FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Coordenação de multi-robots num ambiente industrial

Pedro Henrique Fernandes Moura

DISSERTAÇÃO

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Pedro Gomes da Costa

Coorientador: José Luis Sousa de Magalhães Lima

25 de Junho de 2018

Resumo

A presente dissertação tem como principal objetivo conseguir obter um movimento cooperativo entre todos os robôs integrantes de um sistema multi robótico aplicado a ambiente industrial, evitando a existência de qualquer situação de colisão ou bloqueio mútuo que pudesse provocar um aumento do tempo de execução final das tarefas ou até mesmo a impossibilidade de realizar alguma delas.

Para que o objetivo fosse cumprido, recorreu-se ao algoritmo de planeamento de trajetórias TEA*. Apresentado, na literatura, como um concorrente aos algoritmos de planeamento *offline*, o TEA* confere a possibilidade de planear em tempo real, ao invés de planear por antecipação. Através de diversos replaneamentos permite combater imprecisões na estimação temporal das posições futuras dos robôs envolvidos no sistema, evitando, com maior eficácia, situações de colisão ou deadlock. Focando numa implementação do algoritmo em ambiente real foi ainda desenvolvida uma nova abordagem complementar, com a ambição de obter uma melhor estimação da localização temporal de cada robô, tentando, dessa forma, aproximar as trajetórias reais dos robôs.

Tendo como base o visualizador 3D GLScene, integrado no ambiente de desenvolvimento Lazarus v1.6.4, após realização de testes às diversas soluções implementadas, foi possível observar, temporalmente, os caminhos planeados e validar a implementação do algoritmo. Ainda que os resultados obtidos tenham sido bastante satisfatórios, é de realçar a inexistência de total eficácia no planeamento, tendo sido a mesma comprovada através de dois testes, para os quais uma situação de bloqueio e outra de colisão foram detetadas. Adicionalmente, com recurso ao ambiente de simulação 3D do software SimTwo, foi ainda possível fazer uma análise comparativa entre a modificação implementada ao TEA* e a sua versão inicial, obtendo-se resultados positivos, ao nível dos tempos de execução do sistema, que conferem relevância à aplicação da primeira.

Abstract

The present dissertation aims to obtain a cooperative movement among all the robots integrating a multi robotic system applied to an industrial environment. It is intended to avoid the existence of any situation of collision or mutual blockade that could cause an increase in the final execution time of the tasks or even the impossibility of performing any of them.

In order to achieve the goal, it was used the TEA* trajectory planning algorithm. Presented, in the literature, as a competitor to offline path planning algorithms, TEA* gives the possibility of planning in real time, rather than planning in advance. Through several replanning, it is possible to combat inaccuracies in the temporal estimation of the future positions of the robots involved in the system, avoiding, more effectively, collision or deadlock situations. Focusing on an implementation of the algorithm in the real environment, a new complementary approach was developed, with the ambition to obtain a better estimation of the temporal location of each robot, thus trying to approximate the real trajectories of the robots.

Based on the GLScene 3D visualizer, integrated in the Lazarus v1.6.4 development environment, after testing the various solutions implemented, it was possible to observe the planned paths over time and to validate the implementation of the algorithm. Although the results obtained were quite satisfactory, it should be noted that there was no total efficiency in the planning, which was proved by two failed tests. Hence, a blocking and collision situation were detected. In addition, using the SimTwo 3D simulation environment, it was also possible to make a comparative analysis between the modified TEA* and its initial version, obtaining positive results to the first one in terms of the execution task time of the system. In conclusion, the modified version of TEA* implemented can be a step forward to the optimization of path planning algorithms applied to mobile robots systems.

Agradecimentos

Ao meu orientador, Doutor Pedro Costa, e ao meu coorientador, Doutor José Lima, gostava de agradecer o conhecimento partilhado, o esforço e o acompanhamento regular prestado ao longo do semestre, que tornou possível o desenvolvimento de todo o projeto.

Gostava ainda de agradecer ao professor, Doutor Paulo Costa, pelo conhecimento e dedicação que colocou no projeto. Participando, juntamente com os meus orientadores, de forma ativa na construção dos robôs e desenvolvimento do sistema de localização, permitiu que fosse dado um passo em frente na aplicação do algoritmo TEA* a um ambiente industrial real.

Aos meus colegas e amigos, Pedro Guedes, Gonçalo Silva e Emanuel Pereira um especial obrigado pelos anos de amizade, experiências, apoio e partilha de conhecimento. Graduar e tornar mestre de engenharia nunca se avistou como tarefa fácil, mas com vocês tudo foi mais simplificado.

Aos meus colegas e amigos, Sandro Magalhães e Sérgio Pinto, um importante obrigado pelo companheirismo demonstrado no decorrer desta dissertação. Mesmo quando todos corríamos contra o tempo, demonstraram estar sempre prontos para ajudar.

Aos meus pais, Cristina e António, o maior agradecimento possível, não só pela paciência e compreensão que tiveram ao longo deste último semestre, mas, fundamentalmente, pela contribuição que tiveram em todo o meu percurso de aprendizagem. Obrigado pelas inúmeras experiências que me fizeram crescer e encarar cada desafio com a maior confiança e profissionalismo. Hoje posso dizer que consegui e, em grande parte, a vocês devo esta vitória, construída desde a infância.

Ao meu irmão Afonso tenho a dizer que és e sempre serás o meu orgulho. Mesmo quando o desafio é grande, mesmo quando nada no meu computador fazia sentido para ti, tentaste, na tua inocência, ajudar e contribuir com algo. Se o conhecimento era escasso, por sua vez o apoio e a força que me deste foram incansáveis. Um grande obrigado.

Para a minha amiga e namorada Gabriela, quaisquer palavras seriam poucas para agradecer a contribuição que tiveste no resultado final de todo este trabalho. Das palavras de apoio à confiança demonstrada, da leitura de longos textos à vontade de aprender e discutir robótica, mostraste acreditar sempre no meu trabalho e no meu valor. Mais do que uma boa ajuda, foste o suporte e a força que precisava em cada momento. Um enorme obrigado.

Pedro Moura

Conteúdo

Lista de Figuras

Abreviaturas e Símbolos

AGV	Automated Guided Vehicle
CA*	Cooperative A*
CPRA*	Cooperative Partial-Refinement A*
D*	Dynamic A*
HCA*	Hierarchical Cooperative A*
LED	Light Emitting Diode
LPA*	Lifelong Planing A*
PC	Personal Computer
PRA*	Partial-Refinement A*
RRA*	Rapidly Replanning A*
S-T A*	Spatio-Temporal A*
S-T-W A*	Spatio-Temporal Wait-Near-Collision A*
TEA*	Time Enhanced A*
TOT	Tabela de Ocupação Temporal
UDP	User Datagram Protocol
WHCA*	Windowed Hierarchical Cooperative A*
XML	Extensible Markup Language

Capítulo 1

Análise Introdutória

O capítulo decorrente é utilizado para introduzir a dissertação "Coordenação de multi-robots num ambiente industrial". O mesmo inicia-se com uma contextualização do tema abordado, na secção 1.1, prossegue com uma apresentação da motivação associada ao projeto na secção 1.2 e finaliza com a descrição dos objetivos propostos na secção 1.3.

1.1 Contexto

A indústria vive momentos de grande desenvolvimento tecnológico, nos quais é necessário estar em constante adaptação às exigências do mercado empresarial. Tais exigências advêm essencialmente da forte concorrência que existe e da consequente necessidade de as empresas arranjarem soluções inovadoras para adquirirem valor, as quais nem sempre assentam na valorização direta do produto final. De facto, os custos de produção, na grande maioria das indústrias, têm um impacto relevante e desencadearam o início de um período de evolução aos sistemas automatizados, sistemas estes que conferem a possibilidade de reduzir mão-de-obra assim como otimizar tempos de produção. No mesmo seguimento, foram introduzidos os AGV (Automated Guided Vehicles) em 1955 e desde então a sua evolução tem sido notória e o seu uso tem vindo a crescer de forma significativa [15].

Os AGV são utilizados, fundamentalmente, em sistemas de movimentação de produtos, com destaque no transporte de materiais entre linhas de produção ou setores de armazéns. Ainda assim, a sua aplicação estende-se a diferentes cenários, como por exemplo, hospitais [16], terminais de distribuição [17] e sistemas de manufatura [18]. A questão fundamental que se coloca é até que ponto a utilização destes veículos robóticos em massa será eficiente e produtiva. E de facto, coordenar o movimento de múltiplos robôs em ambientes restritos é uma tarefa bastante complexa cuja evolução não tende a estagnar devido à grande preocupação em melhorar os tempos de execução que surgem aliados à eficiência dos movimentos planeados para cada robô. Nessa sequência aplica-se, portanto, o estudo de algoritmos de planeamento de trajetórias, que, a partir das localizações dos robôs medidas e/ou estimadas sejam capazes de controlar o tráfego de uma

determinada frota, planeando caminhos seguros e otimizados, que atendam fundamentalmente à importância de evitar colisões e situações de bloqueio mútuo, tais como *deadlocks* e *livelocks*.

1.2 Motivação

Os sistemas multi-robô estão em constante evolução e, atualmente, um dos principais focos de investigação incide no planeamento de trajetórias que assegurem uma coordenação eficaz e ótima entre todos os AGV envolvidos. Como tal, diversas abordagens têm vindo a ser propostas e implementadas com vista a obtenção de melhorias sucessivas no que diz respeito, essencialmente, ao tempo de execução médio e final do sistema bem como ao número de conflitos existentes. Nesse mesmo contexto, em [11] e [14] é proposto o Time Enhanced A* (TEA*), um algoritmo desenvolvido com o propósito de permitir ao sistema multi-AGV evitar situações de colisão e bloqueio mútuo, através da introdução de noção temporal ao planeamento, tendo como base o algoritmo de pesquisa A*. Tendo em consideração que a sua aplicação ainda não permite a resolução da totalidade dos casos de gestão de bloqueios mútuos, move este projeto a procura por uma eficácia crescente e pela otimização da eficiência do algoritmo.

1.3 Objetivos

Pretende-se com o desenvolvimento desta dissertação a aplicação do algoritmo de pesquisa TEA* a problemas de coordenação em sistemas multi-robóticos.

O principal objetivo passa por conseguir obter um movimento cooperativo entre todos os robôs envolvidos no sistema, evitando a existência de qualquer situação de colisão ou bloqueio mútuo que conduza à não realização de todas as tarefas. Adicionalmente, pretende-se ainda apostar no aumento da eficiência do planeamento, através da procura de otimizações temporais na realização dos trajetos, que proporcionem uma diminuição do tempo de execução médio e final das tarefas.

Para que seja possível a realização de diversos testes de coordenação, será ainda fundamental o desenvolvimento de plataformas virtuais, com recurso a ambientes de visualização e/ou simulação.

O desenvolvimento de todo o software deverá ainda ser pensado e implementado tendo como vista uma futura aplicação em ambientes industriais reais.

Capítulo 2

Revisão Bibliográfica

Um sistema multi-AGV caracteriza-se pela utilização de um conjunto de robôs móveis capazes de executar um conjunto de tarefas de forma mais flexível e tolerante a falhas do que um robô único mais poderoso. Para que a utilização destes sistemas seja útil e eficiente existe uma forte necessidade de promover um comportamento cooperativo entre todos os agentes envolvidos no sistema.

Este capítulo foca-se numa apresentação de um panorama geral da evolução dos diversos sistemas robóticos cooperativos implementados até ao momento. Nesse seguimento, são descritas e diferenciadas, na secção 2.1, as duas arquiteturas em que estes sistemas se podem inserir. Segue-se na secção 2.2, uma revisão dos algoritmos de pesquisa utilizados quer em ambiente estático quer dinâmico, os quais são precedidos dos métodos de planeamento de trajetória mais considerados para movimento coordenado. Por fim é feita, na secção 2.3, uma descrição das situações de bloqueio mútuo passíveis de ocorrer em sistemas que consideram a utilização de recursos partilhados.

2.1 Arquiteturas de um Sistema Robótico Cooperativo

A definição de uma arquitetura para um sistema robótico cooperativo é fundamental para a determinação das suas capacidades e limitações [19]. Com o decorrer da investigação nas últimas décadas, duas arquiteturas principais se destacam e se distinguem pela possibilidade ou não de conferir autonomia a todos os robôs que constituem o sistema. São elas as arquiteturas centralizadas e as arquiteturas descentralizadas, apresentadas em 2.1.1 e 2.1.2, respetivamente.

2.1.1 Arquiteturas Centralizadas

As arquiteturas centralizadas têm vindo a ser caracterizadas pela existência de uma unidade central de planeamento que, em simultâneo, coordena os movimentos e planeia as trajetórias para todos os robôs constituintes de um sistema, [10, 20]. Por outras palavras, recorrem a um robô líder, com conhecimento global e capacidade para organizar os movimentos de uma equipa. Isto faz com que todos os restantes robôs percam a autonomia e se movimentem mediante determinadas

ordens. Em [10] esta centralização é vista como uma possibilidade para obter soluções finais completas, conseguidas devido ao conhecimento de todos os estados do sistema a cada instante do planeamento. Contudo, é considerado que um aumento do número de robôs em utilização pode provocar um crescimento exponencial do tempo computacional, funcionando como uma contrariedade à eficiência do sistema. Ainda assim, diversas abordagens centralizadas têm sido propostas, entre elas [21] aborda uma nova estratégia de planeamento com base em diagramas de coordenação, na qual um sistema central de planeamento define uma região de colisão para cada par de caminhos definidos, tornando possível a execução de trajetórias livres de obstáculos. Por sua vez, [22] propõe um planeamento através de janelas temporais, nas quais os possíveis caminhos para cada veículo são inseridos, de forma a testar a existência ou não de sobreposição temporal num mesmo arco face a outros veículos.

2.1.2 Arquiteturas Descentralizadas

Com o objetivo de proporcionar a cada veículo um planeamento autónomo surgiram as arquiteturas descentralizadas, também designadas na literatura por distribuídas. Caracterizadas por uma maior tolerância a falhas e a mudanças dinâmicas do ambiente [20], estas arquiteturas revelam um maior domínio na aplicação a sistemas robóticos cooperativos [19]. Segundo [10], associado a este tipo de arquiteturas, é possível distinguir um método de planeamento por prioridades e um método de coordenação do caminho.

Tendo em consideração os métodos de planeamento por prioridades, cada robô planeia o seu trajeto individualmente tendo em consideração os trajetos que já foram planeados pelos veículos de prioridade superior. Embora neste caso, as opiniões divirjam quanto à existência ou não de descentralização do planeamento [19], [23] defende que o facto de ser dada prioridade a um robô não implica que os outros o sigam ou executem algo por ele planeado. Pelo contrário, os robôs com menor prioridade mantêm a sua autonomia na escolha do trajeto que desejam, mas apenas dentro de um conjunto de possibilidades que permitam evitar a colisão com os restantes. Além disso, [23] afirma ainda que a ordem de prioridades definida apenas é relevante se houver alguma possibilidade de os robôs interferirem uns com os outros durante a realização das suas tarefas. De facto, segundo [10], a ordem com que estas são executadas pode influenciar significativamente a performance do sistema, implicando um menor número de trajetos viáveis encontrados, isto é, livres de colisão. Por esse motivo, começa por apresentar três abordagens de planeamento baseados em prioridades fixas, as quais evoluem, de seguida, para uma variante final que aplica uma estratégia de realocação de prioridades adaptativa. Nesta é proposto um aumento da prioridade do robô, sempre que o mesmo não encontre um caminho viável para atingir o seu destino. Observa-se assim uma maior taxa de sucesso, principalmente quando se consideram os resultados do planeamento para ambientes mais movimentados.

Relativamente ao método de coordenação do caminho, a abordagem assenta na possibilidade de cada robô planear, inicialmente, o seu caminho independentemente de qualquer informação proveniente dos restantes robôs. Tal é possível devido à utilização de algumas estratégias de coordenação nos momentos que antecedem uma interseção de dois robôs. [24] sugere, portanto, a

possibilidade de definir prioridades de passagem através de negociações entre os robôs envolvidos ou pela utilização de uma entidade que recebe a descrição da situação atual e através de determinadas regras, fica encarregue de efetuar a prioritização. Por sua vez, [25] propõe a utilização de perfis de velocidade, nos quais são introduzidos atrasos sempre que se prevê um possível ponto de colisão entre dois robôs. Também [26] refere a utilização de estratégias de mudança de velocidade ou atrasos do movimento, aliando a estas a utilização de mapas de colisão e escalonamento temporal da trajetória.

No que respeita ao conhecimento dos planos de todos os robôs, [13] propõe uma solução totalmente descentralizada, na qual os robôs comunicam diretamente entre si, em vez de acederem a uma base de dados comum. Daqui insurge o problema de sobrecarga na comunicação quando o número de robôs aumenta, o qual é solucionada permitindo a comunicação apenas entre robôs que se encontrem na mesma rede de coordenação.

2.2 Planeamento de movimento cooperativo

Uma vez definidas as capacidades e limitações do sistema através da sua arquitetura, é necessário implementar uma abordagem de planeamento da trajetória para todos os robôs constituintes do sistema.

Por planeamento de trajetória pode ser entendido a representação de um caminho no decorrer do tempo. Dado um ponto de partida e um ponto de chegada, o objetivo passa por determinar os caminhos viáveis que unem as duas localizações e que permitem a não colisão com qualquer obstáculo, seja ele estático ou dinâmico. Para ser possível a obtenção desse trajeto seguro, existe a necessidade de definir, em primeiro lugar, um espaço de configuração para o ambiente (Cespaço), que seja capaz de refletir o espaço livre onde os diferentes caminhos poderão estar inseridos (Clivre) bem como os obstáculos (Cobstáculos) existentes. Após a definição do ambiente, diversos métodos de planeamento podem, então, ser implementados com vista a identificação e análise dos possíveis trajetos para os robôs. É ainda de realçar que, para determinados métodos, como por exemplo Roadmap e Decomposição em células, os trajetos resultantes são ainda convertidos em grafos, o que conduz à necessidade de utilizar algoritmos de pesquisa que sejam capazes de determinar os caminhos de menor custo, de uma forma ótima ou próxima do ótimo.

Nas secções 2.2.1 e 2.2.2, é apresentado um panorama geral das várias metodologias que têm vindo a ser propostas, no que respeita ao planeamento de trajetórias e algoritmos de pesquisa, direcionados para a coordenação de movimento cooperativo.

2.2.1 Planeamento de trajetórias

2.2.1.1 Roadmap

A ideia base de uma abordagem roadmap passa por conseguir estabelecer conetividade entre todo o espaço livre de obstáculos (Clivre), utilizando-se para isso nós e ligações. Uma vez conseguida a construção dessa rede de caminhos, a mesma é representada em grafos para que possa,

posteriormente, ser efetuada uma pesquisa à procura de um caminho que estabelece a conexão entre o ponto de partida e o ponto de chegada.

O foco principal desta abordagem passa, portanto, pela construção do roadmap. Métodos como Visibility Graphs e Diagramas de Voronoi têm vindo a ser aplicados e aparecem descritos nas secções seguintes.

Visibility Graphs

Visibility graphs podem ser vistos como gráficos a duas dimensões que representam o espaço livre e ocupado do ambiente. Sendo considerados como nós do Cespaço os vértices dos obstáculos, apenas existem caminhos livres entre o ponto de partida e o ponto de chegada, se existirem ligações entre nós parciais que não intersetem o espaço de nenhum obstáculo. Assim sendo, uma ligação entre dois nós só é vista como possível, se estes estiverem visíveis um para o outro. De realçar ainda que também as arestas dos obstáculos podem ser tidas em conta como espaço livre e possível de percorrer. Contudo, esse detalhe pode trazer complicações adicionais, na medida em que permite o robô andar no limite da colisão com o obstáculo.

Na figura 2.1, é ilustrado um exemplo de um visibility graph constituído por 3 regiões escuras a simular os obstáculos. O trajeto a negrito é representativo do caminho mais curto entre o ponto inicial e final, a passar pelos vértices dos obstáculos.

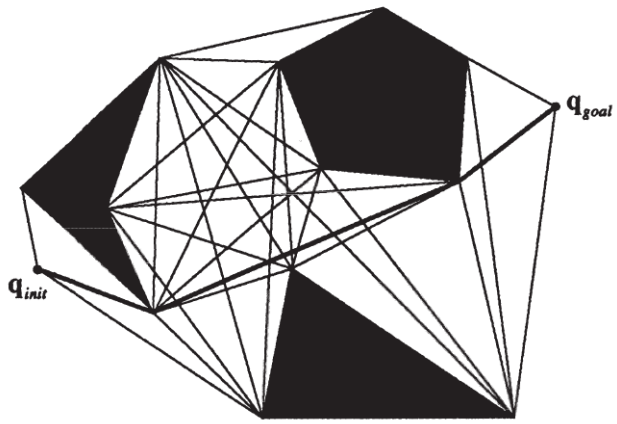


Figura 2.1: Exemplo de um Visibility Graph [1].

O algoritmo para este método de construção de um roadmap resume-se, então, à construção de uma rede de trajetos visíveis e consequente procura na rede de um caminho entre o ponto inicial e final. Se nenhuma solução for encontrada, retorna-se falha na pesquisa.

Diagrama de Voronoi

Um diagrama Voronoi é obtido através de um conjunto de segmentos e arcos, cujos pontos que lhes pertencem estão equidistantes de dois ou mais obstáculos, tal como aparece ilustrado na

figura 2.2. Por exemplo, é possível observar que todos os pontos constituintes do segmento S1 (sinalizado pelo quadrado laranja) estão à mesma distância de E3 e E7 (sinalizados pelos círculos azuis).

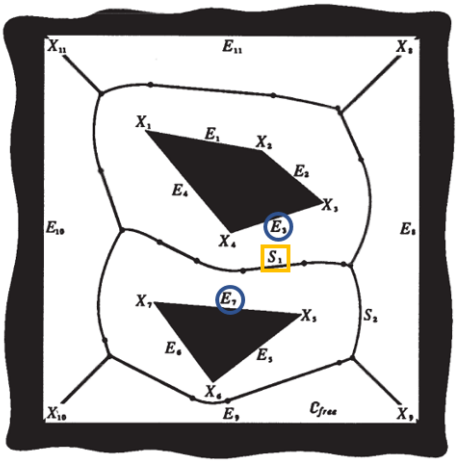


Figura 2.2: Exemplo de um Diagrama de Voronoi [1].

Para efetuar o planeamento da trajetória, se considerarmos que nem o ponto inicial nem o ponto final coincidem com o Diagrama de Voronoi, o procedimento pode ser descrito da seguinte maneira: inicia-se com o deslocamento do ponto inicial até ao ponto mais próximo do diagrama e prossegue-se com a pesquisa do caminho mínimo que levará o robô até ao ponto do diagrama mais próximo do ponto final, para o qual é por fim deslocado.

Relativamente aos Visibility Graphs, os Diagramas de Voronoi apresentam a vantagem de planear caminhos, que mantêm o robô o mais afastado possível dos obstáculos, evitando desta maneira colisões. Em contrapartida, para ganhar uma maior segurança no trajeto, abdicam da possibilidade de obter caminhos mínimos entre o ponto de partida e o ponto de chegada.

2.2.1.2 Decomposição em células

Esta abordagem consiste na divisão do Cespaço em células, entre as quais é possível definir um conjunto de caminhos. Estes são representados numa estrutura em grafo, para posterior análise de quais os que podem ser seguidos para ir desde o ponto inicial até ao ponto final.

Mediante a representação pretendida para o ambiente considerado, é possível decompô-lo em células exatas ou em células aproximadas.

Decomposição em células exatas

Quando se opta por uma decomposição em células exatas, o mapa resultante refletirá com exatidão o ambiente real, mantendo uma distinção clara entre o Clivre e o Cobstáculos. Da decomposição resulta um conjunto de regiões não sobrepostas que, quando aplicada a união das mesmas,

obtém-se rigorosamente o Cespaço. No que respeita à decomposição do Clivre, as geometrias mais utilizadas para as células resultantes são polígonos convexos ou trapézios.

Numa decomposição por polígonos convexos, as células são obtidas através dos segmentos de reta que unem todos os vértices dos obstáculos, tal como é ilustrado na figura 2.3. Por sua vez, a obtenção de um caminho livre de obstáculos entre os pontos de partida e chegada pode ser conseguida através da união dos pontos médios dos segmentos que fazem fronteira entre regiões consecutivas.

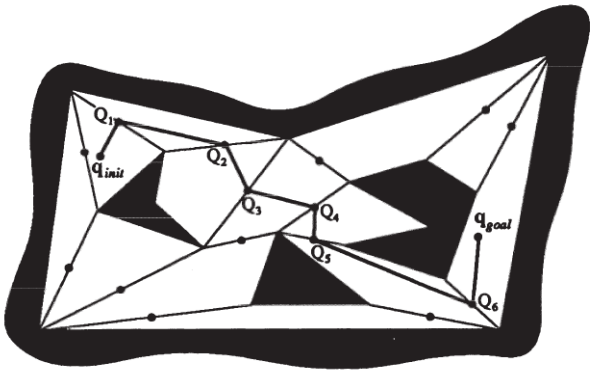


Figura 2.3: Exemplo de uma decomposição exata por polígonos convexos [1].

A decomposição por trapézios é conseguida traçando um conjunto de segmentos de reta paralelos a um eixo pré-definido e a passar pelos vértices dos obstáculos. Esta abordagem garante o formato de um trapézio para cada célula do espaço livre. A figura 2.4 ilustra precisamente essa decomposição, considerando o paralelismo dos segmentos face ao eixo y.

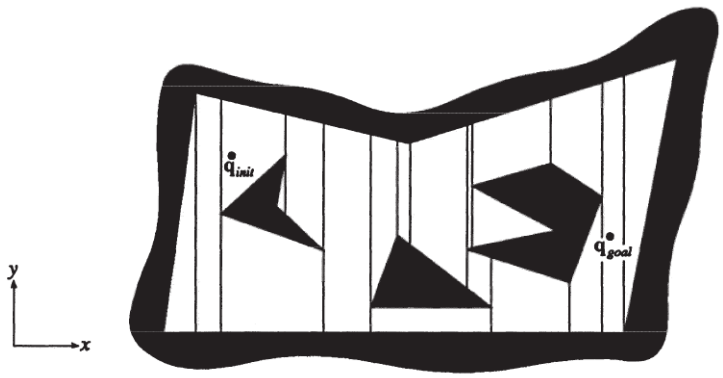


Figura 2.4: Exemplo de uma decomposição exata por trapézios [1].

Decomposição em células aproximadas

Uma decomposição por células aproximadas distingue-se de uma por células exatas, pelo menor rigor que é tido na representação do ambiente real. O facto desta abordagem considerar para

as células formatos simples e pré-especificados, como por exemplo quadrados, implica que quanto mais complexa for a geometria dos obstáculos, mais difícil se torna a representação do seu formato. Isso dá origem à existência de células livres, ocupadas e semi-ocupadas. De notar ainda que o erro de representação está diretamente relacionado com o tamanho considerado para as células. Quanto menor este for, mais células existirão e mais se aproximará o mapa do real. Além disso, contando que o caminho se planeia ao longo das células livres, maior será a probabilidade de encontrar um caminho entre o nó de partida e o nó de chegada, uma vez que a percentagem de células semi-ocupadas diminuirá.

A aproximação considerada é compensada pela vantagem de reduzir a complexidade e o tempo de construção do Cespaço.

Várias abordagens podem ser utilizadas para efetuar a decomposição, no entanto, as que mais se destacam são por célula fixa ou por Quadtree.

Quando se utiliza células fixas, tal como o nome indica, o espaço livre é decomposto por células de tamanho invariável. Tem a vantagem de ser mais simples, contudo, para obter zonas de maior precisão junto dos obstáculos, criam-se também outras zonas com uma quantidade de células desnecessárias, que aumentam a complexidade de uma posterior pesquisa.

Em oposição às células fixas existem as células em Quadtree. Nestas aplica-se uma metodologia de decomposição recursiva. O espaço começa dividido em quatro células iguais e, sempre que uma delas não pertencer integralmente ao espaço livre, é novamente decomposta em quatro células iguais. Esse processo repete-se até que seja atingido um limite de resolução mínimo pré-definido. Esta recursividade permite a obtenção de maior precisão nas fronteiras com os obstáculos e menor precisão nas zonas livres mais espaçadas. Contudo, estando o planeamento de trajetória condicionado à união dos centros das células, o caminho mínimo entre o ponto de partida e o ponto de chegada pode não ser conseguido.

Na figura 2.5 é ilustrada a diferença entre uma decomposição por células fixas e por células em Quadtree.

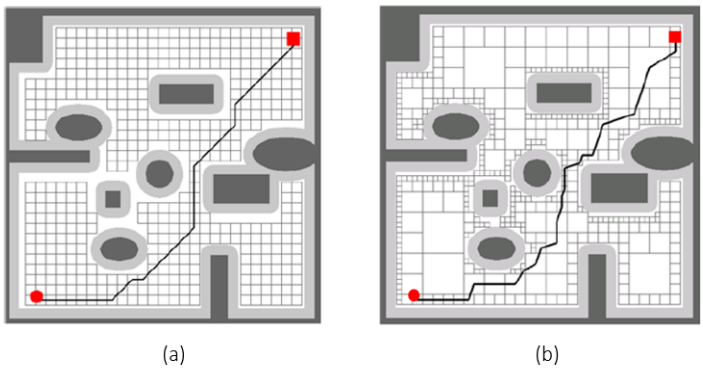


Figura 2.5: Exemplo de uma decomposição de células aproximadas: (a) por célula fixa (b) em Quadtree [2].

2.2.1.3 Velocity Obstacle

O conceito de velocidade dos obstáculos foi introduzido por [27], o qual considerou para os mesmos trajetórias lineares conhecidas. Nesse seguimento foi proposto um método de primeira ordem para determinar o conjunto de velocidades que o robô teria de assumir para causar uma colisão com um obstáculo num instante de tempo futuro, movendo-se a uma dada velocidade.

Mais tarde, [28] propôs uma evolução da sua abordagem anterior através de um método de segunda ordem que permite considerar trajetórias arbitrárias para os obstáculos, introduzindo o conceito de velocidade não linear para os mesmos, a qual tem em conta a forma, a velocidade e a curvatura do seu movimento.

É de salientar que o conceito de velocidade dos obstáculos assume que estes se movem passivamente no ambiente, sem qualquer perceção do que os rodeia. Isso faz com que, quando considerado movimento cooperativo, o robô que se encontra a planear o seu trajeto não tenha em consideração que também os outros robôs têm capacidade para decidir e evitar colisões. Segundo [3] isso provoca oscilações indesejáveis e irrealistas no movimento do robô, como pode ser observado, à esquerda, na figura 2.6. Para garantir um movimento livre de colisões, em [3] é então proposto o conceito de velocidade recíproca dos obstáculos, a partir do qual cada robô passa a ter em conta o comportamento reativo dos restantes. Na figura 2.6, à direita, é conseguido um movimento sem oscilações, utilizando o conceito de velocidade recíproca.

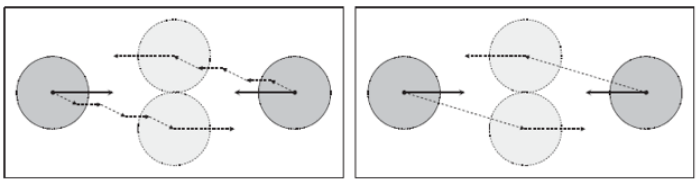


Figura 2.6: Demonstração dos conceitos de (a) Velocity Obstacle (b) Reciprocal Velocity Obstacle [3].

Posteriormente, [29] propôs uma adaptação do horizonte temporal a definir para a velocidade de um obstáculo, a qual, quando considerada muito antes de uma possível interseção, será sempre uma abordagem segura, contudo, pouco eficiente na medida em que pode impedir o robô de efetuar determinadas manobras, consideradas arriscadas, mas na realidade possíveis. Em contrapartida, considerar um horizonte temporal muito pequeno pode fazer com que o robô se aperceba da possível colisão com o obstáculo demasiado tarde e já não tenha tempo de se desviar, sendo por esse motivo, importante conseguir atingir um ponto intermédio. [29] foca-se então num método capaz de determinar o mínimo horizonte temporal que confira segurança ao movimento do robô, quer em ambientes estáticos quer dinâmicos.

2.2.1.4 Outras abordagens

Vários outros métodos têm vindo a ser propostos, de modo a conseguir-se planear a trajetória coordenada de um conjunto de robôs.

Nesse seguimento, abordagens baseadas em campos de potencial começaram a ser adaptadas para entrar em conta com o movimento cooperativo. [30] focou-se na implementação de um novo método online que entra em conta com um ambiente dinâmico na presença de objetos dinâmicos. Nestes casos, o nó final pretendido funciona como uma força de atração, ao contrário de todos os outros robôs que são vistos como forças repulsivas. Segundo [5] as abordagens baseadas em campos de potencial têm a desvantagem de apresentar soluções computacionalmente mais demoradas ou por vezes nem apresentar soluções possíveis quando confrontadas com ambientes estreitos.

Começaram também a surgir abordagens baseadas na tentativa de coordenar os robôs, colocando para os mesmos movimento com uma direção uniforme, que evite colidirem entre eles. Em [4] é proposto a utilização de mapas de direção que encorajam os robôs a seguir determinada trajetória e a moverem-se de um modo mais uniforme. Isso é conseguido através do armazenamento de um vetor de direção para cada célula, o qual é atualizado sempre que um robô entra ou sai dela. É então efetuada uma média pesada entre a direção do vetor já armazenada e a nova direção com que um novo robô chega à célula. O encorajamento é feito através de pesos estabelecidos que penalizam os movimentos que vão contra a direção média de cada célula.

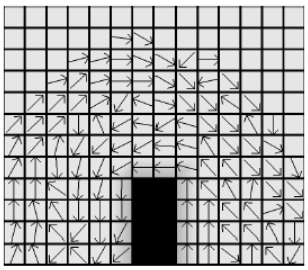


Figura 2.7: Exemplo de um mapa de direção com pesos estabelecidos [4].

Um conjunto de abordagens para lidar, em tempo real, com vários robôs em constante movimento, incorporando a dinâmica dos mesmos na representação das células do mapa, foi ainda proposto por [5]. Cada célula passa a incluir conhecimento acerca dos robôs que poderão interferir na trajetória que está a ser planeada, colocando-os como possíveis obstáculos no mapa. A estimação desses pontos de possível colisão é conseguida pela análise da velocidade a que cada robô se move. A determinação dos pontos de colisão não está no foco do seu documento, mas está na base da aplicação das técnicas propostas para aproximar um mapa, que é dinâmico, por um mapa estático.

Para começar, considerou que, quando um ponto de colisão é inserido no mapa como obstáculo, o tamanho deste deve depender da distância ao robô que está a efetuar o planeamento. À

medida que esta aumenta, a importância do obstáculo pode ser diminuída e, a partir de certo valor, deixar mesmo de ser considerada. Na figura 2.8, aparece ilustrada esta situação.

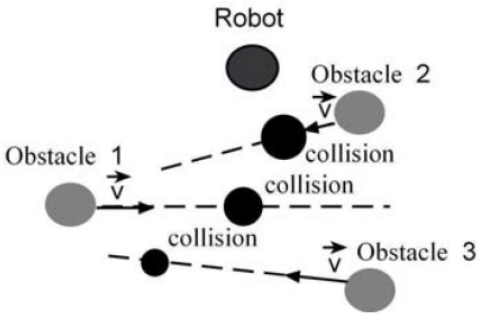


Figura 2.8: Demonstração da diminuição da importância dos obstáculos com o aumento da distância ao robô que efetua o planeamento [5].

De seguida propôs a criação de uma zona de segurança à volta dos obstáculos, a qual é conseguida aumentando o custo de atravessar as células que envolvem a célula do obstáculo. O valor desse custo diminui até ao custo unitário à medida que se afasta do obstáculo, fazendo desvanecer a zona de segurança, como demonstrado na figura 2.9.

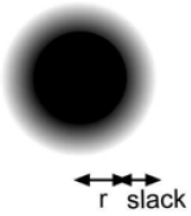


Figura 2.9: Demonstração da zona de segurança criada à volta do obstáculo [5].

A terceira técnica é direcionada para casos em que o robô assume uma trajetória paralela ao obstáculo na tentativa de o evitar. Um longo período poderia ser necessário para se conseguir resolver a situação de colisão eminente. Desse modo foi proposto a criação de uma zona adicional para expandir o tamanho do obstáculo na direção do seu movimento, aumentando assim desta forma o custo de seguir nesse sentido face ao custo de efetuar a passagem pela parte de trás (figura 2.10).

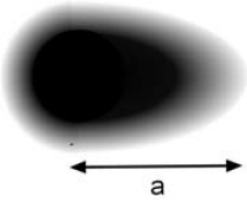


Figura 2.10: Demonstração do aumento do obstáculo numa determinada direção [5].

Por último, quando se pretende criar direções preferenciais para o movimento do robô, isso pode ser conseguido através do aumento do custo de percorrer as células vizinhas ao caminho com a direção pretendida. Além disso, em casos como futebol robótico, pretende-se mesmo que o robô atinja a bola (obstáculo) segundo uma determinada direção que favoreça o objetivo da sua equipa, (figura 2.11). Se não houver intenção de criar uma direção bastante restrita, pode ser aumentado o custo na vizinhança de forma gradual.

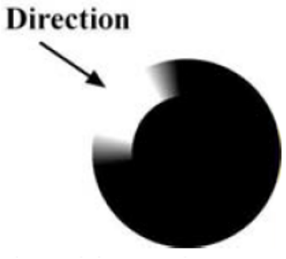


Figura 2.11: Demonstração da criação de uma direção preferencial para atingir o obstáculo [5].

2.2.2 Algoritmos de pesquisa

Uma vez definido, ao longo de um grafo, um conjunto de trajetórias possíveis para o movimento de um robô, é necessário aplicar algoritmos de pesquisa. Estes conferem a possibilidade de determinar soluções que conduzam o robô do ponto de partida ao ponto de chegada. Mediante o método de pesquisa utilizado, quatro fatores principais podem ser analisados para descrever a eficácia e eficiência dos mesmos: completude, complexidade temporal, complexidade espacial e otimalidade.

A completude de um algoritmo indica se este é capaz de determinar uma solução viável entre dois pontos pretendidos. Por sua vez, a complexidade temporal fornece uma medida do tempo necessário para a resolução de um problema, o qual está frequentemente associado ao número de nós que são necessários expandir até atingir uma solução ou até percorrer todo o grafo. Durante a pesquisa, é ainda imprescindível armazenar informação dos nós em memória, cuja quantidade necessária é dada pela complexidade espacial. Por último, e de igual importância, está a obtenção de uma solução ótima ou não, a qual só é conseguida se a solução obtida é a melhor de todas as possíveis, no que respeita ao custo de atravessar todas as ligações que constituem o caminho escolhido para ir desde o ponto inicial até ao ponto final.

Dentro de um conjunto de algoritmos existentes é ainda possível distinguir, pela existência ou não de informação entre nós do grafo, dois principais grupos de algoritmos, mais comummente designados de algoritmos com e sem heurística, respetivamente. Na literatura, a utilização dos primeiros destaca-se devido à capacidade que estes oferecem de utilizar as informações fornecidas em cada ligação, para mais rapidamente atingir uma solução de mais baixo custo e recorrendo a menos recursos de tempo e espaço.

Nas secções 2.2.2.1 e 2.2.2.2, são apresentados um conjunto de algoritmos que se enquadram em cada um dos grupos mencionados. De notar ainda a existência de diversas subsecções no grupo dos algoritmos com heurística, as quais fazem referência a modificações introduzidas ao algoritmo A* que lhe permitiram lidar, de forma mais eficiente, com pesquisas em ambientes dinâmicos.

2.2.2.1 Algoritmos sem heurística

No grupo dos algoritmos sem heurística, é possível distinguir os algoritmos de pesquisa por largura, por profundidade, por aprofundamento limitado e por aprofundamento iterativo. A diferença principal entre eles assenta na direção da pesquisa no grafo bem como nas limitações que são impostas. Para caracterizar e comparar os algoritmos será considerado a letra b para representar o número máximo de ligações para um nó, a letra d para a profundidade a que é encontrado o nó objetivo e a letra m para a máxima profundidade do grafo.

Numa pesquisa por largura, todos os nós que se encontrem diretamente ligados ao nó atual são explorados seguidamente. Isto faz com que a pesquisa do grafo se faça por camadas, tal como é possível observar na figura 2.12.

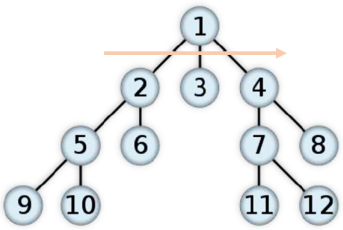


Figura 2.12: Pesquisa por largura num grafo.

Este algoritmo apenas pode ser considerado completo se o valor de b for finito. Além disso, só poderá apresentar soluções ótimas se o custo de percorrer cada ligação for igual ao longo de todo o grafo.

Numa pesquisa por profundidade, o grafo é explorado desde o nó inicial até ao mais profundo, voltando ao nó anterior sempre que não for detetado mais nenhum nó abaixo do atual. A pesquisa é feita de cima para baixo e da esquerda para a direita, como pode ser observado na figura 2.13.

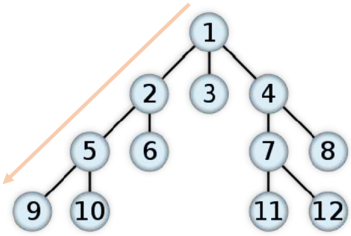


Figura 2.13: Pesquisa por profundidade num grafo.

Este algoritmo apenas pode ser considerado completo se o valor de b for finito. Além disso, não apresenta soluções ótimas, uma vez que, ao percorrer o grafo em profundidade, a primeira solução detetada poderá não corresponder à de caminho mínimo.

Uma pesquisa por aprofundamento limitado é semelhante a uma pesquisa por profundidade, contudo é pré-definido um limite de camadas a ser exploradas. Por esse motivo, quando esse limite é atingido, dá-se o recuo ao nó anterior e continua-se a pesquisa. Este algoritmo apenas apresenta completude se houver alguma solução a ser atingida dentro do limite imposto. De notar também que não é ótimo.

Uma pesquisa por aprofundamento iterativo assemelha-se a uma de aprofundamento limitado, no entanto o limite vai sendo aumentado recursivamente até ser alcançada uma solução. Desta forma, garante completude mas não cumpre otimalidade.

2.2.2.2 Algoritmos com heurística

No que respeita a algoritmos com heurística, entre os mais reconhecidos encontram-se o algoritmo de Dijkstra, o algoritmo guloso e o algoritmo A*.

O algoritmo de Dijkstra caracteriza-se por explorar cada vizinhança de um nó procurando o custo mais baixo em relação ao ponto de partida. Como não entra em conta com o custo até ao nó objetivo, apenas apresenta soluções ótimas se o custo for igual para todas as ligações. Além disso, é um algoritmo completo.

Em contrapartida, o algoritmo guloso difere do Dijkstra na medida em que explora a vizinhança do nó atual, procurando o nó com um custo mais baixo relativamente ao ponto de chegada definido. Apresenta completude na pesquisa, mas as soluções obtidas não são ótimas.

Por sua vez, o algoritmo A*, introduzido por [31], já efetua a pesquisa para cada nó tendo em conta o custo até ao ponto de partida e o custo até ao ponto de chegada. Esta característica confere-lhe a possibilidade de obter soluções ótimas e completas.

Para avaliar o custo de cada nó recorre-se, então, a uma função f(s), a qual é representativa do custo de ir de um nó s até um nó objetivo, tendo sempre em consideração o custo que vai desde o nó inicial até ao nó s, dado por g(s), e o custo que vai desde o nó s até ao nó final, dado por h(s).

$$f(s) = g(s) + h(s) \tag{2.1}$$

Na figura 2.14 é ilustrado um trajeto exemplo, composto por um nó inicial (s), um nó final (f) e dois nós intermédios (m e n). Entre os nós estão representadas ligações com custos de atravessamento associados. Focando no nó m, tem-se que f(m)=g(m)+h(m), sendo g(m)=6 e h(m)=3. É de notar que o valor de g(m) é determinado pelo custo mínimo desde o nó de início até ao nó em análise.

Na implementação do algoritmo são sempre consideradas duas listas: uma lista aberta (geralmente designada por O-list), onde são armazenados todos os nós candidatos para exploração, e ainda uma lista fechada (geralmente designada de C-list), que contém os nós já explorados previamente. Todos os nós que estejam inseridos nestas listas armazenam ainda um apontador para o "nó

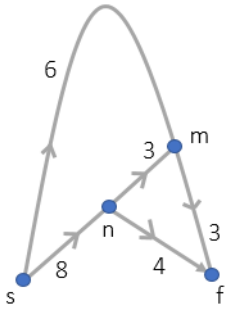


Figura 2.14: Trajeto definido por nós e ligações.

pai", sendo este o nó que, considerando o custo ótimo, lhes deu origem. Isto permite que, quando alcançado o nó objetivo, se tenha conhecimento do caminho de custo mínimo que foi realizado para o atingir.

Ainda que seja computacionalmente pesado e que exija recursos de memória significativos, a possibilidade de obter soluções ótimas na pesquisa dos trajetos faz do algoritmo A* uma opção de extrema relevância, quando considerados ambientes estáticos, isto é, ambientes que não sofrem alterações no decorrer do tempo e nos quais o planeamento pode ser feito em avanço, por haver conhecimento à priori de tudo o que rodeia o robô. Contudo, quando considerados sistemas robóticos de movimento cooperativo entre múltiplas entidades, é necessário entrar em conta com a dinâmica dos obstáculos. Nestes casos, o algoritmo A*, na sua forma standard, torna-se bastante ineficiente. Segundo [32], a resolução do problema passa a ser mesmo intratável, uma vez que o fator de ramificação cresce exponencialmente com o número de robôs considerados no ambiente. Por este motivo, várias modificações ao algoritmo A* têm vindo a ser propostas e implementadas. Um panorama geral da sua evolução é apresentado nas secções seguintes.

Dynamic A*

Sendo uma das principais desvantagens do algoritmo A*, a complexidade no planeamento, em 1994 foi proposto por [33] um método de pesquisa semelhante, que permitiu reduzir o tempo de processamento. Designado por D*, *"Dynamic A*"*, este algoritmo destaca-se pela capacidade de recalcular a trajetória a cada iteração, sem ser necessário recalcular os custos associados de todo o espaço disponível desde a raiz do problema, para descobrir a informação necessária para ir de um nó inicial a um nó final. Uma vez caracterizado pela sua capacidade de replanear a cada iteração, o algoritmo D* viabiliza a hipótese dos custos das ligações bem como as posições dos obstáculos serem alterados com o decorrer do tempo, sem influenciar o bom desempenho do mesmo. Segundo [33], o algoritmo está ainda preparado para funcionar em ambientes desconhecidos ou parcialmente conhecidos, mantendo a completude e otimalidade da pesquisa.

Lifelong Planing A*

Também [6] introduziu um método incremental, denominado LPA*, *"Lifelong Planing A*"*, que permite reutilizar informação prévia do planeamento, diminuindo o tempo de processamento. Dando como exemplo um *website* que planeie o caminho mais rápido de um aeroporto a um centro de conferência, [6] sugere como vantagem do seu método a reutilização de partes de uma pesquisa anterior que sejam idênticas à nova árvore de pesquisa, de modo a evitar o planeamento desde a raiz do problema. Ou seja, considerando um caminho original planeado, que por algum motivo ficou bloqueado a meio do percurso, o algoritmo LPA* aproveita o processamento dos nós já feito anteriormente e replaneia apenas os nós cuja distância ao nó inicial foram alteradas, conseguindo assim uma maior rapidez computacional. Nas figuras 2.15 e 2.16 é possível observar a cinzento os nós explorados durante a pesquisa, tendo como ponto de partida a célula R4 e ponto de chegada a célula R17. Numa pesquisa inicial, o LPA* explora tantos nós como o A*, contudo quando existe alteração do ambiente, apenas os nós cuja distância mínima à origem foi modificada é que são processados, reduzindo assim o peso computacional no replaneamento.

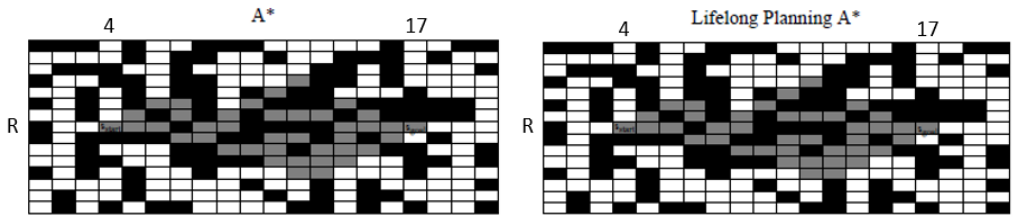


Figura 2.15: Nós expandidos no planeamento do trajeto original, por LPA*, face ao A* [6].

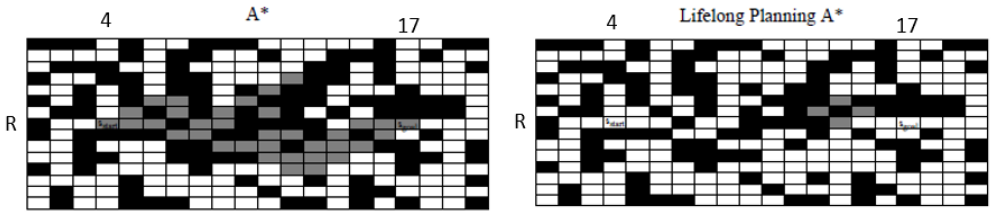


Figura 2.16: Nós expandidos no replaneamento do trajeto, por LPA*, após mudança do ambiente, face ao A* [6].

[6] refere, como área de aplicação do LPA*, ambientes conhecidos e valida a sua eficiência face ao método de pesquisa completa A*, utilizando como fatores de comparação o número de vértices expandidos, o número de vértices acedidos e o número total de trocas de pais e filhos no grafo.

Dynamic A* Lite

Posteriormente, [7] introduz o algoritmo D* Lite como uma evolução ao algoritmo LPA* anteriormente por si proposto. O D* Lite foi elaborado na base do LPA*, ao qual se conferiu a possibilidade de replanear o trajeto em casos onde o nó objetivo da pesquisa é alterado. Ao contrário do LPA*, que a cada fase de replaneamento traça novo caminho desde o ponto de partida ao ponto de chegada, o D* Lite apenas recalcula o caminho mínimo desde a posição atual do robô até ao ponto final. Como pode ser visto na figura 2.17, o algoritmo preocupa-se, fundamentalmente, com as distâncias ao nó objetivo, valores esses registados nas células. De notar ainda que, quando encontrado um novo obstáculo, apenas são explorados os nós cuja distância ao objetivo tenha sido alterada ou não previamente calculada, e que apresentem relevância no replaneamento do caminho mínimo.

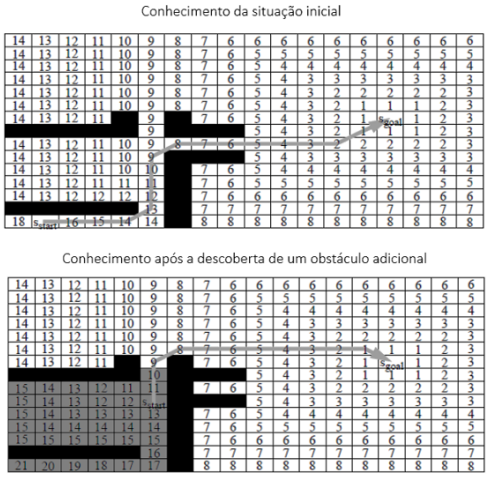


Figura 2.17: Exemplo de planeamento através do algoritmo D* Lite [7].

O algoritmo D* Lite assemelha-se ao algoritmo D* na medida em que planeia os movimentos do robô de forma idêntica e pode ser igualmente aplicado a ambientes desconhecidos, contudo é apresentado como sendo algoritmicamente diferente. [7] afirma ainda que, quando analisado experimentalmente, o D* Lite aparenta ser um pouco mais eficiente que o D*.

Ainda que os algoritmos D*, LPA* e D* Lite apresentem um acréscimo de eficiência na pesquisa bastante significativo face ao algoritmo A*, eles estão preparados, essencialmente, para lidar com ambientes dinâmicos, na presença de obstáculos estáticos conhecidos ou desconhecidos. No entanto, [10] sugere que, quando considerados ambientes dinâmicos com obstáculos em movimento, como é o caso dos sistemas robóticos cooperativos, eles não funcionam de forma eficiente.

Para lidar com vários robôs em movimento num mesmo ambiente, é necessário garantir que a qualquer instante, os seu trajetos não se intersetam. Assim sendo, começou a surgir na literatura um conjunto de abordagens que entram em conta com a noção temporal durante o planeamento cooperativo.

Cooperative A*

Baseando-se nesse conceito, [34] introduziu o algoritmo CA*, *"Cooperative A*"*, o qual assume a existência de conhecimento total, por parte de cada robô, dos trajetos planeados ao longo do tempo por todos os outros robôs. Para tal ser possível, sugere a utilização de uma tabela de reservas que pode ser vista como uma grelha a três dimensões, a qual é composta por duas componentes espaciais e uma temporal. Na tabela são inseridos e assinalados como ocupados os caminhos dos robôs já planeados. Como o algoritmo se baseia num planeamento por prioridades fixas, todos os robôs seguintes aos de maior prioridade terão de ter em conta toda a informação armazenada e partilhada na tabela de reservas, de modo a evitar caminhos compostos por colisões. Uma vez que este método é calculado em avanço com vista a determinação de um caminho ótimo, problemas de bloqueio mútuo entre robôs não poderão ser previstos e tratados pelo algoritmo.

Hierarchical Cooperative A*

Segundo [34], utilizar a distância de Manhattan (soma das diferenças absolutas entre as coordenadas de dois pontos) como heurística num espaço a três dimensões, como é o caso do CA*, pode conduzir a uma performance reduzida quando considerados ambientes mais complexos. Por esse motivo, propõe a utilização de uma heurística hierárquica, baseada numa técnica de pesquisa abstrata, designada *Hierarchical A**, [35]. Esta, por sua vez, cria uma série hierárquica de abstrações do espaço, em que cada uma vai sendo mais geral que a anterior. A heurística é, então, construída com base numa abstração a duas dimensões da grelha 3D, utilizada no CA*. Surgiu assim o HCA*, *Hierarchical Cooperative A**, cuja diferença para o CA* está na utilização de uma heurística que ignora as restrições temporais e a tabela de reservas, preocupando-se apenas com a distância calculada até ao destino sem entrar em conta com potenciais interações entre robôs. Segundo [9], o seu bom funcionamento é expectável, uma vez que o tempo necessário para viajar um caminho está altamente correlacionado com a distância viajada. Para efetuar a pesquisa no domínio abstrato, é utilizado o algoritmo RRA* (Reverse A*), o qual efetua uma pesquisa na direção oposta ao A*, isto é, a pesquisa inicia-se no nó de destino e termina no nó expandido, o que permite conhecer a distância ótima entre os mesmos.

O HCA*, permite, portanto, reduzir o custo de pesquisa cooperativa, contudo, não tanto como seria desejável para aplicações em tempo real.

Windowed Hierarchical Cooperative A*

Na sequência do algoritmo HCA*, [34] desenvolveu ainda o algoritmo WHCA*, *Windowed Hierarchical Cooperative A**, que permite reduzir o tempo da pesquisa completa inicial, através da imposição de limites temporais. Considera não ser necessária a utilização de intervalos de tempo bastante alargados para a obtenção de um planeamento eficiente, o que lhe permite fazer uma pesquisa de trajetos parciais entre o ponto de partida e o destino. Apenas é necessário que a janela

temporal definida seja movida periodicamente para possibilitar o cálculo de novas rotas parciais à medida que o robô segue o seu caminho. A eficiência do algoritmo depende da janela temporal definida. Quanto maior esta for mais este aproxima o seu comportamento do HCA*, anulando o efeito pretendido. Em contrapartida, se for utilizada uma janela temporal curta, possíveis situações de bloqueio mútuo poderão ocorrer.

O algoritmo WHCA* apresenta a desvantagem de associar a si custos de memória significativos, devendo-se estes ao armazenamento das pesquisas efetuadas através do RRA*. Além disso surgem ainda custos provenientes da pesquisa completa que é efetuada dentro de determinada janela temporal, antes de todos os robôs efetuarem o seu movimento. Para resolver estes problemas, [9] propôs duas novas abordagens.

A primeira assenta na base já existente do WHCA*. Além de serem adicionadas mais quatro direções de movimento possível face às quatro já anteriormente propostas, recorre ainda à heurística do RRA*, não para uma pesquisa completa de todo o espaço, mas apenas para uma pesquisa abstrata, que permite reduzir a memória de armazenamento de informação necessária. Quanto maior for o nível de abstração definido, menor será a quantidade de memória necessária, contudo resultará numa heurística menos precisa e como consequência num maior número de nós expandidos.

A segunda abordagem combina o algoritmo WHCA* com os conceitos do método PRA*, *Partial-Refinement A**, introduzido por [8]. A ideia principal do PRA* passa pela construção de um mapa de resolução inferior (figura 2.18), cujos resultados do planeamento através do mesmo serão submetidos a uma técnica de aperfeiçoamento. Como pode ser visto na figura 2.19, o PRA* confere uma abstração hierárquica, na qual os níveis superiores vão sendo cada vez mais abstratos e cuja localização virtual será uma correspondente média da localização de todos os nós filhos.

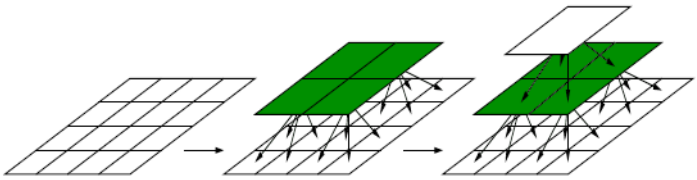


Figura 2.18: Construção de um mapa de resolução inferior, através de um processo de abstração hierárquica, [8].

Pode ainda ser observada a obtenção dos trajetos nas camadas mais baixas de abstração, tendo como ponto de partida a camada mais alta, onde o custo computacional da pesquisa do caminho é muito mais baixo. A partir deste apenas existe uma ampliação do número de nós da trajetória à medida que se desce nas camadas.

Focando na fusão dos conceitos do WHCA* com o PRA*, surgiu, então, o algoritmo CPRA*, *Cooperative Partial-Refinement A**. Este substitui a pesquisa por A* da camada de abstração mais baixa do método PRA*, pela pesquisa WHCA*. Deste modo, embora apresente aproximadamente os mesmos custos que o WHCA* a cada fase de replaneamento, é adicionado ao PRA* a vantagem

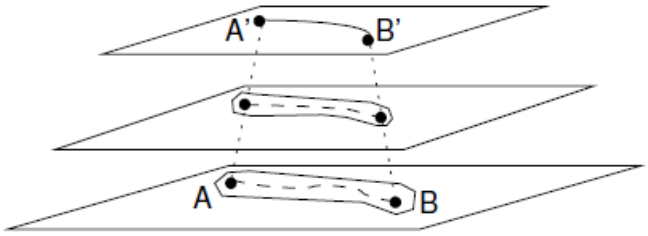


Figura 2.19: Abstração e descoberta do caminho na camada mais baixa de abstração, através das camadas de nível superior, [9].

de poder ajustar dinamicamente o tamanho da janela temporal se for necessário, como em casos de maior número de requisições de trajeto em simultâneo, por parte dos robôs.

De acordo com [9], o algoritmo CPRA* poderá ser uma melhor opção se forem utilizados muitos robôs e restrições de memória apertadas. Mas, em contrapartida, se for possível aguentar um elevado custo inicial e se a memória não for um problema, WHCA* pode fornecer uma melhor performance, após o instante inicial do planeamento ser ultrapassado.

3D A*

Em 2011, movido também pela necessidade de adaptar o algoritmo A* para um planeamento cooperativo, [10] começou por introduzir o algoritmo 3D A*. Este adiciona uma dimensão temporal às duas dimensões espaciais, geralmente consideradas, possibilitando a representação de coordenadas na forma (x,y,t). Esse detalhe possibilita o conhecimento da posição de cada robô a cada instante temporal e confere ainda aos mesmos a alternativa de permanecer com movimento estacionário numa mesma célula, de modo a evitar colisões. Contudo, a exploração de um mapa em grelha com 3 dimensões apresenta um custo computacional bastante elevado, principalmente se for considerada a possibilidade de pesquisar todo o ambiente, no pior caso.

Spatio-Temporal A*

Para lidar com o problema computacional do algoritmo 3D A*, [10] propôs um novo método, designado S-T A*, *"Spatio-Temporal A*"*, ao qual excluiu a dimensão temporal anteriormente considerada no mapa em grelha. Ainda assim, manteve a noção de planeamento no tempo através da introdução de tabelas de ocupação temporal (TOT). Cada nó do espaço 2D passa a ter uma TOT, que apresenta o registo de ocupação dentro dos limites temporais considerados, permitindo assim a cada robô representar o seu caminho planeado a cada instante. É ainda considerada a existência de um planeamento por prioridades fixas, na qual os robôs de prioridade inferior têm de ter em conta as posições da TOT de cada célula já reservadas. Se o caminho originalmente planeado e considerado de custo mais baixo para esses robôs, já entrar em ponto de colisão com outra

trajetória definida, caminhos alternativos têm de ser tidos em conta, mesmo que isso corresponda a um aumento do tempo de execução da tarefa. Esta situação é ilustrada na figura 2.20.

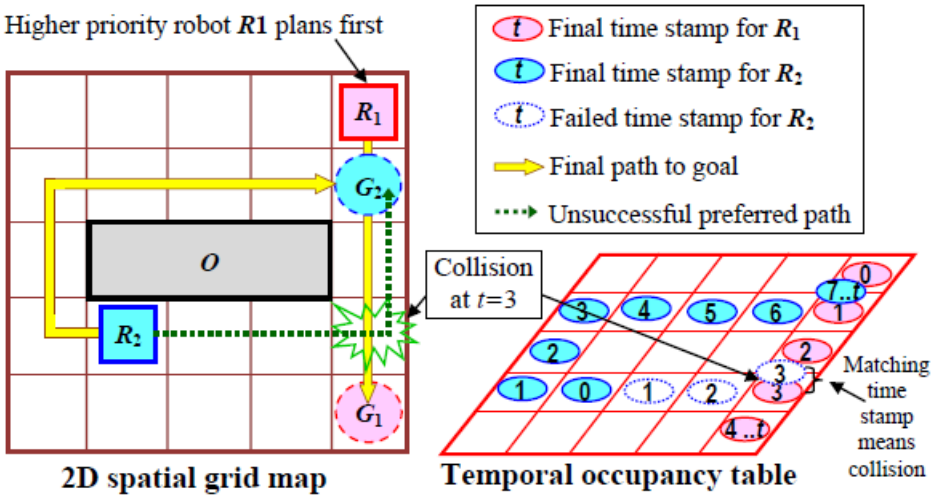


Figura 2.20: Deteção de potencial colisão e determinação de caminho alternativo, através do algoritmo S-T A* [10].

Spatio-Temporal Wait-Near-Collision A*

Embora o algoritmo S-T A* seja bastante eficiente computacionalmente, segundo afirma [10], existem situações de falha na procura de soluções viáveis, quando a percentagem de robôs no ambiente aumenta. Estas poderiam ser solucionadas permitindo a espera do robô num nó apropriado, para evitar colisão. Por esse motivo, propôs ainda evolução do seu método S-T A* para S-T-W A*, *Spatio-Temporal Wait-Near-Collision A**, adicionando-lhe a possibilidade de espera o mais perto possível do ponto de colisão. Tal foi conseguido pela alteração da forma de analisar as células vizinhas já ocupadas em determinado instante temporal. Em vez de serem vistos como nós inexploráveis devido à sua ocupação, levando à procura de novos caminhos, como acontece no S-T A*, passou-se, pois, a efetuar a procura do seu nó antecedente mais perto, onde pode aguardar o tempo necessário até que o caminho fique livre. Mesmo considerando este tempo de espera, o tempo de execução da tarefa poderá manter-se ótimo quando comparado com as restantes alternativas.

Spatio-Temporal Wait-Near-Collision A* com prioridades adaptativas

Mesmo com a introdução do S-T-W A*, [10] verificou que, quando confrontado com ambientes bastante completos, havia casos em que o algoritmo falhava. Com o objetivo de o tornar mais eficiente, evoluiu a sua solução introduzindo-lhe o conceito de prioridades adaptativas. Pretende-se com isto, que um robô que esteja a planear o seu caminho e não encontre uma solução livre de colisões, possa subir na escala de prioridades e replanear o seu trajeto, aumentando as hipóteses de atingir um caminho viável. Embora não resolva por completo o problema, consegue aumentar

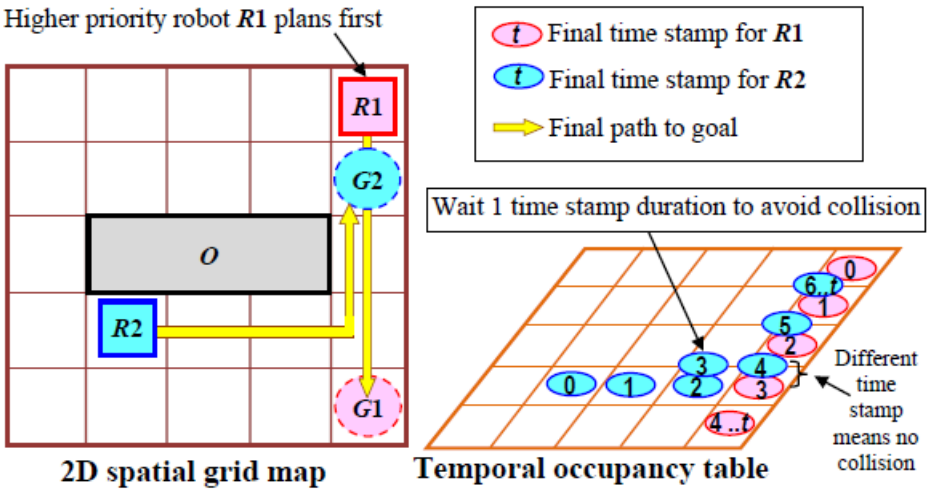


Figura 2.21: Deteção de potencial colisão e espera perto da interseção, através do algoritmo S-T-W A* [10].

as probabilidades de atingir uma solução mais eficiente. Esta adaptação conduz a um possível problema de alocação cíclica de prioridades, isto é, o robô pode subir na hierarquia e mesmo assim não encontrar uma solução de caminho livre, levando-o para a prioridade original. Para evitar um ciclo infinito na realocação, [10] propõe um verificador simples que deteta a existência de ciclo e termina o planeamento desse robô, retornando insucesso no resultado.

Time Enhanced A*

Mais recentemente, em 2015, um novo algoritmo de pesquisa cooperativa com noção temporal foi proposto em [11]. Denominado de TEA*, *Time Enhanced A**, este algoritmo caracteriza-se pela aplicação em cenários de decomposição em células e pela utilização de diferentes camadas temporais a cada instante do planeamento, como ilustrado na figura 2.22. O TEA* entra em conta com as posições dos robôs e com as mudanças dinâmicas do ambiente no decorrer do tempo para calcular o caminho mínimo. Tal é possível através da exploração das camadas temporais seguintes.

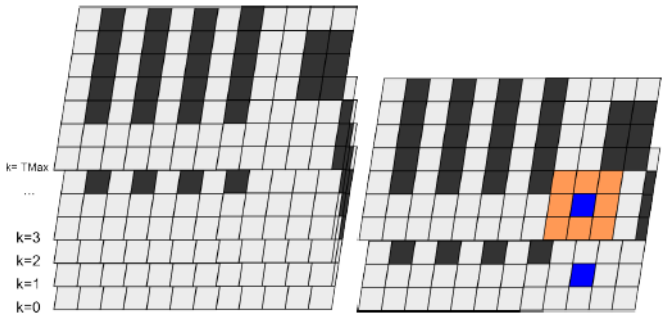


Figura 2.22: (a) Camadas temporais utilizadas no TEA*. (b) Demonstração da expansão dos nós vizinhos para um nó atual [11].

Com a introdução do domínio temporal ao algoritmo A*, as células vizinhas do nó atual que vão ser exploradas, passam a pertencer à camada temporal seguinte, como representado na figura 2.22 (b). Por sua vez, a célula vizinha analisada contém sempre a posição atual do robô, o que lhe permite manter a sua posição entre instantes consecutivos do planeamento. Tal facto é visto como uma vantagem, na medida em que lhe permite evitar longos desvios que poderiam advir de caminhos alternativos ao caminho mínimo. É ainda de notar que o número de camadas temporais a considerar está dependente das dimensões do mapa assim como do número de obstáculos que forem encontrados pelo robô durante o seu trajeto.

Na implementação do algoritmo é assumido um planeamento por prioridades entre os vários robôs que constituem o sistema. Por conseguinte, todos os robôs devem colocar o seu trajeto planeado como obstáculo para que, à medida que se desce na hierarquia das prioridades, os robôs tenham em consideração possíveis pontos de colisão. Além disso, também a posição de todos os robôs inseridos no ambiente é colocada como obstáculo. Contudo, é de realçar que essa apenas é considerada na camada temporal atual e seguinte, k=0 e k=1, de modo a ser possível prevenir situações de deadlock. Ao retirar de obstáculo a posição dos robôs em conflito, a partir de k=2 permite que o robô com maior prioridade possa replanear o seu trajeto, colocando-o como obstáculos para os restantes. Assim sendo, também o robô de prioridade inferior reajustará a sua posição.

Posteriormente, é sugerido por [14] uma evolução do algoritmo. Passou, portanto, a ser implementado com base em grafos, em oposição aos mapas em grelha anteriormente considerados. Esta alteração tem como objetivo reduzir o tempo de processamento e facilitar uma possível integração em cenários industriais. Além disso, foi ainda introduzido um método de resolução de *livelocks*, o qual se baseia na troca das prioridades com que os veículos são analisados. Este último ponto será apresentado na secção 2.3.

[36] surge ainda na sequência dos dois trabalhos previamente mencionados e compara resultados experimentais com um estado de arte alternativo, de modo a validar a aplicação do algoritmo TEA*.

2.3 Deadlocks e Livelocks

Deadlocks podem ser descritos como situações de bloqueio em que dois ou mais AGV tentam alocar um mesmo recurso, não permitindo que nem um nem outro possam prosseguir o seu caminho, bloqueando o sistema, de forma total ou parcial.

[12] refere 4 condições necessárias para a existência de *deadlocks*:

1. Uma tarefa requer controlo exclusivo dos recursos por elas alocados ("condição de exclusão mútua");

2. Uma tarefa mantém um determinado recurso ocupado enquanto espera pela alocação de um novo recurso ("condição de espera");

3. Os recursos não podem ser removidos de uma tarefa até esta os utilizar por completo ("condição de não preempção");

4. Formação de uma cadeia de tarefas circular, na qual cada uma tenta alocar o recurso seguinte sem sucesso ("condição de espera circular").

Na figura 2.23 está ilustrado um exemplo de ocorrência de deadlock, onde as quatro condições são verificadas. Uma vez considerado um exemplo de tráfego de automóveis, cada veículo requer o controlo exclusivo da sua posição, não podendo ser ocupada por mais nenhum. Além disso, todos os veículos seguram o seu espaço enquanto esperam pela possibilidade de avançar. Sendo eles veículos numa estrada, também não podem simplesmente ser removidos da sua posição. Por fim, verifica-se uma espera circular, a qual é originada devido à geometria dos caminhos e à direção tomada pelos veículos.

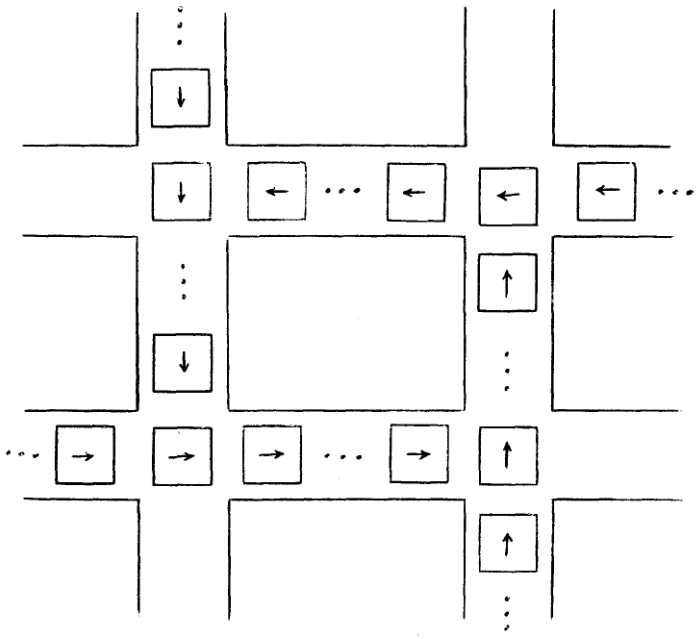


Figura 2.23: Exemplo de ocorrência de deadlock para tráfego de veículos [12].

Se pelo menos uma das condições acima mencionadas puder ser prevenida, evita-se, consequentemente, a ocorrência de *deadlocks*.

Para prevenção das condições, [37] sugere determinados cuidados que devem ser tidos em conta quando se dimensiona um sistema, com possibilidade de recursos partilhados. Como princípio base, não deve haver a possibilidade de uma tarefa requerer um recurso enquanto outro recurso estiver a ela alocado, isto se houver a mínima hipótese desse recurso puder vir a ser requerido por outra tarefa. Contudo, se esta abordagem não tiver sido aplicada ao sistema, e um recurso for negado a uma tarefa, ela própria deve encarregar-se de procurar alternativas. Além disso, outra boa prática consiste em utilizar uma requisição dos recursos de forma coletiva, de modo a que cada tarefa não possa prosseguir sem que todos os recursos tenham sido obtidos.

Também [13] analisa a existência de várias situações de *deadlocks*, propondo como resolução das mesmas uma abordagem de coordenação altruísta, na qual os robôs estão preparados para ceder na alocação de um recurso face a outro veículo, mesmo que não contribua para seu próprio

beneficio. A coordenação é conseguida através de comunicação direta entre eles. As situações de deadlock abordadas aparecem ilustradas na figura 2.24. Entre elas, podemos observar casos como (g), (h) e (i) em que a resolução da situação de bloqueio é feita movendo um dos robôs para trás até que apareça um nó que permita efetuar a espera para dar passagem. Por sua vez os últimos três casos assemelham-se ao anterior, no entanto, como a comunicação está a ser estabelecida entre r e s, e ambos estão bloqueados, um dos robôs efetuará uma rotação de modo a poder comunicar com o robô nas suas costas, podendo depois entre eles resolver o bloqueio.

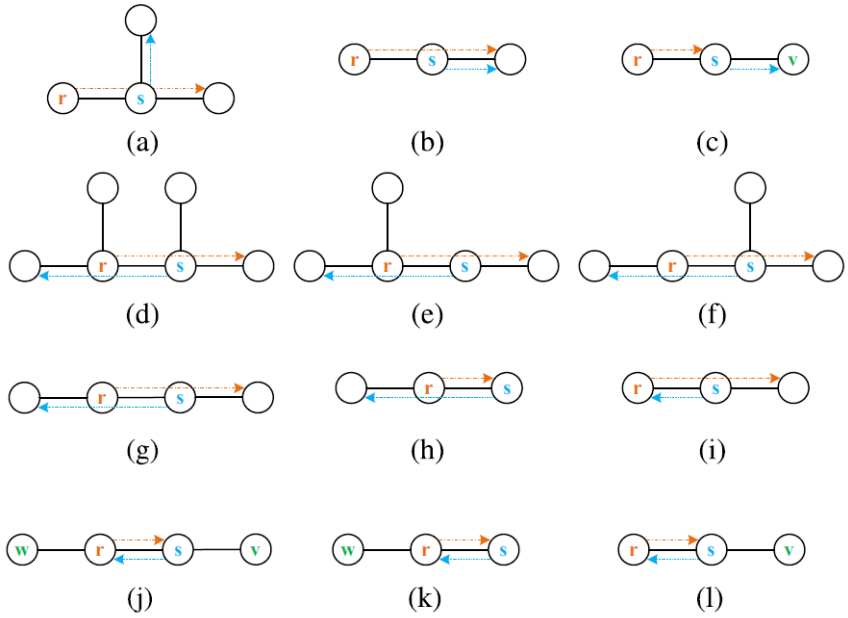


Figura 2.24: Exemplos de situações de *deadlocks* [13].

Outras abordagens podem ser seguidas com vista a resolução deste conjunto de situações. Entre elas, destaca-se a utilização de redes de Petri [38] e ainda modelos de autómatos finitos [39].

Além de *deadlocks*, também *livelocks* podem ocorrer num sistema de recursos partilhados. Estes, por sua vez, acontecem quando um robô está continuamente a tentar alocar um recurso mas fica bloqueado, uma vez que está continuamente a ser ocupado por outros AGV, figura 2.25.

Uma alternativa de resolução de *livelocks* é proposta por [14]. Este sugere uma modificação na ordem com que os trajetos dos AGV são replaneados, quando se verifica paragens durante um determinado período de tempo. Na figura 2.26 a posição inicial dos AGV é dada pelos quadrados verde e azul, respetivamente AGV1 e AGV2. Como o AGV2 atinge a sua posição final primeiro que o AGV1, um bloqueio entre ambos é originado no corredor, devido à ausência de caminhos livres possíveis. Uma vez que o AGV1 planeia o seu trajeto primeiro que o AGV2, não consegue saber em avanço o seu caminho pretendido. Por esse motivo a alternativa passa por alterar a ordem das prioridades no planeamento sempre que se detete situação de bloqueio, no instante atual e seguinte. Assim sendo, posteriormente a isso, o AGV2 passa a planear primeiro o seu

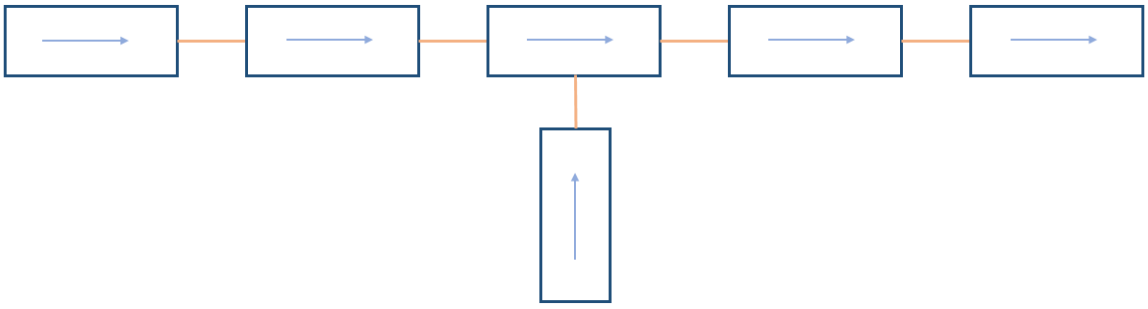


Figura 2.25: Exemplo de uma situação de ocorrência de *livelock*.

trajeto, ficando o AGV1 a conhecer as suas intenções. Com base nesse conhecimento, já pode planear um recuo de modo a resolver a situação de *livelock*.

Figura 2.26: Exemplo de uma situação de ocorrência de *livelock* [14].

Capítulo 3

Método de Planeamento de Trajetórias Implementado

Sendo o objetivo fulcral desta dissertação a coordenação eficiente do movimento de múltiplos robôs, foi necessário adotar um método de planeamento de trajetórias que, além de evitar situações de colisão e de bloqueios, fosse capaz de otimizar o tempo de execução médio e final do conjunto. Deste modo, dois passos fundamentais foram seguidos: em primeiro lugar, procedeu-se à decomposição em células do espaço de atuação ($C_{\text{espaço}}$) nas suas zonas livres (C_{livre}) e zonas ocupadas com obstáculos ($C_{\text{obstáculos}}$) e, por final, foi implementado um algoritmo de pesquisa com vista a determinação do caminho ótimo dentro do conjunto de possibilidades geradas. Ambos serão apresentados, respetivamente, nas secções 3.1 e 3.2.

3.1 Decomposição do $C_{\text{espaço}}$ em células

Uma vez definida a aplicação de uma decomposição do $C_{\text{espaço}}$ com recurso a células, várias abordagens poderiam ser seguidas, consoante o nível de precisão e complexidade pretendido. Entre as diversas existentes e referidas na secção 2.2.1.2, optou-se pela utilização de uma decomposição aproximada por células fixas, com o intuito de reduzir a complexidade computacional associada ao planeamento. Esta é conseguida com recurso a formas geométricas mais simples (quadrados) e de tamanho invariável ao longo de todo o $C_{\text{espaço}}$, como será apresentado e descrito na secção 3.1.1.

3.1.1 Decomposição aproximada por células fixas

Esta metodologia assenta na divisão do mapa utilizado em células com tamanho igual e geometria fixa, através das quais é representado o estado livre ou ocupado das diversas zonas pertencentes ao espaço de atuação.

Dependendo da forma dos obstáculos considerados e da geometria utilizada para as células, pode ser conseguida ou não uma distinção clara entre o C_{livre} e o $C_{\text{obstáculos}}$. Quanto mais nítida for essa divisão, maior será também a probabilidade de obter um caminho viável e mais eficiente

desde o ponto de partida até ao ponto de chegada pré-definido para o robô. Para tal ser possível, é necessário diminuir o tamanho das células tanto quanto desejável tendo sempre em consideração que essa diminuição implicará um aumento do número de iterações no planeamento e, consequentemente, um aumento da complexidade do algoritmo.

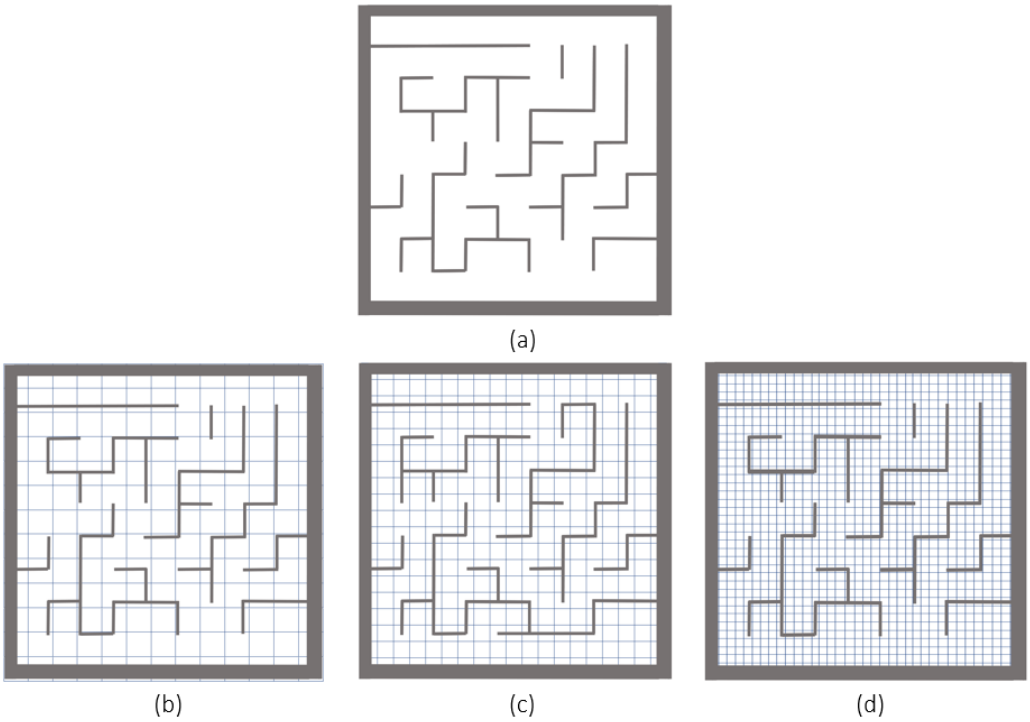


Figura 3.1: Exemplos (b), (c) e (d) de decomposição aproximada por célula fixa do mapa representado em (a).

É de notar que a falta de exatidão na divisão entre o C_{livre} e o $C_{\text{obstáculos}}$ conduz à existência de células semi-ocupadas, que não podem ser consideradas como caminho seguro no planeamento e que, por isso, são vistas como totalmente ocupadas. Esse detalhe faz com que, quando utilizada baixa precisão, isto é, células de tamanho elevado, possa não ser possível obter nenhum caminho totalmente livre de obstáculos entre o ponto inicial e final. Esse pormenor pode ser observado na decomposição representada pela figura 3.1(b), na qual diversas zonas do mapa ficaram completamente obstruídas. Por sua vez, as decomposições das figuras 3.1(c) e 3.1(d), já apresentam ambas caminhos viáveis para atingir qualquer ponto do mapa, contudo, diferem no tamanho do conjunto solução disponibilizado.

Tabela 3.1: Número total de células utilizadas nas decomposições apresentadas na figura 3.1

Decomposição	Número total de células
(b)	169
(c)	361
(d)	1444

É importante determinar um equilíbrio entre a otimização pretendida para os caminhos planeados e a complexidade que vai ser necessária suportar para os conseguir obter.

No caso particular desta dissertação, a escolha recaiu sobre uma decomposição em 361 células. Tanto a distância entre obstáculos como as dimensões dos robôs a utilizar foram tidos como parâmetros decisivos nessa escolha. Sendo considerados para o planeamento os pontos centrais de cada célula e a necessidade de manter alguma margem de segurança face aos obstáculos, observando as dimensões dos robôs detalhadas na secção 5.2, uma diminuição do tamanho da célula e consequente aumento de complexidade, não traria vantagem significativa na definição das rotas. Assumindo o robô posicionado no centro da célula, o espaço livre entre a sua plataforma e as paredes da plataforma será de aproximadamente 2,6 centímetros. Esta distância tenderia ainda a ser inferior se fosse tida em conta uma decomposição em células mais pequenas, conduzindo a pouca segurança no movimento do robô.

3.2 Algoritmo de pesquisa TEA*

Assim que decomposto o espaço de atuação, é necessário recorrer a um algoritmo de pesquisa que seja capaz de selecionar um caminho ótimo no que respeita a tempos de execução. Visto que é fundamental evitar colisões e situações de bloqueio entre robôs ao longo do tempo, o algoritmo TEA* (Time Enhanced A*) alia a si a vantagem de conseguir planear com base nas posições futuras de cada veículo em circulação. Tal é possível devido à introdução do conceito de camada temporal para simbolizar cada instante do planeamento. Assim, o algoritmo TEA* consegue promover uma pesquisa com base no algoritmo A*, desde um ponto de partida até um ponto de chegada, ao longo de diversas camadas temporais, tantas quanto necessário.

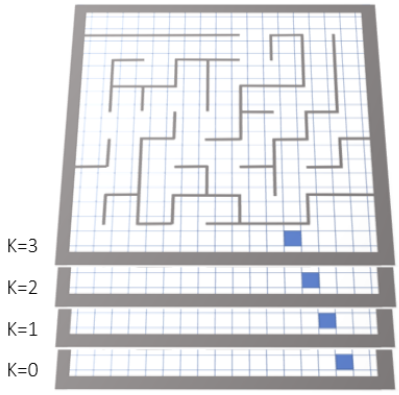


Figura 3.2: Propagação do movimento ao longo das camadas temporais (K) utilizadas pelo TEA*.

No que respeita ao algoritmo A*, este caracteriza-se pela possibilidade de efetuar pesquisas ótimas, isto é, dado um conjunto constituído por diversas células e ligações viáveis entre elas, o algoritmo é capaz de determinar qual a sequência que minimiza uma função custo, a qual será dada pela soma dos custos associados a cada ligação. Estes custos dependem do fator que se

pretende otimizar, podendo variar entre tempo, distância, taxas de circulação, entre outros. No caso particular desta dissertação, o fator que se pretende minimizar é o tempo de execução total de um conjunto de tarefas, o qual não dependerá necessariamente da distância. Situações de trajetos mais curtos que apresentem um elevado número de trocas de direção ou constrangimentos na circulação que atrasem o movimento são bons exemplos dessa não dependência.

Ainda no que se refere aos custos utilizados, o algoritmo A* explora cada vizinhança de uma célula, procurando o custo mais baixo ao longo de todo o trajeto. Para isso, para cada célula C expandida, entra em consideração com o custo somado desde o ponto de partida até ao ponto atual bem como com o custo desde o ponto atual até ao ponto de chegada, *g(C)* e *h(C)* respetivamente, sendo que o primeiro depende da sequência de exploração efetuada até ao momento.

$$f(C) = g(C) + h(C) \tag{3.1}$$

Além dos custos já referidos, também os conceitos de células abertas e células fechadas aparecem aliados à pesquisa por A*. As células abertas são geradas após uma expansão de vizinhança por parte de outra célula. Diz-se, portanto, que foram visitadas e os custos atualizados. Para que sejam tidas em conta no planeamento como células passíveis de seguir são colocadas numa lista, a qual é ordenada por custos crescentes de modo a otimizar a pesquisa. Por sua vez, uma célula fechada, além de visitada já foi expandida, isto é, já deu origem a uma nova vizinhança de células abertas. Quando a célula, cujas coordenadas correspondem às do ponto final desejado para o robô, é fechada significa que se obteve o caminho ótimo entre o ponto de partida e o ponto de chegada. Uma vez que cada célula vai deixando um rasto da sua origem, isto é, guarda a informação da célula que a gerou, o trajeto pode ser conhecido, posteriormente, através de uma pesquisa inversa desde o ponto de chegada até ao ponto de partida.

O pseudocódigo do A* é apresentado no algoritmo 1 e o mesmo é exemplificado graficamente, passo a passo, no anexo A. No que respeita à simbologia utilizada, O representa o conjunto das células abertas, T_{robots} e $T_{camadas}$ designam, respetivamente, o número total de robôs e camadas temporais em utilização, δ o custo temporal associado ao deslocamento entre células e $C_{inicial}$, $C_{corrente}$, C_{final}, $C_{vizinha}$ e $C_{pai}(C)$ identificam, respetivamente, a célula de partida, a célula atual, a célula de chegada, a célula vizinha e a célula pai que deu origem à célula C.

No que respeita ao custo δ, o mesmo dependerá da posição relativa da célula vizinha face à célula corrente. Isto significa que, mediante o conjunto vizinhança escolhido, δ poderá tomar diferentes valores. Focando numa vizinhança de conetividade 8, como ilustrado nas figuras 3.3 e 3.4 (b), é possível observar que a distância para percorrer um segmento horizontal ou vertical entre células será inferior à distância de percorrer uma diagonal. Decompondo o espaço em células quadradas, a proporção entre ambas as distâncias é dada por $1{:}\sqrt{2}$. Assumindo uma velocidade de valor constante e igual para cada ligação entre células, pode-se afirmar, então, que o tempo de percorrer as mesmas também variará de igual forma. Consequentemente, pode-se assumir a existência de um custo δ diferente para cada situação, diferindo entre eles na proporção anteriormente referida.

Algoritmo 1: A*

input : $C_{inicial}$ e C_{final}
output: Caminho ótimo

$C_{inicial}$ é inserida em O com $f(C_{inicial}) = h(C_{inicial})$;
while $(O \notin \emptyset)$ *or* $(i < T_{iter})$ **do**
  Retira-se de O célula com menor custo f;
  São geradas as células vizinhas sucessoras;
  while $C_{vizinha}$ **do**
    if $(state(C_{vizinha}) = CLOSED)$ *OR* $(state(C_{vizinha}) = OBSTACLE)$ **then**
      continue;
    else if $C_{vizinha} \notin O$ **then**
      $g(C_{vizinha}) = g(C_{corrente}) + \delta$;
      $h(C_{vizinha}) = heurstica(C_{vizinha}, C_{final})$;
      $f(C_{vizinha}) = g(C_{vizinha}) + h(C_{vizinha})$;
      $C_{pai}(C_{vizinha}) = C_{corrente}$;
      É inserida $C_{vizinha}$ em O;
    else if $C_{vizinha} \in O$ **then**
      if $g(C_{vizinha}) > g(C_{corrente}) + \delta$ **then**
        $g(C_{vizinha}) = g(C_{corrente}) + \delta$;
        $C_{pai}(C_{vizinha}) = C_{corrente}$;
  end
  $state(C_{corrente}) = CLOSED$;
end

Como heurística para o cálculo do custo *h(C)*, é utilizada a distância euclidiana, a partir da qual se representa a distância de uma célula à célula destino, em linha reta.

$$h(C) = \sqrt{(C_{vizinha_x} - C_{final_x})^2 + (C_{vizinha_y} - C_{final_y})^2} \tag{3.2}$$

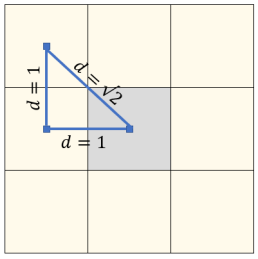


Figura 3.3: Distâncias entre células unitárias considerando um conjunto vizinhança de conetividade 8.

Através da introdução do domínio temporal ao algoritmo de pesquisa A*, a vizinhança da célula corrente passa a pertencer à camada temporal seguinte, tal como representado na figura 3.4 (a) e (b). Este detalhe permite associar a cada célula o instante para o qual ela ficará ocupada pelo robô. Desta forma, todo o seu trajeto será conhecido temporalmente, possibilitando o conhecimento dos pontos de colisão com outros veículos que efetuem o planeamento posteriormente.

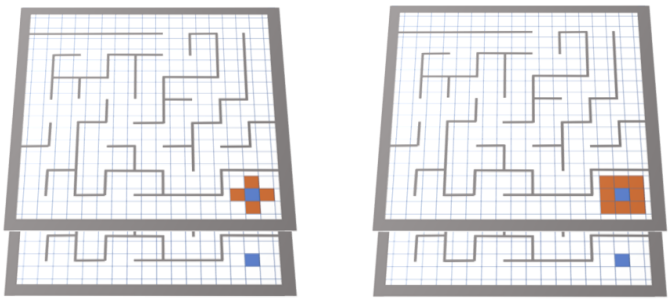


Figura 3.4: Expansão temporal da vizinhança da célula corrente da camada temporal 0 (K=0) para a camada temporal 1 (K=1), considerando: (a) conetividade 4 (b) conetividade 8.

Inicialmente, as posições de todos os robôs são colocadas como obstáculo em K={0,1}. Deste modo, evita-se que robôs de menor prioridade possam influenciar o planeamento de robôs de maior prioridade. Nesse sentido, os de menor prioridade têm de adaptar a sua posição mediante os caminhos já planeados e colocados como obstáculos ao longo das camadas temporais. Essa situação aparece ilustrada na figura 3.5. Se o robô verde, cuja prioridade é inferior, pudesse manter a sua posição como obstáculo para K=[0,T_{MAX}], tentaria ficar o mais próximo possível do seu ponto final e não permitiria que o robô amarelo conseguisse determinar um caminho viável para atingir a sua célula objetivo. Não conseguindo manter a sua posição como obstáculo além da segunda camada temporal, o robô amarelo planeia o seu caminho, coloca-o como obstáculo no tempo e, consequentemente, força o robô verde a desviar a sua posição até que ele consiga ultrapassar o entroncamento. Apenas depois o verde poderá ir em direção à célula por si pretendida.

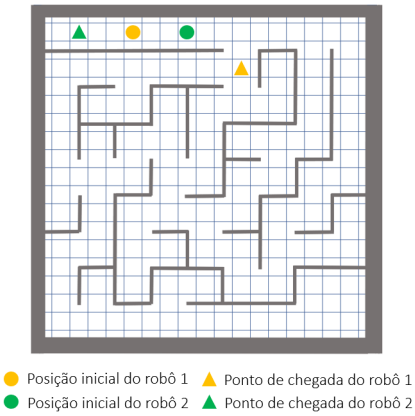


Figura 3.5: Situação de bloqueio em que o robô verde não poderá permanecer imóvel na sua posição, sendo ele o detentor da menor prioridade.

Adicionalmente, é considerada a hipótese de o robô manter a sua posição entre instantes consecutivos do planeamento, conseguida apenas por uma variação temporal das suas coordenadas. Assim, casos em que seja detetada uma possível colisão entre robôs e, por isso planeado um trajeto mais longo face ao trajeto ótimo, passam a apresentar soluções mais eficientes, com aplicação de uma situação de paragem e espera.

Contudo, nem sempre esta situação é temporalmente vantajosa, uma vez que o caminho alternativo pode tornar-se mais rápido que a primeira opção quando lhe somada o tempo de espera ao tempo do percurso. Por esse motivo, associado às situações de paragem voluntária, é necessário introduzir um novo custo δ, não nulo. Ainda que não exista deslocamento da sua posição no espaço entre instantes do planeamento, o tempo progrediu. Não sendo a distância mas sim o tempo de execução final o fator decisivo para a escolha do caminho ótimo, é, portanto, essencial refletir essa paragem no custo total do trajeto. Além disso, a introdução de um custo nulo levaria a uma situação de paragem ilimitada no tempo. Isto porque, será sempre dada prioridade à exploração das células que minimizem o custo de deslocamento. Assim, se aos momentos de paragem estiver associado um custo positivo e não nulo, a cada instante que passa, o custo total será crescente e poderá exceder o custo relativo a movimento efetivo. Assegura-se, desta forma, a continuidade na procura de novas células que aproximem o robô do seu ponto final pretendido.

De realçar ainda que o número de camadas temporais a considerar para o planeamento está dependente do número de células necessárias a percorrer para atingir o objetivo. Está, portanto, dependente das dimensões, geometria e decomposição do espaço de atuação assim como do número de obstáculos que forem encontrados pelo robô durante o seu trajeto.

O pseudocódigo do TEA*, para múltiplos robôs, é apresentado no algoritmo 2 e recorre à mesma simbologia utilizada e referida no algoritmo 1, à qual é apenas acrescentado $K(C)$ para representar a camada temporal K associada a uma célula C. De realçar a existência de prioridades no planeamento entre os diversos robôs. O espaço de atuação livre à medida que se desce na escala de prioridades será cada vez mais reduzido devido à introdução dos caminhos já planeados e, respetiva vizinhança, como obstáculos.

Para a aplicação do algoritmo, as células na vizinhança serão determinadas com base numa pesquisa de conetividade 8. Isto significa que qualquer célula que partilhe um só vértice ou aresta com a célula corrente fará parte do conjunto vizinhança. Deste modo, dá-se a possibilidade aos robôs de efetuarem movimentos diagonais, desde que os mesmos não representem colisão com nenhum obstáculo. Este detalhe permitirá, numa boa parte dos casos, otimizar o tempo final dos trajetos.

O TEA* apresenta ainda a particularidade de ser executado de um modo online. Embora os caminhos dos robôs sejam planeados temporalmente, podem existir atrasos ou falhas no movimento que conduzam à existência de erro nas posições estimadas. Posto isso, a cada vez que são recebidas leituras das posições atuais dos robôs, os caminhos são constantemente recalculados, contrariamente a um método offline, no qual o planeamento é todo efetuado numa fase prévia ao movimento dos robôs. Sendo o TEA* online, as possibilidades de garantir um controlo do tráfego seguro e eficaz, isto é, livre de colisões e bloqueios, aumentam significativamente.

3.2.1 Complementos Adicionais do TEA*

O algoritmo TEA* procura fundamentalmente promover um planeamento de caminhos ótimos no tempo, garantindo a ausência de colisão entre diversos veículos. De facto, tal é conseguido pela

Algoritmo 2: TEA*

input : $C_{inicial}$ e C_{final} de cada robô
output: Caminho ótimo para todos os robôs
while $robot_i < T_{robots}$ **do**
  $C_{inicial}$ é inserida em O com $f(C_{inicial}) = h(C_{inicial})$;
  while $(O \notin \emptyset)$ OR $(i < T_{iter})$ **do**
    Retira-se de O célula com menor custo f;
    if $C_{corrente} = C_{final}$ **then**
      break;
    else if $camada_i > T_{camadas}$ **then**
      break;
    else
      São geradas as células vizinhas sucessoras;
      $K(C_{vizinha}) = K(C_{corrente}) + 1$;
      while $C_{vizinha}$ **do**
        if $(state(C_{vizinha}) = CLOSED)$ OR $(state(C_{vizinha}) = OBSTACLE)$ **then**
          continue;
        else if $C_{vizinha} \notin O$ **then**
          São calculados os custos f($C_{vizinha}$) e g($C_{vizinha}$);
          $C_{pai}(C_{vizinha}) = C_{corrente}$;
          É inserida $C_{vizinha}$ em O;
        else if $C_{vizinha} \in O$ **then**
          if $g(C_{vizinha}) > g(C_{corrente}) + \delta$ **then**
            $g(C_{vizinha}) = g(C_{corrente}) + \delta$;
            $C_{pai}(C_{vizinha}) = C_{corrente}$;
      end
      $state(C_{corrente}) = CLOSED$;
  end
  Armazenamento do caminho planeado;
  O caminho planeado e sua vizinhança são colocados como obstáculo para os restantes robôs de menor prioridade;
end

capacidade de planear caminhos livres de obstáculos, os quais podem ser estáticos ou dinâmicos desde que sejam conhecidos ao longo de várias camadas temporais. Contudo, quando se aborda um problema de coordenação num espaço de atuação limitado é necessário ter em conta que as possibilidades de atingir a célula objetivo também são limitadas. Isso faz com que situações de bloqueio no tempo possam acontecer e, consequentemente, nem sempre seja possível obter caminhos alternativos para todos os robôs. Esses bloqueios tornam-se ainda mais frequentes atendendo à existência de um ordem fixa de planeamento entre os robôs. Os de maior prioridade planeiam sem se preocupar com a possibilidade ou não de bloquearem a passagem dos restantes.

Um bom exemplo de bloqueio aparece ilustrado na figura 3.6, na qual a cor amarela é utilizada para o robô de maior prioridade. Não existindo qualquer preocupação do mesmo face ao robô verde, este último terá de recuar para lhe dar passagem. No entanto, por mais que recue, não

existe uma única célula que garanta a sua espera sem interferir com o robô amarelo. Acontece que, devido à alta prioridade, a posição do robô amarelo será colocada como obstáculo ao longo de todas as camadas temporais até atingir o seu ponto de chegada. Isso o fará avançar, acreditando não existir qualquer outro robô no seu caminho e originando, consequentemente, um ponto de colisão. Ainda que o robô verde tivesse mais células para recuar além do ponto objetivo do amarelo, este último permaneceria imóvel nessa célula ao longo do tempo, não permitindo que o verde planeasse o seu caminho sem existir colisão, saindo, portanto, o objetivo do conjunto prejudicado.

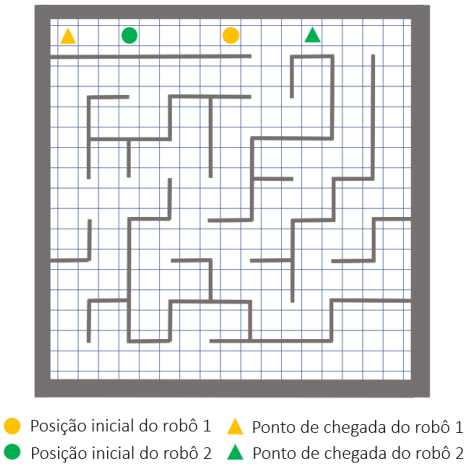


Figura 3.6: Situação de bloqueio com previsível colisão entre os robôs se o robô amarelo mantiver a prioridade mais alta.

Seria, então, necessário instruir todos os robôs com uma capacidade altruísta, isto é, poder abdicar da sua prioridade em prol de uma maior eficiência de todo o conjunto. Nesse seguimento, optou-se pela utilização de uma troca de prioridades ao longo da cadeia hierárquica, a qual é aplicada sempre que é detetada a incapacidade de um robô planear o seu caminho. Essa incapacidade é conhecida quando a lista de células abertas O fica vazia antes de a célula objetivo ser explorada ou ainda quando se verifica uma sobreposição da posição do robô de menor prioridade com a posição da célula de chegada do robô de maior prioridade, depois de este já a ter atingido.

A figura 3.7 representa o comportamento de ambos os robôs utilizados na figura 3.6, após a implementação do método de troca de prioridades, mantendo iguais as posições iniciais e finais de cada um. Nesta situação particular, o robô verde não tem como recuar para permitir o robô amarelo atingir o seu objetivo. Contudo, tendo o robô amarelo a prioridade no planeamento, o verde chega a um ponto que fica rodeado de obstáculos, inclusive a sua própria posição. Deste modo, deixam de existir células passíveis de exploração e é detetado o bloqueio. Nesse momento, o robô verde troca a ordem de planeamento com o robô amarelo, assumindo assim uma maior prioridade. Ao planear, então, o seu trajeto e ao colocá-lo de seguida como obstáculo ao longo das camadas temporais, faz com que o robô amarelo se desvie da sua trajetória e espere noutra célula para lhe dar passagem. Apenas depois do robô verde passar o cruzamento é que o robô amarelo avança em direção ao seu objetivo, mantendo sempre a distância de segurança, equivalente a uma célula, em relação ao companheiro.

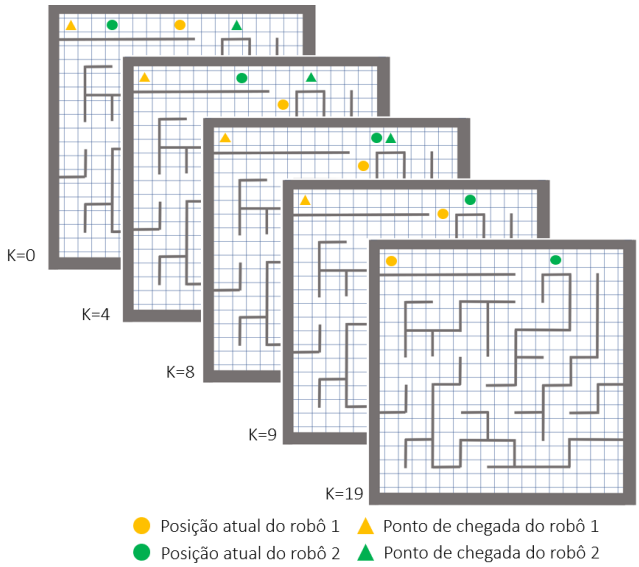


Figura 3.7: Posições planeadas ao longo das camadas temporais(K) para a situação de bloqueio da figura 3.6.

Por sua vez, a figura 3.8 representa uma situação de bloqueio bastante semelhante à da figura 3.6, contudo diferem uma da outra na forma como ambas as situações são detetadas. Enquanto que no caso da figura 3.6, o robô verde não tem como recuar para o robô amarelo atingir o seu objetivo, no caso da figura 3.8 isso já não acontece. O robô verde passa a ter uma célula onde consegue permanecer imóvel sem interferir com o caminho planeado pelo amarelo. Desse modo, se o ponto de chegada fosse considerado como obstáculo em todas as camadas, depois do mesmo ser atingido, o robô verde exploraria constantemente a sua própria célula mas para instantes temporais diferentes, processo que se iria repetir até ser excedido um número máximo de iterações pré-definido. Para evitar isso, o ponto de chegada do robô amarelo apenas é colocado como obstáculo na camada temporal em que é atingido, ficando a célula livre em camadas superiores. É importante reparar que isso leva a que o robô verde planeie um caminho com um ponto de colisão, no entanto, é esse mesmo detalhe que permite detetar o bloqueio e, consecutivamente, ativar a troca de prioridades entre ambos os robôs. Posteriormente, efetua-se novamente o planeamento das trajetórias.

É possível observar ainda situações para as quais, colocar o ponto de chegada do robô de maior prioridade como obstáculo, não influenciaria o objetivo do robô verde. Este seria, portanto, capaz de pesquisar caminhos alternativos. Porém, esse desvio de trajetória para evitar uma célula pode implicar um caminho bastante mais ineficiente em termos de tempo de execução. É o caso da situação representada nas figuras 3.9(a) e 3.9(b), na qual o robô verde conseguiria planear um caminho significativamente mais longo, libertando a passagem ao robô amarelo.

No entanto, se o robô amarelo abdicar da sua prioridade face ao verde, o custo relativo ao trajeto do último será bastante inferior, assim como o tempo de execução final do conjunto também diminuirá. Isso é, novamente, conseguido pela deteção da sobreposição da posição corrente do verde com o ponto de chegada do amarelo, camadas temporais após o mesmo ter explorado e

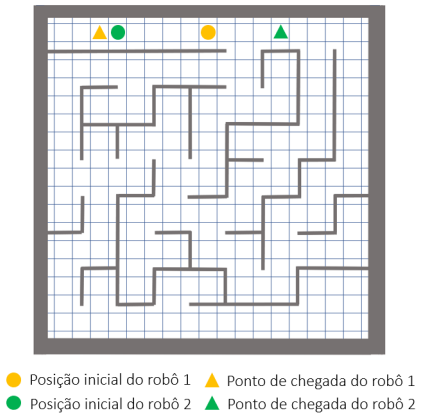


Figura 3.8: Situação de bloqueio do robô verde, com colisão prevenida.

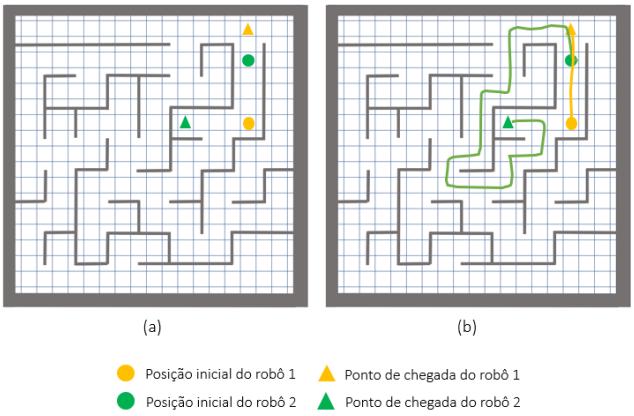


Figura 3.9: Situação de bloqueio com otimização do tempo total de execução do conjunto dos dois robôs através de troca de prioridades.

fechado essa célula. Assim como demonstrado na figura 3.10, o robô amarelo passa a recuar para ceder passagem ao verde e só de seguida é que avança em direção ao seu objetivo.

Nem sempre a otimização descrita pode ser aplicada com sucesso. Mediante as posições iniciais e finais definidas para um par de robôs, pode acontecer um caso de bloqueio (figura 3.11) em que, quer recue um quer recue o outro, o seu companheiro detetará sempre sobreposição da sua posição com o ponto de chegada do outro e terá de optar pelo trajeto mais longo se pretender alcançar o seu objetivo. Nessas situações, os robôs ficariam indefinidamente a trocar as suas prioridades. Para que isso não aconteça, é estabelecido um limite máximo de trocas que, quando excedido, faz com que o robô de maior prioridade coloque o seu ponto de chegada como obstáculo ao longo de todas as camadas temporais que procedem o momento de exploração e fecho da célula objetivo. Assim, sabendo que não terá qualquer hipótese de passar por essa célula, o robô de prioridade mais baixa segue pelo caminho alternativo.

Ainda no que respeita à otimização anteriormente referida, é possível associar a ela uma outra falha quando um robô de menor prioridade atinge a sua célula objetivo numa camada temporal

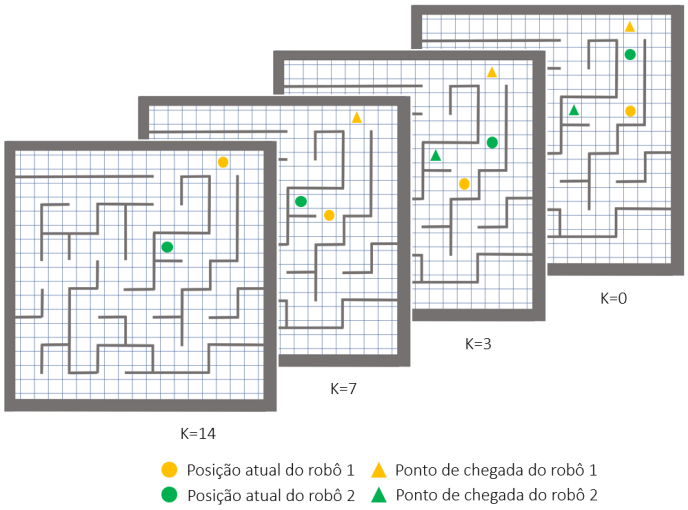


Figura 3.10: Posições planeadas ao longo das camadas temporais(K) para a situação de bloqueio da figura 3.9.

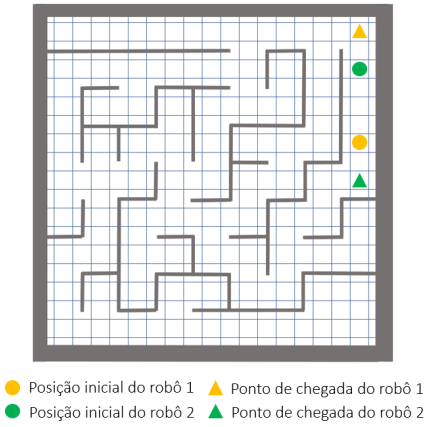


Figura 3.11: Situação de bloqueio sem possibilidade de recorrer à otimização aplicada na figura 3.9.

inferior à camada em que um robô de maior prioridade lá passa. Observando com maior detalhe na figura 3.12, o robô verde, ao detetar possível sobreposição com o ponto de chegada do robô amarelo, procura subir na hierarquia de prioridades, de modo a encurtar o seu trajeto. Tal é-lhe concedido e todo o planeamento correria bem se o robô amarelo não estivesse tão perto da sua célula objetivo, já que isso lhe permite atingi-la ainda antes do robô verde por lá passar. Uma vez que o robô amarelo se encontre lá, na célula permanecerá imóvel ao longo das camadas temporais seguintes, isto porque a pesquisa do algoritmo TEA* para um robô singular termina no momento em que a célula final é explorada e fechada. Contudo, sendo o robô verde o detentor da maior prioridade assume que o seu companheiro não interferirá no seu trajeto e dá-se uma possível colisão.

Para detetar este conjunto de situações, optou-se por realizar uma validação inversa das prioridades. Por outras palavras, após qualquer troca efetuada, verifica-se se um robô de maior prioridade interseta a célula objetivo do companheiro, numa qualquer camada temporal superior à qual ele a alcançou. Em caso afirmativo, volta-se a trocar as prioridades para repor a ordem original, replaneiam-se as trajetórias e passa-se a colocar as células objetivo como obstáculo ao longo das camadas temporais posteriores à mesma ser atingida.

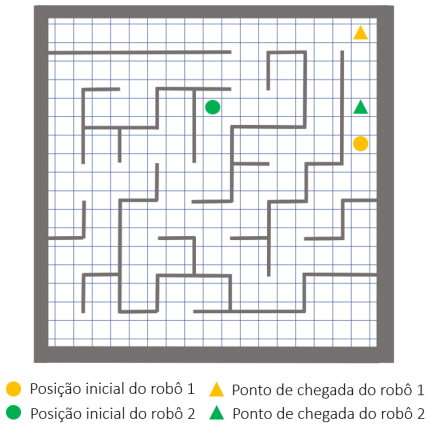


Figura 3.12: Situação de bloqueio que alia a si a necessidade de efetuar uma validação inversa das prioridades.

3.2.2 Modificação proposta ao TEA*

O algoritmo de pesquisa TEA*, tal como proposto em [14, 36] e como descrito acima, pressupõe um planeamento temporal que assume a possibilidade de um robô se deslocar em todas as direções, sem a necessidade de efetuar uma única rotação. De um modo mais simplificado, o algoritmo não entra em conta com a direção atual e considera ser possível o deslocamento do robô desde a célula corrente para qualquer uma das células vizinhas, num mesmo intervalo de tempo. Focando na figura 3.13 (a), tendo em consideração que a direção atual real do robô é Norte, o tempo de deslocamento ao longo do segmento AB será equivalente ao tempo de uma rotação de 90º de Norte para Este, procedida de um deslocamento ao longo do segmento AC. Como é possível verificar nas figuras 3.13 (b) e (c), o robô atinge ambas as células vizinhas na camada temporal seguinte. De facto, isso apenas seria possível se o mesmo fosse omnidirecional, ao invés de diferencial, caso dos robôs utilizados nesta dissertação.

Quando aplicado o algoritmo TEA* tal como descrito, prevê-se que as posições futuras reais estimadas utilizadas no planeamento possam ser influenciadas, pela negativa, pelos atrasos provenientes das rotações. Sendo o TEA* um algoritmo executado de um modo online, dentro de uma determinada cadência as posições reais são novamente conhecidas e todas as trajetórias são replaneadas. Mediante a cadência utilizada, a inexistência de colisões entre os diversos robôs poderá, então, continuar a ser ou não assegurada. Uma cadência baixa conduz a uma atualização das posições menos frequente e, consequentemente, a um maior erro na localização, a cada instante,

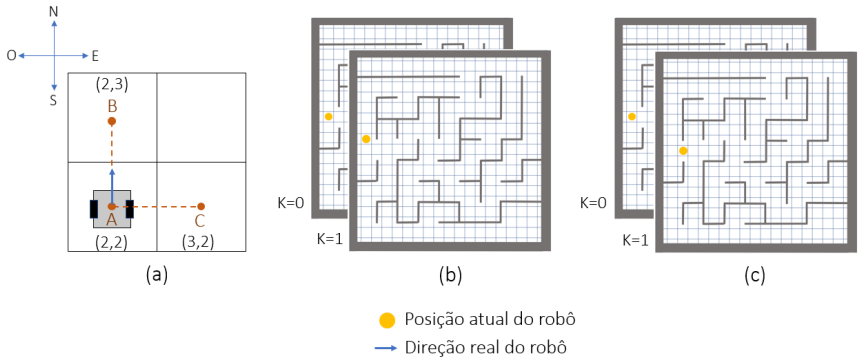


Figura 3.13: (a) Está representado o robô numa porção de mapa decomposto em células quadradas; (b) e (c) Representam o planeamento temporal relativo ao deslocamento no segmento AB e AC, respetivamente, baseando-se na implementação do TEA* anterior à abordagem proposta na secção presente.

dos obstáculos dinâmicos. Assumindo que é utilizado um valor suficientemente grande para assegurar o controlo do tráfego, pode ainda ser analisado o nível de eficiência do planeamento, no que respeita a tempos de execução. Uma má previsão das posições futuras, poderá conduzir os robôs de menor prioridade a definirem caminhos alternativos mais longos para se desviar de obstáculos, em determinadas camadas temporais, que na realidade não existirão. Além disso, havendo uma acumulação maior de erro entre leituras das posições, também o número de mudanças de trajetória poderão ser maiores, atrasando, por sua vez, a chegada à célula objetivo.

Tomando em conta estas considerações acerca da eficácia e eficiência do controlo do movimento dos robôs, é proposta e estudada uma nova abordagem, na expectativa de aproximar o planeamento das posições futuras ao comportamento em ambiente real. Focando nesse objetivo, é introduzida no algoritmo a noção de direção do robô, sendo esta armazenada para cada célula que o robô visita. Deste modo, sabendo a deslocação em XX e YY necessária para atingir uma célula vizinha, sabe-se também a direção a seguir pelo robô. Conhecendo ainda a sua direção atual, é possível saber, pela diferença entre as direções, qual o valor de rotação que terá de ser efetuado.

Sabendo da existência de rotação em determinado momento, melhores equivalências temporais poderão ser estabelecidas. Nesse seguimento, considerou-se para esta abordagem:

- Movimento horizontal, vertical ou diagonal, segundo a direção atual, equivalerá a um deslocamento temporal de uma camada;
- Rotação de 90º procedida de um movimento horizontal, vertical ou diagonal, equivalerá a um deslocamento temporal de duas camadas;
- Rotação de 45º procedida de um movimento horizontal ou vertical, equivalerá a um deslocamento temporal de uma camada;
- Rotação de 45º procedida de um movimento diagonal, equivalerá a um deslocamento temporal de duas camadas;

Recorrendo à situação ilustrada na figura 3.13, o deslocamento temporal entre as células (2,2) e (2,3) mantém-se equivalente a uma camada, enquanto que o deslocamento temporal entre as células (2,2) e (3,2) será de duas camadas. Esta nova abordagem está representada na figura 3.14 (c). Como se pode verificar, na camada intermédia que representa o estado da rotação (K=1), o robô permanece imóvel na mesma célula. A esta paragem é ainda associado um custo δ de modo a refletir o tempo de rotação. Assim o custo total δ de mover da célula (2,2) para a célula (3,2) será a soma do custo de rotação mais o custo associado ao movimento horizontal ao longo do segmento AC. Assim, em situações como a da figura 3.15, que se pretende ir da célula A à célula D, será sempre dada preferência ao movimento na direção atual. Tendo ambos os trajetos o mesmo número de células para deslocar, prevalece aquele com menor número de rotações, de modo a procurar o caminho ótimo em termos de tempo de execução.

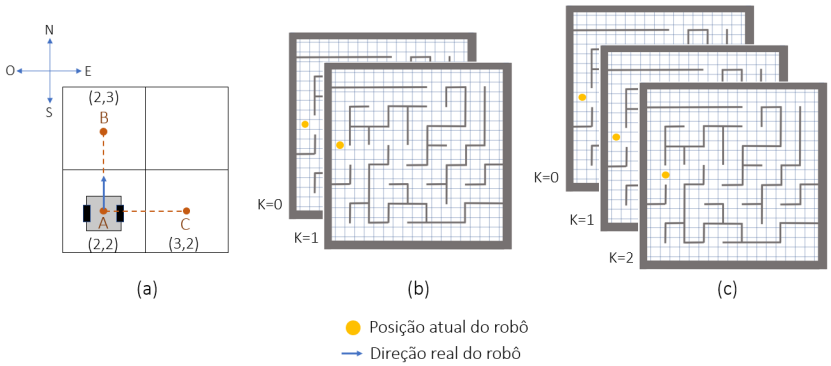


Figura 3.14: (a) Está representado o robô numa porção de mapa decomposto em células quadradas; (b) e (c) Representam o planeamento temporal relativo ao deslocamento no segmento AB e AC, respetivamente, baseando-se na implementação do TEA* posterior à abordagem proposta na secção presente.

De notar que, no que respeita à implementação do algoritmo e recorrendo ao exemplo da figura 3.14 (c), quando a célula (2,2) é expandida e visita a célula (3,2), esta última é colocada na lista dos nós abertos com referência temporal de K=2 e direção Este. Se, posteriormente, a célula for considerada para o caminho ótimo, é possível aferir a paragem em K=1 para efetuar a rotação, com base nas informações guardadas.

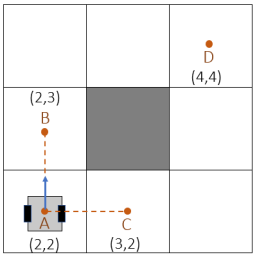


Figura 3.15: Demonstração de dois caminhos equivalentes entre a célula A e D, que diferem entre si no número de rotações a efetuar, sendo a direção atual do robô dada pela seta azul.

3.3 Conclusão

O presente capítulo tinha como objetivo possibilitar uma compreensão mais profunda do algoritmo TEA*, focando detalhadamente nos diversos métodos implementados para resolver diferentes situações de colisão ou deadlock. Com recurso aos mesmos, gera-se a possibilidade de obter um planeamento coordenado, evitando a sobreposição de robôs numa mesma célula, em iguais camadas temporais. Além de se procurar a eficácia do planeamento, aposta-se ainda na eficiência temporal quando confrontado com possíveis colisões. O planeamento de uma situação de espera ou de caminho alternativo está dependente do número de camadas temporais previstas para a trajetória. Através da implementação de trocas de prioridades tornou-se ainda possível a resolução de situações de bloqueio entre robôs. Ao conferir-lhes a possibilidade de abdicar da sua prioridade, prejudicando o tempo de execução da sua tarefa em prol do sistema, obtém-se um planeamento menos vulnerável a falhas e mais otimizado no que respeita a tempos de execução final de todo o conjunto. De realçar ainda o desenvolvimento de uma validação inversa das prioridades, para evitar casos em que robôs de menor prioridade possam atingir a sua célula objetivo em camadas temporais inferiores à passagem de robôs de maior prioridade pela mesma. Embora conduza ao planeamento de um caminho alternativo mais longo, garante-se, mais uma vez, a eficácia do planeamento.

Adicionalmente, foi desenvolvida e proposta uma nova abordagem, complementar ao algoritmo TEA*, com a ambição de aproximar o planeamento temporal das trajetórias reais dos robôs. Por outras palavras, espera-se que, ao diferenciar temporalmente movimentos simples de movimentos com rotação mais prolongada, se consiga estabelecer uma melhor previsão da futura posição dos robôs face a todos os outros que se encontram no espaço de atuação. Passa-se a ter em conta, no planeamento, a utilização de robôs diferenciais, ao invés de omnidirecionais.

Todos os métodos implementados e descritos no capítulo serão testados e validados, posteriormente, no capítulo 5, com simultânea apresentação dos caminhos planeados a três dimensões.

Capítulo 4

Controlo de Trajetória

No presente capítulo será abordado o método de controlo das velocidades dos robôs implementado, com o intuito de aplicar o algoritmo TEA* em sistemas multi-robóticos. Desse modo, será primeiramente apresentado, na secção 4.1, o controlo em malha fechada desenvolvido entre os robôs e o PC que processa o TEA*. Uma vez obtida uma trajetória de referência a seguir, são utilizados controladores para manter o erro de distância e orientação aproximadamente nulo, os quais são descritos na secção 4.2.

4.1 Controlo em Malha Fechada das Velocidades

De modo a validar o algoritmo TEA* simulador e futuramente em ambiente real, é necessário desenvolver um controlo do movimento dos robôs com base nos caminhos planeados. Nesse sentido, é fundamental conhecer as suas posições e orientações ao longo de todo o trajeto para que, com base nessa informação, seja possível tomar a decisão da trajetória a seguir a cada instante e, posteriormente, controlar a velocidade pretendida para os motores. Remete-se, portanto, para a implementação de um controlo em malha fechada, que segue o conceito ilustrado na figura 4.1.

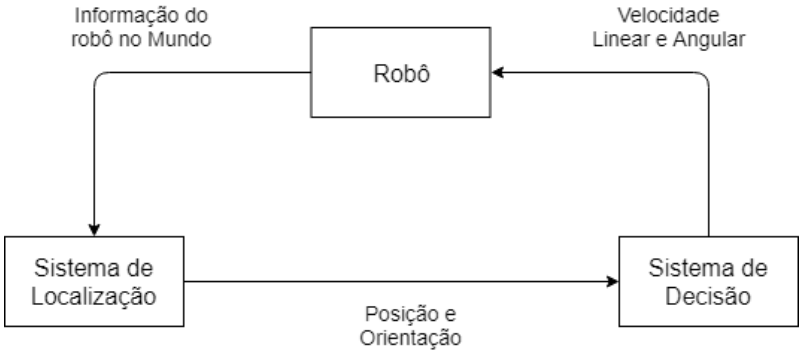


Figura 4.1: Esquema concetual do controlo em malha fechada da trajetória de cada robô.

No caso particular desta dissertação, pretende-se preparar um ambiente de simulação 3D, com base no software SimTwo, que permita testar, em tempo real e de forma realista vários problemas

de coordenação, solucionados com recurso ao TEA*. Pretende-se ainda que o mesmo seja capaz de comunicar via UDP com o IDE Lazarus. Estabelecendo uma comparação com o esquema concetual da figura 4.1, o SimTwo incluirá a presença dos robôs e um sistema de localização próprio e, por sua vez, no Lazarus será aplicado o sistema de decisão.

Para que a aplicação de testes em simulador seja, portanto, possível, será necessário conhecer a posição e orientação de cada robô em intervalos de tempo curtos. Contrariamente a uma situação de ambiente real, em que seria imprescindível um sistema de localização fiável, o próprio simulador disponibiliza já os parâmetros pretendidos, os quais poderão ser enviados para o Lazarus com uma cadência de 40 milissegundos (equivalente ao ciclo de controlo utilizado no SimTwo). Uma vez recebidos, são comparados com a posição e direção pretendida para o robô, sendo estas dadas pela próxima camada temporal do planeamento de trajetória TEA*. Dependendo da distância a esse ponto, são definidas a velocidade linear e angular a utilizar para o movimento do robô, As mesmas são recebidas do lado do SimTwo e aí é feito o cálculo de conversão para obter a velocidade a colocar em cada motor. Este processo repetir-se-à iterativamente e com utilização de uma cadência fixa, quer na receção da informação proveniente do sistema de localização, quer no envio das velocidades para o robô.

Do ponto de vista do Lazarus, as tarefas de receção e envio da informação foram implementadas para correr em simultâneo. Primeiramente, é representado o processo de receção na figura 4.2.

É de notar que, sendo o TEA* um algoritmo de planeamento online, o mesmo será executado diversas vezes ao longo de toda a trajetória de um robô, até o mesmo atingir a sua célula objetivo. Isso permitirá, assim, que atrasos ou avanços no movimento dos robôs, face ao planeado temporalmente, não conduzam a possíveis colisões. Ainda assim, considerou-se não ser necessário planear a cada leitura recebida, proveniente do sistema de localização, uma vez que a diferença de posição e orientação entre leituras seria reduzida. Deste modo, optou-se por executar o planeamento apenas quando um robô atinge a próxima célula pretendida, a qual é dada pela camada temporal 1 ou 2 do trajeto, mediante a existência ou não de rotação (de acordo com abordagem proposta na secção 3.2.2). Apenas esse robô define o seu novo ponto desejado, mantendo-se todos os outros com o movimento atual. No que respeita à decisão do próximo movimento, quatro casos base podem, portanto, ser analisados. Os mesmos aparecem listados de seguida:

- Se o par de coordenadas (x,y) atual de um robô coincidir com o par de coordenadas da célula objetivo, atribui-se valor nulo à velocidade linear e angular, de modo a imobilizar o seu movimento.

- Se o par de coordenadas (x,y) atual de um robô diferir do par de coordenadas planeado para a primeira camada temporal, está-se perante um movimento retilíneo ou perante uma rotação de 45º procedida de movimento retilíneo.

- Se o par de coordenadas (x,y) atual de um robô for igual ao par de coordenadas planeado para a primeira camada temporal, diferindo na direção pretendida, está-se perante um

movimento de rotação de 90º ou perante um movimento circular procedido de movimento retilíneo.

- Se o par de coordenadas (x,y) atual de um robô for igual ao par de coordenadas planeado para a primeira camada temporal, sem diferir na direção pretendida, considera-se uma situação de paragem, colocando o valor nulo na velocidade angular e linear.

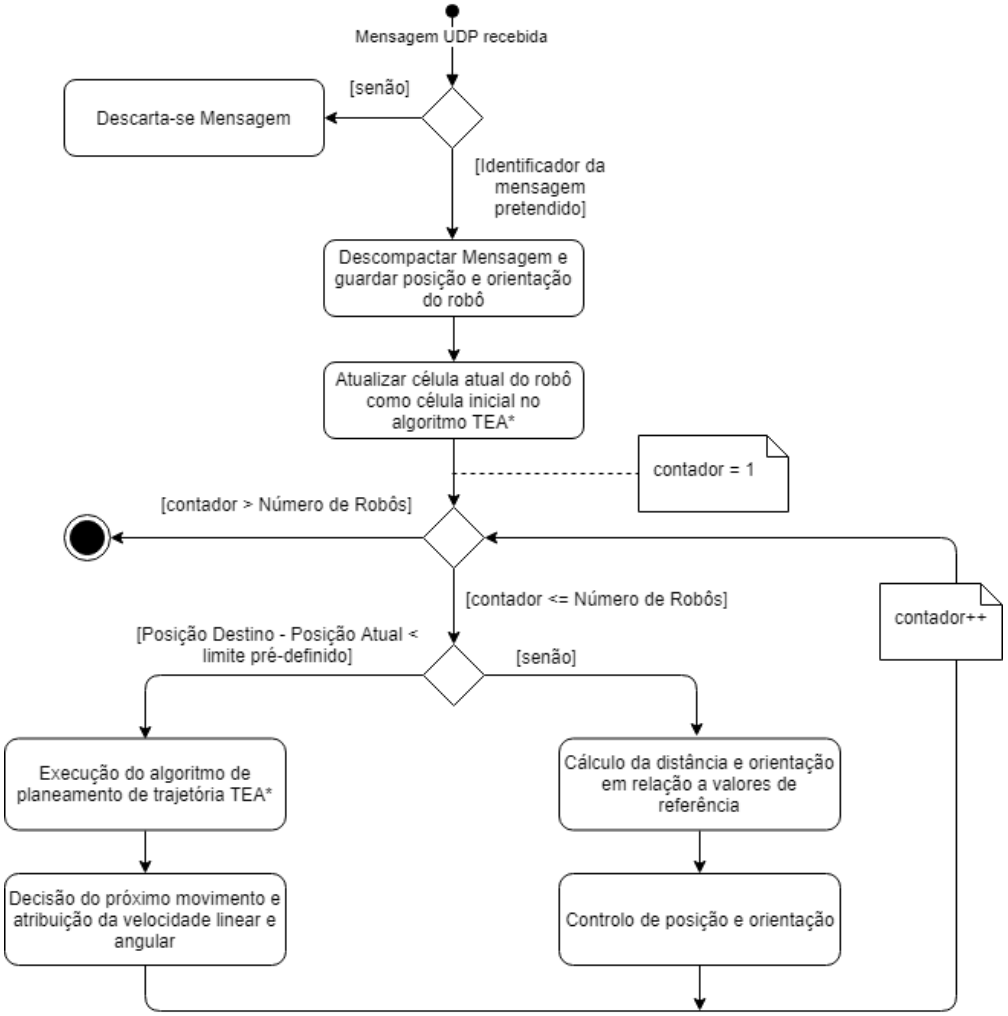
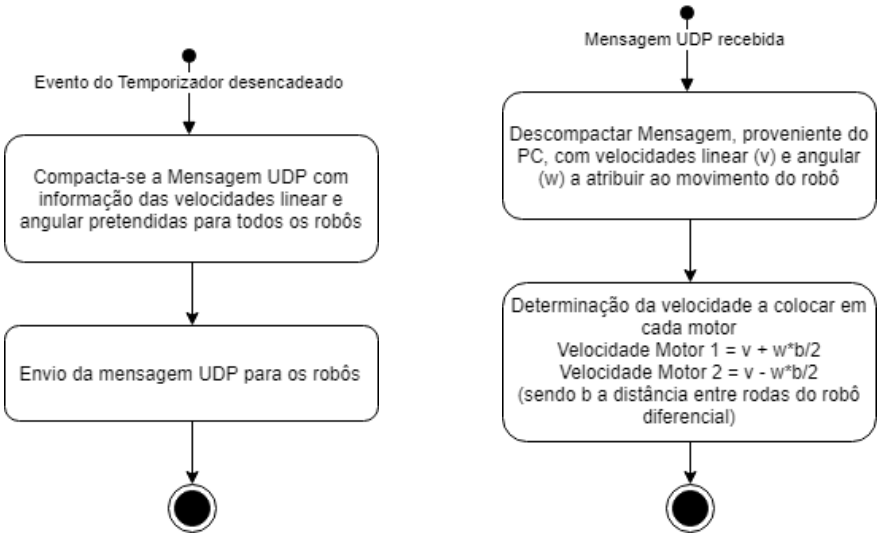


Figura 4.2: Diagrama de atividade do processo de receção de informação na perspetiva do PC.

Tendo sido o próximo movimento decidido e o valor das velocidades linear e angular calculados, é necessário compactar a informação e transmitir a mensagem, via UDP para os robôs. Para isso, é possível recorrer a um temporizador para marcar uma cadência temporal fixa de transmissão. Se esta for igual ao ciclo de controlo utilizado pelo simulador, pode-se garantir assim uma atualização do valor das velocidades em todos os ciclos. Nas figuras 4.3a e 4.3b, são representados os diagramas de atividade do processo de envio receção da mensagem.



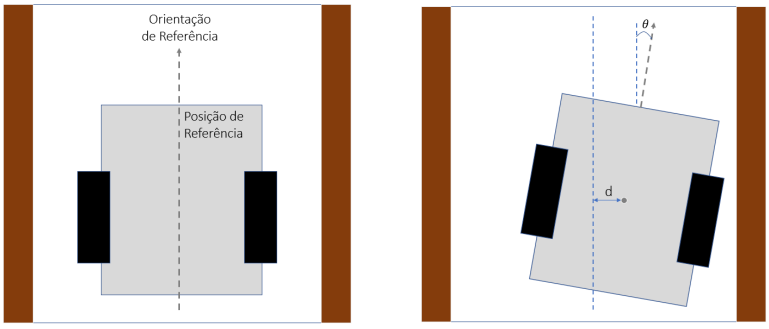
(a) (b)

Figura 4.3: Diagramas de atividade: (a) Processo de envio de informação na perspetiva do PC. (b) Processo de receção de informação na perspetiva de cada robô.

4.2 Controladores de Seguimento de Trajetória

Sendo necessário garantir um movimento preciso dos robôs dentro da plataforma de teste, é fundamental aplicar controladores ao seguimento de trajetória. Desta forma, foram implementados um seguidor de trajetória retilínea e um seguidor de trajetória circular, os quais ajustam a velocidade linear e angular a atribuir ao robô, de modo a colocar a sua posição e orientação dentro de uma gama de referência limitada.

No que respeita ao seguidor de trajetória retilínea, o objetivo passa por colocar a posição do robô centrada entre as paredes presentes na plataforma e a sua orientação coincidente com a direção planeada pelo algoritmo TEA*, tal como apresentado na figura 4.4a.



(a) (b)

Figura 4.4: Seguidor de trajetória retilínea: (a) Posição e orientação do robô dentro dos limites de referência. (b) Posição e orientação do robô fora dos limites de referência: d representa a diferença à posição de referência e θ a diferença ao ângulo de referência.

Para conduzir o robô aos valores de referência pretendidos é mantida a velocidade linear constante e é ajustada a velocidade angular com recurso à equação 4.1. K_θ e K_d representam fatores de ganho de ângulo e distância, respetivamente.

$$w = -K_\theta * \theta - K_d * d \tag{4.1}$$

Focando no seguidor de trajetória circular, pretende-se obter um movimento composto por velocidade linear e angular que permita manter o centro do robô aproximadamente equidistante do centro de rotação. Para isso ser conseguido, é necessário orientar o robô segundo a direção da tangente à circunferência, em cada ponto da trajetória, tal como pode ser observado na figura 4.6a.

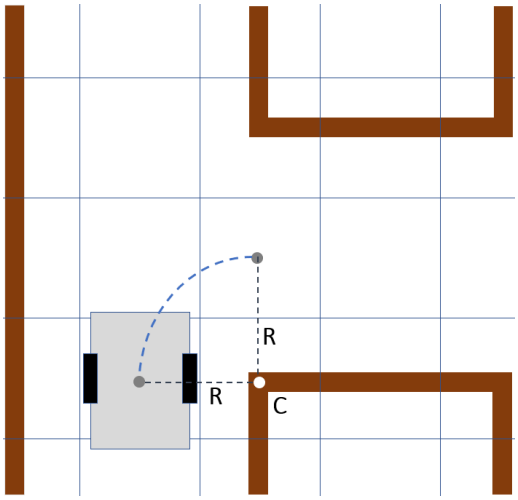
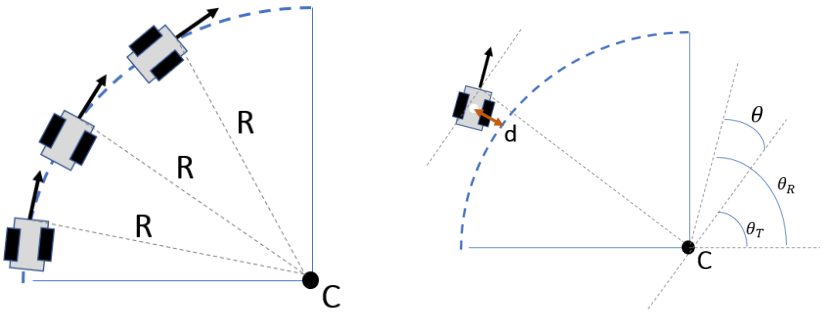


Figura 4.5: Seguidor de trajetória circular: R representa o raio da circunferência e C o centro de rotação.



(a) (b)

Figura 4.6: Seguidor de trajetória retilínea: (a) Orientação do robô segundo a tangente à circunferência. (b) Posição e orientação do robô fora dos limites de referência: d representa a diferença à posição de referência e θ a diferença ao ângulo de referência.

Para conduzir o robô aos valores de referência pretendidos é mantida a velocidade linear constante e é ajustada a velocidade angular com recurso à equação 4.2. É de realçar a soma da com-

ponente w_0, em torno da qual se efetuarão os ajustes de velocidade angular. K_θ e K_d representam fatores de ganho de ângulo e distância, respetivamente.

$$w = -K_\theta * \theta - K_d * d + w_0 \tag{4.2}$$

$$w_0 = v/R \tag{4.3}$$

4.3 Conclusão

O presente capítulo focou-se na descrição do controlo de trajetória em malha fechada, necessário para a aplicação do algoritmo TEA* a um ambiente de simulação realista. Através da posição e orientação fornecida, tornou-se possível o replaneamento online das trajetórias ótimas para cada robô, conferindo ao sistema multi robótico uma capacidade de coordenação significativamente mais eficaz. Com base nos caminhos planeados, são determinadas as velocidades linear e angular a atribuir aos robôs. Fecha-se, desta forma, a malha de controlo e situações de colisão e bloqueios entre robôs passam a ser resolvidas em tempo real.

Adicionalmente, foram ainda implementados dois controladores de seguimento de trajetória, circular e retilínea, para conferir um movimento preciso dentro de valores de posição e orientação de referência.

Capítulo 5

Implementação do Sistema

Pretende-se com o presente capítulo apresentar os testes realizados ao algoritmo TEA* e, consequentemente, os resultados que daí advieram. Os métodos implementados foram validados com recurso a plataformas virtuais, as quais são descritas na secção 5.1. É de notar que as características consideradas para essas plataformas foram definidas tendo como vista uma futura aplicação do TEA* em problemas de coordenação em ambiente real. Desse modo, são apresentadas, na secção 5.2, as dimensões, componentes eletrónicos e respetivas ligações, utilizados na construção dos robôs que serão considerados para testes futuros numa plataforma real, disponibilizada em laboratório. Por final, na secção 5.3, serão apresentadas diversas configurações iniciais para as posições dos robôs e correspondentes testes de coordenação, analisando-se, posteriormente, o tempo de execução final do conjunto bem como os trajetos planeados pelo algoritmo. Para os testes realizados, em primeiro lugar, será utilizada apenas a versão do TEA*, com inclusão da abordagem proposta na secção 3.2.2 e, posteriormente, será feita uma análise comparativa da mesma com a versão inicial do TEA*, apresentada previamente e implementada tendo como base a formulação descrita em [14, 36]. Ambas as análises serão apresentadas, respetivamente, nas secções 5.3.1 e 5.3.2, tendo sido utilizado para a primeira um visualizador 3D, designado de GLScene, e para a segundo um software de simulação 3D, denominado por SimTwo.

5.1 Plataformas para realização de testes

De modo a poderem ser realizados diversos testes ao algoritmo TEA* implementado, foram criadas três configurações (figura 5.1), a inserir numa plataforma quadrada com paredes equidistantes entre si. Ambas as plataformas seguem a decomposição por célula fixa, referida na secção 3.1, para a qual são utilizadas 361 células quadradas.

A duas primeiras configurações utilizadas apresentam a forma de um labirinto, definido de forma aleatória, contudo, atendendo à existência de diversos entroncamentos que tornem possível o deslocamento entre um ponto de partida e um ponto de chegada, a partir de diferentes trajetos. Além disso, inclui-se na configuração a existência de corredores sem saída que permitam testar situações de bloqueio entre dois robôs.

A segunda configuração pretende simular, de forma mais aproximada, um ambiente industrial. A presença de diversos corredores estreitos, um corredor partilhado e inúmeras estações de partida e chegada são algumas das características que lhe conferem o caráter industrial pretendido. Adicionalmente, é incluída uma estação intermédia, onde podem ser simuladas várias tarefas a realizar por um robô, a meio do seu trajeto.

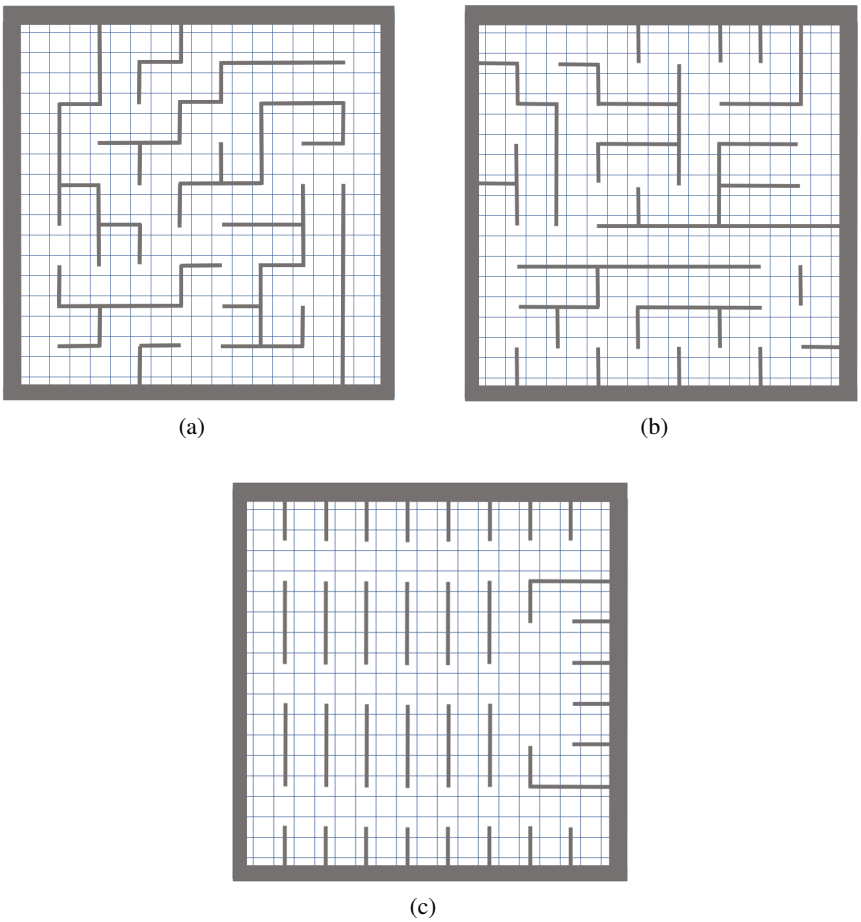


Figura 5.1: (a) Configuração Labirinto 1. (b) Configuração Labirinto 2. (c) Configuração Industrial.

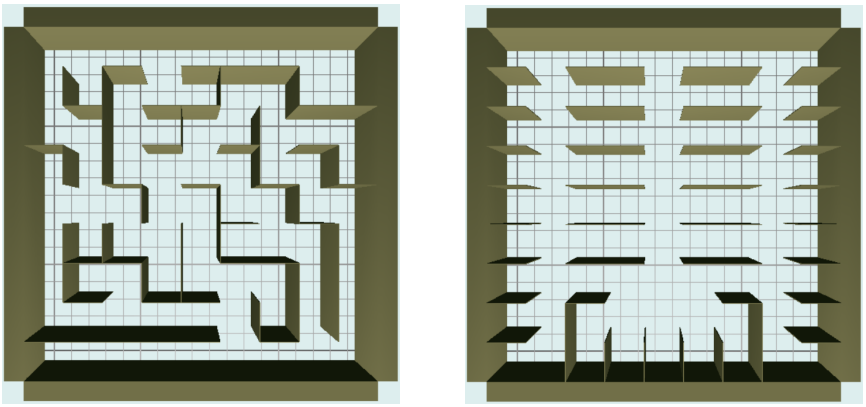
As três configurações foram adaptadas para cenários virtuais desenvolvidos com recurso a dois softwares distintos.

A primeira plataforma virtual de teste foi criada com base numa biblioteca de visualização 3D, designada por GLScene. Esta ferramenta foi incluída no ambiente de desenvolvimento integrado de software Lazarus v1.6.4, o qual já tinha sido utilizado para a implementação do TEA*. O principal objetivo da sua utilização passa por validar o planeamento fornecido pelo algoritmo, sem ser necessário entrar em conta com condições dinâmicas de movimento e controlo de trajetória. Desta forma, apenas são visualizadas as posições futuras dos robôs ao longo das diversas camadas temporais.

O design em GLScene funciona na base de uma construção a três dimensões, cujos blocos têm de ser localizados no espaço, tendo em conta um referencial XYZ. Deste modo, para a criação do mapa pretendido, foi necessário atribuir coordenadas e propriedades aos blocos representativos de obstáculos. De forma a tornar esse processo automático, recorreu-se à utilização de ficheiros XML para representar cada configuração desejada, tendo sempre em consideração que as coordenadas a atribuir aos obstáculos estão dependentes da divisão por células que foi efetuada para o espaço de atuação dos robôs.

Com o objetivo de otimizar a implementação dos ficheiros XML, optou-se pela representação de cada conjunto de células que se encontrem num mesmo segmento de reta, ao invés de representar o estado de cada célula do mapa isoladamente. Deste modo, cada segmento será descrito no ficheiro XML pelas suas coordenadas XX e YY, iniciais e finais.

Os mapas obtidos em GLScene para uma das configurações Labirinto e para a configuração Industrial, após leitura dos ficheiros XML no IDE Lazarus, estão representados nas figuras 5.2a e 5.2b, respetivamente.



(a) (b)

Figura 5.2: GLScene: (a) Configuração Labirinto 1. (b) Configuração Industrial.

A segunda plataforma virtual de teste foi desenvolvida com recurso ao SimTwo. Caracterizado por ser um sistema de simulação 3D, o mesmo foi utilizado para testar e validar o controlo em malha fechada do movimento dos robôs, o qual se pretende aplicar numa situação de teste em plataforma real. Ao conferir realismo na dinâmica dos robôs, através da implementação das suas propriedades físicas, o simulador criado permite aproximar o comportamento real que os mesmos apresentarão. Adicionalmente, através do realismo conseguido em simulação e, consequente diferenciação temporal entre os movimentos realizados entre células (seguimento de linha, seguimento de curva ou rotação), foi ainda possível fazer uma análise comparativa entre os tempos de execução de um conjunto de tarefas, quando utilizada a abordagem proposta na secção 3.2.2 ou a versão inicial do TEA*, apresentada previamente e implementada tendo como base a formulação descrita em [14, 36].

A construção da plataforma seguiu uma implementação semelhante à que foi desenvolvida no GLScene. Com recurso ao mesmo ficheiro XML, descrito anteriormente, foi possível conhecer

as coordenadas dos obstáculos presentes no mapa e, a partir delas, gerar um outro ficheiro XML, a utilizar no SimTwo, o qual contém as propriedades físicas das paredes bem como a localização das mesmas segundo um referencial XYZ. Tendo em vista o objetivo já referido para a utilização do simulador, somente os mapas para as configurações Labirinto foram implementados. Estes encontram-se ilustrados nas figuras 5.3, 5.4 e 5.5.

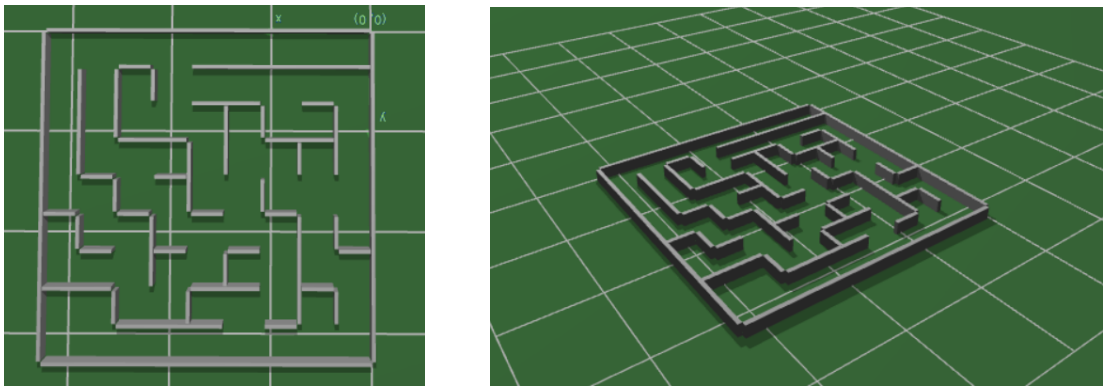


Figura 5.3: SimTwo: Configuração Labirinto 1.

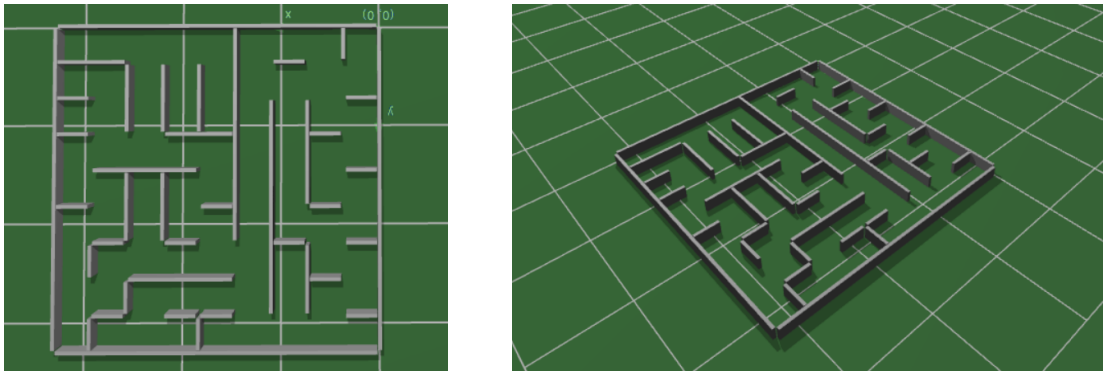


Figura 5.4: SimTwo: Configuração Labirinto 2.

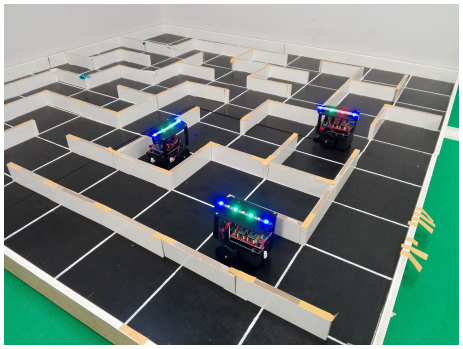


Figura 5.5: Plataforma de teste real disponibilizada em laboratório, juntamente com os robôs desenvolvidos.

5.2 Descrição dos Robôs Diferenciais a utilizar

Com o intuito de validar, futuramente, o algoritmo TEA* em ambiente real, foram desenvolvidos três robôs diferenciais, equivalentes no que respeita a hardware e software, e cuja área por eles ocupada pode ser aproximada por um retângulo com largura de 11 centímetros e comprimento de 11,5 centímetros (figuras 5.6 e 5.7). Deste modo, permite a movimentação em qualquer zona da plataforma, sem colidir com obstáculos estáticos.

É de salientar que as suas dimensões excedem o tamanho das laterais das células (8,52 centímetros) consideradas na decomposição do espaço de configuração da plataforma. Por esse motivo, é fundamental entrar em conta com o conceito de vizinhança obstáculo no planeamento através do algoritmo TEA*. Por outras palavras, para garantir que não existem colisões, um robô não poderá considerar a possibilidade de planear para si posições que pertençam à vizinhança de outros robôs de prioridade superior.

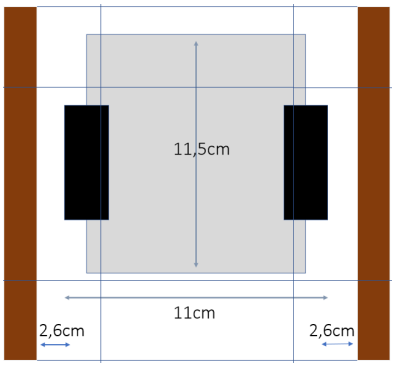


Figura 5.6: Dimensões do robô diferencial utilizado bem como do espaço livre entre o mesmo e as paredes da plataforma.

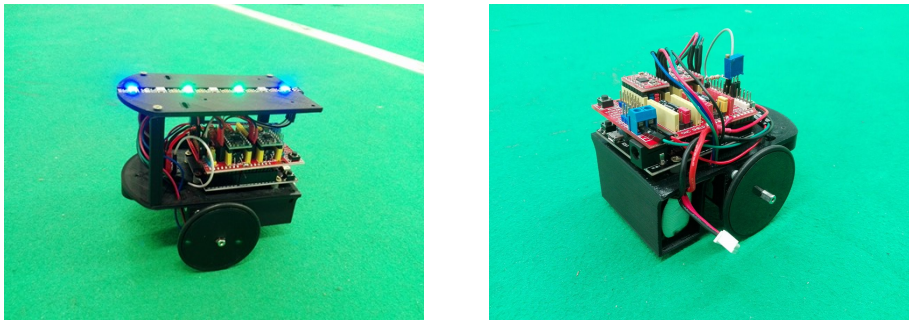


Figura 5.7: Robô Diferencial utilizado.

O hardware dos robôs foi desenvolvido tendo como base o esquema conceptual presente na figura 5.8.

O movimento de cada robô presente na plataforma de teste seguirá o planeamento sugerido pelo algoritmo TEA*. Para que isso se torne possível em contexto real, foi utilizado um microcontrolador com módulo de comunicações wireless integrado, através do qual se pretende controlar

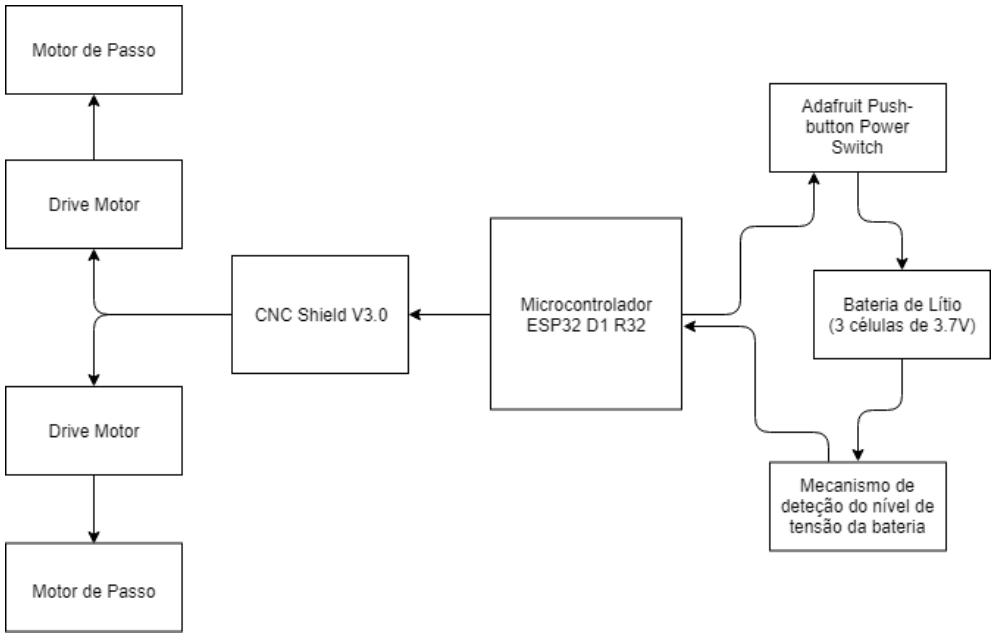


Figura 5.8: Esquema Conceptual do hardware dos robôs.

a velocidade dos motores. Deste modo, o planeamento é executado no PC com uma determinada cadência e, com base no caminho traçado, é determinada a velocidade a colocar nos motores, necessária para conduzir o robô entre dois pontos. A informação das velocidades é enviada para o microcontrolador a partir de mensagens encapsuladas segundo o protocolo UDP. É de salientar ainda que, uma vez que não se pretende executar movimentos com velocidades elevadas, optou-se pelo uso de motores de passo, com o objetivo de conferir maior precisão no movimento.

A *drive* utilizada é integrada na *shield* e funciona como um amplificador da corrente que é fornecida ao motor. Os sinais de baixa corrente enviados pelo microcontrolador são transformados em sinais de corrente superior do lado do motor, tornando possível o seu movimento. Além disso, o seu circuito eletrónico apresenta integrado duas pontes em H, uma para cada motor, de modo a que tensões positivas e negativas possam lhe ser fornecidas, possibilitando, assim, o movimento em ambos os sentidos de rotação.

Adicionalmente, com o objetivo de proteger a bateria de Lítio de descargas completas, foi introduzido um mecanismo de deteção do nível de tensão da mesma. Quando esta passa a baixo de um limite pré-definido, e regulado através de um potenciómetro, o microcontrolador atua sobre um pino de entrada do circuito integrado do botão de pressão, cortando a alimentação.

É ainda de realçar a inexistência de sensores. Tal se deve à utilização de um sistema de visão que, através de uma câmara, localizada sobre a plataforma, é capaz de detetar os LEDs colocados sobre cada robô e fornecer, assim, numa determinada cadência, as suas posições e orientações, sendo estes os parâmetros necessários à execução do algoritmo TEA*.

5.3 Testes e Resultados

Nesta secção serão apresentados os testes realizados ao algoritmo TEA* e consequentes resultados obtidos, tendo como ambiente de visualização e simulação, o GLScene e o SimTwo, respetivamente. Para cada teste descrito pretende-se fundamentalmente obter a coordenação eficaz de um conjunto de múltiplos robôs, evitando colisões e situações de deadlock que bloqueiem por completo o sistema. Além da eficácia pretendida, também serão testadas algumas otimizações à eficiência do algoritmo.

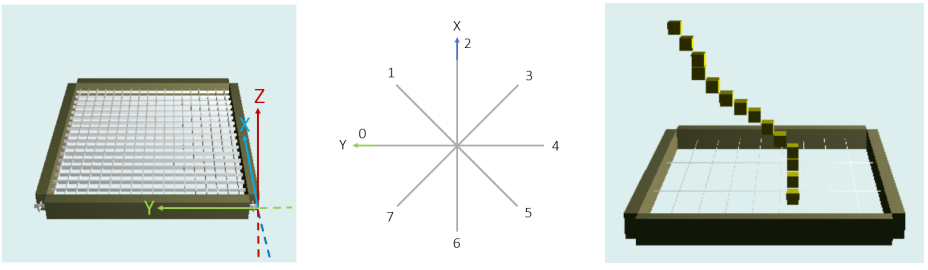
Para a realização dos testes foi utilizado um PC com um processador Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz e 8 Gb de RAM.

5.3.1 GLScene

Com recurso às capacidades de visualização a três dimensões do GLScene, foram realizados testes diversificados ao algoritmo de planeamento, variando a quantidade de robôs utilizados, as suas posições iniciais e finais, bem como as suas direções iniciais consideradas.

Cada teste realizado pretende simular diferentes situações, descritas detalhadamente no capítulo 3 e possíveis de ocorrer quando considerado um planeamento por prioridades e movimento simultâneo de todos os robôs. Em primeiro lugar, serão utilizadas apenas duas unidades para demonstrar cada caso de estudo e, por final, será elevado o nível de complexidade do sistema através da utilização de um número crescente de robôs sobre a plataforma.

Com o intuito de tornar possível a visualização dos caminhos planeados no tempo serão utilizadas representações a três dimensões, como a da figura 5.9c, nas quais cada nível (cubo) representará uma camada temporal. O fator tempo terá, como origem e camada temporal 0, o solo da plataforma de teste e, crescerá, portanto, ao longo do eixo ZZ. É de salientar que, nos casos em que se observe a permanência do robô numa célula, entre camadas temporais consecutivas, o mesmo poderá estar perante uma situação de espera para evitar colisão ou poderá estar a executar o movimento de rotação, de modo a alcançar a direção pretendida. Deste modo, será ainda apresentada para cada robô a sua direção inicial no plano XY, a qual seguirá a numeração utilizada no esquema da figura 5.9b.

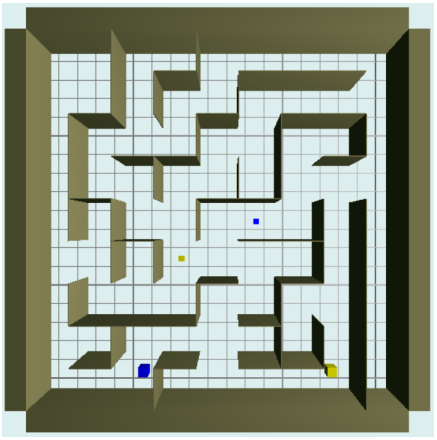


(a) (b) (c)

Figura 5.9: (a) Referencial utilizado no GLScene. (b) Direções utilizadas no plano XY. (c) Caminho planeado e representado no GLScene.

5.3.1.1 Cruzamento em camadas temporais diferentes

O presente teste, cuja configuração inicial é dada pela figura 5.10, permite validar o desempenho base do algoritmo TEA*, através do planeamento de trajetórias para dois robôs que não entrarão em rota de colisão, em nenhum instante temporal. Embora seja considerada a passagem de ambos em células de iguais coordenadas, a mesma acontece para camadas temporais diferentes, não existindo, portanto, qualquer interferência entre os caminhos planeados.



Posição Inicial do Robô Amarelo
Posição Inicial do Robô Azul
Posição Final do Robô Amarelo
Posição Final do Robô Azul

	Robôs	
	Amarelo	Azul
Direção Inicial	2	6
Prioridade Inicial	1	2

*Prioridade de valor igual a 1 é considerada a maior prioridade na hierarquia

Figura 5.10: Configuração inicial do teste 1: Cruzamento em camadas temporais diferentes.

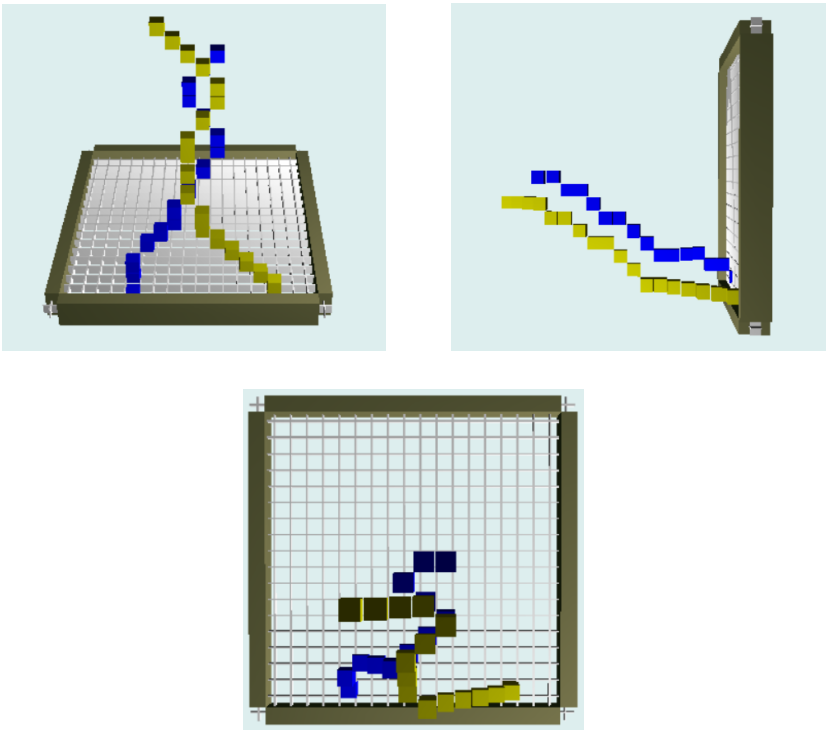


Figura 5.11: Caminhos 3D planeados com base no algoritmo TEA* para o teste 1, representados segundo diferentes ângulos.

Como pode ser observado na figura 5.11, segundo dois ângulos diferentes, os caminhos amarelo e azul nunca se intersetam ao longo das camadas temporais. Além disso, é possível ainda verificar, na vista de topo presente na mesma figura, que o caminho entre o ponto inicial e final foi atingido para ambos os robôs.

Tabela 5.1: Tempos de execução dos trajetos planeados pelo TEA* para o teste 1.

	Robôs	
	Amarelo	Azul
Tempo de Execução Final (s)	18.33	15.30

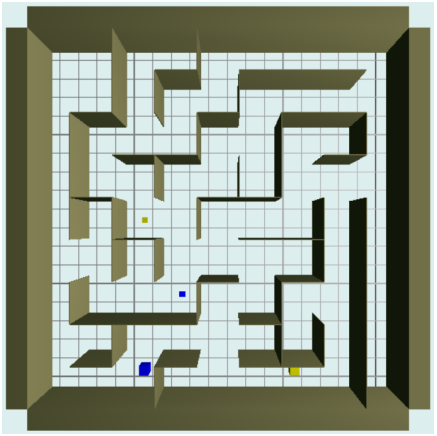
5.3.1.2 Cruzamento com previsível interseção na mesma camada temporal

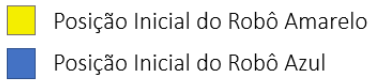
Quando se prevê uma possível passagem de dois robôs pela mesma célula, na mesma camada temporal, fica-se perante um caso de colisão futura. Para evitar essa situação, é necessário impedir que não seja planeada a mesma posição para ambos, num mesmo instante. Deste modo, duas soluções podem ser consideradas. Pode-se optar ou por planear, para o robô de menor prioridade, um caminho alternativo diferente do original, ou escolher manter o primeiro caminho planeado, permitindo ao robô esperar próximo do ponto de colisão, durante as camadas temporais que forem necessárias para que não exista conflito. Mediante as duas opções referidas, o algoritmo TEA* planeia o caminho que melhor otimizar o tempo de execução, o qual corresponderá à utilização de um número inferior de camadas temporais.

Ambas as soluções foram testadas e são apresentadas de seguida na presente secção.

Planeamento de caminho alternativo

Para o primeiro caso, é apresentada uma configuração (figura 5.12), que levará o robô azul a planear o caminho alternativo.



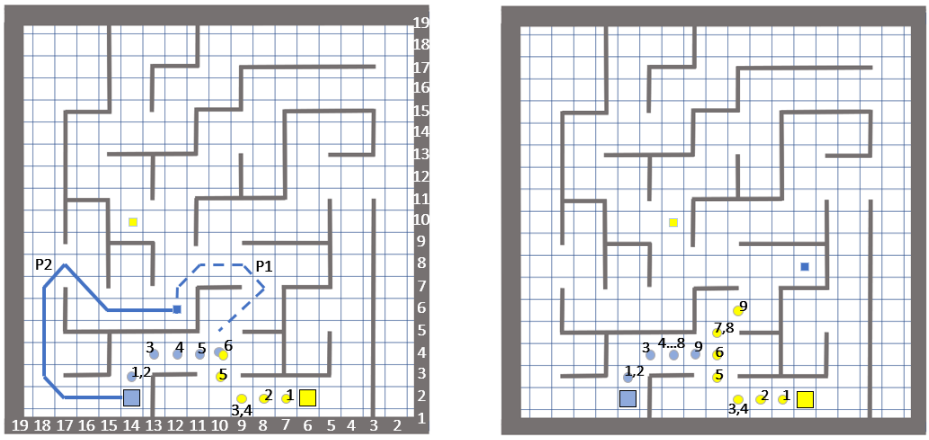


	Robôs	
	Amarelo	Azul
Direção Inicial	0	2
Prioridade Inicial	1	2

*Prioridade de valor igual a 1 é considerada a maior prioridade na hierarquia

Figura 5.12: Configuração inicial do teste 2: Planeamento de caminho alternativo.

Observando a figura 5.13a, é possível identificar como previsível ponto de colisão a célula (4,10), uma vez que o seu centro está a uma distância temporal de seis camadas, quer do robô amarelo quer do robô azul, assumindo para os mesmos o caminho que otimiza o tempo de chegada à célula objetivo. Tendo em conta a menor prioridade do robô azul, apenas os seus caminhos serão analisados.



(a) (b)

Figura 5.13: Teste 2: (a) Demonstração da possível colisão que poderia acontecer na célula (4,10). Representação do caminho ótimo P1 e do caminho alternativo P2. (b) Demonstração da situação de espera junto do ponto de colisão, caso P1 fosse a opção escolhida. Os números em cada célula simbolizam cada camada temporal.

Se o robô azul prosseguir pelo caminho P2, nenhum obstáculo dinâmico será encontrado e o deslocamento temporal será igual a 18 camadas. Por sua vez, se for tomado o caminho P1, a espera junto do ponto de colisão será de 4 camadas (figura 5.13b), o que equivaleria a um deslocamento temporal total de 19 camadas, superior ao do caminho P2. Deste modo, será este último, o trajeto que otimizará o tempo de execução e será, portanto, o escolhido. É importante notar que, por motivos de segurança, nenhum robô poderá entrar na vizinhança mais próxima de outro, justificando assim as 3 camadas temporais de espera.

Tal como previsto, é possível confirmar pela representação 3D presente na figura 5.14, que o caminho P2 foi o escolhido pelo robô azul para otimizar o tempo de execução.

Tabela 5.2: Tempos de execução dos trajetos planeados pelo TEA* para o teste 2.

	Robôs	
	Amarelo	Azul
Tempo de Execução Final (s)	18.26	18.26

O facto de terem sido registados, na tabela 5.2, tempos de execução iguais sugere que ambos os robôs percorreram o mesmo número de camadas temporais, alcançando o seu objetivo simultaneamente.

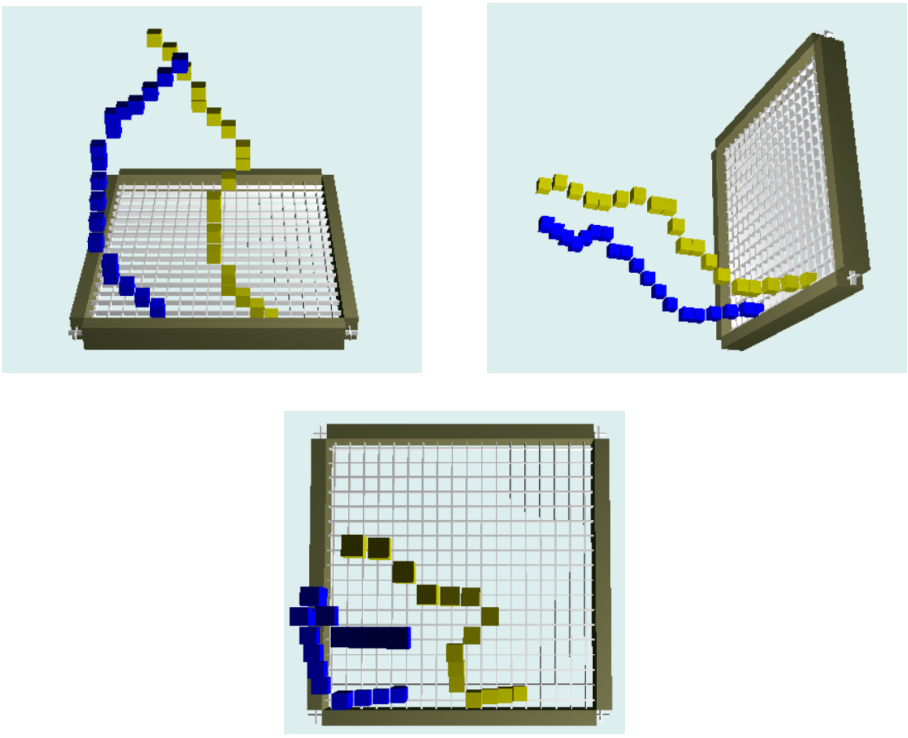
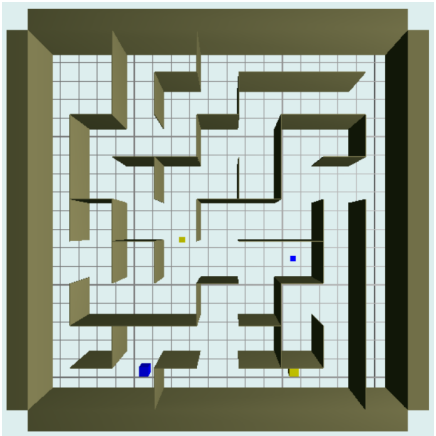


Figura 5.14: Caminhos 3D planeados com base no algoritmo TEA* para o teste 2, representados segundo diferentes ângulos.

Planeamento de caminho com situação de espera



Posição Inicial do Robô Amarelo
Posição Inicial do Robô Azul
Posição Final do Robô Amarelo
Posição Final do Robô Azul

	Robôs	
	Amarelo	Azul
Direção Inicial	0	2
Prioridade Inicial	1	2

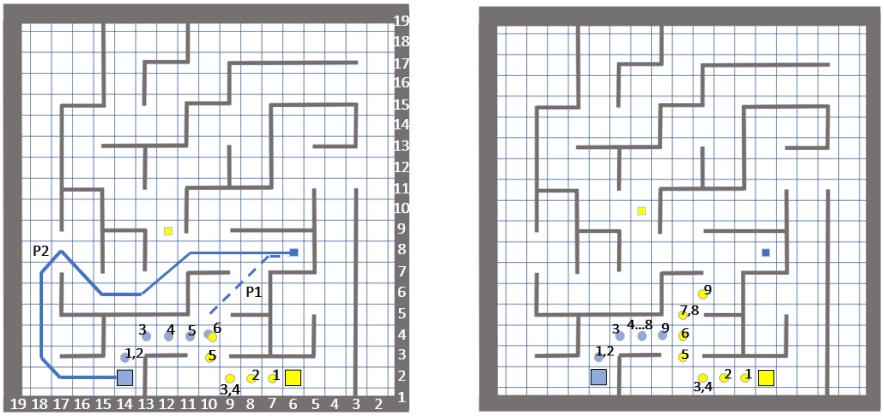
*Prioridade de valor igual a 1 é considerada a maior prioridade na hierarquia

Figura 5.15: Configuração inicial do teste 3: Planeamento de caminho com situação de espera.

Relativamente ao teste realizado para planeamento de caminho alternativo, foram alteradas as células objetivo dos robôs. Também nesta nova configuração (figura 5.15) se prevê uma possível colisão na célula (4,10), contudo, neste caso, o planeamento de um caminho alternativo para o robô azul atingir o seu ponto objetivo, deixa de ser vantajoso, em termos temporais, face a uma

possível espera próximo da zona de colisão.

Focando na figura 5.16, se o robô azul prosseguir pelo caminho P2, nenhum obstáculo dinâmico será encontrado e o deslocamento temporal será igual a 25 camadas. Por sua vez, se for tomado o caminho P1, a espera junto do ponto de colisão será de 4 camadas, o que equivaleria a um deslocamento temporal total de 14 camadas, inferior ao do caminho P2. Deste modo, o trajeto P1 otimizará o tempo de execução e será, portanto, o escolhido.



(a) (b)

Figura 5.16: Teste 3: (a) Demonstração da possível colisão que poderia acontecer na célula (4,10). Representação do caminho ótimo P1 e do caminho alternativo P2. (b) Demonstração da situação de espera junto do ponto de colisão, caso P1 fosse a opção escolhida. Os números em cada célula simbolizam cada camada temporal.

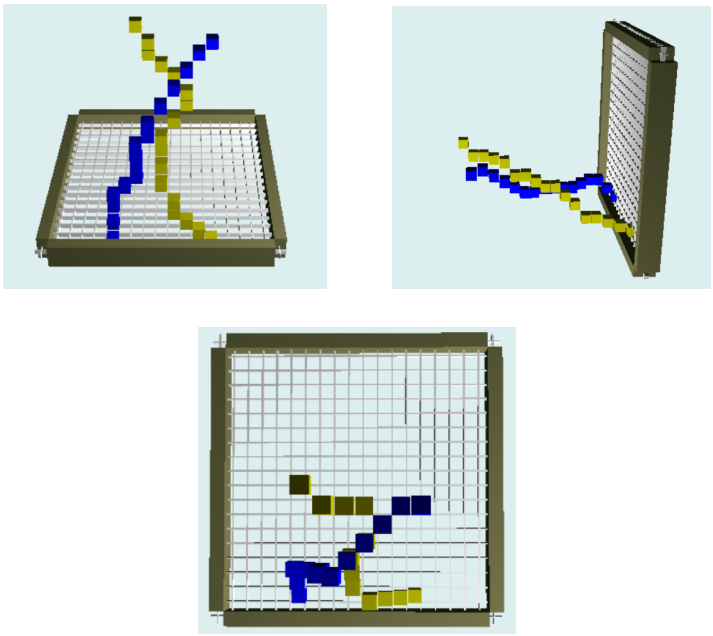


Figura 5.17: Caminhos 3D planeados com base no algoritmo TEA* para o teste 3, representados segundo diferentes ângulos.

Tal como previsto, é possível confirmar pela representação 3D presente na figura 5.17, que o caminho P1 foi o escolhido pelo robô azul para otimizar o tempo de execução.

Tabela 5.3: Tempos de execução dos trajetos planeados pelo TEA* para o teste 3.

	Robôs	
	Amarelo	Azul
Tempo de Execução Final (s)	16.22	15.19

5.3.1.3 Desvio de caminho para evitar situações de deadlock

No contexto de coordenação de um sistema robótico, uma situação de deadlock pode ser descrita como um bloqueio entre, pelo menos dois robôs, no qual ambos se impedem mutuamente de atingir as suas células objetivo. Assim sendo, para que o objetivo global do sistema não seja comprometido, algum dos robôs tem de ser capaz de ceder passagem, mesmo que implique um aumento do tempo de execução da sua tarefa.

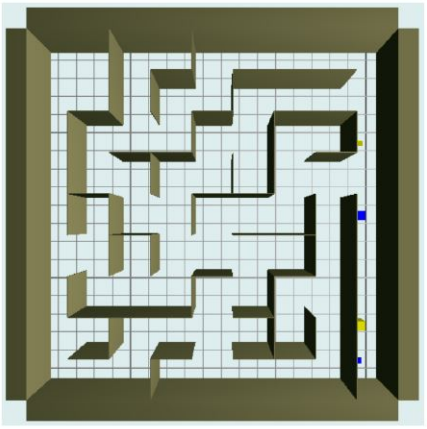


Figura 5.18: Configuração inicial do teste 4: Desvio de caminho para evitar situações de deadlock.

Observando a figura 5.18 e, tendo em consideração que o robô azul apresenta uma prioridade inferior, será ele mesmo a ter de ceder passagem. Tal acontece porque o robô amarelo coloca o caminho por si planeado como obstáculo, forçando o robô azul a evitar a coexistência nas mesmas células, em iguais camadas temporais. Apenas, posteriormente à passagem do amarelo, o robô azul poderá prosseguir em direção ao seu objetivo. O comportamento descrito pode ser comprovado através da representação 3D dos caminhos planeados na figura 5.19.

Tabela 5.4: Tempos de execução dos trajetos planeados pelo TEA* para o teste 4.

	Robôs	
	Amarelo	Azul
Tempo de Execução Final (s)	10.26	21.29

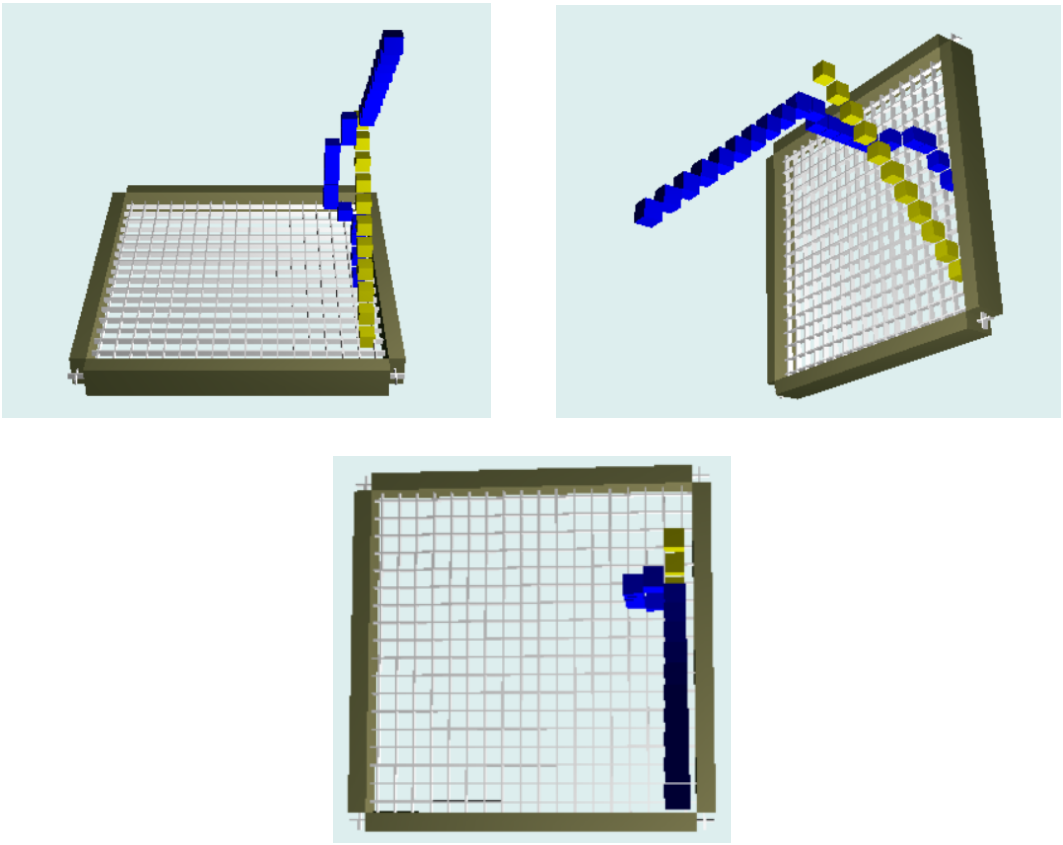
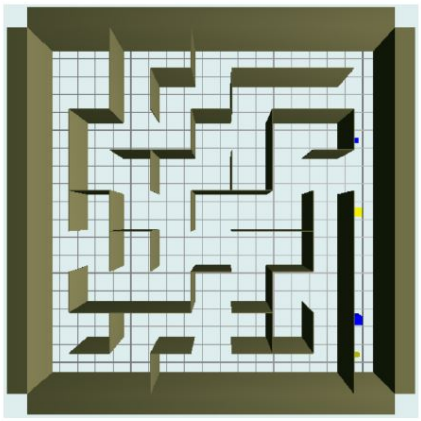


Figura 5.19: Caminhos 3D planeados com base no algoritmo TEA* para o teste 4, representados segundo diferentes ângulos.

5.3.1.4 Troca de prioridades

A utilização de um conjunto de prioridades fixas nem sempre permite a obtenção de caminhos livres de colisão para todos os robôs. Por esse motivo, é importante atribuir-lhes a capacidade de prejudicar o tempo de execução da sua tarefa em prol da eficácia de todo o sistema. Para tal ser possível, é considerada a hipótese de os robôs alternarem as suas prioridades, sempre que a célula objetivo é considerada como inatingível, mediante as condições iniciais. Deste modo, casos como os da figura 5.20, passam a ser também cobertos pelo algoritmo TEA*, podendo ser comprovado o resultado positivo da sua implementação através dos caminhos 3D ilustrados na figura 5.21. Uma vez que o robô azul não conseguiria, de nenhum modo, desviar-se de modo a que o amarelo pudesse atingir a célula objetivo, as prioridades de ambos invertem-se. A partir desse momento, o robô azul planeia o seu caminho e coloca-o como obstáculo para o robô amarelo, forçando-o a desviar-se.



Posição Inicial do Robô Amarelo
Posição Inicial do Robô Azul
Posição Final do Robô Amarelo
Posição Final do Robô Azul

	Robôs	
	Amarelo	Azul
Direção Inicial	0	6
Prioridade Inicial	1	2

*Prioridade de valor igual a 1 é considerada a maior prioridade na hierarquia

Figura 5.20: Configuração inicial do teste 5: Troca de prioridades.

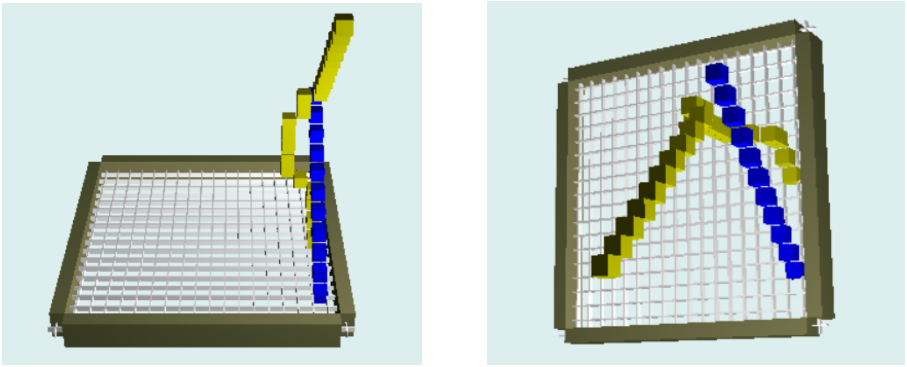


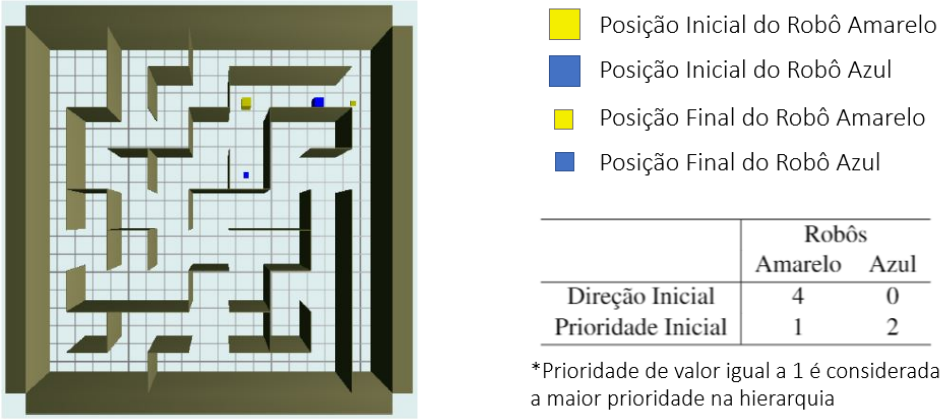
Figura 5.21: Caminhos 3D planeados com base no algoritmo TEA* para o teste 5, representados segundo diferentes ângulos.

Tabela 5.5: Tempos de execução dos trajetos planeados pelo TEA* para o teste 5.

	Robôs	
	Amarelo	Azul
Tempo de Execução Final (s)	21.66	10.55

A aplicação do sistema de troca de prioridades à situação descrita e ilustrada acima, visa melhorar a eficácia do planeamento, quando confrontado com casos de deadlock. Porém, podem existir situações distintas da anterior, para as quais a obtenção de um caminho viável até à célula objetivo não estará em causa. Ainda assim, os caminhos poderão ser bastante ineficientes, em termos de tempo de execução, quando comparados com os caminhos planeados após uma inversão de prioridades. Por esse motivo, foi implementado um mecanismo de deteção dessas situações e, posterior, otimização das mesmas. Será possível ainda verificar que nem sempre o mecanismo de deteção apresentará resultados positivos, direcionando, portanto, o planeamento para uma situação livre de otimizações.

Planeamento com otimização temporal possível



	Robôs	
	Amarelo	Azul
Direção Inicial	4	0
Prioridade Inicial	1	2

*Prioridade de valor igual a 1 é considerada a maior prioridade na hierarquia

Figura 5.22: Configuração inicial do teste 6: Planeamento com otimização temporal possível.

Apresentando o robô amarelo uma prioridade superior, é de esperar que, para a configuração representada na figura 5.22 o robô azul tenha de recuar de modo a permitir a sua passagem até à célula objetivo. Contudo, estando esta localizada no cruzamento, ou seja célula (16,8), o robô azul não terá mais possibilidade de por aí passar, sendo forçado a planear um caminho alternativo, tal como sugerido na figura 5.23. De modo a tornar o algoritmo mais eficiente, assim que o robô azul deteta uma possível colisão com o amarelo na célula (18,6), dá-se a inversão de prioridades. O robô amarelo passa a recuar para dar passagem ao azul, permitindo-lhe não prosseguir com o caminho mais longo. Ao invés do robô azul percorrer 28 camadas temporais, apenas percorre 8. Por sua vez, o robô amarelo fica com um trajeto um pouco mais longo, devido ao desvio. Contudo, contabiliza um total de 16 camadas temporais, bastante inferior às 28 do caminho alternativo do robô azul. Otimiza-se assim o tempo de execução final do sistema, podendo ser comprovado o resultado positivo da sua implementação através dos caminhos 3D ilustrados na figura 5.24.

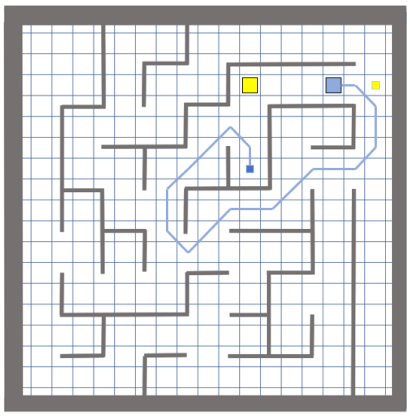


Figura 5.23: Demonstração do caminho alternativo planeado pelo robô azul, caso não fosse aplicada uma inversão de prioridades para otimizar o tempo de execução das tarefas.

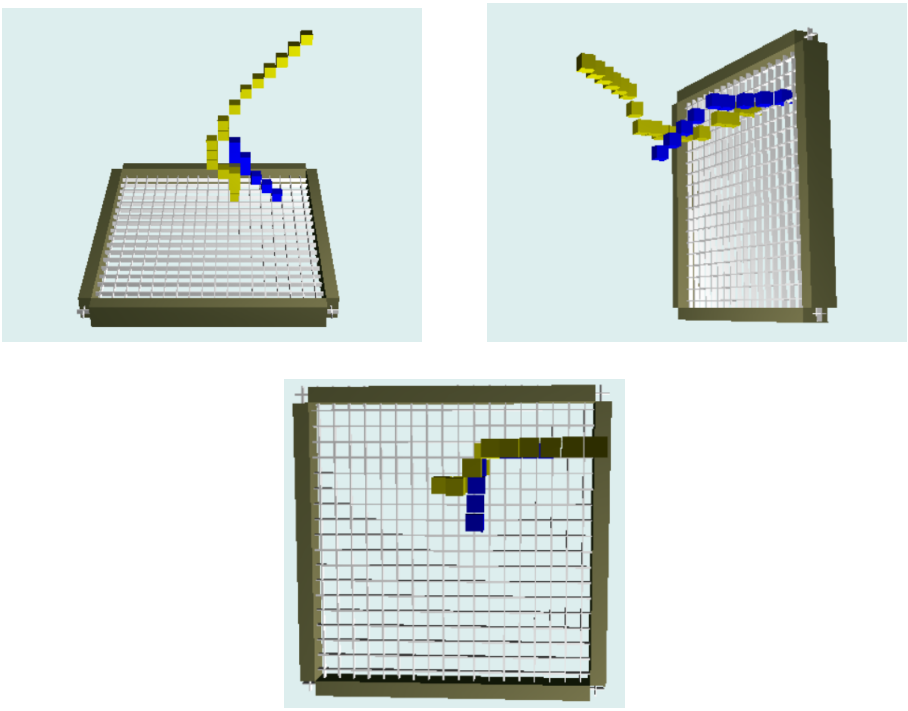
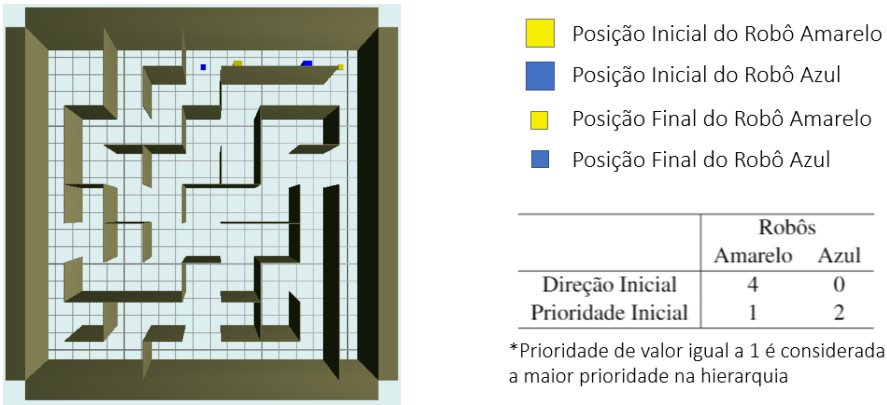


Figura 5.24: Caminhos 3D planeados com base no algoritmo TEA* para o teste 6, representados segundo diferentes ângulos.

Tabela 5.6: Tempos de execução dos trajetos planeados pelo TEA* para o teste 6.

	Robôs	
	Amarelo	Azul
Tempo de Execução Final (s)	16.33	8.18

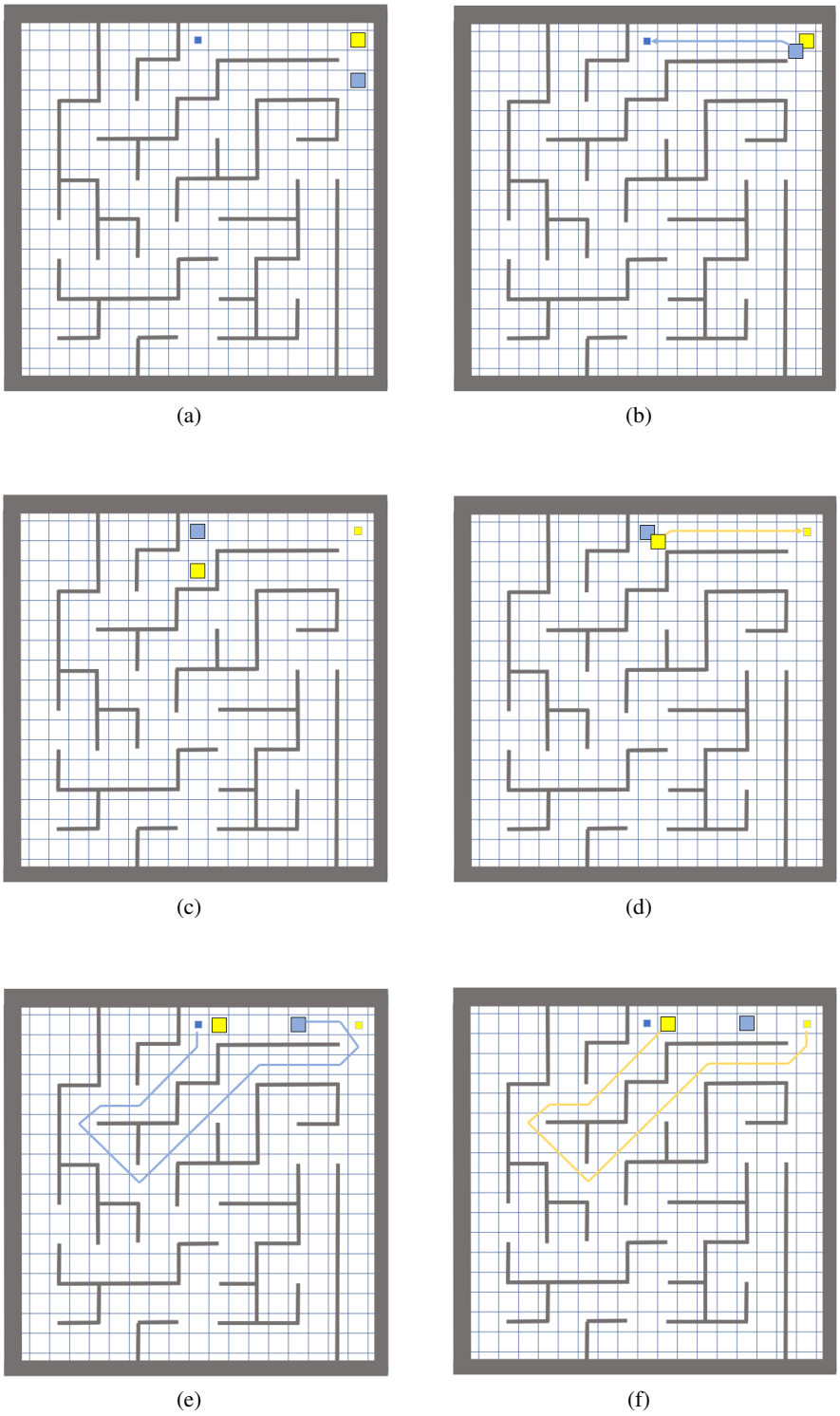
Planeamento sem otimização temporal possível



	Robôs	
	Amarelo	Azul
Direção Inicial	4	0
Prioridade Inicial	1	2

*Prioridade de valor igual a 1 é considerada a maior prioridade na hierarquia

Figura 5.25: Configuração inicial do teste 7: Planeamento sem otimização temporal possível.



(a) (b) (c) (d) (e) (f)

Figura 5.26: Teste 7: (a),(b): Robô amarelo detém a maior prioridade. Se robô azul tentasse otimizar o seu trajeto, coexistindo na vizinhança do amarelo, a colisão seria expectável. (c),(d): Situação idêntica à anterior com prioridades invertidas. (e),(f): Caminhos alternativos planeados tendo em conta as prioridades e o mecanismo implementado no algoritmo TEA*.

O presente teste, cuja configuração inicial é dada pela figura 5.25, permite confirmar a existência de casos para os quais a otimização por troca de prioridades falha. Por muito que ambos os robôs tentem inverter as suas prioridades, o primeiro a planear obstruirá a passagem ao chegar à sua célula objetivo e fará sempre com que o caminho ótimo, em termos temporais, do outro robô, não possa ser executado. Desse modo, os robôs excedem um limite máximo de trocas pré-definido e, nesse momento, o robô que apresentar maior prioridade planeia e coloca o seu trajeto como obstáculo, forçando o outro a definir um caminho alternativo mais longo. É de salientar uma vez mais que, por motivos de segurança, dois robôs não poderão coexistir na vizinhança um do outro. O comportamento descrito é ilustrado na figura 5.26.

No teste realizado, após ser excedido um limite máximo de 10 trocas de prioridades, o robô amarelo manteve o planeamento prioritário. Desta forma, o robô azul planeia o seu caminho alternativo, não ótimo no que respeita a tempo de execução, mas eficaz uma vez que o objetivo é alcançado sem prejudicar a realização da sua tarefa. O planeamento 3D pode ser observado na figura 5.27.

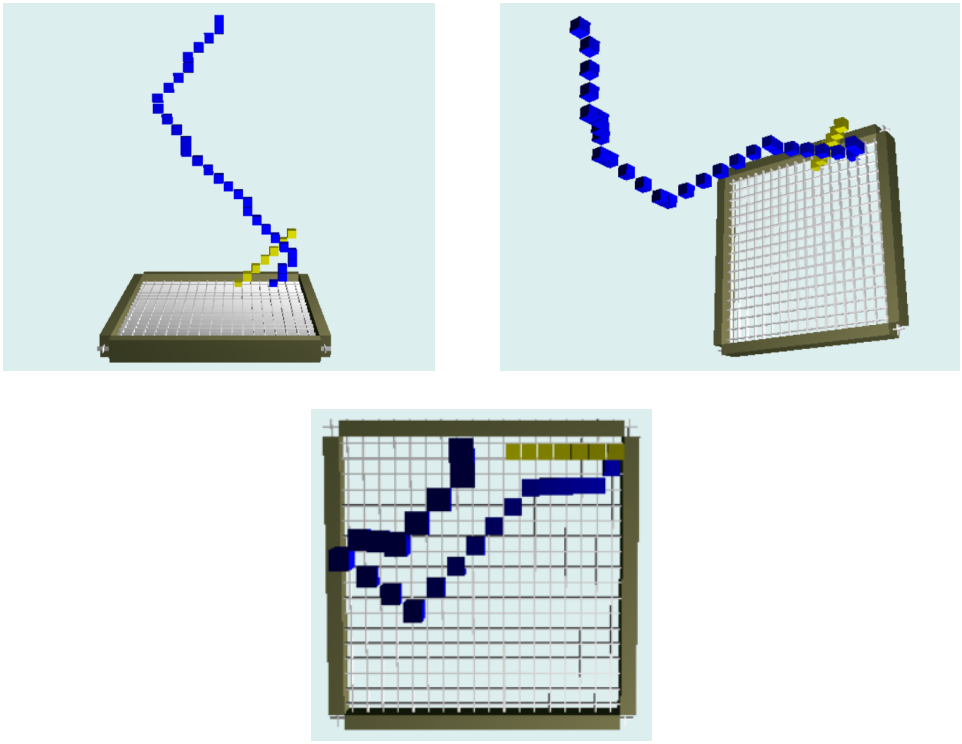


Figura 5.27: Caminhos 3D planeados com base no algoritmo TEA* para o teste 7, representados segundo diferentes ângulos.

Tabela 5.7: Tempos de execução dos trajetos planeados pelo TEA* para o teste 7.

	Robôs	
	Amarelo	Azul
Tempo de Execução Final (s)	6.27	32.31

5.3.1.5 Validação inversa das prioridades

Partindo do pressuposto que, na implementação descrita do TEA* no capítulo 3, qualquer robô deixa de verificar a existência de colisão para camadas temporais superiores à que atingiu a sua célula objetivo, nem sempre um robô com prioridade superior consegue evitar situações de possível colisão. Pode, portanto, acontecer o caso ilustrado nas figuras 5.28 e 5.29, no qual o robô azul, de menor prioridade, atingirá, em camadas inferiores, a sua célula pretendida, terminando a pesquisa temporal e bloqueando, consequentemente, a passagem do amarelo pelo mesmo local em instantes superiores. Sendo o robô amarelo detentor da maior prioridade, confiaria no trajeto por si planeado, ignorando a possibilidade de outro robô o intersetar em qualquer camada temporal. Deste modo, através da validação inversa das prioridades, esses casos são detetados, invertendo as prioridades do robô azul e amarelo. Por conseguinte, a célula objetivo do robô azul já será vista como obstáculo pelo amarelo em camadas temporais superiores, forçando-o a planear um caminho alternativo.

Posição Inicial do Robô Amarelo
Posição Inicial do Robô Azul
Posição Final do Robô Amarelo
Posição Final do Robô Azul

	Robôs	
	Amarelo	Azul
Direção Inicial	2	4
Prioridade Inicial	1	2

*Prioridade de valor igual a 1 é considerada a maior prioridade na hierarquia

Figura 5.28: Configuração inicial do teste 8: Validação inversa das prioridades.

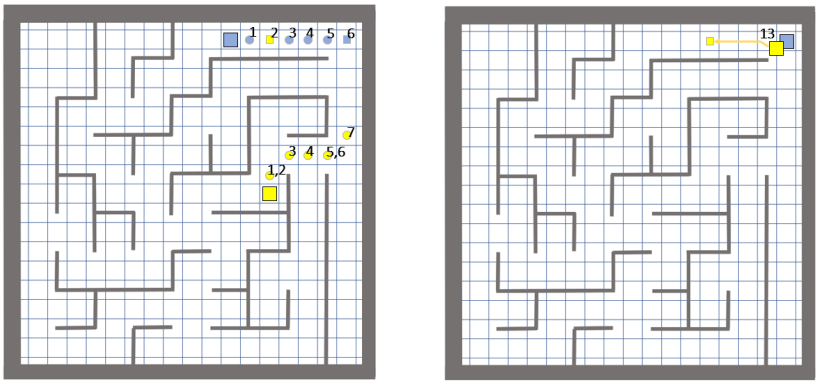


Figura 5.29: Planeamento que seria realizado se não fosse implementada a validação inversa de prioridades. A posição dos robôs em cada camada temporal é representada pelos círculos da cor respetiva.

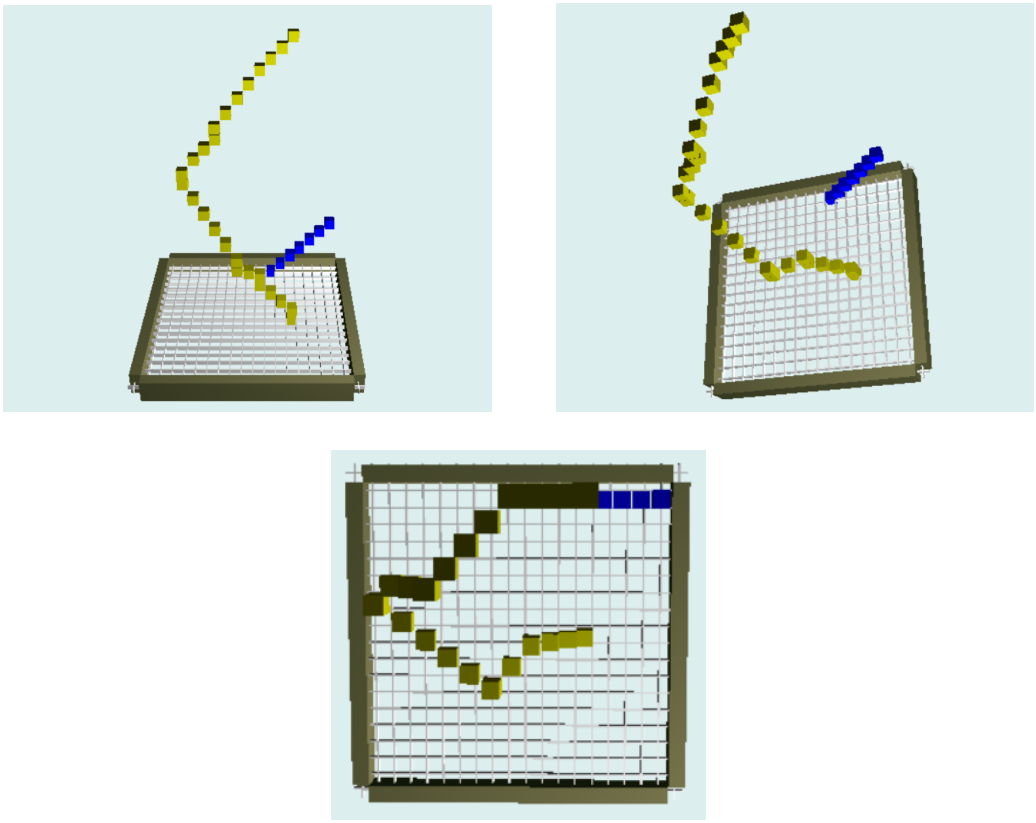


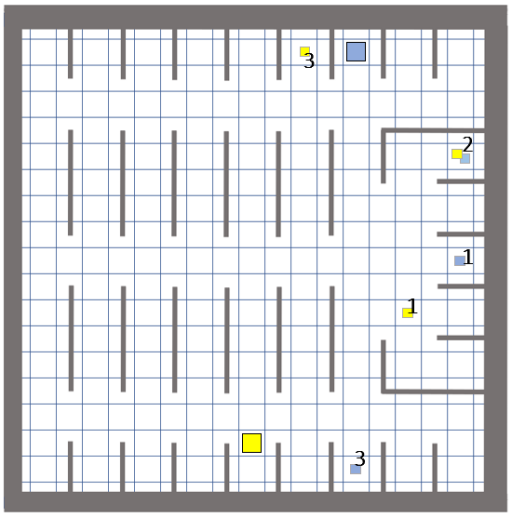
Figura 5.30: Caminhos 3D planeados com base no algoritmo TEA* para o teste 8, representados segundo diferentes ângulos.

Tabela 5.8: Tempos de execução dos trajetos planeados pelo TEA* para o teste 8.

	Robôs	
	Amarelo	Azul
Tempo de Execução Final (s)	6.34	30.17

5.3.1.6 Submissões

A implementação de submissões tem como objetivo permitir a cada robô realizar um conjunto de tarefas de forma consecutiva. No presente teste, cuja configuração inicial é dada pela figura 5.31, pretende-se validar a sua execução, considerando para efeitos de planeamento uma missão global. Deste modo, todas as submissões podem ser planeadas com base na prioridade inicial, permitindo ainda aos outros robôs conhecer o seu trajeto completo. Em contrapartida, se o planeamento de cada submissão fosse feito apenas após o robô atingir a célula objetivo da submissão anterior, robôs de menor prioridade poderiam realizar movimentos desnecessários, causados pela falta de conhecimento acerca dos próximos movimentos dos robôs de maior prioridade. O comportamento descrito é ilustrado detalhadamente nas figuras 5.32 e 5.33. É de realçar que a configuração da plataforma de teste foi alterada para a Industrial.



(a)

	Robôs	
	Amarelo	Azul
Direção Inicial	2	6
Prioridade Inicial	1	2

(b)

Figura 5.31: (a) Posição inicial de cada robô é dada pelo quadrado de tamanho superior. Submissões são dadas pelos quadrados de tamanho inferior, os quais têm associado a si o número de ordem da tarefa. (b) Parâmetros de inicialização do teste 9. A prioridade de valor igual a 1 é considerada a maior prioridade.

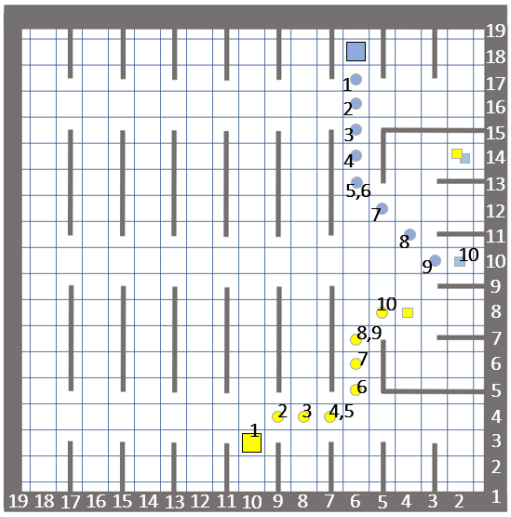
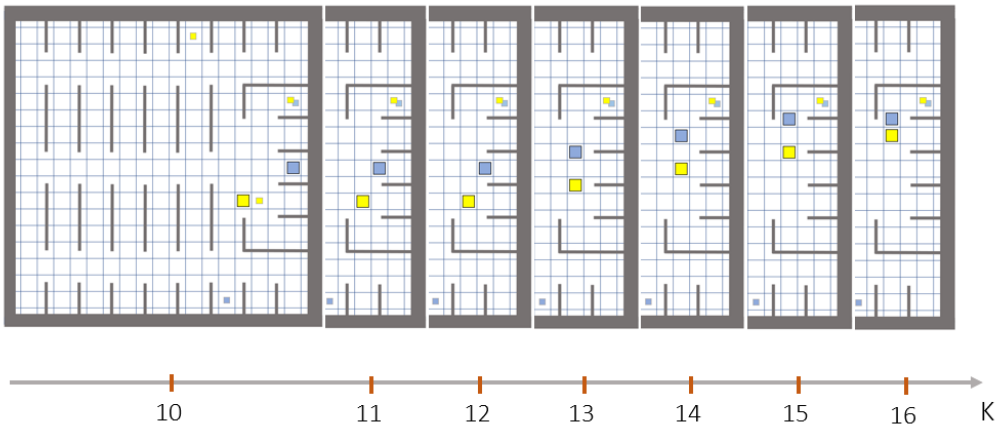
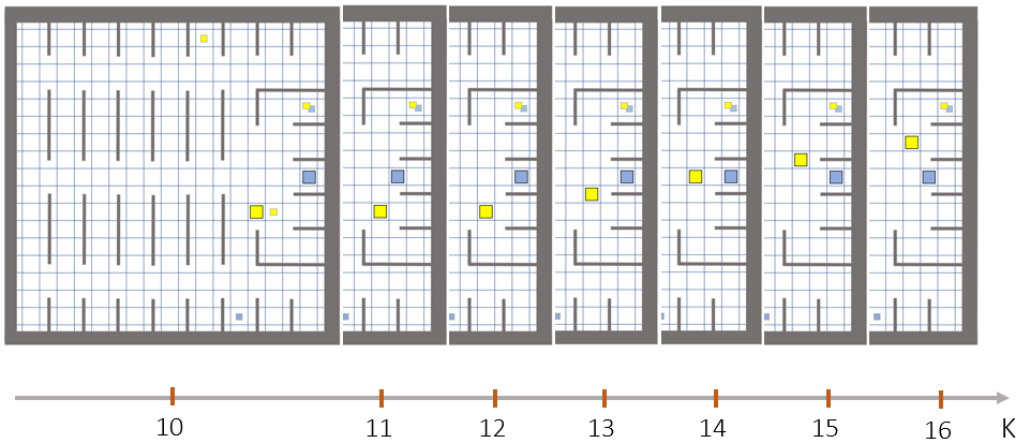


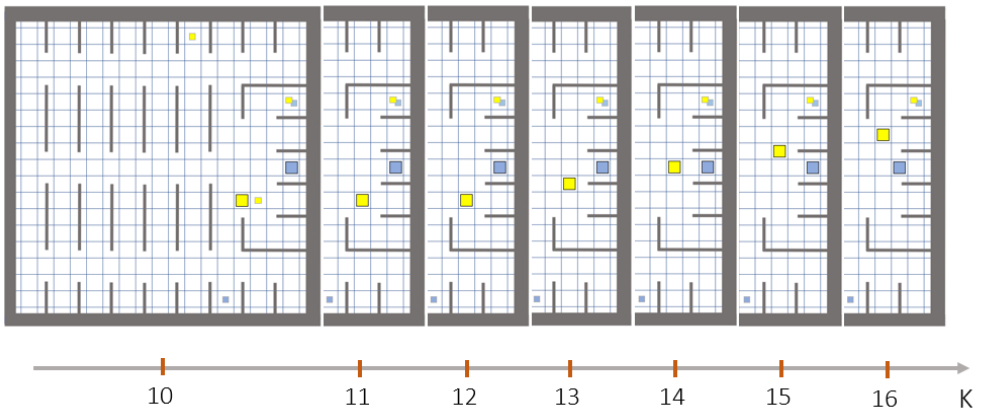
Figura 5.32: Planeamento por TEA* das primeiras 10 camadas temporais, independentemente de ser implementado uma missão global ou de ser considerado um planeamento individual por submissão.



(a) Robô amarelo detém a maior prioridade e região de colisão é detetada entre ambos os robôs. Robô azul entraria na vizinhança do robô amarelo, comprometendo a segurança na circulação.



(b) Resolução da região de colisão considerando um planeamento individual por submissão.



(c) Resolução da região de colisão considerando um planeamento global para o conjunto das submissões.

Figura 5.33: Planeamento das submissões com base no algoritmo TEA*. No eixo K é representado o número de cada camada temporal.

Quando considerado um planeamento individual por submissão, como demonstrado na figura 5.33b, o robô amarelo apenas planeia a sua segunda submissão após atingir a célula objetivo da primeira, na camada temporal 11. Previamente, o robô azul ainda não consegue detetar a região de colisão ilustrada na figura 5.33a. Deste modo, planeia movimentar-se em direção à célula objetivo da sua segunda submissão, deslocando-se de (10,2) para (10,3), em K=11. Contudo, na camada temporal seguinte, a segunda submissão do robô amarelo já é tida em conta no planeamento, levando o robô azul a detetar a futura colisão e, por conseguinte, a recuar. Posteriormente, espera a passagem do robô amarelo e só de seguida é que avança em direção ao seu objetivo.

Por sua vez, quando utilizado um planeamento com base numa missão global para o conjunto das submissões (tal como na presente dissertação e como representado na figura 5.33c), o robô azul passa a conseguir detetar a região de colisão em qualquer altura do planeamento. Dessa forma, evita movimentar-se em K=11, permanecendo imóvel na sua célula até que o robô amarelo passe.

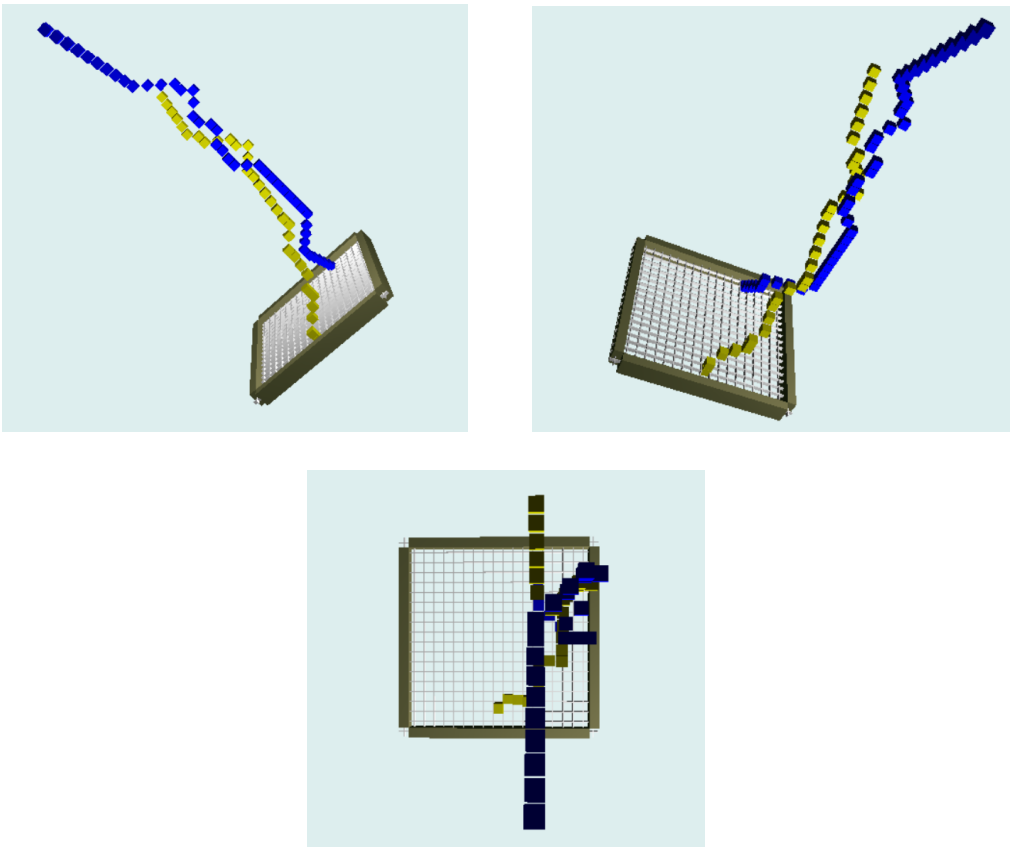


Figura 5.34: Caminhos planeados com base no algoritmo TEA* para o teste 9, representados segundo diferentes ângulos. Representação 3D das missões globais.

Tabela 5.9: Tempos de execução dos trajetos planeados pelo TEA* para o teste 9.

	Robôs	
	Amarelo	Azul
Tempo de Execução Final (s)	31.60	43.77

5.3.1.7 Crescimento do tráfego do sistema

Pretende-se com este teste observar a resposta do algoritmo com o crescente aumento de complexidade do sistema. Para isso, o sistema será povoado com 3 e 8 robôs, tanto para a plataforma em labirinto como também para a plataforma industrial (figuras 5.35, 5.37, 5.39 e 5.41), e os seus resultados serão apresentados com recurso às representações 3D provenientes do GLScene (figuras 5.36, 5.38, 5.40 e 5.42).

3 Robôs na Plataforma Labirinto

(a)

	Robôs		
	Amarelo	Azul	Laranja
Direção Inicial	2	4	6
Prioridade Inicial	1	2	3

(b)

Figura 5.35: (a) Posição inicial e final de cada robô é dada pelos cubos de tamanho superior e inferior, respetivamente. (b) Parâmetros de inicialização do teste 10. A prioridade de valor igual a 1 é considerada a maior prioridade.

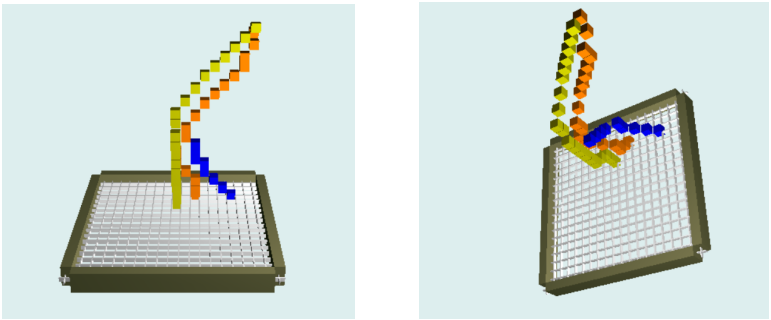
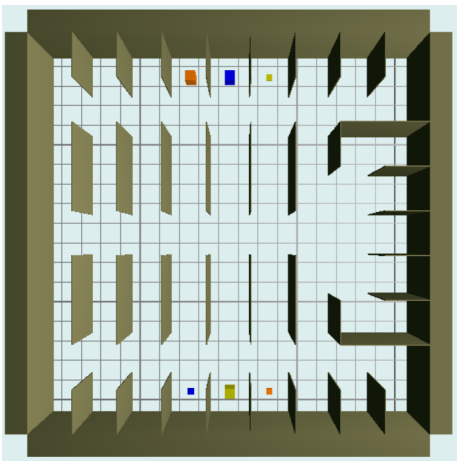


Figura 5.36: Caminhos 3D planeados com base no algoritmo TEA* para o teste 10, representados segundo ângulos diferentes.

Tabela 5.10: Tempos de execução dos trajetos planeados pelo TEA* para o teste 10.

	Robôs		
	Amarelo	Azul	Laranja
Tempo de Execução Final (s)	8.20	18.43	19.36

3 Robôs na Plataforma Industrial



(a)

	Robôs		
	Amarelo	Azul	Laranja
Direção Inicial	2	6	6
Prioridade Inicial	1	2	3

(b)

Figura 5.37: (a) Posição inicial e final de cada robô é dada pelos cubos de tamanho superior e inferior, respetivamente. (b) Parâmetros de inicialização do teste 11. A prioridade de valor igual a 1 é considerada a maior prioridade.

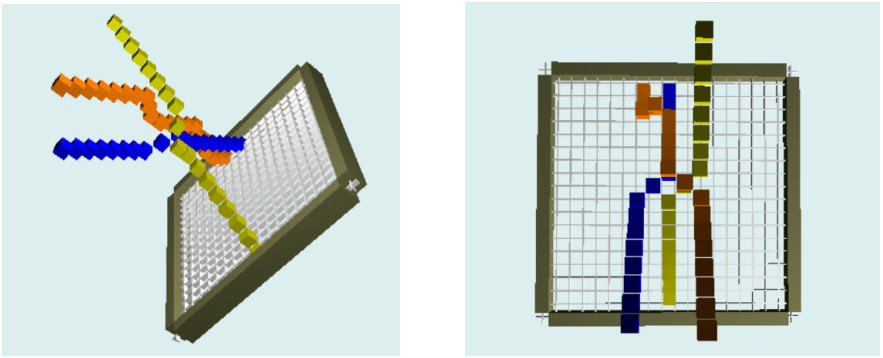


Figura 5.38: Caminhos 3D planeados com base no algoritmo TEA* para o teste 11, representados segundo ângulos diferentes.

Tabela 5.11: Tempos de execução dos trajetos planeados pelo TEA* para o teste 11.

	Robôs		
	Amarelo	Azul	Laranja
Tempo de Execução Final (s)	17.40	17.40	21.41

8 Robôs na Plataforma Labirinto

(a)

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	1	4	1	2
Prioridade Inicial	1	2	3	4
	Castanho	Rosa	Violeta	Ciano
Direção Inicial	2	5	7	0
Prioridade Inicial	5	6	7	8

(b)

Figura 5.39: (a) Posição inicial e final de cada robô é dada pelos cubos de tamanho superior e inferior, respetivamente. (b) Parâmetros de inicialização do teste 12. A prioridade de valor igual a 1 é considerada a maior prioridade.

Tabela 5.12: Tempos de execução dos trajetos planeados pelo TEA* para o teste 12.

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Tempo de Execução Final (s)	55.45	39.79	13.64	9.54
	Castanho	Rosa	Violeta	Ciano
Tempo de Execução Final (s)	58.64	31.14	46.32	36.04

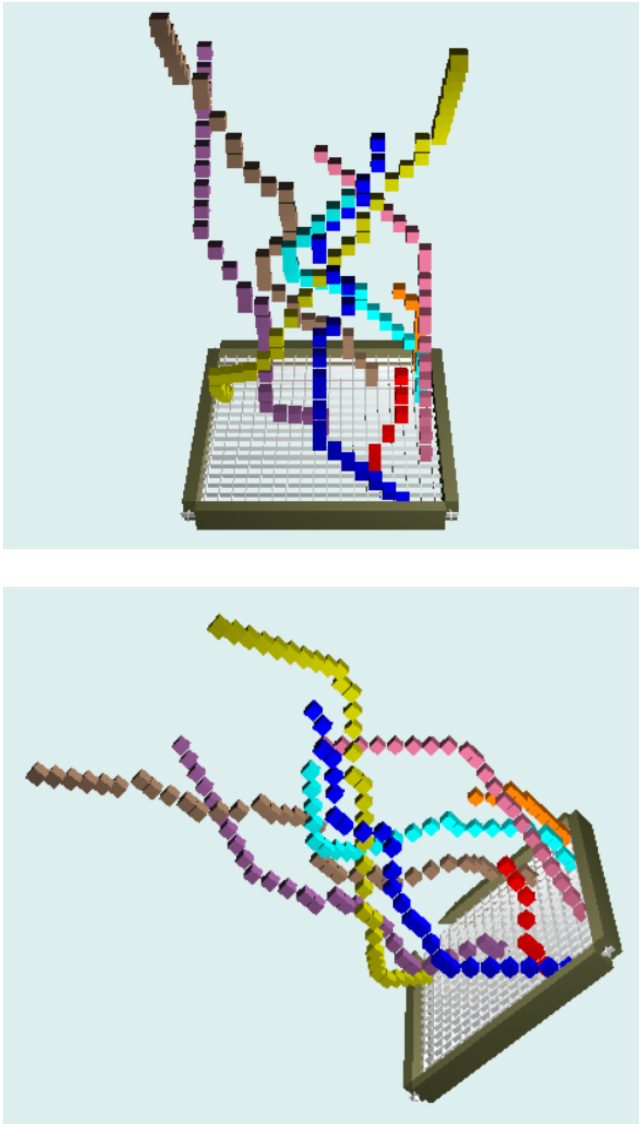
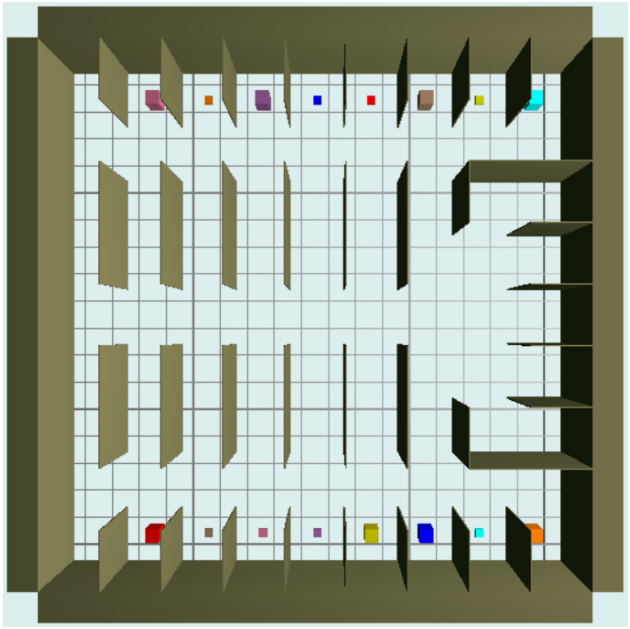


Figura 5.40: Caminhos 3D planeados com base no algoritmo TEA* para o teste 12, representados segundo ângulos diferentes.

8 Robôs na Plataforma Industrial



(a)

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	2	2	2	2
Prioridade Inicial	1	2	3	4
	Castanho	Rosa	Violeta	Ciano
Direção Inicial	6	6	6	6
Prioridade Inicial	5	6	7	8

(b)

Figura 5.41: (a) Posição inicial e final de cada robô é dada pelos cubos de tamanho superior e inferior, respetivamente. (b) Parâmetros de inicialização do teste 13. A prioridade de valor igual a 1 é considerada a maior prioridade.

Tabela 5.13: Tempos de execução dos trajetos planeados pelo TEA* para o teste 13.

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Tempo de Execução Final (s)	18.46	20.27	26.55	23.44
	Castanho	Rosa	Violeta	Ciano
Tempo de Execução Final (s)	25.85	22.33	17.38	33.65

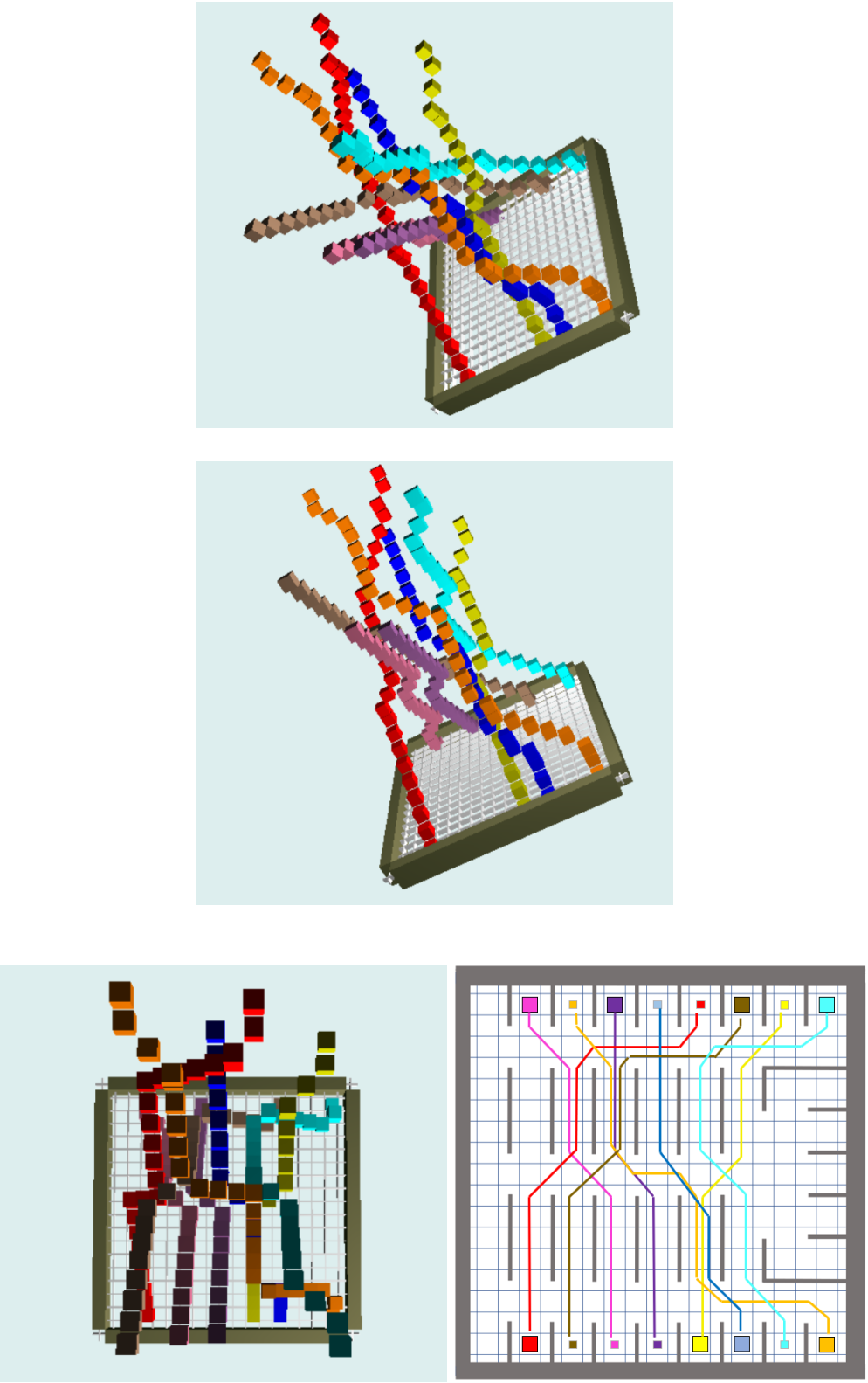
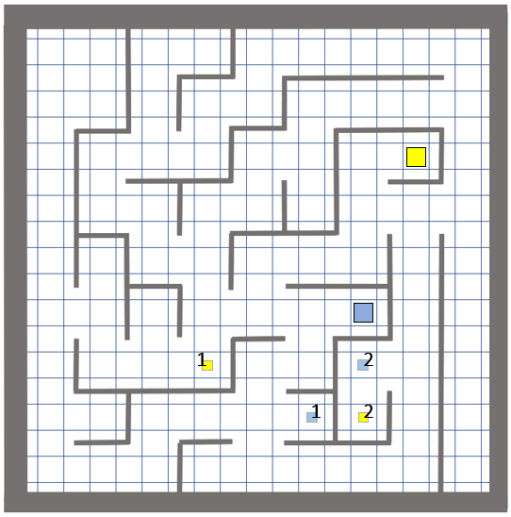


Figura 5.42: Caminhos 3D planeados com base no algoritmo TEA* para o teste 13, representados segundo ângulos diferentes.

5.3.1.8 Casos em que o planeamento falha

Caso 1



(a)

	Robôs	
	Amarelo	Azul
Direção Inicial	6	6
Prioridade Inicial	1	2

(b)

Figura 5.43: (a) Posição inicial de cada robô é dada pelo quadrado de tamanho superior. Submissões são dadas pelos quadrados de tamanho inferior, os quais têm associado a si o número de ordem da tarefa. (b) Parâmetros de inicialização do teste 14. A prioridade de valor igual a 1 é considerada a maior prioridade.

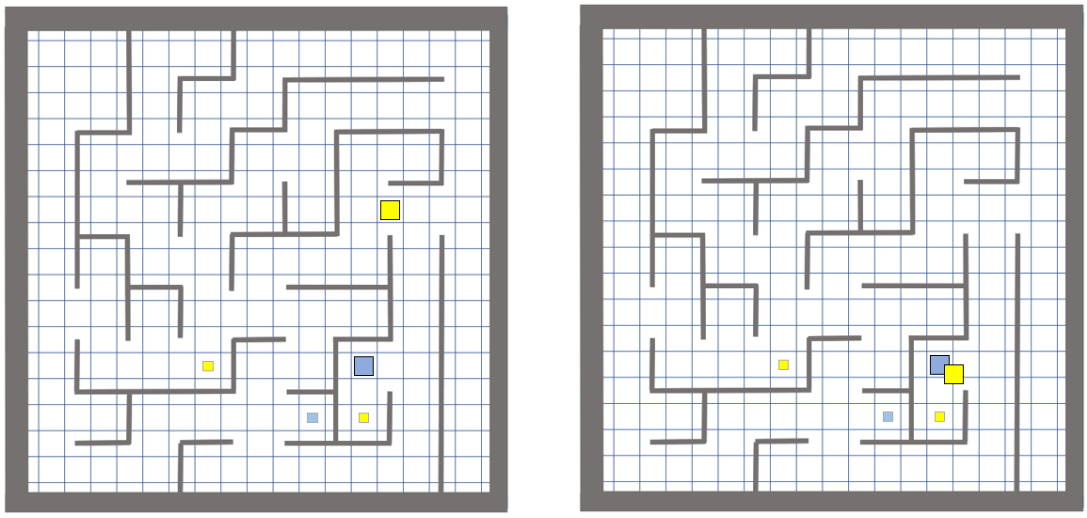
Na figura 5.43 encontra-se ilustrado um dos casos para o qual o algoritmo TEA* implementado não está preparado para resolver.

Como é demonstrado na figura 5.44, o robô azul atinge a célula objetivo da sua última submissão na camada temporal 24. Contudo, nesse instante, o robô amarelo ainda se encontra a executar o seu trajeto. Acontece que, apresentando ele a prioridade superior, assume caminho livre e, na tentativa de atingir o seu ponto final, é expectável que exista uma colisão com o robô azul na camada temporal 32.

O caso descrito apresenta semelhanças a outros já abordados nesta dissertação e solucionados através da implementação da validação inversa das prioridades. Ainda assim, difere dos mesmos, na medida em que deteta a situação mas não consegue planear um caminho alternativo para o robô amarelo, visto que não existe qualquer viabilidade de atingir a célula objetivo sem interferir com o robô azul.

Ainda que o problema não tenha sido solucionado e, por conseguinte, a eficácia do sistema na realização das tarefas seja afetada, é possível evitar que a colisão exista. A partir de uma troca de

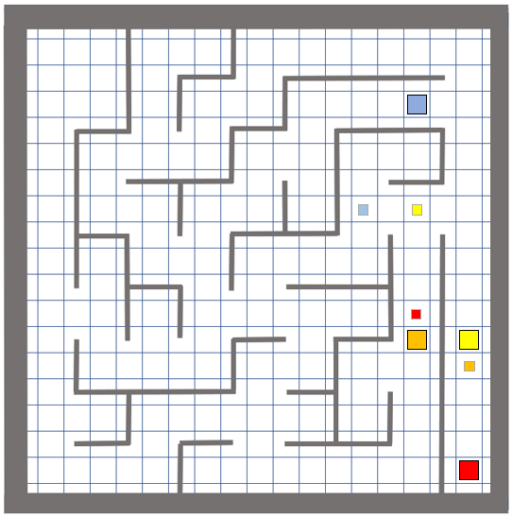
prioridades entre os robôs, o amarelo passa a detetar obstáculo indefinidamente na célula (6,6) e vizinhança. Consequentemente, fica impossibilitado de fechar a sua célula objetivo, excedendo um número de iterações máximo pré-definido no algoritmo TEA*. Assegura-se, assim, a segurança dos robôs.



(a) (b)

Figura 5.44: (a) Camada temporal 24. Robô azul atinge a célula objetivo da sua última submissão. (b) Camada temporal 32. Se robô amarelo mantivesse a prioridade superior, colidiria com o robô azul.

Caso 2



(a)

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	2	2	2	2
Prioridade Inicial	1	2	3	4

(b)

Figura 5.45: (a) Posição inicial de cada robô é dada pelo quadrado de tamanho superior. Submissões são dadas pelos quadrados de tamanho inferior, os quais têm associado a si o número de ordem da tarefa. (b) Parâmetros de inicialização do teste 15. A prioridade de valor igual a 1 é considerada a maior prioridade.

Na figura 5.45 encontra-se ilustrado outro caso para o qual o algoritmo TEA* implementado não está preparado para resolver.

Acontece que, com o intuito de otimizar temporalmente os trajetos planeados, como testado na secção 5.3.1.4, permite-se a troca de prioridades entre robôs, a qual pode comprometer o bom planeamento do sistema. Para este caso em particular, pode ser observado na tabela 5.14, a sequência de trocas que é efetuada quando o planeamento é executado. Em primeiro lugar, os robôs azul e laranja escalam na hierarquia, conseguindo diminuir o tempo de execução dos seus trajetos e deixando o amarelo com uma prioridade inferior. De seguida, tanto o robô amarelo como o vermelho tentam insistentemente conseguir ficar com a prioridade 3, sendo a disputa solucionada apenas quando um limite máximo, pré-definido, de 10 trocas é excedido. Nesse momento, o robô que permanecer no final da hierarquia terá de planear tendo em conta que os trajetos dos outros três serão colocados como obstáculos ao longo do tempo. Para a configuração em análise, representada na figura 5.45, quer seja o robô amarelo ou o vermelho a ficar com a prioridade mais baixa, em nenhum dos casos, será possível encontrar um caminho viável, livre de colisão. Para uma melhor compreensão, é demonstrado nas figuras 5.46 e 5.47, as situações de colisão para ambos os casos.

Tabela 5.14: Sequência de troca de prioridades entre os robôs, quando executado o algoritmo TEA*, para o teste 15 e, quando considerado um limite máximo de 10 trocas.

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Prioridade Inicial	1	2	3	4
Troca 1	2	1	3	4
Troca 2	3	1	2	4
Troca 3	4	1	2	3
Troca 4	3	1	2	4
Troca 5	4	1	2	3
Troca 6	3	1	2	4
Troca 7	4	1	2	3
Troca 8	3	1	2	4
Troca 9	4	1	2	3
Troca 10	3	1	2	4

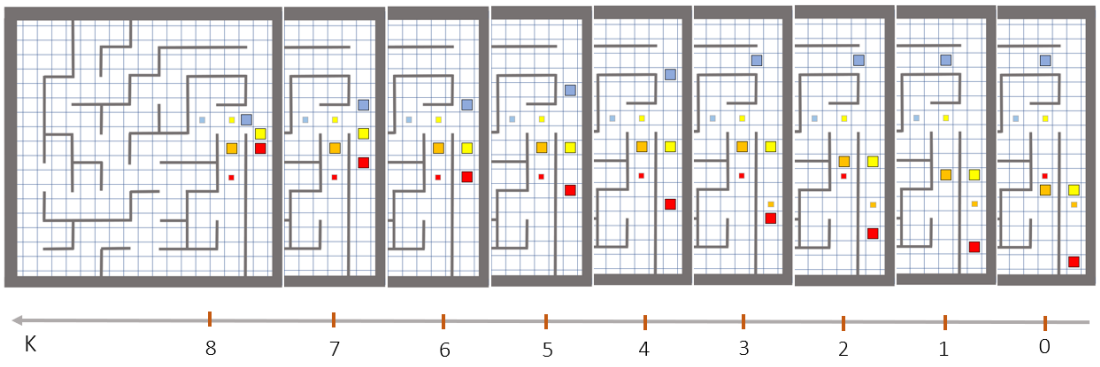


Figura 5.46: Planeamento por TEA*, ao longo das camadas temporais K, para o caso em que o robô vermelho apresenta prioridade 3 e o robô amarelo apresenta prioridade 4.

Sendo o robô vermelho o detentor da prioridade 3, este apresenta a capacidade de planear primeiro que o robô amarelo. Desse modo, excetuando para K={0,1}, o robô vermelho planeia o seu trajeto sem ter em consideração a posição do amarelo. Este detalhe faz com que, na figura 5.46, na camada temporal 8, exista um momento de colisão. Estando o robô amarelo inserido na vizinhança do robô azul, não pode seguir na direção 2 mas, ao mesmo tempo, também não lhe é possível recuar devido à presença do robô vermelho. Assim, passa a coexistir na região de colisão de dois outros robôs, sem caminho alternativo, conduzindo ao insucesso do planeamento.

Por sua vez, quando existe inversão das prioridades entre o vermelho e o amarelo, uma colisão idêntica acontece. Focando na figura 5.47, é possível reparar, na camada temporal 13, que o robô amarelo, em vez de prosseguir em frente, opta por esperar mais próximo do cruzamento, de modo a atingir a sua célula objetivo no menor tempo possível. Isto acontece devido à prioridade mais elevada e, consequente falta de conhecimento acerca da posição do robô vermelho. Desta forma, este ultimo perde qualquer possibilidade de planear um caminho alternativo para evitar a vizinhança mais próxima dos outros dois robôs.

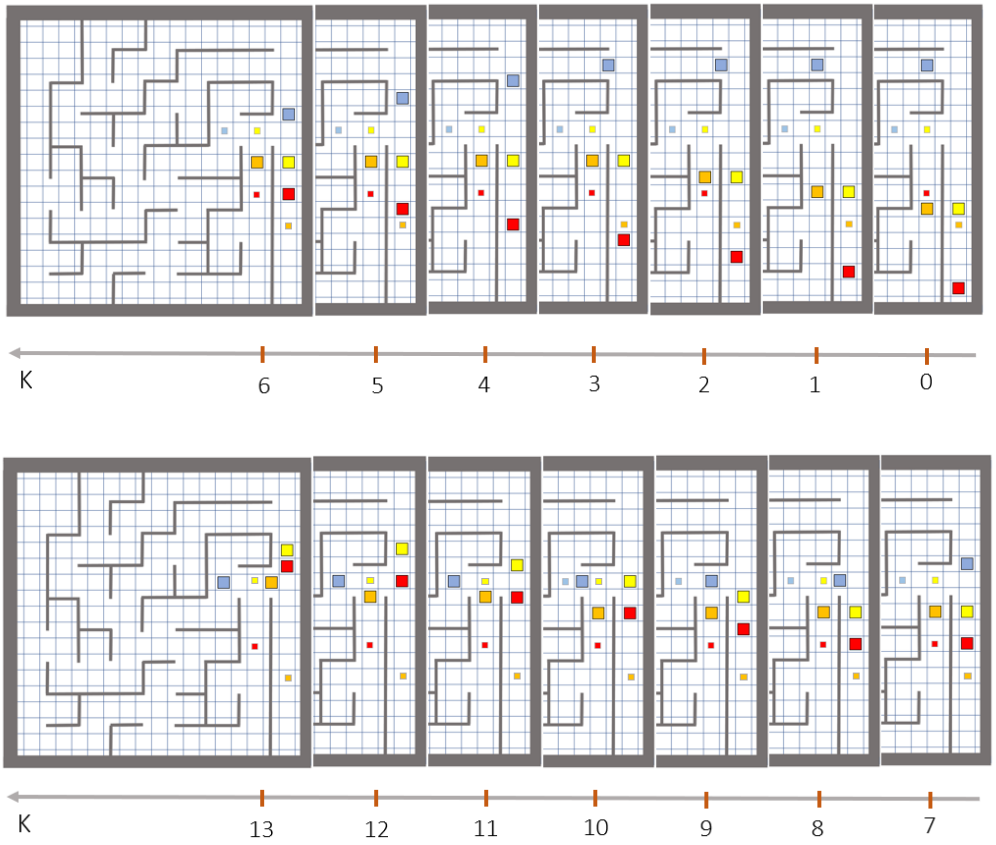


Figura 5.47: Planeamento por TEA*, ao longo das camadas temporais K, para o caso em que o robô vermelho apresenta prioridade 4 e o robô amarelo apresenta prioridade 3.

Ainda que a otimização temporal por troca de prioridades seja vantajosa numa grande generalidade dos casos, no que respeita à eficiência do sistema, situações como a ilustrada neste teste podem acontecer comprometendo o objetivo de não colisão requerido ao algoritmo TEA*. Para este teste em particular, se a referida tentativa de otimização não tivesse sido aplicada, a ausência de colisão teria sido assegurada e as células objetivo teriam sido todas alcançadas (figura 5.48), contudo, o tempo de execução final do sistema seria bastante elevado.

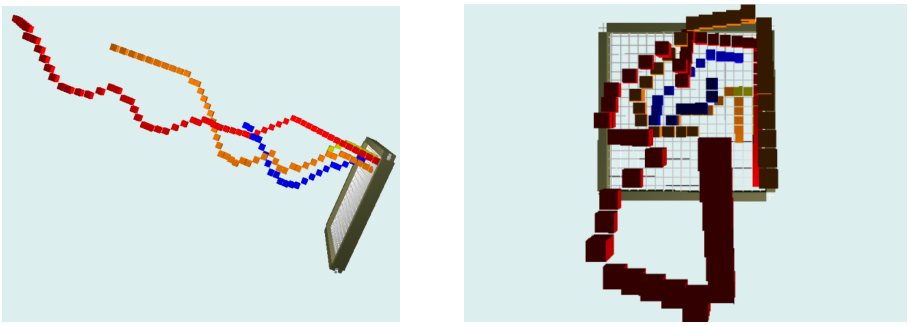


Figura 5.48: Caminhos 3D planeados com base no algoritmo TEA* para o teste 15, considerando a ausência de otimização temporal por troca de prioridades.

5.3.2 SimTwo

Numa primeira fase, a utilização do ambiente de simulação, fornecido pelo SimTwo, foi fundamental para validar a implementação do controlo em malha fechada da trajetória dos robôs, descrito no capítulo 4. Os testes que se seguirão ao algoritmo TEA* pressupõe, igualmente, a validação do controlo de trajetória, visto que a sua ausência implicaria a incapacidade de realizar os trajetos planeados.

No que respeita ao algoritmo de planeamento de trajetória, TEA*, pretende-se testar uma possível otimização temporal. Através da abordagem proposta na secção 3.2.2, ambiciona-se atingir um planeamento mais fidedigno das trajetórias reais dos robôs. Ainda que o planeamento continue a não traduzir com exatidão o comportamento real dos robôs, ao longo do tempo, espera-se que, ao diferenciar temporalmente movimentos simples de movimentos com rotação mais prolongada, se consiga estabelecer uma melhor previsão da futura posição dos robôs face a todos os outros que se encontram no espaço de atuação. De uma forma resumida, a diferenciação é conseguida, estabelecendo equivalências de 1 ou 2 camadas temporais para cada movimento, consoante o tempo consumido.

Assumindo, para os robôs presentes no simulador, uma velocidade linear de 0.12 m/s e uma velocidade angular de 0.66 rad/s, obteve-se os seguintes intervalos de tempo para cada movimento:

Tabela 5.15: Intervalo de tempo para cada movimento.

Movimento	Intervalo de tempo (s)
Deslocamento horizontal ou vertical	1.5
Deslocamento diagonal	2
Rotação de 45º seguida de deslocamento horizontal/vertical	2.69
Um quarto de circunferência entre duas células	2.9
Rotação de 90º seguida de deslocamento horizontal/vertical	3.88

Para os primeiros três movimentos da tabela 5.15, foi estabelecida uma equivalência de 1 camada temporal no planeamento, enquanto que para os dois restantes foram consideradas duas camadas temporais. É importante realçar que o movimento de um quarto de circunferência entre duas células corresponde, na secção 3.2.2, a uma rotação de 45º seguida de movimento diagonal, contudo, na implementação do controlo de trajetória utilizou-se um movimento circular, para conferir maior segurança ao robô, no que respeita à não colisão com as paredes da plataforma.

Uma vez definidas as equivalências, foram, portanto, realizados sete testes que pretendem conduzir a uma análise comparativa entre a solução proposta nesta dissertação e a versão inicial do TEA*, implementada com base na formulação descrita em [14, 36]. Entre os diversos testes, é variado as posições e orientações iniciais dos robôs bem como as suas células de destino. Adicionalmente, é ainda variada a configuração do espaço de obstáculos, de modo a diversificar os testes realizados.

Relativamente ao primeiro teste realizado será feita uma análise mais detalhada do resultado, a qual poderá ser aplicada de igual forma para os restantes que se seguirem. O raciocínio inerente

a cada teste mantém-se invariável. É de notar que se manteve a utilização do mesmo referencial XY e a numeração associada a cada direção nesse mesmo plano (figura 5.49).

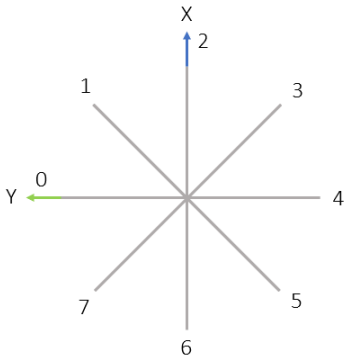
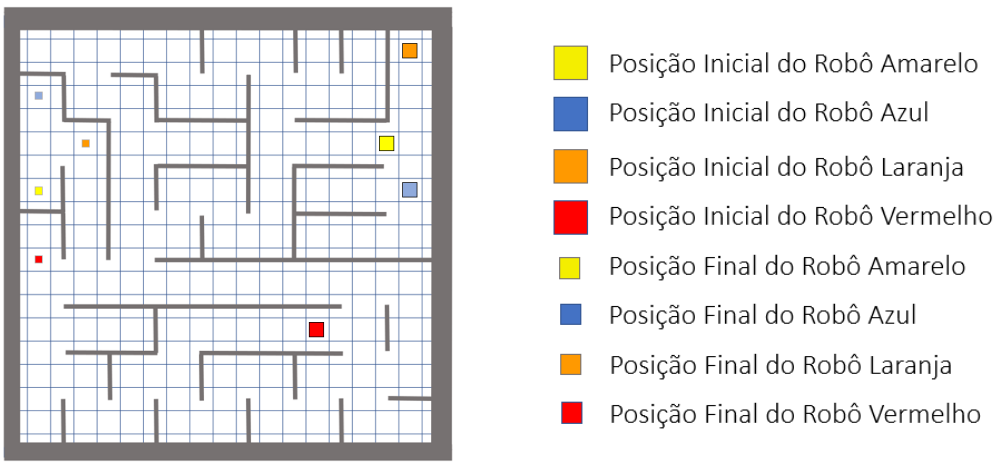


Figura 5.49: Direções utilizadas no plano XY.

Direcionando, portanto, o foco para o caso particular do teste 1, cuja configuração inicial é dada pela figura 5.50, serão feitas em paralelo duas análises. Por um lado, pretende-se observar o trajeto ao nível das camadas temporais, de modo a compreender a viabilidade de executar um determinado caminho, isto é, conhecer, através da estimação das posições futuras dos outros robôs, a existência ou não de colisão e a consequente necessidade de espera para a evitar. Por outro lado, é necessário analisar o custo total do trajeto, uma vez que é este parâmetro que permite ao TEA* escolher a solução ótima para ir de um ponto inicial a um ponto de destino. A conjugação de ambos os fatores em análise conduzirá ao trajeto planeado. Os custos associados a cada movimento entre duas células vizinhas são apresentados na tabela 5.16.



	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	0	2	6	0
Prioridade Inicial	1	2	3	4

Figura 5.50: Configuração inicial do teste 1.

Tabela 5.16: Custos associados a movimentos entre células vizinhas numa camada temporal.

Movimento	Custo (s)
Deslocamento horizontal ou vertical	0.5
Deslocamento diagonal	0.707
Deslocamento nulo devido a espera ou rotação	0.5

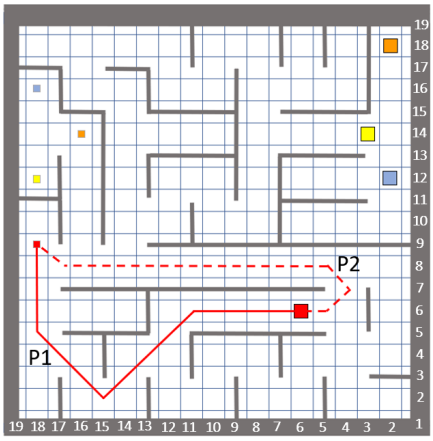


Figura 5.51: Caminhos mais viáveis para o robô vermelho.

Através da figura 5.51, é possível observar que o robô vermelho apresenta como possibilidades os caminhos P1 e P2. Estes apresentariam um custo total do trajeto de 9.940 e 8.621, respetivamente, considerando a movimentação isolada do robô, isto é, sem influência de outros robôs no espaço de atuação. Contudo, estamos perante um caso de sistema multi robótico, pelo que é necessário entrar em conta com o trajeto planeado pelos outros robôs e suas prioridades.

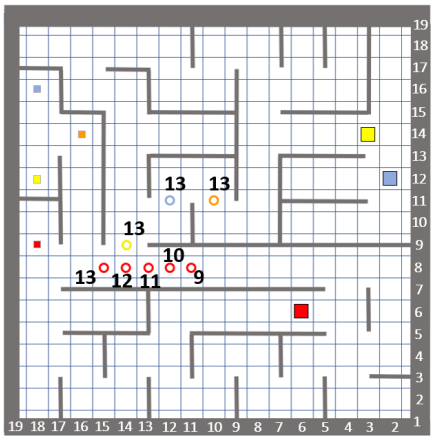


Figura 5.52: Representação das camadas temporais em que se encontrariam os robôs em determinada célula, para a versão inicial do TEA*.

Observando, portanto, a figura 5.52, é visível que, quando o robô amarelo atinge a camada temporal 13, passa a coexistir na vizinhança atual do robô vermelho nessa mesma camada. Apresentando o robô amarelo uma prioridade superior, o robô vermelho terá de planear o seu trajeto

de modo a evitar possíveis colisões. Desse modo, terá de entrar numa situação de espera, ficando imobilizado na célula (12,12). É necessário, no entanto reparar que também os robôs azul e laranja se aproximam da interseção e impedirão o vermelho de passar à sua frente. Mantendo ele a sua posição até à passagem do laranja (prioridade 3), a sua espera contabilizará um total de 9 camadas temporais. Se for calculado, novamente, o custo total do trajeto, o mesmo já assumirá um valor de 13.121 (8.621 + 0.5*9), ultrapassando assim o custo de prosseguir pelo caminho P1. O caminho P2 torna-se, assim, desfavorável e é colocado de parte.

É importante notar que todo este procedimento calculado refere-se à utilização da versão inicial do TEA*, para a qual a movimentação entre células vizinhas corresponde sempre a uma camada temporal. No entanto, tal como observado na tabela 5.15, sendo um movimento circular ou de rotação de 90º aproximadamente o dobro de um deslocamento horizontal/vertical, pode desde já perceber-se que o trajeto do robô amarelo poderá ser atrasado pelos inúmeros obstáculos no caminho. Por sua vez, o robô vermelho apresenta o caminho P2, fundamentalmente em linha reta, prevendo-se um atraso bastante inferior. Deste modo, aplicando a abordagem proposta em 3.2.2, esse comportamento poderá ser traduzido com maior exatidão e no melhor dos casos, possibilitar o robô vermelho de avançar da célula (12,12) em diante, sem ter que esperar a passagem do amarelo, uma vez que este não conseguirá alcançar a sua vizinhança. De facto, repare-se na figura 5.53 que não existe qualquer possibilidade de ambos os robôs colidirem.

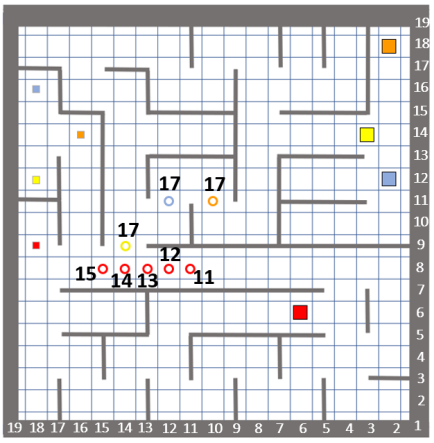


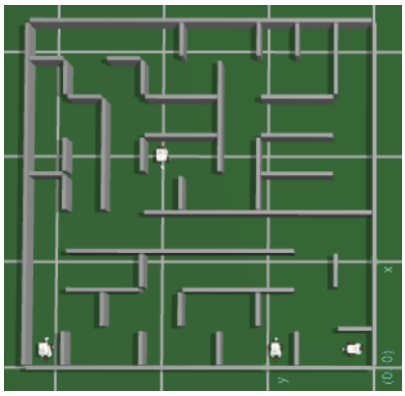
Figura 5.53: Representação das camadas temporais em que se encontrariam os robôs em determinada célula, para a versão modificada do TEA* proposta em 3.2.2.

Sendo o planeamento temporal favorável ao robô vermelho, apenas é necessário confirmar se o custo do trajeto P2, sem espera e para a abordagem proposta em 3.2.2, supera o custo de P1. Com valores de 10.949 e 10.121, respetivamente, para os custos de P1 e P2, é seguro afirmar que o trajeto P2 seria a opção que otimizava temporalmente o trajeto entre o ponto inicial e o ponto de destino. Recorrendo ao simulador desenvolvido, foi possível realizar o teste descrito, confirmando-se toda a base teórica referida. Na tabela 5.17, é possível ainda observar que a versão do TEA* proposta nesta dissertação permitiu reduzir em cerca de 4 segundos o tempo da tarefa do robô vermelho, bem como o tempo de execução médio do conjunto.

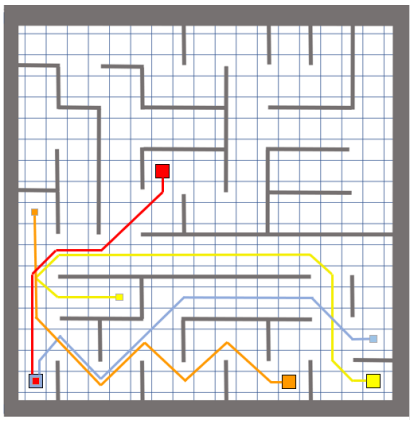
Tabela 5.17: Tempo de execução da tarefa de cada robô para ambas as versões do TEA*.

	Tempo de execução dos robôs (s)			
	Amarelo	Azul	Laranja	Vermelho
Versão inicial do TEA*	44.68	51.69	53.03	31.88
Versão modificada do TEA*	44.70	51.96	55.83	27.47

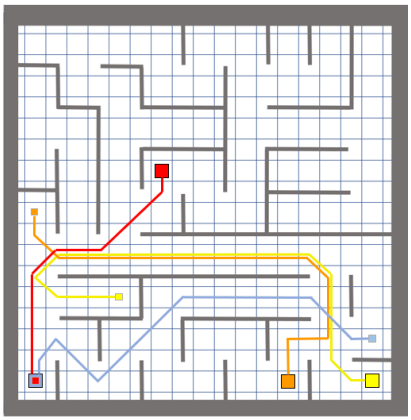
Focando agora nos restantes testes realizados, serão apresentadas as configurações iniciais utilizadas em conjunto com os caminhos planeados verificados em simulação (figuras 5.54, 5.55, 5.56, 5.57, 5.58 e 5.59). Por final, será ainda exposta uma tabela que reúna a informação do tempo de execução relativo a cada robô, para cada teste.



(a)



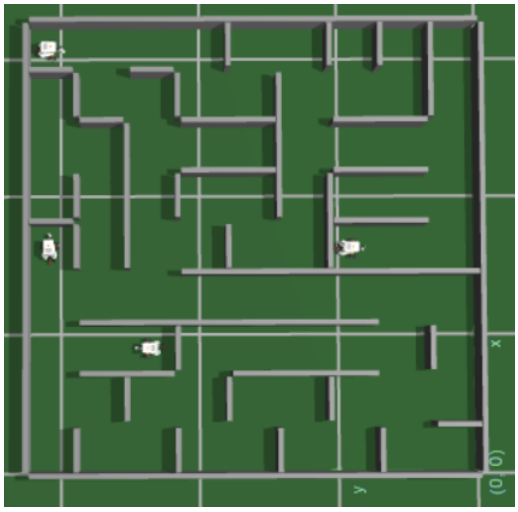
(b)



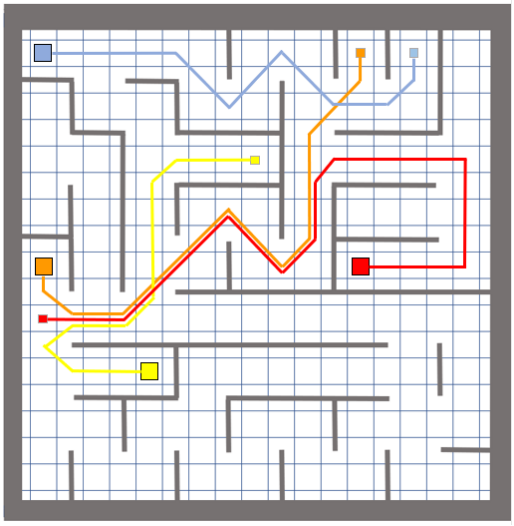
(c)

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	0	2	2	6
Prioridade Inicial	1	2	3	4

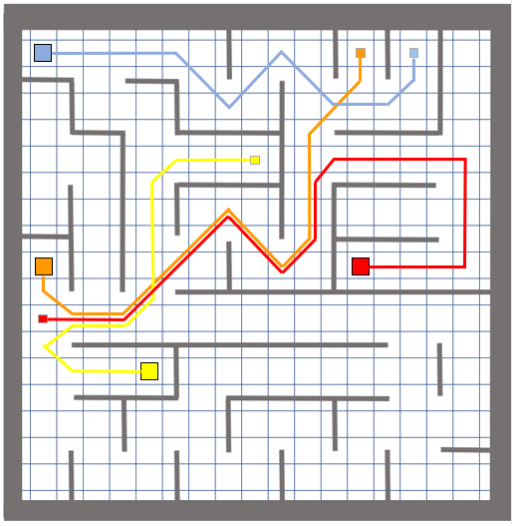
Figura 5.54: (a) Configuração inicial do teste 2 no SimTwo (b) Caminhos planeados quando aplicada a versão inicial do TEA* (c) Caminhos planeados quando aplicada versão modificada do TEA*.



(a)



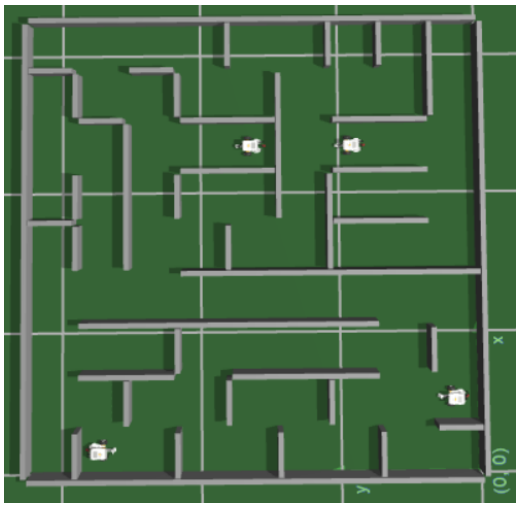
(b)



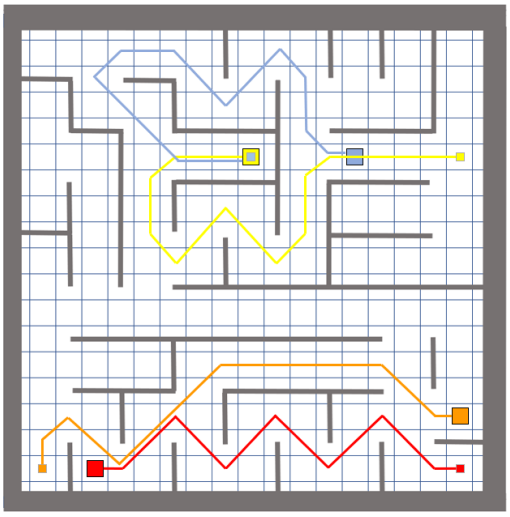
(c)

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	0	4	6	4
Prioridade Inicial	1	2	3	4

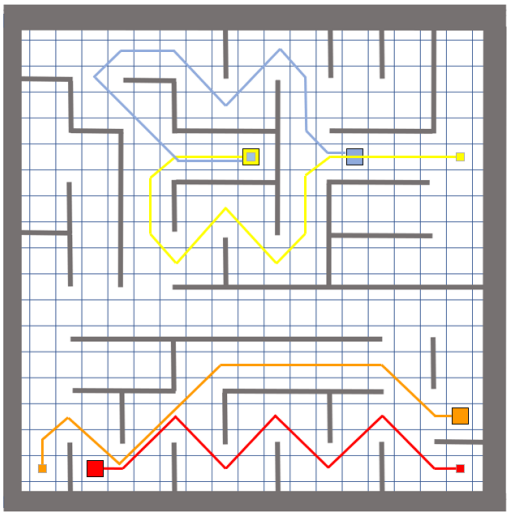
Figura 5.55: (a) Configuração inicial do teste 3 no SimTwo (b) Caminhos planeados quando aplicada a versão inicial do TEA* (c) Caminhos planeados quando aplicada versão modificada do TEA*.



(a)



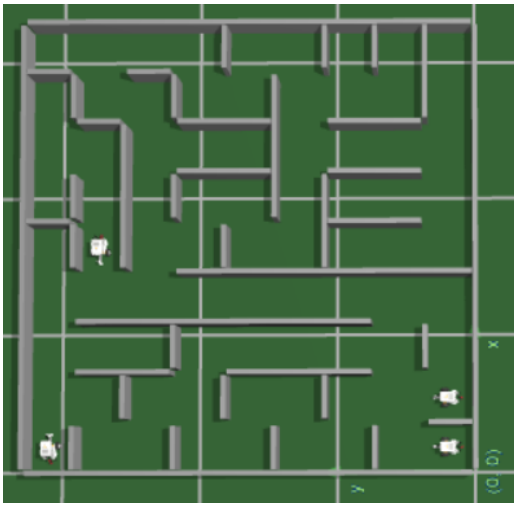
(b)



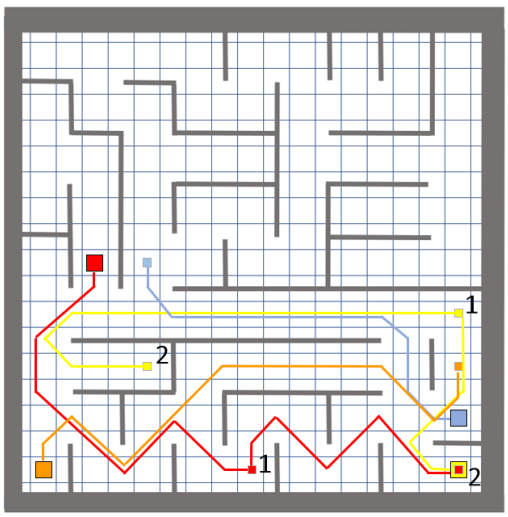
(c)

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	0	0	0	4
Prioridade Inicial	1	2	3	4

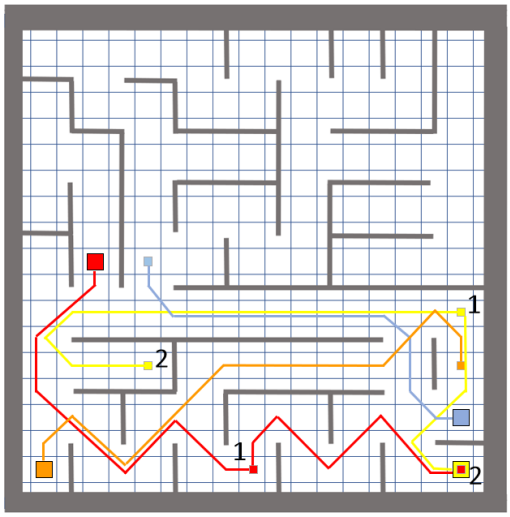
Figura 5.56: (a) Configuração inicial do teste 4 no SimTwo (b) Caminhos planeados quando aplicada a versão inicial do TEA* (c) Caminhos planeados quando aplicada versão modificada do TEA*.



(a)



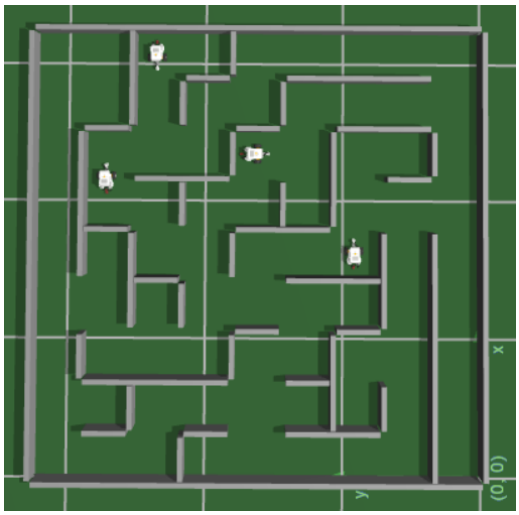
(b)



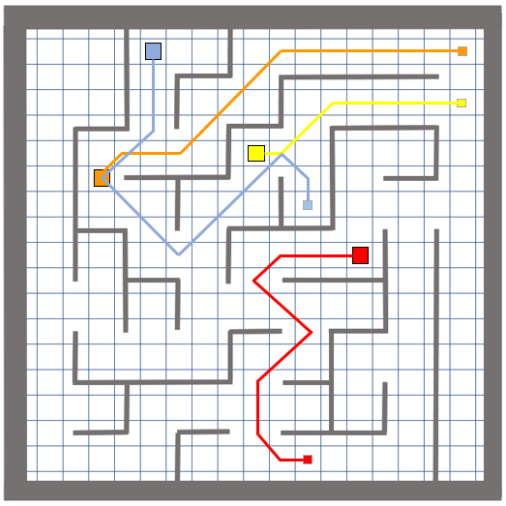
(c)

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	0	0	2	6
Prioridade Inicial	1	2	3	4

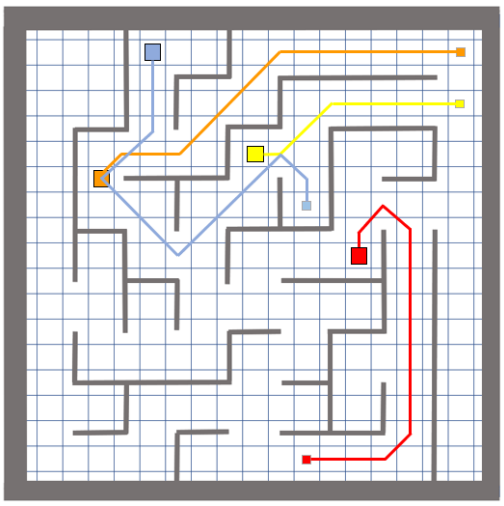
Figura 5.57: (a) Configuração inicial do teste 5 no SimTwo (b) Caminhos planeados quando aplicada a versão inicial do TEA* (c) Caminhos planeados quando aplicada versão modificada do TEA*.



(a)



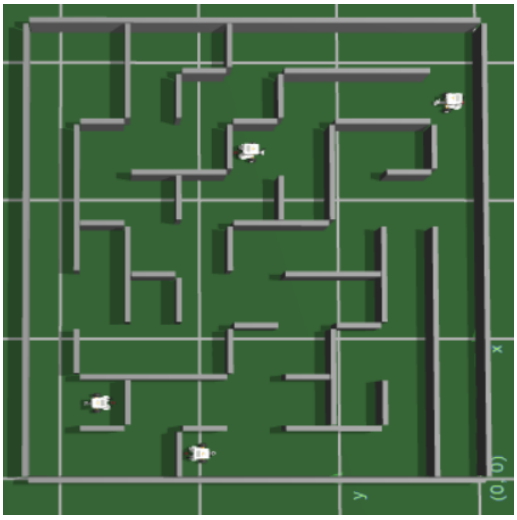
(b)



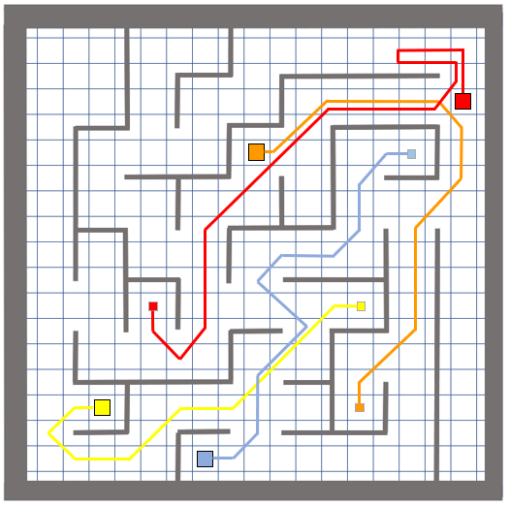
(c)

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	4	6	2	2
Prioridade Inicial	1	2	3	4

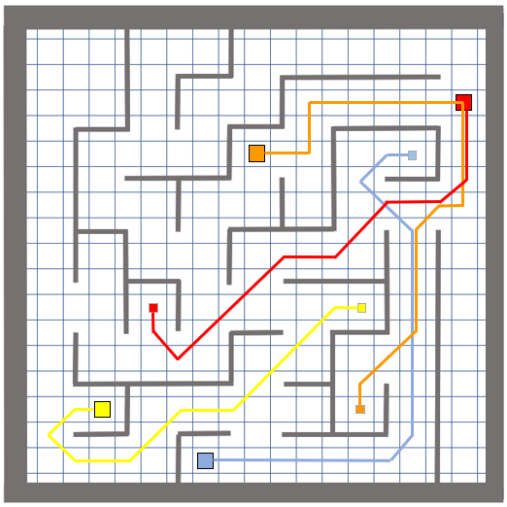
Figura 5.58: (a) Configuração inicial do teste 6 no SimTwo (b) Caminhos planeados quando aplicada a versão inicial do TEA* (c) Caminhos planeados quando aplicada versão modificada do TEA*.



(a)



(b)



(c)

	Robôs			
	Amarelo	Azul	Laranja	Vermelho
Direção Inicial	0	4	4	0
Prioridade Inicial	1	2	3	4

Figura 5.59: (a) Configuração inicial do teste 7 no SimTwo (b) Caminhos planeados quando aplicada a versão inicial do TEA* (c) Caminhos planeados quando aplicada versão modificada do TEA*.

Tabela 5.18: Tempo de execução da tarefa de cada robô para ambas as versões do TEA*.

	Tempo de execução dos robôs (s)			
	Amarelo	Azul	Laranja	Vermelho
Teste 2				
Versão inicial do TEA*	41.33	35.30	47.28	22.83
Versão modificada do TEA*	41.41	35.39	41.83	22.81
Teste 3				
Versão inicial do TEA*	28.71	30.89	38.48	62.07
Versão modificada do TEA*	28.80	30.84	38.37	62.05
Teste 4				
Versão inicial do TEA*	38.75	37.50	35.11	33.20
Versão modificada do TEA*	38.78	37.52	35.33	33.33
Teste 5				
Versão inicial do TEA*	49.08	26.19	37.58	51.80
Versão modificada do TEA*	48.67	26.20	37.66	51.36
Teste 6				
Versão inicial do TEA*	14.69	33.50	26.30	27.00
Versão modificada do TEA*	14.66	34.95	27.23	26.41
Teste 7				
Versão inicial do TEA*	29.74	32.97	37.08	52.01
Versão modificada do TEA*	29.66	35.52	45.78	35.05

Partindo do pressuposto que o tempo final do conjunto corresponde ao intervalo de tempo decorrido para o último robô a terminar a tarefa, analisando os resultados obtidos na tabela 5.18, é possível verificar que para o teste 2 e para o teste 7, esse tempo diminuiu. Para os restantes testes, os trajetos planeados e, consequentemente, os tempos de execução das tarefas mantiveram-se semelhantes. Deste modo, incluindo ainda o teste 1 na análise, obteve-se melhorias temporais em 3 de 7 testes realizados, sendo que no teste 1 apenas se verificaram ao nível do tempo de execução de uma tarefa individual, sem se refletir no tempo final de todo o conjunto.

Foi ainda possível verificar que, quando aplicados os 7 testes, e para ambas as versões do TEA*, os caminhos planeados inicialmente pelo algoritmo mantiveram-se iguais aos caminhos executados desde o ponto inicial ao ponto de destino. Não existiu, portanto, a necessidade de alterar o trajeto de nenhum robô, no decorrer do percurso.

É possível concluir deste estudo que, para os testes realizados, a modificação sugerida ao TEA* na secção 3.2.2, apresentará tempos de execução final do conjunto iguais ou inferiores à versão inicial. Dessa forma, apresenta-se como uma possível otimização ao TEA*, quando aplicado a cenários de simulação realista. Embora o número de testes seja reduzido para dar garantias de sobrevalorização de uma versão face à outra, os resultados apresentam-se bastante positivos e conferem relevância à modificação implementada.

5.4 Conclusão

Após ter sido descrito, detalhadamente, no capítulo 3, os diversos métodos implementados no algoritmo TEA*, foi necessário testar e validar o seu desenvolvimento. Deste modo, foi criada uma plataforma virtual, tendo como base o visualizador 3D GLScene, integrado no ambiente de desenvolvimento Lazarus, através da qual os caminhos planeados para cada um dos casos selecionados, puderam ser observados temporalmente, verificando o sucesso do algoritmo. Ainda que os resultados obtidos tenham sido bastante satisfatórios, é de realçar a inexistência de total eficácia no planeamento. Para o demonstrar, foram ilustradas e, devidamente justificadas, duas situações, para as quais o planeamento falha. É, portanto, possível deparar com casos em que robôs atingem as suas células objetivo, bloqueando as únicas passagens existentes, necessárias para que outros atinjam também os seus destinos. Além disso, também é demonstrado que a otimização por troca de prioridades implementada no algoritmo para reduzir o tempo de execução final, por vezes pode conduzir a situações de colisão que não existiriam se as prioridades iniciais dos robôs não fossem alteradas e os mesmos planeassem caminhos mais longos, temporalmente.

É de notar que as configurações iniciais das posições e orientação dos robôs, assim como o *layout* dimensionado têm uma grande influência no planeamento realizado. Deste modo, deve existir também alguma sensibilidade no dimensionamento de ambos os fatores, de modo a evitar, fundamentalmente, a imobilização de robôs em células objetivo, que possam representar bloqueios com ausência de caminho alternativo para outros. Estabelecendo uma comparação entre as configurações apresentadas ao longo do capítulo, é possível afirmar que a utilização de estações de carga e descarga unitárias, localizadas no espaço tal como na configuração industrial, pode trazer vantagens no que respeita à diminuição de possíveis bloqueios e aumento da eficácia do planeamento.

Adicionalmente, recorreu-se ainda a uma outra plataforma virtual, integrada no ambiente de simulação do software SimTwo, a partir da qual foi possível testar o controlo em malha fechada da trajetória dos robôs, descrito no capítulo 4, e ainda testar tempos de execução das tarefas planeadas pelo algoritmo TEA*, quando utilizada a versão inicial e a versão modificada, proposta na secção 3.2.2. Os resultados obtidos pela análise comparativa entre ambas as abordagens, demonstraram ser bastante satisfatórios, ao possibilitar, para alguns dos casos, a redução do tempo final do conjunto ou de uma tarefa individual, sem nunca conduzir ao aumento do tempo para os restantes testes.

Capítulo 6

Conclusões e Trabalho Futuro

No presente capítulo serão apresentados uma breve revisão aos objetivos propostos e indicações de possível trabalho a realizar futuramente, de modo a dar seguimento aos conteúdos implementados nesta dissertação.

6.1 Satisfação dos Objetivos

A presente dissertação tinha como principal objetivo conseguir obter um movimento cooperativo entre todos os robôs envolvidos num sistema multi robótico, evitando a existência de qualquer situação de colisão ou bloqueio mútuo que pudesse provocar um aumento do tempo de execução final das tarefas ou até mesmo a impossibilidade de realizar alguma delas. Nesse seguimento, foi implementado o algoritmo TEA*, proposto em [14, 36], com recurso à linguagem de programação *Free Pascal* e ao ambiente de desenvolvimento Lazarus v1.6.4. Diversos métodos foram aplicados para resolver situações de colisão ou deadlock que pudessem surgir. Com recurso aos mesmos, gerou-se a possibilidade de obter um planeamento coordenado, evitando a sobreposição de robôs numa mesma célula, em iguais camadas temporais. Além de se procurar a eficácia do algoritmo, apostou-se ainda na eficiência temporal, aplicando algumas otimizações ao planeamento. Após a execução de diversos testes e validação dos caminhos planeados através de uma visualização a três dimensões, é possível aferir que foi obtido um planeamento pouco vulnerável a falhas e otimizado no que respeita a tempos de execução final de todo o conjunto. Ainda que os resultados obtidos tenham sido bastante satisfatórios, é de realçar a inexistência de total eficácia no planeamento, tendo sido a mesma comprovada através de dois testes, para os quais uma situação de bloqueio e outra de colisão foram detetadas, ilustradas e justificadas.

Tentando estabelecer uma ponte de ligação entre a validação do TEA* em simulação e uma futura implementação de movimento coordenado em ambiente real, foi ainda desenvolvido um ambiente de simulação realista com o objetivo de testar um controlo de trajetória em malha fechada que possibilite o replaneamento do algoritmo no decorrer da execução das tarefas. Existindo *feedback* da posição e orientação dos robôs, foi possível o planeamento de novos caminhos e posterior controlo das velocidades linear e angular dos robôs. Exclui-se, assim, a utilização de um

planeamento *offline* que poderia conduzir a situações de colisão ou deadlock, devido a uma má estimação, no tempo, das posições futuras dos robôs envolvidos no sistema. Recorrendo igualmente ao simulador, foi ainda testada uma possível otimização temporal, que pretendia estabelecer uma melhor previsão da futura posição dos robôs face a todos os outros que se encontram no espaço de atuação, diferenciando temporalmente os diferentes movimentos executados. Com base nos testes realizados, foi possível identificar, para 3 de 7 casos, melhorias satisfatórias no tempo de execução final do sistema ou de uma tarefa isolada, sem conduzir nos restantes 4 casos a um aumento do mesmo. Desta forma, foi possível atribuir relevância à modificação implementada no TEA*.

6.2 Trabalho Futuro

Tal como descrito e demonstrado, após a aplicação de testes ao algoritmo, existem ainda casos em que o algoritmo TEA* se apresenta vulnerável a falhas no planeamento. Focando nesses casos, seria relevante a obtenção de uma solução que permitisse contornar essas dificuldades na coordenação. Um possível rumo a seguir poderia passar por manter o replaneamento das trajetórias, ao longo de todas as camadas temporais até que o último robô atinja a sua célula objetivo. Permanecendo o robô ativo, bloqueios como o ilustrado no capítulo 5, poderão vir a ser resolvidos, isto considerando que é permitido aos robôs abandonarem a sua célula objetivo por instantes. Assim, se um robô de menor prioridade chegar à sua célula objetivo, em camadas temporais baixas relativamente a um robô de maior prioridade que também por lá irá necessitar de passar em camadas temporais superiores, o primeiro terá obrigatoriamente de se desviar para evitar colidir com a região obstáculo do segundo. Posteriormente, poderia retornar à sua célula de destino.

Adicionalmente, pode ainda ser implementado um novo método para evitar que um conjunto de troca de prioridades, realizadas com vista a otimização temporal dos trajetos, possam conduzir a uma situação de ineficácia do sistema. Numa futura solução, em vez de se anular por completo a otimização, poderá ser feita uma promoção dos robôs de menor prioridade, para os quais o planeamento falha, até que o algoritmo retorne novamente sucesso na execução. Deste modo, poderá ser conseguido manter parte da otimização efetuada. Apenas no pior dos casos teriam de ser anuladas todas as trocas de prioridade feitas à partida.

Direcionando para a aplicação do TEA* a situações de coordenação em ambiente real, poderia ainda ser dado seguimento à análise comparativa entre a versão inicial do TEA*, implementada com base na formulação descrita em [14, 36], e a modificação proposta nesta dissertação. Um aumento dos testes realizados poderá conferir, à segunda, maior relevância se se comprovar a existência nula ou reduzida de casos em que a mesma possa conduzir a uma aumento do tempo de execução final do conjunto multi robótico, face às melhorias conseguidas. Melhores equivalências entre os movimentos dos robôs e as camadas temporais utilizadas, podem ser experimentadas, na expectativa de aproximar ainda melhor a estimação da posição futura real dos robôs.

Por final, seria de extrema relevância quebrar barreiras entre a simulação e o a realidade, procurando a validação do TEA* em cenários de ambiente real.

Anexo A

Algoritmo A*

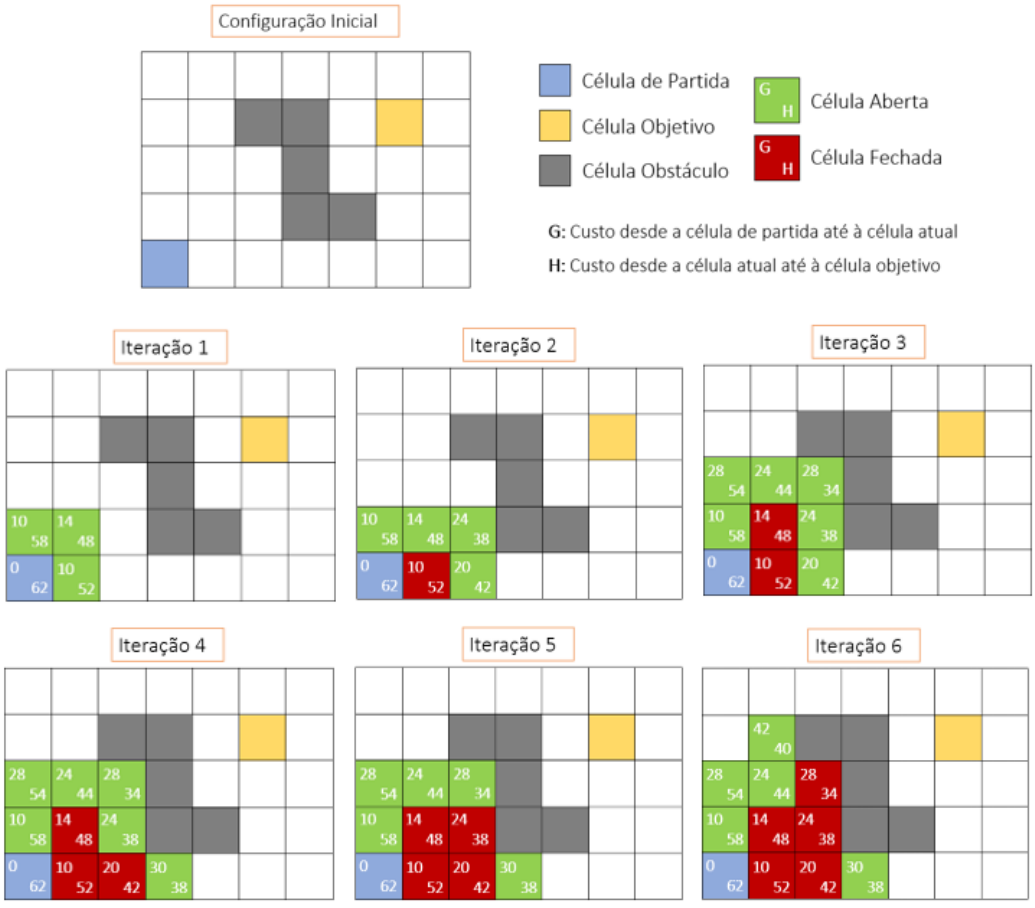


Figura A.1: Algoritmo A* - parte 1

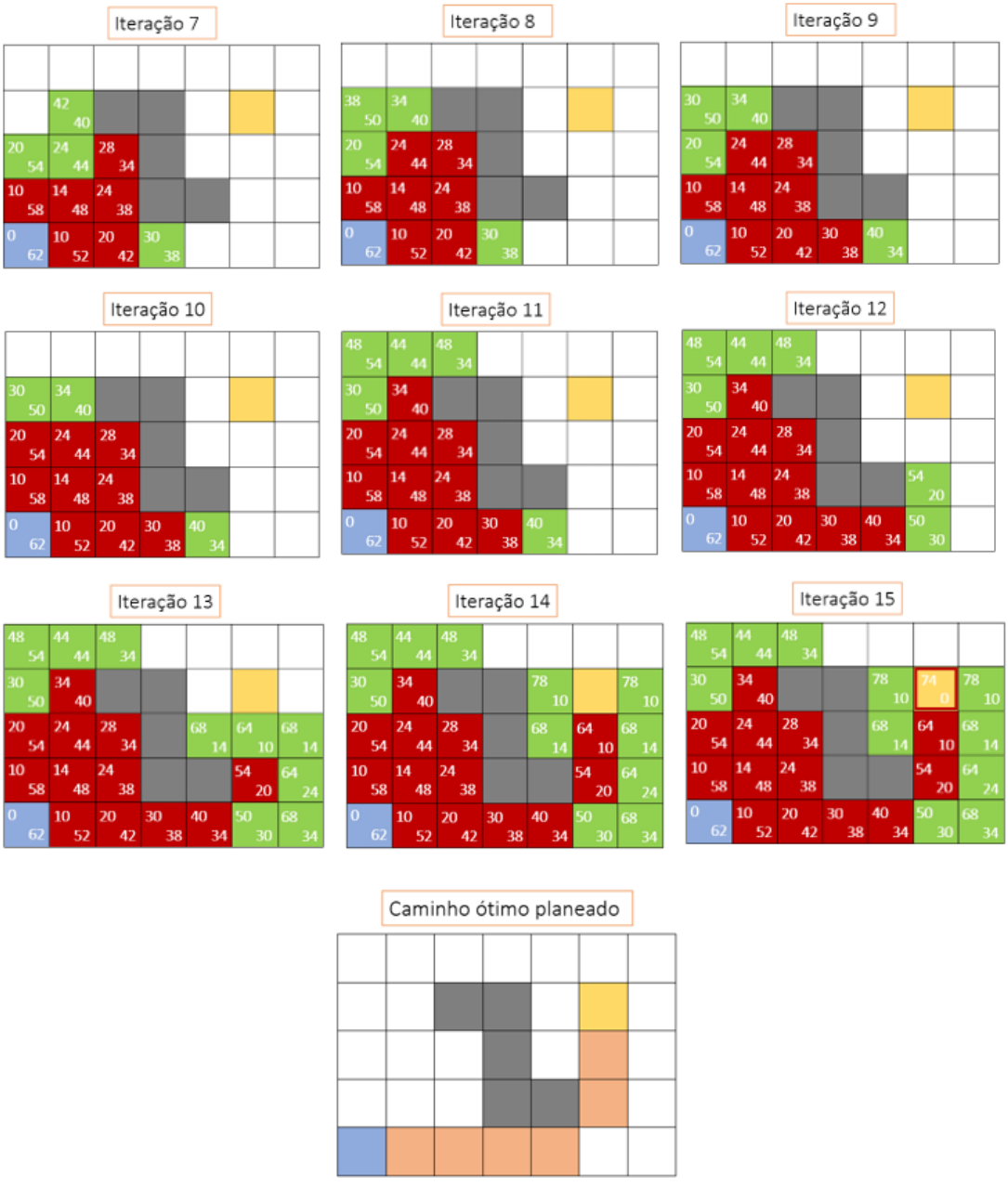


Figura A.2: Algoritmo A* - parte 2

Anexo B

Esquema Elétrico dos Robôs

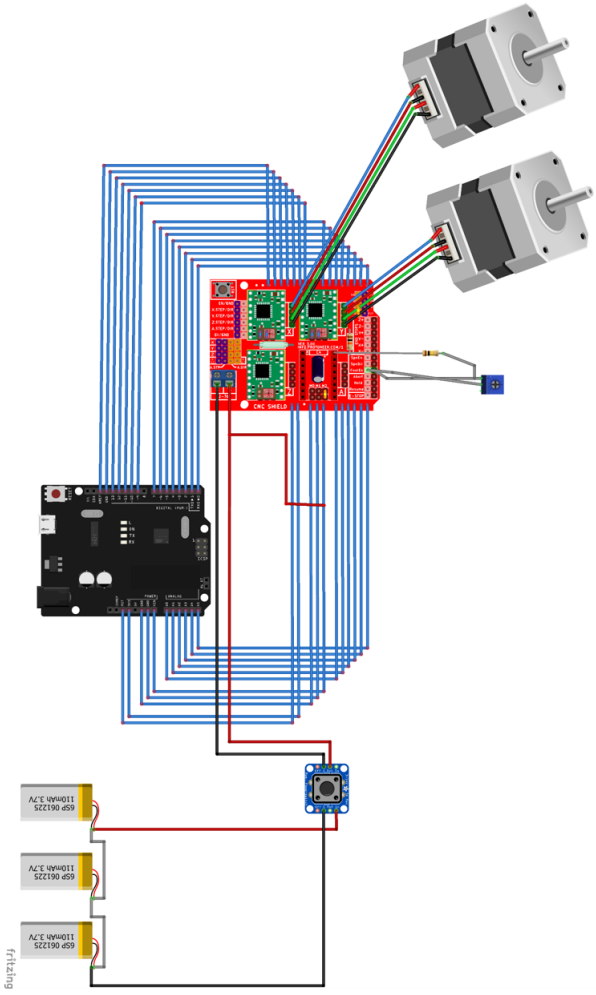


Figura B.1: Esquema elétrico dos robôs

Referências

[1] Jean-Claude Latombe. *Robot motion planning*. 1991. doi:10.1016/1049-9660(91)90042-N.

[2] Pedro Costa. *Planeamento Cooperativo de Tarefas e Trajectórias em Múltiplos Robôs*. Tese de doutoramento, 2011.

[3] Jur Van Den Berg, Ming Lin, e Dinesh Manocha. Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation. Em *IEEE Internatioal Conference on Robotics and Automation*, páginas 1928–1935, 2008. doi:10.1109/ROBOT.2008.4543489.

[4] Renee Jansen e Nathan Sturtevant. A New Approach to Cooperative Pathfinding. Em *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, páginas 1401–1404, 2008. URL: http://web.cs.du.edu/{~}sturtevant/papers/coop{_}path.pdf.

[5] António Paulo Moreira, Paulo J.Costa, e Pedro Costa. Real-time path planning using a modified A* algorithm. Em *9th Conference on Mobile Robots and Competitions*, páginas 141–146, 2009.

[6] Sven Koenig e Maxim Likhachev. Incremental A*. Em *In Advances in Neural Information Processing Systems (NIPS)*, páginas 1539–1546, 2002.

[7] Sven Koenig e Maxim Likhachev. Fast Replanning for Navigation in Unknown Terrain. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, XX(Y), 2002. doi:10.1109/ROBOT.2002.1013481.

[8] Nathan Sturtevant e Michael Buro. Partial Pathfinding Using Map Abstraction and Refinement. Em *Proceedings of the National Conference on Artificial Intelligence*, páginas 1392–1397, 2005. URL: http://www.aaai.org/Papers/AAAI/2005/AAAI05-221.pdf.

[9] Nathan Sturtevant e Michael Buro. Improving Collaborative Pathfinding Using Map Abstraction. Em *Proceedings of the Second Artificial Intelligence for Interactive Digital Entertainment Conference*, páginas 80–85, 2006.

[10] Wenjie Wang e Wooi Boon Goh. Multi-robot path planning with the spatio-temporal A* algorithm and its variants. Em *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, páginas 313–329, 2011.

[11] Joana Santos, Pedro Costa, Luis F. Rocha, A. Paulo Moreira, e Germano Veiga. Time enhanced A: Towards the development of a new approach for Multi-Robot Coordination. Em *Proceedings of the IEEE International Conference on Industrial Technology*, páginas 3314–3319, 2015. doi:10.1109/ICIT.2015.7125589.

[12] A.Shoshani E.G.Coffman, Jr, M.J.Elphick. System Deadlocks. *ACM Computing Surveys (CSUR)*, 3(2):67–78, 1971.

[13] Changyun Wei, Koen V. Hindriks, e Catholijn M. Jonker. Altruistic coordination for multi-robot cooperative pathfinding. *Applied Intelligence*, 44(2):269–281, 2016. doi:10.1007/s10489-015-0660-3.

[14] Joana Santos e Pedro Costa. Validation of a Time Based Routing Algorithm using a Realistic Automatic Warehouse Scenario. 2015.

[15] Iris F.A. Vis. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709, 2006. arXiv:arXiv:1011.1669v3, doi:10.1016/j.ejor.2004.09.020.

[16] Yong-Hwi Kim Yong-Hwi Kim e Byung Kook Kim Byung Kook Kim. A multi-robot task planning system minimizing the total execution time for hospital service. Em *Control Automation and Systems (ICCAS), 2010 International Conference on*, páginas 379–384, 2010.

[17] Rajeeva Lochana Moorthy, Wee Hock-Guan, Ng Wing-Cheong, e Teo Chung-Piaw. Cyclic deadlock prediction and avoidance for zone-controlled AGV system. *International Journal of Production Economics*, 83(3):309–324, 2003. doi:10.1016/S0925-5273(02)00370-5.

[18] E. G. Hernández-Martinez, Sergio A. Foyo-Valdés, Erika S. Puga-Velazquez, e J. A.Meda Campaña. Motion coordination of AGV's in FMS using petri nets. Em *IFAC-PapersOnLine*, volume 28, páginas 187–192, 2015. doi:10.1016/j.ifacol.2015.06.079.

[19] Y.U. Cao, A.S. Fukunaga, A.B. Kahng, e F. Meng. Cooperative mobile robotics: antecedents and directions. *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, 1:226–234, 1997. URL: http://ieeexplore.ieee.org/document/525801/, doi:10.1109/IROS.1995.525801.

[20] Antonio Krnjak, Ivica Draganjac, Stjepan Bogdan, Tamara Petrovic, Damjan Miklic, e Zdenko Kovacic. Decentralized control of free ranging AGVs in warehouse environments. Em *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2015-June, páginas 2034–2041, 2015. doi:10.1109/ICRA.2015.7139465.

[21] Roberto Olmi, Cristian Secchi, e Cesare Fantuzzi. Coordinating the motion of multiple AGVs in automatic warehouses. 2010.

[22] Nenad Smolic-rocak, Stjepan Bogdan, Zdenko Kovacic, e Tamara Petrovic. Time Windows Based Dynamic Routing in Multi-AGV Systems. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING*, 7(1):151–155, 2010.

[23] T. Lozano-Pérez Erdmann, M. On Multiple Moving Objects. *Algorithmica*, 2:477–521, 1987.

[24] Dit-Yan Yeung e George A Bekey. A Decentralized Approach to the Motion Planning Problem for Multiple Mobile Robots. Em *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, páginas 1779–1784, 1987.

[25] Parker Guo, Yi , E. A Distributed and Optimal Motion Planning Approach for Multiple Mobile Robots. Em *Proceedings - IEEE International Conference on Robotics and Automation*, volume 3, páginas 2612–2619, 2002.

[26] B H Lee e C S G Lee. Collision-Free Motion Planning of Two Robots. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS*, 17(1):21–32, 1987.

[27] P Fiorini e Z Shiller. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17(Copyright 1998, IEE):760–772, 1998. URL: http://dblp.uni-trier.de/db/journals/ijrr/ijrr17.html{#}FioriniS98, arXiv:arXiv:1011.1669v3, doi:Doi 10.1177/027836499801700706.

[28] S.Sekhavat Z. Shiller, F. Large. Motion Planning in Dynamic Environments : Obstacles Moving Along Arbitrary Trajectories University of California. Em *IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[29] Zvi Shiller, Oren Gal, e Ariel Raz. Adaptive Time Horizon for On-Line Obstacle Avoidance in Dynamic Environments. páginas 3539–3544, 2011.

[30] S S Ge e Y J Cui. Dynamic Motion Planning for Mobile. *Electrical Engineering*, 13(Med):207–222, 2000.

[31] Peter E. Hart, Nils J. Nilsson, e Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. URL: http://portal.acm.org/citation.cfm?doid=1056777.1056779, doi:10.1145/1056777.1056779.

[32] Ko Hsin Cindy Wang e Adi Botea. Tractable multi-agent path planning on grid maps. Em *IJCAI International Joint Conference on Artificial Intelligence*, páginas 1870–1875, 2009.

[33] Anthony Stentz. Optimal and Efficient Path Planning for Partially Known Environments. Em *In Proceedings IEEE International Conference on Robotics and Automation*, número May, páginas 3310–3317, 1994. URL: http://link.springer.com/10.1007/978-1-4615-6325-9{_}11.

[34] David Silver. Cooperative Pathfinding. Em *Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference*, páginas 117–122, 2005.

[35] A.J.MacDonald Holte, Robert C , M.B.Perez, R.M.Zimmer. Hierarchical A *: Searching Abstraction Hierarchies Efficiently. Em *Proceedings of the National Conference on Artificial Intelligence*, páginas 530–535, 1996.

[36] Joana Santos, Pedro Costa, e Germano Veiga. A * Based Routing and Scheduling Modules for Multiple AGVs in an industrial scenario.

[37] Claude Barfield e Mark Groombridge. Avoiding deadlock in multitasking systems. *IBM Systems Journal*, 7(2):74–84, 1968. doi:10.4324/9780203563274.

[38] ZhiWu Li, NaiQi Wu, e MengChu Zhou. Deadlock Control of Automated Manufacturing Systems Based on Petri Nets; A Literature Review. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(4):437–462, 2012. doi:10.1109/TSMCC.2011.2160626.

[39] a Yalcin e T O Boucher. Deadlock avoidance in Flexible Manufacturing Systems. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 16(4):424–429, 2000.