UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

STÉFANO TERCI GASPERAZZO

UM ALGORITMO PSO HÍBRIDO PARA PLANEJAMENTO DE CAMINHOS EM NAVEGAÇÃO DE VEÍCULOS UTILIZANDO A*

VITÓRIA–ES
2014

STÉFANO TERCI GASPERAZZO

UM ALGORITMO PSO HÍBRIDO PARA PLANEJAMENTO DE CAMINHOS EM NAVEGAÇÃO DE VEÍCULOS UTILIZANDO A*

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

VITÓRIA–ES
2014

STÉFANO TERCI GASPERAZZO

UM ALGORITMO PSO HÍBRIDO PARA PLANEJAMENTO DE CAMINHOS EM NAVEGAÇÃO DE VEÍCULOS UTILIZANDO A*

Dissertação submetida ao programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisição parcial para a obtenção do Grau de Mestre em Informática.

COMISSÃO EXAMINADORA

Profa. Dra. Maria Claudia Silva Boeres
Universidade Federal do Espírito Santo
Orientadora

Profa. Dra. Maria Cristina Rangel
Universidade Federal do Espírito Santo
Co-orientadora

Profa. Dra. Claudine Santos Badue Gonçalves
Universidade Federal do Espírito Santo

Profa. Dra. Luciana Salete Buriol
Universidade Federal do Rio Grande do Sul

Dedicatória

Dedico este trabalho a todos familiares, amigos e professores que contribuíram de alguma forma para que sua conclusão fosse possível.

Agradecimentos

Agradeço primeiramente à minha família, pelo exemplo de honestidade, responsabilidade e trabalho, que me fizeram ser quem sou.

À minha noiva Tassiana pelo suporte nos momentos mais difíceis, pelo carinho, paciência e dedicação.

Também agradeço a minha orientadora Maria Claudia Silva Boeres e co-orientadora Maria Cristina Rangel pelos ensinamentos nesta jornada e por tornarem o desenvolvimento da pesquisa mais agradável. Aos demais professores do Programa de Pós-Graduação em Informática pela importância na minha formação acadêmica.

Aos meus colegas de trabalho da Coordenadoria de Tecnologia da Informação do Instituto Federal do Espírito Santo Campus Serra e demais colegas do Laboratório de Otimização que estiveram ao meu lado durante esse etapa da minha vida.

Resumo

Utilizar robôs autônomos capazes de planejar o seu caminho é um desafio que atrai vários pesquisadores na área de navegação de robôs. Neste contexto, este trabalho tem como objetivo implementar um algoritmo PSO híbrido para o planejamento de caminhos em ambientes estáticos para veículos holonômicos e não holonômicos. O algoritmo proposto possui duas fases: a primeira utiliza o algoritmo A* para encontrar uma trajetória inicial viável que o algoritmo PSO otimiza na segunda fase. Por fim, uma fase de pós planejamento pode ser aplicada no caminho a fim de adaptá-lo às restrições cinemáticas do veículo não holonômico. O modelo *Ackerman* foi considerado para os experimentos. O ambiente de simulação de robótica CARMEN (*Carnegie Mellon Robot Navigation Toolkit*) foi utilizado para realização de todos os experimentos computacionais considerando cinco instâncias de mapas geradas artificialmente com obstáculos. O desempenho do algoritmo desenvolvido, A*PSO, foi comparado com os algoritmos A*, PSO convencional e A* Estado Híbrido. A análise dos resultados indicou que o algoritmo A*PSO híbrido desenvolvido superou em qualidade de solução o PSO convencional. Apesar de ter encontrado melhores soluções em 40% das instâncias quando comparado com o A*, o A*PSO apresentou trajetórias com menos pontos de guinada. Investigando os resultados obtidos para o modelo não holonômico, o A*PSO obteve caminhos maiores entretanto mais suaves e seguros.

Palavras Chave: Veículos autônomos, A*, PSO, planejamento de caminhos.

Abstract

Autonomous robots with the ability of planning their own way is a challenge that attracts many researchers in the area of robot navigation. In this context, this work aims to implement a hybrid PSO algorithm for planning paths in static environments for holonomic and non-holonomic vehicles. The proposed algorithm has two phases: the first uses A* algorithm to generates an initial and feasible trajectory which is optimized by the PSO algorithm in the second stage. Finally a post path planning phase can be applied in order to adapt it to non-holonomic vehicle kinematic constraints. The Ackerman model has been considered for the experiments. The Carnegie Mellon Robot Navigation Toolkit (CARMEN) was used to perform the computational experiments considering five instances of maps artificially generated with obstacles. The performance of the A*PSO algorithm was compared with A*, PSO and A*-Hybrid State. The results of the dynamic instances were not compared with other algorithms. The computational results indicates that the algorithm A*PSO outperformes the PSO algorithm. With respect to the algorithm A*, the A*PSO achieved better solutions for 40% of the tested instances, but all of them, with less waypoints. For non-holonomic instances, the A*PSO obtained longer paths, however smoother and safer.

Keywords: Autonomous Vehicles, A*, PSO, path planning.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

A crescente utilização de robôs em situações complexas, tais como desarmamento de bombas, manutenção em ambientes radioativos e exploração de planetas mostram a necessidade de aperfeiçoamento da maneira como os robôs entendem o ambiente ao seu redor a fim de estender sua aplicabilidade (RAJA; PUGAZHENTH, 2012). Com os avanços tecnológicos no setor automotivo e na robótica, os veículos autônomos - carros guiados por computadores sem intervenção humana - ganham cada vez mais destaque, uma vez que sua aplicação tende a resolver dois grandes problemas atuais: congestionamentos e acidentes de trânsito (SEONGMOON; LEWIS; WHITE, 2005).

Realizando tarefas rotineiras e cansativas para seres humanos, os veículos autônomos estão sendo mais utilizados em ambientes urbanos. A empresa *Google* recentemente obteve aprovação nos Estados Unidos para circular com seus carros autônomos em determinadas cidades daquele país. Competições entre universidades e empresas privadas são anualmente realizadas, sendo uma das mais famosas realizadas pela DARPA (*Defense Advanced Research Projects Agency*). A utilização de veículos controlados por computadores não só otimizaria o trânsito mas também evitaria acidentes, uma vez que computadores não estão submetidos ao estresse, cansaço, fadiga, dentre outros motivadores de acidentes (BUEHLER; IAGNEMMA; SINGH, 2007).

Usualmente um esquema de controle de um veículo autônomo é subdividido em blocos de ações a fim de facilitar sua implementação. Estes blocos, apresentados na Figura 1, compreendem toda a etapa de planejamento e execução dos movimentos de um veículo autônomo. Na caixa *percepção do ambiente*, o sensoriamento é realizado coletando informações cruas do mundo real para então interpretá-las com objetivo de modelar o ambiente em um mapa local. A próxima etapa consiste em definir a localização do veículo identificando sua posição atual e construir o mapa de navegação global. Inserindo comandos de entrada na etapa de *Planejamento de Caminho* informando o destino até onde o veículo deverá se mover é possível construir um caminho, livre de colisões através do sensoriamento cognitivo do ambiente. Na caixa *controle de movimentos* é realizada uma simulação da execução do caminho para determinar os comandos de movimentos que serão enviados aos atuadores para que o veículo possa executar de fato o movimento e se locomover da posição atual até o destino guiando o veículo por todo o caminho estabelecido (SIEGWART; NOURBAKHSH, 2004).

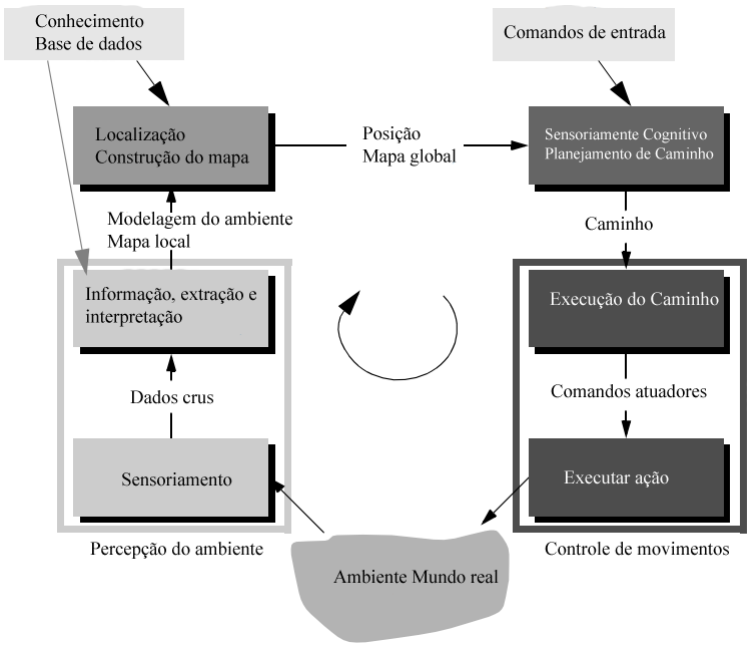


Figura 1: Diagrama de controle de veículos autônomos (fonte: (SIEGWART; NOURBAKHSH, 2004)).

O planejamento de caminhos é uma importante etapa do processo de navegação de um veículo autônomo que consiste em encontrar uma trajetória livre de colisão de um ponto inicial a um destino satisfazendo algum critério, tal como, menor consumo de energia, menor caminho, menor tempo, dentre outros. A maioria das técnicas que tratam esse problema adota o critério de menor caminho (RAJA; PUGAZHENTH, 2012).

Os algoritmos para planejamento de caminhos são classificados conforme dois aspectos: completude (exato ou heurístico) e escopo (global ou local). Em ambientes estáticos, isto é, quando o ponto inicial, o destino e os obstáculos já são conhecidos e não sofrem alterações durante o processo de navegação, configura-se um planejamento global ou *offline*. Em ambientes dinâmicos, quando os obstáculos e seus movimentos não são conhecidos, configura-se um planejamento local ou *online*. Os algoritmos exatos apresentam uma solução ótima (caminho mínimo) quando ela existe ou provam que não existe uma solução e costumam ser computacionalmente mais caros. A introdução de algoritmos heurísticos visa encontrar um caminho que seja bom, em um tempo de execução reduzido, sem a garantia da otimalidade (HWANG; AHUJA, 1992).

As abordagens exatas se tornaram inviáveis em virtude da complexidade dos ambientes onde os veículos autônomos estão inseridos atualmente pois a representação dos obstáculos de diversas formas consome um elevado tempo computacional, além disso, encontrar caminhos livres de colisão nesses ambientes é um problema altamente combinatório. Os dois problemas mencionados dificultam a aplicação no planejamento em tempo real de um veículo (ASHIRU; CZARNECKI; ROUTEN, 1996).

A utilização de heurísticas e meta-heurísticas para resolver o problema de planejamento de caminhos nos escopos dinâmico (local) e estático (global) cresceu ao longo dos últimos anos (MASEHIAN; SEDIGHIZADEH, 2007). Buscando uma maior eficiência, tais métodos podem ser combinados com algoritmos clássicos da literatura.

Uma abordagem clássica da literatura para planejamento de caminhos que utiliza técnicas heurísticas é o algoritmo de busca A* (HART; NILSSON; RAPHAEL, 1968). Após a fase de construção do mapa a partir de informações coletadas do mundo real, é possível aplicar um método que codifique o mapa em um grafo para que o algoritmo seja executado. O algoritmo explora os vértices do grafo em busca do menor caminho entre os pontos inicial e final utilizando informações do próprio ambiente para calcular o próximo vértice a ser explorado.

Tratando o problema de planejamento de caminhos como um problema de otimização é possível formular o problema de forma a utilizar meta-heurísticas objetivando minimizar algum critério. A meta-heurística Enxame de Partículas (*Particle Swarm Optimization* - PSO) foi introduzida por Kennedy e Eberhart (KENNEDY; EBERHART, 1995) e se mostrou capaz de resolver, de maneira eficiente, vários tipos de problemas complexos (ALBA, 2009). O PSO é um algoritmo baseado em população e a perturbação utilizada na tentativa de melhorar a solução do problema é baseada na coletividade entre os indivíduos da população.

Neste trabalho o planejamento de caminhos em ambientes estáticos é explorado, bem como o algoritmo A* e a meta-heurística PSO combinados para resolver o problema de caminhos utilizando-se de conhecimentos do ambiente. O desempenho do planejador foi comparado com os algoritmos A* (HART; NILSSON; RAPHAEL, 1968), A* Estado Híbrido (DOLGOV et al., 2010) e PSO proposto por (DUNWEI; LI; MING, 2009).

1.1 Motivação

No Laboratório de Computação de Alto Desempenho (*LCAD*) da Universidade Federal do Espírito Santo existe um grupo de pesquisa fortemente engajado em projetos com veículos autônomos, que desenvolve o veículo IARA (*Intelligent Autonomous Robotic Automobile*), disponível em `http://www.lcad.inf.ufes.br`. A motivação deste trabalho é investigar a utilização de meta-heurísticas combinadas com algoritmos clássicos para futuramente integrar esses métodos ao planejamento de caminhos utilizado no projeto IARA.

1.2 Objetivo

O objetivo deste trabalho é desenvolver um algoritmo capaz de planejar caminhos em ambientes estáticos utilizando modelos de veículos que possuem restrições cinemáticas (não holonômicos) e também para os modelos que não possuem tais restrições (holonômicos). Para alcançar este objetivo, desenvolveu-se um algoritmo híbrido utilizando o algoritmo A* como gerador de soluções iniciais para o algoritmo PSO proposto em (DUNWEI; LI; MING, 2009) sendo implementado e executado no ambiente de robótica *Carnegie Mellon Robot Navigation Toolkit* (*CARMEN*).

1.3 Contribuições

Para que o algoritmo desenvolvido neste trabalho - denominado A*PSO, seja aplicado, o mundo real é discretizado em um grafo onde os nós representam pontos no plano cartesiano (x,y). O A*PSO possui duas fases distintas, onde a primeira é a geração de um caminho inicial através da aplicação do A* no grafo citado anteriormente. A partir deste caminho, o A*PSO executa a segunda fase, onde o algoritmo PSO tenta otimizar o caminho. Vale ressaltar que a solução advinda da primeira fase é um caminho no grafo onde os nós são pontos (x,y) e a segunda fase é realizada sobre o plano cartesiano, independente do grafo. Simulações foram realizadas em cinco instâncias e o A*PSO estabeleceu caminhos com uma quantidade menor de pontos de guinada, tornando a trajetória mais suave com relação ao A*.

1.4 Organização do texto

Este trabalho apresenta, além dessa introdução, mais seis capítulos. No capítulo 2 são destacados os algoritmos da literatura para planejamento de caminhos. No capítulo 3 define-se o problema de planejamento de caminhos tratado neste trabalho e no capítulo 4 são discutidos os algoritmos utilizados para o desenvolvimento do A*PSO, descrito no capítulo 5. Os resultados computacionais e discussões são apresentados no capítulo 6 e por fim, no capítulo 7, as conclusões e as propostas de trabalhos futuros.

2 *Trabalhos correlatos*

Em 1960, no instituto de pesquisas de *Stanford* (*Stanford Research Institute*), foi concebido o primeiro veículo controlado por um programa de computador chamado *Shakey*. Durante quase uma década os inventores de *Shakey* se depararam com os mesmos problemas encontrados atualmente no contexto de veículos autônomos tais como, desvio de obstáculos, reconhecimento de objetos e mapeamento do ambiente (TOMATIS, 2001).

Com os sensores e sua crescente utilização, os programadores de veículos autônomos puderam dispor de muitas informações do ambiente para criar algoritmos capazes de solucionar com maior eficiência os problemas encontrados. No entanto, se faz necessária uma atenção especial às incertezas e erros gerados pelos mesmos. As principais tecnologias de sensores estereoscópicos utilizados atualmente incluem sonar, laser, câmeras, dentre outros (LEONARD, 2007).

Em 1979, Lozano-Perez e Wesley, (LOZANO-PÉREZ; WESLEY, 1979), definiram o importante conceito de *C-Space* onde a dimensão do corpo rígido do veículo é modelado como um ponto no plano e dois conjuntos são criados: *C-Free*, onde todas as configurações válidas em que o veículo pode assumir são mapeadas e *C-Obstacles* onde são mapeadas as posições inviáveis. Esta simplificação está sendo utilizada até os dias atuais e renovou o interesse dos pesquisadores nesta área (RAJA; PUGAZHENTH, 2012).

Em meio aos trabalhos desenvolvidos para veículos autônomos destacam-se as competições realizadas pela agência americana *DARPA* (*Defense Advanced Research Projects Agency*). Ao todo a *DARPA* organizou três competições entre 2004 e 2007 a fim de aprimorar o desenvolvimento e pesquisa das tecnologias empregadas nos veículos autônomos. Várias inovações e soluções para problemas puderam ser compartilhados entre os participantes e os resultados provaram que seria possível a criação de veículos completamente autônomos ou com inteligencia embarcada.

Em 2004 foi realizada a primeira *DARPA Grand Challenge* onde nenhum veículo conseguiu completar os 204 km do circuito. No ano seguinte, em 2005, cinco veículos conseguiram completar o percurso e o vencedor foi o veículo *Stanley* da *Stanford University*. O veículo vencedor não utilizou algoritmo para planejar caminhos uma vez que o mapa fornecido já continha o caminho a ser seguido pelo veículo, sendo necessário

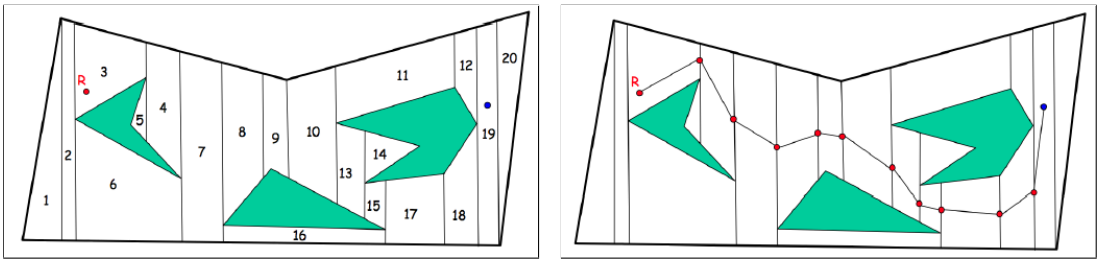
somente algoritmos para controlar a velocidade e evitar colisões (semelhante a mudança de faixas em uma via para carros). Em 2007 ocorreu a última competição realizada pela *DARPA* e o vencedor foi o veículo *Boss* desenvolvido pela *Carnegie Mellon University* utilizando o algoritmo D* (KOENIG; LIKHACHEV, 2002). O segundo colocado, o veículo *Junior* utilizou o algoritmo A* Estado Híbrido (DOLGOV et al., 2010). *Odin* foi o terceiro colocado utilizando também uma variante do algoritmo A*. Já o quarto lugar, o veículo *Talos*, utilizou o algoritmo RRT (*Rapidly-exploring Random Tree*) comumente utilizado nos dias atuais por diversos planejadores. Todos esses algoritmos estão detalhados na coletânea (BUEHLER; IAGNEMMA; SINGH, 2007).

2.1 Abordagens Clássicas

As abordagens clássicas recebem esta nomenclatura pois agrupam os principais algoritmos desenvolvidos na literatura para solucionar o problema de planejamento de caminhos em ambientes estáticos e dinâmicos. Nestas abordagens, geralmente os algoritmos possuem uma fase de representação do ambiente em seguida, um caminho é construído nesta estrutura. Podemos destacar: Decomposição Celular, Mapas de Caminhos e Campos Potenciais (MASEHIAN; SEDIGHIZADEH, 2007).

O método de decomposição celular consiste em dividir o espaço livre *C-Free* em regiões menores denominadas células. Um grafo é formado a partir destas regiões cujos vértices correspondem às células e as arestas são definidas pelas adjacências entre as células. Estes métodos são classificados como exatos quando a união das células formadas corresponde exatamente ao *C-Free* e como aproximados quando as células definidas estão contidas no *C-Free*.

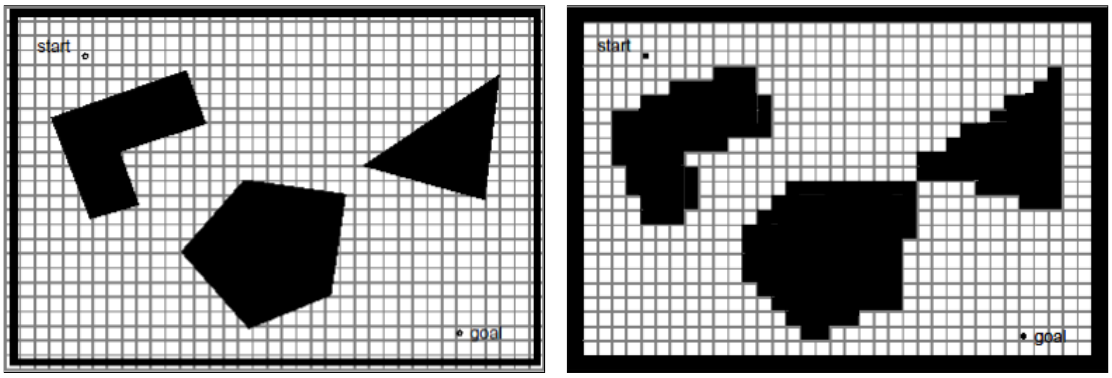
Um método exato de decomposição celular bastante difundido é o trapezoidal (LATOMBE, 1991). Sendo o mapa e os obstáculos definidos como polígonos, o método traça um segmento de reta vertical perpendicular ao eixo X sempre que um vértice de um polígono é reconhecido identificando as células como ilustra a Figura 2(a). Podemos observar nesta figura que o espaço de busca foi dividido em 20 células. Após definida a sequência, um caminho é traçado do ponto inicial *R* (célula 3) até o ponto de destino (célula 19). Inicialmente o ponto inicial *R* é conectado ao ponto médio do segmento de reta correspondente à célula onde está localizado. O algoritmo continua conectando os pontos médios de cada segmento de reta das células adjacentes até chegar ao destino. Finalmente o ponto destino é conectado ao ponto médio da célula correspondente formando o caminho apresentado na Figura 2(b). Como é um método exato de divisão do mapa em células, poderão surgir *waypoints* (pontos que compõem o caminho) desnecessários na trajetória encontrada. Pode-se verificar que é viável conectar o segundo ponto ao oitavo sem interceptar obstáculos (DOH; KIM; CHUNG, 2007).



(a) Decomposição Trapezoidal do espaço de busca. (b) Caminho encontrado no mapa decomposto.

Figura 2: Decomposição celular exata.

Na decomposição celular aproximada (LOZANO-PÉREZ, 1983; ZHU; LATOMBE, 1991; LIKHACHEV et al., 2005; HACHOUR, 2008), o mapa é ladrilhado em células de mesma dimensão formando uma malha, conforme Figura 3(a), denominada *Grid*. Cada célula possui um parâmetro indicando se ela é viável ou não, ou seja, pertence ao conjunto *C-Free* ou *C-Obstacle*. A Figura 3(b) apresenta as células pertencentes ao *C-Free* em branco e em preto as células do conjunto *C-Obstacles*. A principal dificuldade desta representação é estabelecer a dimensão das células. Se pequena, apresenta trajetórias mais precisas porém existe um gasto computacional maior já que o conjunto *C-Free* passa a ter mais elementos. Se grande então a busca retornará uma solução em tempo computacional reduzido todavia apresenta, geralmente, uma solução com qualidade inferior em relação a comprimento e suavidade (GUANZHENG; HUAN; AARON, 2007).

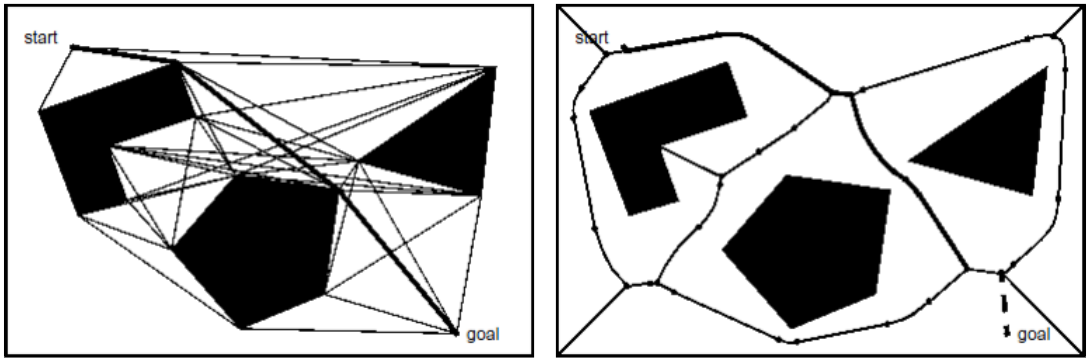


(a) Mapa ladrilhado. (b) Ajuste dos identificadores de cada célula: cor preta, caso obstáculo; cor branca: caso contrário.

Figura 3: Decomposição celular aproximada.

A técnica de Mapas de Caminhos representam o mundo em um mapa onde vários caminhos são construídos conectando os pontos inicial e final de acordo com algum critério. Os principais algoritmos para criação destra representação são o grafo de visibilidade (Figura 4(a)) e o diagrama de Voronoi (Figura 4(b)). No grafo de visibilidade a construção do conjunto *C-Free* consiste em percorrer cada vértice dos obstáculos do mapa e criar uma aresta para todos os vértices visíveis a partir dele. Este método quase sempre encontra o caminho mínimo (SIEGWART; NOURBAKHSH, 2004), mas um dos seus principais

problemas é que o caminho encontrado encosta nos obstáculos. O diagrama de Voronoi resolveu este problema, pois tende a criar caminhos mais distante dos obstáculos. O diagrama de Voronoi é gerado a partir da equidistância entre dois obstáculos e/ou as bordas do mapa (BORTOFF, 2000).



(a) Grafo de visibilidade. (b) Grafo de voronoi.

Figura 4: Mapa de caminhos

Tanto para as representações Mapa de Caminhos quanto para Decomposição celular apresentadas é necessária a criação de um grafo a partir do conjunto *C-Free* onde os vértices são posições válidas (não obstáculos) e as arestas representam caminhos viáveis (livre de colisão) entre pares de vértices. Após a definição do grafo e suas arestas valoradas é possível executar algoritmos de busca em grafo, como o algoritmo de Dijkstra, para estabelecer um caminho entre os pontos inicial e final caminhando pelo grafo vértice à vértice. Apesar destas técnicas funcionarem bem em mapas estáticos, quando aplicadas em mapas dinâmicos não funcionam tão bem devido ao custo computacional de atualizar o grafo toda vez que ocorre uma mudança no ambiente (FERGUSON; M.; STENTZ, 2005).

O algoritmo de Dijkstra, clássico na literatura de algoritmos de caminhos mínimos em grafos, foi publicado em 1959 por Elbert Dijkstra. Este algoritmo é ótimo e consiste em, dado um vértice inicial, calcular o caminho mínimo deste para todos os outros vértices. Em (HWANIL; BYUNGHEE; KABIL, 2008) o algoritmo de Dijkstra gera a solução ótima de caminho mínimo no grafo obtido a partir de uma representação do ambiente no formato *MAKLINK* (HABIB; ASAMA, 1991). Nesta modelagem os vértices do grafo são definidos a partir do ponto médio dos segmentos de reta que conectam dois vértices dos obstáculos onde não haja colisão. As arestas deste grafo conectam os vértices do grafo caso não exista colisão com obstáculos. Os autores utilizaram o algoritmo de Dijkstra para encontrar o caminho mínimo neste grafo, porém a solução encontrada não corresponde ao menor caminho no ambiente real entre origem e destino. Por esse motivo o PSO é usado a partir desta solução visando ajustes para sua melhoria. Os resultados após a otimização realizada pelo *PSO* foram melhores em todos os experimentos. Em (VLASSIS et al., 1996) os autores utilizam o algoritmo de Dijkstra modificado, inserindo uma heurística que faz o

algoritmo escolher um caminho mais suave e mais afastado dos obstáculos. Os resultados demonstram que o algoritmo foi capaz de encontrar o caminho mínimo respeitando uma certa distância dos obstáculos e a curvatura da trajetória.

O método Campo Potencial desenvolvido parte do princípio das forças de atração e repulsão exercidas pelo ponto final e obstáculos respectivamente (KHATIB, 1985). O método trata o veículo como uma partícula inserida dentro do campo potencial artificial gerado por essas forças onde o veículo, por possuir força de repulsão, é atraído pela força de atração do ponto final e desvia de obstáculos por também possuírem força de repulsão. Este método é bastante utilizado em mapas dinâmicos pois não necessita de conhecimento prévio do ambiente para executar o planejamento já que os obstáculos e as forças que atuam sobre eles são calculadas a todo instante em tempo real. A Figura 5 ilustra um caminho encontrado pelo método Campo Potencial onde o obstáculo e veículo possuem forças repulsivas e o ponto final força atrativa. No entanto o veículo pode cair em "armadilhas" (mínimos locais) formadas em regiões onde as forças do campo potencial artificial se anulem, por exemplo: corredores, quinas de obstáculos, obstáculos com formato de **U**, dentre outras (LATOMBE, 1991). Procura-se minimizar essas regiões com a introdução de ruído e perturbações na função que gera os campos potências próximos aos obstáculos na tentativa de eliminar regiões de anulação das forças. Além disso, a introdução de memória ao algoritmo evita que o veículo entre novamente nestas regiões. O sucesso deste método está na simplicidade de sua implementação e principalmente, na estratégia utilizada para geração do campo potencial artificial (BALCH, 1993).

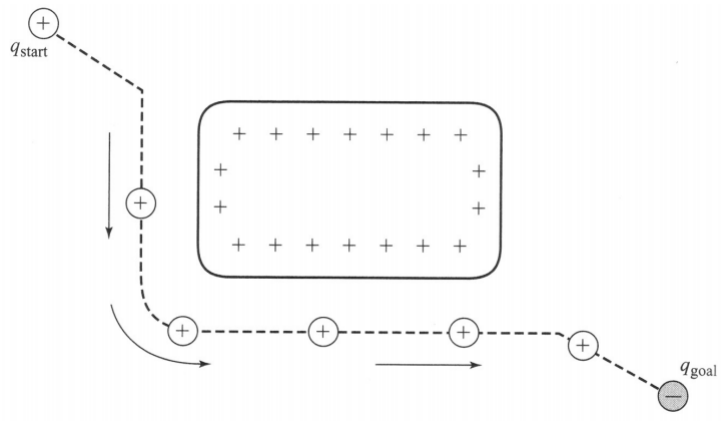


Figura 5: Esquema básico de um campo potencial.

2.2 Abordagens Heurísticas

Em virtude da complexidade do ambiente, muitas vezes as abordagens clássicas se tornam custosas para resolver o problema de planejamento de caminhos e em certas ocasiões, não conseguem escapar do mínimo local (SUGIHARA; SMITH, 1997). Para resolver

o problema, vários autores utilizaram heurísticas e meta-heurísticas a fim de encontrar uma boa solução em um tempo computacional razoável em detrimento da otimalidade. Dentre as principais meta-heurísticas utilizadas, destacam-se o Algoritmo Genético (AG) e o *Particle Swarm Optmization* (PSO) conforme observado em *survey* da literatura (MASEHIAN; SEDIGHIZADEH, 2007).

O algoritmo A* é bastante utilizado em planejamento de caminhos onde o mapa é representado por grafo. Assim como no algoritmo de Dijkstra, uma árvore de exploração dos vértices é construída a partir do vértice inicial, mas possui mecanismos de poda que permitem a exploração de menos vértices para a obtenção do caminho mínimo. O algoritmo A* utiliza a função $f(s) = g(s) + h(s)$ para guiar a exploração dos vértices do grafo a partir do vértice atual *s*. A função *g(s)*, consiste em avaliar o custo real do vértice de partida até o atual *s* e $h(s)$ é uma heurística que estima o custo do vértice atual *s* até o vértice de destino. A otimalidade do algoritmo é garantida caso *h(s)* subestime o custo para se chegar ao destino, ou seja, $h(s) \leq custo(s)$, para todo *s* do grafo. Neste caso, diz-se que h é admissível (HART; NILSSON; RAPHAEL, 1968).

O ganhador da competição realizada pela *DARPA* em 2007 utilizou o algoritmo A* Estado Híbrido para realizar o planejamento de caminhos (DOLGOV et al., 2010). O algoritmo A* Estado Híbrido calcula o valor da heurística h utilizando outras duas heurísticas: *Heurística Não-Holonômica sem Obstáculos* que leva em consideração a cinemática do veículo e *Heurística Holonômica com Obstáculos* que estima o valor do vértice atual até o vértice destino desviando de obstáculos. A utilização destas duas heurísticas em conjunto permite ao algoritmo estabelecer, a partir de um vértice, o próximo vértice do caminho em direção ao objetivo respeitando as limitações físicas do modelo adotado.

Para ambientes dinâmicos, onde o A* convencional encontra dificuldades devido a atualização contínua do mapa, foram criadas as versões incrementais e *anytime* do A*. As versões incrementais visam replanejar a trajetória quando alguma alteração no ambiente é percebida. As versões *anytime* se preocupam em encontrar uma trajetória subótima o mais rápido possível dentro de um espaço de tempo limitado, e caso haja tempo disponível neste intervalo limitado de tempo, o algoritmo tenta melhorar a solução realizando alterações na heurística *h* (RIOS; CHAIMOWICZ, 2010). Os principais algoritmos desenvolvidos na versão incremental foram o *Dynamic A** (STENTZ, 1995) e *D* Lite* (KOENIG; LIKHACHEV, 2002) devido à heurística *h* implementada e à forma como as atualizações do ambiente são processadas. Esses algoritmos conseguem replanejar a trajetória levando em consideração o planejamento já realizado até o momento, o que garante um menor esforço computacional. O principal algoritmo criado na versão *anytime* foi o ARA* (*Anytime Repairing A**) (LIKHACHEV; GORDON; THRUN, 2004). Visando unir as duas versões, foi desenvolvido o algoritmo AD* (*Anytime Dynamic A**) (LIKHACHEV et al., 2005).

Um algoritmo bastante difundido para resolver o problema de planejamento de caminhos é o *RRT* (LAVALLE, 1998). Este algoritmo se baseia na construção de uma árvore de configurações através de amostras aleatórias uniformes no *C-Free* a partir de um ponto de origem expandindo a árvore em direção ao destino e evitando colisões. Os estados da árvore correspondem às configurações do robô e são conectados através de arcos que representam um comando que foi aplicado para alcançar o novo estado a partir do estado anterior. Quando o estado destino é alcançado, uma trajetória do estado inicial até o estado destino é traçada. Várias modificações foram realizadas no algoritmo *RRT* básico para melhorar sua performance, principalmente na heurística que calcula o estado aleatório a ser explorado (LAVALLE; KUFFNER, 2001).

Recentemente o algoritmo *RRT**, uma extensão do *RRT*, foi desenvolvido e sua principal característica é encontrar um caminho assintoticamente ótimo entre o estado inicial e final para modelos de veículos com dinâmica de movimentos simples. Para veículos holonômicos um segmento de reta representa o caminho mínimo entre dois estados e é facilmente calculado pelo *RRT**. Porém para veículos não holonômicos, onde existe um grau menor de liberdade para os movimentos isto não é trivial (KARAMAN; FRAZZOLI, 2011). Em (WEBB; BERG, 2013) os autores desenvolveram o *Kinodynamic RRT** com novas primitivas para conectar dois estados da árvore de configurações possibilitando, apesar do algoritmo gastar mais tempo computacional, encontrar caminhos para veículos não holonômicos.

A técnica Algoritmo Genético (AG) está inserido na classe dos algoritmos evolutivos que se baseiam em população de cromossomos criado em 1975 por John Holland. As variáveis dos cromossomos são chamados de genes e seus principais operadores são a seleção natural, *crossover* e mutação. O operador de seleção consiste em avaliar a população de cromossomos através de uma função objetivo e selecionar os melhores cromossomos para criar uma nova população. A operação de *crossover* cria uma nova população através do cruzamento de dois ou mais cromossomos misturando parte de um cromossomo com parte de outro. O operador de mutação cria uma nova população aleatória ou parcialmente aleatória. Além dos operadores básicos mencionados, tem-se o elitismo que consiste em perdurar os cromossomos com melhor avaliação para as próximas gerações (HOLLAND, 1975).

Em (SOLANO; JONES, 1993) um método utilizando AG para encontrar um caminho em ambiente estático é proposto. Neste trabalho os autores consideram uma circunferência de raio r cujo centro é o ponto inicial S (posição atual do veículo) para então definir o cromossomo como um ponto viável (que não é obstáculo) p dentro da circunferência r. A função objetivo consiste em calcular a distância do ponto p até o destino T. O algoritmo

tenta minimizar a função objetivo e quando o ponto p é determinado ele é setado como posição atual e então o AG é executado novamente até alcançar o objetivo T.

No trabalho de (SUGIHARA; SMITH, 1997), os autores utilizaram AG com cromossomos de tamanho fixo para planejar caminhos em mapas estáticos. O método encontra trajetórias rapidamente em ambientes com poucos obstáculos porém em ambientes complexos, isto é, com mais obstáculos, o algoritmo encontrou dificuldades para encontrar uma solução. Para resolver este problema, um algoritmo de planejamento utilizando AG com cromossomos de tamanho variável foi criado onde cada gene representa a direção e a distância do movimento subsequente. Testes realizados em mapas complexos mostraram que este algoritmo não obteve solução pois os cromossomos possuíam muitos genes sendo difícil estabelecer um caminho viável no mapa passando por todos os genes (pontos) (JIANPING; YANG, 2003). Trabalhos mais recentes (ALAJLAN et al., 2013; LEE; KANG; KIM, 2013; FERARIU; CIMPANU, 2014) utilizam cromossomos de tamanho variável, corrigindo os problemas ocorridos nas versões anteriores, e critérios multiobjetivos para encontrar caminhos em ambientes estáticos e dinâmicos. Os resultados mostram que o algoritmo encontra boas soluções porém o tempo computacional, dependendo da complexidade dos mapas, pode ser muito elevado.

O algoritmo PSO faz parte de uma classe de algoritmos denominados *Swarm Intelligence* (Inteligência de Enxames). Esta subárea da inteligência computacional é inspirada na inteligência coletiva observada em animais da natureza. Em especial, o PSO foi inspirado no comportamento de pássaros, insetos e peixes, desenvolvido em 1995 pelo psicólogo social James Kennedy e o engenheiro eletricista Russell Eberhart. Inicialmente uma população de partículas (soluções) é gerada aleatoriamente e em seguida, cada partícula atualiza sua posição levando em consideração o aprendizado passado (melhor posição encontrada pela partícula no enxame) e o aprendizado coletivo (melhor posição encontrada por uma partícula em todo o enxame). Após atualizar sua posição, a partícula analisa se o valor da solução encontrado é melhor, caso contrário, a partícula poderá ser descartada ou sofre uma penalização dependendo do tipo de implementação do algoritmo. Tal procedimento é executado até que a solução do problema seja encontrada ou algum critério de parada seja atingido tais como, número máximo de iterações, número máximo de iterações sem melhora da solução, entre outros (KENNEDY; EBERHART, 1995).

Alguns trabalhos na literatura apresentam uma combinação do PSO com outros algoritmos. Em (YUANQING et al., 2004), após modelado o ambiente estático em um grafo do tipo *MAKLINK* (HABIB; ASAMA, 1991), o algoritmo de Dijkstra é utilizado para encontrar o menor caminho entre o ponto inicial e final e, em seguida, o PSO é utilizado com operador de mutação para otimizar esta trajetória. O algoritmo conseguiu diminuir o comprimento do caminho em todos os testes realizados. No trabalho (SASKA et al.,

2006), os autores propuseram um algoritmo baseado em curvas de *Ferguson* (conectar dois pontos no plano através de uma curva construída utilizando parâmetros) onde seus parâmetros foram otimizados utilizando o PSO. Os resultados obtidos foram comparados com os algoritmos campos potenciais e grafo de visibilidade e mostrou-se mais eficiente pois o caminho encontrado foi mais suave e obtido com menor tempo computacional.

Os autores de (QIAORONG; GUOCHANG, 2008) propõem a utilização do PSO binário (onde as variáveis do problema assumem os valores 0 ou 1) utilizando operadores de mutação para solucionar o problema de planejamento de caminhos onde os obstáculos são polígonos e seus vértices são numerados de 1 a n. O tamanho da partícula é dado pelo número de vértices dos obstáculos do mapa e cada *bit* da partícula assume o valor 1 ou 0 caso o vértice pertença ou não ao caminho, respectivamente.

No trabalho (ZHANG; GONG; ZHANG, 2013), é proposto um planejador em ambientes dinâmicos utilizando o PSO. Inicialmente o PSO define uma trajetória global e posteriormente esta trajetória é otimizada em virtude das mudanças no ambiente. Atualmente os esforços utilizando esta técnica visam encontrar caminhos em ambientes dinâmicos com multiobjetivo, a citar, tamanho do caminho, escapar de zonas perigosas, suavização do caminho, dentre outros (GONG; ZHANG; ZHANG, 2011; LIGUO et al., 2013; KIBAEK; JONGHWAN, 2013).

No contexto geral de técnicas e algoritmos para planejamento de caminhos observa-se um aumento na utilização de heurísticas e meta-heurísticas aplicadas ao referido problema (MASEHIAN; SEDIGHIZADEH, 2007). Diante disto, os dois próximos capítulos resumem os conceitos e algoritmos utilizados na elaboração do algoritmo A*PSO proposto neste trabalho.

3 *Problema do planejamento de caminhos*

Neste capítulo são apresentados e detalhados os conceitos básicos para o entendimento da definição e formulação do problema de planejamento de caminhos como um problema de otimização adotado neste trabalho.

3.1 Mapas

Representar o mundo real de forma que um computador possa entender é uma importante etapa no processo de navegação de veículos autônomos. Esta pode ser uma tarefa difícil se imaginarmos as inúmeras possibilidades de movimentação no mundo real e a representação dos obstáculos e dos robôs. Em essência, todas as abordagens para o problema de caminhos são semelhantes quando formulados com *C-Space*: os problemas são reduzidos a encontrar uma sequência de pontos entre os pontos inicial e final no espaço de configuração (HWANG; AHUJA, 1992).

Várias técnicas foram introduzidas para mapear o mundo real em um espaço de configuração. Dentre elas, destacam-se as representações *grid* (Decomposição Celular), Grafo de Visibilidade e de Voronoi (Mapas de Caminhos) e representações geométricas. A representação do mundo real a partir de um mapa utilizando a técnica *grid* bidimensional, onde cada célula representa uma posição que o robô poderá ocupar foi apresentada pela primeira vez em 1990. Além disso, a célula possui uma informação sobre o estado do espaço real: 1 caso seja um obstáculo, 0 caso seja um espaço livre e -1 para o caso de local desconhecido (ELFES, 1990).

Na Figura 6 um mapa do tipo *grid* é apresentado. As células em branco representam uma área livre, as pretas representam um obstáculo e as cinzas uma região desconhecida. Estas informações são muito importantes para validar um movimento realizado pelo robô. Caso o movimento passe por uma célula com valor igual a 1 (preta) ou -1 (cinza), então este movimento é inviável.

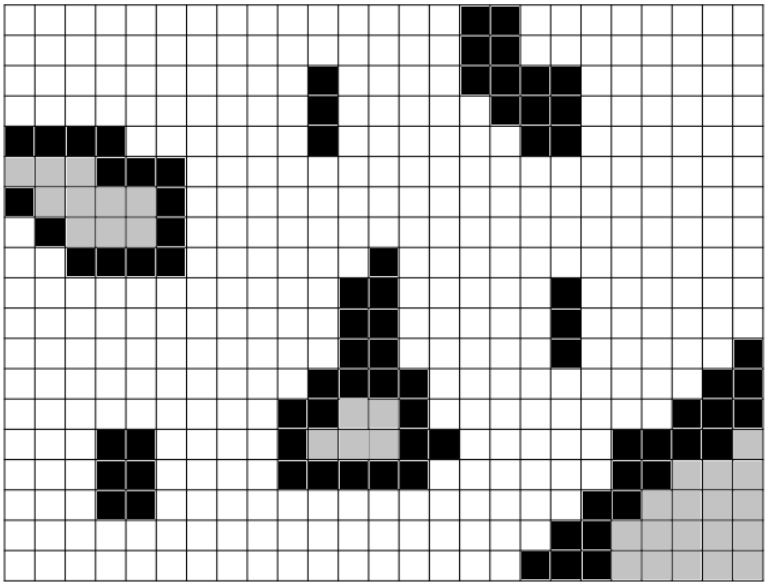


Figura 6: Mapa do tipo *grid*.

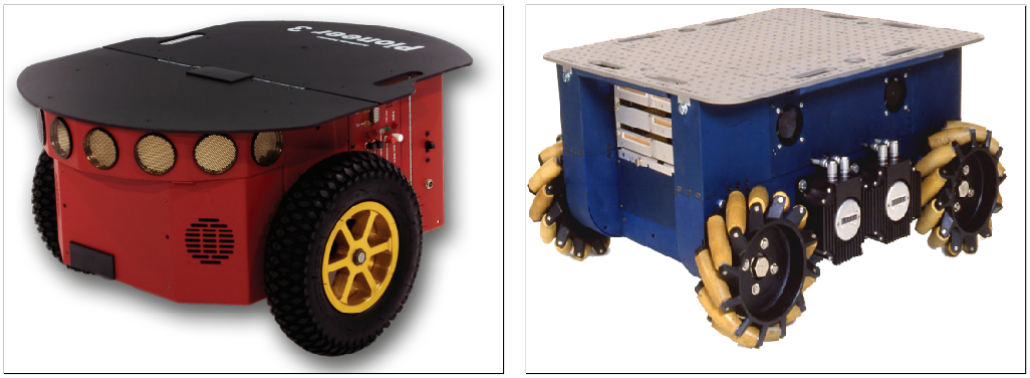
3.2 Modelos de veículo autônomos

Os veículos possuem diversos modelos anfíbios, aéreos e terrestres onde, dependendo da forma como foram projetados, possuem limitações de movimentação devido às suas naturezas físicas (COLYER; ECONOMOU, 1998). A forma como o robô é modelado é de extrema importância para o planejamento de caminhos. Existem diversos tipos de robôs: aquáticos, aéreos, terrestres, dentre outros. Especificamente neste trabalho serão discutidos dois modelos terrestres: holonômicos e não holonômicos. Não levar em consideração as restrições de movimento do robô durante a fase de planejamento pode resultar em movimentos cujo robô não consegue efetuar. Por este motivo é fundamental considerar a natureza física do veículo. O conceito de holonomia está relacionado entre o número de graus de liberdade existentes e o número de graus de liberdade controláveis. Um robô é dito não holonômico se o grau de liberdade controlável é inferior ao grau de liberdade existente. Caso esses números sejam iguais, então o veículo é caracterizado como holonômico (ALMEIDA, 1996). Nas subseções seguintes veremos exemplos dos dois tipos de veículos.

3.2.1 Modelos holonômicos

Estes modelos são caracterizados por conseguir atingir qualquer posição vizinha à posição atual do veículo e possuem um modelo cinemático mais simples. Os principais modelos holonômicos são os diferenciais e os omnidirecionais. No primeiro, o veículo possui duas rodas e gira em torno do seu próprio eixo para atingir qualquer posição vizinha. Enquanto que no segundo tipo, o veículo dispõe de mecanismos que, para atingir uma posição vizinha, não é necessário fazer movimentos de rotação. O modelo 3-DX da fabricante *Pioneer* é um exemplo de veículo diferencial (Figura 7(a)). Um exemplo de

veículo omnidirecional é o *URANUS*, projeto da universidade *Carnegie Mellon* desenvolvido em 1985 (Figura 7(b)).



(a) Modelo diferencial Pioneer 3-DX. (b) Modelo omnidirecional URANUS.

Figura 7: *Modelos holonômicos*

3.2.2 Modelos não holonômicos

A introdução de restrições cinemáticas de movimento tornam o problema de planejamento de caminhos mais complexo visto que o veículo não poderá assumir qualquer posição a partir da atual. Várias modelagens foram elaboradas para simular um veículo não holonômico. Dentre as mais utilizadas, destaca-se a modelagem *Ackerman* (SIEGWART; NOURBAKHSH, 2004). Na figura 8 é disposto o modelo *Ackerman* e suas principais variáveis: a posição do veículo é dada pelas coordenadas x e y localizadas entre as rodas traseiras, a orientação é dada pelo ângulo θ, o ângulo φ corresponde ao ângulo das rodas dianteiras. Os atuadores de uma posição são a velocidade v, o intervalo de tempo ΔT e o ângulo φ. O parâmetro L corresponde à distância entre os eixos dianteiro e traseiro. Assim, o modelo cinemático de movimentação *Ackerman* é dado por:

$$x_{t+1} = x_t + \Delta t_t * v_t * cos(\theta_t) \tag{3.1}$$

$$y_{t+1} = y_t + \Delta t_t * v_t * sin(\theta_t) \tag{3.2}$$

$$\theta_{t+1} = \theta_t + \Delta t_t * v_t * \frac{tan(\theta_t)}{L} \tag{3.3}$$

A partir desses valores, o vetor de estado (posição) do veículo é definido como:

$$s = [x, y, \theta]^T \tag{3.4}$$

e o vetor de controle (informações passadas ao módulo de execução de movimentos) é definido como:

$$u = [v, \varphi, \Delta t]^T \tag{3.5}$$

Nas equações 3.1, 3.2 e 3.3, x_{t+1}, y_{t+1} e θ_{t+1} representam a nova posição do veículo dada pelo modelo *Ackerman* e fornecem informações suficientes para realizar a estimativa de um caminho de um estado inicial s_0 a um final s_f respeitando a cinemática do veículo.

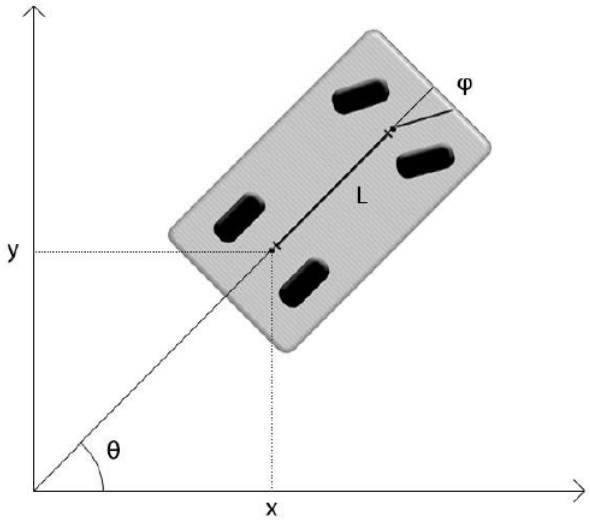


Figura 8: Modelo *Ackerman*.

3.3 Definição do problema

Dado um veículo autônomo correspondente a um ponto em um ambiente bidimensional com obstáculos estáticos, deseja-se encontrar um caminho entre dois pontos, *S* e *T*, que deve ser:

- **Viável**: O caminho não pode colidir com nenhum obstáculo.
- **Mínimo**: Encontrar o menor caminho entre *S* e *T*.
- **Restrito ao modelo do veículo**: Respeitar a cinemática do veículo autônomo utilizado (neste trabalho, modelo *Ackerman* e holonômico).

A modelagem do ambiente corresponde a mapear cada célula do mapa *grid* em um ponto do plano bidimensional onde este ponto representa uma possível posição que o veículo pode assumir.

3.4 Formulação do problema

O modelo matemático adotado neste trabalho para representar o problema foi baseado no artigo (DUNWEI; LI; MING, 2009) com a diferença que os pontos *S* e *T* não necessitam estar sob o eixo *X* do plano cartesiano e os obstáculos podem assumir qualquer forma geométrica. O ambiente modelado é apresentado na Figura 9. Podemos observar que cada um dos obstáculos $O_1, O_2, ..., O_m$ têm sua área delimitada por um agrupamento de pontos, sendo *m* a quantidade de obstáculos no mapa. Alguns pontos no mapa, denominados *waypoints*, fora das áreas ocupadas pelos obstáculos, são escolhidos para formar um caminho. Após determinados k *waypoints*, o caminho é formado pela ligação de *S* ao primeiro *waypoint* e dos *waypoints* subsequentes até *T*. Com isso, teremos *k+1* segmentos de reta formando o caminho com *k waypoints*. O caminho é dito viável quando nenhum segmento de reta formado intercepta algum obstáculo.

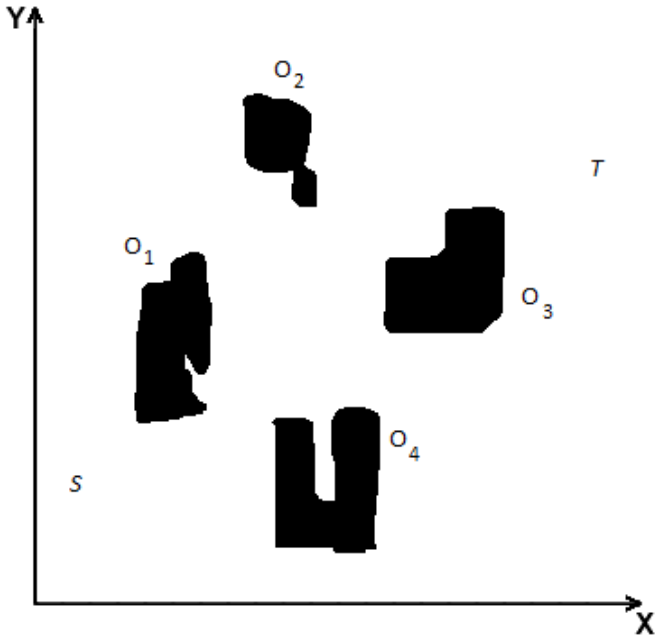


Figura 9: Modelo do ambiente.

Definindo *k waypoints* como as variáveis de decisão, *S* como A_0 e *T* como A_{k+1}, o caminho formado tem o seguinte formato: $\overline{A_0A_1}, \overline{A_1A_2}, ..., \overline{A_kA_{k+1}}$ onde $A_i = (x_i, y_i)$ para todo $i = \{0, \ldots, k\}$ e $A_0 = (x_s, y_s)$ e $A_{k+1} = (x_t, y_t)$. Com isso podemos definir uma função *L* que calcula o comprimento do caminho através da soma dos comprimentos de todos os segmentos de reta que o compõem.

$$\begin{aligned} L(\overline{A_0A_1...A_{k+1}}) &= \sqrt{(x_1 - x_s)^2 + (y_1 - y_s)^2} \\ &+ \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \\ &+ ... + \sqrt{(x_t - x_k)^2 + (y_k^2)} \end{aligned} \tag{3.6}$$

Como nosso objetivo é minimizar o tamanho do caminho encontrado e, dispondo da função L dada pela equação (3.6) e das restrições dos obstáculos, podemos formular o problema de planejamento de caminho mínimo como o seguinte problema de otimização:

$$\begin{aligned} min\ L(\overline{A_0 A_1 ... A_{k+1}}) \quad &= \quad \sqrt{(x_1 - x_s)^2 + (y_1 - y_s)^2} \\ &+ \quad \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \\ &+ \quad ... + \sqrt{(x_t - x_k)^2 + (y_k^2)} \\ s.a. \quad & A_j A_{j+1} \cap O_i = \emptyset, i = 1, 2, ..., m, j = 0, 1, ..., k \\ & A_j \in E, j = 0, 1, ..., k \end{aligned} \tag{3.7}$$

onde E corresponde ao universo de pontos onde o robô pode se mover.

No próximo capítulo são discutidos algoritmos que solucionam o problema de planejamento de caminhos considerando os dois modelos, holonômico e *Ackerman*, utilizando mapas do tipo *grid*.

4 *Métodos utilizados para planejamento dos caminhos*

Este capítulo visa apresentar com mais detalhes os algoritmos A*, PSO, Reeds & Shepp e Bresenham utilizados na elaboração do algoritmo híbrido A*PSO proposto neste trabalho.

4.1 Algoritmo A*

Os algoritmos de busca geralmente utilizam uma lista para manter os elementos (vértices) a serem explorados. Algumas técnicas, como a *best-first search*, utilizam uma função de avaliação para ordenar a lista de forma que o primeiro elemento é julgado como o melhor a ser explorado naquele momento (RUSSELL; NORVIG, 2009). Os algoritmos de busca são geralmente utilizados em árvores de buscas estruturadas em grafos, onde o objetivo é encontrar um vértice específico na árvore ou então um caminho entre dois vértices.

As buscas podem ser classificadas como informadas ou não informadas. A busca é dita informada quando utiliza informações do problema para agilizar o processo tentando reduzir o número de nós explorados e consequentemente reduzir o tempo de processamento. As buscas desinformadas, ou cegas, na prática são mais lentas que as buscas informadas pois visitam mais nós. (RUSSELL; NORVIG, 2009).

O algoritmo A* é considerado um algoritmo de busca informada e usa a ideia de dois outros algoritmos: busca de custo uniforme (*uniform-cost-search*) e busca gulosa (*greedy-search*).

Na busca de custo uniforme o próximo vértice n do grafo a ser visitado será o vértice de menor custo da fila calculado por uma função $g(n)$. Esta função calcula o custo do vértice raiz até n. Se a condição $g(sucessor) \geq g(n)$ para todo vértice n for atendida, então a busca de custo uniforme retornará o caminho de menor custo sem precisar visitar todos os vértices. Observa-se que se houver algum custo negativo na árvore de busca então o algoritmo deverá visitar todos os vértices para que a otimalidade seja mantida. O principal problema desse algoritmo consiste em ter que percorrer a fila em busca do

vértice de menor custo. Dependendo da quantidade de vértices na árvore de busca isto poderá ser inviável. O algoritmo é completo já que sempre retornará uma solução caso exista ou falhará caso não exista solução (RUSSELL; NORVIG, 2009).

A outra estratégia utilizada pelo A* é a busca gulosa. Neste algoritmo a fila dos vértices é organizada em virtude de uma função de avaliação que calcula o custo do vértice atual n até o vértice de destino, sendo assim, o próximo vértice a ser visitado será o que for julgado mais próximo ao destino. É difícil precisar, na maioria dos problemas, o custo real do vértice n ao vértice destino, portanto é utilizada uma função de avaliação que estima esse valor. Esta função é chamada de função heurística ($h(n)$. Se $h(n) = 0$ então o algoritmo chegou ao destino. Este algoritmo é muito rápido porém não é otimal visto que poderá retornar uma solução que não é a de custo mínimo. É incompleto já que pode cair em *loops* infinitos principalmente se encontrar um vértice folha (RUSSELL; NORVIG, 2009).

O algoritmo A* utiliza uma função de avaliação que é a soma de $g(n)$ e $h(n)$. Com isso o A* consegue cortar vértices não promissores (característica da busca gulosa) e será completo e otimal (característica da busca de custo uniforme) dependendo da função $h(n)$ utilizada. Como $g(n)$ calcula o custo do vértice raiz até n e $h(n)$ calcula a estimativa de menor custo de n até o vértice destino, temos:

$$f(n) = g(n) + h(n) \tag{4.1}$$

que consiste na solução de custo mínimo passando por n.

Entretanto o algoritmo A* somente será completo e otimal caso haja uma restrição na função $h(n)$. Essa restrição consiste em que o custo da função $h(n)$ nunca superestime o custo real de n até o destino. Logo $h(n)$ deverá ser uma heurística admissível pois estas são consideradas otimistas, ou seja, sempre acham que o custo para resolver o problema é menor ou igual do que ele realmente é. A característica otimista é passada para a função f, portanto $f(n)$ nunca irá superestimar o custo real da melhor solução passando por n. Um exemplo clássico de heurística admissível no problema de caminho mínimo é a distância euclidiana, como ela é a menor distância entre dois pontos, o valor de $h(n)$ nunca será maior que o valor real de n até o destino (RUSSELL; NORVIG, 2009). Outras funções para determinar $h(n)$ podem ser utilizadas, como a distância de *manhattan*, que, apesar de não ser admissível, na prática encontra uma solução mais rápido que a euclidiana por visitar, em geral, um número menor de vértices. Isto acontece pois ao calcular ambas as distâncias entre dois pontos, *h(manhattan)* é sempre maior que *h(euclidiana)*, sendo então a distância de *manhattan* dominante com relação à distância euclidiana e consequentemente mais vértices são podados durante a busca.

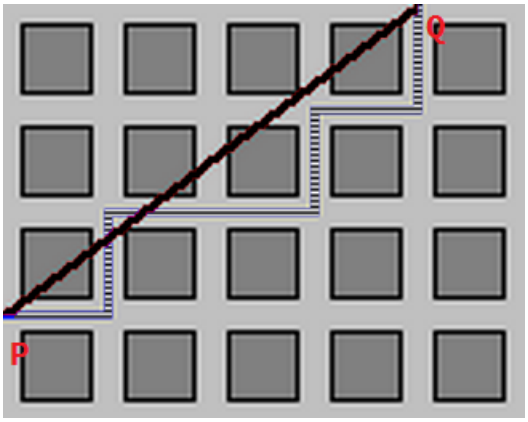


Figura 10: Distância de *manhattan* (tracejada) e distância euclidiana (em preto).

A Figura 10 mostra uma comparação entre a distância euclidiana e *manhattan*. Tendo *P* as coordenadas *(x1,y1)* e *Q (x2,y2)*, a distância de *manhattan* é igual a $|x1 - x2| + |y1 - y2|$. A distância euclidiana é calculada utilizando a equação $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

No Algoritmo 1 as listas *Aberta* e *Fechada* são inicializadas, nas linhas 1 e 2, com o vértice n_i e vazia respectivamente. Enquanto a *ListaAberta* não é vazia (linha 3), o algoritmo extrai o vértice n com melhor valor de função $f(n)$ da lista *Aberta* e adiciona à *Fechada*. Se n é igual a n_f o algoritmo é encerrado e retorna o caminho até n_f caso contrário a busca continua. Os vizinhos n' de n que estão na lista *Fechada* são descartados pois já foram visitados. Os vizinhos que não estão em nenhuma das listas tem seus valores de *g*, *h* e f calculados e o anterior setado para *n* (linhas 11 à 14). Na linha 15 então o vizinho *n'* é adicionado à lista *Aberta*. Se os vértices n' já estiverem na lista *Aberta*, então o teste da linha 16 é executado. Caso o teste seja verdade, o caminho até n' passando por n tem um custo menor e então o antecessor de n' é setado para n pois apresenta um custo menor. Quando a lista *Aberta* está vazia, o algoritmo termina e no vetor $anterior$ está o caminho entre n_i e n_f.

4.2 Algoritmo PSO

Os algoritmos evolutivos, onde o PSO está incluído, são caracterizados por possuírem uma população de soluções. Cada solução é avaliada através de uma função de adaptação, denominada *fitness*, onde o valor de solução desta função indica o quão a solução é boa. O algoritmo realiza perturbações nos indivíduos da população a fim de criar novas soluções que caminhem em direção à solução ótima. Os membros da população podem trocar informações com seus vizinhos de várias formas diferentes, o que define uma topologia. Este mecanismo ressalta o senso coletivo entre os indivíduos que conseguem aprender com seus vizinhos buscando sempre o mesmo objetivo, isto é, encontrar a solução ótima.

Algoritmo 1: *Pseudocódigo do Algoritmo A**

Entrada: Grafo G, vértice inicial n_i, vértice destino n_f
Saída: Caminho de n_i até n_f

$ListaAberta \leftarrow n_i$;
$ListaFechada \leftarrow 0$;
enquanto *ListaAberta não é vazia* **faça**
    `/* desenfila() retorna o vértice com menor valor de f */`
    $n \leftarrow desenfila(ListaAberta)$;
    remove *n* da *ListaAberta* e coloca na *ListaFechada*;
    se $n = n_f$ **então**
        Termina algoritmo;
    fim se
    para cada *vizinho* n' *de* n *que não está na ListaFecha* **faça**
        se $n' \notin ListaAberta$ **então**
            `/* Função c() calcula o custo de um vértice para outro */`
            $g(n') = g(n) + c(n, n')$;
            $h(n') =$ *distância do destino até n'*;
            $f(n') = g(n') + h(n')$;
            $anterior(n') \leftarrow n$;
            $ListaAberta \leftarrow n'$;
        senão se $g(n') > g(n) + c(n, n')$ **então**
            $anterior(n') \leftarrow n$;
        fim se
    fim para cada
fim enquanto
retorna anterior;

No algoritmo PSO, é definida uma população de partículas (possíveis soluções) que procuram melhorar suas posições levando em consideração o aprendizado passado (melhor posição encontrada pela partícula no enxame, denotada por *pBest*) e o aprendizado coletivo (melhor posição encontrada por uma partícula em todo o enxame, denotada por *gBest*). Com isso, a partícula é capaz de evoluir para uma solução melhor que a atual. Tal procedimento é executado até que a solução do problema seja encontrada ou algum critério de parada seja atingido como: número máximo de iterações, número máximo de iterações sem melhora da solução, entre outros. O PSO objetiva minimizar ou maximizar uma função objetivo (*fitness*) aplicada sobre um conjunto de variáveis representadas pela própria partícula (solução do problema) (KENNEDY; EBERHART, 1995).

O PSO é inicializado com partículas aleatórias cujas posições são atualizadas conforme uma determinada velocidade $v(t + 1)$. O desafio principal do algoritmo é como atualizar a velocidade de forma a caminhar para a solução do problema, seguindo a ideia de psicologia social onde as partículas trocam informações entre si (KENNEDY; EBERHART, 1995).

As equações 4.2 e 4.3 representam na versão clássica do PSO, respectivamente as

descrições matemáticas da velocidade e da posição da partícula $x(t+1)$:

$$v(t+1) = v(t) + c_1 * r_1 * (\textit{pBest} - x(t)) + c_2 * r_2 * (\textit{gBest} - x(t)) \tag{4.2}$$

$$x(t+1) = x(t) + v(t+1) \tag{4.3}$$

Os parâmetros c_1 e c_2 são fatores de aprendizado denominados cognitivo local e cognitivo social respectivamente, r_1 e r_2 são números aleatórios escolhidos entre [0,1] que garantem maior diversidade ao algoritmo. O valor de *fitness* associado ao *pBest* corresponde a melhor posição encontrada pela partícula $x(t)$ e o valor de *fitness* de *gBest* à melhor posição dentre todas as partículas do enxame. Pode-se verificar que a velocidade depende do aprendizado passado e da interação coletiva social. No Algoritmo 2 é apresentado o pseudo-código do algoritmo clássico do PSO.

Algoritmo 2: Pseudocódigo do PSO

Entrada: Tamanho do enxame, número máximo de iterações, c_1, c_2, função fitness f
Saída: *gBest* (melhor solução do enxame)

```
1  para cada Partícula faça
2  |  inicializa Partículas;
3  fim para cada
4  repita
5  |  para cada Partícula faça
6  |  |  se f(Partcula) < f(pBest) então
7  |  |  |  pBest ← Partcula;
8  |  |  fim se
9  |  |  se f(pBest) < f(gBest) então
10 |  |  |  gBest ← pBest;
11 |  |  fim se
12 |  fim para cada
13 |  para cada Partícula faça
14 |  |  Atualiza velocidade da Partícula conforme Equação 4.2;
15 |  |  Atualiza posição da Partícula conforme Equação 4.3;
16 |  fim para cada
17 até Número máximo de iterações;
```

No Algoritmo 2 a inicialização aleatória das partículas é realizada nas linhas de 1 a 3. Entre as linhas 4 e 18 efetivamente as evoluções das partículas acontecem. Nas linhas 5 a 13 é realizado o procedimento para verificar se houve melhora da solução no enxame, caso sim, então os valores *pBest* e *gBest* são atualizados. Por fim, a posição e a velocidade das partículas são atualizadas nas linhas de 14 a 17. O algoritmo é encerrado caso o critério de parada da linha 18 seja atingido.

Um mecanismo de controle de fronteira é utilizado no algoritmo PSO tanto para a velocidade quanto para a posição das partículas. Este mecanismo define um intervalo

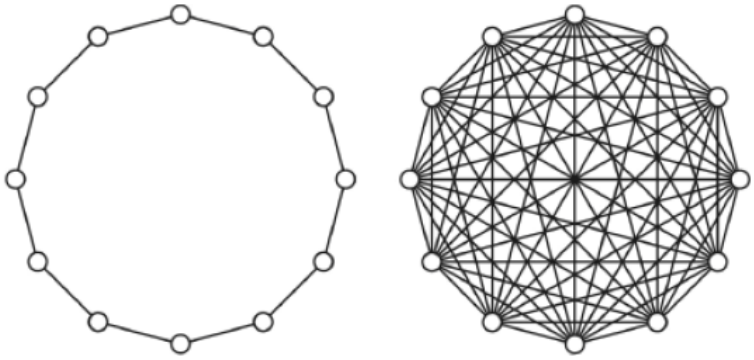
inferior e superior onde a velocidade e a posição não podem exceder. Caso excedam, os valores da velocidade e posição são ajustados para os valores limite do intervalo (YUHUI; EBERHART, 1999).

Uma melhoria no algoritmo clássico do PSO foi a introdução de um fator inércia (*w*) visando melhorar a velocidade de convergência do algoritmo sem alterar sua estrutura (YUHUI; EBERHART, 1999). A inércia é um fator escalar e está associado à velocidade, podendo ser modificado durante a iteração do algoritmo. Estudos sugerem que *w* deve ser inicializado com um valor alto próximo a 2, o que garante que as partículas irão explorar mais o espaço de busca. Com o decrescimento de *w* para valores abaixo de 1 (um) durante as iterações, pode-se observar um refinamento na solução incumbente do algoritmo visto que a velocidade ao final das iterações do algoritmo será menor. A introdução deste fator resulta na equação:

$$v(t+1) = w * v(t) + c_1 * r_1 * (\textit{pBest} - x(t)) + c_2 * r_2 * (\textit{gBest} - x(t)) \tag{4.4}$$

onde a alteração consiste somente na adição do fator de inércia *w* multiplicando a velocidade $v(t)$.

A topologia de comunicação entre as partículas é outro componente que implica diretamente no desempenho do algoritmo PSO. Esta topologia rege como as partículas irão trocar informações entre si. As principais topologias são a topologia local e a topologia global apresentadas na Figura 11.



(a) Topologia local (b) Topologia global

Figura 11: *Topologias de comunicação das partículas do PSO*

Na topologia local (Figura 11(a)) a troca de informação se dá somente com os vizinhos adjacentes à partícula. Este mecanismo confere uma convergência mais suave, escapando melhor de mínimos locais, pois a troca de informação é mais lenta e a partícula demora a receber a informação sobre a melhor solução do enxame. Embora esta topologia seja mais lenta que a global, ela geralmente encontra uma solução de melhor

qualidade (ENGELBRECHT, 2006).

Na Figura 11(b) é observada a topologia global onde a informação é compartilhada entre todas as partículas do enxame. Nesta topologia, o algoritmo PSO tende a convergir mais rápido mas geralmente a qualidade da solução é pior que na topologia local (ENGELBRECHT, 2006).

4.3 Curvas Reeds e Shepp

O algoritmo de Reeds e Shepp definido em (REEDS; SHEPP, 1990), consiste em realizar a junção de segmentos de retas e/ou arcos de círculos, formando um caminho entre dois pontos no mapa. Os arcos em questão respeitam o raio de curvatura mínimo do modelo do veículo utilizado, garantindo assim que o caminho encontrado satisfaça as restrições de movimentação do veículo como por exemplo, o modelo *Ackerman* apresentado na Seção 3.2.2.

Uma base de palavras para representar o movimento identificadas por *C*, *S* e | representam respectivamente uma movimentação em arco, movimentação em linha reta e troca de marcha (entre ré e frente). Seis movimentos primitivos são definidos: RETO(R), DIREITA(R) e ESQUERDA(L), tanto para frente(+) quanto para trás(-). Mesclando essas duas configurações os autores provaram ser possível encontrar um caminho mínimo, entre dois pontos, sem desviar de obstáculos e respeitando as limitações de curvatura do veículo utilizando 48 configurações. A Figura 12 mostra todas as 48 configurações para uma dada escolha de origem e destino. Na Figura 13 é exibido um exemplo de uma curva $R_{\alpha}^{+}L_{\beta}^{-}R_{\gamma}^{+}$. Neste exemplo, R_{α}^{+} indica um movimento para direta (R) e para frente (+) estabelecendo um ângulo α. A próxima ação realiza um movimento para a esquerda (L) e para trás (-) mantendo o ângulo β durante o movimento. Finalmente o último movimento é realizado para direita (R) e para frente (+) com ângulo γ e então é estabelecido um caminho entre q_I até q_G.

4.4 Algoritmo de Bresenham

Este algoritmo foi desenvolvido em 1965 por J. E. Bresenham com o objetivo de criar uma sequência de comandos para impressão de retas em uma impressora digital (BRESENHAM, 1965). Utilizando-se apenas de operações baratas como adição e subtração de inteiros e troca de *bits*, o algoritmo consegue destacar todos as células que são interceptados por uma reta em um *grid* 1x1m^2.

Analisando uma reta $\overline{pq}$ no primeiro octante, o algoritmo define quais células são ativadas conectando *p* a *q* célula à célula. A cada iteração, partindo de $p(x, y)$, *x* é incre-

PALAVRAS BASES	MOVIMENTOS PRIMITIVOS
$C\|C\|C$	$(L^+R^-L^+)(L^-R^+L^-)(R^+L^-R^+)(R^-L^+R^-)$
$CC\|C$	$(L^+R^+L^-)(L^-R^-L^+)(R^+L^+R^-)(R^-L^-R^+)$
$C\|CC$	$(L^+R^-L^-)(L^-R^+L^+)(R^+L^-R^-)(R^-L^+R^+)$
CSC	$(L^+S^+L^+)(L^-S^-L^-)(R^+S^+R^+)(R^-S^-R^-)$ $(L^+S^+R^+)(L^-S^-R^-)(R^+S^+L^+)(R^-S^-L^-)$
$CC_\beta\|C_\beta C$	$(L^+R_\beta^+L_\beta^-R^-)(L^-R_\beta^-L_\beta^+R^+)(R^+L_\beta^+R_\beta^-L^-)(R^-L_\beta^-R_\beta^+L^+)$
$C\|C_\beta C_\beta\|C$	$(L^+R_\beta^-L_\beta^-R^+)(L^-R_\beta^+L_\beta^+R^-)(R^+L_\beta^-R_\beta^-L^+)(R^-L_\beta^+R_\beta^+L^-)$
$C\|C_{\pi/2}SC$	$(L^+R_{\pi/2}^-S^-R^-)(L^-R_{\pi/2}^+S^+R^+)(R^+L_{\pi/2}^-S^-L^-)(R^-L_{\pi/2}^+S^+L^+)$ $(L^+R_{\pi/2}^-S^-L^-)(L^-R_{\pi/2}^+S^+L^+)\ (R^+L_{\pi/2}^-S^-R^-)(R^-L_{\pi/2}^+S^+R^+)$
$CSC_{\pi/2}\|C$	$(L^+S^+L_{\pi/2}^+R^-)(L^-S^-L_{\pi/2}^-R^+)(R^+S^+R_{\pi/2}^+L^-)(R^-S^-R_{\pi/2}^-L^+)$ $(R^+S^+L_{\pi/2}^+R^-)(R^-S^-L_{\pi/2}^-R^+)(L^+S^+R_{\pi/2}^+L^-)(L^-S^-R_{\pi/2}^-L^+)$
$C\|C_{\pi/2}SC_{\pi/2}\|C$	$(L^+R_{\pi/2}^-S^-L_{\pi/2}^-R^+)(L^-R_{\pi/2}^+S^+L_{\pi/2}^+R^-)$ $(R^+L_{\pi/2}^-S^-R_{\pi/2}^-L^+)(R^-L_{\pi/2}^+S^+R_{\pi/2}^+L^-)$

Figura 12: 48 possibilidades de configuração de movimento do algoritmo Reeds e Shepp.

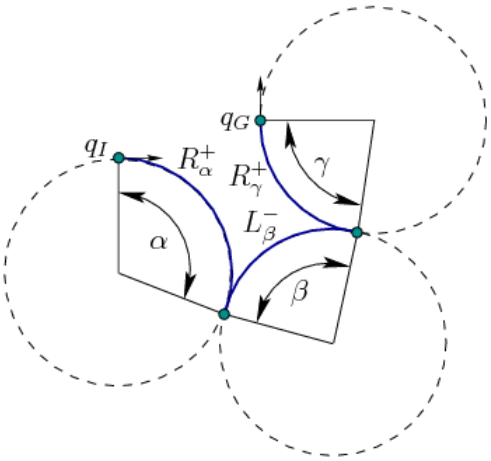


Figura 13: Exemplo de curva $R_\alpha^+ L_\beta^- R_\gamma^+$.

mentado em 1 e y é incrementado em função da distância entre a reta $\overline{pq}$ e a célula a ser ativada. Este procedimento é apresentado no Algoritmo 3.

A inicialização das variáveis acontece das linhas 1 a 5. Na linha 6 um *loop* é definido com número de iterações igual a *deltax*. A função *defineCelula* marca qual célula do *grid* deve ser ativada. Na linha 8 se o erro calculado for maior ou igual a 0 então y também é incrementado. Na linha 10 o erro é recalculado. Na linha 12 x é incrementado em 1 e finalmente na linha 13 o erro é ajustado para a próxima iteração. A Figura 14 exibe as células ativadas para um exemplo de execução do algoritmo.

Algoritmo 3: *Algoritmo de Bresenham*

Entrada: $x1, y1, x2, y2$
Saída: $celulasTranspostas$

```
1  x ← x1;
2  y ← y1;
3  deltax ← x2 − x1;
4  deltay ← y2 − y1;
5  erro ← 2 ∗ deltay − deltax;
6  para (i = 0; i < deltax; i = i + 1) faça
7  |  defineCelula(x, y);
8  |  enquanto erro >= 0 faça
9  |  |  y = y + 1;
10 |  |  erro = erro − 2 ∗ deltax;
11 |  fim enquanto
12 |  x = x + 1;
13 |  erro = erro + 2 ∗ deltay;
14 fim para
```

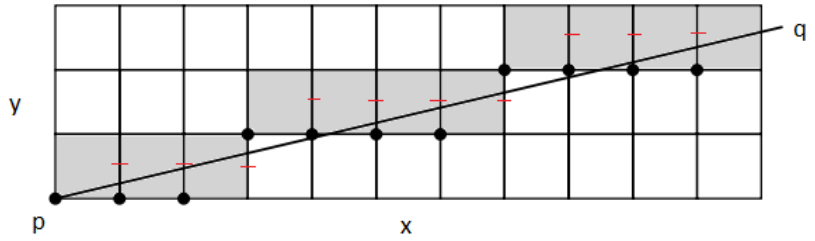


Figura 14: Células ativadas pelo Algoritmo de Bresenham de uma reta $\overline{pq}$.

4.5 Limitações dos algoritmos A* e PSO

Após a modelagem do ambiente em um grafo, o algoritmo A* pode ser utilizado para encontrar um caminho de um vértice inicial a um vértice final. Para o planejamento de veículos holonômicos é possível utilizar o grafo de navegação gerado, pois a partir de um vértice, através da realização de movimentos de rotação e translação, é possível chegar a qualquer vértice vizinho construindo um caminho até alcançar o objetivo. Em modelos não holonômicos, onde existe restrições na movimentação do veículo, nem todos os vértices vizinhos são válidos nesta modelagem do grafo. Isto acontece pois o modelo depende da sua orientação para determinar os vizinhos que são atingíveis, gerando inconsistências para marcar os vértices já visitados.

Dependendo do modo como a discretização é realizada pode ocorrer algumas situações não favoráveis: se a resolução do *grid* for pequena, apesar de encontrar um caminho mais preciso, a árvore de busca torna-se muito grande podendo inviabilizar o algoritmo com relação ao tempo de execução. Caso a resolução seja grande, a qualidade do caminho encontrado geralmente é pior pois o este passa a ser menos preciso e consequentemente seu comprimento é mais extenso. Tendo em vista que o algoritmo A* caminha ponto a ponto no mapa *grid* até o objetivo, é possível encontrar situações onde o algoritmo não

encontre o caminho mais curto entre dois pontos quaisquer conforme a Figura 15. Nesta Figura, o caminho encontrado pelo algoritmo A*, linha em azul, possui o comprimento de 10.65 metros pois ele é definido célula a célula do ponto inicial ao final. A linha em vermelho mostra a menor distância entre os dois pontos, a distância euclidiana, igual 9.84 metros.

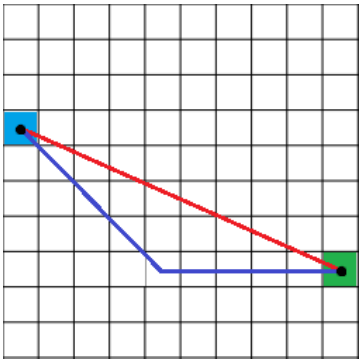


Figura 15: Caminho do algoritmo A* versus segmento de reta

Apesar do PSO descrito em (DUNWEI; LI; MING, 2009) não apresentar a limitação identificada na Figura 15, outra limitação foi detectada. Naquele trabalho, o número de *waypoints* de um caminho é um parâmetro de entrada e os autores sugerem o valor 4. Durante testes com este algoritmo observamos que em mapas complexos com muitos obstáculos, formar um caminho conectando origem ao destino passando pelos quatro *waypoints* sem interceptar obstáculo é inviável. Testes preliminares com valores mais altos para esse parâmetro mostraram que o aumento da geração aleatória de mais *waypoints* onerou o tempo de execução do algoritmo, sem contribuir significativamente para a viabilidade da solução.

No capítulo seguinte é apresentado o algoritmo A*PSO desenvolvido neste trabalho que procura aproveitar as qualidades do A* e do PSO para amenizar as deficiências descritas acima desses mesmos algoritmos.

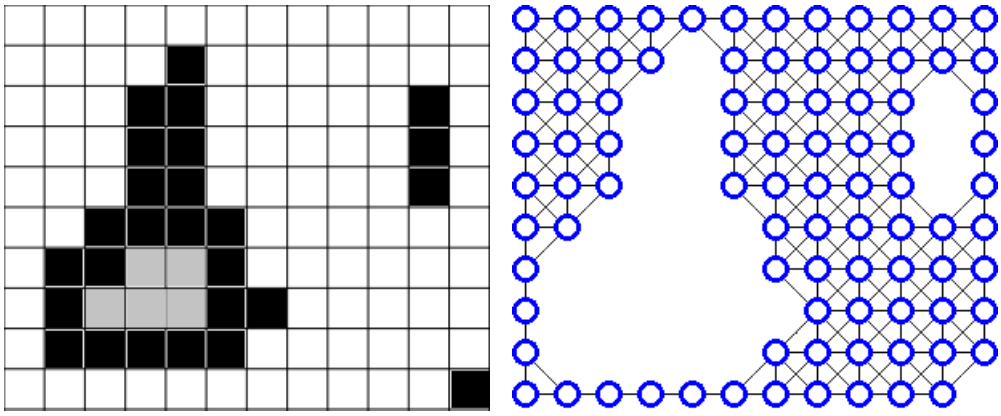
5 *Algoritmo A*PSO*

Neste capítulo é apresentado o algoritmo híbrido A*PSO desenvolvido neste trabalho para resolver o problema de planejamento de caminhos, considerando o modelo de mapa *grid* e de robôs holonômicos e não holonômicos em mapas estáticos.

5.1 Modelagem do mapa

O método *grid* é utilizado para discretizar o mundo real em uma malha bidimensional com resolução $1 \times 1m^2$ onde cada célula da malha é representada por um ponto no plano cujo veículo pode assumir como posição. Este plano então é mapeado em um grafo de navegação onde os pontos correspondem aos vértices deste grafo e as arestas identificam um possível movimento de um vértice para outro no grafo. O custo das arestas é representado pela distância de *manhattan* entre os vértices, neste caso, como a resolução é de $1 \times 1m^2$, o custo de cada aresta pode assumir dois valores: 1 caso o movimento seja horizontal ou vertical e 2 caso o movimento seja diagonal. Os vértices do grafo de navegação possuem a informação da posição (x,y) que o ponto representa no plano.

A Figura 16(a) ilustra um mapa discretizado do tipo *grid* onde os obstáculos são marcados em preto e áreas livres em branco. A Figura 16(b) apresenta o mapa codificado em um grafo onde os círculos azuis representam os vértices válidos (posição que o robô pode assumir) e as arestas são as linhas conectando os vértices. As células obstáculos, conforme Seção 3.1, não são mapeados em vértices do grafo. Esta modelagem será utilizada pelo algoritmo A* neste trabalho e na primeira fase do algoritmo A*PSO desenvolvido. O PSO implementado neste trabalho não necessita codificar o mapa em um grafo. Ele utiliza a formulação matemática disposta na Seção 3.4 para o planejamento de caminhos e tenta otimizar a trajetória inicial definida pelo A* com relação ao seu comprimento.



(a) Mapa representado por uma malha. (b) Codificação do mapa em um grafo.

Figura 16: Codificação do mapa em um grafo;

5.2 O algoritmo PSO para planejamento de caminhos

A partícula do PSO corresponde a uma solução da formulação do problema apresentada na Seção 3.4, ou seja, um sequência de *waypoints* conectando o ponto inicial ao ponto final formando um caminho. Como cada *waypoint* possui 2 dimensões, portanto cada partícula tem $2 \times k$ componentes. Neste trabalho consideramos uma partícula $x_i(t)$ como $x_i(t)$ = ($x_{i1}(t)$,$x_{i2}(t)$,...,$x_{i2k-1}(t)$,$x_{i2k}(t)$) e a solução codificada em *waypoints* $A_1^i(x_{i1}(t),y_{i1}(t))$,...,$A_k^i(x_{ik}(t),y_{ik}(t))$.

O objetivo do PSO consiste em otimizar $x_i(t)$ e para tanto, uma função de avaliação $F(x_i(t))$ é utilizada. Inicialmente $x_i(t)$ é decodificado em *waypoints* $A^i(x_i(t),y_i(t))$ e então o caminho formado passando pelos *waypoints* A_1^i,...,A_k^i é avaliado tendo seu comprimento calculado através da Equação 5.1.

$$F(x_i(t)) = L(\overline{A_0 A_1 ... A_{na+1}}) \tag{5.1}$$

Neste trabalho somente as soluções viáveis perduram no enxame, isto é, se o *waypoint* $A_i^j(x_{ij}(t),y_{ij}(t))$ não estiver em uma área ocupada pelos obstáculos O_l (*l = 0,...m*) e os segmentos de retas formados por $A_i^j(x_{ij}(t),y_{ij}(t))$ e $A_i^{j+1}(x_{ij+1}(t),y_{ij+1}(t))$, *i = 1,2,...,k*; *j = 1,2,...,2k* não interceptarem obstáculos O_l (*l = 0,...l*) então a *fitness* de $x_i(t)$ é calculada de acordo com a Equação 5.1. Caso contrário a partícula é descartada e outra é gerada até que o caminho não intercepte nenhum obstáculo.

Depois de definidas a partícula e a função de avaliação é possível executar o algoritmo PSO para planejamento de caminhos. Dispondo de um mapa *grid*, do ponto inicial e final e dos parâmetros do PSO (número máximo de iterações, $c1$, $c2$, w, tamanho da população) o algoritmo pode ser executado. Inicialmente o PSO gera uma quantidade aleatória

de partículas de acordo com o tamanho da população onde cada *waypoint* $A_i^j(x_{ij}(t),y_{ij}(t))$ recebe valores aleatórios tomados respectivamente nos intervalos [0,*l*] e [0,*a*], onde *l* e *a* são a largura e a altura do mapa. Ao final do processo de geração das partículas o algoritmo verifica sua viabilidade e caso haja colisão com os obstáculos ela é descartada.

Após a inicialização das partículas, o enxame possui um número de caminhos igual ao tamanho da população do enxame e a partícula que possui o menor valor de *fitness* é denominada *gBest*. A partícula possui uma memória onde mantém o seu melhor valor de *fitness* no enxame, o *pBest*. Continuando o algoritmo, a evolução das partículas acontece até que o critério de parada seja atingido. Ao final, a melhor partícula é retornada como o melhor caminho encontrado entre os pontos inicial e final. Os detalhes da implementação do algoritmo são apresentados no decorrer deste capítulo.

O algoritmo proposto e implementado neste trabalho, denotado por A*PSO, combina as estratégias do A* e PSO para planejar um caminho de um veículo em ambientes estáticos. O A*PSO é composto por duas fases, onde a primeira é responsável por construir um caminho utilizando o algoritmo A*. Nesta fase é preciso encontrar um caminho inicial rapidamente para que seja refinado na segunda fase. Antes de passar para a segunda fase, o caminho encontrado pelo A* sofre alguns ajustes. Na segunda fase, o PSO utiliza o caminho inicial ajustado como guia para a criação das partículas do enxame, realizando um planejamento global. Por fim, a fase de pós-planejamento é aplicada aos modelos não holonômicos. Um esquema macro do algoritmo é disposto na Figura 17 e os detalhes das duas fases são apresentados nas subseções seguintes.

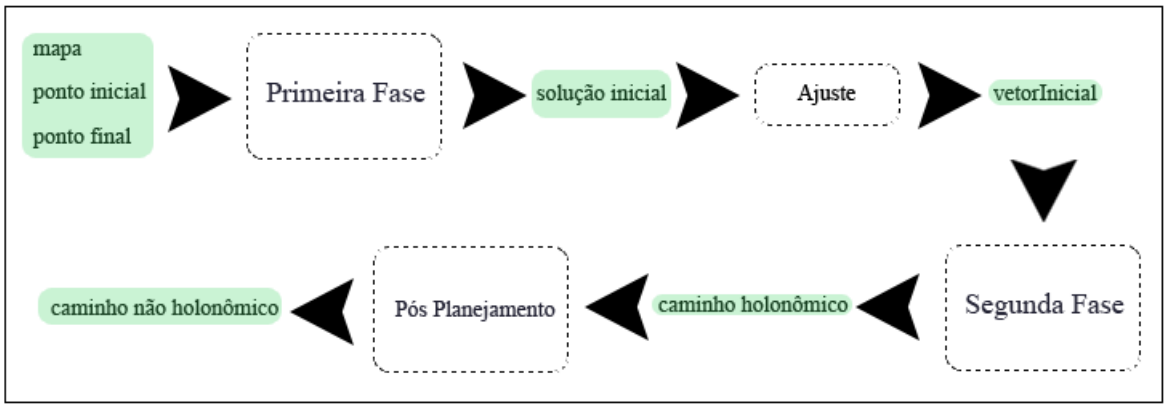


Figura 17: Esquema do algoritmo A*PSO.

5.3 Primeira fase: Algoritmo A*

O módulo A* recebe o mapa na forma de matriz e a transforma em um grafo. Todos os elementos da matriz correspondente ao mapa que não são obstáculos são incluídos no grafo e uma aresta é criada para cada vizinho não obstáculo. Desta forma, modela-se

somente movimentos livres de colisão, de acordo com a Seção 5.1.

Após modelado o ambiente em um grafo e definidos os pontos inicial e final, o algoritmo A* (Algoritmo 1) é executado e retorna um caminho composto por vários *waypoints* (os vértices do grafo) que compõem um caminho vértice a vértice no grafo gerado a partir do mapa conectando o ponto inicial ao ponto final. No exemplo da Figura 18 é exibido um caminho encontrado pelo algoritmo A* tendo como entrada o ponto inicial *S* e final *T*. Sendo $S = A_0$ e $T = A_{k+1}$, o caminho conectando *S* a *T* foi definido com 15 *waypoints* destacados em verde (coloração mais clara).

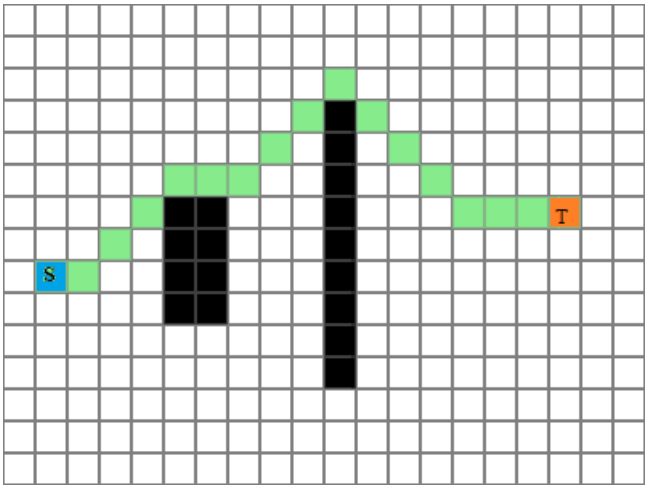


Figura 18: Caminho gerado pelo A*.

5.4 Ajuste do caminho encontrado pelo A*

O caminho encontrado na Figura 18 possui 15 *waypoints* e submetê-lo diretamente para o algoritmo PSO da segunda fase resultaria em um problema de 30 dimensões visto que cada *waypoint* possui duas dimensões, de acordo com Seção 5.2. Aumentar o número de dimensões do problema significa aumentar sua complexidade e consequentemente consumir mais tempo computacional para otimizar o problema, causando um atraso na segunda fase. Tentando resolver este problema, foi criado um procedimento para ajustar o caminho inicial gerado pelo A* visando encontrar *waypoints* estratégicos neste caminho a fim de guiar a criação das partículas do PSO a partir de um número menor de *waypoints* que são armazenados em um vetor chamado de *vetorInicial*.

Como o A* gera caminhos com muitos *waypoints*, um procedimento de descarte foi adotado e consiste em percorrer, ponto a ponto, o caminho definido inicialmente pelo A* em busca de pontos onde haja mudança de direção com relação ao ponto anterior. Foi definido um parâmetro D, do tipo inteiro, com o objetivo de eliminar pontos gerados pelo A* que sejam muito próximos. Desta forma, se a distância entre um ponto onde houve

mudança de direção para outro seja menor que *D*, este é ignorado e o procedimento continua analisando o próximo ponto. Um valor aceitável para *D* corresponde a um percentual p do tamanho da largura do mapa.

O pseudocódigo da função de ajuste do caminho inicial gerado pelo algoritmo A* durante a primeira fase é apresentado em Algoritmo 4. Lembrando que o custo mencionado neste algoritmo é o **custo** da aresta com valores 1 para movimentos horizontais e verticais e 2 para movimentos diagonais.

Algoritmo 4: Função de ajuste

Entrada: Caminho $traj$ do algoritmo A*, parâmetro p, mapa map
Saída: vetorInicial[]

```
   /* função largura(mapa) retorna a largura do mapa          */
1  D = p * largura(map);
2  buffer = -1; j = 0; i = 1;
   /* w corresponde a um waypoint                             */
3  para cada w em traj faça
       /* função custo retorna o custo de um ponto a outro
          adjacente                                            */
4      se buffer ≥ 0 e buffer ≠ custo(traj[i-1],traj[i]) então
           /* função dist retorna a distância de um ponto a outro
              */
5          se dist(traj[i-1],traj[i]) ≥ D então
               /* vetorInicial armazena os pontos onde houve
                  mudança da direção na trajetória oriunda do A*.
                  */
6              vetorInicial[j] ← traj[i];
7              j = j + 1;
8          fim se
9      fim se
10     buffer ← custo(traj[i − 1], traj[i]);
11     i = i + 1;
12 fim para cada
```

Na linha 1 do Algoritmo 4 o parâmetro D é inicializado multiplicando o percentual p por uma função que retorna a largura do mapa. Na linha 2 são inicializadas as variáveis de controle. Da linha 3 a linha 12 é realizada a análise, ponto a ponto, do caminho oriundo do algoritmo A*. Na linha 4 é feito o teste de mudança de direção, definida por um custo diferente da iteração anterior. Na linha 5 é verificado se os pontos possuem uma distância (euclidiana) superior ao parâmetro *D*, caso sim, então este ponto é armazenado no *vetorInicial*. Na linha 10 é atualizado o *buffer*. O *buffer* armazena o valor do custo do movimento anterior, caso o custo do movimento atual seja diferente do *buffer*, então houve mudança na direção.

Observe na Figura 19 a redução no número de *waypoints* gerados pela função de

ajuste que passou de 15 para 5. Pode-se notar que os pontos destacados definem mudança de direção.

5.5 Segunda fase: Algoritmo PSO - Planejamento Global

Ao invés de criar as partículas aleatoriamente conforme Seção 5.2, foi utilizado o vetor *vetorInicial* oriundo do ajuste do caminho obtido pelo A* na primeira fase para guiar a criação das partículas do PSO. Sendo assim, a quantidade de *waypoints* de uma partícula passou a ser determinada de forma dinâmica tendo o tamanho igual ao número de elementos de *vetorInicial*.

Dispondo do vetor *vetorInicial*, a inicialização de uma partícula deixou de ser totalmente aleatória no mapa para ser monitorada. Para isso, foi introduzido o parâmetro *R*, que corresponde a um raio de tamanho igual a um percentual r da largura do mapa aplicado a cada elemento do vetor *vetorInicial*, criando zonas onde os *waypoints* das partículas poderão ser criados aleatoriamente durante sua inicialização. A Figura 19 mostra essas zonas circulares (em azul) cujos centros são os *waypoints* do *vetorInicial* (em laranja).

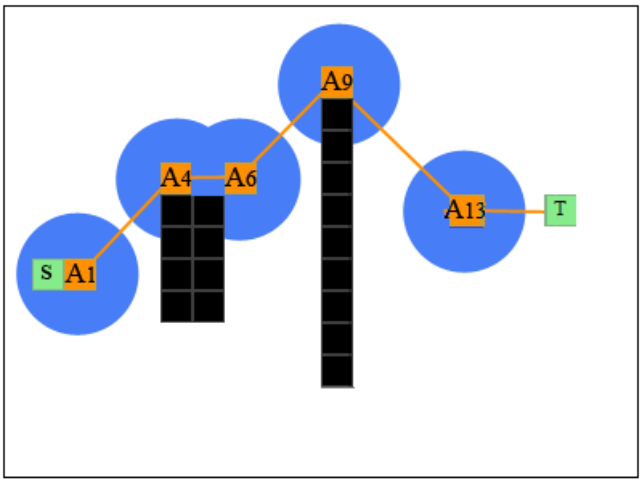


Figura 19: Zonas de criação dos *waypoints*.

A duas fases competem ao algoritmo híbrido A*PSO resolver o problema para modelos holonômicos em mapas estáticos e seu pseudocódigo é apresentado no Algoritmo 5. A próxima seção apresenta como o algoritmo A*PSO realiza a adaptação da trajetória encontrada para os modelos de veículos não holonômicos.

As linhas 1 e 2 contemplam a primeira fase do algoritmo onde é executado o A* e definido o *vetorInicial*. Na linha 3 é inicializado o parâmetro R que define as zonas circulares destacadas na Figura 19. Nas linhas 4 a 7 são inicializadas as partículas guiadas pelo

Algoritmo 5: Algoritmo Híbrido A*PSO.

Entrada: Ponto inicial S, ponto final T, mapa map, tamanho do enxame, número máximo de iterações, c_1, c_2, w, parâmetro r, parâmetro p, função fitness f (Equação 5.1)
Saída: Caminho $trajfinal$

```
1  trajAstar ← aEstrela(S, T, map);
2  vetorInicial ← ajuste(trajAstar, map, p);
   /* função largura(mapa) retorna a largura do mapa          */
3  R = r * largura(map);
4  para cada Partícula faça
5      repita
6          inicializaPartículas(R,vetorInicial);
7      até Todas Partículas do enxame sejam viáveis;
8  fim para cada
9  repita
10     para cada Partícula faça
11         se f(Particula) < f(pBest) então
12             f(pBes)t ← f(Particula);
13         fim se
14         se f(pBest) < f(gBest) então
15             gBest ← pBest;
16         fim se
17     fim para cada
18     para cada Partícula faça
19         repita
20             Atualiza velocidade da Partícula conforme Equação 4.2;
21             Atualiza posição da Partícula conforme Equação 4.3;
22         até Todas Partículas do enxame sejam viáveis;
23     fim para cada
24 até Número máximo de iterações ou não melhoria da solução;
25 trajfinal ← gBest;
```

vetorInicial até que todas partículas do enxame sejam viáveis, ou seja, livres de colisão. Cada partícula é avaliada pela Equação 5.1, linha 11, e caso o valor seja melhor que o atual, então *pBest* é atualizado. Na linha 15, caso haja alguma partícula que possua valor de solução melhor que *gBest*, então *gBest* também é atualizado. Nas linhas 19 à 24 as partículas do enxame são atualizadas até que todas sejam viáveis. Ao final do algoritmo, na linha 26, a partícula com melhor valor de solução *gBest*, ou seja, menor caminho encontrado é retornado.

Os testes de colisão das linhas 7 e 23 são realizados executando o algoritmo de Bresenham. Para isso uma função executa o algoritmo de Bresenham em todos os segmentos de retas que formam o caminho. Se alguma célula destacada pelo algoritmo de Bresenham nos segmentos de retas analisados for uma célula obstáculo então o caminho é considerado inviável. A partícula então é descartada e uma nova será gerada.

5.6 Pós-planejamento - Veículos não holonômicos

O algoritmo A*PSO até agora apresentado resolve o problema de veículos holonômicos em ambientes estáticos. Para adaptação do caminho encontrado para modelos não holonômicos, utilizando a geometria *Ackerman*, uma fase pós-planejamento é necessária. Nesta fase o algoritmo *Reeds e Shepp* (*RS*) é executado recebendo como parâmetros pares de *waypoints* que formam o caminho estabelecido para o modelo holonômico. O resultado desta operação consiste em um conjunto de *waypoints* interligando os pares através de um caminho que respeita as restrições cinemáticas do veículo. Esses novos conjuntos entre pares de pontos são concatenados e ao final do processo é estabelecido um caminho entre a origem e o destino que atende as restrições cinemáticas do veículo.

Como o algoritmo *RS* padrão não verifica se há colisão no caminho encontrado, um teste que verifica se o *waypoint* gerado pelo *RS* está sob um obstáculo foi implementado. Nesta fase, o algoritmo *RS* é executado para o caminho encontrado pelo Algoritmo 5 (A*PSO). Se ao final da execução do algoritmo *RS* não houver nenhuma colisão, tem-se um caminho livre de colisão que atende às restrições de movimentação do veículo. Se nenhum caminho válido for retornado nesta fase, o processo de navegação é reiniciado.

Algoritmo 6: Algoritmo de Pós Planejamento

Entrada: Ponto inicial S, ponto final T, mapa map, tamanho do enxame, número máximo de iterações, c_1, c_2, w, parâmetro r, parâmetro p, parâmetro $raioCurvatura$, parâmetro $tamVeiculo$

Saída: Caminho $trajAckerman$

```
1  trajAstar ← aEstrela(S, T, map);
2  vetorInicial ← ajuste(trajAstar, map, p);
3  trajfinal ← algoritmoA * PSO(S, T, map, parametrosPSO);
4  w_i ← trajfinal.waypoints[0];
5  w_j ← trajfinal.waypoints[1];
6  enquanto w_j não for igual ao destino faça
7      se ChecaRS(w_i, w_j) != 0 então
8          trajAckerman =
           trajAckerman + RS(w_i, w_j, raioCurvatura, tamVeiculo);
9      senão
10         Interrompe algoritmo de pós planejamento
11     fim se
12     i = i + 1;
13     j = j + 1;
14 fim enquanto
```

O Algoritmo 6 exibe como o caminho para veículo holonômico é transformado em

um caminho que respeita a cinemática *Ackerman* é obtido. A função *ChecaRS* na linha 7 retorna 1 caso não haja colisão em um caminho entre os pares de *waypoints* w_i e w_j. Caso isto se confirme, um novo caminho, respeitando o raio de curvatura do veículo e seu tamanho, é definido entre estes dois *waypoints*. Caso haja colisão o algoritmo é interrompido na linha 10. O retorno deste algoritmo é um caminho do ponto *S* ao ponto *T* respeitando a cinemática *Ackerman*.

6 *Resultados e discussões*

Neste capítulo são apresentados os resultados computacionais do algoritmo A*PSO para planejamento de caminhos em mapas estáticos para veículos holonômicos e não holonômicos, além da descrição do ambiente de robótica utilizado para a implementação e execução dos testes nas instâncias criadas. O algoritmo A*PSO é comparado com o PSO e o A* (distância de *Manhattan*) para os modelos holonômicos e com o A* Estado Híbrido, adaptado no *CARMEN* em (GONÇALVES, 2013), para os modelos não holonômicos.

6.1 Ferramenta CARMEN

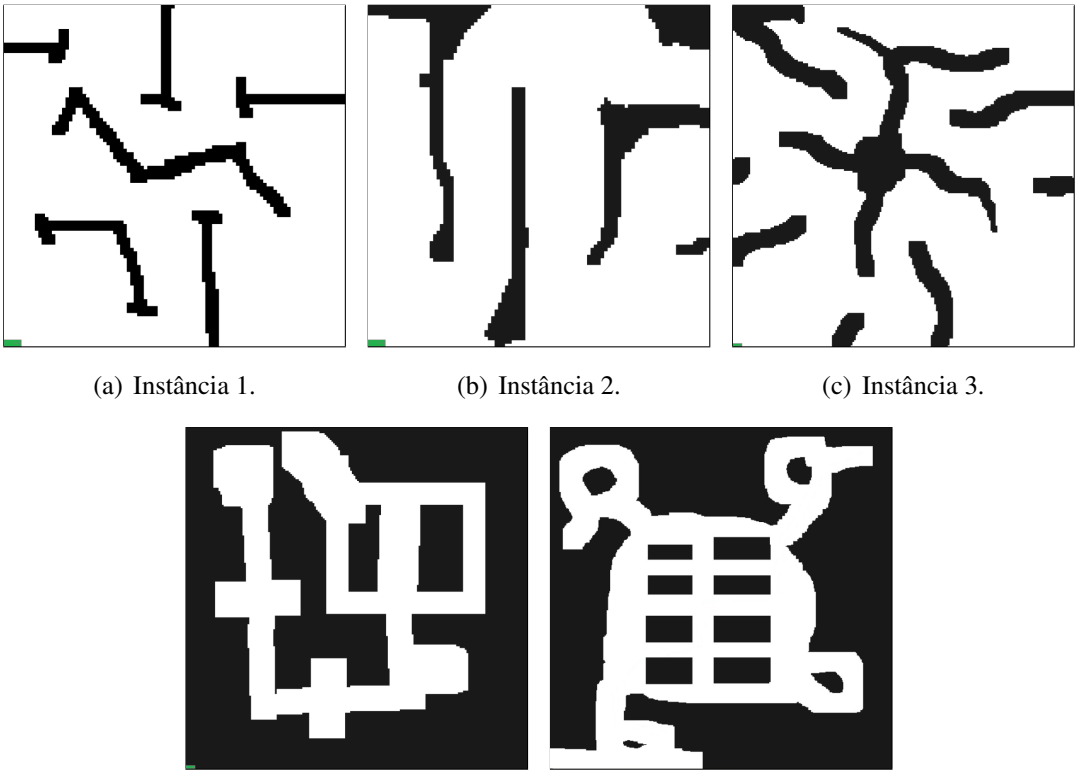
O *CARMEN* é um sistema controlador de robótica *open-source* desenvolvido pela *Carnegie Mellon University* para auxiliar a criação de novos algoritmos para robôs (MONTEMERLO; ROY; THRUN, 2003). Esse sistema foi desenvolvido sobre uma arquitetura modular onde é possível a inclusão de novos módulos de maneira amigável além de dispor de módulos básicos para localização, construção de mapas, navegação, entre outros. Suporta as linguagens C e C++ e está disponível para download em `http://carmen.sourceforge.net/`.

A comunicação entre módulos é feita através de troca de mensagens utilizando o paradigma *publish/subscribe* (DROSOU; STEFANIDIS; PITOURA, 2009). Todas as mensagens trocadas entre os módulos são processadas e distribuídas por um módulo central através de uma assinatura registrada neste módulo. Para que os módulos possam enviar mensagens, primeiramente devem registrá-las no módulo central.

Neste trabalho foi desenvolvido um módulo para o ambiente *CARMEN* que realiza o planejamento de caminhos utilizando o algoritmo proposto A*PSO. A assinatura deste novo módulo são as mensagens de requisição do mapa e pontos inicial e final da trajetória a ser planejada. A requisição do mapa é realizada no início do algoritmo e o retorno desta mensagem consiste em uma matriz bidimensional onde cada elemento corresponde a uma célula do mapa *grid*. Os pontos inicial e final são recebidos pelo algoritmo contendo os valores $x, y \ e \ \theta$ que representam as coordenadas do ponto (x, y) na matriz (plano) e a orientação respectivamente.

6.2 Conjunto de instâncias e cenas

Cinco instâncias foram criadas para realização dos experimentos. Para cada instância foram estabelecidos cinco conjuntos de configuração de entrada (ponto inicial e final) definidas como cenas. Na Figura 20 são apresentadas as instâncias criadas para representar os ambientes estáticos. As regiões em preto representam obstáculos e as regiões em branco são as posições livres onde o veículo pode percorrer ou assumir como posição.



(a) Instância 1. (b) Instância 2. (c) Instância 3.

(d) Instância 4. (e) Instância 5.

Figura 20: Instâncias simulando mapas estáticos.

O resumo das configurações das instâncias é apresentado na Tabela 1. Além das dimensões da instância (primeira e segunda colunas), é mostrado nas colunas seguintes, percentual de células que são obstáculos.

Instância	**Dimensão**	**% de células obstáculos**
1	100x100	11,02
2	100x100	15,87
3	200x200	20,14
4	200x200	49,55
5	500x500	60,14

Tabela 1: Configurações das instâncias com relação aos obstáculos.

Uma cena é definida a partir de uma instância, adicionando-se informações dos pontos inicial e final do caminho. A Tabela 2 exibe cinco cenas respectivamente para cada uma das cinco instâncias. As duas primeiras colunas representam a instância e as cenas daquela instância respectivamente. Na terceira coluna é mostrado o ponto inicial *S* no formato (x,y,θ), onde x e y representam a posição no plano e θ a orientação do veículo. A ultima coluna exibe o ponto final para cada cena. A construção das instâncias foi baseada na observação de estacionamentos e mapas de ambientes já existentes. A cada instância, aumentou-se o grau de dificuldade com relação a dimensão e a quantidade de obstáculos.

Instância	**Cena**	**Ponto Inicial S(x;y;θ)**	**Ponto Final T(x;y;θ)**
1	1	S(5;10;1,570796)	T(90;90;1,570)
	2	S(4; 6;0,000)	T(86; 8;0,000)
	3	S(90;58;-3,141)	T(6;10;-1,570)
	4	S(92;96;-3,141)	T(8;96;-3,141)
	5	S(75; 6;1,570)	T(24;23;1,570)
2	1	S(5;90;-1,570)	T(91;52;1,570)
	2	S(91;52;-1,570)	T(5;90;1,570)
	3	S(90;80;-1,570)	T(90;50;0,000)
	4	S(90;50;0,000)	T(90;80;-1,570)
	5	S(55; 6;1,570)	T(15;13;-3,141)
3	1	S(96;73;0,000)	T(127;191;-3,141)
	2	S(8;166;-1,570)	T(173;14;0,000)
	3	S(10;10;1,570)	T(190;190;0,000)
	4	S(95;13;1,570)	T(48;176;-3,141)
	5	S(100;85;0,000)	T(68;134;0,000)
4	1	S(25;186;0,000)	T(160;190;-3,141)
	2	S(54;145;-3,141)	T(85;130;1,570)
	3	S(150;190;0,000)	T(100;10;-1,570)
	4	S(182; 6;1,570)	T(65;185;-3,141)
	5	S(190;100;1,570)	T(53;148;0,000)
5	1	S(34;332;1,570)	T(437;97;1,570)
	2	S(34;332;1,570)	T(454;458;0,000)
	3	S(28;11;0,000)	T(454;458;0,000)
	4	S(438;96;-3,141)	T(200;12;0,000)
	5	S(167;51;-3,141)	T(455;455;1,570)

Tabela 2: Configurações das cenas.

6.3 Parametrização dos algoritmos A*PSO, PSO e Reeds-Shepp

Nesta seção são apresentados os valores dos parâmetros relativos aos algoritmos A*PSO, PSO e RS, utilizados nos experimentos realizados com todas as instâncias. Esses valores foram definidos empiricamente ou por limitações físicas do veículo e são os mesmos para todos os experimentos.

A Tabela 3 mostra um conjunto de parâmetros que foram testados para analisar os melhores valores com relação a qualidade da solução. Na primeira coluna é identificada a instância. Da segunda à quarta coluna são testados os valores para os parâmetros *c1*, *c2* e tamanho da população (POP) respectivamente. Nas últimas colunas são apresentados o caminho médio (CM) em metros, o tempo de processamento (TE) em segundos e a média do número de iterações (MI) respectivamente nesta ordem.

Instância	c1	c2	POP	CM	TE	MI
	1	1	10	150,111	0,041	97,5
	1	1	20	148,294	0,047	135,0
	1	1	30	147,510	0,053	177,0
	1,6	1,6	10	150,078	0,047	109,0
1	1,6	1,6	20	148,916	0,046	117,5
	1,6	1,6	30	***147,203***	0,048	113,0
	2	2	10	151,050	0,040	82,5
	2	2	20	150,149	0,041	82,5
	2	2	30	148,893	0,050	83,0
	1	1	10	256,456	0,320	75,0
	1	1	20	251,105	0,683	145,0
	1	1	30	248,686	0,934	94,0
	1,6	1,6	10	257,460	0,377	90,0
2	1,6	1,6	20	249,367	0,727	113,5
	1,6	1,6	30	***247,639***	0,971	97,5
	2	2	10	263,506	0,342	69,5
	2	2	20	257,843	0,677	72,0
	2	2	30	255,757	0,980	69,0
	1	1	10	350,781	0,160	80,0
	1	1	20	348,836	0,189	86,0
	1	1	30	340,252	0,215	102,5
	1,6	1,6	10	352,210	0,145	62,5
3	1,6	1,6	20	346,387	0,179	81,0
	1,6	1,6	30	***343,275***	0,226	93,0
	2	2	10	355,564	0,143	71,0
	2	2	20	353,041	0,172	60,5
	2	2	30	351,253	0,219	82,5
	1	1	10	293,296	0,090	94,5
	1	1	20	290,682	0,105	127,0
	1	1	30	***289,616***	0,118	141,5
	1,6	1,6	10	294,510	0,087	61,5
4	1,6	1,6	20	291,415	0,105	98,5
	1,6	1,6	30	290,456	0,128	131,5
	2	2	10	295,063	0,085	81,5
	2	2	20	293,112	0,092	75,5
	2	2	30	292,651	0,116	91,0
	1	1	10	711,224	0,380	108,5
	1	1	20	700,697	0,442	75,0
	1	1	30	693,690	0,559	189,5
	1,6	1,6	10	704,889	0,370	85,0
5	1,6	1,6	20	696,517	0,433	92,0
	1,6	1,6	30	***692,294***	0,525	85,5
	2	2	10	734,253	0,379	47,0
	2	2	20	718,499	0,431	63,0
	2	2	30	713,677	0,540	58,0

Tabela 3: Teste de parâmetros do algoritmo PSO.

Analisando a Tabela 3 constata-se que os valores de *c1* e *c2* iguais a 1,6 com tamanho da população igual à 30 obtiveram a melhor solução em todas as instâncias com relação à distância exceto na instância 4. Com a população acima de 30 indivíduos o algoritmo dispensou muito tempo computacional sem melhora significativa no quesito comprimento de trajetória. Também foram testados os valores de tolerância (número de iterações sem melhoria do valor da solução) iguais à 50 e 100 porém isto não influenciou na qualidade

da solução encontrada. O menor menor valor para o parâmetro de tolerância foi escolhido pois o tempo computacional de execução foi inferior. O número máximo de iterações foi avaliado com valores iguais à 100 e 500 sendo que não houve melhora na solução pois o algoritmo, em 95% dos casos, encerrou pelo critério de tolerância.

A lista completa dos parâmetros com os respectivos valores utilizados estão dispostos na Tabela 4. Os parâmetros c_1 e c_2 recebem o valor 1,6 e correspondem aos coeficientes de aprendizado do algoritmo PSO. O valor de *w*, fator de inércia, decresce linearmente de acordo com o número de iterações, começando com o valor 0,9 e decaindo até 0,2 (a cada passo a diferença entre 0,9 e 0,2 dividido pelo número máximo de iterações). O tamanho da população e o número de iterações são 30 e 500, respectivamente. O parâmetro de tolerância, caso o valor da *fitness* não melhore por um determinado número de iterações foi fixado em 50. O número de dimensões do problema é computado a partir da função de ajuste definida na Seção 5.4. Os valores de *R* e *D* foram obtidos a partir de testes e análises preliminares. Os parâmetros do algoritmo Reeds-Shepp correspondem à limitação física da curvatura das rodas dianteiras do veículo simulado e da largura do veículo respectivamente.

	Parâmetro	**Valor**
A*PSO e PSO	c1	1,6
	c2	1,6
	w inicial	0,9
	w final	0,2
	Tamanho da população	30
	Número máximo de iterações	500
	Tolerância	50
	Dimensão PSO (DUNWEI; LI; MING, 2009)	4
	Dimensão A*PSO	Obtido de acordo com a função de ajuste descrita na Seção 5.4
	R	0,2 * largura do mapa
	D	0,03 * largura do mapa
Reeds-Shepp	Ângulo Máximo da Roda Dianteira	0,46 radianos
	Largura do veículo	2,625 metros

Tabela 4: Parâmetros utilizados nos experimentos.

6.4 Resultados Computacionais

O tempo de execução e o comprimento total do caminho foram os valores analisados nas comparações entre os algoritmos. Todos os experimentos foram executados em um computador Intel Core i5 3.10GHz com 4GB de memória RAM. Para mensurar o tempo de execução é considerado todo o planejamento, desde a transformação do mapa em uma estrutura legível (transformação do mapa em um grafo pelo algoritmo A*) para os algoritmos de planejamento até o cálculo do caminho mínimo do ponto inicial ao final. O tempo de plotagem da trajetória no ambiente *CARMEN* e da exibição dos resultados do planejamento não são considerados.

Por se tratarem de algoritmos estocásticos, o A*PSO e PSO são executados dez vezes para cada cena e as médias dos resultados são armazenadas. Os algoritmos A* (com distância de *manhattan*) e A* Estado Híbrido são executados somente uma vez pois são determinísticos e sempre retornam a mesma solução para cada cena.

6.4.1 Resultados dos experimentos

Nesta seção são apresentados os resultados obtidos dos experimentos realizados nas instâncias. Apresentamos e discutimos a média do tempo de execução, a média do comprimento do caminho, o desvio padrão do comprimento do caminho, o melhor caminho encontrado e o menor tempo de execução obtido para cada modelo de veículo.

6.4.1.1 Modelos holonômicos

Os resultados obtidos com respeito ao comprimento de caminho para os veículos holonômicos são sumarizados na Tabela 5 e seus respectivos tempos de execução, na Tabela 6. A primeira coluna da Tabela 5 identifica cada instância e a segunda, as cenas daquela instância para as quais são realizados os testes. Nas seis colunas seguintes são apresentados os valores (em metros) de comprimento médio (CM), desvio padrão (DP) e comprimento do menor caminho (MC) obtidos respectivamente pelos algoritmos A*PSO e PSO; na nona coluna, o comprimento do menor caminho (MC) obtido pelo A* e finalmente, nas duas últimas colunas, as distâncias relativas dos algoritmos PSO (DR_PSO) e A* (DR_A*) em relação ao A*PSO. A distância relativa foi calculada sempre em relação ao A*PSO e a fórmula utilizada é $(A*PSO - PSO)/A*PSO$ para as comparações com o PSO e $(A*PSO - A*)/A*PSO$ para as comparações com o A*. O símbolo $'-'$ indica que não houve término da execução no tempo limite estipulado em dois minutos.

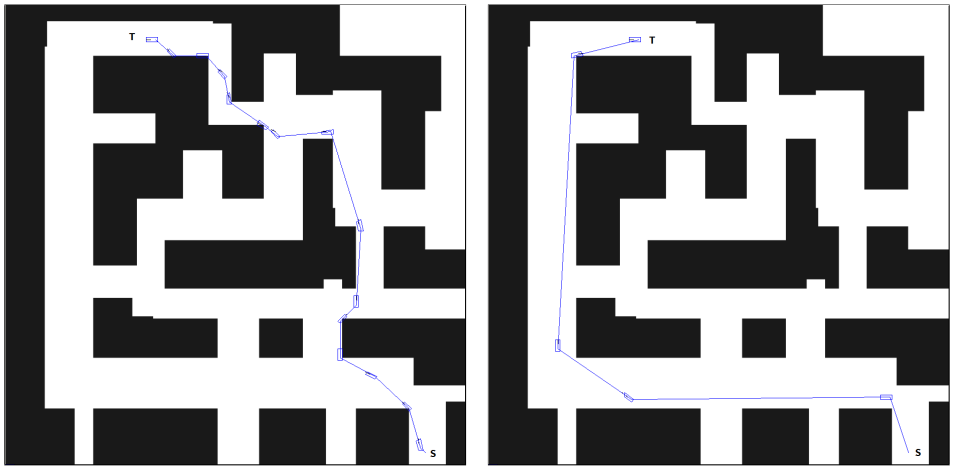
		A*PSO			PSO			A*	Erro Relativo	
Instância	**Cena**	**CM**	**DP**	**MC**	**CM**	**DP**	**MC**	**MC**	**DR_PSO**	**DR_A***
1	1	156,91	4,20	148,05	161,89	6,21	154,09	155,22	-3,17	1,08
	2	124,20	0,92	122,80	136,31	5,85	129,30	126,32	-9,75	-1,71
	3	130,69	2,90	126,66	145,00	5,52	138,94	128,52	-10,95	1,66
	4	104,10	1,20	102,73	109,04	8,36	87,17	107,78	-4,74	-3,53
	5	110,43	2,93	107,17	132,16	4,41	126,63	110,32	-19,67	0,10
2	1	250,60	2,57	247,11	281,76	12,51	260,43	242,06	-12,43	3,41
	2	261,94	8,34	251,77	281,16	12,63	250,03	242,06	-7,34	7,59
	3	118,03	2,70	114,06	132,78	6,96	123,93	116,32	-12,50	1,45
	4	121,59	2,91	117,00	131,64	7,48	121,00	116,32	-8,27	4,33
	5	146,67	1,92	143,29	151,65	5,82	144,83	151,32	-3,40	-3,17
3	1	270,99	9,13	259,84	295,28	16,65	268,30	242,96	-8,96	10,34
	2	274,69	2,29	270,45	301,98	9,26	286,62	281,91	-9,93	-2,63
	3	347,58	4,67	340,17	-	-	-	360,55	NA	-3,73
	4	236,23	5,95	229,33	251,29	7,69	239,48	228,73	-6,37	3,17
	5	220,19	6,89	209,50	239,07	14,19	221,69	211,06	-8,57	4,15
4	1	291,29	6,05	283,40	-	-	-	285,95	NA	1,86
	2	148,44	2,95	145,90	162,26	8,52	147,20	146,12	-9,71	0,83
	3	248,35	3,87	244,20	-	-	-	249,21	NA	-0,44
	4	267,28	3,85	259,50	338,60	4,82	330,70	262,51	-26,26	1,97
	5	263,34	4,89	253,70	-	-	-	234,38	NA	4,52
5	1	474,34	0,08	473,00	489,41	6,61	480,90	482,61	-3,10	-1,73
	2	575,22	12,40	547,79	668,44	26,05	623,59	559,69	-16,09	2,80
	3	718,54	7,93	707,15	-	-	-	718,34	NA	0,04
	4	487,33	9,33	477,62	522,98	22,11	482,90	489,96	-7,34	-0,56
	5	669,92	14,54	652,89	-	-	-	666,72	NA	0,49

Tabela 5: Comprimentos de caminho obtidos pelos algoritmos A*PSO, PSO e A* para veículos holonômicos em instâncias estáticas.

De acordo com as distâncias relativas apresentadas na Tabela 5, podemos observar que todas as soluções do A*PSO foram melhores que as do PSO, diminuindo em média 10% dos comprimentos dos caminhos encontrados. Já quando comparado ao A*, podemos notar que em 60% dos casos, o comprimento do caminho encontrado pelo A* foi melhor, porém os menores caminhos (MC) definidos pelo A*PSO (quarta coluna) apresentam comprimento menor em 76% dos testes. Observamos ainda que o PSO não conseguiu calcular caminhos em algumas instâncias, diferentemente do A*PSO e A* que encontraram solução para todas as instâncias. A falha do PSO em alguns casos se deve ao número reduzido de *waypoints* (apenas 4), sendo difícil construir um caminho livre de colisões.

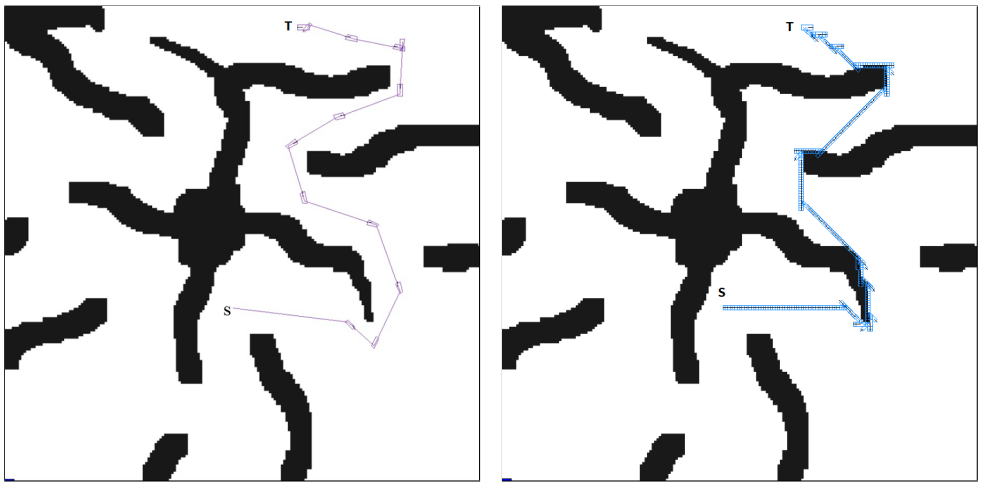
As Figuras 21 a 24 apresentam os caminhos encontrados para algumas instâncias pelos algoritmos A*PSO e PSO respectivamente. A Figura 21 mostra o menor caminho encontrado pelo A*PSO durante as dez execuções para a cena 4 da instância 4 onde obteve-se a maior vantagem (26,26%) quando comparado ao PSO. O experimento onde a maior desvantagem (10,34%) quando comparado com A* foi dado na cena 1 da instância 3 é disposto na Figura 22. Percebe-se que o A* define um caminho muito próximo aos

obstáculos justificando ter alcançado menores caminhos em mais testes que o A*PSO. Na Figura 23 é exibido o caminho onde o A*PSO encontrou a maior vantagem relação ao A* (cena 3 da instância 3). A Figura 24 mostra a instância 4 cena 1 e os caminhos encontrados por A*PSO e A*. Neste caso, o PSO não obteve uma solução.



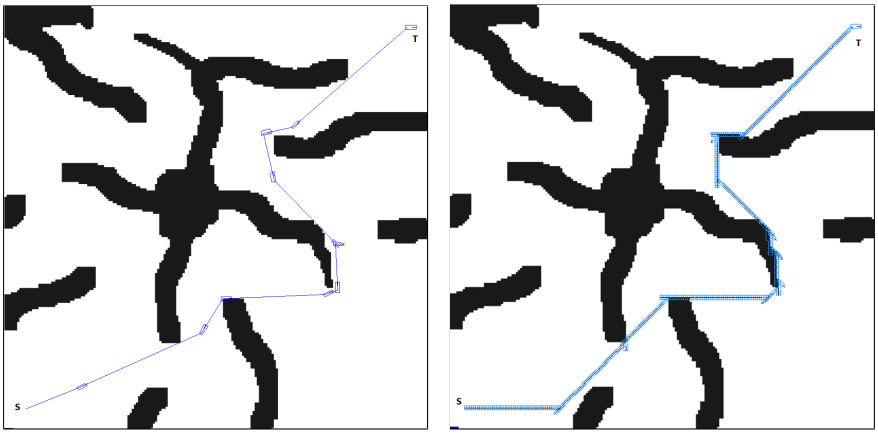
(a) Caminho definido pelo A*PSO. (b) Caminho definido pelo PSO.

Figura 21: Caminhos encontrados na cena 4 da instância 4.



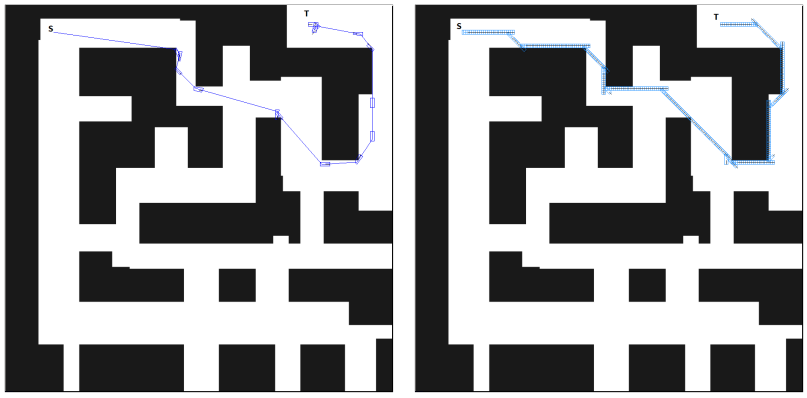
(a) Caminho definido pelo A*PSO. (b) Caminho definido pelo A*.

Figura 22: Caminhos encontrados na cena 1 da instância 3.



(a) Caminho definido pelo A*PSO. (b) Caminho definido pelo A*.

Figura 23: Caminhos encontrados na cena 3 da instância 3.



(a) Caminho definido pelo A*PSO. (b) Caminho definido pelo A*.

Figura 24: Caminhos encontrados na cena 1 da instância 4.

A primeira coluna da Tabela 6 identifica cada instância e a segunda, as cenas daquela instância para as quais são realizados os testes. Nas quatro colunas seguintes são apresentados os valores (em segundos) de tempo de execução (TE) tempo de execução mínimo (TM) obtidos respectivamente pelos algoritmos A*PSO e PSO; na quinta coluna, o tempo de execução (TE) obtido pelo A* e finalmente, nas duas últimas colunas, as distâncias relativas dos algoritmos PSO (DR_PSO) e A (DR_A*) em relação ao A*PSO. A distância relativa foi calculada sempre em relação ao A*PSO e a fórmula utilizada é $(A*PSO - PSO)/A*PSO$ para as comparações com o PSO e $(A*PSO - A*)/A*PSO$ para as comparações com o A*. O símbolo $'-'$ indica que não houve término da execução no tempo limite estipulado em dois minutos.

		A*PSO		PSO		A*	Distância Relativa	
Instância	Cena	TE	TM	TE	TM	TE	DR_PSO	DR_A*
1	1	0,04	0,03	2,41	1,33	0,03	-5122,97	21,42
	2	0,04	0,04	0,10	0,06	0,04	-140,25	10,26
	3	0,04	0,03	0,14	0,08	0,03	-233,16	27,68
	4	0,04	0,03	0,73	0,57	0,03	-1609,18	16,64
	5	0,04	0,04	0,47	0,28	0,04	-994,06	6,20
2	1	0,06	0,04	3,87	2,77	0,03	-6265,84	47,95
	2	0,10	0,08	3,28	2,21	0,03	-3061,26	63,87
	3	0,04	0,03	0,01	0,01	0,03	58,74	15,37
	4	0,04	0,03	0,01	0,01	0,03	54,65	14,75
	5	0,04	0,03	0,04	0,03	0,03	-13,27	28,68
3	1	0,15	0,14	7,37	3,54	0,10	-4590,66	32,57
	2	0,11	0,10	3,02	1,78	0,11	-2524,97	2,59
	3	0,16	0,14	-	-	0,11	-	29,51
	4	0,14	0,13	4,98	3,66	0,12	-3382,61	16,42
	5	0,12	0,12	0,76	0,38	0,11	-511,89	6,28
4	1	0,14	0,11	-	-	0,07	-	51,06
	2	0,07	0,07	9,15	5,04	0,06	-11717,08	11,47
	3	0,19	0,17	-	-	0,07	-	63,39
	4	0,20	0,16	2,19	1,06	0,07	-957,84	65,33
	5	0,12	0,10	-	-	0,07	-	41,61
5	1	0,33	0,32	31,20	19,38	0,31	-9252,26	6,68
	2	0,79	0,64	1,96	1,23	0,31	-147,37	60,63
	3	0,46	0,42	-	-	0,32	-	29,87
	4	0,43	0,42	0,86	0,50	0,35	-98,00	19,34
	5	0,75	0,64	-	-	0,31	-	58,85

Tabela 6: Tempo de execução obtidos pelos algoritmos A*PSO, PSO e A* para veículos holonômicos em instâncias estáticas.

Podemos verificar na Tabela 6 que o tempo de execução do A* foi menor em todos os testes com relação ao A*PSO, cerca de 29% em média. Isto se deu pois o A*PSO executa o algoritmo A* na sua primeira fase para definir uma solução inicial. Em relação ao PSO, o tempo de execução do A*PSO foi maior nas cenas 3 e 4 da instância 2 devido as cenas serem muito simples, diferentemente no restante dos testes onde, em cenas mais complexas, o ganho foi superior a 5000%, como por exemplo na cena 1 da instância 2. Em média, o ganho no tempo de execução foi de 2500%. Isto se deu pois o A*PSO é guiado por uma solução inicial e o PSO gerar muitas trajetórias inviáveis que são descartadas..

6.4.1.2 Modelos não holonômicos

Para estes modelos, o algoritmo desenvolvido A*PSO foi comparado com o algoritmo A* Estado Híbrido e foram utilizados o mesmo conjunto de instâncias e cenas para ambos. O A* Estado Híbrido foi proposto em (DOLGOV et al., 2010) e descreve o algoritmo A* capaz de estabelecer restrições com relação às limitações cinemáticas do veículo. Este algoritmo foi adaptado e implementado no *CARMEN* na dissertação de mestrado de (GONÇALVES, 2013).

Os resultados alcançados para os veículos não holonômicos são enumerados nas Tabelas 7 e 8 para o comprimento de caminho e tempo de execução respectivamente. A primeira coluna da Tabela 7 identifica cada instância e a próxima, as cenas daquela instância para as quais são realizados os testes. Nas três colunas seguintes são apresentados os valores (em metros) de comprimento médio (CM), desvio padrão (DP) e comprimento do menor caminho (MC) obtidos pelo algoritmo A*PSO; na sexta coluna, o comprimento do menor caminho (MC) obtido pelo A* Estado Híbrido e finalmente, na última coluna, a distância relativa do algoritmo A* Estado Híbrido (DR_A*EH) em relação ao A*PSO. A distância relativa foi calculada sempre em relação à média dos valores obtidos pelo A*PSO e a fórmula utilizada é $(A*PSO - A*EH)/A*PSO$.

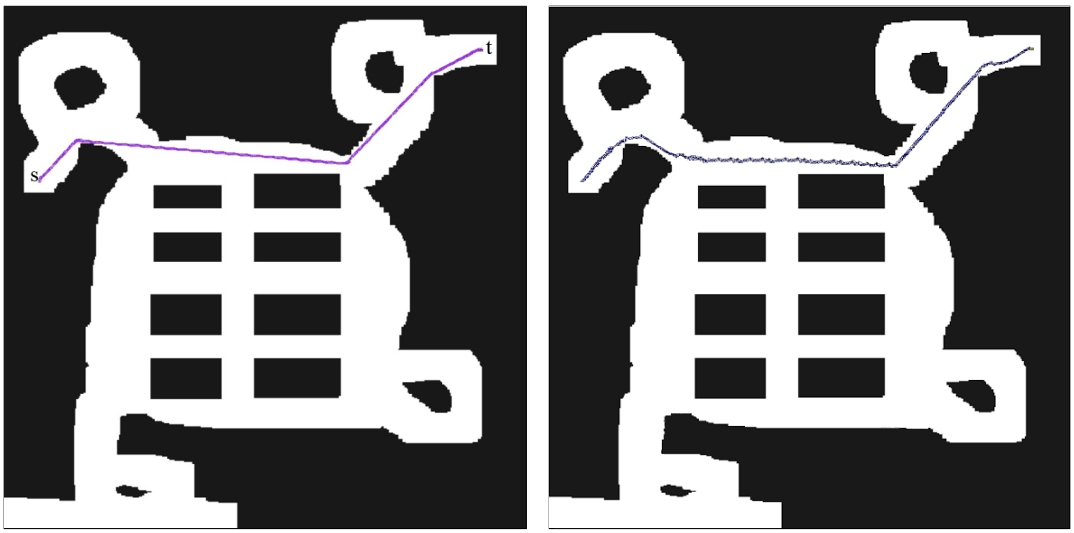
		A*PSO			A* Estado Híbrido	Erro Relativo
Instância	**Cena**	**CM**	**DP**	**MC**	**MC**	**DR_A*EH**
1	1	179,78	6,86	170,30	156,00	13,23
	2	140,21	2,55	135,81	138,00	1,58
	3	152,57	4,17	146,31	134,00	12,17
	4	115,01	2,92	110,51	106,00	7,83
	5	130,86	2,42	125,79	118,00	9,83
2	1	283,64	3,67	277,34	252,00	11,15
	2	308,88	10,38	296,45	249,00	19,39
	3	137,28	4,08	131,88	126,00	8,22
	4	149,48	8,48	138,96	138,00	7,68
	5	161,10	1,96	156,52	153,00	5,03
3	1	317,86	8,77	306,01	243,00	23,55
	2	293,60	5,37	284,43	279,00	4,97
	3	381,50	8,57	364,93	336,00	11,93
	4	254,61	9,97	235,91	228,00	10,45
	5	256,58	10,36	237,30	216,00	15,82
4	1	333,17	8,39	317,21	285,00	14,46
	2	168,61	4,22	162,21	150,00	11,04
	3	272,65	8,77	258,19	252,00	7,57
	4	288,70	4,76	283,97	261,00	9,59
	5	277,27	5,09	271,13	240,00	13,44
5	1	617,42	9,90	600,56	549,00	11,08
	2	480,19	1,25	478,62	486,00	-1,21
	3	751,96	10,22	733,75	678,00	9,84
	4	492,32	4,29	485,12	492,00	0,06
	5	716,23	23,95	678,22	645,00	9,95

Tabela 7: Comprimentos de caminho obtidos pelos algoritmos A*PSO e A* Estado Híbrido para veículos não holonômicos em instâncias estáticas.

Observando-se a Tabela 7, percebe-se que o algoritmo A* Estado Híbrido obteve caminhos menores que o A*PSO, cerca de 9%, em quase todos os experimentos, exceto na cena 2 da instância 5 onde o A*PSO obteve um caminho 1,21% menor. O tamanho do caminho encontrado pelo A*PSO é maior pois este encontra um caminho mais suave e

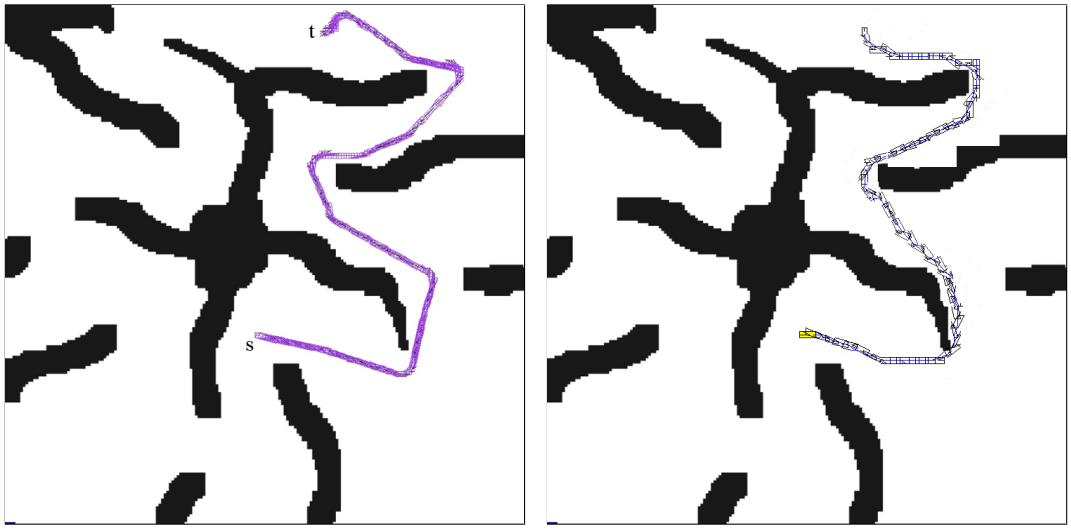
mais seguro com relação aos obstáculos. Vale ressaltar que as maiores diferenças estão nas cenas mais complexas de cada instância.

A Figura 25 mostra o menor caminho encontrado pelo A*PSO durante as dez execuções para a cena 2 da instância 5 onde obteve-se menor comprimento de caminho quando comparado ao A* Estado Híbrido. O caminho encontrado para a cena 1 da instância 3, onde o A*PSO obteve o maior caminho com relação ao A* Estado Híbrido, é disposto na Figura 26. Isto ocorreu neste caso pois o algoritmo A* Estado Híbrido encontrou um caminho mais retilíneo e mais próximo aos obstáculos.



(a) Caminho definido pelo A*PSO. (b) Caminho definido pelo A* Estado Híbrido.

Figura 25: Caminho encontrado pelo A*PSO na cena 2 da instância 5.



(a) Caminho definido pelo A*PSO. (b) Caminho definido pelo A* Estado Híbrido.

Figura 26: Caminhos encontrados na cena 1 da instância 3.

Na Tabela 8 a primeira coluna da identifica cada instância e a próxima, as cenas daquela instância para as quais são realizados os testes. Nas duas colunas seguintes são apresentados os valores (em segundos) de tempo de execução (TE), tempo médio (TM) A*PSO; na quinta coluna, o tempo de execução (TE) obtido pelo A* Estado Híbrido e finalmente, na última coluna, a distância relativa do algoritmo A* Estado Híbrido (DR_A*EH) em relação ao A*PSO. A distância relativa foi calculada sempre em relação à média dos valores obtidos pelo A*PSO e a fórmula utilizada é $(A*PSO - A*EH)/A*PSO$.

		A*PSO		A* Estado Híbrido	Erro Relativo
Instância	**Cena**	**TE**	**TM**	**TE**	**DR_A*EH**
1	1	0,137	0,099	0,028	79,53
	2	0,040	0,024	0,017	57,50
	3	0,075	0,051	0,031	58,72
	4	0,039	0,030	0,017	56,52
	5	0,048	0,034	0,031	34,87
2	1	0,220	0,161	0,056	74,57
	2	0,757	0,505	0,070	90,75
	3	0,023	0,014	0,030	-33,33
	4	0,035	0,018	0,026	26,35
	5	0,027	0,019	0,043	-60,45
3	1	0,293	0,236	0,147	49,81
	2	0,052	0,032	0,111	-114,70
	3	3,500	2,162	0,226	93,54
	4	0,091	0,064	0,132	-44,58
	5	0,058	0,039	0,145	-150,87
4	1	5,298	2,019	0,100	98,11
	2	0,138	0,111	0,032	76,83
	3	5,175	2,359	0,024	99,54
	4	13,904	9,743	0,060	99,57
	5	1,484	0,910	0,107	92,79
5	1	3,720	2,611	0,263	92,93
	2	0,061	0,035	0,212	-247,54
	3	2,023	1,310	0,524	74,10
	4	0,099	0,064	0,379	-282,06
	5	8,144	5,225	0,442	94,57

Tabela 8: Tempo de execução obtidos pelos algoritmos A*PSO e A* Estado Híbrido para veículos holonômicos em instâncias estáticas.

Analisando-se a Tabela 8, verifica-se que o tempo computacional do algoritmo A* Estado Híbrido foi menor em 72% dos casos em média cerca de 16,68%. Isso pode ser explicado pelo fato do A*PSO incluir uma fase que é o A*, porém com uma função heurística distinta (a distância de Manhattan), além da fase de execução do PSO.

7 Conclusões e trabalhos futuros

O objetivo deste trabalho foi desenvolver um algoritmo A*PSO híbrido, utilizando o algoritmo A* como gerador de solução inicial para o PSO, para resolver o problema de planejamento de caminhos em ambientes estáticos aplicados em veículos holonômicos e não holonômicos.

O algoritmo A* foi utilizado na primeira fase do algoritmo desenvolvido em virtude dos estudos previamente realizados e por possuir uma boa performance quando utilizado em mapas do tipo *grid*. O PSO foi escolhido devido a sua simplicidade da modelagem de uma partícula do enxame como um caminho composto por *waypoints* definidos em um mapa bidimensional.

A introdução da primeira fase, onde o algoritmo A* é executado para definição de um caminho inicial, permitiu que o algoritmo A*PSO encontrasse caminhos onde o algoritmo PSO, definido por (DUNWEI; LI; MING, 2009), não obteve solução. Naquele trabalho, o número de *waypoints* era um parâmetro fixo de entrada do algoritmo e em muitas ocasiões, onde a instância era mais complexa em virtude da quantidade e posicionamento dos obstáculos, não foi possível definir um caminho viável mesmo aumentando-se o quantidade de *waypoints*. Percebemos a necessidade de realizar um procedimento de descarte de *waypoints* nesta trajetória inicial definida pelo A* pois a solução possui muitos *waypoints* formando o caminho.

Após o ajuste na etapa de criação inicial da população do enxame do PSO com a utilização de informações obtidas pela trajetória encontrada pelo A*, foi possível obter caminhos 9,92% menores em média com relação ao PSO para os modelos holonômicos em ambientes estáticos. O algoritmo A*PSO obteve, em média, caminhos mais compridos cerca de 1,25% com relação ao A* porém as soluções apresentadas possuem menos *waypoints* e são mais seguras quando observadas a distância do traço da trajetória com respeito aos obstáculos. Houve um aumento no tempo de execução em cerca de 30% quando comparado ao A* pois o algortimo A* foi utilizado como primeira fase do A*PSO (seção 5.3). Já com relação ao PSO, o A*PSO foi em média 2658% mais rápido.

A modelagem de veículos utilizando a geometria *Ackerman* deu-se por ser a mesma utilizada em outros planejadores desenvolvidos no LCAD, como o A* Estado Híbrido.

Em comparações para estes modelos, o A*PSO se demonstrou competitivo visto que, apesar do tamanho dos caminhos encontrados serem maiores em média cerca de 9,95% do que os encontrados pelo A* Estado Híbrido, as soluções encontradas são mais suaves e seguras pois mantêm uma distância maior com relação aos obstáculos. O algoritmo A*PSO, apesar de encontrar boas soluções, ainda precisa de ajustes a fim de eliminar *waypoints* desnecessários, o que por sua vez podem causar a formação de um caminho ruidoso para o veículo, como observado em alguns testes.

Como trabalho futuro pretende-se: (i) melhorar a performance do algoritmo A* para fornecer soluções iniciais em um tempo computacional menor testando diferentes funções heurísticas $h(s)$; (ii) aperfeiçoar o teste de colisão utilizado para verificar a viabilidade da partícula do PSO com relação ao tempo de análise; (iii) considerar o corpo rígido do veículo ao invés de tratá-lo como um ponto no plano; (iv) Otimizar os ajustes dos parâmetros do algoritmo PSO; (v) realizar testes de comparação com mais algoritmos de planejamento de caminhos; (vi) tratar instâncias maiores.

Almeja-se também implementar o algoritmo desenvolvido neste trabalho no veículo autônomo IARA disponível no LCAD (Laboratório de Computação de Alto Desempenho) da UFES. Para isto é necessário que o módulo do *CARMEN* desenvolvido neste trabalho comunique-se com outros módulos do IARA uma vez que foram utilizados apenas os módulos de simulação. Isto posto, os principais aspectos a serem considerados são: (i) adaptar o algoritmo para encontrar trajetórias em pistas; (ii) planejar caminhos em mapas instantâneos gerados periodicamente; (iii) tratar obstáculos móveis.

Referências Bibliográficas

ALAJLAN, M. et al. Global path planning for mobile robots in largescale grid environments using genetic algorithms. In: *Individual and Collective Behaviours in Robotics (ICBR) 2013 International Conference on*. [S.l.: s.n.], 2013. p. 1–8.

ALBA, E. *Optimization techniques for solving complex problems*. [S.l.]: Hoboken, N.J., 2009. (Wiley series on parallel and distributed computing).

ALMEIDA, J. M. S. *Controlo hibrido de robots nao holonomicos*. Dissertação (Mestrado) — Faculdade de Engenharia da Universidade do Porto, 1996.

ASHIRU, I.; CZARNECKI, C.; ROUTEN, T. Characteristics of a genetic based approach to path planning for mobile robots. *J. Netw. Comput. Appl.*, Academic Press Ltd., London, UK, UK, v. 19, n. 2, p. 149–169, abr. 1996. ISSN 1084-8045. Disponível em: <http://dx.doi.org/10.1006/jnca.1996.0012>.

BALCH, T. Avoiding the past: a simple but effective strategy for reactive navigation. In: *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. [S.l.: s.n.], 1993. p. 678–685 vol.1.

BORTOFF, S. A. Path planning for uavs. In: *American Control Conference, 2000. Proceedings of the 2000*. [S.l.: s.n.], 2000. v. 1, p. 364–368.

BRESENHAM, J. E. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, v. 4, n. 1, p. 25–30, 1965.

BUEHLER, M.; IAGNEMMA, K.; SINGH, S. *The 2005 DARPA Grand Challenge: The Great Robot Race*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2007.

COLYER, R. E.; ECONOMOU, J. T. Comparison of steering geometries for multi-wheeled vehicles by modelling and simulation. In: *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*. [S.l.: s.n.], 1998. v. 3, p. 3131–3133.

DOH, N. L.; KIM, C.; CHUNG, W. K. A practical path planner for the robotic vacuum cleaner in rectilinear environments. *IEEE Trans. on Consum. Electron.*, IEEE Press, Piscataway, NJ, USA, v. 53, n. 2, p. 519–527, may 2007.

DOLGOV, D. et al. Path planning for autonomous vehicles in unknown semi-structured environments. *International Journal of Robotics Research*, v. 29, n. 5, p. 485–501, apr 2010.

DROSOU, M.; STEFANIDIS, K.; PITOURA, E. Preferenceaware publish and subscribe delivery with diversity. In: *Proceedings of the Third ACM International Conference on Distributed EventBased Systems*. [S.l.]: ACM, 2009.

DUNWEI, G.; LI, L.; MING, L. Robot path planning in uncertain environments based on particle swarm optimization. In: *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*. [S.l.: s.n.], 2009. p. 2127–2134.

ELFES, A. Occupancy grids: A stochastic spatial representation for active robot perception. In: *Proceedings of the Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-90)*. [S.l.]: AUAI Press, 1990. p. 136–146.

ENGELBRECHT, A. P. *Fundamentals of Computational Swarm Intelligence*. [S.l.]: John Wiley & Sons, 2006. ISBN 0470091916.

FERARIU, L.; CIMPANU, C. Multiobjective hybrid evolutionary path planning with adaptive pareto ranking of variablelength chromosomes. In: *Applied Machine Intelligence and Informatics (SAMI), 2014 IEEE 12th International Symposium on*. [S.l.: s.n.], 2014. p. 73–78.

FERGUSON, D.; M., L.; STENTZ, A. A guide to heuristic based path planning. In: *in: Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems at The International Conference on Automated Planning and Scheduling (ICAPS*. [S.l.: s.n.], 2005.

GONÇALVES, M. A. *Algoritmo A-Estrela de Estado Híbrido aplicado a navegação autônoma de veículos*. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, 2013.

GONG, D.; ZHANG, J.; ZHANG, Y. Multiobjective particle swarm optimization for robot path planning in environment with danger sources. *Journal of Computers*, v. 6, n. 8, p. 1–7, Dec 2011.

GUANZHENG, T.; HUAN, H.; AARON, S. Ant colony system algorithm for realtime globally optimal path planning of mobile robots. *Acta Automatica Sinica*, v. 33, n. 3, p. 279–285, 2007.

HABIB, M.; ASAMA, H. Efficient method to generate collision free paths for an autonomous mobile robot based on new free space structuring approach. In: *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*. [S.l.: s.n.], 1991. p. 563–567 vol.2.

HACHOUR, O. A genetic fpga algorithm path planning of an autonomous mobile robot. In: *Proceedings of the 10th WSEAS International Conference on Mathematical Methods, Computational Techniques and Intelligent Systems*. [S.l.]: World Scientific and Engineering Academy and Society (WSEAS), 2008. p. 66–71.

HART, P.; NILSSON, N.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, v. 4, n. 2, p. 100–107, July 1968.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. [S.l.]: The University of Michigan Press, 1975.

HWANIL, K.; BYUNGHEE, L.; KABIL, K. Path planning algorithm using the particle swarm optimization and the improved dijkstra algorithm. In: *Computational Intelligence and Industrial Application, 2008. PACIIA '08. PacificAsia Workshop on*. [S.l.: s.n.], 2008. v. 2, p. 1002–1004.

HWANG, Y. K.; AHUJA, N. Gross motion planning a survey. *ACM Comput. Surv.*, v. 24, n. 3, p. 219–291, sep 1992. ISSN 0360-0300.

JIANPING, T.; YANG, S. X. Genetic algorithm based path planning for a mobile robot. In: *Robotics and Automation, 2003. Proceedings. ICRA 03 IEEE International Conference on*. [S.l.: s.n.], 2003. v. 1, p. 1221–1226 vol.1.

KARAMAN, S.; FRAZZOLI, E. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, Sage Publications, Inc., Thousand Oaks, CA, USA, v. 30, n. 7, p. 846–894, jun. 2011. ISSN 0278-3649.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*. [S.l.: s.n.], 1995. v. 4, p. 1942–1948 vol.4.

KHATIB, O. Realtime obstacle avoidance for manipulators and mobile robots. In: *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*. [S.l.: s.n.], 1985. v. 2, p. 500–505.

KIBAEK, L.; JONGHWAN, K. Multiobjective particle swarm optimization with preference-based sort and its application to path following footstep optimization for humanoid robots. *Evolutionary Computation, IEEE Transactions on*, v. 17, n. 6, p. 755–766, Dec 2013.

KOENIG, S.; LIKHACHEV, M. Improved fast replanning for robot navigation in unknown terrain. In: *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*. [S.l.: s.n.], 2002. v. 1, p. 968–975.

LATOMBE, J. C. *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991. ISBN 079239206X.

LAVALLE, S. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. [S.l.], 1998.

LAVALLE, S. M.; KUFFNER, J. J. Rapidly-Exploring Random Trees: Progress and Prospects. In: DONALD, B. R.; LYNCH, K. M.; RUS, D. (Ed.). [S.l.]: A K Peters, 2001. p. 293–308.

LEE, J.; KANG, B. Y.; KIM, D. W. Fast genetic algorithm for robot path planning. *Electronics Letters*, v. 49, n. 23, p. 1449–1451, Nov 2013.

LEONARD, J. J. Challenges for autonomous mobile robots. In: *Machine Vision and Image Processing Conference, 2007. IMVIP 2007. International*. [S.l.: s.n.], 2007. p. 4–4.

LIGUO, L. et al. A multiobjective optimization based on adaptive environmental selection. In: *Instrumentation and Measurement, Sensor Network and Automation (IMSNA), 2013 2nd International Symposium on*. [S.l.: s.n.], 2013. p. 999–1003.

LIKHACHEV, M. et al. Anytime dynamic a*: An anytime, replanning algorithm. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. [S.l.: s.n.], 2005. p. 262–271.

LIKHACHEV, M.; GORDON, G.; THRUN, S. Ara*: Anytime a* with provable bounds on suboptimality. In: *IN ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 16: PROCEEDINGS OF THE 2003 CONFERENCE (NIPS03*. [S.l.]: MIT Press, 2004.

LOZANO-PÉREZ, T. Spatial planning: A configuration space approach. *Computers, IEEE Transactions on*, C-32, n. 2, p. 108–120, Feb 1983.

LOZANO-PÉREZ, T.; WESLEY, M. A. An algorithm for planning collisionfree paths among polyhedral obstacles. *Commun. ACM*, ACM, New York, NY, USA, v. 22, n. 10, p. 560–570, oct 1979.

MASEHIAN, E.; SEDIGHIZADEH, D. Classic and heuristic approaches in robot motion planning a chronological review. In: *Proc. World Academy of Science, Engineering and Technology*. [S.l.: s.n.], 2007. p. 101–106.

MONTEMERLO, D.; ROY, N.; THRUN, S. Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (carmen) toolkit. In: *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. [S.l.: s.n.], 2003. v. 3, p. 2436–2441 vol.3.

QIAORONG, Z.; GUOCHANG, G. Path planning based on improved binary particle swarm optimization algorithm. In: *Robotics, Automation and Mechatronics, 2008 IEEE Conference on*. [S.l.: s.n.], 2008. p. 462–466.

RAJA, P.; PUGAZHENTH, S. Optimal path planning of mobile robots: A review. *International Journal of Physical Sciences*, v. 7, n. 9, p. 1314–1320, 2012.

REEDS, J. A.; SHEPP, L. A. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, v. 145, n. 2, p. 367–393, 1990.

RIOS, L. H. O.; CHAIMOWICZ, L. A survey and classification of a* based best-first heuristic search algorithms. In: *Proceedings of the 20th Brazilian Conference on Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer-Verlag, 2010. (SBIA'10), p. 253–262. ISBN 3-642-16137-5, 978-3-642-16137-7.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

SASKA, M. et al. Robot path planning using particle swarm optimization of ferguson splines. In: *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*. [S.l.: s.n.], 2006. p. 833–839.

SEONGMOON, K.; LEWIS, M. E.; WHITE, C. C. Optimal vehicle routing with real-time traffic information. *Intelligent Transportation Systems, IEEE Transactions on*, v. 6, n. 2, p. 178–188, June 2005.

SIEGWART, R.; NOURBAKHSH, I. R. *Introduction to Autonomous Mobile Robots*. Scituate, MA, USA: Bradford Company, 2004. ISBN 026219502X.

SOLANO, J.; JONES, D. I. Generation of collisionfree paths, a genetic approach. In: *Genetic Algorithms for Control Systems Engineering, IEE Colloquium on*. [S.l.: s.n.], 1993. p. 5/1–5/6.

STENTZ, A. The focussed d* algorithm for realtime replanning. In: *In Proceedings of the International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 1995. p. 1652–1659.

SUGIHARA, K.; SMITH, J. Genetic algorithms for adaptive motion planning of an autonomous mobile robot. In: *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*. [S.l.: s.n.], 1997. p. 138–143.

TOMATIS, N. *HYBRID, METRIC TOPOLOGICAL, MOBILE ROBOT NAVIGATION*. Tese (Doutorado) — Escola Politecnica Federal de Lausana, 2001.

VLASSIS, N. et al. Global path planning for autonomous qualitative navigation. In: *Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on*. [S.l.: s.n.], 1996. p. 354–359.

WEBB, D.; BERG, J. van den. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. [S.l.: s.n.], 2013. p. 5054–5061. ISSN 1050-4729.

YUANQING, Q. et al. Path planning for mobile robot using the particle swarm optimization with mutation operator. In: *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*. [S.l.: s.n.], 2004. v. 4, p. 2473–2478 vol.4.

YUHUI, S.; EBERHART, R. C. Empirical study of particle swarm optimization. In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. [S.l.: s.n.], 1999. v. 3.

ZHANG, Y.; GONG, D.; ZHANG, J. Robot path planning in uncertain environment using multiobjective particle swarm optimization. *Neurocomputing*, v. 103, n. 0, p. 172–185, 2013. ISSN 0925-2312.

ZHU, D.; LATOMBE, J. C. New heuristic algorithms for efficient hierarchical path planning. *Robotics and Automation, IEEE Transactions on*, v. 7, n. 1, p. 9–20, Feb 1991.