Implementação de um planejador de rotas baseado em ACS para ambientes bidimensionais estáticos utilizando o ROS

Esther de S. Araújo[1], **Anfranserai M. Dias**[2]

[1]Lab. de Robótica e Controle (LARC) – Universidade Estadual de Feira de Santana (UEFS)
Avenida Transnordestina, S/N, Novo Horizonte
Feira de Santana – BA, Brasil – 44036-900

{estheras,anfranserai}@ecomp.uefs.br

Abstract. *The present work describes the implementation of a route planner using artificial intelligence, aiming to automate the process of defining and executing a trajectory in a static two-dimensional environment.*
The route planner was applied to a robot using the Robot Operating System development environment, enabling its navigation process in an obstacle-filled environment. This work addresses aspects related to the use of the Ant Colony System algorithm for route planning, based on a Voronoi Graph, and the adaptation and implementation of this method within the ROS framework.

Resumo. *O presente trabalho descreve a implementação de um planejador de rotas com aplicação de inteligência artificial, de forma a automatizar o processo de definição e execução de uma trajetória em um ambiente bidimensional estático.*
O planejador de rotas foi aplicado a um robô, utilizando o ambiente de desenvolvimento do Robot Operating System, possibilitando seu processo de navegação em um ambiente com obstáculos. Neste trabalho, são abordados aspectos em relação ao uso do algoritmo Ant Colony System para o planejamento de rotas a partir de um Grafo de Voronoi e à adaptação e implementação deste método no ambiente do ROS.

1. Introdução

Robôs autônomos são muito utilizados na execução de tarefas em áreas hostis, que oferecem riscos aos seres humanos, fato que denota a importância da robótica atualmente. Dentre as aplicações, podem ser citadas: a exploração espacial, agricultura e indústria. Para executar uma determinada tarefa, o robô precisa realizar um deslocamento, também chamado de navegação, que é o processo de partir de uma posição inicial para uma final [Latombe 2004]. Tendo em vista que ambientes reais são constituídos de objetos, paredes, móveis e etc, o robô pode colidir com essas partes. Então, faz-se necessário um planejamento de rotas do robô no seu espaço de trabalho, permitindo que ele se desloque evitando colisões. Esse problema foi caracterizado por [Schwartz and Sharir 1983], como "O Problema do Movimento de Pianos".

O planejamento de trajetos eficientes minimiza o tempo de realização da tarefa, e consequentemente, economiza recursos do sistema, como por exemplo a bateria. Métodos de otimização determinísticos como o Dijkstra sempre encontram o menor caminho, entretanto, dependendo do tamanho do grafo, se tornam inviáveis devido ao custo computacional necessário para avaliar todos os caminhos possíveis. Nesse sentido, a aplicação

de algoritmos com abordagens naturais pode proporcionar resultados mais eficientes. Uma dessas abordagens mimetiza o comportamento de uma colônia de formigas, onde indivíduos simples colaboram para resolver problemas complexos de otimização, uma alternativa eficaz aos métodos tradicionais [Anibrika et al. 2020]. Nesse sentido, em [Zhang et al. 2020], é proposta uma otimização colaborativa adaptativa de colônia de formigas com múltiplos papéis dinâmicos, com tratamento de mínimos locais, a qual é aplicada ao planejamento de trajetória de robôs.

No que diz respeito à aplicação de algoritmos naturais em planejamento de rotas, [De Matos and Dias 2013] apresentam um exemplo de implementação do *Ant Colony System*. A proposta apresentada pelos autores consiste em uma aplicação Java que permite ao usuário criar um mapa com obstáculos por meio da interface da aplicação. Em seguida, o usuário marca os pontos de partida e destino do robô. O programa calcula, então, a decomposição do espaço de trabalho (livre de obstáculos), por meio da Triangulação de Delaunay/Voronoi, e esse processo resulta na construção de um grafo que servirá de base para o planejamento da trajetória do robô. Posteriormente, o algoritmo de otimização de rotas baseado na Colônia de Formigas é executado, e retorna a rota sugerida via interface da aplicação.

O propósito fundamental do estudo do presente artigo é realizar a adaptação do planejador desenvolvido anteriormente por [De Matos and Dias 2013] para a plataforma ROS (*Robot Operating System*). No contexto do ROS, o ambiente de trabalho do robô é descrito como uma grade de ocupação (*Occupancy Grid*). As informações presentes na grade de ocupação descrevem a probabilidade de haver obstáculos em cada uma de suas células. O primeiro passo do desenvolvimento visa preencher a grade de ocupação a partir da descrição do mapa obtida pela Triangulação de Delaunay/Voronoi. Após a obtenção e visualização do grafo, é possível definir o alvo do robô e executar o algoritmo de otimização de rotas. A execução do *Ant Colony System* (ACS), retorna então, uma sugestão de rota para o robô, permitindo assim seu processo de navegação.

2. Fundamentação Teórica

Esta seção apresenta conceitos fundamentais para a implementação do planejador de rotas baseado em *Ant Colony System* (ACS) no ambiente do ROS. Além de conceitos necessários para a integração do planejador no processo de navegação de um robô.

2.1. Problema do Caminho Mínimo

O Problema do Caminho Mínimo envolve a busca pelo trajeto mais curto de um ponto de origem até um destino específicos, dado um determinado grafo, com o objetivo de minimizar o custo total do percurso. Dentre os algoritmos mais conhecidos para resolver este problema, está o algoritmo de Dijkstra [Sadavare and Kulkarni 2012]. Entretanto, métodos tradicionais como o Dijkstra apresentam deficiências na resolução de problemas em grafos de grande escala, apresentando complexidade computacional elevada e, consequentemente, um custo computacional inviável. Portanto, a possibilidade de aplicar novas técnicas têm sido continuamente investigada.

2.2. Otimização com Colônia de Formigas

O comportamento social que as formigas desempenham ao procurar por caminhos mais curtos entre o seu ninho e uma fonte de alimento inspirou uma classe de algoritmos da In-

teligência Artificial chamada *Ant Colony Optimization* (ACO) [Dorigo and Stützle 2004]. Inicialmente, cada formiga escolhe aleatoriamente um dos dois caminhos possíveis, como visto na Figura 1. Com o passar do tempo, a aleatoriedade da escolha inicial vai sendo reduzida por conta do caminho mais vantajoso (nesse caso, o mais curto).

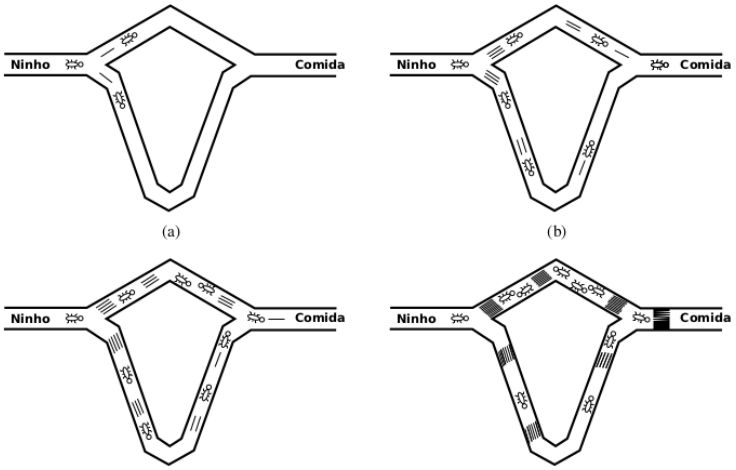


Figura 1: Comportamento das formigas com base no feromônio. [De Matos and Dias 2013]

Como o caminho mais curto tem maior circulação de formigas, este será o caminho que recebe mais feromônios, e as próximas formigas optarão por esse caminho em detrimento do outro.

2.2.1. Ant Colony System

O ACS introduz o conceito de atualização local do feromônio, processo realizado por todas as formigas no fim de cada passo dado no caminho. Cada formiga atualiza a última aresta utilizada com base na seguinte regra [Dorigo et al. 2006]:

$$\tau_{ij}(t+1) = (1-\varphi) \cdot \tau_{ij}(t) + \varphi \cdot \tau_0 \tag{1}$$

onde, $\tau_{ij}(t+1)$ define a quantidade de feromônio que será depositada na aresta (i, j); $\tau_{ij}(t)$ representa a quantidade de feromônio existente na aresta (i, j) no instante t; $\varphi \in (0, 1]$ é o coeficiente de decaimento do feromônio; e τ_0 é o valor inicial do feromônio (presente em todas as arestas no início da execução do algoritmo).

A atualização global do feromônio, ao final do processo de exploração das formigas em uma iteração, se dá da seguinte forma:

$$\tau_{ij}(t+n) = \begin{cases} (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}^{melhor} & \text{se } (i,j) \in \text{melhor caminho} \\ \tau_{ij}(t+n) & \text{caso contrário} \end{cases} \tag{2}$$

$$\Delta\tau_{ij}^{melhor} = \frac{1}{L_{melhor}} \tag{3}$$

onde, $\tau_{ij}(t+n)$ define a quantidade de feromônio que será depositada na aresta (i, j) na iteração n; ρ é um coeficiente em que (1 - ρ) represente a evaporação do feromônio entre t e $t+n$; e L_{melhor} é o comprimento percorrido pela formiga com o melhor resultado. O valor de L_{melhor} depende da decisão do projetista.

Para calcular a probabilidade de escolha de cada nó para ser o próximo passo do *tour*, a estratégia de heurística do ACO tradicional utiliza informações de direção. A probabilidade na qual uma formiga k, localizada no nó i, se move em direção ao nó j é dada por:

$$P_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \cdot [n_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{il}(t)]^{\alpha} \cdot [n_{il}]^{\beta}} & \text{se } j \in N_i^k \\ 0 & \text{caso contrário} \end{cases} \tag{4}$$

onde $\in N_i^k$ define a lista de nós que podem ser visitados pela formiga k; α e β são parâmetros que denotam a relevância da intensidade do feromônio e a distância entre os nós, respectivamente. Ajustes nos valores de α e β são utilizados de forma a enfatizar diferentes resultados, por exemplo: obter caminhos com menos nós (α mais alto e β mais baixo) ou obter caminhos com menor distância (α mais baixo e β mais alto).

Na Equação 5, é definida a função heurística do nó i para o nó j.

$$n_{ij} = \frac{1}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}} \tag{5}$$

Um outro ponto importante a ser citado na implementação do ACS é a regra de decisão utilizada pela formiga durante seu processo de caminhada. Quando localizada no nó i, uma formiga k se move em direção ao nó j de acordo com a regra pseudoaleatória dada por:

$$j = \begin{cases} argmax_{l \in N_i^k} \{[\tau_{il}]^{\alpha} \cdot [n_{il}]^{\beta}\} & \text{se } q \leq q0 \\ P_{ij}^{k}(t) & \text{caso contrário} \end{cases} \tag{6}$$

onde N_i^k são os nós permitidos para a formiga k, localizada no ponto i; q é uma variável aleatória e uniformemente distribuída no intervalo [0,1]; e $q0$ ($0 \leq q0 \leq 1$) é um parâmetro que define a propensão que as formigas têm para exploração (seguir caminhos novos) ou explotação (basear-se apenas nas informações de feromônios já conhecidas).

2.3. Robot Operating System

O *Robot Operating System* (ROS) é um conjunto de bibliotecas de *software* e ferramentas para programação de robôs. O ROS funciona com um padrão *Publisher/Subscriber*, e para tal, utiliza elementos como Nós, Mensagens, Serviços e Tópicos. Para controlar o deslocamento do robô entre dois pontos, o ROS dispõe da Pilha de Navegação. Ela pode ser entendida como um conjunto de nós que auxiliam o robô a se locomover de um ponto a outro, de forma autônoma. A navegação do robô é dividida em planejamento global, que se refere à trajetória a longo prazo, utilizando todo o ambiente; e planejamento local, referente ao desvio de obstáculos dentro de um dado círculo em torno do robô, utilizando apenas informações dos sensores para determinar o ambiente. A pilha se utiliza dos dados de odometria, juntamente com as transformações entre os sistemas de coordenadas de modo a determinar a localização do robô com relação ao ambiente [ROS.org 2022].

3. Planejador de Rotas

O planejador de rotas desenvolvido em um trabalho anterior [De Matos and Dias 2013] foi adaptado ao ROS, como o descrito a seguir.

3.1. Geração de grafos a partir de um mapa do ROS

Os mapas são representações do ambiente no qual o robô opera, onde são incluídas informações como obstáculos e espaços livres, assim como exibido na Figura 2.

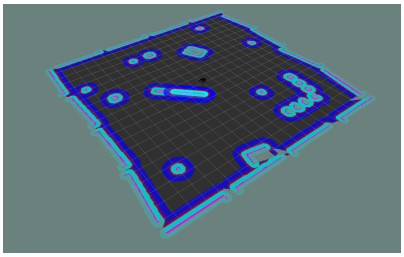


Figura 2: Exemplo de mapa do ROS.

O caminho resultante do planejador de rotas é calculado *offline*. O grafo que representa o mapa é gerado pela decomposição do espaço de Voronoi, que foi implementada utilizando o *metapackage* do ROS `tuw_multi_robot` [Binder et al. 2019]. O Grafo de Voronoi representa caminhos dentro do espaço livre de obstáculos. Os dados das arestas do grafo são publicados no tópico *segments*. Por fim, o planejador de rotas utiliza o algoritmo ACS para otimizar a rota a ser seguida pelo robô dentro deste grafo. A Figura 3 mostra um grafo sobre o mapa base, utilizando a RVIZ, ferramenta de visualização do ROS.

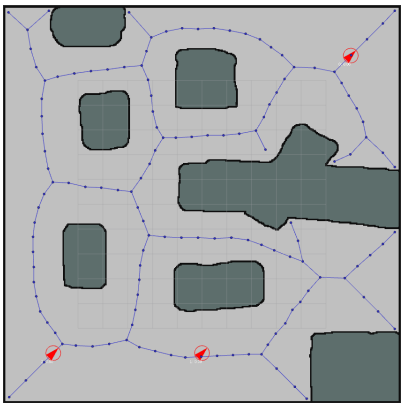


Figura 3: Exemplo de grafo gerado a partir de um mapa do ROS.

3.2. Implementação do ACS

Para a implementação do ACS, foi criado um pacote integrado ao `tuw_voronoi_graph` para acessar os dados do grafo que é gerado a partir de um determinado mapa. O nó responsável por gerar uma rota otimizada pelo ACS está inscrito em três tópicos que possibilitam a execução do algoritmo: *segments* (que possui os dados do grafo gerado pelo pacote `tuw_voronoi_graph`; *pose* (que possui a localização atual do robô) e *clicked_point* (que possui a localização do ponto alvo escolhido pelo usuário). O nó do ACS

recebe os parâmetros de localização do robô e o alvo escolhido pelo usuário, que são conectados por uma aresta ao nó do grafo que esteja mais próximo. Após a execução do ACS, o programa retorna se o alvo foi encontrado, o caminho até o alvo e seu custo em distância.

3.2.1. Caminhos Sem Saída (*Deadlocks*)

Os grafos de Voronoi que são gerados a partir da decomposição do espaço de trabalho são caracterizados como grafos esparsos. A abordagem de exploração do ACS foi inspirada no "Problema do Caixeiro Viajante", onde uma formiga não pode voltar a um ponto que já foi visitado. O método tem como premissa que a formiga explore o máximo de nós possíveis em sua caminhada. Entretanto, por conta do grafo esparso, as formigas podem ficar presas, isto é, sem nenhum nó disponível para seu próximo passo. A Figura 4 mostra o caso no qual a formiga sai do nó A para o B. Nesse cenário, ela não possui nenhum nó vizinho que não tenha sido anteriormente visitado. Portanto, ela se encontra em um caminho sem saída, também conhecido como mínimo local.

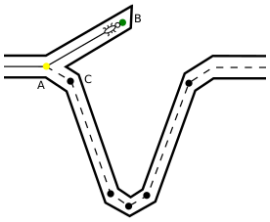


Figura 4: Deadlock. [De Matos and Dias 2013]

Para resolver este problema, foi utilizada a estratégia de retração [Wu et al. 2023, Dai et al. 2019]. Além disso, na implementação proposta, as arestas dos chamados mínimos locais são penalizadas, isto é, os nós que levam a estas arestas são adicionados a uma lista de nós proibidos. Essa lista é visível para todas as formigas, o que impede que as demais sigam por caminhos sem saída já identificados.

4. Resultados

Para os testes de desempenho do ACO, foram postos em comparação dados como a distância total percorrida e a acurácia do algoritmo aplicado em 100 mapas de teste.

4.1. Cenários de Teste

A fim de validar a implementação proposta, foram selecionados 100 mapas usados no trabalho de [De Matos and Dias 2013]. Os mapas do trabalho anterior são arquivos `.map` que possuem a descrição, em formato de texto, da localização dos obstáculos no ambiente. Foi necessário criar um parser para fazer a leitura dos arquivos `.map` e criar novos arquivos do tipo `.pgm` para serem utilizados pelo pacote do ROS que gera os grafos de Voronoi. Dessa forma, foram criados mapas similares aos existentes no trabalho base, com os mesmos tipos de obstáculos, utilizando dimensões de 16x16 (m). O ambiente de testes é composto por uma variedade de mapas, como por exemplo: mapas com obstáculos desconectados de tamanhos iguais (Figura 5a), mapas compostos por paredes com acessos

livres (Figura 5b) e mapas com obstáculos desconectados de tamanhos variados (Figura 5c).

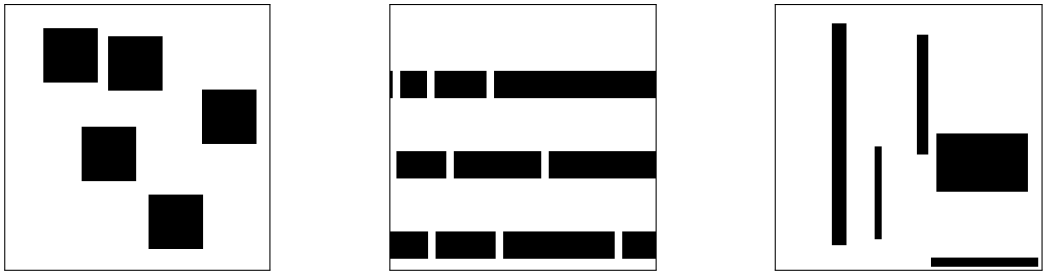


Figura 5: Mapas de teste.

No momento em que os mapas são gerados, também são criados arquivos de configuração para a execução do ACS em cada mapa, com pontos de partida e alvo para o robô. Para cada mapa, esses pontos são gerados aleatoriamente, tendo como única condição pertencerem ao espaço livre de obstáculos.

4.2. Testes

Para a validação do método ACS desenvolvido no presente trabalho, foi implementado um outro método descrito em [Li et al. 2005], adaptado ao ambiente do ROS, com o intuito de fazer uma comparação de desempenho.

Eles apresentaram uma aplicação que utiliza o A* para a solução do grafo. Como o artigo em questão não traz muitos detalhes sobre a implementação do A*, foi feita uma aplicação simples deste algoritmo, realizando a busca no grafo utilizando uma heurística que combina o custo acumulado (distância euclidiana do ponto de partida até o ponto atual) e o custo até a chegada (distância do ponto atual até o alvo), assim como descrito por [Bolc and Cytowski 1992].

Para avaliar a adaptabilidade e capacidade de otimização do método ACS desenvolvido no presente trabalho, foi feita uma análise do desempenho deste com relação ao A* em diferentes tipos de mapas. Neste caso, em mapas com obstáculos dispostos aleatoriamente no ambiente e mapas compostos por paredes de acessos livres (Figura 6). Nos testes foram utilizados os parâmetros da Tabela 1.

Tabela 1: Parâmetros ACS.

Parâmetro	Formigas	Iterações	Repetições	β	α	ρ	φ	τ_0	q0
Valor	50	2	1	2	1	0,1	0,1	1	0,5

Os algoritmos da classe ACO têm um caráter aleatório na escolha de nós da trajetória. Visando diminuir o impacto dessa aleatoriedade nos resultados, foram executados 100 experimentos em um mapa de cada tipo, extraindo o tamanho da trajetória solução a cada execução. Os resultados estatísticos extraídos do ACS implementado no presente trabalho e do A* para diversos ambientes são ilustrados na Tabela 2.

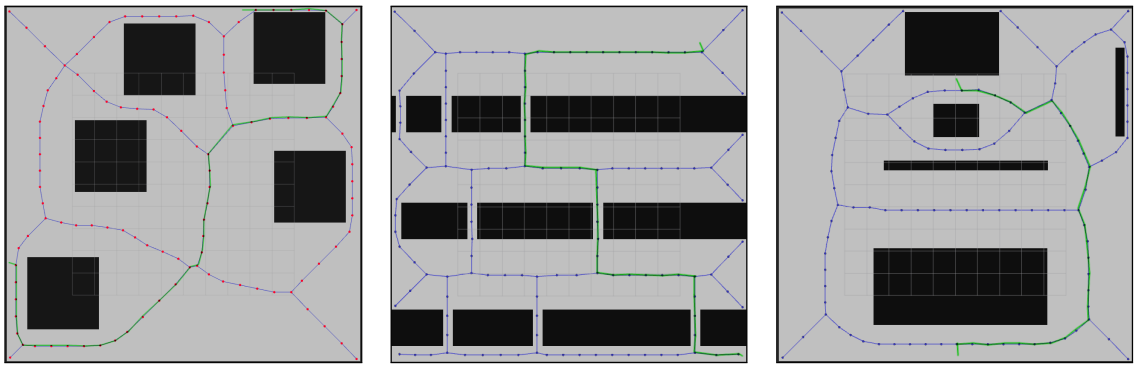
Vale salientar que o A*, aplicado no grafo de Voronoi, falhou na busca pelo alvo em algumas das execuções realizadas nesse teste, como visto na coluna da taxa de acertos

Tabela 2: Comparação do custo da trajetória.

Mapa	Algoritmo	Mais curto (m)	Média (m)	Desvio padrão	Acertos
(a)	A*	20,76	28,90	5,61	100
	ACS	33,21	35,18	1,32	100
(b)	A*	36,61	39,10	1,09	67
	ACS	31,80	42,20	5,99	100
(c)	A*	22,25	26,37	4.45	100
	ACS	22,25	36,75	5,95	100

(Tabela 2). As métricas de média e desvio padrão foram calculadas exclusivamente com base nas execuções em que cada algoritmo conseguiu encontrar o alvo.

A Figura 6 mostra o melhor caminho global gerado pelo ACS em diferentes cenários, dentre os 100 experimentos realizados. No caso do mapa da Figura 6a, o A*



(a) Tamanho fixo **(b) Paredes** **(c) Tamanho variado**

Figura 6: Trajetória do melhor resultado com ACS.

obteve o melhor resultado em termos de comprimento do caminho, entretanto, apresentou também o maior valor de desvio padrão, o que demonstra uma certa instabilidade com relação a solução. Nesse ambiente, o ACS não obteve o melhor desempenho na métrica de comprimento do caminho, entretanto, obteve uma taxa máxima de acerto e um desvio padrão relativamente baixo em comparação ao A*, sendo assim, pode-se argumentar que nesse cenário o ACS se mostrou mais robusto e estável que o outro algoritmo.

No caso dos testes realizados no ambiente do mapa apresentado na Figura 6b, nas métricas de desvio padrão e média de comprimento do caminho o ACS apresentou o pior resultado, entretanto, obteve taxa máxima de acerto, ao contrário do A*. A taxa de acerto do A* nessas condições se mostrou bem abaixo do ACS, sendo capaz de encontrar o alvo em apenas 67% dos testes realizados nesse ambiente. Analisando o desempenho dos algoritmos no mapa da Figura 6c, ambos obtiveram taxa máxima de sucesso, entretanto, o ACS obteve o pior desempenho em termos de custo do caminho.

A validação da rota executada foi feita pela simulação da trajetória otimizada obtida pelo ACS, por meio da ferramenta de visualização do ROS (RViz) e o simulador de robótica 3D (Gazebo). Na simulação, foi utilizado o modelo do Pioneer3AT (Figura 7), disponível no laboratório.

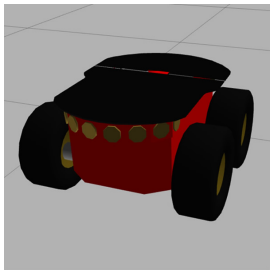
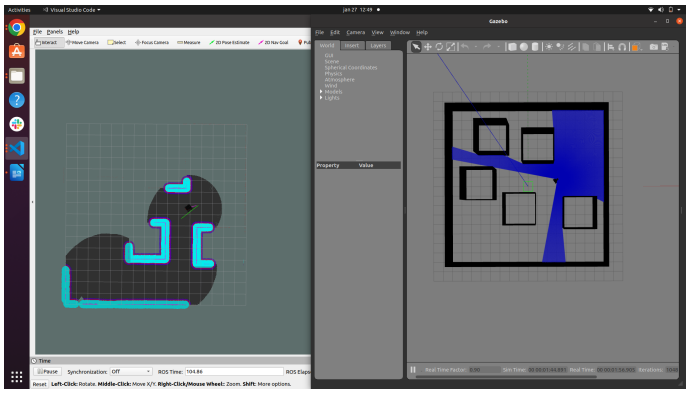
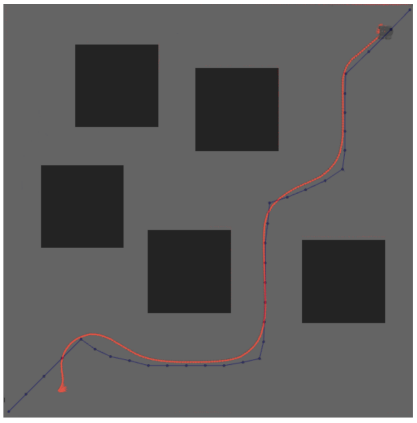


Figura 7: Modelo do Pioneer3AT.

Uma vez determinado o alvo do robô utilizando o botão *Publish Point* na tela do RViz, o ACS é executado e publica as coordenadas de cada ponto da trajetória encontrada. O nó responsável pela navegação recebe essas coordenadas e as envia, uma a uma, para a Pilha de Navegação previamente configurada, utilizando o pacote `move_base`, executando assim a trajetória solução. A Figura 8a ilustra o processo de navegação do robô no ambiente do ROS, enquanto a Figura 8b compara a trajetória planejada, mostrada em azul, com o caminho real percorrido pelo robô, em vermelho.



(a) Navegação do Pioneer3AT.



(b) Trajetória: planejada e realizada.

Figura 8: Navegação e comparação de trajetória.

5. Conclusão

No planejamento de rotas, processo fundamental para a operação de robôs autônomos, é necessário garantir que o robô atue garantindo a própria segurança e a dos demais presentes no ambiente, traçando rotas que o impeçam de colidir com obstáculos. Visando resolver tais dificuldades, neste trabalho foi desenvolvido um planejador de rotas com a aplicação do algoritmo natural ACS, dentro do ambiente do ROS, o que permitiu o processo de navegação de um robô a partir da trajetória obtida.

As simulações realizadas permitiram validar o funcionamento do planejador em conjunto com o navegador do robô, sendo o navegador desenvolvido aplicável a qualquer tipo de robô móvel. No caso da demonstração utilizando um robô real disponível no laboratório, foram desenvolvidas as partes necessárias ao funcionamento do navegador, integradas ao ROS, como por exemplo o processo de deslocamento e a coleta de dados dos sensores (referentes a odometria do robô). Entretanto, não foi possível apresentar no

presente artigo testes com o robô real, devido a problemas nas baterias utilizadas, que não puderam ser resolvidos em tempo hábil. No entanto, é previsto que isso seja solucionado de forma a evoluir o sistema em trabalhos futuros, permitindo assim a realização de testes mais completos.

Referências

Anibrika, B. S., Asante, M., Hayfron-Acquah, B., and Ghann, P. (2020). A survey of modern ant colony optimization algorithms for manet: Routing challenges, perpectives and paradigms. *International journal of engineering research and technology*, 9.

Binder, B., Beck, F., König, F., and Bader, M. (2019). Multi robot route planning (mrrp): Extended spatial-temporal prioritized planning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4133–4139.

Bolc, L. and Cytowski, J. (1992). *Search Methods for Artificial Intelligence*. Academic Press.

Dai, X., Long, S., Zhang, Z., and Gong, D. (2019). Mobile robot path planning based on ant colony algorithm with a* heuristic method. *Frontiers in Neurorobotics*, 13.

De Matos, J. M. A. and Dias, A. M. (2013). Modelagem e implementação de um sistema off-line de planejamento de rotas para ambientes bidimensionais estáticos.

Dorigo, M., Birattari, M., and Stützle, T. (2006). Ant colony optimization-artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. The MIT Press, Cambridge, MA.

Latombe, J. C. (2004). *Robot Motion Planning*. Kluwer Academic Publishers, USA, 8th edition.

Li, Y., Dong, T., Bikdash, M., and Song, Y.-D. (2005). Path planning for unmanned vehicles using ant colony optimization on a dynamic voronoi diagram. pages 716–721.

ROS.org (2022). Wiki: Documentation. `http://wiki.ros.org/`.

Sadavare, A. and Kulkarni, R. (2012). A review of application of graph theory for network. *International Journal of Computer Science and Information Technologies*, 3(6):5296–5300.

Schwartz, J. T. and Sharir, M. (1983). On the piano movers' problem: Iii. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2(3):46–75.

Wu, S., Li, Q., and Wei, W. (2023). Application of ant colony optimization algorithm based on triangle inequality principle and partition method strategy in robot path planning. *Axioms*, 12(6).

Zhang, D., You, X., Liu, S., and Pan, H. (2020). Dynamic multi-role adaptive collaborative ant colony optimization for robot path planning. *IEEE Access*, 8:129958–129974.