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

Stéfano Terci Gasperazzo
Universidade Federal do Espírito Santo – UFES
Av. Fernando Ferrari, 514, Goiabeiras – 29.075-910 – Vitória – ES
stefano@ifes.edu.br

Judismar Arpini Junior
Universidade Federal do Espírito Santo – UFES
Av. Fernando Ferrari, 514, Goiabeiras – 29.075-910 – Vitória – ES
solid.judis@gmail.com

Maria Claudia Silva Boeres, Maria Cristina Rangel
Universidade Federal do Espírito Santo – UFES
Av. Fernando Ferrari, 514, Goiabeiras – 29.075-910 – Vitória – ES
{boeres,crangel}@inf.ufes.br

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 dinâmicos. O mundo é discretizado em forma de mapas ladrilhados e cada quadrado representa ou não um obstáculo. 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. O ambiente de simulação de robótica *CARMEN* (*Carnegie Mellon Robot Navigation Toolkit*) foi utilizado para realização de todos os experimentos computacionais considerando cinco mapas gerados artificialmente com obstáculos estáticos ou dinâmicos. A análise dos resultados indicou que o algoritmo PSO híbrido proposto superou em qualidade de solução o PSO convencional, para essas instâncias.
PALAVRAS CHAVE. Planejamento de caminho, A*, PSO híbrido.
Área principal. MH - Metaheurísticas

ABSTRACT

Using autonomous robots able to plan their own path is a challenge which is attracting many researchers in robot navigation area. In this context, this paper aims to implement a hybrid PSO algorithm for path planning in dynamic environments. The world is tiled in the form of discrete maps where each square represents or not an obstacle. The proposed algorithm has two phases: the first one uses the A* algorithm to find a feasible initial path which the PSO algorithm optimizes in the second phase. All computational experiments were carried out over five artificial maps created with rectangular static or dynamic obstacles, using the Carnegie Mellon Robot Navigation Toolkit (CARMEN). The results analysis over these instances pointed out that the proposed hybrid PSO algorithm outperforms in solution quality the PSO canonical version.
KEYWORDS. Path planning, A*, hybrid PSO.
Main area. MH - Metaheuristics

1. Introdução

A crescente utilização de robôs autônomos 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. A navegação de um robô autônomo envolve a percepção do ambiente, localização e construção do mapa a ser explorado, planejamento de rotas, sensoriamento cognitivo e controle dos movimentos [Raja e Pugazhenth(2012)].

O problema de planejamento de caminho consiste em encontrar uma rota livre de colisão de um ponto inicial a um destino satisfazendo algum critério: menor caminho, menor consumo de energia, menor tempo, dentre outros. A maioria das técnicas utiliza o critério de menor caminho [Raja e Pugazhenth(2012)]. O ambiente onde se deseja planejar o caminho pode ser estático, isto é, quando o ponto inicial, o destino e os obstáculos já são conhecidos e não sofrem alterações durante o processo de navegação, configurando-se um planejamento global. No entanto, quando os obstáculos e seus movimentos não são conhecidos, configura-se um planejamento local. Neste caso, a resolução deste problema é mais complexa devido à incerteza do ambiente [Goldman(1994)].

Várias abordagens clássicas para resolver os dois tipos de planejamento foram definidas ao longo dos anos. Estas abordagens encontram boas soluções, porém apresentam um elevado custo computacional e demandam muito tempo [Raja e Pugazhenth(2012)]. Além disso, essas abordagens se tornam obsoletas em virtude da complexidade dos ambientes onde os robôs autônomos estão inseridos. Canny e Reif (1987) definem que o problema passa a ser *NP-hard* na medida em que se aumenta o número de obstáculos. Em ambientes dinâmicos a dificuldade de planejar um caminho é ainda maior [Canny e Reif(1987)].

Sabe-se que resolver problemas da natureza *NP-hard* através de métodos exatos ou exaustivos é inviável devido ao grau de complexidade. Com isso, a utilização de técnicas específicas de heurísticas e meta-heurísticas foram combinadas com os algoritmos clássicos a fim de encontrar uma solução de boa qualidade sem garantia de otimalidade, e então resolver o problema de planejamento de caminhos [Khakmardan et al.(2011)].

Uma abordagem clássica para planejamento de caminhos que utiliza técnicas de heurísticas é o algoritmo A* [Hart et al.(1968)]. Nesta abordagem, um grafo é formado a partir de um mundo bidimensional (como um retângulo), dividido em vários quadrados de mesma dimensão, formando uma malha. O centro de cada quadrado é mapeado em um vértice e cada vértice tem uma aresta com vértices *s'* que foram originados de quadrados adjacentes ao quadrado *s*. Alguns quadrados são obstáculos portanto, não são vértices do grafo. O algoritmo A* explora os vértices do grafo em busca do melhor caminho [Gonçalves(2013)].

O algoritmo de enxame de partículas (*Particle Swarm Optimization* - PSO) é uma metaheurística introduzida por Kennedy e Eberhart [Kennedy e Eberhart(1995)]. Esta técnica é baseada no comportamento social de partículas (possível solução do problema) em um enxame. Cada partícula possui uma velocidade e uma posição e tenta aperfeiçoá-las observando ou imitando partículas melhores durante as iterações do algoritmo. O PSO se mostrou capaz de resolver, de maneira eficiente, vários tipos de problemas complexos. Estudos mostraram como o PSO é competente para resolver o problema de planejamento de caminhos [Mohemmed et al.(2008), Zhao e Yan(2005), Hao et al.(2007)] usando, por exemplo, modelos multiobjetivos que, além de buscar o menor caminho, procuram desviar de regiões perigosas no mapa [Zhang et al.(2013)].

Este trabalho visa desenvolver um algoritmo híbrido utilizando o algoritmo A* como gerador de soluções iniciais para o algoritmo PSO proposto em Gong et al.(2009) sendo implementado e executado no ambiente de robótica *Carnegie Mellon Robot Navigation Toolkit* (*CARMEN*) para solucionar o problema de planejamento de caminhos de robôs diferenciais em mapas dinâmicos. O CARMEN é um *software* controlador de robótica *open-source* desenvolvido pela Carnegie Mellon University para auxiliar a criação de novos algoritmos para robôs [Montemerlo et al.(2003)].

Este artigo está estruturado da seguinte forma: na seção 2 é formulado o problema cujo

trabalho se dispõe a resolver. Nas seções 3 e 4 são discutidos os algoritmos A* e PSO, respectivamente. Na seção 5 é apresentado o algoritmo híbrido desenvolvido. Os testes de simulação e resultados alcançados são apresentados e discutidos na seção 6 e na seção 7, as conclusões e propostas de trabalhos futuros.

2. Definição e modelagem do problema

O problema de planejamento de caminhos tratado neste artigo consiste em: dados dois pontos *S* e *T* como origem e destino, encontrar uma trajetória viável de *S* até *T*, desviando de obstáculos, cujo custo seja o mínimo possível. Neste caso, o custo consiste no tamanho total da trajetória encontrada entre *S* e *T*. Nesta modelagem, todos os obstáculos considerados são retangulares e não se sobrepõem.

O modelo matemático adotado para representar o problema foi aquele proposto em [Gong et al.(2009)] com a diferença que os pontos *S* e *T* não necessitam estar sob o eixo *X* do plano cartesiano. O ambiente modelado é apresentado na figura 1. Podemos observar que cada um dos obstáculos retangulares $O_1, O_2, ..., O_{no}$ tem sua área delimitada por linhas cuja interseção são os pontos $(x_i^1, y_i^1), (x_i^2, y_i^2), (x_i^3, y_i^3), (x_i^4, y_i^4)$, *i* = 1,2,...,*no*, sendo *no* a quantidade de obstáculos no mapa. Alguns pontos no mapa, denominados *waypoints*, fora das áreas ocupadas pelos obstáculos, são escolhidos para formar uma trajetória. Após determinados os *waypoints*, a trajetória será formada pela ligação de *S* ao primeiro *waypoint* e dos *waypoints* subsequentes até *T*. Com isso, teremos *na+1* segmentos de reta formando a trajetória com *na waypoints*. A trajetória é dita viável quando nenhum segmento de reta formado por ela intercepta algum obstáculo.

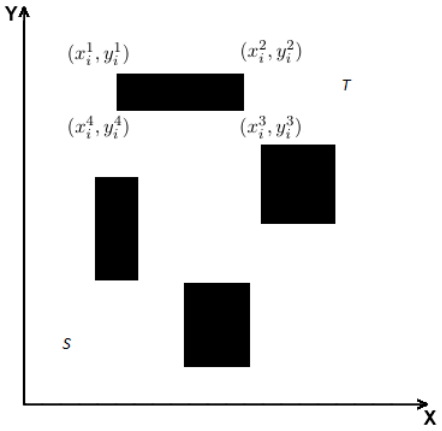


Figura 1: Modelo do ambiente.

Definindo *na waypoints* como as variáveis de decisão, *S* como A_0 e *T* como A_{na+1}, a trajetória formada tem o seguinte formato: $\overline{A_0A_1}, \overline{A_1A_2}, ..., \overline{A_{na}A_{na+1}}$ onde $A_i = (x_i, y_i)$ para todo *i* = 0,...,*na* e $A_0 = (x_s, y_s)$ e $A_{na+1} = (x_t, y_t)$. Com isso podemos definir uma função *L* para somar todos os segmentos de reta que compõem a trajetória, que calcula o comprimento da mesma.

$$\begin{aligned} L(\overline{A_0A_1...A_{na+1}}) &= \sqrt{(x_1 - x_s)^2 + (y_1 - y_s)^2} \\ &+ \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \\ &+ ... + \sqrt{(x_t - x_{na})^2 + (y_{na}^2)} \end{aligned} \quad (1)$$

Como nosso objetivo é minimizar o tamanho do caminho encontrado e, dispondo da função *L* em (1) e das restrições dos obstáculos, podemos formular o problema de caminho mínimo como o seguinte problema de otimização:

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

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

3. Algoritmo A*

O algoritmo A* é classificado na literatura como um algoritmo de busca em grafos, proposto em [Hart et al.(1968)], que visita seus vértices através da exploração de uma árvore de possibilidades guiada por uma função $f(n) = g(n) + h(n)$, calculada para cada vértice n, onde $g(n)$ representa a distância real do caminho traçado do vértice inicial até n e $h(n)$ é uma estimativa do custo de partir de n e chegar ao vértice destino. Cada vértice visitado possui seus adjacentes, ainda não visitados, organizados em uma fila, cuja ordem de inserção é definida de acordo com o valor de $f(n)$ [Russell e Norvig(2009)].

O algoritmo é completo e otimal caso a heurística $h(n)$ seja admissível, ou seja, nunca superestima o custo real de n até o nó objetivo. Um exemplo clássico de heurística admissível no problema de caminho mínimo é a distância euclidiana, como ela é a menor distancia entre dois pontos, o valor de $h(n)$ nunca será maior que o valor real de n até o objetivo [Russell e Norvig(2009)].

Tradicionalmente este algoritmo é usado para determinar o menor caminho entre dois vértices em um grafo [Hart et al.(1968)]. Neste trabalho estamos considerando um caso particular de representação do grafo com seus vértices determinados como pontos no plano. Mais especificamente, o mapa de navegação será representado por uma malha 1x1m^2 onde cada célula representará um vértice e cada aresta apresenta um caminho possível entre vértices vizinhos. As arestas são valoradas pelas distâncias euclidianas entre seus vértices extremos.

4. Algoritmo PSO

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 de animais da natureza. Em especial, o PSO foi inspirado no comportamento de pássaros, insetos e peixes e foi desenvolvido em 1995 pelo psicólogo social James Kennedy e o engenheiro eletricista Russell Eberhart [Kennedy e Eberhart(1995)].

No algoritmo PSO, é definida uma população de partículas (possível solução) e então estas partículas procuram melhorar suas posições 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). 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: número máximo de iterações, número máximo de iterações sem melhora da solução, entre outros. O PSO tenta minimizar ou maximizar uma função objetivo (*fitness*) aplicada sobre um conjunto de variáveis representadas pela própria partícula (solução do problema) [Kennedy e Eberhart(1995)].

O PSO é inicializado com partículas aleatórias e então as posições das partículas 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 e Eberhart(1995)].

As equações 3 e 4 representam na versão clássica do PSO, respectivamente as descrições matemáticas da velocidade e da posição da partícula $x(t + 1)$:

$$v(t+1) = w * v(t) + c_1 * r_1 * (pBest - x(t)) + c_2 * r_2 * (gBest - x(t)) \quad (3)$$

$$x(t+1) = x(t) + v(t+1) \quad (4)$$

Nestas equações, w é um fator de ponderação de inércia que é decrescido durante a execução do algoritmo e garante um balanceamento melhor entre a exploração aleatória e a utilização da informação adquirida pelas partículas [Shi e Eberhart(1999)]. 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 *pBest* é o valor de solução da melhor posição encontrada pela partícula $x(t)$ e *gBest*, o valor de solução da 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 1 é apresentado o pseudo-código do algoritmo clássico do PSO.

Algoritmo 1: *Particle Swarm Optimization*

Entrada: Tamanho do enxame, número máximo de iterações, c_1, c_2, w
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 | | Calcular fitness da Partícula;
7 | | **se** $fitness < pBest$ **então**
8 | | | $pBest \leftarrow fitness$;
9 | | **fim se**
10 | | **se** $pBest < gBest$ **então**
11 | | | $gBest \leftarrow pBest$;
12 | | **fim se**
13 | **fim para cada**
14 | **para cada** *Partícula* **faça**
15 | | Atualiza velocidade da Partícula conforme equação 3;
16 | | Atualiza posição da Partícula conforme equação 4;
17 | **fim para cada**
18 **até** *Número máximo de iterações*;

No algoritmo 1 a inicialização aleatória das partículas é realizado 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 e, no caso afirmativo, 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.

5. Algoritmo híbrido proposto - A*PSO

O algoritmo proposto e implementado neste trabalho, denotado por A*PSO, combina as estratégias do A* e PSO para planejar uma trajetória de um robô em ambiente dinâmico, ou seja, o robô não tem o conhecimento total dos obstáculos no mapa. O A*PSO é composto por três fases, onde a primeira é responsável por construir uma trajetória utilizando o algoritmo A*. Nesta fase é preciso encontrar uma trajetória inicial rapidamente para que seja refinada na segunda fase. Antes de passar para a segunda fase, a trajetória encontrada pelo A* sofre alguns ajustes. Na segunda fase, o PSO utiliza a trajetória inicial ajustada como guia para a criação das partículas do enxame, realizando um planejamento global. Por fim, na terceira fase é realizado o planejamento local, desde que encontrada uma trajetória, o robô começa a se movimentar e verifica se houve alguma alteração de acordo com incertezas inseridas no ambiente. Os detalhes das três fases são apresentados nas subseções seguintes.

5.1. Primeira fase: Algoritmo A*

O mapa no ambiente *CARMEN* corresponde a uma matriz em que cada elemento é uma posição cujo robô pode se mover. Esse elemento pode ser um obstáculo ou não.

O módulo A* recebe o mapa fornecido pelo ambiente *CARMEN* 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.

Após modelado o ambiente em um grafo e definidos os pontos inicial e final, o algoritmo A* é executado e retorna uma trajetória composta por vários pontos (os nós do grafo) que compõem um caminho nó a nó no grafo gerado a partir do mapa em direção ao objetivo. No exemplo da figura 2 podemos ver uma trajetória formada após a execução do algoritmo A*. Esta trajetória é modelada em um vetor cujos elementos determinam uma sequência de *waypoints* que o robô irá se mover até atingir o ponto final. No exemplo da figura 2 foram criados 17 *waypoints* sendo o ponto inicial *S* representado como A_0 e o ponto final *T* como A_{na}, com *na* = 0,...,16.

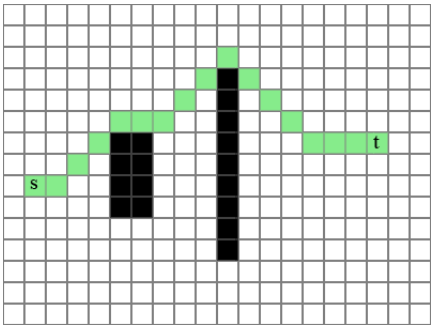


Figura 2: Trajetória gerada pelo A*.

5.2. Ajuste da trajetória do A*

A trajetória encontrada na figura 2 possui 17 *waypoints* e submetê-la diretamente para o algoritmo PSO da segunda fase resultaria em um problema de 34 dimensões visto que cada *waypoint* possui duas dimensões. 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 a trajetória inicial gerada pelo A* visando encontrar pontos estratégicos nesta trajetória a fim de guiar a criação das partículas do PSO a partir de um número menor de *waypoints*.

Como o A* gera trajetórias com muitos pontos, adotamos um procedimento de descarte que consiste em percorrer, ponto a ponto, a trajetória definida inicialmente pelo A* em busca de pontos onde haja mudança de direção com relação ao ponto anterior. Foi definido um parâmetro inteiro para que pontos muito próximos sejam ignorados e não aumente a dimensão do problema, denominado *D*. Caso a distância de um ponto onde houve mudança de direção para outro seja menor que *D*, este é ignorado e o procedimento continua analisando o próximo ponto. Testes empíricos mostraram que um valor aceitável para *D* corresponde a um percentual p do tamanho da largura do mapa. Em todos os testes realizados, adotamos $p = 0.02$.

No exemplo da figura 2, os pontos extraídos da trajetória foram A_1,A_4,A_6,A_9,A_{13} e serão armazenados em um vetor denotado *vetorInit*.

O pseudo-código da função de ajuste da trajetória inicial gerada pelo algoritmo A* durante a primeira fase é apresentado em Algoritmo 2.

Na linha 1 do algoritmo 2 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, da trajetória oriunda do algoritmo A*. Na

Algoritmo 2: Função de ajuste

Entrada: Trajetória $traj$ do algoritmo A*, parâmetro p, mapa map
Saída: vetorInit
/* 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* $\neq$ *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])* $\geq D$ **então**
6     $vetorInit[j] \leftarrow traj[i]$;
7     $j = j + 1$;
8   **fim se**
9 **fim se**
10 $buffer \leftarrow custo(traj[i-1], traj[i])$;
11 $i = i + 1$;
12 **fim para cada**

linha 4 é feito o teste de mudança de direção, definida por um custo diferente da iteração anterior. O custo consiste no valor do movimento realizado, 1 para movimentos horizontais e verticais e $\sqrt{2}$ para movimentos diagonais. Na linha 5 é verificado se os pontos possuem uma distância superior ao parâmetro *D*, caso sim, então este ponto é armazenado no *vetorInit*. 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.

5.3. Segunda fase: Algoritmo PSO - Planejamento Global

A partícula do PSO corresponde a uma solução, ou seja, um trajetória que é composta por *na waypoints*. Como cada *waypoint* possui 2 dimensões, a partícula tem o tamanho de *2na*. Neste trabalho consideramos uma partícula $x_i(t)$ como $x_i(t) = (x_{i1}(t),x_{i2}(t),\ldots,x_{i2na-1}(t),x_{i2na}(t))$ e a solução codificada em *waypoints* $A_1^i(x_{i1}(t),y_{i1}(t)),\ldots,A_{na}^i(x_{ina}(t),y_{ina}(t))$.

O objetivo do PSO consiste em otimizar $x_i(t)$ e para tanto, uma função $F(x_i(t))$ é utilizada. Inicialmente $x_i(t)$ é decodificado em *waypoints* $A_1^i(x_{i1}(t),y_{i1}(t)),...,A_{na}^i(x_{ina}(t),y_{ina}(t))$ que formam uma trajetória e seu comprimento é calculado através da equação 1. A função de avaliação (*fitness*) pode ser definida como:

$$F(x_i(t)) = L(\overline{A_0 A_1 ... A_{na+1}}) \tag{5}$$

Neste trabalho somente as soluções viáveis, ou seja, trajetórias livres de colisões são consideradas e perduram no enxame, isto é, no caso do *waypoint* $A_i^j(x_{ij}(t),y_{ij}(t))$ não está em uma área ocupada pelos obstáculos O_k(*k = 0,...no*) 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,...,na*; *j = 1,2,...,2na* não intercepte os obstáculos O_k(*k = 0,...no*) então a *fitness* de $x_i(t)$ é calculada de acordo com a equação 5. Caso contrário a partícula é descartada e outra é gerada até que a trajetória não intercepte nenhum obstáculo.

Em [Shi e Eberhart(1999)], o número de *waypoints* de uma trajetória é um parâmetro de entrada. Naquele trabalho, os autores utilizaram o valor 4 para esse parâmetro. Durante testes com este algoritmo observamos que em mapas complexos com muitos obstáculos, formar uma trajetória ligando somente 4 *waypoints* que não intercepte obstáculo é inviável. Optamos então por utilizar o vetor *vetorInit* oriundo do ajuste da trajetória obtida 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 *vetorInit*.

Dispondo do vetor *vetorInit*, 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 *vetorInit* criando zonas onde os *waypoints* das partículas poderão ser criados aleatoriamente durante sua inicialização. Em todos os testes realizados, adotamos $r = 0.2$.

Essas duas primeiras fases compõem o algoritmo híbrido A*PSO proposto neste trabalho, cujo pseudo-código é apresentado em Algoritmo 3.

Algoritmo 3: 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
Saída: Trajetória $trajfinal$

```
1  trajAstar ← aStar(S, T, map);
2  vetorInit ← 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          inicializa Partículas;
7      até Todas Partículas do enxame sejam viáveis;
8  fim para cada
9  repita
10     para cada Partícula faça
           /* Função F referente a equação 5                            */
11         fitness ← F(x);
12         se fitness < pBest então
13             pBest ← fitness;
14         fim se
15         se pBest < gBest então
16             gBest ← pBest;
17         fim se
18     fim para cada
19     para cada Partícula faça
20         repita
21             Atualiza velocidade da Partícula conforme equação 3;
22             Atualiza posição da Partícula conforme equação 4;
23         até Todas Partículas do enxame sejam viáveis;
24     fim para cada
25 até Número máximo de iterações ou não melhoria da solução;
26 trajfinal ← gBest;
```

As linhas 1 e 2 contemplam a primeira fase do algoritmo onde é executado o A* e definido o *vetorInit*. Na linha 3 é inicializado o parâmetro R. Nas linhas 4 a 7 são inicializadas as partículas guiadas pelo *vetorInit* até que todas partículas do enxame sejam viáveis, ou seja, livres de colisão. Cada partícula é avaliada pela equação 5, 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 trajetória encontrada é retornada.

5.4. Terceira fase: Planejamento Local

Após a realização do planejamento global executando o algoritmo 3 em que o robô não conhece totalmente o ambiente, duas incertezas são inseridas no ambiente para simular um ambiente dinâmico. Cada obstáculo do ambiente possui um identificador que define a incerteza e seu valor (1, 2 ou 3) é escolhido aleatoriamente antes da execução do planejamento global. Os identificadores 1, 2 e 3 são detalhados a seguir:

1. O obstáculo realmente existe no ambiente porém durante a fase de modelagem ele foi marcado como não existente. Isso poderá resultar na colisão com este obstáculo.

2. O obstáculo não existe no ambiente mas durante a fase de modelagem foi marcado como existente. Isto poderá comprometer o tamanho da trajetória ao desviar de um obstáculo que não existe.

3. Obstáculos que existem em todas as fases do algoritmo.

Ao caminhar pela trajetória global ponto a ponto o robô realiza um teste do ponto atual ao próximo traçando um segmento de reta e analisando se este intercepta algum obstáculo. Se interceptar um obstáculo cujo identificador seja 1, então o robô realiza um planejamento local que consiste em definir sua posição atual como ponto inicial e o próximo *waypoint* viável como ponto final e então executa o algoritmo 3 replanejando este trecho da trajetória. Caso intercepte um obstáculo com identificador igual a 2, o robô define sua posição atual como ponto inicial e traça uma linha reta (menor distância entre dois pontos) para o próximo *waypoint* viável que não intercepte obstáculos, otimizando desta forma a trajetória global. Como a trajetória global já desviou de obstáculos com identificador igual a 3, este trecho da trajetória é mantido.

6. Resultados computacionais

Os testes foram realizados em um computador Intel Core i5 3.10GHz com 4GB de memória RAM. Foram criados cinco mapas para realização dos testes e suas configurações podem ser verificadas na tabela 1. Foram realizados testes comparativos com o algoritmo do artigo [Gong et al.(2009)] e com o desenvolvido neste trabalho, A*PSO.

Mapas	Dimensões(m)	Ponto inicial	Ponto final	Número de obstáculos
(a)	100x100	(5,50)	(95,50)	3
(b)	200x200	(8,115)	(180,60)	9
(c)	200x200	(5,120)	(180,120)	10
(d)	200x200	(5,5)	(195,5)	8
(e)	500x500	(20,20)	(490,490)	14

Tabela 1: Configuração dos mapas

Os parâmetros do algoritmo PSO e do algoritmo híbrido A*PSO foram configurados com os mesmos valores para todos os experimentos. Os parâmetros c_1 e c_2 receberam o valor 1,6. A variável w decresce linearmente de acordo com o número de iterações, começando com o valor de 0,9 e decaindo até 0,2 a cada passo a diferença entre 0.9 e 0.2 divido pelo número máximo de iterações. O tamanho da população é igual a 30 e o número de iterações igual a 500. O parâmetro de tolerância, caso o valor da *fitness* não melhore por um determinado número de iterações, fixado em 80. Estes parâmetros foram definidos a partir de testes e análises preliminares.

Os algoritmos PSO e A*PSO foram executados 10 vezes para cada instância do problema, devido às características de aleatoriedade presentes nestes algoritmos. Os resultados obtidos estão dispostos na tabela 2 e correspondem à média das 10 execuções. A coluna *Melhoria* apresenta quanto, em percentual, o A*PSO conseguiu melhorar ou piorar a solução do PSO. Este resultado foi obtido através da expressão $(APSO - PSO)/PSO) * 100$.

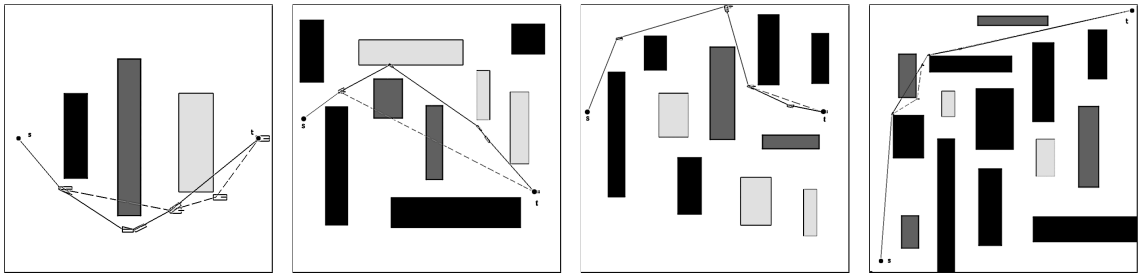
	PSO		A*PSO		Melhoria	
Mapas	c(m)	t(s)	c(m)	t(s)	c(%)	t(%)
(a)	114,764	0,068	108,556	0,107	-5,4	58,1
(b)	206,498	0,061	194,358	0,329	-5,8	444
(c)	290,953	0,064	228,723	0,294	-21,3	356,4
(d)	-	-	545,279	3,231	-	-
(e)	835,817	0,104	798,821	1,193	-4,4	1043,6

c(.):comprimento; t(.):tempo; -:tempo de execução excedido

Tabela 2: Comparativo das trajetórias encontradas pelos algoritmos e tempos computacionais

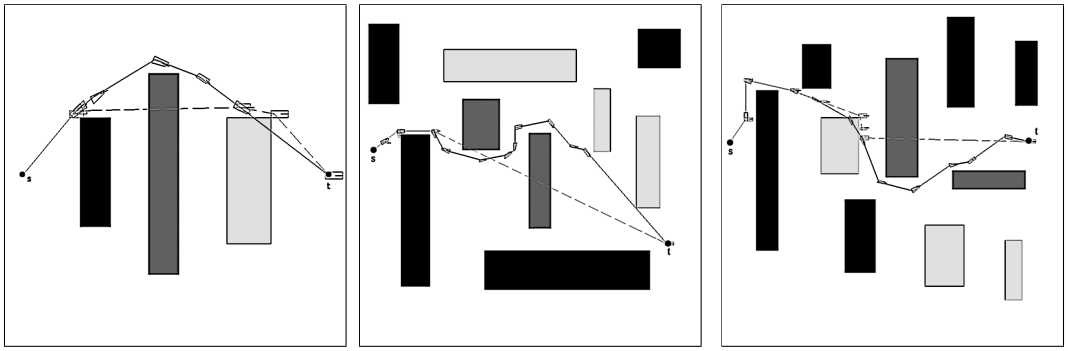
As figuras 3 e 4 mostram respectivamente as melhores trajetórias encontradas dentre as 10 execuções pelos algoritmos PSO e A*PSO nos mapas. Em preto é apresentada a trajetória do planejamento global, onde não se conhece o ambiente por completo. A linha tracejada apresenta

efetivamente a trajetória percorrida pelo robô desviando dos obstáculos efetuando planejamentos locais, se necessário. As legendas em cada mapa indicam respectivamente o custo da trajetória e o seu tempo de execução para o planejamento local conforme apresentados na tabela 2. Os obstáculos em cinza claro representam obstáculos com incerteza 1, em cinza escuro com incerteza 2 e em preto com incerteza 3 (seção 5.4).

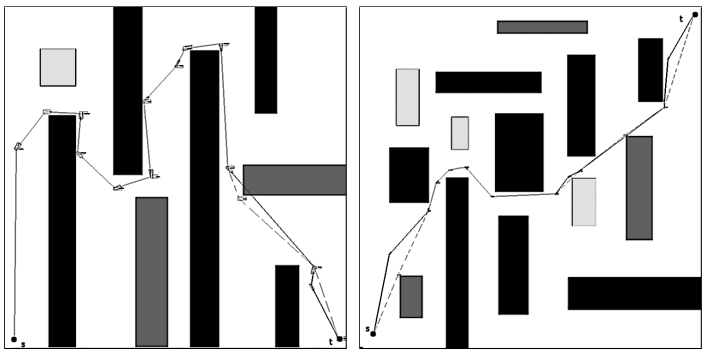


(a) c: 114.764m / t: 0.068s (b) c: 206.498m / t: 0.061s (c) c: 290.953m / t: 0.064s (d) c: 835.817m / t: 0.104s

Figura 3: Melhores trajetórias encontradas pelo algoritmo PSO



(a) c: 108.556m / t: 0.107s (b) c: 194.358m / t: 0.329s (c) c: 228.723m / t: 0.294s



(d) c: 545.279m / t: 3.231s (e) c: 798.821m / t: 1.193s

Figura 4: Melhores trajetórias encontradas pelo algoritmo A*PSO

Os resultados mostram que houve uma diminuição no tamanho da trajetória em todos os mapas analisados, quando comparamos PSO e A*PSO. Em contrapartida houve um aumento no tempo de execução do A*PSO pois esse algoritmo trabalha com problemas de dimensões maiores e consequentemente, necessita de maior tempo de processamento para otimizar a solução (ver tabela 2). O mapa *d* foi criado para simular um labirinto para trazer dificuldade aos algoritmos executados. Podemos observar que o PSO não conseguiu apresentar uma solução viável com apenas quatro *waypoints*. O algoritmo A*PSO, neste mapa, gerou uma solução, porém, com maior tempo de

execução, quando comparado aos resultados das outras instâncias. No mapa *e* vemos claramente a dificuldade do algoritmo PSO desviar de muitos obstáculos estabelecendo então uma trajetória que envolve os obstáculos (figura 3(d)). O A*PSO obteve uma solução melhor por conseguir criar trajetória com maior número de *waypoints* (figura 4(e)).

7. Conclusão e trabalhos futuros

Desenvolvemos um algoritmo híbrido chamado A*PSO utilizando o algoritmo A* como gerador de solução inicial para o PSO. O algoritmo desenvolvido resolveu o problema em mapas complexos em que o PSO não conseguiu encontrar trajetórias. Fazendo uma associação das posições das partículas do PSO com posições reais em que o robô poderia assumir, conseguimos definir trajetórias reais e viáveis com um bom tempo de execução. Após a realização do planejamento global e inserindo incertezas no mapa, simulamos a ação do robô caminhar na trajetória encontrada desviando de obstáculos. Ao se deparar com um obstáculo o robô realizou planejamento local para evitar colisões.

O algoritmo se mostrou bastante eficiente resolvendo o problema em mapas complexos. Isto foi observado ao longo do trabalho através de vários experimentos. O algoritmo A*PSO conseguiu diminuir o tamanho da trajetória em todos os testes e, em mapas onde o PSO não obteve solução, o A*PSO conseguiu obter uma solução.

A limitação dos obstáculos retangulares restringe a utilização destes algoritmos em ambientes mais próximos do mundo real. Como trabalhos futuros, pretendemos estender e adaptar os algoritmos para lidar com outros tipos de obstáculos, de formato livre, por exemplo e com outros tipos de robôs, respeitando a cinemática de outros modelos.

Referências

Canny, J. e Reif, J. (1987). New lower bound techniques for robot motion planning problems. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 49–60.

Goldman, J. (1994). Path planning problems and solutions. In *Aerospace and Electronics Conference, 1994. NAECON 1994., Proceedings of the IEEE 1994 National*, pages 105–108 vol.1.

Gonçalves, M. A. (2013). Algoritmos para navegação autônoma com veículos ackermann. Master's thesis, Universidade Federal do Espírito Santo.

Gong, D., Lu, L., e Li, M. (2009). Robot path planning in uncertain environments based on particle swarm optimization. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2127–2134.

Hao, Y., Zu, W., e Zhao, Y. (2007). Real-time obstacle avoidance method based on polar coordination particle swarm optimization in dynamic environment. In *Industrial Electronics and Applications, 2007. ICIEA 2007. 2nd IEEE Conference on*, pages 1612–1617.

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

Kennedy, J. e Eberhart, R. (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4.

Khakmardan, S., Poostchi, H., e Akbarzadeh-T, M.-R. (2011). Solving traveling salesman problem by a hybrid combination of pso and extremal optimization. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1501–1507.

Mohemmed, A. W., Sahoo, N. C., e Geok, T. K. (2008). Solving shortest path problem using particle swarm optimization. *Applied Soft Computing*, 8(4):1643 – 1653.

Montemerlo, D., Roy, N., e Thrun, S. (2003). 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*, volume 3, pages 2436–2441 vol.3.

Raja, P. e Pugazhenth, S. (2012). Optimal path planning of mobile robots: A review. *International Journal of Physical Sciences*, 7(9):1314–1320.

Russell, S. e Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

Shi, Y. e Eberhart, R. (1999). Empirical study of particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages –1950 Vol. 3.

Zhang, Y., wei Gong, D., e hua Zhang, J. (2013). Robot path planning in uncertain environment using multi-objective particle swarm optimization. *Neurocomputing*, 103(0):172 – 185.

Zhao, Q. e Yan, S. (2005). Collision-free path planning for mobile robots using chaotic particle swarm optimization. In Wang, L., Chen, K., e Ong, Y., editors, *Advances in Natural Computation*, volume 3612 of *Lecture Notes in Computer Science*, pages 632–635. Springer Berlin Heidelberg.