COPPE/UFRJ

PLANEJAMENTO DE TRAJETÓRIA PARA UM ROBÔ MÓVEL COM DUAS RODAS UTILIZANDO UM ALGORITMO A-ESTRELA MODIFICADO

Sônia Cristina Bastos de Souza

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Antonio Carneiro de Mesquita Filho.
Jorge Lopes de Souza Leão.

Rio de Janeiro
Dezembro de 2008

PLANEJAMENTO DE TRAJETÓRIA PARA UM ROBÔ MÓVEL COM DUAS RODAS UTILIZANDO UM ALGORITMO A-ESTRELA MODIFICADO

Sônia Cristina Bastos de Souza

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

Prof. Antonio Carneiro de Mesquita Filho, Dr. d'État.

Prof. Jorge Lopes de Souza Leão, Dr. Ing.

Prof. Ramon Romankevicius Costa, D.SC.

Prof. José Franco Machado do Amaral, D.SC.

RIO DE JANEIRO, RJ - BRASIL.
DEZEMBRO DE 2008

Souza, Sônia Cristina Bastos de

Planejamento de Trajetória para um Robô Móvel com duas Rodas Utilizando um Algoritmo A-Estrela Modificado / Sônia Cristina Bastos de Souza. – Rio de Janeiro: UFRJ/COPPE, 2008.

XIII, 97 p.: il.; 29,7 cm.

Orientador: Antonio Carneiro de Mesquita Filho

Jorge Lopes de Souza Leão

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia Elétrica, 2008.

Referencias Bibliográficas: p 55-57

1. Navegação Robótica. 2. Algoritmos de Busca. 3. Algoritmo A-Estrela. I. Mesquita Filho, Antonio Carneiro de et al. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Titulo.

Este trabalho é dedicado aos meus pais Sr. Laurindo e Sra. Vera Lúcia, meus eternos incentivadores e apoiadores. A meus irmãos incentivadores Sergio Henrique Bastos de Souza (Ten Cel Bombeiro Militar do RJ) e Marcos Paulo Bastos de Souza (Cap Ten Fuzileiro Naval da Marinha do Brasil). E ao meu querido sobrinho Pedro Henrique.

Agradecimentos

Neste momento, gostaria de agradecer às pessoas que foram fundamentais para a conclusão dessa dissertação.

Primeiramente a Deus que me guiou e inspirou no decorrer dessa jornada.

Ao Comitê de Treinamento do IBGE que me permitiu cursar o mestrado.

Ao meu orientador Antonio Carneiro Mesquita por seu apóio fundamental no inicio deste curso e por sua objetividade de informações e ações ao longo de todo esse período.

Ao meu orientador Jorge Lopes de Souza Leão pela ajuda na escolha do tema e orientação durante o desenvolvimento da dissertação, bem como pela compreensão e apóio nas dificuldades encontradas no decorrer deste trabalho.

Ao M.Sc Denir Valencio Campos, meu amigo, que além de meu incentivar, várias vezes me ajudou.

Ao meu amigo e companheiro de trabalho do IBGE, Walter Oliveira Vieira que sempre esteve pronto a me apoiar e ajudar no decorrer deste curso.

Ao meu colega de pesquisa M. SC. Salomão Gonçalves de Oliveira Junior.

Enfim, a todos que me aconselharam nos momentos difíceis e que me ajudaram a entender que não importa o grau de dificuldade nem a imensidão dos obstáculos a serem ultrapassados, pois o mundo não pára diante de nossas dificuldades e limitações. Essas pessoas, mesmo sem saber, de alguma forma contribuíram para o sucesso desse trabalho.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PLANEJAMENTO DE TRAJETÓRIA PARA UM ROBÔ MÓVEL COM DUAS RODAS UTILIZANDO UM ALGORITMO A-ESTRELA MODIFICADO

Sônia Cristina Bastos de Souza

Dezembro / 2008

Orientadores: Antônio Carneiro de Mesquita Filho
Jorge Lopes de Souza Leão

Programa: Engenharia Elétrica

A navegação, fundamental à robótica móvel, é uma das tarefas mais complexas na problemática da locomoção de robôs móveis. Tal complexidade advém do fato de que a navegação deve integrar sensoriamento, atuação, planejamento, arquitetura, hardware, eficiência e computação. Assim, a integração de todos esses pontos é inerente à obtenção de uma boa navegação robótica. Nesse sentido, o planejamento de trajetória em robótica móvel objetiva prover aos robôs a capacidade de planejar seus próprios movimentos, sem a necessidade de interferência humana. Além disso, a elaboração de um plano de movimentação é uma tarefa com elevado grau de dificuldade. Assim, nesse contexto, este trabalho propõe-se utilizar o método de decomposição celular para a discretização do espaço de estados de um robô e o desenvolvimento e implementação de uma estratégia baseada na modificação de um algoritmo A-Estrela. Do mesmo modo para a geração de trajetórias, apresenta uma resolução baseada no caminho obtido pelo planejador e nas restrições cinemáticas do robô. Além disso, propõe validar a proposta por meio do envio da trajetória final codificada via radiofreqüência para um robô móvel que ao final da recepção executa perfeitamente o trajeto planejado.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

TRAJECTORY PLANNING FOR A MOBILE ROBOT WITH TWO WHEELS USING A MODIFIED ALGORITHM A-STAR.

Sônia Cristina Bastos de Souza

December / 2008

Advisors: Antônio Carneiro de Mesquita Filho

Jorge Lopes de Souza Leão

Department: Eletrical Engeneering

The navigation, key to mobile robotics, is one of the most complex problems in the locomotion of mobile robots. This complexity arises from the fact that navigation should integrate sensing, actuation, planning, architecture, hardware, computing and efficiency. Thus, the integration of all these points is inherent in obtaining a good robotic navigation. In that sense, the trajectory planning in mobile robotics aims to provide robots the ability to plan their movements without the need for human interference. Furthermore, developing a plan for handling is a task with a high degree of difficulty. So, in that context, this study proposes to use the decomposition method of cellular discretization to the area of states of a robot and developing and implementing a strategy based on the modification of an A-Star algorithm. Similarly to generate trajectories, presented a resolution based on the path planner and obtained by the restrictions of cinematic robot. It also proposes validate the proposal by sending us the final trajectory via coded radio frequencies for a mobile robot that performs the receiving end of the course perfectly planned.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

Nomenclatura

API - Application Programming Interface (Interface de Programação de Aplicativos)

AWT - Abstract Windowing ToolKit (kit de Ferramentas Abstratas para Janelas)

SWING – Kit de ferramenteas que estende/substitui a AWT.

1. Introdução

“Se todo instrumento pudesse, dada uma ordem, trabalhar por si mesmo, como um arco que toca sozinho a cítara, os empreendedores poderiam dispor menos dos trabalhadores e os patrões dos escravos”. A citação de Aristóteles (séc. IV a.C.), falando do antigo desejo de utilizar máquinas inteligentes para realizar tarefas humanas, ainda que em épocas tão remotas, já era o prenúncio da motivação para criação dos diversos sistemas robotizados atuais. Aristóteles, sem saber, já sonhava com os robôs que, hoje, fazem parte da realidade de nossa sociedade.

É fato que há algumas décadas atrás, os robôs faziam parte apenas da ficção científica, fruto da imaginação do homem. Mas, atualmente, os robôs e os sistemas robotizados atuam até nas tarefas mais simples do dia a dia. Exemplos disso são os elevadores que, com simples toque de um botão, movem -se de um andar a outro, param e abre-se a porta, tudo automaticamente. Bem como, as operações bancárias realizadas nos caixas eletrônicos sem nenhuma intervenção humana. Ou ainda, o portão da garagem que pode ser acionado de longe, ainda dentro do veículo. Semelhantemente, na industria, a precisão dos robôs é indispensável. E na medicina, vidas são salvas graças ao avanço tecnológico. Dessa forma, de simples mecanismos a sofisticados robôs, como os de pesquisa submarina ou interplanetária, vê-se uma abrangência muito grande da tecnologia. Rigorosamente falando, os robôs de hoje executam cada vez mais as tarefas que o ser humano não pode ou não quer fazer.

Um robô pode ser rápido, forte e mais preciso que um humano, entretanto ele tem pouca ou quase nenhuma inteligência e na maioria das vezes é cego e insensível. Apesar disto, quando corretamente programado, ele pode realizar atividades complexas com sucesso. Nesse sentido, dentro da área da robótica, os robôs móveis têm recebido atenção especial. Eles se propõem a realizar uma variedade de tarefas mais complexas que seus antecessores, os robôs industriais. Para isso, são necessárias técnicas que lhes permitam interagir de forma efetiva com o ambiente. Neste contexto, a parte mais essencial desta interação é o sistema de navegação. Essa habilidade representa um dos maiores problemas em robótica móvel. Conceitualmente, em robótica, navegar consiste em guiar um robô em um espaço de trabalho durante um determinado intervalo de tempo, por um caminho que possa ser percorrido e que leve o robô de uma posição e orientação iniciais para uma posição e orientação finais. Esta é a principal tarefa que um robô móvel deve executar. Essa tarefa envolve subproblemas tais como a localização do robô no espaço de trabalho, o planejamento de um caminho admissível, a geração de uma trajetória e a sua execução. Além disso, robôs móveis autônomos com rodas possuem restrições

cinemáticas, ou seja, restrições não-holonômicas, que também influenciam na tarefa de navegação.

Ainda sob o enfoque da navegação robótica, cabe ressaltar que o planejamento de caminhos normalmente é realizado em três etapas. A primeira consiste em definir o espaço de configuração, ou seja, o espaço de estados do robô definidos pela posição, pela orientação e pelos seus ângulos de articulação. Já a segunda constitui-se na aplicação de técnicas sobre o espaço de configurações do robô para determinar o conjunto de estados válidos para a busca. Para tal, existe na literatura (RUSSELL S. & NORVING P., 2003) sobre planejamento de caminhos de robôs diversos métodos para essa finalidade, porém os mais comumente utilizados são a decomposição celular e a esqueletização. O primeiro método apresenta a vantagem de levar a uma pesquisa discreta; já o segundo calcula um “esqueleto” unidimensional do espaço e tende a não ser ótimo e geralmente não traz a melhor solução. Finalmente, a terceira etapa do planejamento de caminhos é composta pela busca no espaço de configuração, isto quer dizer, consiste na utilização de qualquer algoritmo de busca para alcançar a posição e a orientação desejada.

Este trabalho aborda principalmente os subproblemas do planejamento de caminhos e a geração de trajetórias aplicados a um robô móvel com duas rodas. Assim, para obter o planejamento de caminho livre de obstáculos, propõe-se utilizar o método de decomposição celular para a discretização do espaço de estados de um robô e o desenvolvimento e implementação de uma estratégia baseada na modificação de um algoritmo A-Estrela. Do mesmo modo para a geração de trajetórias, apresenta uma resolução baseada no caminho obtido pelo planejador e nas restrições cinemáticas do robô Além disso, propõe validar a proposta por meio do envio da trajetória final codificada via radiofreqüência para um robô móvel que ao final da recepção executa perfeitamente o trajeto planejado. É relevante ressaltar que também faz parte da validação deste trabalho o planejador PCT (Planejador de Caminhos e Trajetórias) desenvolvido na linguagem Java, ele implementa as funções de planejamento de caminhos, geração e transmissão das trajetórias. E para legitimar ainda mais este trabalho, utilizou-se o robô apresentado por (OLIVEIRA JR, 2008) para receber e executar a trajetória resultante do planejamento.

2. Navegação Robótica Móvel

A navegação de robôs móveis refere-se à habilidade de um robô locomover-se de uma posição à outra com base nas informações parciais de sua posição física e do ambiente que o contém. No contexto da robótica móvel o principal problema enfrentado no desenvolvimento de robôs móveis é a navegação, a qual envolve tarefas básicas como mapeamento e planejamento.

O mapeamento consiste na modelagem do ambiente que contém o robô através do uso de mapas obtidos pelo sistema sensorial ou previamente armazenados.

Já o planejamento atribui aos robôs a capacidade de idealizar seus próprios movimentos, sem a necessidade da interferência humana. A tarefa de planejar pode ser considerada como uma simplificação na qual as questões principais são isoladas e estudadas em profundidade antes de se analisar as dificuldades adicionais. Assim, considera-se que o robô é o único objeto móvel no espaço de trabalho. Ignoram-se as propriedades dinâmicas do robô, evitando, dessa forma, as questões temporais. Do mesmo modo, a movimentação também é reduzida para uma movimentação sem contato, na qual as possíveis interações entre o robô e os objetos físicos do ambiente, são desprezadas. Além disso, assume-se que o robô é um único objeto rígido, isto é, não apresenta partes móveis, como braços, pernas ou outros, e que sua movimentação é limitada apenas pelos obstáculos dispostos espaço de trabalho. Enfim, o problema de planejamento em navegação robótica pode ser definido como a busca de um percurso a ser seguido, ou a seqüência de ações a serem tomadas para que o robô possa, saindo de um ponto de partida, chegar a um ponto meta, evitando a colisão com um conjunto de obstáculos conhecidos (JAHANBIN & FALLSIDE, 1988; SCHALKOFF, 1990).

O método de planejamento do caminho tem duas vertentes principais: o global e o local, que devem ser utilizados simultaneamente para que se possa ter um melhor desempenho do robô em um ambiente real.

O planejamento global é responsável pelo mapeamento do ambiente onde está inserido o robô em um modelo simplificado, estático, pré-gravado (CROWLEY, 1985), que lhe permite traçar um caminho mesmo por um local que não consiga perceber com seus sensores, seja por estar fora do alcance ou por estar obstruído por algum obstáculo (FIRBY, 1994). Já o local é responsável pela navegação curta (FIRBY, 1994), baseada nos valores coletados pelos seus sensores. Uma vez modelado o ambiente, o planejamento local será o responsável pela localização do robô neste mundo simplificado e dinâmico, devido à constante atualização dos dados não fornecidos pelo modelo global. (JAHANBIN & FALLSIDE, 1988), sendo as

aproximações geométricas, por meio de linhas retas, uma excelente solução.

Existem diferentes técnicas matemáticas para se alcançar o planejamento do caminho, sendo os Roadmaps, as Decomposições em Células e o Campo Potencial as principais. A seguir, é apresentada uma visão geral dessas técnicas.

2.1. Abordagem de Planejamento - Roadmaps

A idéia desta abordagem consiste em reduzir as informações ambientais a um grafo representando os possíveis caminhos (OTTONI G. L., 2000). Uma vez que o roadmap é construído, ele é utilizado como um conjunto de caminhos padronizados. Então, o planejamento de trajetórias reduz-se a conectar as posições iniciais e finais do robô ao roadmap e buscar um caminho entre estes dois pontos. Se existir algum caminho, ele será dado pela concatenação de três subcaminhos: um conectando a posição inicial até algum ponto do roadmap, outro subcaminho pertencente ao roadmap e, finalmente, um subcaminho que leve do último ponto escolhido do roadmap até a posição final desejada.

Geralmente, métodos roadmaps são rápidos e simples de se implementar, mas eles não fornecem uma boa representação das informações do ambiente. Vários métodos baseados nesta abordagem foram propostos. Entre eles: grafos de visibilidade e diagramas de Voronoi.

2.1.1. Grafos de Visibilidade

Um grafo de visibilidade é formado pela ligação das posições de um ambiente de acordo com a visibilidade de uma posição a outra. Inicialmente, este conceito apareceu na literatura da ciência das informações geográficas (DAVIDSON ET AL. ,1993) no contexto da determinação do roteamento através de um conjunto de linhas de visadas preferenciais. Esses grafos também são muito utilizados no campo da navegação robótica em ambientes complexos bi e tridimensionais.

No caso da navegação bidimensional, onde um robô encontra-se em um ambiente entre um conjunto de obstáculos poligonais, o grafo de visibilidade baseia-se no conjunto dos vértices dos polígonos e de sua visibilidade mútua. Assim, esse grafo contém todos os menores trajetos existentes no ambiente. Nos casos tridimensionais mais complexidade é introduzida porque os menores caminhos podem estar tangentes às arestas de um obstáculo poliédrico, então estes também devem ser considerados como vértices no grafo, mas o princípio é o mesmo.

O caráter de um grafo de visibilidade é dependente da complexidade das

propriedades geométricas do ambiente do qual ele é derivado. Sendo isto que o torna um objeto interessante de estudo. Nesse sentido, um grafo de visibilidade genérico pode conectar qualquer conjunto de pontos e várias técnicas de análise para construção das medidas para estas localizações podem ser utilizadas.

Resumidamente, um grafo de visibilidade é obtido gerando-se segmentos de reta entre os pares de vértices dos obstáculos. Todos os segmentos de reta que estiverem inteiramente na região do espaço livre são adicionados ao grafo. No planejamento de trajetória a posição origem e a de destino são representadas como vértices, gerando, assim um grafo de conectividade onde um algoritmo de procura deve ser utilizado para encontrar um caminho livre.

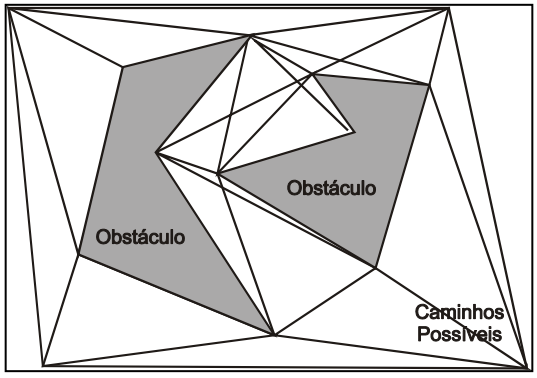


Figura 1 -Grafo de visibilidade

Na Figura 1 o caminho mais curto encontrado no grafo de visibilidade é o caminho ótimo para o problema especificado. Mas os caminhos encontrados no grafo esbarram nos obstáculos. Isso não é aceitável em um problema de navegação robótica, por esse motivo os obstáculos devem ser modificados, criando-se um espaço de configuração.

Um método de navegação que utilize grafo de visibilidade não permite navegar em um ambiente com obstáculos móveis, além disso, a localização do robô móvel deve ser conhecida durante toda a navegação.

2.1.2. Diagramas de Voronoi

Os diagramas de Voronoi foram inventados e reinventados em várias épocas em diversos contextos diferentes (ERICKSON J. 2008). E é por isso que eles podem ser encontrados na análise da fragmentação cósmica de Descartes, mas sua primeira definição geralmente é atribuída a Voronoi ou a Dirichlet.

O nome Theissen foi por vezes associado a estes diagramas por meteorologistas, e os físicos podem conhecê-los por qualquer um dos nomes Wigner,

Seitz, ou Broullin. Já os biólogos têm redescoberto este diagrama; Brown os discutiu em 1965, e Mead os descreveu em 1966.

Uma definição formal para o Diagrama de Voronoi é: dado um conjunto S de n pontos no plano deseja-se determinar para cada ponto p de S qual é a região V(p) dos pontos do plano que estão mais próximos de p do que de qualquer outro ponto em S. As regiões determinadas por cada ponto formam uma partição do plano chamada de Diagrama de Voronoi [Voronoi, 2004]. A Figura 8 apresenta o Diagrama de Voronoi aplicado a 14 (quatorze) pontos. Cada ponto do espaço R2 possui pelo menos um vizinho mais próximo. Logo, ele pertence à pelo menos um polígono de Voronoi. Assim, o Diagrama de Voronoi cobre completamente o plano.

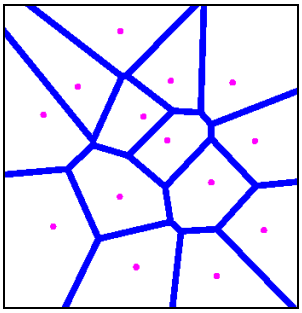


Figura 2 -Diagrama de Voronoi gerado a partir de 14 pontos.

A construção do Diagrama de Voronoi a partir da triangulação de Delaunay (REZENDE ET AL., 2000). é a opção mais comum quando se recorre a algoritmos de triangulação. No entanto, as triangulações de Delaunay a partir do Diagrama de Voronoi são imediatas estabelecendo-se a união entre os pontos que partilham as arestas do diagrama de Voronoi.

Bem como nas demais áreas de pesquisa, na robótica, mais especificamente no planejamento, o diagrama de Voronoi se apresenta como um excelente método de otimização de dados e algoritmos para um melhor desempenho dos sistemas. No planejamento de caminhos, a região bidimensional em que o robô se locomove geralmente contém obstáculos, cada um deles pode ser representado por polígonos côncavos ou convexos. Em um caso como esse, para encontrar o diagrama generalizado de Voronoi para esta coleção de polígonos, pode-se calcular o diagrama através de uma aproximação, convertendo os obstáculos em uma série de pontos. Feito isso, o próximo passo é calcular o diagrama para esta coleção de pontos. E em seguida, os segmentos do diagrama que interceptam algum obstáculo são eliminados. Uma vez calculado o diagrama, é preciso adicionar a posição do robô e o destino ao conjunto de pontos e a partir dele, utilizar algum algoritmo de busca para encontrar um

caminho da posição origem até a posição de destino, que é um subconjunto do diagrama de Voronoi. A vantagem desse método é que ele tende a maximizar a distância entre os obstáculos e o sistema robótico.

2.2. Abordagem de Planejamento - Decomposição Celular

Consiste na decomposição do espaço livre do robô em regiões simples chamadas células, de forma que um caminho entre duas configurações em células diferentes pode ser facilmente gerado. Um grafo não-direcionado representando a relação de adjacência entre as células é então construído, e sobre este grafo a busca do caminho é realizada. Este grafo é chamado de grafo de conectividade (OTTONI G. L. 2000). Os nós deste grafo são as células extraídas do espaço livre: dois nós estão conectados por uma aresta se e somente se duas células correspondentes são adjacentes. Então, o resultado da busca é uma seqüência de células chamada de canal. Logo, um caminho livre pode ser então computado a partir desta seqüência.

O método de decomposição celular é subdividido em duas categorias: Métodos exatos e aproximados(OTTONI G. L. 2000).

2.2.1. Decomposição Celular Exata

Decompõe o espaço livre em células cuja união é exatamente o espaço livre. Este conjunto de células é dito completo, pois sempre que possível este conjunto permite que um caminho entre duas configurações qualquer seja obtido, desde que seja utilizado um algoritmo de busca apropriado(OTTONI G. L. 2000).

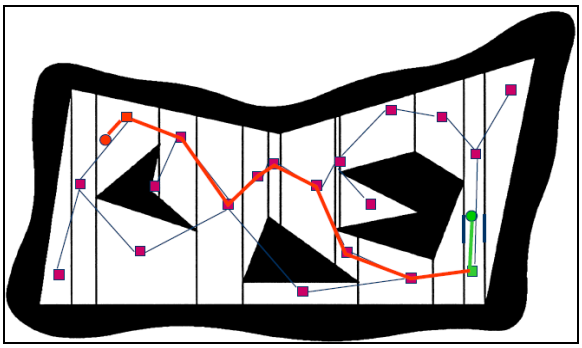


Figura 3 -Decomposição Celular Exata

2.2.2. Decomposição Celular Aproximada

Produzem células de formas pré-definidas cuja união está estritamente inclusa no espaço livre. Geralmente, eles não ser completos, pois dependendo da precisão utilizada podem não encontrar um caminho entre duas configurações mesmo que ele exista. Porém, normalmente, a precisão destes métodos pode ser ajustada arbitrariamente, a custo de espaço de armazenamento e tempo de processamento. Ainda assim, estes métodos são mais simples, por isso, na prática são utilizados com maior freqüência(OTTONI G. L. 2000).

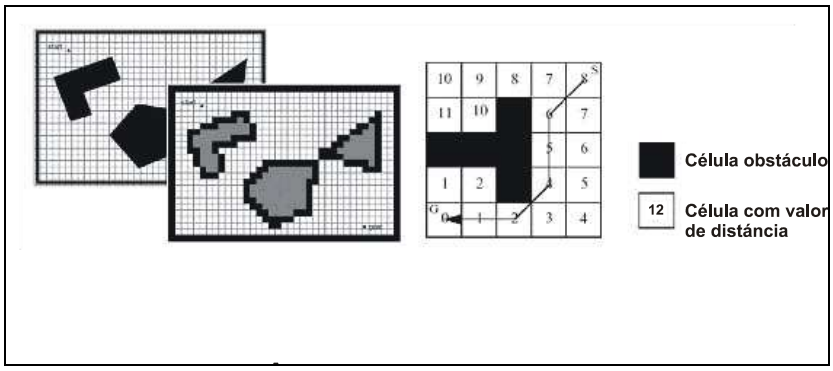


Figura 4 -Decomposição Celular Aproximada.

2.3. Abordagem de Planejamento - Campo Potencial

O método do campo potencial (EDER R. , 2003) consiste em calcular a direção e o sentido da trajetória de acordo com a direção e o sentido da resultante das forças que se aplicam ao sistema robótico em cada instante da navegação. A possibilidade de planejamento de trajetórias em espaços de configuração dinâmicos é umas das principais características deste método, uma vez que, não é necessária a criação prévia de nenhuma estrutura de dados, na qual o planejamento de trajetória irá basear-se. A metáfora sugere que, dentro do espaço de trabalho, o robô (carga de prova) é submetido à ação de um determinado potencial, que neste ambiente é determinado pela configuração do alvo e dos obstáculos. Neste caso, o alvo é representado por uma carga de sinal oposto à carga de prova e os obstáculos, por cargas com sinais iguais aos da carga de prova. O potencial resultante deve produzir forças de repulsão entre o robô e os obstáculos, bem como uma força de atração entre o rôbo e o alvo.

Outra vantagem deste método em relação aos anteriores é o seu desempenho. Por isso, devido a pouca complexidade dos cálculos exigidos para a elaboração do planejamento de trajetória, a aplicação deste método é recomendada para espaços de configuração dinâmica, visto que a alteração da configuração do espaço deve ser analisada da forma mais rápida possível para que eventuais choques com obstáculos dinâmicos sejam evitados.

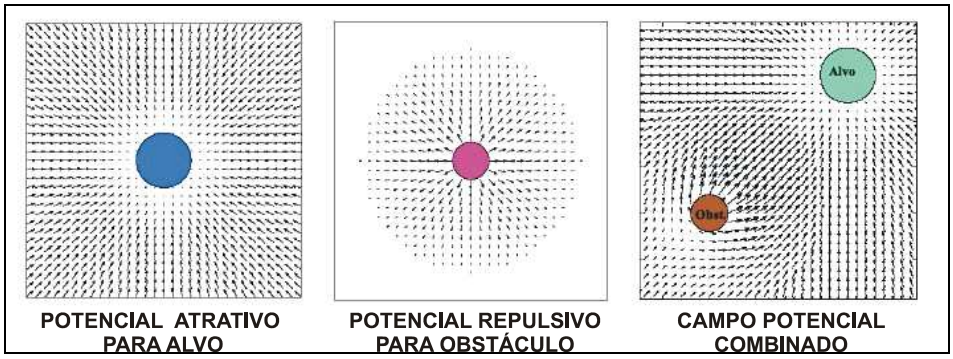


Figura 5 - Campo Potencial

Apesar dos bons resultados fornecidos pelos métodos de campos potenciais, eles possuem uma série de problemas inerentes, que independem de uma implementação em particular, tais como: situações de armadilha devido a mínimos locais (comportamento cíclico); situações nas quais eles não conseguem passar através de obstáculos muito próximos, situações de oscilação na presença de obstáculos ou em corredores estreitos e outros.

Dentre os problemas citados, o mais conhecido é o problema dos mínimos locais ou situações de armadilha. Um mínimo local pode ocorrer quando um robô entra em uma área sem saída (por exemplo, um obstáculo em forma de U). Estas armadilhas podem ser criadas por diversas configurações de obstáculos.

3. Algoritmos de Busca

Algoritmos de busca são técnicas aplicadas a problemas de alta complexidade teórica e que não são resolvidos com métodos de programação convencionais, principalmente os de natureza puramente numérica.

Dessa forma, para que a complexidade dos problemas do mundo real seja reduzida é necessário obter algum conhecimento específico do domínio do problema. Além disso, a redução do processo de solução por meio de algum algoritmo de busca, também diminui essa complexidade consideravelmente. Do mesmo modo a utilização de heurísticas, geralmente, reduz a explosão combinatória das possibilidades de busca. Com isso, o trabalho humano é reduzido à atuação empírica de identificação e formalização das representações de estados; dos parâmetros heurísticos; das operações de transformações atômicas, das transformações que atinjam a solução com tempos e tamanhos de memória aceitáveis.

Sob o aspecto teórico, um grafo de pesquisa com estados associados, operadores (também conhecido como ações) e custos pode representar o relacionamento entre os algoritmos de busca, conforme a figura 9.

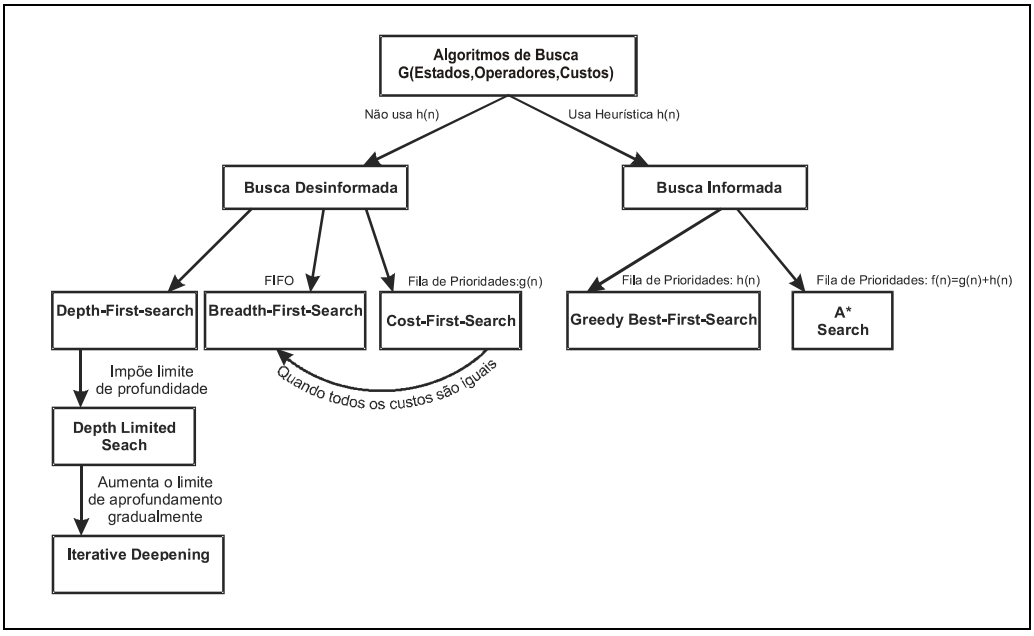


Figura 6 - Relacionamento entre os algoritmos de busca

A idéia principal dos algoritmos de busca é a exploração simulada do espaço de estados via a geração dos sucessores dos estados já explorados. Nesse enfoque, as estratégias de busca visam escolher a ordem de expansão dos estados em análise. Elas dividem-se em estratégias não-informadas e informadas.

As estratégias de busca não informada, também conhecida como busca cega ou blind search são estratégias de busca baseadas em tentativas de solução por força bruta onde o único conhecimento que pode ser aplicado ao problema para determinar o próximo passo rumo a uma solução é dado por uma função de enfileiramento. Tais estratégias podem encontrar uma solução para o problema simplesmente gerando novos estados e testando os contra o objetivo. Dentre estas estratégias as mais importantes são conhecidas como busca em largura (breadth search), busca em profundidade (depth search), busca em profundidade limitada (depth limited search) e busca com aprofundamento iterativo (iterative deepening search).

As estratégias de busca informada diferem das estratégias de busca não informada por acrescentarem uma informação a mais na determinação da ordem de expansão dos estados durante o processo de busca. Esta informação é chamada de função de avaliação, ou heurística, e consiste em uma forma de mensurar a probabilidade de um estado convergir para uma solução baseado em seu estado corrente. As estratégias mais conhecidas de busca informada são a busca “gulosa” (greedy search) e a busca A* (A* search).

3.1. Heurística

A palavra heurística quando usada como substantivo, identifica a arte ou a ciência do descobrimento, uma disciplina suscetível de ser investigada formalmente. Quando aparece como adjetivo, refere-se a coisas mais concretas, como estratégias heurísticas, regras heurísticas ou silogismos e conclusões heurísticas. Naturalmente que estes usos estão intimamente relacionados já que a heurística usualmente propõe estratégias heurísticas, que guiam o descobrimento.

A heurística, como parte do método científico, visa favorecer o acesso a novos desenvolvimentos teóricos, ou descobertas empíricas. Sendo assim, um procedimento heurístico é um método de aproximação das soluções dos problemas, que não segue um percurso claro, mas se baseia na intuição e nas circunstâncias a fim de gerar novos conhecimentos. É o oposto do procedimento algorítmico. Sendo assim, a heurística de uma teoria deve indicar os caminhos e possibilidades a serem aprofundadas na tentativa de torná-la uma teoria progressiva, isto é, capaz de garantir um desenvolvimento empírico, provedor de novos fatos não percebidos no momento da elaboração do núcleo dessa teoria. Em IA, geralmente, a utilização de heurística é necessária quando o problema não tem solução exata, ou quando o problema tem uma solução exata, mas é demasiadamente complexo para permitir uma solução de força bruta. Portanto, as heurísticas também são falíveis, porque elas dependem de

informações limitadas, e por isso podem conduzir a uma solução otimizada ou a um beco sem saída.

No sentido de obter sempre boas soluções, os algoritmos de busca heurística utilizam informações sobre o problema para guiar o trajeto da busca no espaço de procura. Essas buscas beneficiam-se da utilização de algumas funções que estimam o custo do estado atual até o estado objetivo, presumindo que tal função seja eficiente. Geralmente, uma heurística incorpora conhecimento específico do domínio do problema para aprimorar a eficiência de buscas cegas.

A seguir serão apresentadas algumas métricas comumente utilizadas em funções heurísticas.

- **Métrica da distância Manhattan**

Essa métrica pode ser definida como a distância entre dois pontos no espaço Euclidiano, em um sistema de coordenadas cartesianas fixo, isto é, consiste na soma das diferenças absolutas dos valores dos atributos. Por exemplo, dados dois pontos "a" e "b" a distância Manhattan é definida como $d(a,b) = | X_b - X_a | + | Y_b - Y_a |$.

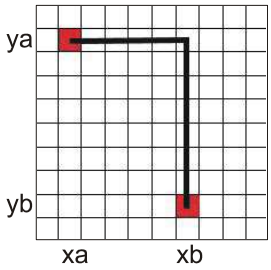


Figura 7 -Distância Manhattan entre dois pontos

- **Métrica da distância Euclidiana**

Dados dois pontos "a" e "b" no espaço Euclidiano a distância euclidiana é definida como $d(a,b) = \sqrt{(X_a - X_b)^2 + (Y_a - Y_b)^2}$. Nem sempre essa métrica é satisfatória. Já que cada coordenada tem o mesmo peso para o cálculo da distância, quando estas coordenadas representam medidas que estão sujeitas a situações aleatórias de diferentes magnitudes.

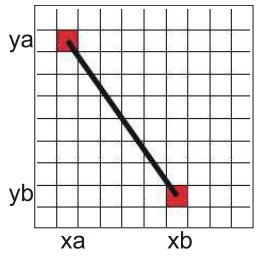


Figura 8 -Distância Euclidiana entre dois pontos

- **Métrica da distância Diagonal**

A métrica da distância diagonal combina aspectos da métrica da distância Manhattan e da Euclidiana. Sendo assim, a métrica resultante é admissível.

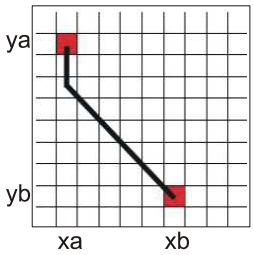


Figura 9 -Distância usando caminho diagonal

O valor resultante dessa métrica é constituído por duas partes, uma reta e uma diagonal. O número de passos diagonais que podem ser tomados é definido por $npd = \min\{|x_a - x_b|, |y_a - y_b|\}$

O algoritmo de busca desenvolvido neste trabalho é fundamentado em algoritmos de busca informados. Devido a isso, serão apresentados os conceitos teóricos envolvidos nos três algoritmos de busca informada que guiam este trabalho: Algoritmo de busca pela Melhor Escolha (Best First Search), Dijkstra e A*(A-Estrela).

3.2. Algoritmo de Busca pela Melhor Escolha (Best first Search)

Este algoritmo (RUSSELL S. & NORVING P., 2003), utilizando uma função de avaliação f(n) como meio de seleção, expande o estado que tiver o menor custo estimado até a seu estado alvo. Embora, esse algoritmo encontre um caminho rapidamente, nem sempre este é o melhor.

Similar aos algoritmos de busca em Largura (Breadth First) e Profundidade (Depth First), o algoritmo de Busca pela Melhor Escolha mantém uma lista fechada dos estados para os quais ele encontrou caminhos e uma lista aberta dos estados filhos destes estados fechados. Entretanto, ao contrário dos algoritmos de busca sem informação, a Busca pela Melhor Escolha utiliza uma função heurística chamada $h_{(n)}$ para guiar-se até o seu alvo. Essa função representa o custo estimado do caminho mais econômico de um estado n até um estado objetivo.

Geralmente, a Busca pela Melhor Escolha implementa sua lista aberta como uma fila de prioridades. Nessa fila os estados são ordenados somente pelos valores de h . E isto, geralmente, possibilita encontrar o estado objetivo rapidamente, mas se a função de avaliação for imprecisa a busca poderá se perder. Além disso, o percurso encontrado poderá não ser o melhor.

A Busca pela Melhor Escolha apresenta os mesmos problemas da busca em profundidade – não é ótima e é incompleta (pode entrar em um caminho infinito e nunca retornar para testar outras possibilidades). Sua complexidade em espaço é b^m e em tempo é b^m, para o pior caso.

- **Descrição do Algoritmo**

A cada iteração do algoritmo, o primeiro estado da lista aberta é movido para a lista fechada. Se este for o estado meta, o algoritmo retorna o caminho. Caso contrário, o algoritmo adiciona à lista aberta todos os estados que estão ligados a este estado fechado e que ainda não estão na lista fechada. Se algum desses estados filhos já estiver na lista aberta ou fechada, o algoritmo certifica-se para que a menor das duas soluções parciais do caminho seja armazenada, pois soluções duplicadas não são registradas. Assim, ao atualizar o histórico dos pais dos estados na lista aberta e fechada, quando esses estados são redescobertos, provavelmente o algoritmo encontrará um caminho menor até a sua meta. Ou seja, o algoritmo avalia heuristicamente os estados em aberto, e ordena a lista de acordo com esses valores

heurísticos. Isso faz com que o “melhor” estado seja colocado na frente da lista aberta.

Por outro lado, é evidente que estas estimativas são de natureza heurística e, por isso, o próximo estado a ser analisado pode ser proveniente de qualquer nível do espaço de busca.

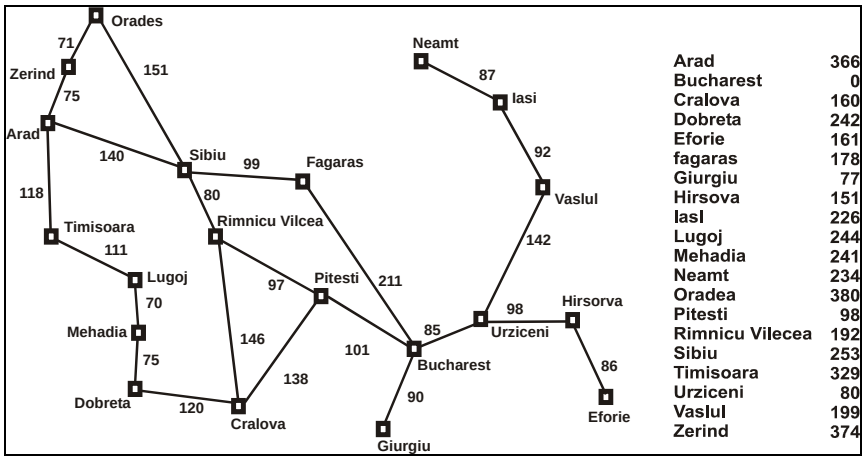


Figura 10 -Mapa simplificado de parte da Romênia

A figura 10 ilustra o caso do problema de localização de rotas na Romênia, usando a heurística de distância em linha reta, h_{DLR}. Se o objetivo é Bucareste, é preciso conhecer as distâncias em linha reta até Bucareste. A seguir as figura 11, 12 e 13 demonstram o progresso da Busca pela Melhor Escolha para encontrar um caminho de Arad até Bucareste.

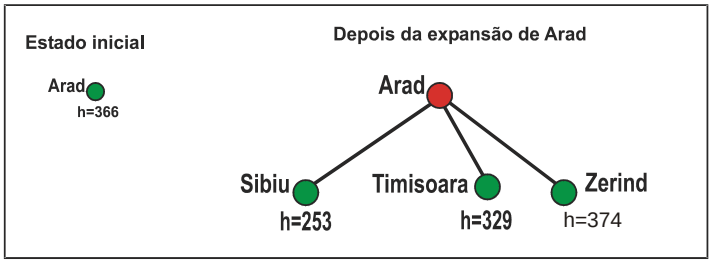


Figura 11 - Busca pela Melhor Escolha para Bucareste – primeira expansão

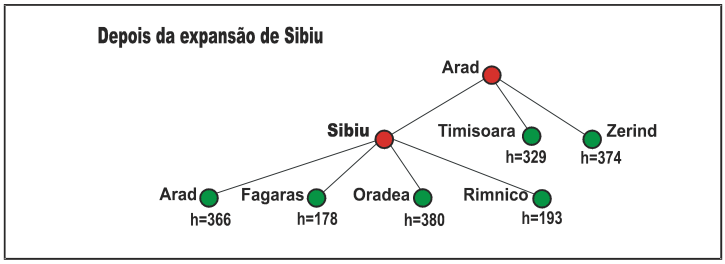


Figura 12 -Busca pela Melhor Escolha para Bucareste – segunda expansão

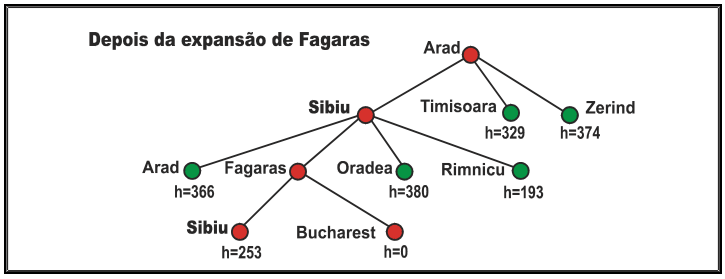


Figura 13 - Busca pela Melhor Escolha para Bucareste – última expansão

- **Pseudocódigo**

BEST-FIRST*(estadoInicial)*
Inicio
nodos ← CRIA-FILA(estadoInicial) **loop**
se nodos é vazio **então retorna** falha
nodo ←OBTEM-MELHOR-NODO(nodos) **se** É-OBJETIVO(nodo) **então retorna** nodo
novos-nodos ←EXPANDE(nodo)
nodos ← ADD-FILA(nodos,novos-nodos) **fim_loop**
Fim
OBTEM-MELHOR-NODO(nodos): implementa a função *f(n)= h(n)*

3.3. Algoritmo de Dijkstra

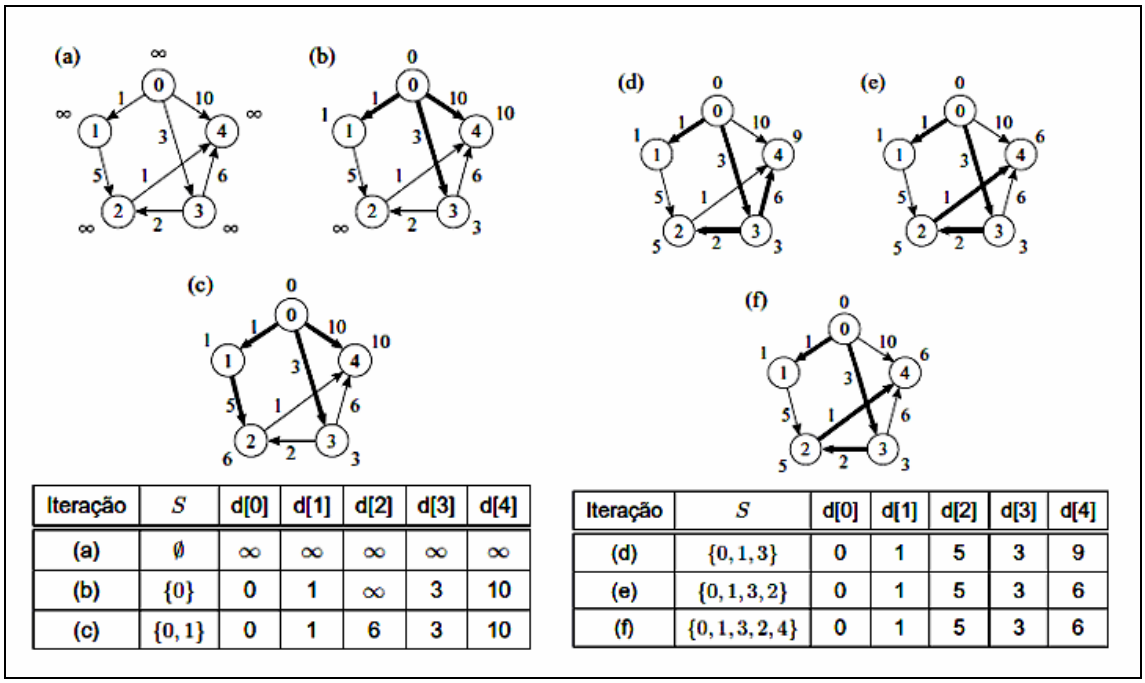
O algoritmo de Dijkstra usa uma estratégia gulosa: sempre escolhe o vértice mais leve (ou o mais próximo) para adicionar ao conjunto da solução. Por isso, sua tática é calcular o caminho de custo mínimo entre os vértices de um grafo. Assim, ao escolher um vértice como raiz da busca, o custo mínimo deste vértice para todos os demais vértices do grafo é calculado. Suas operações são simples e proporcionam um bom nível de desempenho. Porém, a estratégia deste algoritmo não garante a exatidão da solução caso haja a presença de arcos com valores negativos.

A maior desvantagem do algoritmo deve-se ao fato de sua busca ser cega, e isso resulta em elevado consumo de tempo e desperdício de recursos. Outra desvantagem é que ele não pode tratar arestas negativas. Pois isto leva a grafos acíclicos que na maioria das vezes não conseguem levar ao menor caminho corretamente.

- **Descrição do Algoritmo**

O algoritmo funciona (KOLLN W L, 2006) mantendo para cara vértice v o custo d[v] do caminho mais curto encontrado entre s e v. Inicialmente este valor é 0 para o vértice de origem s (d[s]=0), e infinito para todos os outros vértices, representando o fato de que não conhecemos nenhum caminho levando a estes vértices (d[v]= ∞ para cada v em V, exceto s). Quando o algoritmo termina, d[v] será o custo do caminho mais curto de s até v, ou in_nito, se tal caminho não existir. A operação básica do algoritmo de Dijkstra é uma relaxação de arestas. Se existe aresta de u até v, então o menor caminho conhecido de s até v (d[u]) pode ser estendido como o caminho de s até v adicionando uma aresta (u,v) aoseu _nal. Este caminho terá o tamanho d[u]+w(u,v). Se este é menor do que o atual d[v], podemos substituir o atual valor de d[v] pelo novo valor. A relaxação de arestas é aplicada até que todos os valores d[v] representem o custo do menor caminho de s até v. O algoritmo é organizado de modo que cada aresta (u,v) terá sido relaxada apenas uma vez, quando d[u] tiver alcançado seu valor _nal. A noção de "relaxação"vem de uma analogia entre estimar o menor caminho e o cumprimento de uma "mola helicoidal de tensão", que não é susceptível a compressões. Inicialmente o caminho de menor custo é uma estimativa exagerada, assim como uma mola esticada. Quando o caminho mais curto é encontrado, o custo estimado é reduzido, e a mola é relaxada. Eventualmente, o caminho mais curto, se existir, é encontrado e a mola estará relaxada a seu cumprimento normal.

O algoritmo mantém dois conjuntos de vértices, S e Q. O conjunto S contém todos os vértices para os quais sabemos os valores d[v] que já é o custo do menor caminho, e o conjunto Q contém todos os outros vértices. O conjunto S começa vazio, e em cada passo um vértice é movido de Q até S. Este vértice é escolhido como o vértice com o menor valor de d[u]. Quando um vértice u é movido para S, o algoritmo relaxa cada aresta de saída (u,v).



Iteração	S	d[0]	d[1]	d[2]	d[3]	d[4]
(a)	$\emptyset$	∞	∞	∞	∞	∞
(b)	$\{0\}$	0	1	∞	3	10
(c)	$\{0,1\}$	0	1	6	3	10

Iteração	S	d[0]	d[1]	d[2]	d[3]	d[4]
(d)	$\{0,1,3\}$	0	1	5	3	9
(e)	$\{0,1,3,2\}$	0	1	5	3	6
(f)	$\{0,1,3,2,4\}$	0	1	5	3	6

Figura 14 -Algoritmo de Dijkstra

- **Pseudocódigo**

```
FUNCTION DIJKSTRA(G, W, S)
  FOR EACH VERTEX V IN V[G]
  D[V] := INFINITY
  PREVIOUS[V] := UNDEFINED
  D[S] := 0
  S := EMPTY SET
  Q := V[G]
  WHILE Q IS NOT AN EMPTY SET
  U := EXTRACT_MIN(Q)
  S := S UNION {U}
  FOR EACH EDGE (U,V) OUTGOING FROM U
  IF D[U] + W(U,V) < D[V]
  D[V] := D[U] + W(U,V)
  PREVIOUS[V] := U
```

3.4. Algoritmo A-Estrela (A-Star)

Nos problemas que envolvem busca, geralmente, o maior interesse é encontrar o caminho mais eficiente ou o de menor custo entre dois pontos previamente estabelecidos. Nesse sentido, quando um algoritmo de busca é utilizado para alcançar um destino inacessível, nenhum resultado será obtido, mas, ainda assim, informações úteis podem ser obtidas.

O algoritmo A-Estrela (A*) é o algoritmo de busca em grafos amplamente utilizado (RUSSELL S. & NORVING P., 2003),. Rigorosamente falando, a utilização de grafos em problemas de busca gera árvores que consumem muito tempo e recursos de processamento. A fim de diminuir tais problemas o algoritmo A-Estrela foi concebido com base na utilização de uma heurística para guiar seu processo de busca. Essa heurística, $h_{(n)}$, estima o custo do deslocamento da posição atual até a posição meta. Com isso o algoritmo minimiza o tamanho da árvore de pesquisa e agiliza todo o processo. Somando-se a isso, o A-Estrela também acompanha o custo necessário para se alcançar cada posição, normalmente esse custo é conhecido pela sigla $g_{(n)}$. Nesse sentido, o custo estimado da solução de menor custo passando por n pode ser expresso por $f_{(n)} = g_{(n)} + h_{(n)}$. Além disso, o A-estrela utiliza o conceito das listas abertas e fechadas para controlar a visita às posições. Ou seja, na lista aberta mantêm-se os registros de todas as posições que foram alcançadas, mas que ainda não foram visitadas e expandidas. Já na lista fechada são mantidos todos os registros das posições que já foram visitadas e expandidas.

O algoritmo A-Estrela é completo. Sendo assim, em uma busca qualquer, dado tempo e memória ilimitada, o algoritmo sempre encontrará o menor caminho, se esse caminho existir. Mesmo que a função heurística definida seja altamente inexata, a posição alvo em algum momento será encontrada.

A-Estrela é ótimo, contanto que a função heurística h seja admissível. Dessa forma, ainda que existam vários caminhos, o melhor sempre será encontrado. A complexidade em tempo e espaço desse algoritmo é igual a $o(b^m)$. Porém, o excessivo consumo de memória pelo A-Estrela tende a ser um problema maior do que a sua complexidade do tempo, pois como ocorre em muitos algoritmos de busca, conforme a busca se aprofunda no espaço de busca, o número de nós na memória cresce rapidamente. E como os grafos que representam espaços de busca de problemas complexos são bastante grandes para serem armazenados na memória, a expansão de muitos nós pode esgotar o espaço na memória e gerar a falha do algoritmo.

- **Descrição do Algoritmo**

Esse método trabalha processando uma lista open, a qual inicialmente contém apenas a célula fonte. Mas essa é uma lista ordenada, e o critério usado para organizar as células é que elas sejam inseridas de acordo com uma distância estimada ao alvo, e não jogadas no fim da fila. Células que estejam em uma menor distância vão para o começo da lista. O algoritmo A* remove a primeira célula e verifica se ela é o alvo. Se não for, as células vizinhas são colocadas na lista nas posições adequadas.

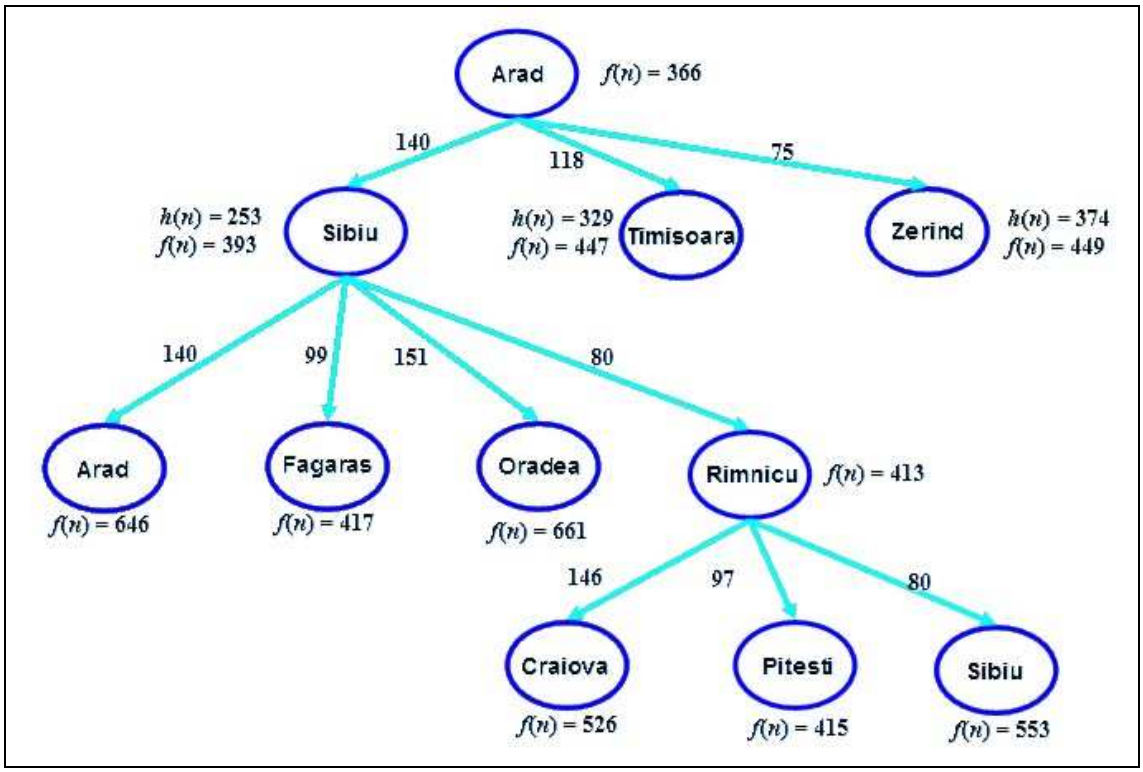


Figura 15 -A* - Busca de caminho ótimo para Bucareste através de Rimnicu e Pitesti.

O algoritmo verifica células vizinhas que já participaram da busca para ver se o caminho entre elas e a fonte é menor do que o encontrado anteriormente. Se isso ocorre, ela é reposicionada de acordo com a novaestimativa de distância. Como na busca em nível, isso continua até que o alvo seja encontrado ou a lista open esteja vazia.

- **Pseudocódigo**

Adicionar o estado inicial à lista aberta.
Enquanto lista aberta tiver elementos e não for adicionado estado final lista fechada
    Procurar elemento com menor f na lista aberta
    Mover para a lista fechada
    Procurar sucessores
    Para cada sucessor
        Se não é transponível ou está na lista fechada ignorar
        Senão
        Se não está na lista aberta adicionar, definir o pai como o nó actual e calcular F, G e H
        Se está, calcular novamente G (custo pelo caminho actual) e sefor menor substituir pai e substituir o G e o total (F).

4. O Planejador de Caminhos e Trajetórias

4.1. Algoritmo Proposto

A robótica orienta a automação de sistemas mecânicos baseados em sensores que possuem alguma capacidade computacional. Porém, para que essa orientação seja concisa é fundamental a concepção de algoritmos capazes de converter especificações das atividades humanas de alto nível como, por exemplo, mover-se, em tarefas de baixo nível exeqüíveis por um robô. Assim, planejar a trajetória de um robô significa especificar um algoritmo capaz de determinar a movimentação deste autômato respeitando suas limitações mecânicas.

Neste capítulo, é apresentada a definição do algoritmo proposto para este trabalho. Para tal optou-se por utilizar o método de decomposição celular aproximada, visto que este, durante as pesquisas, foi considerado o mais adequado ao tratamento do problema de busca de caminhos para aplicação robótica. Já que o principal objetivo do sistema que seria implementado era planejar a trajetória de um robô móvel em um ambiente de navegação com obstáculos. Este sistema também deveria ser capaz de converter uma trajetória previamente planejada em uma seqüência de trincas numéricas condizentes com as especificações do robô apresentado por (OLIVEIRA JR, 2008), bem como interagir remotamente, via radiofreqüência, com este autômato.

A proposta do algoritmo abrange a utilização do algoritmo A-Estrela, sendo que neste trabalho o A-Estrela empregou o conceito de expansão da fronteira de vizinhança de dois saltos. Este conceito será definido mais adiante.

O A-Estrela foi escolhido como base para o algoritmo proposto devido a suas características de eficiência, garantia de encontrar uma solução sempre que possível e facilidade de implementação.

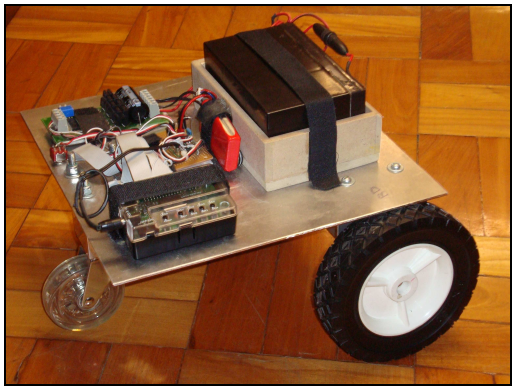


Figura 16 – Robô utilizado nos testes.

4.2. O Conceito de Vizinhança em Busca Heurística.

A aplicação do conceito de vizinhança é uma prática que tenta reduzir o espaço de busca, diminuindo as possíveis trocas para um determinado número de estados, impedindo dessa forma a análise de estados muitos distantes do estado alvo. Neste tipo de abordagem, o ponto mais crítico é a seleção da estrutura da vizinhança, ou seja, a forma na qual ela será definida. Por princípio tem-se que quanto maior a vizinhança, melhor a qualidade das soluções localmente ótimas e maior a precisão da solução final obtida. Por outro lado, quanto maior for a vizinhança, maior será a duração da busca na vizinhança em cada iteração. Por esse motivo, uma vizinhança maior não produz necessariamente uma heurística mais eficaz, a não ser que seja possível efetuar a busca de uma forma muito eficiente nessa vizinhança. Enfim, a geometria de um algoritmo de busca define a vizinhança de um estado, isto é, os estados que podem ser alcançados em um único passo a partir do estado inicial.

- **Vizinhança N4(p)**

Um estado p, de coordenadas (x, y) em um espaço de busca representado em forma de grade, possui um conjunto de 4 vizinhos horizontais e verticais nas seguintes coordenadas: : (x+1, y), (x−1, y), (x, y+1), (x, y−1). O conjunto destes estados vizinhos denota-se por N4(p). Neste conjunto, todos os estados vizinhos de p encontram-se a uma unidade de distância de (x, y).

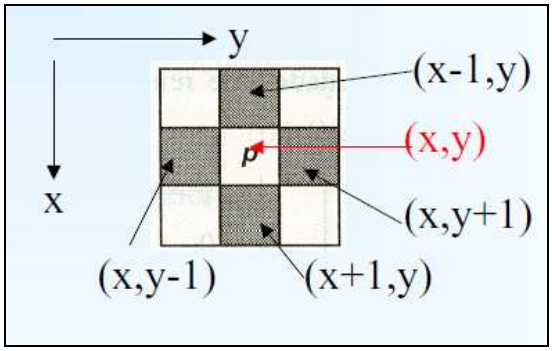


Figura 17 -Vizinhança N4(p)

- **Vizinhança N8(p)**

A vizinhança N8 de um estado p, de coordenadas(x,y) em um espaço de busca representado em forma de grade, é definida como: $N_{8(P)} = N_{4(P)} \cup N_{D(P)}$

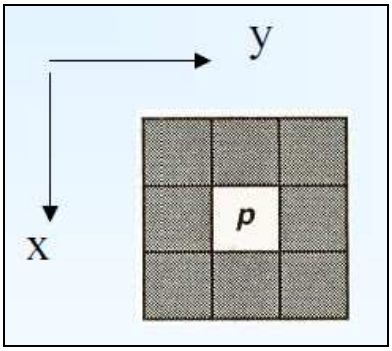


Figura 18 - Vizinhança N8(p)

- **Vizinhança ND(p)**

Em um estado p, de coordenadas (x, y) em um espaço de busca representado em forma de grade, os 4 estados vizinhos diagonais de p encontram-se nas coordenadas: (x+1,y+1), (x+1,y−1), (x−1,y+1), (x−1,y−1), sendo que o conjunto destes estados vizinhos denota-se ND(p).

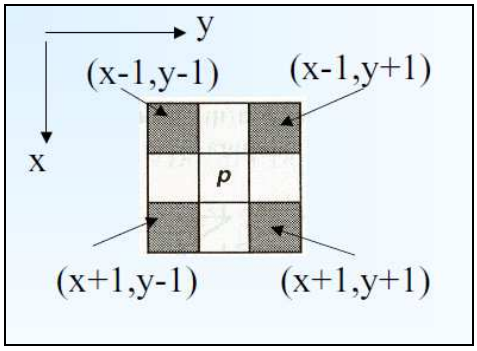


Figura 19 -Vizinhança ND(p)

4.3. A Expansão da Fronteira de Vizinhança do Algoritmo

O algoritmo proposto procura melhorar sua eficiência utilizando a regra de vizinhança N8 para explorar o espaço de busca. Sendo que nesta abordagem a fronteira da vizinhança N8 é definida de acordo com o conceito de saltos simples e duplos. Um salto simples ocorre quando a vizinhança de um estado **p** com

coordenadas (x, y) em um espaço de busca representado em forma de grade, é definida pelos estados de coordenadas (x-1, y-1), (x-1, y), (x-1, y+1), (x, y-1), (x, y+1), (x+1, y-1), (x+1, y), (x+1, y+1). Já um salto duplo ocorre quando a vizinhança de um estado p com coordenadas (x, y) é definida pelos estados de coordenadas (x-2, y-2), (x-2, y), (x-2, y+2), (x, y-2), (x, y+2), (x+2, y-2), (x+2, y), (x+2, y+2).

Assim, durante o processamento o algoritmo escolhe o tipo de salto mais adequado à configuração do espaço de busca.

Dessa forma, a cada iteração do laço principal o algoritmo examina a possibilidade de alcançar o vizinho mais próximo realizando dois saltos. Caso isso não seja possível, o algoritmo procura o vizinho mais próximo realizando apenas um salto. Essa estratégia busca avançar mais rapidamente em direção a meta. Além disso, ao realizar dois saltos para encontrar os próximos estados vizinhos, o algoritmo evita expandir estados desnecessários. E isso caracteriza economia de tempo de processamento e menor consumo de memória.

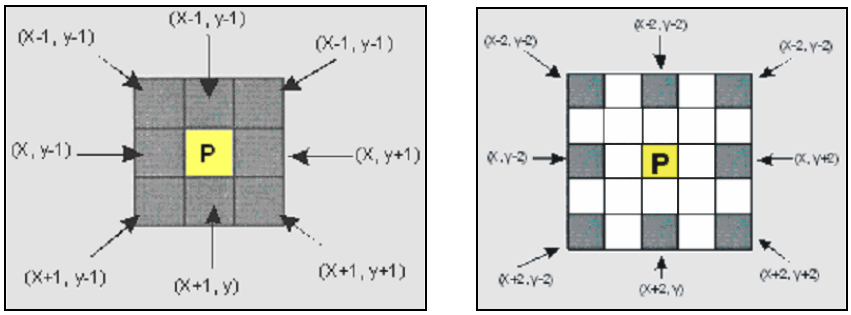


Figura 20 - Vizinhança N8 - Salto simples e salto duplo; respectivamente

4.4. Espaço de Configuração

Para facilitar o planejamento do movimento, a configuração espacial pode ser usada como uma ferramenta de apóio aos algoritmos de planejamento de caminhos. Nesse sentido, um espaço de configuração C, é o espaço de todas as configurações possíveis do robô. Isto é, o espaço de configuração é uma transformação do espaço físico onde o robô tem tamanho bem definido, para um espaço no qual o robô é tratado como um ponto.

- **Espaço Livre**

O espaço livre F $\subseteq$ C, é a porção do espaço de configuração que é livre de colisão. Ou seja, são as áreas que não são ocupadas por obstáculos.

- **Caminho Livre**

O objetivo do planejamento de movimento, é encontrar um caminho em **F** que ligue a posição inicial à posição final. Sendo assim, caminho livre é o percurso pertencente inteiramente ao espaço livre e que não entra em contato com nenhum obstáculo.

4.5. Configuração Espacial do Algoritmo Proposto

Neste trabalho, para realizar o planejamento de caminhos foi utilizado o método de decomposição em células, onde se utilizou um mapa de bits para armazenar um ambiente retangular de tamanho ajustável, decomposto em células quadradas para representar os espaços livres e os obstáculos. Assim, foi possível tratar o problema como o de um robô pontual movimentando-se em um ambiente bidimensional. E ainda, o robô pode ser rotacionado sem que haja movimento de translação, o que permite que seja desconsiderada a sua orientação no espaço de configurações, obtendo assim um problema com duas dimensões.

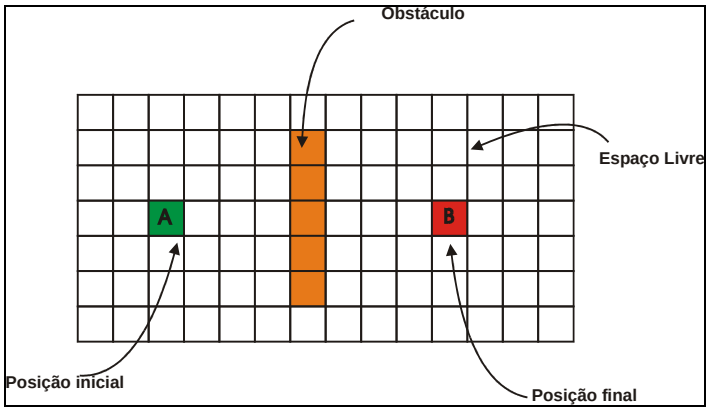


Figura 21 - Espaço de configuração

4.6. Algoritmo Proposto – A Busca do Caminho

Como dito anteriormente, para efetuar a busca de uma trajetória no espaço de configurações, foi utilizado um algoritmo de busca A-Estrela otimizado para trabalhar com um conceito de expansão da fronteira de vizinhança de até dois saltos. Porém, antes de efetivamente descrever o procedimento de busca do algoritmo, é necessário apresentar mais algumas definições:

— **Mapa**: é o espaço de configuração. Nele está contida toda a configuração do ambiente de busca (localização dos estado inicial e final, obstáculos e estados passáveis);

— **Lista Aberta**: Contém informações de todos os estados que estão em avaliação;

— **Lista Fechada**: Contém informações de todos os estados totalmente analisados;

— **Heap Binário (Fila de Prioridades)**: É uma árvore binária mantida na forma de vetor, nesta abordagem, ele é utilizado para procurar o estado com o menor valor heurístico. Sendo assim, o menor valor de F estará sempre na raiz.

- **Descrição do Algoritmo**

Já que a área de busca foi simplificada de acordo com o espaço de configuração da figura 21, deve-se agora ministrar o processo para encontrar o menor caminho entre “A” e “B”. Logo, partindo de “A”, analisando–se os estados vizinhos de acordo com o conceito da expansão da fronteira de vizinhança, definida no item 4.3, e geralmente, procurando para fora até que “B” seja alcançado, a busca ocorrerá da seguinte forma:

— Começando pelo estado de partida “A” e acrescentando-o a lista aberta;

— Em seguida, a partir de “A”, examinam-se todos os estados alcançáveis realizando apenas um salto, se nenhum destes estados for um obstáculo, realiza-se mais um salto e examinam-se todos os estados alcançáveis nesse segundo salto, se houver algum estado (nó) obstáculo, retrocede-se para a vizinhança do primeiro salto. Senão a busca continua na vizinhança do segundo salto.

— Após definir a próxima vizinhança, isto é, os estados mais próximos. Seja no primeiro ou no segundo salto, acrescentam-se esses estados passáveis à lista aberta. Para cada um deles, o estado “A” é salvo como seu estado pai, ou seja, a informação de que o estado “A” gerou cada um desses estados será armazenada.

— Remove-se o estado “A” da lista aberta acrescentando-o na lista fechada.

Em relação ao posicionamento dos obstáculos, ocasionalmente, o espaço de configuração pode se encontrar em uma das seguintes situações:

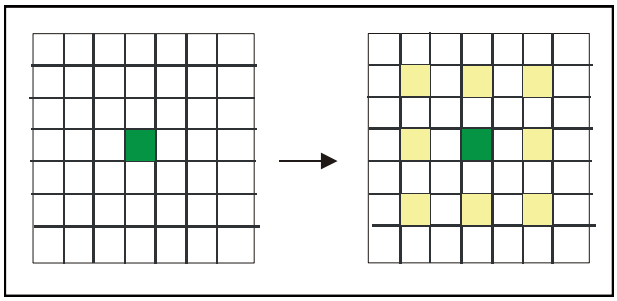


Figura 22 - Caso 1 - Salto Duplo Sem Interceptação de Obstáculos

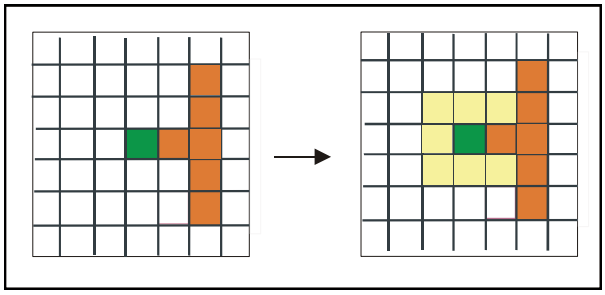


Figura 23 -Caso 2 - Salto Duplo com Interceptação de Obstáculos.

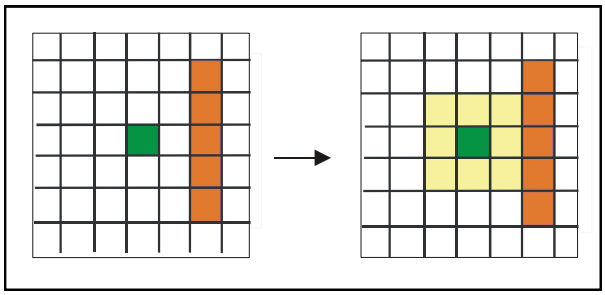


Figura 24 - Caso 3 - Salto Duplo com Interceptação de Obstáculos.

O espaço de configuração da figura 25 é utilizado para ilustrar a busca que o algoritmo realiza a de acordo com os casos das figuras anteriores.

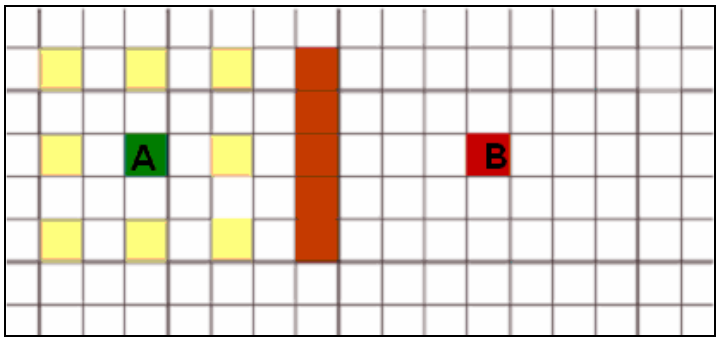


Figura 25 - Algoritmo de busca realizando salto duplo

Nas ilustrações, o estado (nó) verde é o estado (nó) pai. Todos os nós vizinhos estão agora na lista aberta e cada um dos estados passou a ter um ponteiro apontando para trás.

Graduação do caminho

A função heurística utilizada para determinar a direção a ser escolhida durante a busca do caminho é: $F = g + h$; onde:

— g: é o custo do movimento para se mover do estado inicial até um determinado estado no espaço, seguindo o caminho criado para chegar até lá.

— h: é o custo estimado do movimento para mover-se de um determinado estado até o estado final, “B”.

O caminho é gerado passando-se repetidamente pela lista aberta e escolhendo o estado com o menor valor heurístico (valor de F).

Como descrito anteriormente, g é o custo do movimento para mover-se do ponto de partida para um determinado estado usando o caminho gerado para chegar lá. Nesta abordagem, determina-se que o valor de g é igual a 10 se o movimento do estado ocorrer no sentido horizontal ou vertical, e igual a 14 para um movimento no sentido diagonal. Estes números foram utilizados porque a distância real para mover diagonalmente é $\sqrt{2}$ vezes o custo de mover horizontalmente ou verticalmente. Como consequência utilizou-se 10 e 14 , para simplificar os cálculos. Assim, o custo g de um dado estado é o custo g do seu pai somado a 10 ou 14 dependendo do sentido do movimento diagonal ou ortogonal.

Existem vários métodos para calcular h. Neste trabalho a implementação apresenta a opção de escolha entre os seguintes métodos:

- Manhattan (City Block): $h = 10*(|X_{atual} - X_{objetivo}| + |Y_{atual} - Y_{objetivo}|)$
- Distância diagonal: $DistX = |X_{atual} - X_{objetivo}|$ e $DistY = |Y_{atual} - Y_{objetivo}|$

  $DistX > DistY$ então $h = 14 * DistY + 10 * (DistX - DistY)$

  $DistY > DistX$ então $h = 14 * DistY + 10 * (DistY - DistX)$
- Distância Máxima: $DistX = |Xatual - Xobjetivo|$ e $DistY = |Yatual - Yobjetivo|$

  $DistX > DistY$ então $h = 10 * DistX$

  $DistY > DistX$ então $h = 10 * DistY$

A função heurística F é obtida pelo somatório de g e h. Os resultados do primeiro passo da busca podem ser vistos na ilustração abaixo, nela são escritos os valores para F, g, e h em cada quadrado. F é impresso na parte superior esquerda, g é impresso na parte inferior esquerda, e h é impresso na parte inferior direita.

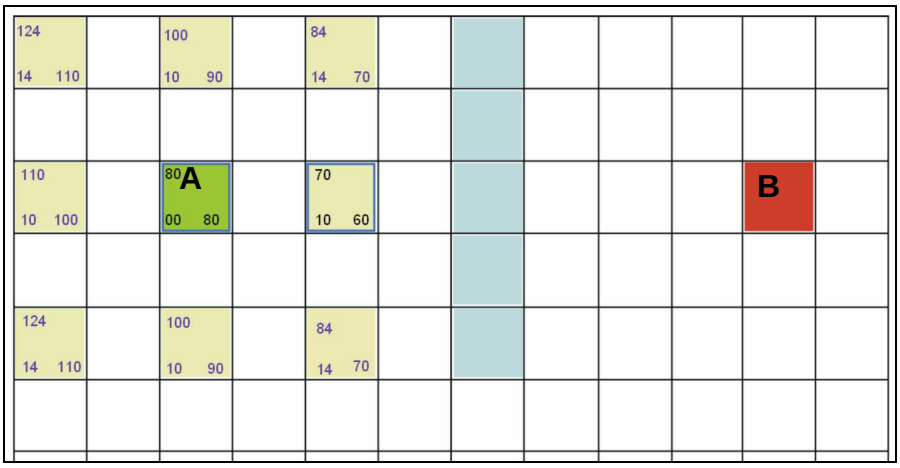


Figura 26 - Salto Duplo – valores da função heurística

Examinando a ilustração percebe-se que $g = 10$ quando o movimento ocorre na direção horizontal ou vertical, mas quando a direção é diagonal $g = 14$.

Os valores de h são calculados estimando a métrica da distância Manhattan até a meta (estado “B”), apenas com movimentos horizontais e verticais e ignorando o obstáculo que está no caminho. Note também que, o somatório de g e h é o valor de F para cada estado.

Avançando a Busca do Caminho

Para continuar o seguinte procedimento é utilizado o menor valor de F na lista aberta.

— Retire-o da lista aberta e acrescente-o à lista fechada. Defina a nova vizinhança;

— Calcule os valores de g, h e F para essa nova vizinhança. Ignorando os estados que já estiverem na lista fechada ou que sejam obstáculos. Em seguida, acrescente-os à lista aberta, se eles já não estiverem lá. Finalmente, faça o estado selecionado o pai dessa nova vizinhança.

— Se algum desses novos estados já estiver na lista aberta, verifique se o g para este estado é menor do que o calculado anteriormente. Se não for, não faça nada. Caso esse novo g seja menor, troque o pai desse estado para o estado

selecionado. Então, recalcule o *F* e *g* para aquele estado.

Assim, após a transferência do estado inicial da lista aberta para a lista fechada, ainda restarão estados na lista aberta. Desses estados, seleciona-se o estado com o menor valor de *F* que será o próximo estado. Na Figura 27 a seleção está destacada em azul, nesse caso o menor valor de *F* é 70.

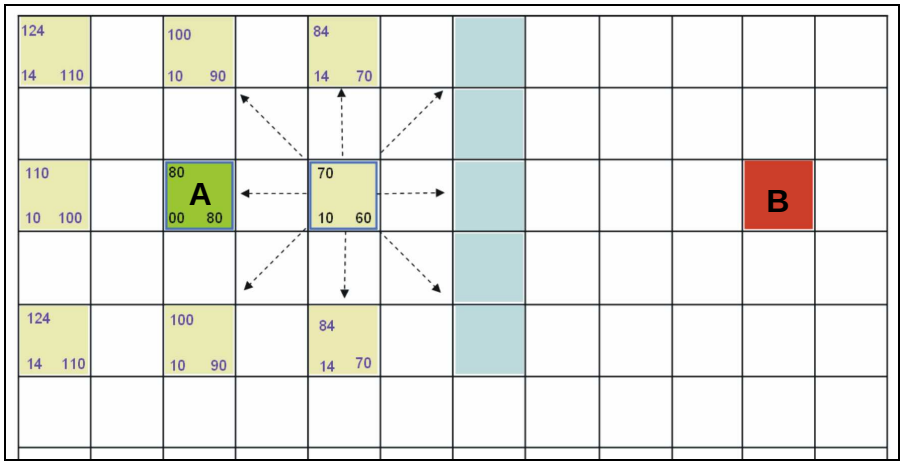


Figura 27 - Menor valor da função F após salto duplo

Os processos de inclusão e remoção na lista aberta e fechada repetem-se sucessivamente até que o nó objetivo seja acrescentado à lista fechada, conforme definição anterior, o mesmo ocorre com o posicionamento dos obstáculos no espaço de busca. A busca continua conforme as figuras a seguir.

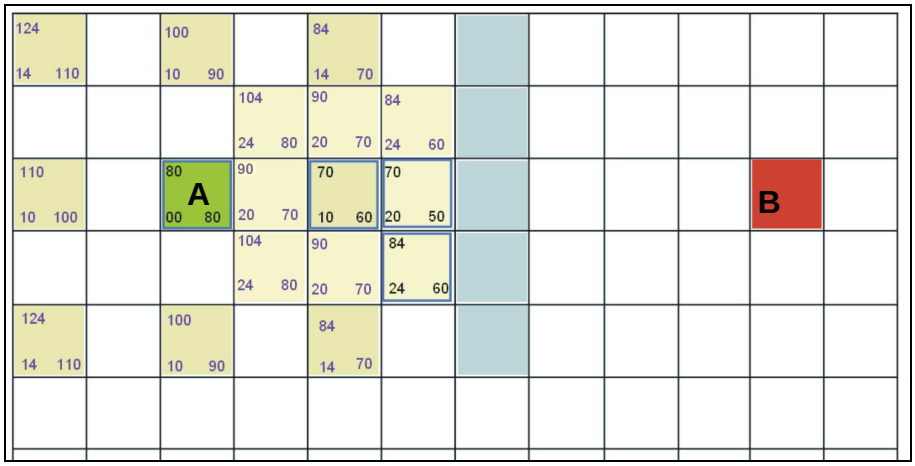


Figura 28 - Salto simples – valores da função heurística

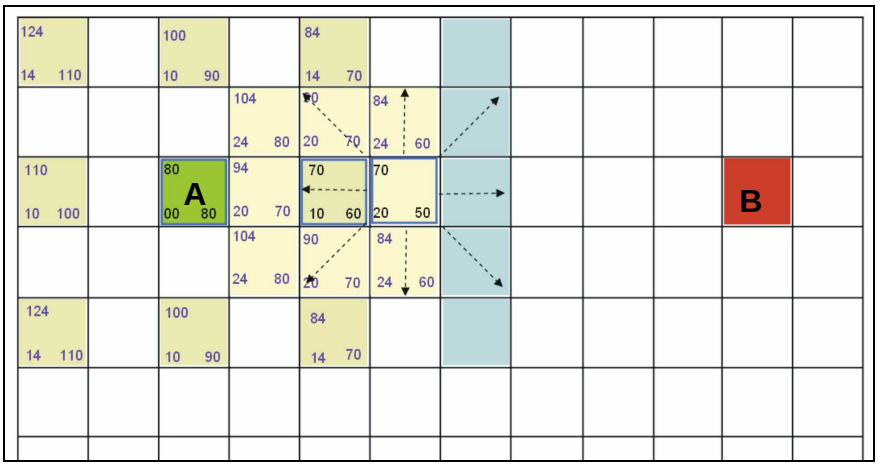


Figura 29 - Seleção do novo pai e valores heurísticos da nova vizinhança

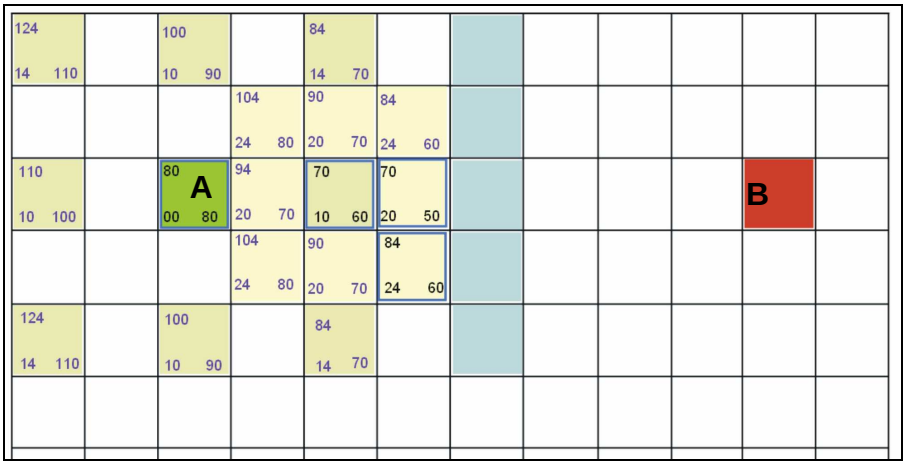


Figura 30 - Outro salto simples com valores heurísticos da nova vizinhança

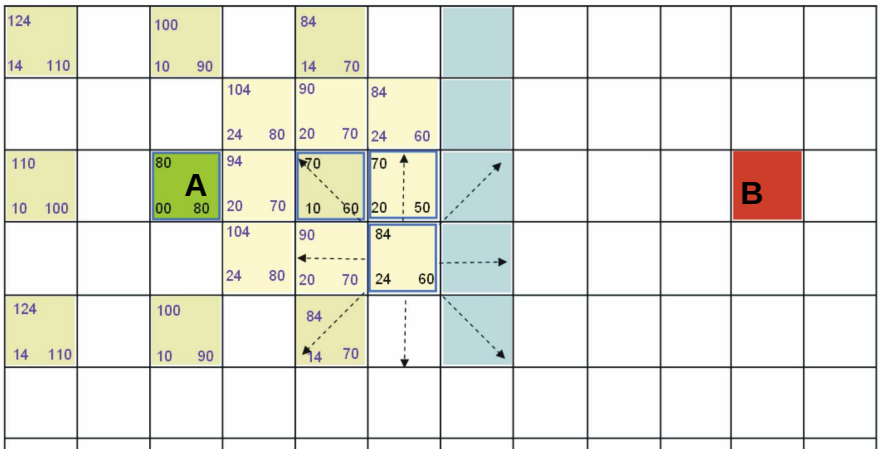


Figura 31 - Seleção do novo pai e valores heurísticos da nova vizinhança

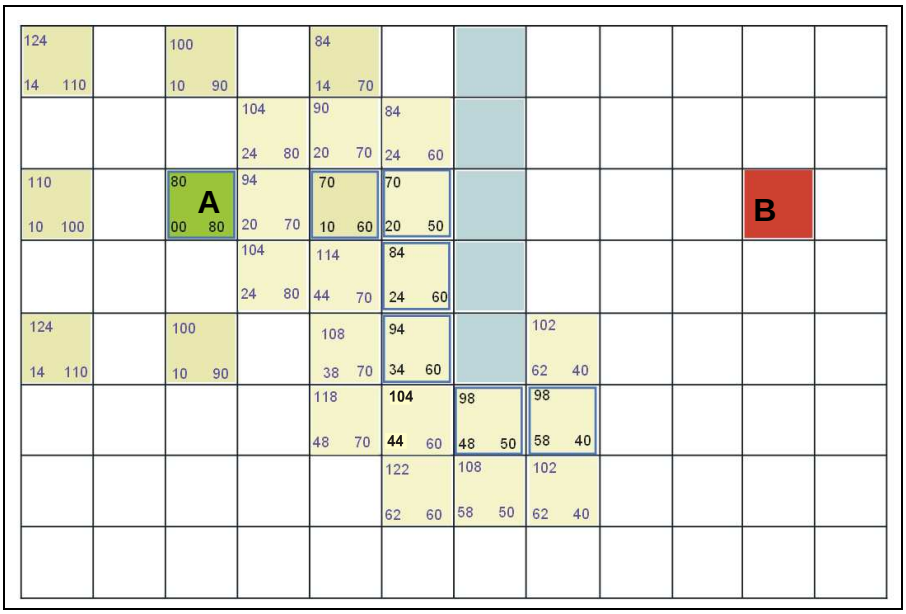


Figura 32 - Seleção do novo pai e valores heurísticos da nova vizinhança

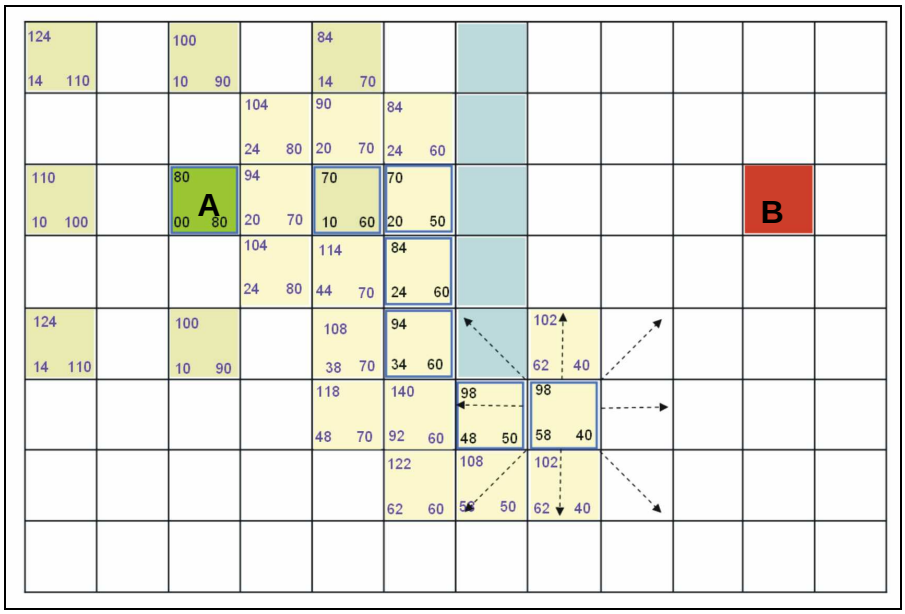


Figura 33 - Seleção do novo pai e valores heurísticos da nova vizinhança

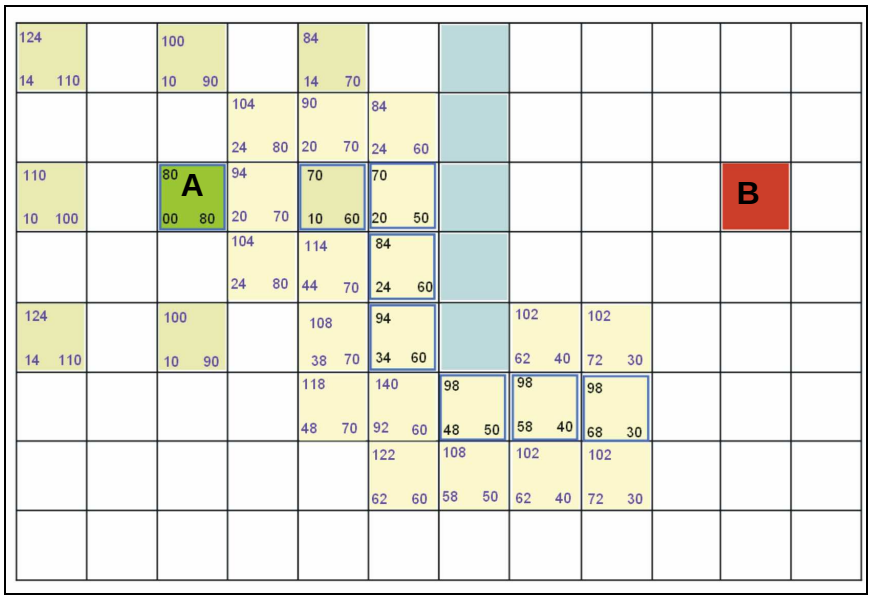


Figura 34 - Seleção do novo pai e valores heurísticos da nova vizinhança

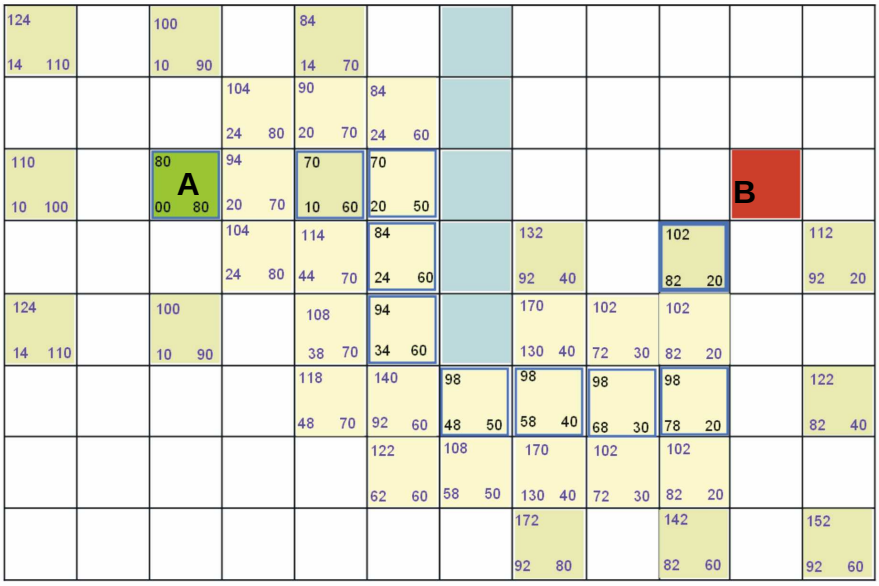


Figura 35 - Seleção do novo pai e valores heurísticos da nova vizinhança

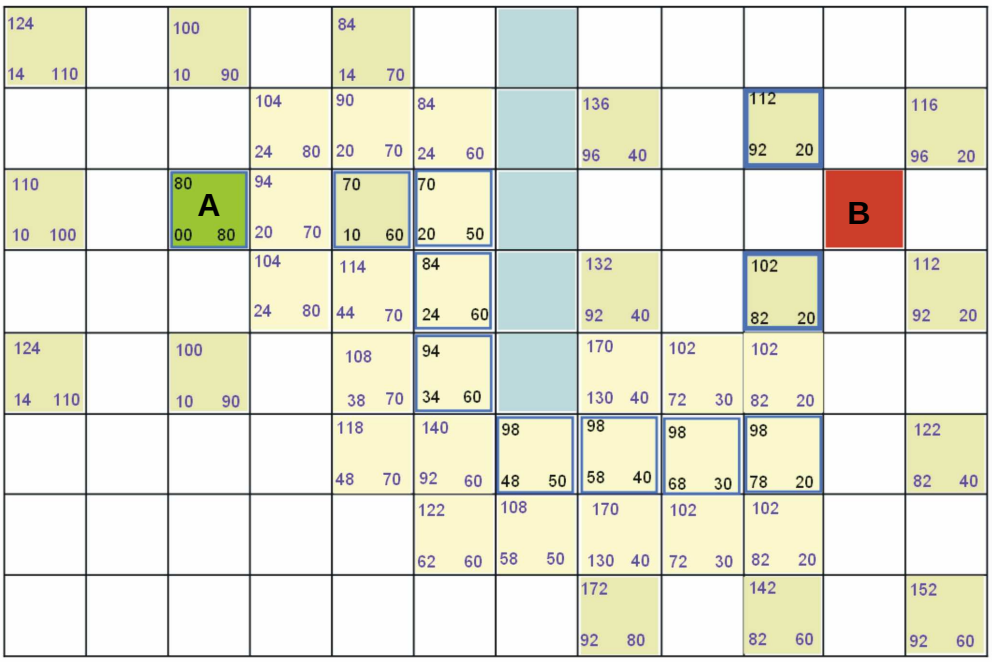


Figura 36 - Seleção do novo pai e valores heurísticos da nova vizinhança

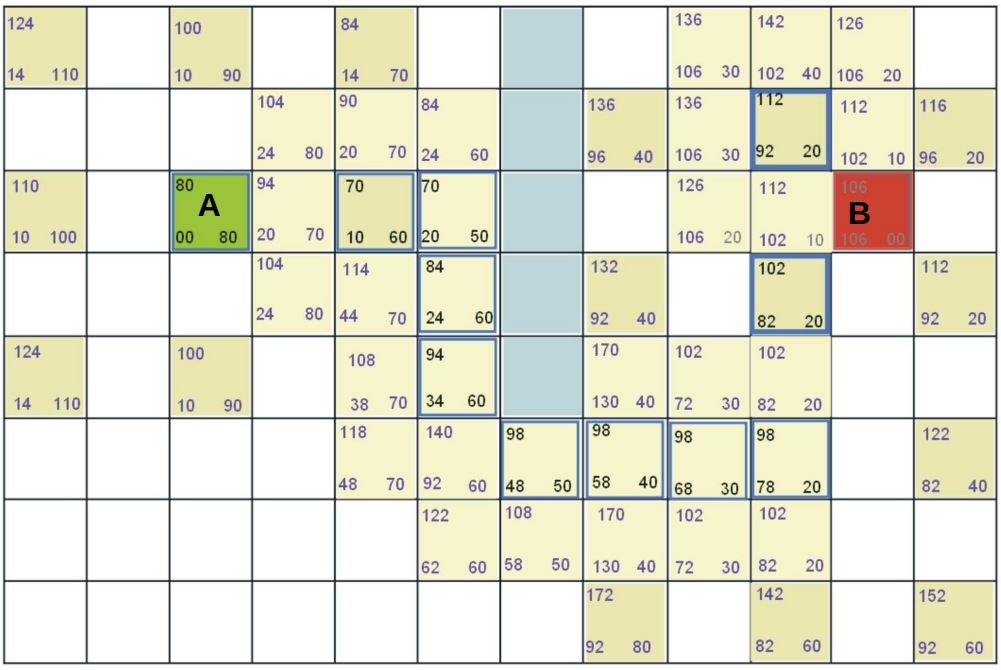


Figura 37 - Seleção do novo pai e valores heurísticos da nova vizinhança

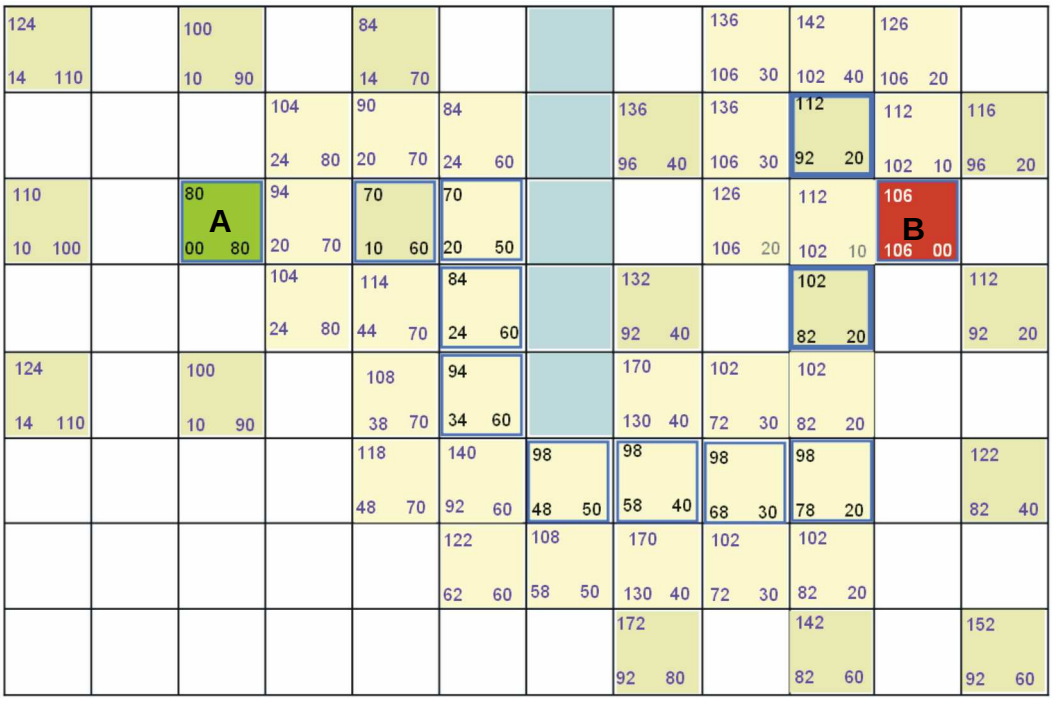


Figura 38 - Seleção do novo pai e valores heurísticos da nova vizinhança

Para determinar o caminho final, é necessário caminhar para trás do nó objetivo, indo de cada nó até o seu nó pai, até que se alcance o nó inicial.

4.7. Algoritmo de Simplificação de Caminhos e Geração de Trajetórias

4.7.1. Algoritmo de Simplificação de Caminhos

Independente da forma de geração do caminho, a solução encontrada por qualquer algoritmo de busca quando aplicado a um espaço de busca em formato de grade, é uma linha que geralmente tem uma forma sinuosa. Da mesma forma, isso também ocorre quando é necessário encontrar o próximo estado (nó), para efetivamente obter o caminho capaz de conduzir um objeto qualquer, até o estado meta. Freqüentemente, o objeto em questão não está sobre um estado (nó) e por isso precisa percorrer um caminho que não se parece natural. Logo, a solução mais adequada para esse problema é realizar um pós-processamento no caminho encontrado. Já que, geralmente sempre existe uma forma de simplificar o caminho encontrado e obter caminhos mais realistas.

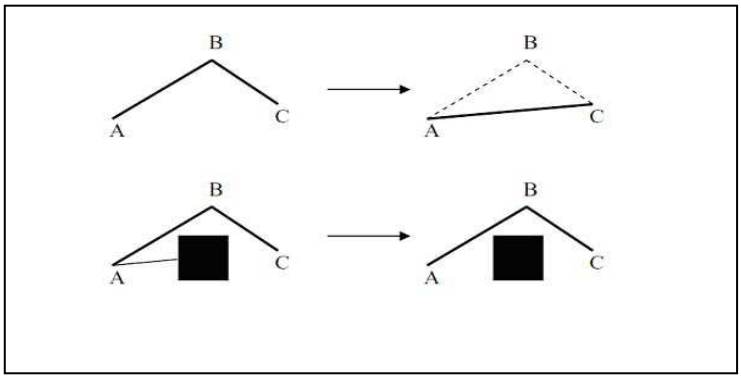


Figura 39 - Remoção de nós de um caminho

Uma estratégia simples para este problema é verificar a visibilidade entre estados vizinhos. Se um desses estados é supérfluo, então os dois são substituídos por apenas um, como mostrado na figura 40.

- **O algoritmo opera da seguinte forma:**

Inicialmente dois iteradores “A1” e “A2” são posicionados no primeiro e segundo nó respectivamente. Então, processam-se os seguintes passos:

1. Selecione a posição origem de “A1”;

2. Selecione a posição destino de “A2”;

3. Se o agente consegue se mover entre esses dois pontos sem nenhuma interseção com elementos estáticos do cenário atribui-se a posição destino de “A1” para a de “A2” e remove-se o nó “A2” do caminho. Posiciona-se “A2” na aresta seguinte a “A1” já modificada;

4. Se o agente não puder se mover entre esses dois pontos atribui-se “A2” para “A1” e avança-se “A2” para o próximo nó;

5. Repetir os passos até que o destino de “A2” seja igual ao destino final do caminho. Na figura 45 são ilustrados os passos do algoritmo, cujo caminho parte do quadrado e chega ao círculo. Deve-se observar que este algoritmo é eficiente, porém não é ótimo. Como observado na figura 41, os dois últimos nós do caminho suavizado poderiam ser transformados em um único. O algoritmo não resolve isso, pois somente testa os nós vizinhos. Dado um nó “A1”, ele deveria ser testado com todos os demais cada vez que “A1” avançasse.

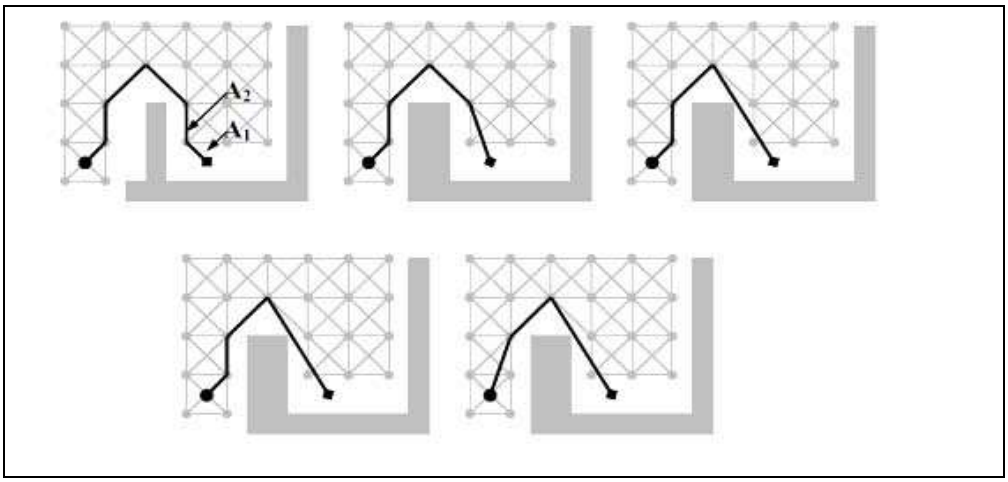


Figura 40 - Etapas do algoritmo de suavização de caminhos

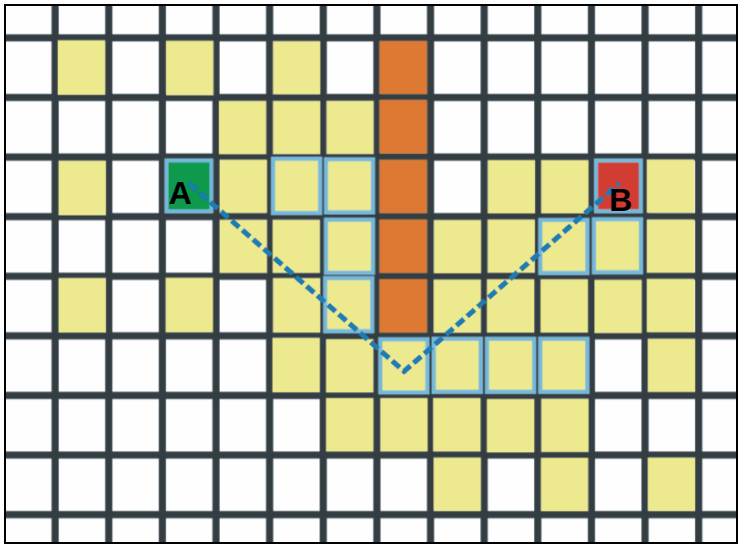


Figura 41 - Aplicação do algoritmo de simplificação de caminhos

Na figura 41, após o processamento do algoritmo de simplificação de caminhos, apenas três pontos constituem o caminho final. A utilização desse método é muito útil para as situações em que se deseja obter o menor caminho, mas a precisão do percurso não seja um fator primordial.

4.7.2. Algoritmo de Geração de Trajetórias

A odometria é (POLLI H. B. & OUTROS, 2006) dos métodos mais amplamente utilizados para estimar a posição de um robô. Sabe-se que a odometria proporciona uma boa precisão em curto prazo, é barata de implantar e permite taxas de amostragem muito altas. A idéia fundamental da odometria é a integração de informação incremental do movimento ao longo do tempo, o qual envolve uma inevitável acumulação de erros. A acumulação de erros de orientação causa grandes erros na estimação da posição, os quais vão aumentando proporcionalmente com à distância percorrida pelo robô. Apesar destas limitações, a odometria é uma parte importante do sistema de navegação de um robô juntamente com medidas do posicionamento absolutas para proporcionar uma estimativa de posição mais confiável. Assim, no contexto desse trabalho para efeito dos cálculos de odometria, assume-se que o robô seja um corpo rígido com rodas que não sofrem deformação ou derrapagens e que se movimentam apenas no plano horizontal. Dessa forma, para esta abordagem considerou-se um espaço de configuração com dimensão fixa de 720x720 pixels onde para efeito de cálculos estabeleceu-se que 1 pixel equivale a 2,6 cm, ou seja, aproximadamente uma área de $2m^2$. Além disso, definiu-se a representação desta área em células quadradas de dimensão fixa de 1x1cm. Sendo assim, após a modelagem do espaço de configuração, do cálculo e simplificação das coordenadas que formam o caminho, a trajetória resultante é calculada e codificada em uma seqüência de trincas de números a qual é enviada via sinal de radiofreqüência para o robô. Essa seqüência de trincas de números é definida formalmente como protocolo e é baseada nas características do robô apresentado por (OLIVEIRA JR, 2008) cuja faixa de velocidade de trabalho é de 34 a 280 mm/s e nos pares de coordenadas que fazem parte do percurso encontrado no planejamento do caminho. A sintaxe do protocolo de comunicação apresenta o seguinte formato:

#tempo*vrd*vre$ (Primeira trinca numérica)

…

#tempo*vrd*vre$@ (Última trinca numérica)

Onde:

- # - caractere que indica o início da trinca;
- **-** caractere que indica separação entre números;
- **$** - caractere que indica o fim da trinca;
- @ - caractere que faz o robô executar o trecho;
- **tempo** - Tempo de execução do trecho.

Cabe ressaltar que o percurso obtido pelos algoritmos de planejamento e simplificação de caminhos é constituído por trechos de retas e curvas alternados. Assim, sejam três pontos A, B e C (par de coordenadas que representam um percurso encontrado). Seja o triângulo ABC, cujos lados medem respectivamente a, b, c. Logo, as trajetórias são obtidas conforme segue:

(1) As medidas de c, a e b , ou seja as distâncias dos entre trechos AB, BC e CA, são obtidas através da distância Euclidiana: $d_{AB} = \sqrt{(x_B - x_A)^2 - (y_B - y_A)^2}$, $d_{BC} = \sqrt{(x_C - x_B)^2 - (y_C - y_B)^2}$ e $d_{CA} = \sqrt{(x_A - x_C)^2 - (y_A - yc)^2}$

(2) Para obter o valor do trecho em curva, deve-se calcular o ângulo entre os trechos AB e BC, ou seja $\hat{B}$, utilizando a lei dos Cossenos:

$$b^2 = a^2 + c^2 - 2ac.\cos\hat{B}$$

$$Logo$$

$$\hat{B} = arc\cos(\hat{B}) = \frac{a^2 + c^2 - b^2}{2ac}$$

(3) O valor da distância do trecho em curva é obtido calculando-se o comprimento do arco de circunferência que é dado por: $l = \frac{\hat{B}\pi R}{180}$

(4) No trecho em curva, o sentido do movimento é dado pelo sinal do ângulo calculado anteriormente. Ou seja, se o valor do ângulo for negativo o sentido é para a direita, caso seja positivo o sentido é para esquerda;

(5) Então, com base na faixa de velocidade de trabalho do robô e nas distâncias percorridas calculadas nos passos anteriores, obtém-se o tempo de deslocamento para cada trecho seguido:

$$tempo\_de\_execução\_do\_percurso = \frac{distância\_percorrida}{velocidade}$$

(6) No protocolo definido anteriormente, uma trinca representa o deslocamento de um trecho específico. Sendo assim, a trajetória final será representada por um conjunto de trincas.

5. Sistema de navegação

Para testar e validar o algoritmo proposto foi desenvolvido o planejador, chamado PCT (Planejador de Caminhos e Trajetórias). Este planejador foi implementado no NetBeans 6.1, que é um ambiente de desenvolvimento integrado (IDE) Java, desenvolvido pela empresa Sun Microsystems (Netbeans,2008). A principal vantagem desta linguagem consiste em sua independência de plataforma, carregamento dinâmico de código pela rede e a facilidade de uso da rede. O PCT retrata um ambiente a ser explorado com uma grade quadrada de 720X720 células. Cada célula representa uma possível posição dentro do espaço de configuração, o planejador foi dimensionado para interagir com o robô ao final da pesquisa do melhor caminho, enviando a este os comandos necessários para a execução do percurso planejado. Dentre várias funcionalidades, o planejador possui ferramentas que permitem desenhar obstáculos de diversas formas. Por isso, o PCT é ferramenta bastante flexível, pois permite configurar o espaço de busca de acordo com as necessidades da simulação.

5.1.1. A Interface

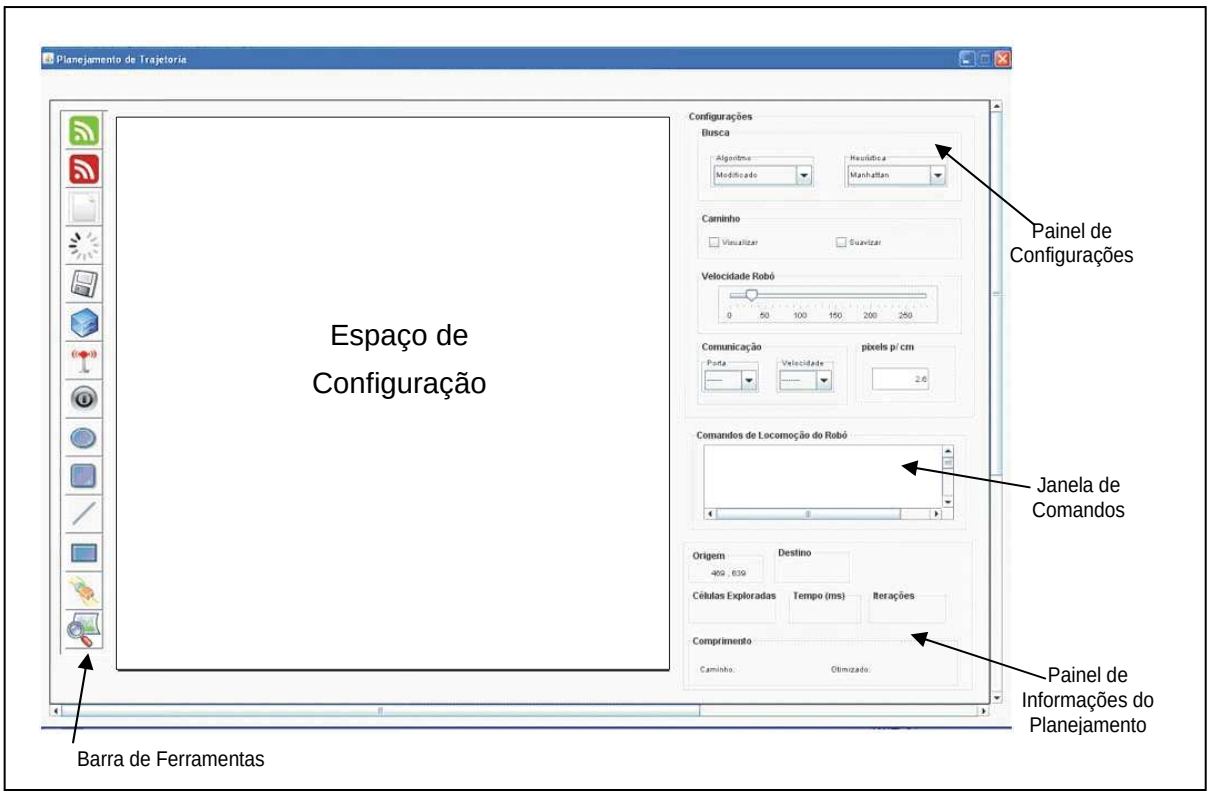


Figura 42 - Interface do Planejador

5.1.2. A Barra de Ferramentas

A barra de ferramentas do planejador possibilita o acesso a todas as ferramentas disponíveis para o controle do planejamento da trajetória. A partir da barra de ferramentas é possível: habilitar o posicionamento do ponto inicial e final no mapa, criar um novo planejamento, executar o planejamento, salvar o resultado do planejamento, abrir o resultado de um planejamento, enviar os comandos de locomoção para o robô, fechar o PCT, desenhar obstáculos em formato oval, quadrado arredondado, linha e quadrado ,apagar através da ferramenta borracha e abrir janela de zoom.

Figura 43 - Barra de Ferramentas do Planejador

5.1.3. O Painel de Configurações

O painel de configurações permite preparar o planejador com as opções mais adequadas a situação desejada. A partir dele é possível configurar: parâmetros referentes à busca da trajetória, habilitar a visualização e suavização do caminho, determinar a velocidade de locomoção do robô, definir parâmetros da comunicação: porta e velocidade e Especificar o valor de conversão de pixels para centímetros.

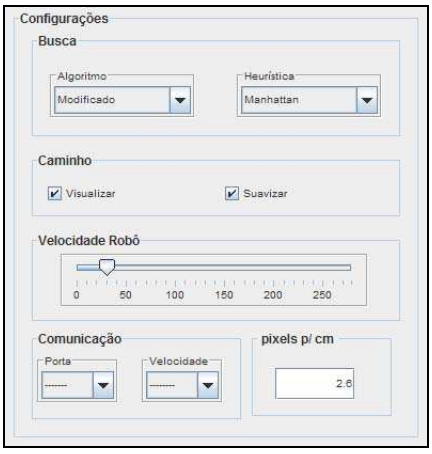


Figura 44 - Painel de Configurações do Planejador

5.1.4. A Janela de Comandos

Permite que o usuário visualize de forma textual os comandos de locomoção que serão enviados a o robô.

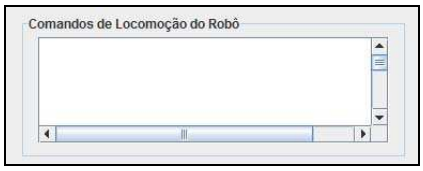


Figura 45 - Janela de Comandos do Planejador

5.1.5. O Painel de Informações do Planejamento

Este painel permite visualizar as seguintes informações durante a execução do planejamento: posição de origem e posição de destino do robô, quantidade total de células exploradas pelo algoritmo, tempo de execução em milisegundos, número de iterações do laço principal do algoritmo e comprimento do caminho encontrado pelo algoritmo de planejamento e do algoritmo de simplificação.

Figura 46 - Painel de Informações do Planejamento

5.1.6. O Espaço de Configuração

É a área gráfica disponibilizada para o usuário desenhar o cenário de busca e onde o planejador apresentará com uma linha azul o caminho encontrado e com uma linha vermelha a trajetória resultante do processo de simplificação.

6. Implementação do Sistema de Navegação

O NetBeans permitiu a implementação e depuração do software do planejador de uma forma visual e as API AWT, SWING e suas funcionalidades facilitarão a implementação da interface. Além disso, as facilidades fornecidas pela API de comunicação serial RXTXcomm foram essenciais ao desenvolvimento do módulo de comunicação da aplicação. O código fonte do planejador encontra-se no apêndice.

A seguir é apresentado o diagrama de blocos das etapas de processamento que o sistema de navegação implementa.

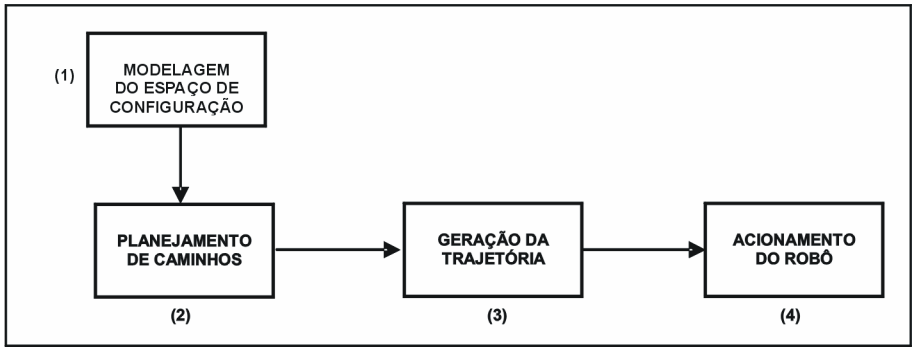


Figura 47 - Diagrama das etapas do Sistema de Navegação.

(1) Após a inicialização do planejador, com o auxílio da ferramenta de desenho da barra de ferramentas, deve-se modelar o espaço de trabalho. Para isso, desenham-se os obstáculos e determina-se a posição de início e término do movimento do robô. Então, inicia-se o processamento do planejamento.

(2) Inicialmente, ocorre o mapeamento do espaço de configuração, esse processo consiste na identificação do valor da cor de cada célula (pixel) e codificação para o sistema de processamento Então, o planejamento de caminhos propriamente dito é iniciado. É nesse momento que planejador executa o algoritmo de busca proposto neste trabalho.

(3) Neste passo é realizado o pós-processamento do caminho, ou seja, o algoritmo de simplificação de caminhos simplifica o caminho encontrado na etapa anterior. Em seguida a trajetória final a ser executada pelo robô é calculada e codificada de acordo com o protocolo definido para este trabalho.

(4) A trajetória final é enviada via sinal de radiofreqüência para o robô.

7. O Robô Utilizado nos Testes

Como a implementação dos testes do sistema de navegação foi baseada no robô de OLIVEIRA JR (2008), a seguir será apresentada as suas principais características .

- **Características técnicas do Robô**

Comunicação bidirecional
Sistema embarcado de sensoriamento
Máxima velocidade linear (Vmax): 0,311 m/s;
Máxima aceleração linear (Amax): 0,272 m/s2;
Máxima velocidade angular (ωmax): 0,311 rad/s;
Máxima aceleração angular (Aωmax): 5,51 rad/s2.
Massa (m): 3,8 Kg;
Momento de Inércia (J): 1,14 Kgxm2;
Distância entre rodas motora direita e esquerda (TR): 0,30 m;
Distância da roda dianteira ao centro de gravidade (LF): 0,21 m;
Distância do eixo traseiro ao centro de gravidade (LR): 0,045 m.

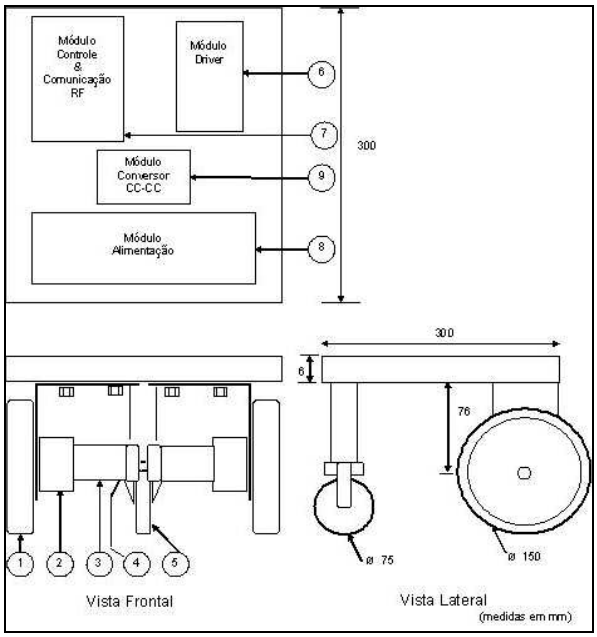


Figura 48 Desenho geométrico do robô, com a localização dos módulos. embarcados.

Fonte: OLIVEIRA JR, 2008

Da figura 48 têm-se:

1 - Roda motora;
2 - Caixa de redução;
3 - Motor DC;
4 - Encoder HEDS 5500 J06 500 PPR;
5 - Roda livre;
6 - Módulo Driver MD22;
7 - Módulo Controle & Comunicação RF SRB 13213 Freescale;
8 - Módulo Alimentação - Baterias de 12 Vdc e 7,4 Vdc;
9 -Módulo Conversor CC-CC.

8. Resultados

Com o objetivo de observar e testar a aplicação desenvolvida, bem como verificar a eficácia da trajetória planejada com o robô de (OLIVEIRA JR, 2008), foram modeladas três configurações de espaço de trabalho no planejador. Assim, inicialmente, o robô foi posicionado em um ponto correspondente a posição inicial do espaço modelado. Em seguida, o botão de inicialização do processo de busca foi acionado. Ao término do processo de busca de caminhos, o PCT, previamente habilitado para permitir a visualização e suavização do caminho, gerou automaticamente as trincas de números referentes aos movimentos que o robô deveria executar. Logo após, o botão de envio de comandos de locomoção para o robô foi acionado e o robô passou a receber a seqüência de trincas via sinal de radiofreqüência. Ao final da comunicação entre o sistema do planejador e o sistema embarcado do autômato, o robô foi então guiado através do ambiente até a sua posição de destino.

8.1. Primeiro Espaço de Configuração

Esta demonstração apresenta o planejamento de trajetória realizado com o algoritmo A* e com o A* Modificado. Ambos foram utilizados para calcular a trajetória do ponto de partida x=20 e y=81 até o ponto x=467 e y=379. Na tabela 1 constam alguns detalhes do processamento dos algoritmos.

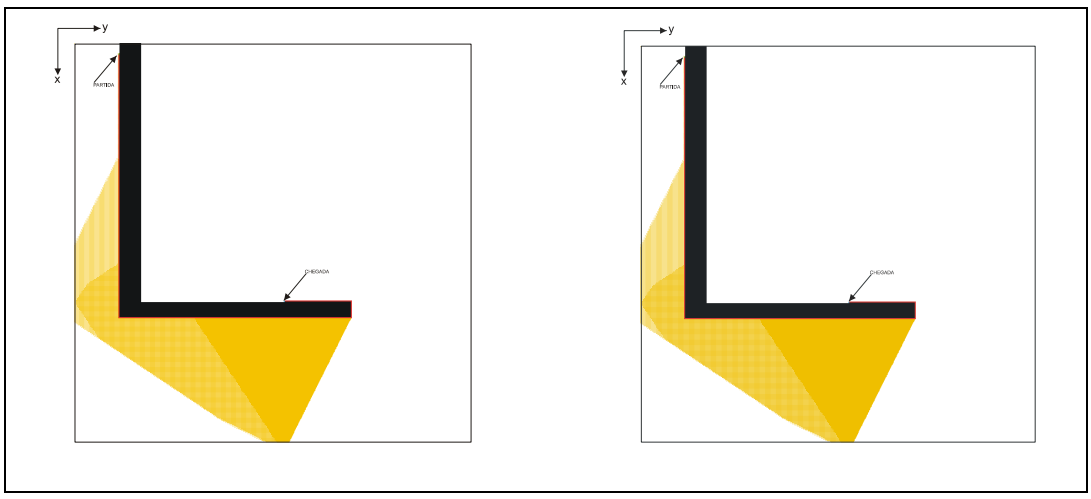


Figura 49 - Planejamento de trajetória no primeiro espaço de configuração – Algoritmo Modificado e A*

Tabela 1: Detalhes do processo de planejamento do primeiro espaço de configuração

Ponto de partida	(20, 81)	
Ponto de chegada	(467, 379)	
Heurística	Distância de Manhattan	
Método	Modificado	A-Estrela
Células Exploradas	68307	106826
Tempo (ms)	26735	42859
Iterações	66258	105894
Caminho	652 células	1037 células
Caminho simplificado	5 células	5 células

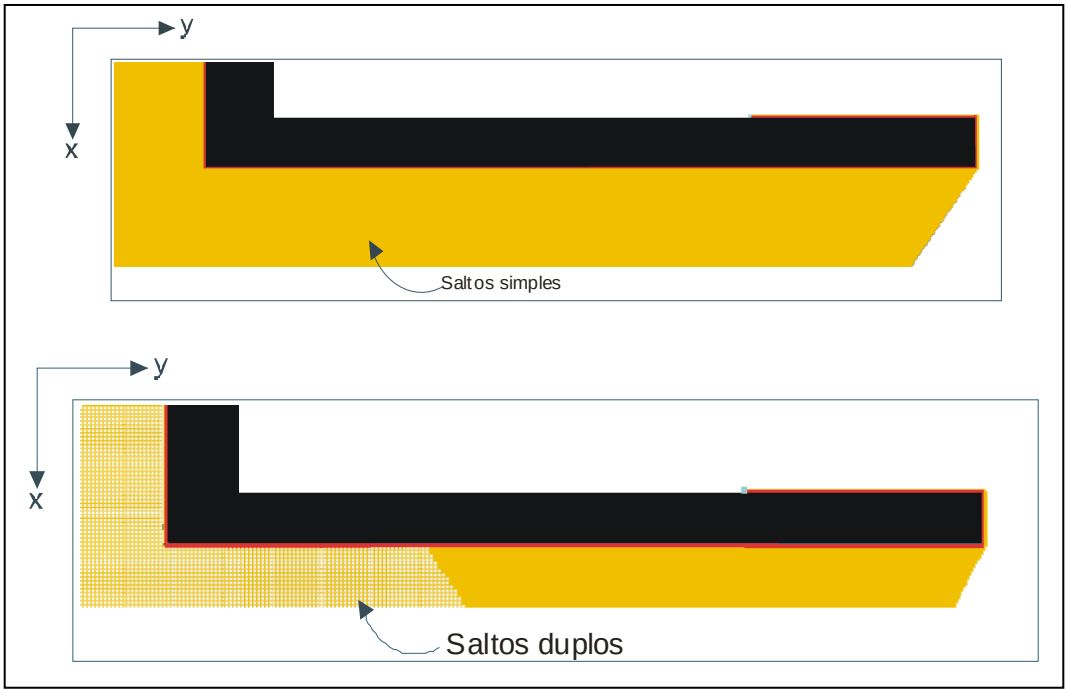


Figura 50 - Detalhamento dos saltos primeiro espaço de configuração

O planejamento é apresentado na figura 53, a linha em vermelho representa a trajetória planejada, a área em amarelo representa o espaço explorado no processo de busca, os obstáculos estão representados em branco e o espaço livre está representado em preto. Devido a precisão definida para este trabalho, os ponto de partida e chegada estão praticamente invisíveis na figura 53, mas os mesmos são representados nas cores verde e vermelho, respectivamente. A linha em vermelho é a trajetoria obtida pelo processamento do algoritmo de simplificação de caminhos, o resultado do algoritmo de busca deveria aparecer em azul, mas como o percurso obtido pela simplificação coincidiu com o percurso da busca, a representação em azul não está presente na figura, tanto no método modificado como no A-Estrela. Salienta-se

que o movimento previsto deu-se sem colisão e com um minimo desvio da trajetória plenejada.

Os detalhes do processamento dos algoritmos presentes na tabela 1 conclusivamente demonstram que a modificação implementada no algoritmo A-Estrela diminui o tempo de processamento praticamente à metade. Além disso, todos os demais valores também foram reduzidos à mesma proporção do tempo, exceto o caminho simplificado que, em ambos os casos, foi obtido pelo mesmo tipo de processo. A figura 50 enfatiza a realização dos saltos duplos no processamento do algoritmo modificado e o salto simples no A*.

8.2. Segundo Espaço de Configuração

Esta demonstração apresenta o planejamento de trajetória realizado com o algoritmo A* e com o A* Modificado. Ambos foram utilizados para calcular a trajetória do ponto de partida x=101 e y=80 até o ponto x=494 e y=303. Na tabela 2 constam alguns detalhes do processamento dos algoritmos.

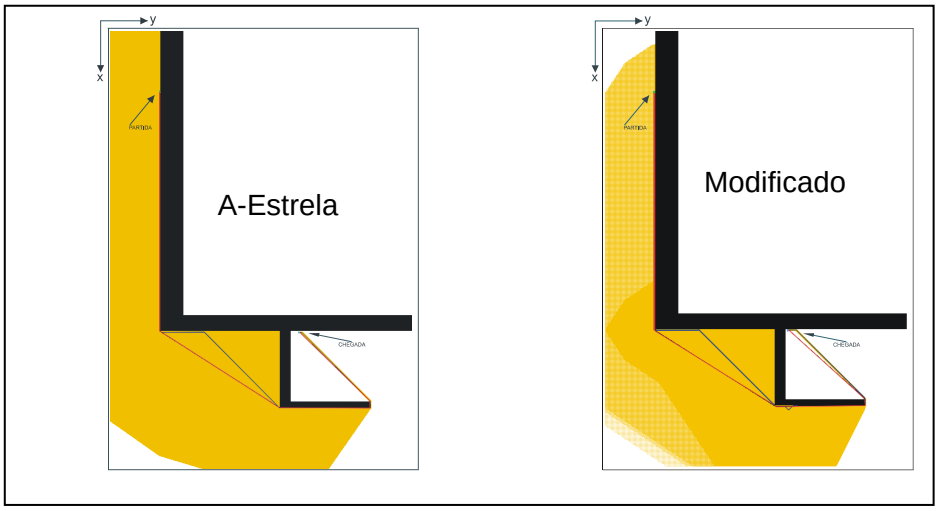


Figura 51 - Planejamento de trajetória no segundo espaço de configuração – Algoritmo Modificado e A*

Tabela 2: Detalhes do processo de planejamento do segundo espaço de configuração

Ponto de partida	(101, 80)	
Ponto de chegada	(494, 303)	
Heurística	Distância de Manhattan	
Método	Modificado	A-Estrela
Células Exploradas	96002	111999
Tempo (ms)	35641	44828
Iterações	94505	1111122
Caminho	615 células	867 células
Caminho simplificado	6 células	6 células

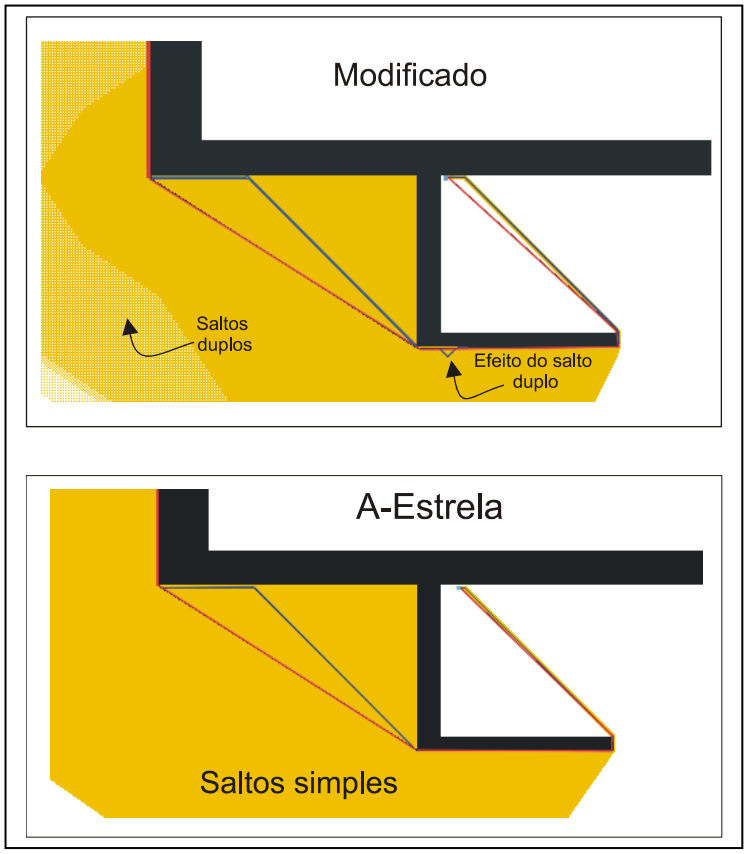


Figura 52 - Detalhamento dos saltos primeiro espaço de configuração

A representação dos resultados do planejamento apresentado na figura 55 segue o mesmo padrão dos resultados apresentados no espaço de configuração anterior. Da mesma forma os resultados apresentados nos detalhes do processamento dos algoritmos presentes na tabela 2 também levam a conclusões já observadas no planejamento anterior. A demonstração do planejamento neste tipo de espaço de configuração objetiva mostrar o efeito do salto duplo em determinadas configurações de espaço e obstáculos. Ocorre que em determinado momento, após a realização de um salto duplo, houve a seleção de um estado que desviou a trajetória do curso normal. Ressalta-se, porém que a simplificação da trajetória elimina totalmente esse problema.

8.3. Terceiro Espaço de Configuração

Esta demonstração apresenta o planejamento de trajetória realizado com o algoritmo A* e com o A* Modificado. Ambos foram utilizados para calcular a trajetória do ponto de partida x=43 e y=68 até o ponto x=618 e y=541. Na tabela 3 constam alguns detalhes do processamento dos algoritmos.

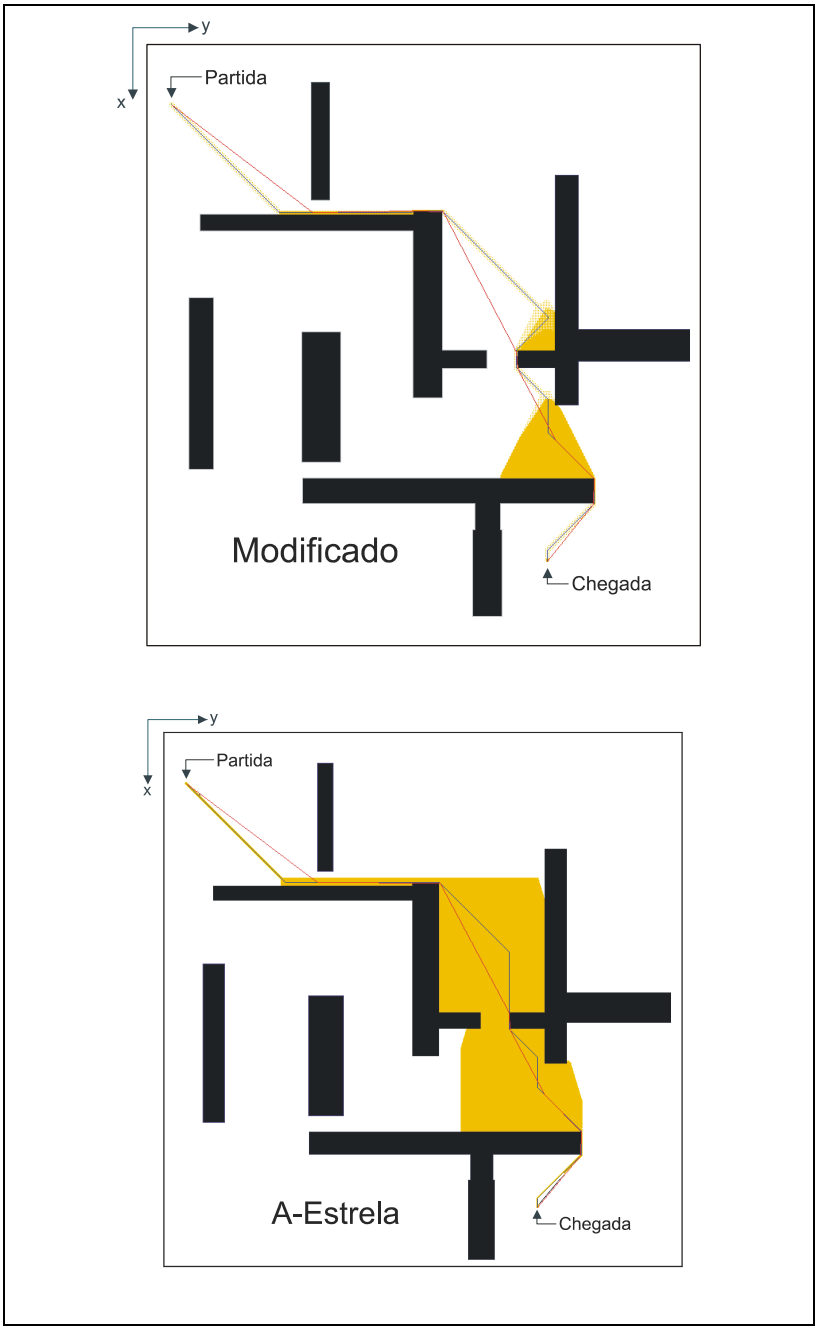


Figura 53 - Planejamento de trajetória no terceiro espaço de configuração – Algoritmo Modificado e A*

Tabela 3: Detalhes do processo de planejamento do terceiro espaço de configuração

Ponto de partida	(43, 68)	
Ponto de chegada	(618, 541)	
Heurística	Distância de Manhattan	
Método	Modificado	A-Estrela
Células Exploradas	**10344**	49741
Tempo (ms)	3609	19532
Iterações	8440	**48254**
Caminho	427 células	780 células
Caminho simplificado	12 células	11células

Este planejamento segue a mesma a representação dos demais. Da mesma forma os resultados apresentados nos detalhes do processamento dos algoritmos presentes na tabela 3 também levam a conclusões já observadas nos planejamentos anteriores. A demonstração do planejamento deste espaço de configuração objetiva mostrar que em uma configuração que seja favorável à realização de saltos duplos, o A* modificado processa a busca do caminho bem mais rápido do que o A*. Ou seja, o ganho em tempo de processamento é bastante significativo para situações como a apresentada.

Todas as trajetórias obtidas desses planejamentos foram executadas pelo robô. Comprovando assim a precisão do planejamento bem como a eficiência da comunicação entre o PCT e o robô. Observou-se que o erro odométrico não afetou significativamente os resultados obtidos devido à baixa velocidade de deslocamento do robô (limitada em 300mm/s) e ao fato de que as dimensões do ambiente explorado foram relativamente pequenas.

9. Conclusões e Trabalhos Futuros

No presente trabalho foi apresentado o Planejador de Trajetória para Robô Móvel Autônomo Utilizando Algoritmo A-Estrela com Salto Duplo. Para tal, diferentes técnicas de planejamento de trajetória de robôs foram pesquisadas. Dentre essas técnicas o método escolhido foi considerado o mais adequado ao problema de locomoção de um autônomo em ambiente com obstáculos. Dessa forma, o algoritmo A-Estrela com Salto Duplo atendeu aos quesitos precisão e eficácia, esperados desde o início das pesquisas desse trabalho. Apesar da nova característica atribuída a este algoritmo, ele pode ser considerado bastante eficiente para situações em que se deseja encontrar caminhos em ambientes com uma quantidade média de obstáculos. Pois em um ambiente com muitos obstáculos, o algoritmo é forçado a trabalhar com saltos simples, na maioria do tempo, diminuindo assim a sua performance.

Os resultados apresentados, tanto da simulação com o Planejador quanto com o robô, comprovaram a eficiência do sistema desenvolvido bem como do algoritmo. O principal problema encontrado foi referente à comunicação entre o sistema de planejamento e o robô, uma vez que esse processo, durante alguns testes, apresentou falha. Isso ocorreu devido à falha nas portas de comunicação do sistema operacional, que por problemas de conflito na configuração não detectava as portas de comunicação corretamente.

O fato de o Planejador ter sido desenvolvido na linguagem Java, o torna um sistema portável, ou seja, o torna independente de plataforma. Com isso, em trabalhos futuros, pode-se pensar até em utilizar o Planejador totalmente embarcado em algum autômato.

Algumas melhorias na interface do Planejador podem ser realizadas, incluindo melhoras na visualização como, por exemplo, alguma representação em três dimensões, que possibilitaria mostrar de forma mais eficiente o ambiente de trabalho do sistema robótico. Assim como, a elaboração de uma abordagem dinâmica deste trabalho deve ser o curso natural de continuidade desta pesquisa, considerando que o algoritmo está pronto, o Planejador já foi implementado e testado. Dessa forma, testes com obstáculos móveis poderão ser realizados num trabalho que queira dar continuidade a este, inclusive adicionando diversos obstáculos móveis. Do mesmo modo, é interessante que o caminho encontrado venha a ser mais próximo da realidade, ou seja, um caminho que permita movimentos de forma contínua e suave.

10. Referências Bibliográficas

Antonio C.M., ***Conceitos Básicos da Teoria de Grafos***, disponível: http://www.inf.ufsc.br/grafos.html , acesso em maio/2008;

BOISSONNAT J.D., BURDICK J., GOLDBERG K., HUTCHINSON S, ***Algorithmic Foundations of Robotics V***, Springer., 2003;

BOUDREAU T.,GLICK J., GREENE S., SPURLIN V.,WOEHR J., ***NetBeans: The Definitive Guide***, O'Reilly, 2003;

CAPEK K., SELVER P., PLAYFAIR, ***R.U.R***., Dover Publications., 2001;

COHN M., MORGAN B., MORRISON M., NYGARD M. T., JOSHI D., TRINKO T., ***Java Developer's Reference***, diponível: http://www.webbasedprogramming.com/java-developers-reference, acesso em dezembro/2007;

CROWLEY, JAMES L. ***Navigation for an Intelligent Mobile Robot***. IEEE Journal of Robotics and Automation, v.RA-1, n.1, p.31-41, mar./1985.

DAVIDSON, D. A., WATSON, A. I., AND SELMAN, P. H., ***An evaluation of GIS as an aid to the planning of proposed developments in rural areas. In Geographical Information Handling: Research and Applications***, P. M. Mather (London: Wiley) , 1993;

DOBRIVOJE P., VIJAY P.B., ***Methods and Tools for Applied Artificial Intelligence***, CRC Press, 1994;

EDER R., ***Navegação Em Ambientes Semiestruturados Por Um Sistema Robótico Autônomo Em Tempo Real***, Dissertação de Mestrado, IME, Rio de Janeiro, 2003;

FIRBY, R. JAMES, ***An Architecture for a Synthetic Vacuum Cleaner***. In: AAAi Fall Symposium Series Workshop On Instantiating Real-World Agents, 1993, Raleigh. Proceedings... [s. l.];

HARVEY M. D., ***Java: Como Programar***, Prentice-Hall, 2005;

JAHANBIN, M.R., FALLSIDE, F., ***Path Planning Using a Wave Simulation Technique in the Configuration Space***. In: GERO, J.S. Artificial intelligence in engineering: robotics and process. Amsterdan: Elsiver, 1988.

JOSEPH F. P., ***The Science Fiction of Isaac Asimov***, Doubleday., 1974;

KOLLN W L, ***Algoritmo de Dijkstra em LISP***,Curso de Ciências da Computação e Programação Funcional, INE5363, Agosto/2006

LAGES W. F., ***Controle e Estimação de Posição e Orientação de Robôs Móveis***, Tese de Doutorado, ITA, São José dos Campos , 1998;

LAVALLE S. M., ***Planning Algorithms***, Cambridge University Press, 2006;

Lora F.A.S., Hemerly E.M., Lages W.F, ***Sistema para Navegação e Guiagem de Robôs Móveis Autônomos***, SBA Controle e Automação, Vol. 9, N°3, 1998;

NETBEANS, ***Netbeans Web Site***, disponível: http://www.netbeans.org, acesso: janeiro/2008;

OLIVEIRA JR. S. G., ***Desenvolvimento De Um Robô Com Rodas Autônomo***, Dissertação de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2008;

OTTONI G. L, ***Planejamento De Trajetórias Para Robôs Móveis***, Projeto de Graduação - Curso de Engenharia de Computação, Fundação Universidade Federal do Rio Grande, 2000;

DE OLIVEIRA V.M., ***Técnicas de Controle de Robôs Móveis***, Dissertação de Mestrado, Universidade do Estado de Santa Catarina – UFSC, Florianópolis – SC, 2001;

PATEL'S A., ***Dijkstra's Algorithm and Best-First-Search***, disponível: www.theory.stanford.edu/AStar.html, acesso: maio/2008;

PIERI E.R., ***Curso de Robótica Móvel Apostila***, UFSC – Universidade Federal de Santa Catarina, 2002;

POLLI H. B., NAZARIO F. G. E AMARAL S., **Navegação Autônoma em Ambientes Desconhecidos Utilizando o Robô Móvel Khepera**, XXI Congresso de Iniciação Cientifica e Tecnológica em Engenharia, 2006;

RAMON F., ***Guia de Consulta Rápida – Java 2***, Novatec Editora, 2001;

REZENDE F.A.V.S, ALMEIDA R.M.V., NOBRE F.F., ***Diagramas de Voronoi para a definição de áreas de abrangência de hospitais públicos no Município do Rio de Janeiro***. In: Cad. Saúde Pública v.16 n.2 Rio de Janeiro, abril./junho 2000.

RIBEIRO C., COSTA A., ROMERO R., ***Robôs Móveis Inteligentes: Princípios e Técnicas***, Anais do XXI Congresso da Sociedade Brasileira de Computação - SBC2001, vol. 03, 2001;

RUSSELL S., NORVING P., ***Artificial Intelligence***, Prentice-Hall, 2003;

SCHALKOFF, ROBERT J., ***Artifitial Intelligence: an engineering aproach***. New York: McGraw Hill, 1990. 646p.

TILOVE R. B., ***Local Obstacle Avoidance for Mobile Robots Based on the Method of Artificial Potentials***, Research Laboratories, Computer Science Dept, General Motors Corporation,1989;

VORONOI; diponível: http://www.voronoi.com/ acesso maio/2008.

11. Apêndices

11.1. Apêndice 1: Listagem do Programa em Java – NetBeans 6.0.1

Trajeto.java

```
package planejatrajeto;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.event.*;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.awt.image.PixelGrabber;
import java.awt.image.RenderedImage;
import java.awt.Point;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.Timer;
import java.util.TimerTask;
import javax.imageio.ImageIO;
import javax.imageio.stream.FileImageOutputStream;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class Trajeto extends JFrame {
    public final int EM_ABERTO = 1;
    public final int EM_FECHADO = 2;
    public final int NAO_INICIADO = 0;
    public final int ENCONTRADO = 1;
    public final int INEXISTENTE = 2;
    public final int LIVRE = 0;
    public final int OCUPADO = 1;
    public final int NUNCA = 0;
    public static int origemX,  origemY;
    public static int destinoX,  destinoY;
    public static int larguraMapa,  alturaMapa;
    public static int itensAbertos;
    public static int canto;
    public static int[] listAberta;
    public static int[] xAberto,  yAberto;
    public static int[] custoF;
    public static int[] custoH;
    public static int[] dimCaminho;
    public static int[] localCaminho;
    public static int[] listaCaminho;
    public static int[] xpath,  ypath;
    public static int[][] walkability;
    public static int[][] listarray;
    public static int[][] paiX,  paiY;
    public static int[][] custoG;
    public static Point[] filhos,  caminho,  auxcaminho;
    public static boolean gon1,  gon2;
    public static int trajetoria[][];
    public static Graphics g;
    public BufferedImage image = null;

    public Trajeto() {
        initComponents();
        this.setVisible(true);
        inicializa();
        inicio1.setVisible(false);
        fim1.setVisible(false);    }
    public static void main(String[] args) {
```

```
    Trajeto trajeto = new Trajeto();}
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {
    Toolbar = new javax.swing.JToolBar();
    btnOrigem = new javax.swing.JButton();
    btnDestino = new javax.swing.JButton();
    btnNovo = new javax.swing.JButton();
    btnSimular = new javax.swing.JButton();
    btnSalvar = new javax.swing.JButton();
    btnAbrir = new javax.swing.JButton();
    btnEnviar = new javax.swing.JButton();
    btnSair = new javax.swing.JButton();
    Separator_1 = new javax.swing.JToolBar.Separator();
    btnCirculo = new javax.swing.JButton();
    btnArredondado = new javax.swing.JButton();
    btnLinha = new javax.swing.JButton();
    btnRetangulo = new javax.swing.JButton();
    btnBorracha = new javax.swing.JButton();
    btnZoom = new javax.swing.JButton();
    LayerEspaco = new javax.swing.JLayeredPane();
    LabelIteracoes = new javax.swing.JLabel();
    lbIteracoes = new java.awt.Label();
    LabelTempExec = new javax.swing.JLabel();
    lbTempoexec = new java.awt.Label();
    LabelPartida = new javax.swing.JLabel();
    posinicial = new javax.swing.JLabel();
    LabelChegada = new javax.swing.JLabel();
    posfinal = new javax.swing.JLabel();
    espaco = new planejatrajeto.PaintPanel();
    inicio1 = new planejatrajeto.inicio();
    fim1 = new planejatrajeto.fim();
    statusbar1 = new planejatrajeto.StatusBar();
    statusbar2 = new planejatrajeto.StatusBar();
    LabelCOtimizado = new javax.swing.JLabel();
    LabelCelExplor = new javax.swing.JLabel();
    lbCelExplor = new java.awt.Label();
    LabelComp = new javax.swing.JLabel();
    lbCaminho = new java.awt.Label();
    lblCotimizado = new java.awt.Label();
    statusbar = new planejatrajeto.StatusBar();
    LabelCelExplor1 = new javax.swing.JLabel();
    LabelComp1 = new javax.swing.JLabel();
    LabelCOtimizado1 = new javax.swing.JLabel();
    ScrollPaneAreaTexto = new javax.swing.JScrollPane();
    AreaTexto = new javax.swing.JTextArea();
    jSeparator1 = new javax.swing.JSeparator();
    jLabel1 = new javax.swing.JLabel();
    cbalgoritmo = new javax.swing.JComboBox();
    ComboMetodo = new javax.swing.JComboBox();
    ComboPorta = new javax.swing.JComboBox();
    ComboVelocidade = new javax.swing.JComboBox();
    velocidade = new javax.swing.JSlider();
    checkCaminho = new javax.swing.JCheckBox();
    checkSuave = new javax.swing.JCheckBox();
    jLabel2 = new javax.swing.JLabel();
    jSeparator2 = new javax.swing.JSeparator();
    jSeparator3 = new javax.swing.JSeparator();
    jLabel3 = new javax.swing.JLabel();
    jSeparator4 = new javax.swing.JSeparator();
    jLabel4 = new javax.swing.JLabel();
    edtPixelCent = new javax.swing.JTextField();
    jLabel5 = new javax.swing.JLabel();
    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("Planejamento de Trajetoria");
    setFont(new java.awt.Font("Arial", 1, 12));
    setForeground(java.awt.Color.white);
    setName("FrmPrincipal"); // NOI18N
```

```
        setResizable(false);

Toolbar.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.BevelBorder.LOWER
ED));
        Toolbar.setOrientation(1);
        Toolbar.setRollover(true);
        Toolbar.setAutoscrolls(true);
        Toolbar.setMargin(new java.awt.Insets(2, 2, 2, 2));
        Toolbar.setMaximumSize(new java.awt.Dimension(64, 600));
        Toolbar.setMinimumSize(new java.awt.Dimension(54, 600));
        Toolbar.setName("Toolbar"); // NOI18N
        Toolbar.setPreferredSize(new java.awt.Dimension(64, 600));

        btnOrigem.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_start.png"))); // NOI18N
        btnOrigem.setToolTipText("Origem");
        btnOrigem.setAlignmentX(0.5F);

btnOrigem.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.creat
eBevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnOrigem.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnOrigem.setDefaultCapable(false);
        btnOrigem.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnOrigem.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnOrigem.setMaximumSize(new java.awt.Dimension(50, 50));
        btnOrigem.setMinimumSize(new java.awt.Dimension(50, 50));
        btnOrigem.setName("btnOrigem"); // NOI18N
        btnOrigem.setOpaque(false);
        btnOrigem.setPreferredSize(new java.awt.Dimension(50, 50));
        btnOrigem.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_start.png"))); // NOI18N
        btnOrigem.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnOrigemMousePressed(evt);
            }
        });
        Toolbar.add(btnOrigem);

        btnDestino.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_end.png"))); // NOI18N
        btnDestino.setToolTipText("Destino");
        btnDestino.setAlignmentX(0.5F);

btnDestino.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.creat
eBevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnDestino.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnDestino.setDefaultCapable(false);
        btnDestino.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnDestino.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnDestino.setMaximumSize(new java.awt.Dimension(50, 50));
        btnDestino.setMinimumSize(new java.awt.Dimension(50, 50));
        btnDestino.setName("btnDestino"); // NOI18N
        btnDestino.setPreferredSize(new java.awt.Dimension(50, 50));
        btnDestino.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_end.png"))); // NOI18N
        btnDestino.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnDestino.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnDestinoMousePressed(evt);}
        });
        Toolbar.add(btnDestino);

        btnNovo.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_filenew.png"))); // NOI18N
        btnNovo.setToolTipText("Novo");
        btnNovo.setAlignmentX(0.5F);
btnNovo.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.createB
```

```
evelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnNovo.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnNovo.setDefaultCapable(false);
        btnNovo.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnNovo.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnNovo.setMaximumSize(new java.awt.Dimension(50, 50));
        btnNovo.setMinimumSize(new java.awt.Dimension(50, 50));
        btnNovo.setName("btnNovo"); // NOI18N
        btnNovo.setPreferredSize(new java.awt.Dimension(50, 50));
        btnNovo.setPressedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_filenew.png"))); // NOI18N
        btnNovo.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnNovo.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                btnNovoMouseClicked(evt);
            }
        });
        Toolbar.add(btnNovo);

        btnSimular.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_process.png"))); // NOI18N
        btnSimular.setToolTipText("Simular");
        btnSimular.setAlignmentX(0.5F);

btnSimular.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.creat
eBevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnSimular.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnSimular.setDefaultCapable(false);
        btnSimular.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnSimular.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnSimular.setMaximumSize(new java.awt.Dimension(50, 50));
        btnSimular.setMinimumSize(new java.awt.Dimension(50, 50));
        btnSimular.setName("btnSimular"); // NOI18N
        btnSimular.setPreferredSize(new java.awt.Dimension(50, 50));
        btnSimular.setPressedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_process.gif"))); // NOI18N
        btnSimular.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnSimular.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                btnSimularMouseClicked(evt);
            }
        });
        Toolbar.add(btnSimular);

        btnSalvar.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_save.png"))); // NOI18N
        btnSalvar.setToolTipText("Salvar");
        btnSalvar.setAlignmentX(0.5F);

btnSalvar.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.create
BevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnSalvar.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnSalvar.setDefaultCapable(false);
        btnSalvar.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnSalvar.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnSalvar.setMaximumSize(new java.awt.Dimension(50, 50));
        btnSalvar.setMinimumSize(new java.awt.Dimension(50, 50));
        btnSalvar.setName("btnsalvar"); // NOI18N
        btnSalvar.setPreferredSize(new java.awt.Dimension(50, 50));
        btnSalvar.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_save.png"))); // NOI18N
        btnSalvar.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnSalvar.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnSalvarMousePressed(evt);
            }
        });
```

```
        Toolbar.add(btnSalvar);

        btnAbrir.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_fileopen.png"))); // NOI18N
        btnAbrir.setToolTipText("Abrir");
        btnAbrir.setAlignmentX(0.5F);

btnAbrir.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.createB
evelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnAbrir.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnAbrir.setDefaultCapable(false);
        btnAbrir.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnAbrir.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnAbrir.setMaximumSize(new java.awt.Dimension(50, 50));
        btnAbrir.setMinimumSize(new java.awt.Dimension(50, 50));
        btnAbrir.setName("btnAbrir"); // NOI18N
        btnAbrir.setPreferredSize(new java.awt.Dimension(50, 50));
        btnAbrir.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_fileopen.png"))); // NOI18N
        btnAbrir.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnAbrir.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnAbrirMousePressed(evt);
            }
        });
        Toolbar.add(btnAbrir);

        btnEnviar.setIcon(new          javax.swing.ImageIcon(getClass().getResource("/icones/unselected_wifi-
modem.png"))); // NOI18N
        btnEnviar.setToolTipText("Enviar");
        btnEnviar.setAlignmentX(0.5F);

btnEnviar.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.create
BevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnEnviar.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnEnviar.setDefaultCapable(false);
        btnEnviar.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnEnviar.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnEnviar.setMaximumSize(new java.awt.Dimension(50, 50));
        btnEnviar.setMinimumSize(new java.awt.Dimension(50, 50));
        btnEnviar.setName("btnEnviar"); // NOI18N
        btnEnviar.setPreferredSize(new java.awt.Dimension(50, 50));
        btnEnviar.setPressedIcon(new javax.swing.ImageIcon(getClass().getResource("/icones/selected_wifi-
modem.png"))); // NOI18N
        btnEnviar.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnEnviar.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnEnviarMousePressed(evt);
            }
        });
        Toolbar.add(btnEnviar);

        btnSair.setIcon(new   javax.swing.ImageIcon(getClass().getResource("/icones/System_exit.png")));   //
NOI18N
        btnSair.setToolTipText("Sair");
        btnSair.setAlignmentX(0.5F);

btnSair.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.createBe
velBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnSair.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnSair.setDefaultCapable(false);
        btnSair.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnSair.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnSair.setMaximumSize(new java.awt.Dimension(50, 50));
        btnSair.setMinimumSize(new java.awt.Dimension(50, 50));
        btnSair.setName("btnSair"); // NOI18N
        btnSair.setPreferredSize(new java.awt.Dimension(50, 50));
```

```
        btnSair.setPressedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_System_exit.png"))); // NOI18N
        btnSair.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnSair.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                btnSairMouseClicked(evt);
            }
        });
        Toolbar.add(btnSair);

        Separator_1.setBorder(javax.swing.BorderFactory.createCompoundBorder());
        Separator_1.setFont(new java.awt.Font("Tahoma", 0, 12));
        Separator_1.setMaximumSize(new java.awt.Dimension(60, 15));
        Separator_1.setOpaque(true);
        Separator_1.setPreferredSize(new java.awt.Dimension(60, 15));
        Separator_1.setRequestFocusEnabled(false);
        Toolbar.add(Separator_1);

        btnCirculo.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_ellipse.png"))); // NOI18N
        btnCirculo.setToolTipText("Circulo");
        btnCirculo.setAlignmentX(0.5F);

btnCirculo.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.creat
eBevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnCirculo.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnCirculo.setDefaultCapable(false);
        btnCirculo.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnCirculo.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnCirculo.setMaximumSize(new java.awt.Dimension(50, 50));
        btnCirculo.setMinimumSize(new java.awt.Dimension(50, 50));
        btnCirculo.setName("btnCirculo"); // NOI18N
        btnCirculo.setPreferredSize(new java.awt.Dimension(50, 50));
        btnCirculo.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_ellipse.png"))); // NOI18N
        btnCirculo.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnCirculo.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnCirculoMousePressed(evt);
            }
        });
        Toolbar.add(btnCirculo);

        btnArredondado.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_rounded.png"))); // NOI18N
        btnArredondado.setToolTipText("Quadrado Arredondado");
        btnArredondado.setAlignmentX(0.5F);

btnArredondado.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.
createBevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnArredondado.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnArredondado.setDefaultCapable(false);
        btnArredondado.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnArredondado.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnArredondado.setMaximumSize(new java.awt.Dimension(50, 50));
        btnArredondado.setMinimumSize(new java.awt.Dimension(50, 50));
        btnArredondado.setName("btnArredondado"); // NOI18N
        btnArredondado.setPreferredSize(new java.awt.Dimension(50, 50));
        btnArredondado.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_rounded.png"))); // NOI18N
        btnArredondado.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnArredondado.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnArredondadoMousePressed(evt);
            }
        });
        Toolbar.add(btnArredondado);
```

```
        btnLinha.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_line.png"))); // NOI18N
        btnLinha.setToolTipText("Linha");
        btnLinha.setAlignmentX(0.5F);
btnLinha.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.create
BevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnLinha.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnLinha.setDefaultCapable(false);
        btnLinha.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnLinha.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnLinha.setMaximumSize(new java.awt.Dimension(50, 50));
        btnLinha.setMinimumSize(new java.awt.Dimension(50, 50));
        btnLinha.setName("btnLinha"); // NOI18N
        btnLinha.setPreferredSize(new java.awt.Dimension(50, 50));
        btnLinha.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_line.png"))); // NOI18N
        btnLinha.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnLinha.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnLinhaMousePressed(evt);
            }
        });
        Toolbar.add(btnLinha);

        btnRetangulo.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_rectangle.png"))); // NOI18N
        btnRetangulo.setToolTipText("Retangulo");
        btnRetangulo.setAlignmentX(0.5F);

btnRetangulo.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.cr
eateBevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnRetangulo.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnRetangulo.setDefaultCapable(false);
        btnRetangulo.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnRetangulo.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnRetangulo.setMaximumSize(new java.awt.Dimension(50, 50));
        btnRetangulo.setMinimumSize(new java.awt.Dimension(50, 50));
        btnRetangulo.setName("btnRetangulo"); // NOI18N
        btnRetangulo.setPreferredSize(new java.awt.Dimension(50, 50));
        btnRetangulo.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_rectangle.png"))); // NOI18N
        btnRetangulo.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnRetangulo.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnRetanguloMousePressed(evt);
            }
        });
        Toolbar.add(btnRetangulo);
        btnBorracha.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/unselected_eraser.png"))); // NOI18N
        btnBorracha.setToolTipText("Apagar");
        btnBorracha.setAlignmentX(0.5F);
btnBorracha.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.cre
ateBevelBorder(javax.swing.border.BevelBorder.RAISED), null));
        btnBorracha.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        btnBorracha.setDefaultCapable(false);
        btnBorracha.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        btnBorracha.setMargin(new java.awt.Insets(0, 0, 0, 0));
        btnBorracha.setMaximumSize(new java.awt.Dimension(50, 50));
        btnBorracha.setMinimumSize(new java.awt.Dimension(50, 50));
        btnBorracha.setName("btnBorracha"); // NOI18N
        btnBorracha.setPreferredSize(new java.awt.Dimension(50, 50));
        btnBorracha.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_eraser.png"))); // NOI18N
        btnBorracha.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        btnBorracha.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
```

```
            btnBorrachaMousePressed(evt);
        }
    });
    Toolbar.add(btnBorracha);
    btnZoom.setIcon(new   javax.swing.ImageIcon(getClass().getResource("/icones/selected_lupa.jpg")));
// NOI18N
    btnZoom.setToolTipText("Zoom");
    btnZoom.setAlignmentX(0.5F);
btnZoom.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.create
BevelBorder(javax.swing.border.BevelBorder.RAISED), null));
    btnZoom.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
    btnZoom.setDefaultCapable(false);
    btnZoom.setFocusable(false);
    btnZoom.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    btnZoom.setMargin(new java.awt.Insets(0, 0, 0, 0));
    btnZoom.setMaximumSize(new java.awt.Dimension(50, 50));
    btnZoom.setMinimumSize(new java.awt.Dimension(50, 50));
    btnZoom.setName("btzoom"); // NOI18N
    btnZoom.setPreferredSize(new java.awt.Dimension(50, 50));
    btnZoom.setSelectedIcon(new
javax.swing.ImageIcon(getClass().getResource("/icones/selected_lupa.jpg"))); // NOI18N
    btnZoom.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
    btnZoom.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mousePressed(java.awt.event.MouseEvent evt) {
            btnZoomMousePressed(evt);
        }
    });
    Toolbar.add(btnZoom);
LayerEspaco.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.BevelBorder.LO
WERED));
    LayerEspaco.setMaximumSize(new java.awt.Dimension(800, 800));
    LayerEspaco.setName("LayerEspaco"); // NOI18N
    LayerEspaco.setPreferredSize(new java.awt.Dimension(800, 800));
    LayerEspaco.setRequestFocusEnabled(false);
    LabelIteracoes.setText("Iterações :");
    LabelIteracoes.setBounds(170, 800, 60, 20);
    LayerEspaco.add(LabelIteracoes, javax.swing.JLayeredPane.DEFAULT_LAYER);
    lbIteracoes.setFont(new java.awt.Font("Tahoma", 0, 11));
    lbIteracoes.setName("lbIteracoes"); // NOI18N
    lbIteracoes.setBounds(250, 800, 50, -1);
    LayerEspaco.add(lbIteracoes, javax.swing.JLayeredPane.DEFAULT_LAYER);
    LabelTempExec.setText("Tempo (ms)");
    LabelTempExec.setBounds(170, 770, 70, 20);
    LayerEspaco.add(LabelTempExec, javax.swing.JLayeredPane.DEFAULT_LAYER);
    LabelTempExec.getAccessibleContext().setAccessibleName("Tempo (ms):");
    lbTempoexec.setFont(new java.awt.Font("Tahoma", 0, 11));
    lbTempoexec.setName("lbTempoExec"); // NOI18N
    lbTempoexec.setBounds(250, 770, 80, 20);
    LayerEspaco.add(lbTempoexec, javax.swing.JLayeredPane.DEFAULT_LAYER);
    LabelPartida.setBackground(new java.awt.Color(255, 255, 255));
    LabelPartida.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
    LabelPartida.setText("Origem:");
    LabelPartida.setBounds(30, 770, 50, 20);
    LayerEspaco.add(LabelPartida, javax.swing.JLayeredPane.DEFAULT_LAYER);
    posinicial.setHorizontalAlignment(javax.swing.SwingConstants.TRAILING);
    posinicial.setBounds(80, 770, 60, 20);
    LayerEspaco.add(posinicial, javax.swing.JLayeredPane.DEFAULT_LAYER);
    LabelChegada.setBackground(new java.awt.Color(255, 255, 255));
    LabelChegada.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
    LabelChegada.setText("Destino:");
    LabelChegada.setBounds(30, 800, 50, 20);
    LayerEspaco.add(LabelChegada, javax.swing.JLayeredPane.DEFAULT_LAYER);
    posfinal.setHorizontalAlignment(javax.swing.SwingConstants.TRAILING);
    posfinal.setBounds(80, 800, 60, 20);
    LayerEspaco.add(posfinal, javax.swing.JLayeredPane.DEFAULT_LAYER);
    espaco.setBackground(new java.awt.Color(255, 255, 255));
    espaco.setMinimumSize(new java.awt.Dimension(721, 721));
```

```
        espaco.setName("espaco"); // NOI18N
        espaco.setPreferredSize(new java.awt.Dimension(721, 721));
        espaco.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                espacoMousePressed(evt);
            }
        });
        javax.swing.GroupLayout espacoLayout = new javax.swing.GroupLayout(espaco);
        espaco.setLayout(espacoLayout);
        espacoLayout.setHorizontalGroup(
            espacoLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(espacoLayout.createSequentialGroup()
                .addGap(52, 52, 52)
                .addComponent(inicio1,                    javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(fim1,                    javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(629, Short.MAX_VALUE))
        );
        espacoLayout.setVerticalGroup(
            espacoLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, espacoLayout.createSequentialGroup()
                .addContainerGap(646, Short.MAX_VALUE)
                .addGroup(espacoLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(fim1,                    javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(inicio1,                    javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(39, 39, 39))
        );
        espaco.setBounds(10, 10, 721, 721);
        LayerEspaco.add(espaco, javax.swing.JLayeredPane.DEFAULT_LAYER);
        statusbar1.setBounds(20, 760, 130, 90);
        LayerEspaco.add(statusbar1, javax.swing.JLayeredPane.DEFAULT_LAYER);
        statusbar2.setBounds(160, 760, 190, 90);
        LayerEspaco.add(statusbar2, javax.swing.JLayeredPane.DEFAULT_LAYER);
        LabelCOtimizado.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        LabelCOtimizado.setText("Otimizado");
        LabelCOtimizado.setBounds(630, 790, 70, 20);
        LayerEspaco.add(LabelCOtimizado, javax.swing.JLayeredPane.DEFAULT_LAYER);
        LabelCelExplor.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        LabelCelExplor.setText(" Exploradas");
        LabelCelExplor.setBounds(380, 790, 70, 20);
        LayerEspaco.add(LabelCelExplor, javax.swing.JLayeredPane.DEFAULT_LAYER);
        lbCelExplor.setAlignment(java.awt.Label.CENTER);
        lbCelExplor.setFont(new java.awt.Font("Tahoma", 0, 11));
        lbCelExplor.setName("lbCelExplor"); // NOI18N
        lbCelExplor.setBounds(380, 820, 70, -1);
        LayerEspaco.add(lbCelExplor, javax.swing.JLayeredPane.DEFAULT_LAYER);
        LabelComp.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        LabelComp.setText(" Caminho");
        LabelComp.setBounds(500, 790, 70, 20);
        LayerEspaco.add(LabelComp, javax.swing.JLayeredPane.DEFAULT_LAYER);
        lbCaminho.setAlignment(java.awt.Label.CENTER);
        lbCaminho.setBounds(500, 820, 70, -1);
        LayerEspaco.add(lbCaminho, javax.swing.JLayeredPane.DEFAULT_LAYER);
        lblCotimizado.setBounds(650, 820, 40, -1);
        LayerEspaco.add(lblCotimizado, javax.swing.JLayeredPane.DEFAULT_LAYER);
        statusbar.setBounds(360, 760, 370, 90);
        LayerEspaco.add(statusbar, javax.swing.JLayeredPane.DEFAULT_LAYER);
        LabelCelExplor1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        LabelCelExplor1.setText("Celulas");
        LabelCelExplor1.setBounds(379, 770, 60, 20);
        LayerEspaco.add(LabelCelExplor1, javax.swing.JLayeredPane.DEFAULT_LAYER);
        LabelComp1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        LabelComp1.setText("Comprimento");
```

```
        LabelComp1.setBounds(490, 770, 90, 20);
        LayerEspaco.add(LabelComp1, javax.swing.JLayeredPane.DEFAULT_LAYER);
        LabelCOtimizado1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        LabelCOtimizado1.setText("Caminho");
        LabelCOtimizado1.setBounds(630, 770, 60, 20);
        LayerEspaco.add(LabelCOtimizado1, javax.swing.JLayeredPane.DEFAULT_LAYER);
        AreaTexto.setColumns(100);
        AreaTexto.setRows(100);
        ScrollPaneAreaTexto.setViewportView(AreaTexto);
        jLabel1.setFont(new java.awt.Font("Tahoma", 1, 12));
        jLabel1.setText("Busca");
        cbalgoritmo.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Modificado", "Astar" }));
cbalgoritmo.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.createEtchedBorder(), "Algoritmo"));
        cbalgoritmo.setName("cbalgoritmo"); // NOI18N
        ComboMetodo.setMaximumRowCount(6);
        ComboMetodo.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Manhattan", "Max(dx,dy)", "Diagonal" }));
ComboMetodo.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.createEtchedBorder(), "Heurística"));
        ComboMetodo.setMinimumSize(new java.awt.Dimension(96, 47));
        ComboMetodo.setName("ComboMetodo"); // NOI18N
        ComboPorta.setMaximumRowCount(12);
        ComboPorta.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "-------", "COM1", "COM2", "COM3", "COM4", "COM5", "COM6", "COM7", "COM8", "COM9", "COM10", "COM11", "COM12" }));

ComboPorta.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.createEtchedBorder(), "Porta"));
        ComboPorta.setMinimumSize(new java.awt.Dimension(96, 47));
        ComboPorta.setName("ComboVelocidade"); // NOI18N

        ComboVelocidade.setMaximumRowCount(6);
        ComboVelocidade.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "--------", "9600", "14400", "19200", "38400", "56000", "57600", "115200", "128000", "230400", "256000", "460800", "921600" }));

ComboVelocidade.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.createEtchedBorder(), "Velocidade"));
        ComboVelocidade.setMinimumSize(new java.awt.Dimension(96, 47));
        ComboVelocidade.setName("ComboMetodo"); // NOI18N

        velocidade.setMajorTickSpacing(50);
        velocidade.setMaximum(280);
        velocidade.setMinorTickSpacing(10);
        velocidade.setPaintLabels(true);
        velocidade.setPaintTicks(true);
        velocidade.setSnapToTicks(true);
        velocidade.setToolTipText("");
        velocidade.setValue(34);
        velocidade.setAutoscrolls(true);

velocidade.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.createEtchedBorder(), "velocidade do Movimento do Robô"));
        velocidade.setMaximumSize(new java.awt.Dimension(30, 30));
        velocidade.setMinimumSize(new java.awt.Dimension(0, 0));
        velocidade.setName("velocidade"); // NOI18N
        velocidade.setPreferredSize(new java.awt.Dimension(280, 50));
        velocidade.setValueIsAdjusting(true);
        checkCaminho.setText("Caminho");
        checkCaminho.setName(""); // NOI18N
        checkSuave.setText("Suavização do Caminho");

        jLabel2.setFont(new java.awt.Font("Tahoma", 1, 12));
        jLabel2.setText("Robô");
        jLabel3.setFont(new java.awt.Font("Tahoma", 1, 12));
```

```
        jLabel3.setText("Visualizar");
        jLabel4.setFont(new java.awt.Font("Tahoma", 1, 12));
        jLabel4.setText("Comunicação");
        edtPixelCent.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        edtPixelCent.setText("2.6");
        jLabel5.setText("Pixel para Centímetro");
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(Toolbar, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(LayerEspaco, javax.swing.GroupLayout.PREFERRED_SIZE, 746,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(jLabel5)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(edtPixelCent, javax.swing.GroupLayout.PREFERRED_SIZE, 44,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 373,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addGroup(layout.createSequentialGroup()
                                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 50,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addGroup(layout.createSequentialGroup()
                                        .addGap(40, 40, 40)
                                        .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 310,
javax.swing.GroupLayout.PREFERRED_SIZE))
                                    .addGroup(layout.createSequentialGroup()
                                        .addComponent(cbalgoritmo, javax.swing.GroupLayout.PREFERRED_SIZE, 140,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 46,
Short.MAX_VALUE)
                                        .addComponent(ComboMetodo, javax.swing.GroupLayout.PREFERRED_SIZE,
140, javax.swing.GroupLayout.PREFERRED_SIZE)
                                        .addGap(24, 24, 24)))
                                .addGap(174, 174, 174))
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED_SIZE, 65,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jSeparator3, javax.swing.GroupLayout.PREFERRED_SIZE, 281,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 50,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(checkCaminho, javax.swing.GroupLayout.PREFERRED_SIZE, 130,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(checkSuave, javax.swing.GroupLayout.DEFAULT_SIZE, 394,
Short.MAX_VALUE))
                            .addGroup(layout.createSequentialGroup()
                                .addGap(40, 40, 40)
                                .addComponent(jSeparator2, javax.swing.GroupLayout.PREFERRED_SIZE, 310,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
                                .addComponent(velocidade, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 342, Short.MAX_VALUE)
```

```
                        .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                            .addComponent(ComboPorta,   javax.swing.GroupLayout.PREFERRED_SIZE,   80,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(ComboVelocidade,  javax.swing.GroupLayout.PREFERRED_SIZE,
80, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                            .addComponent(jLabel4,      javax.swing.GroupLayout.PREFERRED_SIZE,      91,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(jSeparator4,      javax.swing.GroupLayout.DEFAULT_SIZE,      247,
Short.MAX_VALUE)))))
                .addGroup(layout.createSequentialGroup()
                    .addComponent(ScrollPaneAreaTexto, javax.swing.GroupLayout.PREFERRED_SIZE, 293,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,              231,
javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addGap(213, 213, 213))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(21, 21, 21)
                .addComponent(Toolbar, javax.swing.GroupLayout.DEFAULT_SIZE, 748, Short.MAX_VALUE)
                .addGap(801, 801, 801))
            .addGroup(layout.createSequentialGroup()
                .addGap(20, 20, 20)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel1)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(10, 10, 10)
                        .addComponent(jSeparator1,      javax.swing.GroupLayout.PREFERRED_SIZE,      10,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(cbalgoritmo,      javax.swing.GroupLayout.PREFERRED_SIZE,      50,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(ComboMetodo,      javax.swing.GroupLayout.PREFERRED_SIZE,      50,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel5)
                    .addComponent(edtPixelCent,      javax.swing.GroupLayout.PREFERRED_SIZE,      28,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(31, 31, 31)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel3)
                    .addComponent(jSeparator3,                  javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(26, 26, 26)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(checkCaminho)
                    .addComponent(checkSuave))
                .addGap(27, 27, 27)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel2)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(10, 10, 10)
                        .addComponent(jSeparator2,      javax.swing.GroupLayout.PREFERRED_SIZE,      10,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(velocidade,          javax.swing.GroupLayout.PREFERRED_SIZE,          71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(88, 88, 88)
```

```
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(jLabel4)
                .addComponent(jSeparator4, javax.swing.GroupLayout.PREFERRED_SIZE, 10,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(ComboPorta, javax.swing.GroupLayout.PREFERRED_SIZE, 50,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(ComboVelocidade, javax.swing.GroupLayout.PREFERRED_SIZE, 50,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 43,
Short.MAX_VALUE)
            .addComponent(ScrollPaneAreaTexto, javax.swing.GroupLayout.PREFERRED_SIZE, 217,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(802, 802, 802))
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(LayerEspaco, javax.swing.GroupLayout.PREFERRED_SIZE, 868,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(691, Short.MAX_VALUE))
    );

    pack();
  }// </editor-fold>
  private void btnSimularMouseClicked(java.awt.event.MouseEvent evt) {
    btnSimular.setSelected(true);
    this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    statusbar.setMessage("Buscando o melhor  caminho");
    Procurar();
    buscacaminho();
    this.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    statusbar.setMessage("Melhor Caminho encontrado");
    btnSimular.setSelected(false);
}

  private void btnNovoMouseClicked(java.awt.event.MouseEvent evt) {

    btnNovo.setSelected(true);
    PaintPanel.clearPaint();
    AreaTexto.setText("");
    inicio1.setVisible(false);
    fim1.setVisible(false);
    repaint();
    reinicializa();
    inicializa();
    origemX = 0;
    origemY = 0;
    destinoX = 0;
    destinoY = 0;
}

  private void btnSairMouseClicked(java.awt.event.MouseEvent evt) {
    System.exit(0);
}

  private void btnOrigemMousePressed(java.awt.event.MouseEvent evt) {
    btnOrigem.isFocusOwner();
    btnOrigem.setSelected(true);
    btnDestino.setSelected(false);
}

  private void btnDestinoMousePressed(java.awt.event.MouseEvent evt) {
    btnDestino.isFocusOwner();
    btnOrigem.setSelected(false);
    btnDestino.setSelected(true);
}
```

```
    private void btnCirculoMousePressed(java.awt.event.MouseEvent evt) {
        btnCirculo.setSelected(true);
        btnLinha.setSelected(false);
        btnRetangulo.setSelected(false);
        btnArredondado.setSelected(false);
        btnBorracha.setSelected(false);
        PaintPanel.setShape(figura.OVAL);
        statusbar.setMessage("Desenhando obstáculos");
    }

    private void btnArredondadoMousePressed(java.awt.event.MouseEvent evt) {
        btnCirculo.setSelected(false);
        btnLinha.setSelected(false);
        btnRetangulo.setSelected(false);
        btnArredondado.setSelected(true);
        btnBorracha.setSelected(false);
        PaintPanel.setShape(figura.RECTROUND);
        statusbar.setMessage("Desenhando obstáculos");
    }

    private void btnLinhaMousePressed(java.awt.event.MouseEvent evt) {
        btnCirculo.setSelected(false);
        btnLinha.setSelected(true);
        btnRetangulo.setSelected(false);
        btnArredondado.setSelected(false);
        btnBorracha.setSelected(false);
        PaintPanel.setShape(figura.LINE);
        statusbar.setMessage("Desenhando obstáculos");
    }

    private void btnRetanguloMousePressed(java.awt.event.MouseEvent evt) {
        btnCirculo.setSelected(false);
        btnLinha.setSelected(false);
        btnRetangulo.setSelected(true);
        btnArredondado.setSelected(false);
        btnBorracha.setSelected(false);
        PaintPanel.setShape(figura.RECTANGLE);
        statusbar.setMessage("Desenhando obstáculos");
    }

    private void btnBorrachaMousePressed(java.awt.event.MouseEvent evt) {
        btnCirculo.setSelected(false);
        btnLinha.setSelected(false);
        btnRetangulo.setSelected(false);
        btnArredondado.setSelected(false);
        btnBorracha.setSelected(true);
        PaintPanel.setColor(Color.white);
        PaintPanel.setShape(figura.ERASE);
        statusbar.setMessage("Apagando");
    }

    private void btnAbrirMousePressed(java.awt.event.MouseEvent evt) {
        btnAbrir.setSelected(true);
        Abrir();
    }

    private void btnSalvarMousePressed(java.awt.event.MouseEvent evt) {
        btnSalvar.setSelected(true);
        Salvar();
    }

    private void btnEnviarMousePressed(java.awt.event.MouseEvent evt) {
        btnEnviar.setSelected(true);

        String trinca;
        String COM = null;
        int VelocidadePorta = -1;
```

```
int c = ComboPorta.getSelectedIndex();
int p = ComboVelocidade.getSelectedIndex();
int estado = 0;

if (estado >= 0) {
    switch (c) {

        case 1:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 2:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 3:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 4:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 5:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 6:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 7:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 8:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 9:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 10:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 11:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        case 12:
            COM = (String) ComboPorta.getSelectedItem();
            break;
        default:
            JOptionPane.showMessageDialog(this, "Selecione a Porta");
            estado = -1;
            btnEnviar.setSelected(false);
            btnEnviar.repaint();
    }
}

if (estado >= 0) {
    switch (p) {
        case 1:
            VelocidadePorta = 9600;
            break;
        case 2:
            VelocidadePorta = 14400;
            break;
        case 3:
            VelocidadePorta = 19200;
            break;
        case 4:
            VelocidadePorta = 38400;
            break;
        case 5:
            VelocidadePorta = 56000;
```

```
                break;
            case 6:
                VelocidadePorta = 57600;
                break;
            case 7:
                VelocidadePorta = 115200;
                break;
            case 8:
                VelocidadePorta = 128000;
                break;
            case 9:
                VelocidadePorta = 230400;
                break;
            case 10:
                VelocidadePorta = 256000;
                break;
            case 11:
                VelocidadePorta = 460800;
                break;
            case 12:
                VelocidadePorta = 921600;
                break;
            default:
                JOptionPane.showMessageDialog(this, "Selecione a Velocidade");
                estado = -1;
                btnEnviar.setSelected(false);
                btnEnviar.repaint();
        }
    }

    if (estado >= 0) {
        this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        trinca = AreaTexto.getText();
        trinca = trinca.replaceAll("\n", "");
        String subtrinca = "";
        int idtrinca = 0;
        SerialComLeitura serialEscrita = new SerialComLeitura(COM, VelocidadePorta, 0);
        serialEscrita.ObterIdDaPorta();
        while (idtrinca != (trinca.length())) {
            if (trinca.charAt(idtrinca) != '$') {
                serialEscrita.HabilitaEscrita();
                serialEscrita.AbrirPorta();
                subtrinca = subtrinca + trinca.charAt(idtrinca);
                serialEscrita.EnviarUmaString(subtrinca);
                delay(80);
                serialEscrita.FecharCom();
                subtrinca = "";
                idtrinca = idtrinca + 1;
            } else {
                serialEscrita.HabilitaEscrita();
                serialEscrita.AbrirPorta();
                subtrinca = subtrinca + trinca.charAt(idtrinca);
                serialEscrita.EnviarUmaString(subtrinca);
                delay(80);
                serialEscrita.FecharCom();
                subtrinca = "";
                idtrinca = idtrinca + 1;
            }
        }
        serialEscrita.HabilitaEscrita();
        serialEscrita.AbrirPorta();
        serialEscrita.EnviarUmaString("@");
        delay(80);
        serialEscrita.FecharCom();
        this.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    }
}
```

```
    private void espacoMousePressed(java.awt.event.MouseEvent evt) {
        if (btnOrigem.isFocusOwner()) {
            PaintPanel.putPositions(origemX, origemY);
            origemX = evt.getX();
            origemY = evt.getY();
            espaco.remove(inicio1);
            inicio1.setBounds(origemX - 6, origemY - 32, 30, 30);
            inicio1.setVisible(true);
            espaco.add(inicio1);
            statusbar.setMessage("Partida (" + Integer.toString(origemY) + " , " + Integer.toString(origemX) +
")");
            statusbar.repaint();
            PaintPanel.setShape(figura.START);
            posinicial.setText(Integer.toString(origemY) + " , " + Integer.toString(origemX));
            this.repaint();
        }
        if (btnDestino.isFocusOwner()) {
            PaintPanel.putPositions(destinoX, destinoY);
            destinoX = evt.getX();
            destinoY = evt.getY();
            espaco.remove(fim1);
            fim1.setBounds(destinoX - 6, destinoY - 32, 30, 30);
            fim1.setVisible(true);
            espaco.add(fim1);
            statusbar.setMessage("Chegada (" + Integer.toString(destinoY) + " , " + Integer.toString(destinoX)
+ ")");
            PaintPanel.setShape(figura.END);
            posfinal.setText(Integer.toString(destinoY) + " , " + Integer.toString(destinoX));
        }
}

    private void btnZoomMousePressed(java.awt.event.MouseEvent evt) {
        image = PaintPanel._bufImage;
        JFrame frame = new JFrame();
        TransformingCanvas canvas = new TransformingCanvas(image);
        canvas.addMouseListener(canvas);
        canvas.addMouseMotionListener(canvas);
        canvas.addMouseWheelListener(canvas);
        frame.setLayout(new BorderLayout());
        frame.getContentPane().add(canvas, BorderLayout.CENTER);
        frame.setSize(500, 500);
        frame.setLocation(200, 200);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setVisible(true);
}
//----------------delay-------------------------------------
    private void delay(int d) {
        try {
            Thread.sleep(d);
        } catch (InterruptedException e) {
            AreaTexto.append("Erro na Thread: " + e);
        }
    }
//------------Reinicialização da execução do algoritmo-----------------------------
    private void reinicializa() {
        alturaMapa = 0;
        larguraMapa = 0;
        walkability = null;
        listAberta = null;
        xAberto = null;
        yAberto = null;
        custoF = null;
        custoH = null;
        dimCaminho = null;
        localCaminho = null;
        listaCaminho = null;
```

```
        xpath = null;
        ypath = null;
        listarray = null;
        paiX = null;
        paiY = null;
        custoG = null;
        filhos = null;
    }
    //------------Inicialização Variáveis do algoritmo----------------------------
    public void inicializa() {
        alturaMapa = 720;
        larguraMapa = 720;
        walkability = new int[alturaMapa + 1][larguraMapa + 1];
        listAberta = new int[(larguraMapa * alturaMapa) + 2];
        xAberto = new int[(larguraMapa * alturaMapa) + 2];
        yAberto = new int[(larguraMapa * alturaMapa) + 2];
        custoF = new int[(larguraMapa + 1) * (alturaMapa + 1)];
        custoH = new int[(larguraMapa * alturaMapa) + 2];
        dimCaminho = new int[2];
        localCaminho = new int[2];
        listaCaminho = new int[(larguraMapa + 1) * (alturaMapa + 1)];
        xpath = new int[2];
        ypath = new int[2];
        listarray = new int[larguraMapa + 1][alturaMapa + 1];
        paiX = new int[larguraMapa + 1][alturaMapa + 1];
        paiY = new int[larguraMapa + 1][alturaMapa + 1];
        custoG = new int[larguraMapa + 1][alturaMapa + 1];
        filhos = new Point[8];
        for (int i = 0; i < 8; i++) {
            filhos[i] = new Point();
        }
        for (int linha = 0; linha <= (alturaMapa - 1); linha++) {
            for (int coluna = 0; coluna <= (larguraMapa - 1); coluna++) {
                walkability[linha][coluna] = 0;
            }
        }
    }
//------------Salvar Texto ou Imagem-------------------------------------------
    private void Salvar() {
        try {
            File arquivo = null;
            JFileChooser arq = new JFileChooser();
            FileNameExtensionFilter filter1 = new FileNameExtensionFilter("Texto", "txt");
            FileNameExtensionFilter filter2 = new FileNameExtensionFilter("Imagem", "png");
            arq.addChoosableFileFilter(filter1);
            arq.addChoosableFileFilter(filter2);
            arq.setAcceptAllFileFilterUsed(false);
            int Result = arq.showSaveDialog(this);
            if (Result == JFileChooser.APPROVE_OPTION) {
                if (arq.getFileFilter() == filter1) {
                    arquivo = arq.getSelectedFile();
                    FileWriter inArq = new FileWriter(arquivo.getPath());
                    inArq.write(AreaTexto.getText());
                    inArq.close();
                    btnSalvar.setSelected(false);
                }
                if (arq.getFileFilter() == filter2) {
                    String nome1 = arq.getSelectedFile().getPath() + ".png";
                    File f = new File(nome1);
                    FileImageOutputStream fios = new FileImageOutputStream(f);
                    String ext = "png";
                    ImageIO.write((RenderedImage) PaintPanel._bufImage, ext, fios);
                    btnSalvar.setSelected(false);
                }
            }
            if (Result == JFileChooser.CANCEL_OPTION) {
                btnSalvar.setSelected(false);
```

```
            }

        } catch (IOException ioe) {
            JOptionPane.showMessageDialog(this, "Erro ao salvar o arquivo");
            btnSalvar.setSelected(false);
        }
    }
//------------Abrir Texto ou Imagem-----------------------------------------
    private void Abrir() {
        BufferedImage abreimagem = null;
        try {
            JFileChooser arq = new JFileChooser();
            int Result = arq.showOpenDialog(this);
            if (Result == JFileChooser.APPROVE_OPTION) {
                File curFile = arq.getSelectedFile();
                abreimagem = ImageIO.read(curFile);
                espaco.getGraphics().drawImage(abreimagem, 1, 1, this);
                PaintPanel._bufImage = abreimagem;
                btnAbrir.setSelected(false);
            }
            if (Result == JFileChooser.CANCEL_OPTION) {
                btnAbrir.setSelected(false);
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "Erro ao abrir o arquivo");
            btnAbrir.setSelected(false);
        }
    }

//------------Buscar Caminho-----------------------------------------------------
    private void buscacaminho() {
        int path = 0;
        int pathx = 0;
        int pathy = 0;
        int tempx = 0;
        int tempy = 0;
        int temp = 0;
        int cellposition = 0;
        int paiXval = 0, paiYval = 0;
        int u = 0, v = 0;
        int a = 0, b = 0;
        int m = 0;
        int i = 0, j = 0, k = 0;
        int x = 0;
        int pathfinderid = 0;
        int tempgcost = 0;
        int squareschecked = 0;
        int novocusto = 0;
        StopWatch s = new StopWatch();
        canto = 0;
        gon1 = false;
        gon2 = false;
        if ((origemX == destinoX) && (origemY == destinoY) && (localCaminho[pathfinderid] > 0)) {
            path = ENCONTRADO;
            System.exit(0);
        }
        if ((origemX == destinoX) && (origemY == destinoY) && (localCaminho[pathfinderid] == 0)) {
            path = INEXISTENTE;
            System.exit(0);
        }
        dimCaminho[pathfinderid] = NAO_INICIADO;
        localCaminho[pathfinderid] = NAO_INICIADO;
        itensAbertos = 1;
        listAberta[1] = 1;
        xAberto[1] = origemX;
        yAberto[1] = origemY;
        int iteracoes = 0;
```

```
lbIteracoes.setText(Integer.toString(iteracoes));
lbIteracoes.setVisible(true);
s.start();
do {
   lbTempoexec.setText("Executando...");
   lbIteracoes.setText(Long.toString(iteracoes));
   if (itensAbertos != 0) {
      iteracoes++;
      paiXval = xAberto[listAberta[1]];
      paiYval = yAberto[listAberta[1]];
      listarray[paiXval][paiYval] = EM_FECHADO;
      itensAbertos = itensAbertos - 1;
      listAberta[1] = listAberta[itensAbertos + 1];
      heapsort();
      if (cbalgoritmo.getSelectedItem().equals("Astar")) {
         gon1 = true;
      } else {
         verificanivel1(paiXval, paiYval);
      }
      if (gon1) {
         i = obterfilhosnivel1(paiXval, paiYval);
      } else {
         i = obterfilhosnivel12(paiXval, paiYval);
      }
      for (int y = 0; y <= filhos.length - 1; y++) {
         g = espaco.getGraphics();
         g.setColor(Color.orange);
         g.fillRect(filhos[y].x, filhos[y].y, 1, 1);
         PaintPanel.fx = filhos[y].x;
         PaintPanel.fy = filhos[y].y;
         PaintPanel.setShape(figura.FILHO);
         espaco.repaint();
      }
      for (j = 0; j <= (i - 1); j = j + 1) {
         a = filhos[j].x;
         b = filhos[j].y;
         if (listarray[a][b] != EM_ABERTO) {
            squareschecked = squareschecked + 1;
            m = itensAbertos + 1;
            listAberta[m] = squareschecked;
            xAberto[listAberta[m]] = a;
            yAberto[listAberta[m]] = b;
            calculacusto(a, b, paiXval, paiYval, m, destinoX, destinoY);
            paiX[a][b] = paiXval;
            paiY[a][b] = paiYval;
            ordenalista(m);
            itensAbertos = itensAbertos + 1;
            listarray[a][b] = EM_ABERTO;
            lbCelExplor.setText(Integer.toString(squareschecked));
         } else {
            atualizacusto(a, b, paiXval, paiYval, m);
            novocusto = novocusto + 1;
         }
      }
   } else {
      path = INEXISTENTE;
      JOptionPane.showMessageDialog(this, " Caminho não encontrado!");
      gon1 = false;
      break;
   }
   if (listarray[destinoX][destinoY] == EM_ABERTO) {
      path = ENCONTRADO;
      gon1 = false;
      break;
   }
} while (NUNCA != 5);
s.stop();
```

```
        lbTempoexec.setText(Long.toString(s.getElapsedTime()));
        pegacaminho(path, pathfinderid);
        if ((checkCaminho.isSelected()) == true) {
            mostrarcaminho();
            if ((checkSuave.isSelected()) == true) {
                suavizar(caminho);
                CalculaVelocidade(caminho);
            }
        }
    }
    private void time(final int i) {
        int delay = 50000;   // delay for 5 sec.
        int period = 10000;  // repeat every sec.
        Timer timer = new Timer();
        timer.scheduleAtFixedRate(new TimerTask() {
            public void run() {
            }
        }, delay, period);
    }

//------------Calculo velocidades-----------------------------------------------
    private void CalculaVelocidade(Point[] _caminho) {
//1 centímetro tem 38 pixels logo 1px = 2.6cm
        double[][] trajeto;
        trajeto = new double[_caminho.length][2];
        int ii = _caminho.length - 1;
//copia dos valores das coordenadas
//do vetor _caminho(inteiros) para o
// vetor trajeto(double) - e necessario devido ao
//ao uso de Math.acos()
        for (int i = 0; i <= ii; i++) {
            trajeto[i][0] = (_caminho[i].x);
            trajeto[i][1] = (_caminho[i].y);
        }
        int i = 0;
        double x1, x2, x3 = 0;
        double y1, y2, y3 = 0;
        double dist_1, dist_2, dist_3 = 0;
        double[] teta;
        double velocidade_exec = 0;
        double velocidade_conversao = 0;
        double velocidade_D, velocidade_E = 0;
        // a quantidade de retas é o número de coordenadas - 1
        int retas = trajeto.length - 1;
        // o numero de curvas é a quantidade de retas -1
        int curvas = retas - 1;
        velocidade_exec = velocidade.getValue();
        double[] distancias = new double[retas + curvas];
       // o tamanho do vetor trajetoria final é igual
        //a qtd de retas + a qtd de curvas
        trajetoria = new int[retas + curvas][3];
        // contém o valor do angulo
        teta = new double[curvas];
        // equação de conversão de velocidade em mm/s para Set Point do MCU
        velocidade_conversao = (velocidade_exec + 2.049) / 0.182;
        //só existem dois pontos, não precisa calcular angulo
        if (retas == 1)
        {   x1 = trajeto[i][0];
            x2 = trajeto[i + 1][0];
            y1 = trajeto[i][1];
            y2 = trajeto[i + 1][1];
            distancias[retas - 1] = Math.sqrt(Math.pow((x2 - x1), 2)
                    + Math.pow((y2 - y1), 2));
            velocidade_D = (int) velocidade_conversao;
            velocidade_E = (int) (velocidade_conversao * -1);
            trajetoria[i][0] = (int) Math.round(distancias[retas - 1]
            *Double.parseDouble(edtPixelCent.getText())/(velocidade_exec*0.021));
```

```
    trajetoria[i][1] = (int) velocidade_D;
    trajetoria[i][2] = (int) velocidade_E;
} else {
    while (i < retas - 1) {
        x1 = trajeto[i][0];
        x2 = trajeto[i + 1][0];
        //calculando o angulo
        x3 = trajeto[i + 2][0];
        y1 = trajeto[i][1];
        y2 = trajeto[i + 1][1];
        //calculando o angulo
        y3 = trajeto[i + 2][1];
        //calculando o angulo
        dist_1 = (Math.sqrt(Math.pow((x2 - x1), 2)
                + Math.pow((y2 - y1), 2))) * 100;
        //calculando o angulo
        dist_2 = (Math.sqrt(Math.pow((x3 - x2), 2)
                + Math.pow((y3 - y2), 2))) * 100;
        //calculando o angulo
        dist_3 = (Math.sqrt(Math.pow((x3 - x1), 2)
                + Math.pow((y3 - y1), 2))) * 100;
        //calculando o angulo
        teta[i] = (Math.pow(dist_2, 2) + Math.pow(dist_1, 2)
                - Math.pow(dist_3, 2)) / (2 * dist_2 * dist_1);
        //calculando o angulo
        teta[i] = Math.round(teta[i] * Math.pow(10, 4)) / Math.pow(10, 4);
        //calculando o angulo
        teta[i] = Math.acos(teta[i]);
        //Teste de curva para direita(+angulo) ou esquerda(-angulo).
        if (x3 > x2 & y3 > y2) {
            if (x1 < x3 & y1 > y3) {
                teta[i] = teta[i] * -1;
            }
        }
        if (x3 < x2 & y3 < y2) {
            if (x1 > x3 & y1 < y2) {
                teta[i] = teta[i] * -1;
            }
        }
        if (x3 < x2 & y3 > y2) {
            if (x1 < x2 & y1 < y3) {
                teta[i] = teta[i] * -1;
            }
        }
        if (x3 > x2 & y3 < y2) {
            if (x1 > x2 & y1 > y3) {
                teta[i] = teta[i] * -1;
            }
        }
        i = i + 1;
    }
    i = 0;
    velocidade_D = (int) velocidade_conversao;
    ii = 0;
   // atualiza o vetor  distancias inserindo a distancia entre os pontos
    while (i < retas) {
        x1 = trajeto[i][0];
        x2 = trajeto[i + 1][0];
        y1 = trajeto[i][1];
        y2 = trajeto[i + 1][1];
        distancias[ii] = Math.sqrt(Math.pow((x2 - x1), 2)
                + Math.pow((y2 - y1), 2));
        i = i + 1;
        ii = ii + 2;
    }
    i = 0;
    ii = 1;
```

```
            while (i <= teta.length - 1) {
                // atualiza o vetor  distancias inserindo o comprimento da curva
                distancias[ii] = teta[i] * (180 / Math.PI);
                ii = ii + 2;
                i = i + 1;
            }
            i = 0;
            ii = 0;
            // calculo tempo velocidade roda direita/esquerda no vetor trajetoria
            while (i <= trajetoria.length - 1)
            {
                double teste = 0;
                //se trecho for entre pontos
                if (i % 2 == 0)
                {   velocidade_D = (int) velocidade_conversao;
                    velocidade_E = (int) (velocidade_conversao * -1);
                    trajetoria[i][0] = (int) Math.abs(Math.round(distancias[i]
                            * Double.parseDouble(edtPixelCent.getText())
                            / (velocidade_exec * 0.021)));
                    trajetoria[i][1] = (int) velocidade_D;
                    trajetoria[i][2] = (int) velocidade_E;
                //se trecho for curva
                } else
                {//se angulo positivo
                    if (teta[ii] < 0)
                    {
                        velocidade_D = (int) (velocidade_conversao * -1);
                        velocidade_E = (int) (velocidade_conversao * -1);
                        teste = (((180 - distancias[i]) * (Math.PI / 180) * 150)
                                / (velocidade_exec * 0.021));
                        trajetoria[i][0] = (int) (((180 + distancias[i])
                          * (Math.PI / 180) * 150) / (velocidade_exec * 0.021));

                        trajetoria[i][0] = Math.abs(trajetoria[i][0]);
                        trajetoria[i][1] = (int) velocidade_D;
                        trajetoria[i][2] = (int) velocidade_E;
                        ii = ii + 1;
                    //se angulo negativo
                    } else
                    {
                        velocidade_D = (int) velocidade_conversao;
                        velocidade_E = (int) velocidade_conversao;
                        teste = (((180 - distancias[i]) * (Math.PI / 180) * 150) / (velocidade_exec * 0.021));
                        trajetoria[i][0] = (int) (((180 - distancias[i]) * (Math.PI / 180) * 150) / (velocidade_exec *
0.021));
                        trajetoria[i][0] = Math.abs(trajetoria[i][0]);
                        trajetoria[i][1] = (int) velocidade_D;
                        trajetoria[i][2] = (int) velocidade_E;
                        ii = ii + 1;
                    }
                }
                i = i + 1;
            }
        }
        DecimalFormat FTempo = new DecimalFormat("00000");
        DecimalFormat FVelox = new DecimalFormat("0000");
        i = 0;
        ii = trajetoria.length - 1;
        AreaTexto.setText(null);
        String saida = "";
        String Tempo = "";
        String RD = "";
        String RE = "";
        for (i = 0; i <= ii; i++) {
            Tempo = FTempo.format(trajetoria[i][0]);
            RD = FVelox.format(trajetoria[i][1]);
            RE = FVelox.format(trajetoria[i][2]);
```

```
            saida = ("#" + Tempo + "*" + RD + "*" + RE + "$\n");
            AreaTexto.append(saida);
            saida = "";
        }
    }
//------------Suavizar-----------------------------------------------------
    public void suavizar(Point[] _caminho) {
        Point a1, a2;
        Point[] auxtrajetoria;
        int i, j, k, u, p, x, y;
        int nobst;
        int limx1, limx2, limy1, limy2;
        auxcaminho = new Point[dimCaminho[0]];
        for (i = 0; i < dimCaminho[0]; i = i + 1) {
            auxcaminho[i] = new Point();
        }
        a1 = new Point();
        a2 = new Point();
        nobst = 0;
        u = 0;
        p = 0;
        a1.x = _caminho[0].x;
        a1.y = _caminho[0].y;
        a2.x = _caminho[1].x;
        a2.y = _caminho[1].y;
        auxcaminho[p].x = _caminho[0].x;
        auxcaminho[p].y = _caminho[0].y;
        u = u + 1;
        p = p + 1;
        do {
            if ((a1.y) <= (a2.y)) {
                limy1 = a1.y;
                limy2 = a2.y;
            } else {
                limy1 = a2.y;
                limy2 = a1.y;
            }
            if ((a1.x) <= (a2.x)) {
                limx1 = a1.x;
                limx2 = a2.x;
            } else {
                limx1 = a2.x;
                limx2 = a1.x;
            }
            for (j = limy1; j <= limy2; j = j + 1) {
                for (k = limx1; k <= limx2; k = k + 1) {
                    if (walkability[k][j] == 1) {
                        nobst = nobst + 1;
                        break;
                    }
                }
            }
            if (nobst == 0) {
                u = u + 1;
                a2.x = _caminho[u].x;
                a2.y = _caminho[u].y;
            } else {
                a1.x = a2.x;
                a1.y = a2.y;
                auxcaminho[p].x = a1.x;
                auxcaminho[p].y = a1.y;
                p = p + 1;
                u = u + 1;
                a2.x = _caminho[u].x;
                a2.y = _caminho[u].y;
                nobst = 0;
            }
```

```
            lblCotimizado.setText(Integer.toOctalString(p));
        } while (!((a2.x == destinoX) && (a2.y == destinoY)));
        auxcaminho[p].x = destinoX;
        auxcaminho[p].y = destinoY;
        lblCotimizado.setText(Integer.toOctalString(p + 1));
        caminho = new Point[p + 1];

        for (i = 0; i < (p + 1); i = i + 1) {
            caminho[i] = new Point();
        }
        System.arraycopy(auxcaminho, 0, caminho, 0, p + 1);
        int tam = caminho.length;
        PaintPanel.Path(caminho, tam, "OptimizedPath");
    }

//------------ReadxPath e ReadyPath-----------------------------------------------------
    int readpathx(int pathfinderid, int localCaminho) {
        int x = 0;
        if (localCaminho <= dimCaminho[pathfinderid]) {
            x = listaCaminho[(localCaminho * 2) + 10];
        }
        return x;
    }
    int readpathy(int pathfinderid, int localCaminho) {
        int y = 0;
        if (localCaminho <= dimCaminho[pathfinderid]) {
            y = listaCaminho[(localCaminho * 2) + 11];
        }
        return y;
    }
//------------MostrarCaminho-----------------------------------------------------
    void mostrarcaminho() {
        int i, x, y;
        int[] _c;
        int[] _o;
        caminho = new Point[dimCaminho[0]];
        for (int j = 0; j < dimCaminho[0]; j = j + 1) {
            caminho[j] = new Point();
        }
        _c = new int[dimCaminho[0]];
        _o = new int[dimCaminho[0]];
        caminho[0].x = origemX;
        caminho[0].y = origemY;
        _c[0] = origemX;
        _o[0] = origemY;
        for (i = 0; i <= (dimCaminho[0] - 2); i = i + 1) {
            x = readpathx(0, i);
            y = readpathy(0, i);
            caminho[i + 1].x = x;
            caminho[i + 1].y = y;
            _c[i + 1] = x;
            _o[i + 1] = y;
        }
        caminho[dimCaminho[0] - 1].x = destinoX;
        caminho[dimCaminho[0] - 1].y = destinoY;
        _c[dimCaminho[0] - 1] = destinoX;
        _o[dimCaminho[0] - 1] = destinoY;
        i = dimCaminho[0];
        PaintPanel.Path(caminho, i, "PathFound");
        repaint();
    }
//------------CalculaGcost-----------------------------------------------------
    int calculagcost(int a, int b, int paiXval, int paiYval) {
        int ge;
        if (((Math.abs(a - paiXval)) == 1) && ((Math.abs(b - paiYval)) == 1)) {
            //custo para ir para diagonal
            ge = 14;
```

```
        } else {
            //custo para ir para vertical ou horizontal
            ge = 10;
        }
        return ge;
    }
//------------AtualizaCusto-------------------------------------------------------
    void atualizacusto(int a, int b, int paiXval, int paiYval, int m) {
        int x, tempgcost, addedgcost;
        addedgcost = calculagcost(a, b, paiXval, paiYval);
        tempgcost = custoG[paiXval][paiYval] + addedgcost;
        if (tempgcost < custoG[a][b]) {
            paiX[a][b] = paiXval;
            paiY[a][b] = paiYval;
            custoG[a][b] = tempgcost;
            if (listarray[a][b] == EM_ABERTO) {
                for (x = 1; x <= itensAbertos; x = x + 1) {
                    if ((xAberto[listAberta[x]] == a)
                            && (yAberto[listAberta[x]] == b)) {
                        custoF[listAberta[x]] = custoG[a][b]
                                + custoH[listAberta[x]];
                        m = x;
                        ordenalista(m);
                        break;
                    }
                }
            }
        }
    }
//------------OrdenaLista---------------------------------------------------------
    void ordenalista(int m) {
        int temp;
        while (m != 1) {
            if (custoF[listAberta[m]] <= custoF[listAberta[m / 2]]) {
                temp = listAberta[m / 2];
                listAberta[m / 2] = listAberta[m];
                listAberta[m] = temp;
                m = m / 2;
            } else {
                break;
            }
        }
    }

//------------EstimatecustoH------------------------------------------------------------
    int estimatecustoH(int a, int b, int destinoX, int destinoY) {
        int distanciax, distanciay, h;
        h = 0;
        distanciax = 0;
        distanciay = 0;
        int c = ComboMetodo.getSelectedIndex();
        int estado = 0;
        if (estado >= 0) {
            switch (c) {
                case 0:
                    h = 10 * ((Math.abs(a - destinoX)) + (Math.abs(b - destinoY)));
                    break;
                case 1:
                    distanciax = Math.abs(a - destinoX);
                    distanciay = Math.abs(b - destinoY);
                    if (distanciax > distanciay) {
                        h = 10 * distanciax;
                    } else {
                        h = 10 * distanciay;
                    }
                    break;
                case 2:
```

```
                distanciax = Math.abs(a - destinoX);
                distanciay = Math.abs(b - destinoY);
                if (distanciax > distanciay) {
                    h = (14 * distanciay) + 10 * (distanciax - distanciay);
                } else {
                    h = (14 * distanciax) + 10 * (distanciay - distanciax);
                }
                break;
            default:
                JOptionPane.showMessageDialog(this, "Selecione a heurística");
                estado = -1;
        }
    }
    return h;
}
//------------CalculaCusto-----------------------------------------------------
    void calculacusto(int a, int b, int paiXval, int paiYval, int m, int destinoX, int destinoY) {
        int addedgcost;
        addedgcost = calculagcost(a, b, paiXval, paiYval);
        custoG[a][b] = custoG[paiXval][paiYval] + addedgcost;
        custoH[listAberta[m]] = estimatecustoH(a, b, destinoX, destinoY);
        custoF[listAberta[m]] = custoG[a][b] + custoH[listAberta[m]];
    }
    void pegacaminho(int path, int pathfinderid) {
        int pathx, pathy, tempx, cellposition;
        if (path == ENCONTRADO) {
            pathx = destinoX;
            pathy = destinoY;
            do {
                tempx = paiX[pathx][pathy];
                pathy = paiY[pathx][pathy];
                pathx = tempx;
                dimCaminho[pathfinderid] = dimCaminho[pathfinderid] + 1;
                lbCaminho.setText(Integer.toString(dimCaminho[0]));
            } while ((pathx != origemX) || (pathy != origemY));
            pathx = destinoX;
            pathy = destinoY;
            cellposition = (dimCaminho[pathfinderid] * 2) + 10;
            do {
                cellposition = cellposition - 2;
                listaCaminho[cellposition] = pathx;
                listaCaminho[cellposition + 1] = pathy;
                tempx = paiX[pathx][pathy];
                pathy = paiY[pathx][pathy];
                pathx = tempx;
            } while ((pathx != origemX) || (pathy != origemY));
            cellposition = 0;
        }
    }
//------------Heapsort---------------------------------------------------------
    void heapsort() {
        int v, u, temp;
        v = 1;
        do {
            u = v;
            if (((2 * u) + 1) <= itensAbertos) {
                if (custoF[listAberta[u]] >= custoF[listAberta[(2 * u)]]) {
                    v = 2 * u;
                }
                if (custoF[listAberta[v]] >= custoF[listAberta[(2 * u) + 1]]) {
                    v = ((2 * u) + 1);
                }
            } else {
                if ((2 * u) <= itensAbertos) {
                    if (custoF[listAberta[u]] >= custoF[listAberta[2 * u]]) {
                        v = (2 * u);
                    }
```

```
                }
            }
            if (u != v) {
                temp = listAberta[u];
                listAberta[u] = listAberta[v];
                listAberta[v] = temp;
            } else {
                break;
            }
        } while (NUNCA != 5);
    }
//------------verificanivel1-----------------------------------------------------
    void verificanivel1(int x, int y) {
        int a, b, k;
        k = 0;
        b = y - 1;
        a = x - 1;
        while (b <= (y + 1)) {
            while (a <= (x + 1)) {
                if (((a + 1) >= destinoX) && ((b + 1) >= destinoY)) {
                    b = y + 1;
                    k = k + 1;
                    break;
                } else {
                    if ((a != -1) && (b != -1) && (a <= (larguraMapa - 1))
                            && (b <= (alturaMapa - 1))) {
                        if (listarray[a][b] != EM_FECHADO)
                        {
                            if (walkability[a][b] == OCUPADO)
                            {
                                k = k + 1;
                            }
                        }
                    }
                }
                a = a + 1;
            }
            a = x - 1;
            b = b + 1;
        }
        if (k == 0) {
            gon1 = false;
            gon2 = true;
        } else {
            gon1 = true;
            gon2 = false;
        }
    }
//------------ultrapassacanto----------------------------------------------------
    void ultrapassacanto(int paiXval, int paiYval, int a, int b) {
        if (a == (paiXval - 1)) {
            if (b == (paiYval - 1)) {
                if (((walkability[paiXval - 1][paiYval]) == OCUPADO)
                        || ((walkability[paiXval][paiYval - 1]) == OCUPADO)) {
                    canto = OCUPADO;
                }
            } else if (b == (paiYval + 1)) {
                if (((walkability[paiXval][paiYval + 1]) == OCUPADO)
                        || ((walkability[paiXval - 1][paiYval]) == OCUPADO)) {
                    canto = OCUPADO;
                }
            }
        } else if (a == (paiXval + 1)) {
            if (b == (paiYval - 1)) {
                if (((walkability[paiXval][paiYval - 1]) == OCUPADO)
                        || ((walkability[paiXval + 1][paiYval]) == OCUPADO)) {
                    canto = OCUPADO;
```

```
                }
            } else if (b == (paiYval + 1)) {
                if (((walkability[paiXval + 1][paiYval]) == OCUPADO)
                        || ((walkability[paiXval][paiYval + 1]) == OCUPADO)) {
                    canto = OCUPADO;
                }
            }
        }
    }
//------------obterfilhosnivel1-------------------------------------------------
    int obterfilhosnivel1(int x, int y) {
        int i = 0;
        int a = 0;
        int b = 0;
        if (gon1) {
            i = 0;
            for (b = y - 1; b <= (y + 1); b = b + 1) {
                for (a = x - 1; a <= (x + 1); a = a + 1) {
                    if ((a != -1) && (b != -1) && (a <= (larguraMapa - 1))
                            && (b <= (alturaMapa - 1))) {
                        if (listarray[a][b] != EM_FECHADO)
                        {
                            if (walkability[a][b] != OCUPADO)
                            {
                                canto = LIVRE;
                                ultrapassacanto(x, y, a, b);
                                if (canto == LIVRE) {
                                    filhos[i].x = a;
                                    filhos[i].y = b;
                                    i = i + 1;
                                }
                            }
                        }
                    }
                }
            }
        }
        return (i);
    }
//------------obterfilhosnivel12-------------------------------------------------
    int obterfilhosnivel12(int x, int y) {
        int i = 0;
        int a = 0;
        int b = 0;
        if (gon2) {
            i = 0;
            b = y - 2;
            for (a = x - 2; a <= (x + 2); a = a + 1) {
                if ((a != -2) && (a != -1) && (b != -2) && (b != -1)
                        && (a <= (larguraMapa - 1)) && (b <= (alturaMapa - 1))) {
                    if (listarray[a][b] != EM_FECHADO)
                    {
                        if (walkability[a][b] == OCUPADO)
                        {
                            if (listarray[a][b + 1] != EM_FECHADO)
                            {
                                if (walkability[a][b + 1] != OCUPADO)
                                {
                                    if (a == (x - 2)) {
                                        filhos[i].x = a + 1;
                                        filhos[i].y = b + 1;
                                        i = i + 1;
                                    }
                                    if (a == x) {
                                        filhos[i].x = a;
                                        filhos[i].y = b + 1;
                                        i = i + 1;
```

```
                    }
                    if (a == (x + 2)) {
                        filhos[i].x = a - 1;
                        filhos[i].y = b + 1;
                        i = i + 1;
                    }
                }
            }
        } else {
            if ((a == (x - 2)) || (a == x) || (a == (x + 2))) {
                filhos[i].x = a;
                filhos[i].y = b;
                i = i + 1;
            }
        }
    }
  }
}
a = x - 2;
for (b = y - 1; b <= (y + 1); b = b + 1) {
   if ((a != -2) && (a != -1) && (b != -2) && (b != -1)
        && (a <= (larguraMapa - 1)) && (b <= (alturaMapa - 1))) {
      if (listarray[a][b] != EM_FECHADO)
      {
         if (walkability[a][b] == OCUPADO)
         {
            if (listarray[a + 1][b] != EM_FECHADO)
            {
               if (b == y) {
                  filhos[i].x = a + 1;
                  filhos[i].y = b;
                  i = i + 1;
               }
            }
         } else {
            if (b == y) {
               filhos[i].x = a;
               filhos[i].y = b;
               i = i + 1;
            }
         }
      }
   }
}
b = y + 2;
for (a = x - 2; a <= (x + 2); a = a + 1) {
   if ((a != -2) && (a != -1) && (b != -2)
    && (b != -1) && (a <= (larguraMapa - 1))
    && (b <= (alturaMapa - 1))) {
      if (listarray[a][b] != EM_FECHADO)
      {
         if (walkability[a][b] == OCUPADO)
         {
            if (listarray[a][b - 1] != EM_FECHADO)
            {
               if (walkability[a][b - 1] != OCUPADO)
               {
                  if (a == (x - 2)) {
                     filhos[i].x = a + 1;
                     filhos[i].y = b - 1;
                     i = i + 1;
                  }
                  if (a == x) {
                     filhos[i].x = a;
                     filhos[i].y = b - 1;
                     i = i + 1;
                  }
```

```
                        if (a == (x + 2)) {
                            filhos[i].x = a - 1;
                            filhos[i].y = b - 1;
                            i = i + 1;
                        }
                    }
                }
            } else {
                if ((a == (x - 2)) || (a == x) || (a == (x + 2))) {
                    filhos[i].x = a;
                    filhos[i].y = b;
                    i = i + 1;
                }
            }
        }
    }
    a = x + 2;
    for (b = y - 1; b <= (y + 1); b = b + 1) {
        if ((a != -2) && (a != -1) && (b != -2)
                && (b != -1) && (a <= (larguraMapa - 1))
                && (b <= (alturaMapa - 1))) {
            if (listarray[a][b] != EM_FECHADO)
            {
                if (walkability[a][b] == OCUPADO)
                {
                    if (listarray[a - 1][b] != EM_FECHADO)
                    {
                        if (b == y) {
                            filhos[i].x = a - 1;
                            filhos[i].y = b;
                            i = i + 1;
                        }
                    }
                } else {
                    if (b == y) {
                        filhos[i].x = a;
                        filhos[i].y = b;
                        i = i + 1;
                    }
                }
            }
        }
    }

    return i;
}
//------------Procurar----------------------------------------------------------
void Procurar() {
    int w = larguraMapa;
    int h = alturaMapa;
    int[] pixels = new int[w * h];
    int coluna;
    int linha;
    String mapa = "";
    int data[] = new int[w * h];
    PixelGrabber pg = new PixelGrabber(PaintPanel._bufImage,
            0, 0, w, h, pixels, 0, w);
    try {
        pg.grabPixels();
    } catch (Exception e) {
        System.out.println("PixelGrabber exception");
    }
    int[] temppixels = (int[]) pg.getPixels();
    int index;
    for (index = 0; index < temppixels.length; index++) {
```

```
        if ((temppixels[index] == -16777216) || (temppixels[index]
                == -16711936) || (temppixels[index] == -65536)
                || (temppixels[index] == -20561)) {
            data[index] = 0;
        } else {
            data[index] = 1;
        }
    }
    index = 0;
    for (linha = 0; linha <= (alturaMapa - 1); linha++) {
        for (coluna = 0; coluna <= (larguraMapa - 1); coluna++) {
            {
                walkability[coluna][linha] = data[index];
                index = index + 1;
            }
        }
    }
}
// Variables declaration - do not modify
private javax.swing.JTextArea AreaTexto;
private javax.swing.JComboBox ComboMetodo;
private javax.swing.JComboBox ComboPorta;
private javax.swing.JComboBox ComboVelocidade;
private javax.swing.JLabel LabelCOtimizado;
private javax.swing.JLabel LabelCOtimizado1;
private javax.swing.JLabel LabelCelExplor;
private javax.swing.JLabel LabelCelExplor1;
private javax.swing.JLabel LabelChegada;
private javax.swing.JLabel LabelComp;
private javax.swing.JLabel LabelComp1;
private javax.swing.JLabel LabelIteracoes;
private javax.swing.JLabel LabelPartida;
private javax.swing.JLabel LabelTempExec;
private javax.swing.JLayeredPane LayerEspaco;
private javax.swing.JScrollPane ScrollPaneAreaTexto;
private javax.swing.JToolBar.Separator Separator_1;
private javax.swing.JToolBar Toolbar;
private javax.swing.JButton btnAbrir;
private javax.swing.JButton btnArredondado;
private javax.swing.JButton btnBorracha;
private javax.swing.JButton btnCirculo;
private javax.swing.JButton btnDestino;
private javax.swing.JButton btnEnviar;
private javax.swing.JButton btnLinha;
private javax.swing.JButton btnNovo;
private javax.swing.JButton btnOrigem;
private javax.swing.JButton btnRetangulo;
private javax.swing.JButton btnSair;
private javax.swing.JButton btnSalvar;
private javax.swing.JButton btnSimular;
private javax.swing.JButton btnZoom;
private javax.swing.JComboBox cbalgoritmo;
private javax.swing.JCheckBox checkCaminho;
private javax.swing.JCheckBox checkSuave;
private javax.swing.JTextField edtPixelCent;
public planejatrajeto.PaintPanel espaco;
private planejatrajeto.fim fim1;
public planejatrajeto.inicio inicio1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JSeparator jSeparator1;
private javax.swing.JSeparator jSeparator2;
private javax.swing.JSeparator jSeparator3;
private javax.swing.JSeparator jSeparator4;
```

```
    private java.awt.Label lbCaminho;
    private java.awt.Label lbCelExplor;
    private java.awt.Label lbIteracoes;
    private java.awt.Label lbTempoexec;
    private java.awt.Label lblCotimizado;
    private javax.swing.JLabel posfinal;
    private javax.swing.JLabel posinicial;
    private planejatrajeto.StatusBar statusbar;
    private planejatrajeto.StatusBar statusbar1;
    private planejatrajeto.StatusBar statusbar2;
    private javax.swing.JSlider velocidade;
    // End of variables declaration
    public void actionPerformed(ActionEvent e) {
    }
}
```

PaintPanel.java

```
package planejatrajeto;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Shape;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.image.BufferedImage;
import javax.swing.JPanel;
public class PaintPanel extends JPanel implements MouseListener, MouseMotionListener{
    private final Color COLOR_BACKGROUND = Color.black; // Cor do fundo
    public static State _state = State.IDLE;
    private static figura _shape;
    private static Color _color;
    private static Point _start = null;
    private static Point _end = null;
    private static int _lastX = 0;
    private static int _lastY = 0;
    private static int[] _Xaxis;
    private static int[] _Yaxis;
    private static int _length = 0;
    public static BufferedImage _bufImage = null;
    public static int fx;
    public static int fy;
  public PaintPanel() {
        this.addMouseListener(this);
        this.addMouseMotionListener(this);
    }
  public static void Path(Point[] trajectory, int len, String kind) {
        String _kind = kind;
        _length = len;
        _Xaxis = new int[_length];
        _Yaxis = new int[_length];
        for(int i=0;i<=len-1;i++){
        _Xaxis[i]=trajectory[i].x;
        _Yaxis[i]=trajectory[i].y;
        }
        _state = State.UPDATE;
        if (_kind.equals("PathFound")) {
            Updates(_bufImage.createGraphics(), "BLUE");
        }
        if (_kind.equals("OptimizedPath")) {
            Updates(_bufImage.createGraphics(), "RED");
        }
    }
    public static void putPositions(int x, int y) {
```

```
    _lastX = x;
    _lastY = y;
}
    public static void clearPaint() {
    _bufImage = null;
}
public static  void setShape(figura shape) {
    _shape = shape;
    if ((_shape.equals(figura.FILHO))||(_shape.equals(figura.FILHO1)))
    {
       drawCurrentShape(_bufImage.createGraphics());
   }

}
public static void setColor(Color color) {
    _color = color;
}
@Override
public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    if (_bufImage == null) {
        int w = this.getWidth();
        int h = this.getHeight();
        _bufImage = (BufferedImage) this.createImage(w, h);
        Graphics2D gc = _bufImage.createGraphics();
        gc.setColor(COLOR_BACKGROUND);
        gc.fillRect(0, 0, w, h);
    }
    g2.drawImage(_bufImage, null, 0, 0);
     if ((_state == State.DRAGGING)) {
        drawCurrentShape(g2);
    }
}
public static void drawCurrentShape(Graphics2D g2) {
    Shape rec = new Rectangle(1,1);
    switch (_shape) {
        case OVAL:
            g2.setColor(Color.white);
            g2.fillOval(_start.x, _start.y, _end.x - _start.x,
                    _end.y - _start.y);
            break;
        case RECTANGLE:
            g2.setColor(Color.white);
            g2.fillRect(_start.x, _start.y, _end.x - _start.x,
                    _end.y - _start.y);
            break;
        case LINE:
            g2.setColor(Color.white);
            BasicStroke stroke = new BasicStroke(10.0f);
            g2.setStroke(stroke);
            g2.drawLine(_start.x, _start.y, _end.x, _end.y);
            break;
        case RECTROUND:
            g2.setColor(Color.white);
            g2.fillRoundRect(_start.x, _start.y, _end.x - _start.x,
                    _end.y - _start.y, 10, 10);
            break;
        case ERASE:
            g2.setColor(Color.black);
            g2.fillRect(_start.x, _start.y, _end.x - _start.x,
                    _end.y - _start.y);
            break;
        case START:
            g2.setColor(Color.black);
            g2.drawLine(_lastX, _lastY,_lastX, _lastY);
            g2.setColor(Color.green);
            g2.drawLine(_start.x,_start.y,_start.x,_start.y);
```

```
                break;
            case END:
                g2.setColor(Color.black);
                g2.drawLine(_lastX, _lastY, _lastX,_lastY);
                g2.setColor(Color.red);
                g2.drawLine(_start.x, _start.y,_start.x, _start.y);
                break;
            case FILHO:
               g2.setColor(Color.orange);
               g2.fillRect(fx,fy,1,1);
                break;
            case FILHO1:
               g2.setColor(Color.cyan);
               g2.fillRect(fx,fy,1,1);
                break;
        }
        g2.setColor(Color.black);
    }
    public static void Updates(Graphics2D g2, String ColorPath) {
        String _colorpath = ColorPath;
        if (_colorpath.equals("BLUE")) {
            g2.setColor(Color.blue);
            g2.drawPolyline(_Xaxis, _Yaxis, _length);
            g2.setColor(Color.green);
            g2.drawLine(_Xaxis[0],_Yaxis[0],_Xaxis[0],_Yaxis[0]);
            g2.setColor(Color.red);
            g2.drawLine(_Xaxis[_length-1],_Yaxis[_length-1],_Xaxis[_length-1],
                    _Yaxis[_length-1]);
        }
       if (_colorpath.equals("RED"))  {
            g2.setColor(Color.RED);
            g2.drawPolyline(_Xaxis, _Yaxis, _length);
            g2.setColor(Color.green);
            g2.drawLine(_Xaxis[0],_Yaxis[0],_Xaxis[0],_Yaxis[0]);
            g2.setColor(Color.red);
            g2.drawLine(_Xaxis[_length-1],_Yaxis[_length-1],_Xaxis[_length-1],
                    _Yaxis[_length-1]);
        }
    if (_colorpath.equals("GRAY"))  {
            g2.setColor(Color.ORANGE);
        for (int j = 0; j <= (_Xaxis.length - 1); j = j + 1) {
            g2.drawRect(_Xaxis[j],_Yaxis[j],1, 1);
  }
 }
}
    public void mousePressed(MouseEvent e) {
        _state = State.DRAGGING;
        _start = e.getPoint();
        _end = _start;
    }
    public void mouseDragged(MouseEvent e) {
        _state = State.DRAGGING;
        _end = e.getPoint();
        this.repaint();
    }
    public void mouseReleased(MouseEvent e) {
        _end = e. getPoint();
        if ((_state == State.DRAGGING)) {
            _state = State.IDLE;
            drawCurrentShape(_bufImage.createGraphics());
            this.repaint();
        }
    }
    public void mouseClicked(MouseEvent e) {    }
    public void mouseEntered(MouseEvent e) {    }
    public void mouseExited(MouseEvent e) {    }
    public void mouseMoved(MouseEvent e) {          }
```

```
}

SerialComm.java
package planejatrajeto;
import gnu.io.CommPortIdentifier;
import  java.util.Enumeration;
public class SerialCom {
    protected String[] portas;
    protected Enumeration listaDePortas;
     public SerialCom(){
         listaDePortas = CommPortIdentifier.getPortIdentifiers();
       }
     public String[] ObterPortas(){
          return portas;
       }
protected void ListarPortas() {
    int i = 0;
    portas = new String[10];
    while (listaDePortas.hasMoreElements()) {
        CommPortIdentifier ips =
          (CommPortIdentifier) listaDePortas.nextElement();
        portas[i] = ips.getName();
        i++;
    }
}
public boolean PortaExiste(String COMp) {
    String temp;
    boolean e = false;
    while (listaDePortas.hasMoreElements()) {
        CommPortIdentifier ips =
          (CommPortIdentifier) listaDePortas.nextElement();
        temp = ips.getName();
        if (temp.equals(COMp) == true) {
            e = true;
        }
    }
    return e;
}
}

SerialCommLeitura.java

package planejatrajeto;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;
import gnu.io.SerialPortEvent;
import gnu.io.SerialPortEventListener;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
public class SerialComLeitura implements Runnable, SerialPortEventListener {
       public String message;
       public String Dadoslidos;
       public int nodeBytes;
       private int baudrate;
       private int timeout;
       private CommPortIdentifier cp;
       private SerialPort porta;
       private OutputStream saida;
       private InputStream entrada;
       private Thread threadLeitura;
       private boolean  IDPortaOK;
       private boolean  PortaOK;
       private boolean  Leitura;
       private boolean  Escrita;
       private String  Porta;
       protected String peso;
```

```
public SerialComLeitura(String p,int b,int t)
{
this.Porta=p;
this.baudrate=b;
this.timeout=t;
}
public  void  HabilitaEscrita()
{
Escrita=true;
Leitura=false;
}
public  void  HabilitaLeitura()
{
Escrita=false;
Leitura=true;
}
public  void  ObterIdDaPorta()
{
    try {
        cp=CommPortIdentifier.getPortIdentifier(porta);
        if (cp==null)
        {
            message="Erro na Porta";
            IDPortaOK=false;
            System.exit(1);
        }
        IDPortaOK=true;
    } catch (Exception e) {
        message="Erro obtendo ID da Porta:"+e;
        IDPortaOK=false;
        System.exit(1);
    }
}
public  void  AbrirPorta(){
try {
      porta=(SerialPort)cp.open("SerialComLeitura",timeout);
      PortaOK=true;
      porta.setSerialPortParams(baudrate,porta.DATABITS_8,porta.STOPBITS_1,
            porta.PARITY_NONE);
      porta.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
      message="";
      message="PortaOK"+"\n";
   } catch (Exception e) {
      PortaOK=false;
      message="";
      message="Erro abrindo comunicação:"+e+"\n";
   }
}
public void LerDados(){
 if(Escrita==false){
    try {
     entrada=porta.getInputStream();
    } catch (Exception e) {
        message="Erro de Stream:"+e;
        System.exit(1);
    }
    try {
     porta.addEventListener(this);
    } catch (Exception e) {
    message="Erro de listener:"+e;
    System.exit(1);
    }
    porta.notifyOnDataAvailable(true);
    try {
    threadLeitura=new Thread(this);
    threadLeitura.start();
    } catch (Exception e) {
```

```
            System.out.println("Erro de Tread"+ e);
          }
          }
     }
public void EnviarUmaString(String msg) {
    if (Escrita == true) {
        try {
            saida = porta.getOutputStream();
            message="";
            message="Fluxo OK"+"\n";
        } catch (Exception e) {
            System.out.println("Erro.STATUS:" + e);
        }
        try {
            message="";
            message="Enviando um byte para " + Porta+"\n"+"Enviando: " + msg;
            saida.write(msg.getBytes());
            Thread.sleep(100);
            saida.flush();
        } catch (Exception e) {
            message="";
            message="Houve um erro durante o envio.";
            message="";
            message="Erro.STATUS:" + e;
            System.exit(1);
        }
    }
    else{
    System.exit(1);
    }
}
public void run(){
    try {
      Thread.sleep(5);
    } catch (Exception e) {
      message="Erro de Thread"+e;
    }
}
public void serialEvent(SerialPortEvent ev){
    StringBuffer bufferLeitura = new StringBuffer();
    int novoDado = 0;
    switch (ev.getEventType()) {
    case SerialPortEvent.BI:
    case SerialPortEvent.OE:
    case SerialPortEvent.FE:
    case SerialPortEvent.PE:
    case SerialPortEvent.CD:
    case SerialPortEvent.CTS:
    case SerialPortEvent.DSR:
    case SerialPortEvent.RI:
    case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
    break;
    case SerialPortEvent.DATA_AVAILABLE:
     while (novoDado != -1) {
      try {
       novoDado = entrada.read();
       if (novoDado == -1) {
         break;
       }
      if ('\r' == (char) novoDado) {
       bufferLeitura.append('\n');
      }
      else {
      bufferLeitura.append((char) novoDado);
       }
     } catch (IOException ioe) {
       System.out.println("Erro de leitura serial:" + ioe);
```

```
    }
}
break;
}
}
public void FecharCom() {
try {
   porta.close();
 } catch (Exception e) {
 message= "Erro fechando porta: " + e;
  System.exit(0);
 }
}
public String obterPorta() {
    return Porta;
}
public int obterBaudrate() {
    return baudrate;
}
}
```

State.java

```
package planejatrajeto;

public enum State{
   IDLE, DRAGGING,UPDATE }
```

TransformingCanvas.java

```
package planejatrajeto;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;
import javax.swing.JComponent;
public class TransformingCanvas extends JComponent implements MouseListener,
                                MouseMotionListener,MouseWheelListener {
                        private double translateX;
                        private double translateY;
                        private double scale;
                        private int lastOffsetX;
                        private int lastOffsetY;
                BufferedImage image;
                        public TransformingCanvas(BufferedImage image) {
                                this.image = image;
                    translateX = 0;
                                translateY = 0;
                                scale = 1;
                                setOpaque(true);
                                setDoubleBuffered(true);
                        }
                        @Override public void paint(Graphics g) {
                                AffineTransform tx = new AffineTransform();
                                tx.translate(translateX, translateY);
                                tx.scale(scale, scale);
                                Graphics2D ourGraphics = (Graphics2D) g;
                                ourGraphics.setColor(Color.WHITE);
                                ourGraphics.fillRect(0, 0, getWidth(), getHeight());

                                ourGraphics.setTransform(tx);
                                ourGraphics.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                                RenderingHints.VALUE_ANTIALIAS_ON);
                                ourGraphics.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
                                                RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
                    ourGraphics.drawRenderedImage(image, tx);
                        }
        public void mousePressed(MouseEvent e) {
```

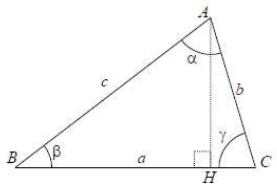
```
                        lastOffsetX = e.getX();
                        lastOffsetY = e.getY();
        }
        public void mouseDragged(MouseEvent e) {
                        int newX = e.getX() - lastOffsetX;
                        int newY = e.getY() - lastOffsetY;
                        lastOffsetX += newX;
                        lastOffsetY += newY;
                        this.translateX += newX;
                        this.translateY += newY;
                        this.repaint();
        }
        public void mouseClicked(MouseEvent e) {}
        public void mouseReleased(MouseEvent e) {}
        public void mouseEntered(MouseEvent e) {}
        public void mouseExited(MouseEvent e) {}
        public void mouseMoved(MouseEvent e) {}
                public void mouseWheelMoved(MouseWheelEvent e) {
                        if(e.getScrollType() == MouseWheelEvent.WHEEL_UNIT_SCROLL) {
                                this.scale += (.1 * e.getWheelRotation());
                                this.scale = Math.max(0.00001, this.scale);
                                this.repaint();
                        }
        }
}
```

11.2. Apêndice 2: Trigonometria

Lei dos co-senos

Num triangulo qualquer ABC, traca-se uma a reta que, a partir do vertice A, encontra o lado BC em angulo reto (perpendicular a BC), como mostra a figura abaixo. No triangulo retangulo ABH, aplica-se o teorema de Pitagoras, obtendo-se:

$$c^2 = BH^2 + AH^2$$



Da mesma forma, o triangulo retangulo AHC fornece:

$$b^2 = HC^2 + AH^2$$

Isolando-se o lado AH das expressoes acima e igualando-as tem-se que

$$c^2 = b^2 + BH^2 - HC^2$$

Porem, lembrando que HC = b cosγ e que AH = a – HC, substituindo-se estes valores na ultima resulta que

$$c^2 = b^2 + a^2 - 2ab\cos\gamma\,.$$

A lei dos co-senos pode entao ser definida como “num triangulo qualquer, o quadrado de um dos lados e igual a soma dos quadrados dos demais, subtraido do duplo produto destes lados pelo co-seno do angulo entre eles”. Uma vez que nao foi estabelecida nenhuma condicao sobre um dos lados, tem-se igualmente que:

$$a^2 = b^2 + c^2 - 2bc\cos\alpha\,.$$

e

$$b^2 = a^2 + c^2 - 2ac\cos\beta\,.$$

Fórmulas

$$\cos(A+B) = \cos A\cos B - \operatorname{sen} A\operatorname{sen} B \qquad \cos(90-\theta) = \operatorname{sen}\theta$$
$$\operatorname{sen}(A+B) = \operatorname{sen} A\cos B + \cos A\operatorname{sen} B \qquad \operatorname{sen}(90-\theta) = \cos\theta$$

$$\tan(A-B) = \frac{\tan A - \tan B}{1+\tan A\tan B} \qquad \tan(90-\theta) = \cot\theta = \frac{1}{\tan\theta}$$

$$\Delta w = \sum_{i=1}^{n}\left|\frac{\partial w}{\partial\theta_i}\right|\Delta\theta_i = \left|\frac{\partial w}{\partial\theta_1}\right|\Delta\theta_1 + \left|\frac{\partial w}{\partial\theta_2}\right|\Delta\theta_2 + \left|\frac{\partial w}{\partial\theta_3}\right|\Delta\theta_3 + \cdots, \qquad \cos^2\theta + \operatorname{sen}^2\theta = 1$$