados da pesquisa.

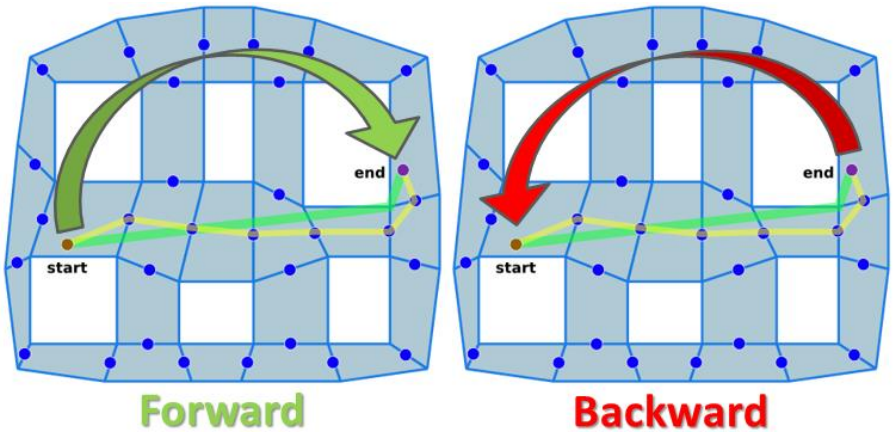


Figura 12[13]: Direções de buscas utilizadas nas pesquisas

4.3.2. Teste comparativos entre os algoritmos selecionados

Inicialmente, foram realizados testes comparativos entre a versão clássica do A* e os 4 algoritmos selecionados em um ambiente estático, no intuito de conhecer detalhes dos comportamentos dos algoritmos em um espaço controlado e, posteriormente, realizar testes mais complexos em um ambiente altamente dinâmico. Os códigos dos planejadores utilizados foram disponibilizados pelo professor *Maxim Likhachev* em sua página pessoal

[13] Imagem extraída e adaptada da página: "*Boost Graph Library astar and navigation mesh*"
< http://stackoverflow.com/questions/19528530/boost-graph-library-astar-and-navigation-mesh >

através da biblioteca SBPL [62], o simulador estático foi disponibilizado pelo professor *Patrick Lester* [63] e o simulador dinâmico pelo professor Paulo Costa [64]. A Figura 13 demonstra uma execução de um dos testes realizados no ambiente estático.

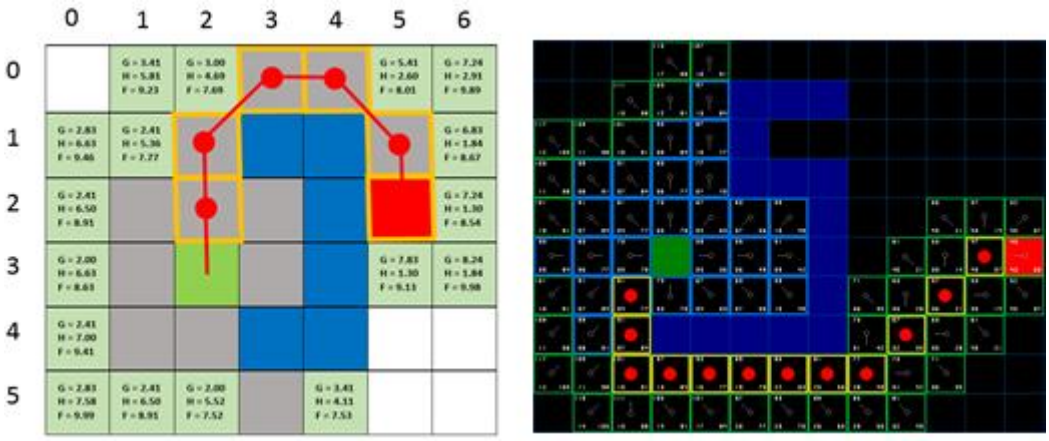


Figura 13: Apresentação dos detalhes de uma trajetória gerada em um ambiente estático

Através dos testes estáticos, foi possível observar detalhes na execução passo a passo de cada algoritmo. Nessas, foram analisadas o tempo de execução dos algoritmos, o uso dos recursos computacionais, a qualidade dos caminhos escolhidos, bem como a eficiência das abordagens utilizadas.

4.3.3. Testes práticos e modificação na função heurística

Através das simulações estáticas e os estudos de diferentes versões dos algoritmos da família A*, foi possível observar algumas particularidades na execução de suas funções heurísticas utilizadas para localizar os objetivos. Entre elas, o fato de todos os algoritmos recentes herdarem a função custo clássica do A*, $f(n) = g(n) + h(n)$, para promover um estado ou descartá-lo de um espaço de busca. Contudo, foi percebido que a ação de armazenar, calcular e atualizar informações dos custos utilizados pelo robô para se deslocar de um estado inicial até o estado atual, através da variável $g(n)$, está cada vez mais subutilizada nas versões atuais dos algoritmos de planejamento que lidam com ambientes complexos e sobretudo dinâmicos. Pois, o foco primordial do planejamento é possibilitar que o robô alcance um determinado objetivo no menor tempo e com a máxima precisão possível. Contudo, na equação clássica desta função a carga de processamento necessária para localizar o objetivo é fragmentado em dois caminhos que empregam funções bem distintas. Cerca de 50% dela é gasta para armazenar e manter atualizado a descrição do passado do algoritmo, ou seja: o custo da rota já realizadas pelo autômato a

cada iteração e, a outra metade, é responsável em fazer o robô de fato alcançar seu objetivo. A desvantagem dessa abordagem é que a medida que o ambiente se torna mais complexo, o qual já é realidade dos espaços de trabalho reais na robótica, mais essa ação atrasará o algoritmo a alcançar seu objetivo.

Em um ambiente dinâmico, essa ação de fornecer importância iguais entre essas duas variáveis traz um prejuízo exponencialmente maior, pois nesse contexto, o espaço de busca circunvizinho e os obstáculos presentes no mesmo se modificam a cada segundo, o que inviabiliza a ação de guardar e atualizar essas informações que apenas desperdiça memória e atrasa a localização do objetivo.

Para contornar esse problema de subutilização da variável *g(n),* foi sugerido a minimização da importância da mesma na equação das respectivas funções custo utilizados pelos algoritmos selecionados, visando verificar se tal modificação melhoraria o desempenho geral destes. Uma ideia parecida foi sugerida em 1970 no algoritmo WA*[65]. O mesmo utiliza uma variável ponderada para minimizar a importância da variável *h(n).* Contudo, tal ideia não apresentou bons resultados.

Para facilitar a análise do comportamento que essa modificação resultaria, foi utilizado o planejador A* como teste rápido para visualização dos efeitos dessa contribuição, visto que o A* foi o algoritmo clássico criado logo após o algoritmo guloso e que implementou a heurística *h(n).*

Dessa forma, o objetivo deste primeiro conjunto de simulações foi observar as diferenças de comportamento entre o algoritmo A* normal e o A* com as alterações propostas na função custo: $F(n) = w*g(n) + h(n)$ em um ambiente altamente dinâmico, com obstáculos móveis que se movem aleatoriamente e em velocidades que, para alguns dos obstáculos, pode ser maior do que a velocidade do próprio robô. Ambas as versões foram implementadas utilizando a decomposição por células e empregando, em suas respectivas heurística, a função euclidiana para antever o deslocamento futuro entre o ponto atual e o ponto destino. Em cada experimento foi minimizada a importância da variável *g(n)* através do uso de variáveis ponderadas, focando assim os recursos na heurística *h(n),* que tentava localizar o estado objetivo.

Durante cada conjunto de testes eram executadas 5 verificações com o algoritmo A* normal e 5 com o algoritmo A* com alteração proposta, em cada avaliação os algoritmos eram analisados de acordo com os seguintes paramentos: tempo de planejamento para localizar o objetivo, número de iterações e quantidade de colisões. Ao final de cada conjunto experimental eram coletados os dados e decrescido o valor 0,1 na

importância da variável ponderada *W* e então os testes eram refeitos considerando esse novo valor. Viu-se que a cada teste realizado, os resultados obtidos eram melhores do que os resultados anteriores e que chegou a um ganho máximo de eficiência quando essa variável foi de fato zerada, ou seja, utilizando como heurística apenas F(s) = h(s).

A otimização observada foi de cerca de 87% no tempo necessário para localizar o objetivo, 65% no número de colisões, uma suave melhora no número de iterações e consequentemente do uso de memória. Os resumos destes testes podem ser vistos na Tabela 4 e os seus detalhes no Apêndice A dessa dissertação.

Tabela 4: Resumo das diferentes versões da função custo aplicada ao algoritmo A*

Parâmetros	**F = G + H**	**F = H**
Número de iterações	600,6	523
Tempo de planejamento (ms)	0,686	0,089
Número de colisões	4	1,4

Vale a pena ressaltar que apesar da similaridade da função custo resultante com o algoritmo guloso [66], os mesmos apresentam uma grande diferença conceitual. O algoritmo guloso surgiu antes do A* e é famoso por apresentar uma grande desvantagem que é a possibilidade de não localizar o objetivo, mesmo que ele exista, devido a ser altamente propenso a ficar preso a mínimos locais que são frequentemente gerados por essa abordagem. Para corrigir esse problema, foi criado em 1968 o algoritmo A* que utiliza listas acesso, as listas aberta e fechada, para controlar a promoção dos estados e armazenar as informações dos nós já visitados. Desse modo, a versão da heurística *f(s) = h(s)* beneficia-se das vantagens de evitar mínimos locais, devido as listas abertas e fechadas herdadas do A*, bem como consegue localizar a trajetória mais rápido, pois executa menos operações em seus ciclos e se resguarda das desvantagens já conhecidas desse algoritmo que é o recálculo consecutivo dos estados já visitados, devido à forte influência da variável *g(n).*

Com o resultado dos testes realizados no algoritmo A* em um ambiente dinâmico, percebeu-se uma ligeira melhora no número de iterações realizadas pelo algoritmo, a minimização no número de colisões e principalmente no tempo necessário para localizar o objetivo, maximizando assim a eficiência do mesmo. Desse modo, resolveu-se verificar se essa modificação também melhoraria os resultados dos atuais e mais eficientes algoritmos da família A* destacados no início da pesquisa. Tais resultados podem ser conferidos na seção a seguir.

5. Resultados

Os resultados que se seguem apresentam uma comparação entre os quatro melhores e mais atuais algoritmos de planejamento de caminho destacados pela literatura. A mesma foi realizada inicialmente em três ambientes estáticos e distintos, a fim de verificar qual das abordagens analisadas (iterativas e descontínua, inferência estatística, técnicas randomizadas ou grafo de experiências) se mostram mais eficiente de acordo com os seguintes parâmetros: quantidade de estados expandidos, custo computacional da solução e o tempo total para localizar a trajetória. É importante salientar que dentre os parâmetros mencionados o mais significativo é o tempo necessário para localizar o objetivo, visto que na robótica móvel o menor tempo de resposta é imprescindível. Esses mesmos algoritmos serão posteriormente avaliados em um ambiente dinâmico. Tais avaliações serão analisadas comparando o algoritmo original e o mesmo com suas funções custo modificadas a fim de descobrir se houve melhoria significativa nos resultados dos algoritmos mais atuais.

A configuração dos ambientes estáticos utilizados nesta primeira análise foram: um ambiente trivial e direto (14x14) – Env1, um ambiente extenso e complexo (99x999) – Env2 e por fim, um ambiente simples, mas apresentando diversos casos de mínimos locais (14x14) – Env3, como é possível observar na Figura 14. Os obstáculos nesses mapas estão representados pela cor verde e os espaços livres pela cor branca

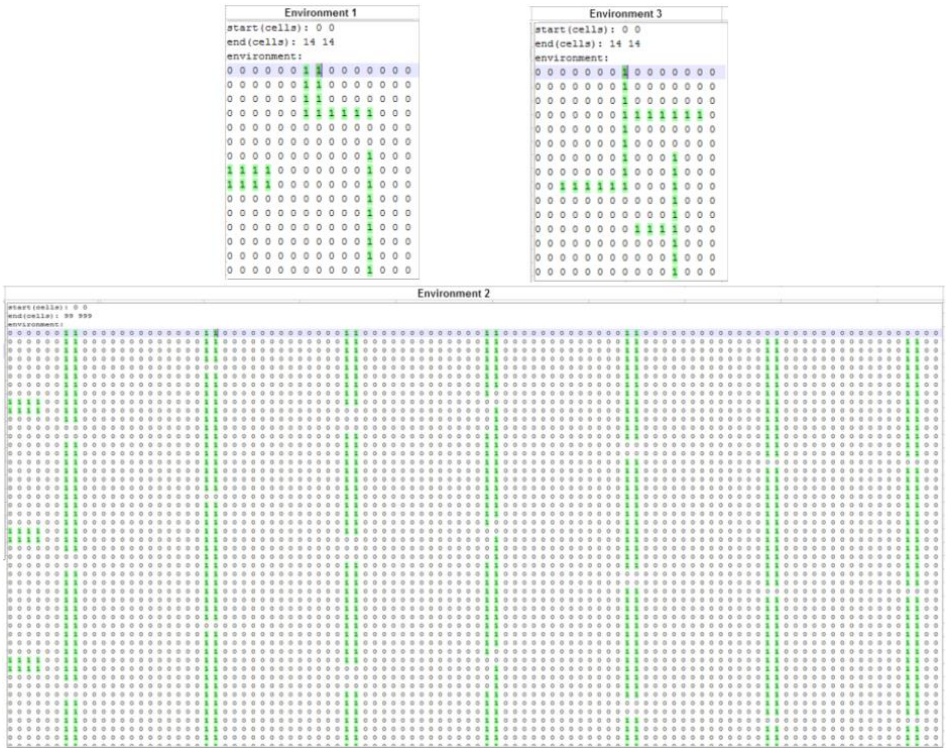


Figura 14: Ambientes utilizados pelos algoritmos

Para cada um dos ambientes acima mencionados, foram realizados cinco testes avaliativos considerando cada um dos algoritmos estudados; em seguida utilizou-se uma média aritmética simples para computar os resultados de cada grupo. Nesses testes foram consideradas as duas principais direções de buscas disponíveis na literatura, a *forward* e a *backward*. Demais detalhes podem ser verificados no Apêndice B dessa dissertação.

Já no ambiente dinâmico, todas as simulações foram realizadas através do software *SimTwo* [67], desenvolvido pelo Professor Paulo Costa, PhD, da FEUP. Este, segundo o autor, utiliza várias bibliotecas *Open sources*, entre elas o motor de simulação de corpos rígidos (ODE) [68] para simular robôs móveis com diferentes configurações e com comportamentos perfeitamente realistas.

O simulador possibilita a criação de vários robôs virtuais com todas características de um robô real, como peso, tamanho, consumo e comportamento cinemático. O mesmo também permite criar diferentes obstáculos virtuais móveis e podem ser utilizados para avaliar a capacidade de replanejamento do agente em um ambiente desconhecido e dinâmicos.

Em [69], é apresentado detalhes da rigorosa parametrização no software, o que torna esta plataforma um ambiente de simulação realista e ideal para testes com algoritmos de planejamento de caminho.

Nos testes apresentados a seguir foram utilizados um agente virtual omnidirecional *turtlebot2*, que possuía vasta liberdade de deslocamento e conseguia se mover em qualquer direção. Esse foi inserido em um ambiente lotado com 12 obstáculos dinâmicos, representados por esferas, que possuíam distintos tamanhos, diferentes pesos e se deslocavam com dissemelhantes velocidades. Os testes visaram analisar a capacidade de planejamento e replanejamento dos algoritmos analisados, bem como mensurar o tempo gasto para localizar a trajetória, o número de iterações efetuadas e a quantidade de colisões geradas no processo. A Figura 15 apresenta o *screenshot* do ambiente dinâmico. Demais detalhes dos testes podem ser acompanhados no Apêndice D dessa dissertação.

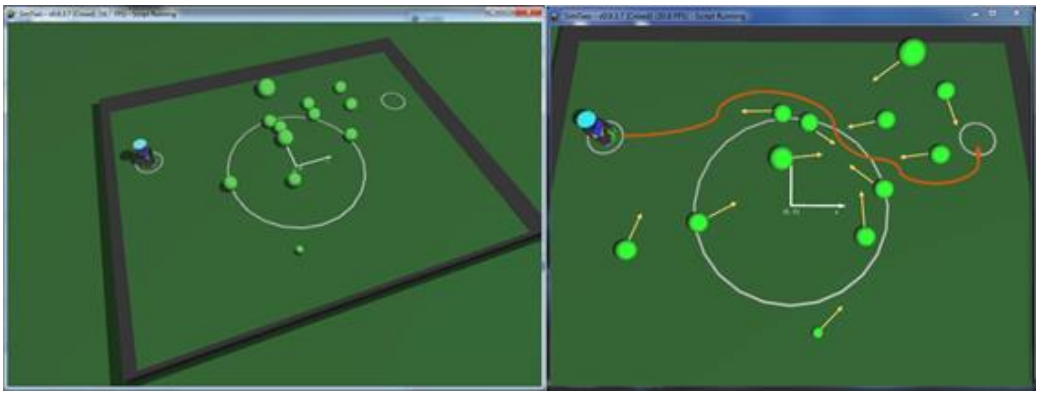


Figura 15: Simulador *SimTwo* executando uma trajetória gerada

É importante mencionar que nas simulações dinâmicas os movimentos dos obstáculos são ajustados para serem semialeatórios, com todos os objetos - robô e obstáculos - iniciando na mesma posição, mas tomando comportamento desconhecidos e/ou aleatórios à medida que se movem e/ou começam a se chocar. Seus movimentos são definidos através de um processo algorítmico simples no *SimTwo*, algoritmo 1, onde os valores de cada uma das esferas são carregados e uma força é aplicada sobre cada uma delas, possibilitando assim que as mesmas se movimentem com uma velocidade constante.

Algoritmo 1: velocidades das esferas

Algorithm set Sphere Speed

1: **If** ∃ Sphere **then**
2: $NexPosition_x = Initial_x + Radius.cos(Speed.(t+\delta))$
3: $NexPosition_y = Initial_y + Radius.sen(Speed.(t+\delta))$
4: **Set** Force of ith-Sphere with
5: $V_x = 100*(NexPosition_x - Position_x)$
6: $V_y = 100*(NexPosition_y - Position_y)$
7: $V_z = 0$
8: **end**
9: **end**

A velocidade de deslocamento do robô também é contínua e o mesmo se move a 0,8 m/s.

Para cada um dos algoritmos analisados foram realizados cinco testes avaliativos considerando as duas principais direções buscas; em seguida, utilizou-se uma média aritmética simples para computar os resultados de cada grupo. Todos os testes foram realizados utilizando um computador com processador i5 e 8 MB de RAM. A seguir são apresentados os resultados obtidos durante os experimentos com essas configurações.

5.1. Ambiente estático

Nessa subseção serão apresentados os resultados obtidos da comparação entre os quatro algoritmos de planejamento de caminho escolhidos em um ambiente estático. Os seguintes critérios serão considerados para mensurar a qualidade da trajetória gerada: número de estados expandidos, custo da solução e o tempo total para localizar a trajetória. Os resultados obtidos com a direção de busca *forward* serão representados nas colunas com as setas verdes e os resultados com a direção *backward* com as setas vermelhas. O melhor desempenho obtido na avaliação, considerando os quatro ambientes será destacado na tabela pela cor verde escuro e o pior pela cor vermelha. Todos os resultados obtidos na avaliação podem ser conferidos na Tabela 5.

Tabela 5: Resultados comparativos entre os algoritmos AD*, ANA*, R* e LazyARA*

		Ambiente 1	Ambiente 2	Ambiente 3	Ambiente 1	Ambiente 2	Ambiente 3
ADStar	Células Expandidas	169	82.387	319	60	92.389	171
	Tempo (seg)	0.016	4.203	0.027	0.009	4.778	0.020
	Custo da Solução	23.898	1.049.948	33.070	23.898	1.049.948	33.070
ANAStar	Células Expandidas	286	3.536.593	391	70	3.918.446	281
	Tempo (seg)	0.021	100.024	0.029	0.006	100.025	0.018
	Custo da Solução	23.898	1.071.706	33.070	23.898	1.112.316	33.070
Rstar	Células Expandidas	2	11	2	5	208	5
	Tempo (seg)	0.012	0.410	0.038	0.030	0.712	0.061
	Custo da Solução	24.726	1.146.596	33.898	24.484	1.135.844	34.726
LazyARA*	Células Expandidas	286	7.871.114	391	70	10.697.291	281
	Tempo (seg)	0.031	7.854	0.031	0.015	9.309	0.017
	Custo da Solução	23.898	1.010.925	33.070	23.898	1.049.948	33.070

É interessante observar que os algoritmos ANA*, AD* e LazyARA*, com direção de busca para trás, apresentaram resultados ligeiramente melhores do que as suas versões com a direção de busca para frente, especialmente nos ambientes 1 e 3 (menores). Em contraste, o algoritmo de busca do R* teve desempenho muito melhor através da busca direta.

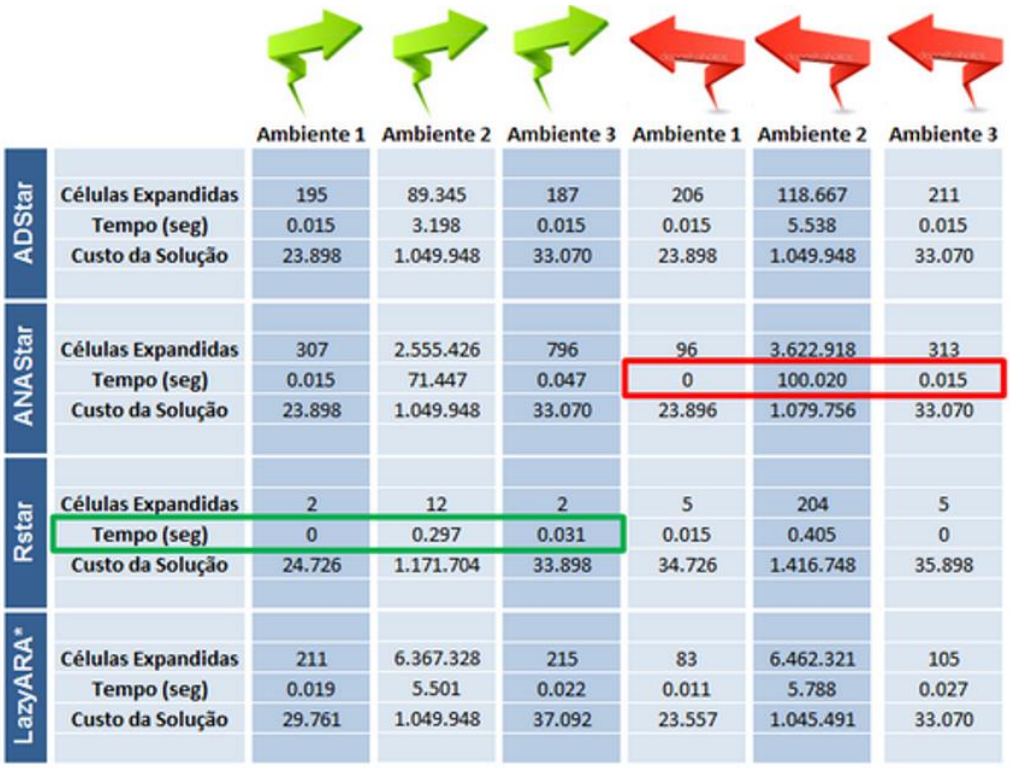
Percebe-se que o número de expansão das células por cada chamada foi consideravelmente menor no algoritmo R*, que ainda conquistou o tempo mínimo de planejamento no ambiente 2 (o maior ambiente). Isso ocorreu tanto nas pesquisas para frente como para trás. Nos ambientes 1 e 3 (os menores) com pesquisa inversa, o algoritmo do ANA* teve o menor tempo de planejamento geral, mesmo com R* ainda realizando um menor número de expansões de célula. Já na busca direta, o algoritmo R* teve melhores resultados com um menor número de expansões e tempo de planejamento. O ANA*, com versão de busca *backward*, apresentou o maior tempo para localizar o objetivo no ambiente mais complexo. Já o LazyARA*, com versão de busca *backward*, teve uma ligeira redução no tempo e nas expansões de células quando comparado com sua versão *forward*. O custo da solução geral foi quase o mesmo em todos os casos.

Sendo assim, é possível inferir que em uma execução de planejamento real, onde a presença de ambientes extensos e com diversos obstáculos é o caso mais comum na robótica, o algoritmo R* obteve o desempenho mais significativo que os demais, especialmente quando se refere ao tempo de planejamento. Tal característica também é muito relevante para ambientes altamente dinâmicos, onde é comum a presença de obstáculos móveis, pois um algoritmo com um baixo tempo de planejamento pode auxiliar imensamente o controlador de trajetória a evitar que o robô colida com esses obstáculos. Demais detalhes dos testes podem ser acompanhados no Apêndice B dessa dissertação.

5.1.1. Ambiente estático com função custo modificada

Nessa subseção, serão apresentados os resultados obtidos da comparação entre os quatro algoritmos de planejamento de caminho escolhidos, utilização a nova função custo otimizada, em um ambiente estático. Serão considerados os mesmos critérios e direções de buscas da pesquisa anterior. O melhor desempenho obtido na avaliação, considerando os quatro ambientes será novamente destacado na tabela pela cor verde escuro e o pior pela cor vermelha. Todos os resultados obtidos na avaliação podem ser conferidos na Tabela 6.

Tabela 6: Resultados comparativos entre os algoritmos AD*, ANA*, R* e LazyARA* com a nova função custo.



		Ambiente 1	Ambiente 2	Ambiente 3	Ambiente 1	Ambiente 2	Ambiente 3
ADStar	Células Expandidas	195	89.345	187	206	118.667	211
	Tempo (seg)	0.015	3.198	0.015	0.015	5.538	0.015
	Custo da Solução	23.898	1.049.948	33.070	23.898	1.049.948	33.070
ANAStar	Células Expandidas	307	2.555.426	796	96	3.622.918	313
	Tempo (seg)	0.015	71.447	0.047	0	100.020	0.015
	Custo da Solução	23.898	1.049.948	33.070	23.896	1.079.756	33.070
Rstar	Células Expandidas	2	12	2	5	204	5
	Tempo (seg)	0	0.297	0.031	0.015	0.405	0
	Custo da Solução	24.726	1.171.704	33.898	34.726	1.416.748	35.898
LazyARA*	Células Expandidas	211	6.367.328	215	83	6.462.321	105
	Tempo (seg)	0.019	5.501	0.022	0.011	5.788	0.027
	Custo da Solução	29.761	1.049.948	37.092	23.557	1.045.491	33.070

Durante a realização dessa nova comparação, surgiu uma curiosa informação que deve ser apresentada desde já para facilitar a compreensão dos resultados. Os algoritmos ANA* com direção de busca *backward* e o algoritmo R* com direção de busca *forward* e *backward*, ambos utilizando a nova função custo modificada, conseguiram localizar seus objetivos em um tempo tão ínfimo que foram considerados instantâneo ou 0 pelos planejadores. Esse fato ocorreu devido a uma particularidade destes que arredondam para zero qualquer valor de tempo inferior a 500 milissegundos para executar o planejamento e localizar o objetivo. Desse modo, podemos interpretar a notação 0 como um tempo extremamente diminuto, mas não instantâneo como aparenta.

É perceptível que a nova função custo conseguiu melhorar os resultados obtidos na análise anterior. A mesma, foi capaz de minimizar em praticamente a metade o tempo necessário para o R*, com direção de busca *forward*, identificar a melhor rota nos ambientes complexos. Já o algoritmo ANA*, com a mesma direção de busca, teve uma melhora em torno de 30% no ambiente 2 *e* uma redução bastante significante nos ambientes 1 e 2 com direção *backward*. O AD*, com a direção de busca *forward*, reduziu em aproximadamente 25% o tempo necessário para localiza o objetivo no segundo ambiente (mais complexo). Por fim, o LazyARA* conseguiu melhorar significativamente

o tempo para localizar o objetivo em todos os ambientes, tanto na direção *forward* como *backward*, bem como minimizar a quantidade de células expandidas na direção de busca direta. Novamente, o custo da solução geral foi quase o mesmo em todos os casos.

Por fim, os resultados da comparação mostraram que o algoritmo mais eficiente para os diferentes espaços foi o R*, com a direção de busca f*orward*, onde o mesmo conseguiu reduzir eficientemente o custo computacional das buscas, bem como a quantidade de células expandidas e principalmente o tempo necessário para localizar os objetivos. Verificou-se ainda que a modificação da função custo conseguiu melhorar significativamente os resultados obtidos, oferecendo uma redução considerável no tempo para localizar o objetivo, na maioria dos ambientes. Contudo, a mesma se mostrou mais eficiente para problemas grandes e complexos, os de fato utilizados na robótica real, como foi representado pelo ambiente complexo Env2. Uma comparação considerando o melhor algoritmo analisado com sua função modificada pode ser vista na Tabela 7 e mais detalhes sobre essa análise podem ser vistos no Apêndice C dessa dissertação.

Tabela 7: Resultados comparativo entre o algoritmo R* e a sua versão modificada.

		Ambiente 1	Ambiente 2	Ambiente 3	Ambiente 1	Ambiente 2	Ambiente 3
Rstar [Normal]	Células Expandidas	2	11	2	5	208	5
	Tempo (seg)	0.012	0.410	0.038	0.030	0.712	0.061
	Custo da Solução	24.726	1.146.596	33.898	24.484	1.135.844	34.726
Rstar [Modificado]	Células Expandidas	2	12	2	5	204	5
	Tempo (seg)	0	0.297	0.031	0.015	0.405	0
	Custo da Solução	24.726	1.171.704	33.898	34.726	1.416.748	35.898

5.2. Ambiente dinâmico

Nessa subseção, serão apresentados os resultados obtidos na comparação entre os quatro algoritmos de planejamento de caminho escolhidos em um ambiente dinâmico. Durante os testes alguns parâmetros foram novamente utilizados para mensurar a eficiência dos algoritmos selecionados, tais como: o tempo para o algoritmo planejar a trajetória que alcance o objetivo, o número de iterações que o mesmo necessitou realizar para traçar a trajetória desejada, a média aritmética do tempo gasto pelo número de iterações, o número de colisões no ambiente lotado, bem como informações da quantidade e percentagem de memória utilizada.

Vale a pena esclarecer que o tempo total calculado pelo planejador e apresentado durante a análise é o resumo dos períodos que o algoritmo necessitou para localizar a melhor solução que interligue o ponto inicial até o ponto objetivo, considerando todos os replanejamentos e ajustes na rota planejada e não o tempo que o robô levou para se deslocar fisicamente do primeiro ponto até esse último. Pois tais informações da duração do deslocamento físicos do autômato são variáveis de acordo com cada tipo de robô existente e não são consideradas nessa análise.

O melhor desempenho obtido nessa avaliação, considerando os quatro algoritmos e as diferentes direções de buscas será novamente destacado na tabela pela cor verde escuro e o pior pela cor vermelha. Os resultados obtidos, por cada algoritmo, considerando a direção de busca *forward* serão destacados pela letra (*F*) e os resultados com a direção *backward* pela letra (*B*). Todos os resultados obtidos na avaliação podem ser conferidos na Tabela 8 abaixo e no Apêndice D dessa dissertação.

Tabela 8: Resultados comparativos entre os algoritmos AD*, ANA*, R* LazyARA* com a nova função custo

Requisitos	AD*		ANA		R*		LazyARA*	
	(F)	(B)	(F)	(B)	(F)	(B)	(F)	(B)
Tempo total (s)	14.2	11.8	116.8	49.7	1.8	3.3	7.8	13.1
Média Geral (ms)	19.29	19.55	55.3	53.85	3.34	5.58	11.09	20.66
Número de iterações	692	598	2370	879	533	584	706	698
Alocação de memória (MB)	16,72	5,18	55.67	13.60	43.04	21.03	40.07	120
Número de Colisões	4	2,67	18.33	11	1.67	0.67	4.67	6

É possível observar, com o auxílio da tabela 8, que os algoritmos AD* e ANA* apresentaram melhores resultados quando utilizados com a direção de busca *backward*, em contraste, os algoritmos R* e LazyARA* tiveram melhor desempenho com a versão *forward*.

Verificou-se que o algoritmo R*, com direção de busca *forward*, foi o algoritmo que apresentou o menor tempo para localizar o objetivo de toda a análise, esse ainda exibiu a menor quantidade de iterações para localizar o objetivo e, consequentemente, foi o algoritmo que obteve melhor resultado no quesito média geral. Contudo, a sua versão com direção de busca *backward* foi a que conquistou o menor número de colisões de todos os experimentos.

O algoritmo LazyARA*, com direção de busca *forward*, apresentou o segundo menor tempo e média geral dessa análise e sua versão com direção de busca backward, demonstrou uma singela melhora de 1% no número de iterações, quando comparada com a versão de busca direta.

Já o algoritmo AD*, com direção de busca *backward*, expôs a terceira menor quantidade de colisões e de iterações da pesquisa, perdendo apenas para o algoritmo R* com as direções de buscas *backward* e *forward*, respectivamente. Em contrapartida, o mesmo foi o algoritmo que consumiu a menor quantidade de memória de toda a experimentação. Já a sua versão com direção de busca *forward*, apresentou apenas uma ligeira melhora de 1% em sua média geral, quando comparada com a primeira versão.

Por fim, o ANA* com direção de busca *forward* foi o algoritmo que apresentou os piores resultados dessa análise. O mesmo necessitou da maior quantidade de tempo para localizar o objetivo, apresentou a maior média geral e a mais elevada quantidade de iterações, bem como quantidade de memória e exibiu o número mais elevado de colisões. Em compensação, a sua versão com direção de busca *backward*, conquistou uma melhora de 57% no tempo necessário para localizar o objetivo, 3% na média, 63% no número de iterações, 76% na quantidade de memória e 40% no número de colisões, quando comparada a versão *forward*, mas mesmo assim ostenta resultados significativamente inferiores aos demais algoritmos.

Sendo assim, como constatado na análise estática, o algoritmo R* com direção de busca *forward* apresenta os melhores resultados quando comparados aos demais. O mesmo conseguiu localizar o objetivo 77% mais rápido e reduzir em 64% o número de colisões quando comparado ao segundo melhor algoritmo da análise

5.2.1. Ambiente dinâmico com função custo modificada

Nessa subseção serão apresentados os resultados obtidos na comparação entre os quatro algoritmos de planejamento de caminho escolhidos, utilizando as modificações sugeridas na função custo, em um ambiente lotado e com alta dinâmica. É importante mencionar que os parâmetros utilizados para mensurar a eficiência dos algoritmos na pesquisa anterior serão reutilizados.

Novamente, o melhor desempenho obtido na análise considerando os quatro algoritmos e as diferentes direções de buscas será destacado pela cor verde escuro e o pior pela cor vermelha. Todos os resultados obtidos na avaliação podem ser conferidos na Tabela 9 abaixo e no Apêndice E dessa dissertação.

Tabela 9: Resultados comparativos entre os algoritmos AD*, ANA*, R* LazyARA* com a nova função custo.

Requisitos	AD* - [Modificado]		ANA* - [Modificado]		R* - [Modificado]		LazyARA* - [Modificado]	
	(F)	(B)	(F)	(B)	(F)	(B)	(F)	(B)
Tempo total (s)	9.8	8.4	16.1	18.7	1.0	2.4	2.7	4.1
Média Geral (ms)	20.56	14.61	23.07	35.56	2.13	4.24	4.59	5.9
Número de iterações	483	577	688	517	511	588	590	700
Alocação de memória (MB)	110.9	20.78	2.15	57.97	19.57	28.71	5.13	106
Número de Colisões	1.67	2	1	1	0	0	1.33	1.33

Por meio da tabela 9, é possível observar que mais uma vez o algoritmo AD* apresenta melhor resultado quando utilizado com a direção de busca *backward*. Em contraste, os algoritmos R*, ANA* e LazyARA* tiveram melhor desempenho com a versão *forward*.

O algoritmo R*, com direção de busca *forward*, mais uma vez foi o algoritmo que apresentou o menor tempo para localizar o objetivo de toda a análise. Esse ainda obteve o melhor resultado no quesito média geral e ainda se destacou pela a ausência de colisões. Sua versão com direção *backward* também conseguiu repetir o feito da ausência de colisões e apresentou bons resultados nos demais quesitos.

O algoritmo LazyARA*, com direção de busca *forward*, apresentou o segundo menor tempo e foi o segundo algoritmo a consumir menos memória, o mesmo ainda conquistou a terceira melhor média geral. Por fim, tanto a sua versão direta como a versão inversa apresentaram a mesma quantidade de colisões.

Já o algoritmo AD*, com direção de busca *forward*, foi o algoritmo que apresentou a menor quantidade de iterações para localizar o objetivo de toda a análise. Já

sua versão *backward* conseguiu melhorar 14% o tempo para localizar o objetivo, 29% a média geral e 81% de redução de memória, quando comparado com a versão direta.

Por fim, o ANA* com direção de busca *forward*, foi o algoritmo que apresentou a melhor utilização de memória de toda a análise. O mesmo ainda apresentou a segunda menor quantidade de colisões do experimento. Já a sua versão *backward* exibiu o pior tempo da análise, a maior média geral e a quantidade mais elevada de memória; em contrapartida reduziu em 25% o número de iterações, quando comparada a sua versão *forward*, e além disso, também conquistou a segunda menor quantidade de colisões da análise.

Dessa forma, o algoritmo R* com direção de busca *forward*, quando comparado aos demais, mais uma vez se mostrou como o mais eficiente nas diferentes ocasiões e direções de buscas. O mesmo conseguiu localizar o objetivo 63% mais rápido, reduzir em 13% o número de iterações, minimizou em 54% a média geral e diminuiu em 100% o número de colisões, quando comparado ao segundo melhor algoritmo da análise.

5.3.Comparação dos resultados

As comparações apresentadas nessa subseção têm como objetivo relacionar os resultados compilados dos 4 algoritmos selecionados, considerando as duas direções de buscas, e compará-los com as suas respectivas versões modificadas que fazem uso da melhoria propostas na pesquisa.

Todos os resultados serão apresentados na Tabela 10 disponibilizada logo abaixo. Na parte superior da mesma, serão apresentados os dados dos algoritmos originais e logo abaixo, os resultados destes com suas funções custo modificadas. Os melhores desempenhos obtidos em cada uma das análises mais uma vez serão destacados pela cor verde escuro e os piores pela cor vermelha.

Tabela 10: Resultados comparativos entre os algoritmos originais e as suas versões modificadas.

Requisitos	AD* - [Normal]		ANA* - [Normal]		R* - [Normal]		LazyARA* - [Normal]	
	(F)	(B)	(F)	(B)	(F)	(B)	(F)	(B)
Tempo total (s)	14.2	11.8	116.8	49.7	1.8	3.3	7.8	13.1
Média Geral (ms)	19.29	19.55	55.3	53.85	3.34	5.58	11.09	20.66
Número de iterações	692	598	2370	879	533	584	706	698
Alocação de memória (MB)	16,72	5,18	55.67	13.60	43.04	21.03	40.07	120
Número de Colisões	4	2,67	18.33	11	1.67	0.67	4.67	6

Requisitos	AD* - [Modificado]		ANA* - [Modificado]		R* - [Modificado]		LazyARA* - [Modificado]	
	(F)	(B)	(F)	(B)	(F)	(B)	(F)	(B)
Tempo total (s)	9.8	8.4	16.1	18.7	1.0	2.4	2.7	4.1
Média Geral (ms)	20.56	14.61	23.07	35.56	2.13	4.24	4.59	5.9
Número de iterações	483	577	688	517	511	588	590	700
Alocação de memória (MB)	110.9	20.78	2.15	57.97	19.57	28.71	5.13	106
Número de Colisões	1.67	2	1	1	0	0	1.33	1.33

Por meio da tabela 10, é possível verificar que que todos os quatro algoritmos, que fizeram uso das modificações propostas, conseguiram melhorar significativamente suas performances em praticamente todos os quesitos.

O algoritmo AD*, utilizando a modificação proposta e com direção de busca forward, conseguiu conquistar uma redução de 31% no tempo necessário para localizar o objetivo, 30% no número de iterações e 58% no número de colisões. Já através de sua versão backward, o mesmo conseguiu melhorar em 29% o tempo para localizar o objetivo, 25% a sua média geral, 25% o número de colisões e conquistou um ínfimo melhoramento, de 4% no número de iterações, quando comparado com a sua versão original.

O algoritmo ANA*, com a direção de busca direta, foi o algoritmo que obteve a maior melhoria observada em todas as verificações. O mesmo, utilizando as modificações propostas, conquistou uma melhora de 95% no número de colisões, 96% na quantidade de memória, conseguiu reduzir em 86% o tempo necessário para localizar o objetivo, minimizou em 71% o número de iterações e diminuiu em 58% a sua média geral. Já na sua versão *backward*, o algoritmo conseguiu melhorar em 91% o número de colisões, 62% o tempo necessário para localizar o objetivo, 41% o número de iterações e 34% na sua média geral.

Já o algoritmo R*, com direção de busca *forward* e fazendo uso das modificações propostas, foi o algoritmo que apresentou os melhores resultados de todas as análises. O mesmo conquistou uma melhora de 100% no número de colisões, quando comparado a sua versão original, além de reduzir em 55% a utilização de memória, 44% no tempo necessário para localizar o objetivo, 36% na média geral e 4% no número de iterações. Já

na sua versão com direção de busca *backward*, o mesmo conseguiu novamente reduzir em 100% o número de colisões, 27% o tempo necessário para localizar o objetivo e 24% a sua média geral.

Por fim, o algoritmo LazyARA*, fazendo uso das modificações propostas e com direção de busca *forward*, conseguiu reduzir em 87% o uso de memória, 72% no número de colisões, 65% no tempo necessário para localizar o objetivo, 59% na média geral e 16% no número de iterações. Já a sua versão com direção de busca *backward*, conquistou uma melhora de 78% no número de colisões, 71% na média geral, 69% no tempo necessário para localizar o objetivo e 12% na utilização de memória.

Sendo assim, é possível perceber que a modificação proposta nessa pesquisa conseguiu melhorar significativamente as performances dos algoritmos selecionados da família A*. A mesma, apesar de simples, conseguiu demonstrar um grande impacto positivo nos resultados. Pois, com a omissão da variável *g(n)* - responsável em armazenar informações subutilizadas do passado dos algoritmos, as equações principais das respectivas funções custo foram simplificadas e os algoritmos passaram a utilizar, de forma mais eficiente, os recursos computacionais disponíveis, não desperdiçando-os com informações não tão necessárias. Com essa simplificação, os mesmos passam a realizar menos cálculos matemáticos e conseguem reduzir significativamente o tempo e o número de iterações para alcançar a configuração desejada. Com tal melhoria, o robô também consegue reagir mais rápido a cada instante e, por conseguinte, a otimização também permiti reduzir o número de colisões e consegue, assim, melhorar os resultados conquistados pelos algoritmos atuais e selecionados.

Desse modo, pode-se perceber que o algoritmo que apresentou os melhores resultados considerando todas as análises foi o R*, com a modificação proposta e com direção de busca *forward.* O mesmo, quando comparado com o 2ª melhor algoritmo, conseguiu melhorar em 100% o número de colisões, 63% o tempo total para localizar o objetivo, 54% a média geral e 13% o número de iterações necessário para alcançar a configuração meta.

6. Conclusão e Trabalhos futuros

Neste artigo foi apresentado uma modificação na função custo que pode ser aplicada a qualquer algoritmo de planejamento de caminho baseado em grid da família A* e usado na robótica móvel. Tal modificação mantém as vantagens características da família A* como a ação de evitar mínimos locais, graças a utilização das listas de controle de acesso abertas e fechadas, bem como consegue localizar o objetivo de forma mais rápida, pois executa menos operações em seus ciclos e se resguarda das desvantagens já conhecidas do algoritmo, que é o recalculo redundante dos estados já visitados, devido à forte influência da variável *g(n).* Esse ainda consegue preservar as características de subotimalidade utilizadas pelos os algoritmos mais atuais de planejamento de caminho e aplicados à robótica móvel.

Para essa modificação, um levantamento dos melhores algoritmos de busca em ambiente estático e dinâmico foi realizado e, através deste, foram analisados os quatro melhores e mais atuais algoritmos apresentados pela literatura, o AD*, ANA*, R* e LazyARA*. Observou-se que suas estratégias de busca apresentavam quase a mesma função custo herdada do algoritmo clássico de geração de trajetória A*, desenvolvido em 1968, mantendo especificamente o fragmento que calcula e armazena o custo passado pelo robô para se mover de uma posição inicial até a posição atual. Durante a execução dos algoritmos foi percebido que essa variável sempre armazena e atualiza as informações do caminho já realizado e, tais informações, são usadas para influenciar a procura pelo objetivo. Contudo, especialmente em situações reais que na grande maioria utilizam ambientes dinâmicos e suas configurações modificam a cada segundo, tal ação é subutilizada e não ajuda efetivamente o algoritmo a localizar o objetivo, apenas aumenta o custo computacional, devido as suas operações realizadas. Por esta razão, foi proposta uma modificação da função usada por estes algoritmos que minimiza essa dependência.

Algumas simulações foram realizadas usando um ambiente estático e um ambiente altamente dinâmico com doze obstáculos que se deslocam de forma aleatória e com velocidades diferentes. Inicialmente foram comparados os 4 algoritmos selecionados considerando diferentes parâmetros e, em seguida, esses foram novamente analisados com a função custo modificada. Os resultados da comparação entre os algoritmos originais escolhidos mostraram que o R*, com direção de busca direta, é o algoritmo mais eficiente para diferentes espaços e pesquisas. Esse foi capaz de reduzir a quantidade de

células expandidas e especialmente o tempo necessário para localizar os objetivos em ambientes estáticos e minimizar o número de colisões, número da iteração e planejamento de tempo em ambientes dinâmicos.

No entanto, a mesma análise considerando os algoritmos com a função custo modificada, demonstrou uma melhoria significativa sobre os resultados dos algoritmos originais. Em ambientes estáticos, esta modificação se mostrou mais eficaz para problemas grandes e complexos, tais como o ambiente 2 - os ambientes que são efetivamente utilizados por robôs reais. Em ambientes altamente dinâmicos, a modificação na função custo demonstrou melhorar consideravelmente o tempo de planejamento e o número de iterações para localizar o objetivo, bem como reduzir a utilização de memória o que, consequentemente, torna o robô mais ágil e habilidoso e, como demonstrado na experimentação, possibilita reduzir e até zerar o número de colisões. Dessa forma, a modificação consegue apresentar melhorias consideráveis nos 4 algoritmos, mas os melhores resultados ainda foram conquistados com o algoritmo R* com direção de busca *forward*;

A partir do trabalho realizado é possível identificar vários pontos que, potencialmente, podem constituir objeto de estudo futuro. Dentre estes a realização de outros experimentos considerando novos algoritmos e classes algorítmicas pertencentes a família A*, tais como como as bidirecionais, de busca paralela, de tempo real e ademais, como apresentado em [70]. Também é sugerida a realização de testes comparativos e análise de desempenho considerando o auxílio de robôs *TurtleBot2* reais, Figura 16, recentemente adquiridos pelo Laboratório de Aplicações em Sistemas Embarcados e Robótica (LaSER) da UFPB.

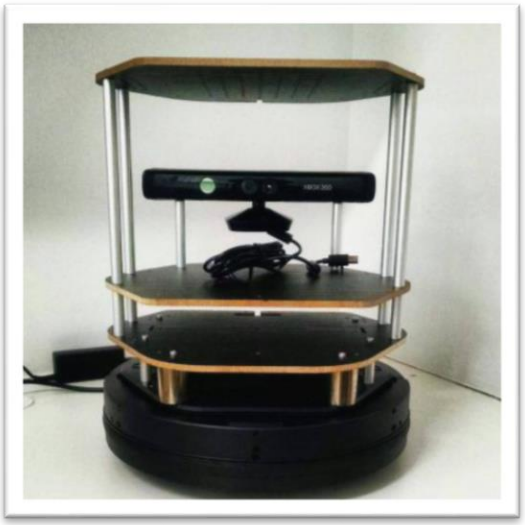


Figura 16: Robô *TurtleBot2* adquirido pelo LaSER

7. Referências

[1] - Thrun, S. Probabilistic robotics. Commun. ACM, 45(3):52–57, 2002.

[2] - Thrun,S., S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hahnel, D. Montemerlo, A. Morris, Z. Omohundro, C. Reverte, and Whittaker W. Autonomous exploration and mapping of abandoned mines. Robotics & Automation Magazine, IEEE, 11(4):79–91, 2004

[3] - NASA. Mars pathfinder. Available in: <http://mars.jpl.nasa.gov/MPF/>, acessed in set, 18, 2014

[4] - NASA, PHILAE. Available in: <http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=2004-006C>> acessed in: set, 19, 2014

[5] - IRobot. iRobot corporation. available in: <http://www.irobot.com/us/Company/About.aspx> accessed in: set, 19, 2014.

[6] - Robomow. Leading Robotic Lawnmower Provider. available in: <http://www.robomow.com/en-USA> accessed in: 20, set, 2014.

[7] - Xiong, M., et al., A Rule-Based Motion Planning for Crowd Simulation. Cyberworlds, International Conference 2009. p. 88-95. 2009

[8] - Yao, J., et al., Path Planning for Virtual Human Motion Using Improved A* Star Algorithm. Information Technology: New Generations, Third International Conference, 2010: p. 1154-1158.

[9] - Heinen, F. J. and F. Osório. Sistema de controle híbrido para robôs móveis autônomos. WTDIA/SBIA - Workshop De Teses e Dissertações De Inteligência Artificial, 2002.

[10] - Lozano-Pérez, T. and M.A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles. Communications of the ACM. 22: p. 560-570. 1979.

[11] - Hwang, Y.K. and N. Ahuja, Gross Motion Planning - a survey. ACM Computing Surveys, 24: p. 219-291, Setembro 1992

[12] - Lozano-Pérez, T., Spatial planning: a configuration space approach. IEEE transactions on computers. C-32: p. 108-120. 1983

[13] - Ge, Q. and J.M. McCarthy, Equations for boundaries of joint obstacles for planar robots, in IEEE international conference on robotics and automation. p. 164-169. 1989.

[14] - Hwang, Y.K., Boundary equations of configuration obstacles for manipulators, in IEEE international conference on robotics ans automation. p. 298-303. 1990

[15] - Branicky, M.S. and W.S. Newman. Rapid computation of configuration space obstacles. in Robotics and Automation, Proceedings. IEEE International Conference on. 1990.

[16] - Gupta, K.K. and Z. Guo, Motion planning for many degrees of freedom: sequencial search with backtracking, in IEEE international conference on robotics ans automation.. p. 2328-2333. 1992.

[17] - Connolly, C.I., J.B. Burns, and R. Weiss. Path planning using Laplace's equation. in Robotics and Automation. Proceedings., 1990 IEEE International Conference on. 1990.

[18] - Latombe, J.C. Robot Motion Planning:. Kluwer international series in engineering and computer science: Robotics. Kluwer Academic Publishers, 1990.

[19] - Canny, J.F., Constructing roadmaps of semi-algebraic sets I: completeness. Artificial intelligence. 37: p. 203-222. 1988.

[20] - Latombe, J.C., Robot motion planning. 1991, Boston, MA: Kluwer Academic Publishers.

[21] - Asano, T., et al. Visibility-polygon search and euclidean shortest paths. in Foundations of Computer Science, 26th Annual Symposium on. 1985.

[22] - Aurenhammer, F., Voronoi diagrams - a survey of a fundamental geometric structure. ACM Computing Surveys, 23: p. 345-405. 1991.

[23] - Canny, J., A Voronoi method for the piano-movers problem, in IEEE international conference on robotics and automation.. p. 530--535. 1985

[24] - Preparata, F.P. and M.I. Shamos, Computational Geometry: An Introduction, New York: Spinger-Verlag. 1985.

[25] - Ahuja, N. and B.J. Schachter, Pattern models. New York: Wiley. 1983.

[26] - Kirkpatrick, D.G. Efficient computation of continuous skeletons. in Foundations of Computer Science, 1979., 20th Annual Symposium on. 1979.

[27] - Canny. J. The Complexity of Robot Motion Planning. ACM Doctoral Dissertation Award. Mit Press, 1988.

[28] - Canny, J.F. and C. Lin, An opportunistic global path planner. Algorithimica, 10: p.102-120. 1993

[29] - Spong, M.W.; S. Hutchinson, e M. Vidyasagar. Robot modeling and control. John Wiley & Sons, 2006.

[30] - Su, W., R. Meng, and C. Yu. A Study on Soccer Robot Path Planning with Fuzzy Artificial Potential Field. in Computing, Control and Industrial Engineering (CCIE), 2010 International Conference on. 2010.

[31] - Ottoni, G. L. Planejamento de trajetórias para robôs móveis. Projeto de Graduação em Engenharia de Computação–FURG, Rio Grande, 2000.

[32] - Costa, P.L.C.G. Planeamento Cooperativo de Tarefas e Trajectórias em Múltiplos Robôs, Faculdade de Engenharia da universidade do Porto, Dissertação de doutarado, 2011

[33] - Mazumder, P. Planar Decomposition for Quadtree Data Structure , Computer vision, graphics, and image processing 38, 258-274. 1987

[34] - Nascimento, T.P.; Costa, P.; Costa, P.G.; Moreira, A.P.; Conceição, A.G.S. A set of novel modifications to improve algorithms from the A* family applied in mobile robotics. Journal of the Brazilian Computer Society Volume: 19 Issue: 2 Pages: 167-179 Published: 2013

[35] - Peasgood M, Clark CM, McPhee J. A complete and scalable strategy for coordinating multiple robots within roadmaps. IEEE Trans Robot 24:238–292. 2008.

[36] - Peter E. Hart; Nils J. Nilsson; Bertram Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics, Volume 4, Issue 2, p.100-107, 1968

[37] - Pinagé,F; Silva, I, A; Pinto, L, D; Um sistema para navegação autônoma de robôs terrestres; IV Congresso de Pesquisa e Inovação da rede Norte e Nordeste de Educação Tecnológica, Belém - PA, 2009.

[38] - Cui, X; Shi, H. A*-based Pathfinding in Modern Computer Games. IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, January

[39] - Korf, Richard. "Depth-first Iterative-Deepening: An Optimal Admissible Tree Search". Artificial Intelligence 27: 97–109, 1985

[40] - Chakrabarti, P.P.; Ghose, S.; Acharya, A. and de Sarkar, S.C. Heuristic Search in Restricted Memory. AIJ, 41, 197-221, 1989

[41] - Stentz, A. Optimal and Efficient Path Planning for Partially-Know Environments In Proceedings IEEE International Conference on Robotics and Automation, 1994

[42] - Stentz, A. The Focussed D* Algorithm for Real-Time Replanning, In Proceedings of the International Joint Conference on Artificial Intelligence: 1652–1659. 1995.

[43] - Likhachev, M.; and Koenig, S. Incremental A* Technical report, College of Computing, Georgia Institute of Technologym Atlanta (Georgia), 2001 .

[44] - Koenig, S. and M. Likhachev, Fast replanning for navigation in unknown terrain. IEEE Transactions Robot. 21(3): p. 354-363. 2005.

[45] - Likhachev, M.; and Gordon, G.; and Thrun, S. ARA*: Anytime A* Search with Provable Bounds on Sub-Optimality, in Proceedings of Conference on Neural Information Processing Systems (NIPS), 2003

[46] - Ferguson, D.; Stentz, A. The Delayed D* Algorithm for Efficient Path Replanning. in International Conference on Robotics and Automation, 2005

[47] - Ferguson, D.; Stentz, A. Field D*: An Interpolation-Based Path Planner and Replanner, in International Symposium on Robotics Research (ISRR), 2005

[48] - Likhachev M, Ferguson D, Gordon G, Stentz A, Thrun S Anytime dynamic A*: an anytime replanning algorithm. In: Proceedings of the international conference on automated planning and scheduling (ICAPS), 2005

[49] - Berg, J.V.; Shah, R.; Huang, A.; and Goldberg, K. Y. Anytime nonparametric A*. In Proceedings of the Twenty Fifth National Conference on Artificial Intelligence. 2011.

[50] - Likhachev, M., and Stentz, A. R* search. Association for the Advancement of Artificial Intelligence, 2008.

[51] - Hwang, V.; Phillips, M.; Srinivasa, S. and Likhachev, M. "Lazy Validation of Experience Graphs," Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2015.

[52] - Zhou, R. and E.A. Hansen, Memory-Bounded A* Graph Search. FLAIRS Conference, 2002: p. 203-209. 2002.

[53] - Ramalingam, G.; T. Reps, An incremental algorithm for a generalization of the shortest-path problem, Journal of Algorithms 21. 267–305. 1996.

[54] - Koenig, S. and M. Likhachev, Improved fast replanning for robot navigation in unknown terrain. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 2002.

[55] - Chakrabarti, P., S. Ghosh, and S. DeSarkar, Admissibility of AO* when heuristics overestimate. Artificial Intelligence. 34: p. 97–113. 1998.

[56] - Rabin, S., A* speed optimizations. In DeLoura, M., ed., Game Programming Gems, p. 272-287. 2000.

[57] - Bonet, B. and H. Geffner, Planning as heuristic search. . Artificial Intelligence, 2001. 129(1-2): p. 5-33.

[58] - Edelkamp, S., Planning with pattern databases. In Proceedings of the European Conference on Planning, 2001.

[59] - Zhou, R. and E. Hansen, Multiple sequence alignment using A*. In Proceedings of the National Conference on Artificial Intelligence (AAAI), 2002.

[60] - Ferguson, D., M. Likhachev, and A. Stentz, A guide to heuristicbased path planning. in Proc. of the Workshop on Planning under Uncertainty for Autonomous Systems, Int. Conf. on Automated Planning and Scheduling (ICAPS'05), 2005.

[61] - Tuckman, B. W. Manual de Investigação em Educação. Lisboa: Fundação Calouste Gulbenkian, p. 374, 2000

[62] - SEARCH-BASED PLANNING LAB. (SBPL). available in: <http://www.sbpl.net/Software>. accessed in: jul, 09, 2015.

[63] - Lester. P. A* Pathfinding for Beginners. Available in: <http://www.policyalmanac.org/games/aStarTutorial.htm>. accessed in: jul, 09, 2015.

[64] - SIMTWO - available in: <http://paginas.fe.up.pt/~paco/wiki/index.php?n=Main.SimTwo>. Acessed in: jul, 09, 2015.

[65] - Weighted A* (Pohl, 1970) - Pohl, I. 1970. Heuristic search viewed as path finding in a graph. Artificial Intelligence 1(3-4):193 – 204. 1970.

[66] - Black, Paul E. (2 February 2005). "greedy algorithm". Dictionary of Algorithms and Data Structures. U.S. National Institute of Standards and Technology (NIST). Retrieved 17 August 2012

[67] - Costa, P., Gon¸calves, J., Lima, J., and Malheiros, P. Simtwo realistic simulator: A tool for the development and validation of robot software. in the International Journal of Theory and Applications of Mathematics Computer Science. 2011.

[68] - Smith R (2010) Open dynamics engine. http://www.ode.org

[69] - Ferreira JR, Moreira APGM (2010) Non-linear model predictive controller for trajectory tracking of an omni-directional robot using a simplified model. In: 9th Portuguese conference on automatic control

[70] - Rios, H. O.; Chaimowicz, L. A survey and Classification of A* based Best-first heuristic search algorithms. Proceedings of the 20th Brazilian conference on Advances in artificial intelligence - SBIA, Pages 253-262, 2010

8. Apêndices

APÊNDICE A:

TESTE COMPARATIVO ENTRE AS FUNÇÕES CUSTO

F = G(s) +H(s)						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Número de iterações	597	610	592	597	604	600,0
Tempo de planejamento (ms)	0,70	0,68	0,69	0,69	0,67	0,686
Número de colisões	6	4	5	2	3	4,00
F(s) = H(s)						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Número de iterações	515	587	499	512	502	523
Tempo de planejamento (ms)	0,12	0,09	0,11	0,085	0,04	0,089
Número de colisões	2	1	2	1	1	1,40

APÊNDICE B:
RESULTADOS COMPARATIVOS ENTRE OS ALGORITMOS AD*, ANA*, R* e LAZYARA* - AMBIENTE ESTÁTICO

AD*

ADSTAR - Ambiente 1 - Forward						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	181,00	178,00	169,00	153,00	164,00	169,00
Tempo (seg)	0,02	0,02	0,02	0,01	0,01	0,02
Custo da Solução	22.987,00	21.578,00	20.884,00	29.613,00	24.431,00	23.898,60
ADSTAR - Ambiente 2 - Forward						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	83.711,00	81.683,00	84.137,00	81.131,00	81.275,00	82.387,40
Tempo (seg)	4,12	4,86	3,84	4,42	3,78	4,20
Custo da Solução	1.024.897,00	1.044.994,00	1.120.788,00	1.046.279,00	1.012.782,00	1.049.948,00
ADSTAR - Ambiente 3 - Forward						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	321,00	296,00	322,00	332,00	325,00	319,20
Tempo (seg)	0,03	0,01	0,02	0,03	0,04	0,03
Custo da Solução	29.692,00	34.396,00	28.488,00	34.641,00	38.133,00	33.070,00
ADSTAR - Ambiente 1 - Backward						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	53,00	61,00	52,00	63,00	71,00	60,00
Tempo (seg)	0,009	0,009	0,007	0,009	0,013	0,009
Custo da Solução	22.671,00	25.476,00	21.041,00	19.645,00	30.658,00	23.898,20
ADSTAR - Ambiente 2 - Backward						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	82.161,00	95.147,00	104.119,00	84.159,00	96.359,00	92.389,00
Tempo (seg)	5.959,00	5.212,00	4.281,00	3.441,00	4.998,00	4.778,20
Custo da Solução	1.103.791,00	1.011.190,00	1.050.592,00	1.017.991,00	1.066.176,00	1.049.948,00
ADSTAR - Ambiente 3 - Backward						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	152,00	174,00	163,00	183,00	184,00	171,20
Tempo (seg)	0,016	0,017	0,021	0,024	0,024	0,020
Custo da Solução	24.061,00	29.099,00	27.061,00	42.116,00	43.013,00	33.070,00

Planejador: ADStar

ANA*

ANASTAR - Ambiente 1 - Forward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	292,00	275,00	284,00	281,00	298,00	286,00
Tempo (seg)	0,020	0,025	0,021	0,020	0,019	0,021
Custo da Solução	24.248,00	22.103,00	24.728,00	25.968,00	22.443,00	23.898,00

ANASTAR - Ambiente 2 - Forward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	2.962.137,00	3.188.103,00	4.012.476,00	4.110.232,00	3.410.018,00	3.536.593,20
Tempo (seg)	101,000	96,016	104,100	104,002	95,001	100,024
Custo da Solução	1.057.435,00	1.122.184,00	1.039.277,00	1.038.035,00	1.101.599,00	1.071.706,00

ANASTAR - Ambiente 3 - Forward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	389,00	401,00	392,00	391,00	382,00	391,00
Tempo (seg)	0,026	0,032	0,028	0,031	0,028	0,029
Custo da Solução	33.342,00	34.908,00	34.547,00	31.617,00	30.936,00	33.070,00

ANASTAR - Ambiente 1 - Backward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	75,00	76,00	63,00	66,00	70,00	70,00
Tempo (seg)	0,009	0,009	0,004	0,004	0,006	0,006
Custo da Solução	24.832,00	25.720,00	22.010,00	22.745,00	24.183,00	23.898,00

ANASTAR - Ambiente 2 - Backward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	3.876.449,00	4.478.302,00	3.747.921,00	3.653.320,00	3.836.238,00	3.918.446,00
Tempo (seg)	104.422,000	107.099,000	91.179,00	93.387,000	104.038,000	100.025,00
Custo da Solução	928.299,00	1.110.315,00	1.082.104,00	1.248.529,00	1.192.333,00	1.112.316,00

ANASTAR - Ambiente 3 - Backward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	279,00	276,00	303,00	282,00	265,00	281,00
Tempo (seg)	0,013	0,019	0,024	0,016	0,018	0,018
Custo da Solução	28.236,00	33.090,00	36.510,00	31.602,00	35.912,00	33.070,00

Planejador: ANAStar

R*

RSTAR - Ambiente 1 - Forward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	2,00	2,00	2,00	3,00	1,00	2,00
Tempo (seg)	0,013	0,012	0,011	0,014	0,012	0,012
Custo da Solução	28.755,00	24.923,00	21.622,00	23.516,00	24.814,00	24.726,00

RSTAR - Ambiente 2 - Forward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	11,00	13,00	12,00	9,00	10,00	11,00
Tempo (seg)	0,401	0,399	0,416	0,407	0,428	0,410
Custo da Solução	1.270.061,00	1.120.194,00	921.442,00	1.260.192,00	1.161.091,00	1.146.596,00

RSTAR - Ambiente 3 - Forward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	1,00	2,00	2,00	3,00	2,00	2,00
Tempo (seg)	0,031	0,039	0,035	0,044	0,039	0,038
Custo da Solução	32.925,00	32.700,00	33.885,00	34.986,00	34.994,00	33.898,00

RSTAR - Ambiente 1 - Backward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	5,00	4,00	5,00	5,00	6,00	5,00
Tempo (seg)	0,026	0,024	0,029	0,034	0,037	0,030
Custo da Solução	22.429,00	22.793,00	25.561,00	28.709,00	22.928,00	24.484,00

RSTAR - Ambiente 2 - Backward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	214,00	203,00	208,00	209,00	206,00	208,00
Tempo (seg)	0,747	0,729	0,709	0,659	0,716	0,712
Custo da Solução	993.254,00	1.118.331,00	1.372.767,00	1.374.764,00	820.104,00	1.135.844,00

RSTAR - Ambiente 3 - Backward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	4,00	5,00	7,00	4,00	5,00	5,00
Tempo (seg)	0,061	0,066	0,069	0,052	0,059	0,061
Custo da Solução	37.295,00	34.560,00	35.592,00	33.886,00	32.297,00	34.726,00

Planejador: RStar

LazyARA*

LAZYARASTAR - Ambiente 1 - Forward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	261,00	317,00	331,00	268,00	253,00	286,00
Tempo (seg)	0,024	0,027	0,033	0,034	0,035	0,031
Custo da Solução	21.880,00	22.199,00	26.079,00	25.619,00	23.713,00	23.898,00
LAZYARASTAR - Ambiente 2 - Forward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	7.740.211,00	8.148.496,00	7.611.921,00	8.264.145,00	7.590.799,00	7.871.114,40
Tempo (seg)	6.989,00	8.614,00	7.798,00	7.529,00	8.340,00	7.854,00
Custo da Solução	1.012.705,00	1.048.039,00	948.702,00	1.009.885,00	1.035.294,00	1.010.925,00
LAZYARASTAR - Ambiente 3 - Forward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	398,00	408,00	390,00	374,00	385,00	391,00
Tempo (seg)	0,025	0,031	0,029	0,034	0,036	0,031
Custo da Solução	34.108,00	34.021,00	31.002,00	34.511,00	31.708,00	33.070,00
LAZYARASTAR - Ambiente 1 - Backward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	72,00	73,00	69,00	66,00	70,00	70,00
Tempo (seg)	0,014	0,019	0,016	0,012	0,014	0,015
Custo da Solução	25.817,00	24.806,00	23.554,00	21.809,00	23.504,00	23.898,00
LAZYARASTAR - Ambiente 2 - Backward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	10.323.849,00	10.769.295,00	9.019.097,00	12.168.174,00	11.206.040,00	10.697.291,00
Tempo (seg)	8.039,00	9.189,00	9.890,00	10.121,00	9.306,00	9.309,00
Custo da Solução	978.962,00	947.241,00	1.243.681,00	1.038.454,00	1.041.402,00	1.049.948,00
LAZYARASTAR - Ambiente 3 - Backward						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	283,00	287,00	264,00	282,00	289,00	281,00
Tempo (seg)	0,016	0,019	0,016	0,019	0,017	0,017
Custo da Solução	29.749,00	25.683,00	32.274,00	41.876,00	35.768,00	33.070,00

Planejador: LazyAra*

APÊNDICE C:
RESULTADOS COMPARATIVOS ENTRE OS ALGORITMOS AD*, ANA*, R* e LAZYARA* - COM NOVA FUNÇÃO CUSTO E AMBIENTE ESTÁTICO

AD* - NFC

ADSTAR - Ambiente 1 - Forward - **NFC**						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	193,00	205,00	202,00	194,00	181,00	195,00
Tempo (seg)	0,019	0,011	0,014	0,015	0,014	0,015
Custo da Solução	24.431,00	20.883,00	21.577,00	29.613,00	22.986,00	23.898,00
ADSTAR - Ambiente 2 - Forward - **NFC**						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	86.291,00	91.621,00	89.491,00	90.261,00	89.061,00	89.345,00
Tempo (seg)	3,301	3,038	3,484	3,149	3,019	3,198
Custo da Solução	1.120.788,00	1.012.782,00	1.024.897,00	1.046.279,00	1.044.994,00	1.049.948,00
ADSTAR - Ambiente 3 - Forward - **NFC**						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	183,00	188,00	184,00	198,00	182,00	187,00
Tempo (seg)	0,011	0,016	0,017	0,015	0,014	0,015
Custo da Solução	34.641,00	29.692,00	38.133,00	34.396,00	28.488,00	33.070,00
ADSTAR - Ambiente 1 - Backward - **NFC**						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	194,00	209,00	198,00	218,00	211,00	206,00
Tempo (seg)	0,016	0,013	0,015	0,012	0,017	0,015
Custo da Solução	19.645,00	25.476,00	22.671,00	21.041,00	30.657,00	23.898,00
ADSTAR - Ambiente 2 - Backward - **NFC**						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	119.678,00	122.787,00	109.982,00	116.997,00	123.891,00	118.667,00
Tempo (seg)	4.918,00	6.409,00	5.509,00	4.918,00	5.936,00	5.538,00
Custo da Solução	1.011.190,00	1.017.991,00	1.066.176,00	1.103.791,00	1.050.592,00	1.049.948,00
ADSTAR - Ambiente 3 - Backward - **NFC**						
Parâmetros	**1º**	**2º**	**3º**	**4º**	**5º**	**Média**
Células Expandidas	209,00	198,00	215,00	217,00	216,00	211,00
Tempo (seg)	0,016	0,014	0,016	0,013	0,014	0,015
Custo da Solução	27.061,00	42.116,00	43.013,00	24.061,00	29.099,00	33.070,00

Planejador: ADStar;

ANA* - NFC

ANASTAR - Ambiente 1 - Forward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	313,00	299,00	310,00	304,00	309,00	307,00
Tempo (seg)	0,016	0,011	0,020	0,016	0,014	0,015
Custo da Solução	24.728,00	25.968,00	24.248,00	22.443,00	22.103,00	23.898,00
ANASTAR - Ambiente 2 - Forward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	3.064.022,00	2.589.113,00	2.392.202,00	1.956.512,00	2.775.281,00	2.555.426,00
Tempo (seg)	74.142,00	72.229,00	70.379,00	68.596,00	71.889,00	71.447,00
Custo da Solução	1.021.399,00	1.037.135,00	1.039.138,00	1.039.680,00	1.112.388,00	1.049.948,00
ANASTAR - Ambiente 3 - Forward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	786,00	811,00	803,00	781,00	799,00	796,00
Tempo (seg)	0,04w	0,049	0,049	0,043	0,047	0,047
Custo da Solução	34.547,00	30.936,00	31.617,00	34.908,00	33.342,00	33.070,00
ANASTAR - Ambiente 1 - Backward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	95,00	89,00	99,00	97,00	100,00	96,00
Tempo (seg)	0,0008	0,0001	0,0009	0,0004	0,0003	0,000
Custo da Solução	22.010,00	24.831,00	24.180,00	22.739,00	25.720,00	23.896,00
ANASTAR - Ambiente 2 - Backward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	3.628.010,00	3.273.900,00	3.808.902,00	3.846.929,00	3.556.849,00	3.622.918,00
Tempo (seg)	104.031,000	107.092,000	93.382,00	104.420,000	91.175,000	100.020,00
Custo da Solução	1.148.408,00	1.142.361,00	1.100.521,00	1.079.100,00	928.390,00	1.079.756,00
ANASTAR - Ambiente 3 - Backward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	321,00	312,00	317,00	299,00	316,00	313,00
Tempo (seg)	0,014	0,015	0,018	0,015	0,014	0,015
Custo da Solução	31.602,00	28.236,00	35.912,00	33.090,00	36.510,00	33.070,00

Planejador: ANAStar

R* - NFC

RSTAR - Ambiente 1 - Forward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	3,00	2,00	1,00	2,00	2,00	2,00
Tempo (seg)	0,0002	0,0004	0,0004	0,0002	0,0004	0,000
Custo da Solução	23.516,00	28.755,00	24.814,00	22.623,00	23.923,00	24.726,20
RSTAR - Ambiente 2 - Forward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	14,00	11,00	13,00	11,00	11,00	12,00
Tempo (seg)	0,311	0,377	0,216	0,207	0,376	0,297
Custo da Solução	1.280.032,00	1.125.014,00	1.372.061,00	1.161.011,00	920.402,00	1.171.704,00
RSTAR - Ambiente 3 - Forward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	2,00	3,00	1,00	2,00	2,00	2,00
Tempo (seg)	0,027	0,031	0,027	0,036	0,036	0,031
Custo da Solução	34.994,00	32.925,00	34.986,00	32.700,00	33.885,00	33.898,00
RSTAR - Ambiente 1 - Backward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	6,00	4,00	6,00	4,00	5,00	5,00
Tempo (seg)	0,017	0,014	0,013	0,017	0,016	0,015
Custo da Solução	34.928,00	32.529,00	37.716,00	32.896,00	35.561,00	34.726,00
RSTAR - Ambiente 2 - Backward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	202,00	207,00	203,00	204,00	204,00	204,00
Tempo (seg)	0,407	0,399	0,391	0,423	0,406	0,405
Custo da Solução	1.418.091,00	1.404.794,00	1.374.187,00	1.490.614,00	1.396.054,00	1.416.748,00
RSTAR - Ambiente 3 - Backward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	5,00	5,00	6,00	4,00	5,00	5,00
Tempo (seg)	0,0004	0,0007	0,0011	0,00012	0,00011	0,000
Custo da Solução	35.960,00	37.592,00	33.397,00	37.555,00	34.986,00	35.898,00

Planejador: RStar

LazyARA* - NFC

LAZYARASTAR - Ambiente 1 - Forward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	228,00	218,00	203,00	207,00	199,00	211,00
Tempo (seg)	0,017	0,018	0,023	0,022	0,017	0,019
Custo da Solução	28.917,00	29.886,00	28.719,00	31.199,00	30.084,00	29.761,00
LAZYARASTAR - Ambiente 2 - Forward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	7.611.921,00	8.264.145,00	7.590.799,00	7.740.211,00	8.148.496,00	7.871.114,40
Tempo (seg)	7.998,00	7.329,00	6.989,00	8.540,00	8.414,00	7.854,00
Custo da Solução	948.702,00	1.035.294,00	1.012.705,00	1.048.039,00	1.009.885,00	1.010.925,00
LAZYARASTAR - Ambiente 3 - Forward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	228,00	231,00	204,00	199,00	213,00	215,00
Tempo (seg)	0,022	0,019	0,023	0,026	0,020	0,022
Custo da Solução	36.011,00	35.701,00	38.576,00	37.170,00	38.002,00	37.092,00
LAZYARASTAR - Ambiente 1 - Backward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	82,00	86,00	85,00	79,00	83,00	83,00
Tempo (seg)	0,010	0,013	0,011	0,012	0,011	0,011
Custo da Solução	23.054,00	23.202,00	25.213,00	21.509,00	24.807,00	23.557,00
LAZYARASTAR - Ambiente 2 - Backward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	7.799.285,00	6.286.090,00	6.393.429,00	5.969.807,00	5.862.994,00	6.462.321,00
Tempo (seg)	5.599,00	5.389,00	6.875,00	5.468,00	5.609,00	5.788,00
Custo da Solução	1.042.597,00	1.225.865,00	969.878,00	949.691,00	1.039.424,00	1.045.491,00
LAZYARASTAR - Ambiente 3 - Backward - NFC						
Parâmetros	1º	2º	3º	4º	5º	Média
Células Expandidas	103,00	102,00	114,00	104,00	102,00	105,00
Tempo (seg)	0,024	0,025	0,029	0,028	0,029	0,027
Custo da Solução	26.683,00	32.274,00	35.768,00	35.749,00	34.876,00	33.070,00

Planejador: LazyARA*;

APÊNDICE D:

RESULTADOS COMPARATIVOS ENTRE OS ALGORITMOS AD*, ANA*, R* e LAZYARA* - AMBIENTE DINÂMICO

	AD* - Forward	1ª	2ª	3ª	4ª	5ª	Média
AD*	Tempo total (s)	7.802,00	10.459,00	24.340,00	15.467,00	12.932,00	14.200,0
	Média Geral (ms)	14,56	17,15	26,17	17,41	21,17	19,29
	Número de iterações	536,00	610,00	930,00	647,00	737,00	692,00
	Alocação de memória (MB)	8,33	9,56	32,28	9,91	23,50	16,72
	Número de colisões	3,00	2,00	7,00	3,00	5,00	4,00
	AD* - Backward	1ª	2ª	3ª	4ª	5ª	Média
	Tempo total (s)	8.972,00	15.754,00	10.692,00	12.118,00	11.464,00	11.800
	Média Geral (ms)	17,12	24,31	17,22	21,21	17,91	19,55
	Número de iterações	524,00	648,00	631,00	599,00	588,00	598,00
	Alocação de memória (MB)	4,63	5,51	5,53	5,14	5,11	5,18
	Número de colisões	2,00	3,00	2,00	3,00	2,00	2,67
ANA*	ANA* - Forward	1ª	2ª	3ª	4ª	5ª	Média
	Tempo total (s)	125.431,00	59.193,00	166.021,00	61.793,00	171.562,00	116.800
	Média Geral (ms)	41,80	73,90	50,17	65,80	45,00	55,33
	Número de iterações	3001,00	801,00	3308,00	954,00	3786,00	2.370,00
	Alocação de memória (MB)	50,38	68,54	48,08	65,69	45,66	55,67
	Número de colisões	23,00	11,00	21,00	16,00	21,00	18,33
	ANA* - Backward	1ª	2ª	3ª	4ª	5ª	Média
	Tempo total (s)	20.167,00	59.699,00	69.398,00	47.919,00	51.317,00	49.700
	Média Geral (ms)	32,06	56,43	73,05	45,54	62,18	53,85
	Número de iterações	629,00	1058,00	950,00	935,00	823,00	879,00
	Alocação de memória (MB)	13,67	15,99	14,71	11,00	12,65	13,60
	Número de colisões	4,00	11,00	19,00	11,00	10,00	11,00
R*	R* - Forward	1ª	2ª	3ª	4ª	5ª	Média
	Tempo total (s)	1.796,00	1.823,00	1.718,00	1.617,00	2.046,00	1.800
	Média Geral (ms)	3,32	3,41	4,28	2,41	3,28	3,34
	Número de iterações	541,00	535,00	524,00	564,00	501,00	533,00
	Alocação de memória (MB)	18,24	62,24	49,15	44,71	40,84	43,04
	Número de colisões	2,00	2,00	1,00	2,00	2,00	1,67
	R* - Backward	1ª	2ª	3ª	4ª	5ª	Média
	Tempo total (s)	3.182,00	3.534,00	3.047,00	3.710,00	3.027,00	3.300
	Média Geral (ms)	5,48	6,14	5,12	6,10	5,06	5,58
	Número de iterações	581,00	576,00	595,00	578,00	590,00	584,00
	Alocação de memória (MB)	10,01	46,24	6,82	37,11	4,98	21,03
	Número de colisões	0,00	1,00	1,00	1,00	1,00	0,67
LazyARA*	Lazy* - Forward	1ª	2ª	3ª	4ª	5ª	Média
	Tempo total (s)	8.963,00	6.723,00	8.117,00	7.283,00	7.914,00	7.800
	Média Geral (ms)	12,59	9,59	11,10	10,89	11,30	11,09
	Número de iterações	712,00	701,00	704,00	709,00	704,00	706,00
	Alocação de memória (MB)	36,11	45,67	38,44	37,68	42,44	40,07
	Número de colisões	4,00	6,00	4,00	6,00	4,00	4,67
	Lazy* - Backward	1ª	2ª	3ª	4ª	5ª	Média
	Tempo total (s)	15.825,00	13.592,00	9.762,00	14.741,00	11.130,00	13.010
	Média Geral (ms)	31,91	13,47	16,60	21,40	19,90	20,66
	Número de iterações	496,00	1009,00	588,00	811,00	586,00	698,00
	Alocação de memória (MB)	133,41	121,50	107,81	127,35	109,94	120,00
	Número de colisões	6,00	8,00	4,00	8,00	4,00	6,00

Versão: Original;

APÊNDICE E:
RESULTADOS COMPARATIVOS ENTRE OS ALGORITMOS AD*, ANA*, R* e LAZYARA* - COM NOVA FUNÇÃO CUSTO E AMBIENTE DINÂMICO

AD* - NFC

AD* - Forward	1ª	2ª	3ª	4ª	5ª	Média
Tempo total (s)	8,92	11,08	9,29	9,97	9,74	9,80
Média Geral (ms)	20,95	19,56	21,97	18,15	22,17	20,56
Número de iterações	434,00	573,00	431,00	502,00	475,00	483,00
Alocação de memória (MB)	323,31	93,10	29,74	49,83	58,50	110,90
Número de colisões	1,00	2,00	1,00	2,00	2,00	1,67

AD* - Backward	1ª	2ª	3ª	4ª	5ª	Média
Tempo total (s)	7,5	6,1	9,1	10,8	8,7	8,4
Média Geral (ms)	16,92	11,99	14,63	16,31	13,22	14,61
Número de iterações	552,00	505,00	674,00	548,00	606,00	577,00
Alocação de memória (MB)	37,81	14,09	29,96	8,97	13,07	20,78
Número de colisões	2,00	1,00	3,00	2,00	2,00	2,00

ANA* - NFC

ANA* - Forward	1ª	2ª	3ª	4ª	5ª	Média
Tempo total (s)	13,09	14,49	17,81	19,09	16,02	16,10
Média Geral (ms)	29,10	19,66	23,18	23,20	20,19	23,07
Número de iterações	798,00	637,00	724,00	701,00	580,00	688,00
Alocação de memória (MB)	2,11	1,81	2,29	2,13	2,41	2,15
Número de colisões	1,00	0,00	1,00	1,00	2,00	1,00

ANA* - Backward	1ª	2ª	3ª	4ª	5ª	Média
Tempo total (s)	17,88	15,95	12,57	17,70	29,40	18,7
Média Geral (ms)	31,31	39,19	32,39	35,85	39,05	35,56
Número de iterações	511,00	482,00	399,00	643,00	550,00	517,00
Alocação de memória (MB)	110,88	52,11	11,30	60,42	55,11	57,96
Número de colisões	0,00	1,00	2,00	1,00	1,00	1,00

R* - NFC

R* - Forward	1ª	2ª	3ª	4ª	5ª	Média
Tempo total (s)	0,914	0,803	1,040	0,823	1,401	1,00
Média Geral (ms)	2,01	2,14	2,04	2,27	2,18	2,13
Número de iterações	504,00	520,00	510,00	507,00	514,00	511,00
Alocação de memória (MB)	18,37	19,31	22,52	18,85	18,81	19,57
Número de colisões	0,00	0,00	0,00	0,00	0,00	0,00

R* - Backward	1ª	2ª	3ª	4ª	5ª	Média
Tempo total (s)	2,33	2,16	2,09	2,71	2,71	2,4
Média Geral (ms)	4,34	3,71	3,91	5,14	4,12	4,24
Número de iterações	608,00	571,00	586,00	580,00	595,00	588,00
Alocação de memória (MB)	26,91	28,49	34,66	22,61	30,88	28,71
Número de colisões	0,00	0,00	0,00	0,00	0,00	0,00

LazyARA* - NFC

Lazy* - Forward	1ª	2ª	3ª	4ª	5ª	Média
Tempo total (s)	2,06	2,16	2,92	3,65	2,71	2,7
Média Geral (ms)	4,17	3,67	4,74	5,28	5,10	4,59
Número de iterações	591,00	583,00	571,00	601,00	604,00	590,00
Alocação de memória (MB)	4,33	5,92	5,15	4,95	5,31	5,13
Número de colisões	1,00	2,00	0,00	2,00	1,00	1,33

Lazy* - Backward	1ª	2ª	3ª	4ª	5ª	Média
Tempo total (s)	4,12	3,94	4,36	4,31	3,76	4,1
Média Geral (ms)	5,80	6,82	6,89	4,27	5,71	5,90
Número de iterações	711,00	769,00	619,00	813,00	588,00	700,00
Alocação de memória (MB)	105,29	104,00	112,84	105,06	102,81	106,00
Número de colisões	2,00	0,00	1,00	3,00	1,00	1,33

Versão: Modificada;