FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Planeamento de Trajetórias e Controlo de um Robô Omnidirecional

Ana Rita Marques Rodrigues

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Dr. Pedro Gomes Da Costa

Co-orientador: Prof. Dr. José Magalhães Lima

18 de Julho de 2017

Resumo

Esta dissertação aborda o problema do planeamento de caminhos para robôs móveis omidirecionais autónomos que operam em ambientes conhecidos. O planeamento de caminhos visa, de uma forma geral, permitir a um robô gerar um caminho entre dois pontos de um mapa da forma mais eficiente possível e de modo a não colidir com obstáculos. O controlador de trajetórias, por sua vez, visa definir as velocidades a aplicar ao robô, de modo a este seguir o caminho desejado de uma forma rápida com grande precisão, ou seja, com o menor erro possível.

Para permitir ao robô planear o seu percurso da melhor forma, exploraram-se diferentes metodologias de planeamento de caminhos, sendo analisadas as vantagens e desvantagens de cada uma delas. Para representar o ambiente, foram selecionados e analisados mapas decompostos em células fixas, em *quadtrees*, em *framed quadtrees* e ainda é sugerida uma nova abordagem para a decomposição em células denominada K-*Framed Quadtrees*. Para encontrar um caminho a partir dessas representações do mapa, é utilizado o algoritmo de pesquisa de grafos A-*star* (A$\star$). É também feita uma análise comparativa entre as representações do mapa implementadas, de forma a concluir em que casos os algoritmos devem ser aplicados.

De forma a controlar o movimento do robô para o mesmo percorrer a trajetória desejada, é implementado um controlador não linear, denominado *Trajectory Linearization Control* (TLC) que consiste na linearização em torno de uma trajetória desejada, permitindo o seguimento de qualquer trajetória, e estabilidade ao longo da mesma sem interpolação dos ganhos.

Abstract

This dissertation addresses the path planning problem for autonomous omidirectional mobile robots that operate in known environments. Path planning aims, in general, to allow a robot to generate a path between two points on a map as efficiently as possible, avoiding collidin with obstacles. On the other hand the trajectory controller aims to define which velocities to apply to the robot, so that it follows the desired path swiftly and precisely, with the smallest error possible. In order to allow the robot to plan its path in the best possible way, several methodologies for path planning were explored, analyzing the existing advantages and disavantages of each one of them.

The environment was represented using several analyzed and selected maps decomposed in grid, quadtrees, framed quadtrees and with a new approach for decomposition into cells, called K- Framed Quadtrees. So that a path could be found from these map representations, the A-start (A$\star$) graph search algorithm was used. A comparative analysis between the implemented map representations was also performed, in order to conclude in which cases the algorithms should be applied. The robot's movement could be controlled in order to tread the desired trajectory, a non-linear controller called Trajectory Linearization Control (TLC) was implemented, which consists in the linearization around a desired trajectory, allowing to follow any path with stability without interpolating the gains.

Agradecimentos

A elaboração desta dissertação só foi possível graças ao contributo, de forma direta ou indireta, de algumas pessoas às quais gostaria de expressar o meu agradecimento.

Em primeiro lugar, quero agradecer ao meu orientador, Doutor Pedro Costa e ao meu co-orientador, Doutor José Lima pela sua disponibilidade, os conhecimentos transmitidos e as palavras de incentivo que foram essenciais para o desenrolar do projeto.

Agradeço também a todos os colegas e amigos que contribuíram para o desenvolvimento desta dissertação, bem como a todos aqueles que conheci e me acompanharam neste percurso académico. Por último, mas não menos importante, um profundo reconhecimento à minha família, em especial aos meus pais e irmãos, pelo apoio incondicional, incentivo e paciência ao longo destes anos. Dedico o mesmo sentimento aos meus amigos mais próximos.

Ana Rodrigues

"Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time"

Thomas A. Edison

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

VG	*Visibility Graph*
RVG	*Reduced Visibility Graph*
DV	Diagrama *Voronoi*
PRM	*Roadmap* Probabilístico
RRT	*Rapidly-exploration Random Tree*
FM	*Fast Marching*
VO	*Velocity Obstacle*
EBVO	*ellipse-based VO*
DQ	Decomposição em *quadtrees*
DFQ	Decomposição em *framed quadtrees*
A $\star$	A-*star*

Capítulo 1

Análise Introdutória

1.1 Contexto

A integração da robótica em ambientes industriais é uma prática cada vez mais comum, pois os robôs efetuam tarefas com maior rapidez. Os robôs móveis têm sido alvo de vários estudos de forma a inclui-los em ambientes complexos como domésticos, industriais, urbanos e militares. Com vista ao desenvolvimento e inovação da robótica no ramo industrial surge a associação EUROC, European Robotics Challenges, que visa impulsionar a colaboração entre a comunidade industrial e a comunidade de investigação, recorrendo a desafios de relevância industrial na robótica [7]. Os robôs omnidirecionais surgem com a necessidade de os robôs se movimentarem mais agilmente nos ambientes nos quais se encontram inseridos, tendo estes a capacidade de se movimentar em qualquer direção sem a necessidade de alterar a sua orientação. Os robôs omnidirecionais destacam-se de robôs direcionais pela sua maior mobilidade, assentando no desenvolvimento e implementação de controladores sofisticados para o planeamento de trajetória.

1.2 Motivação

O planeamento de trajetórias e o controlo dos sistemas robóticos implementados nos diversos ambientes é uma preocupação, uma vez que estes devem ser ágeis, eficazes e rápidos, mantendo os padrões de segurança, evitando colisões e situações que possam colocar em perigo seres humanos presentes no mesmo ambiente. O facto de os robôs omnidirecionais permitirem o movimento em qualquer direção sem que seja alterada a sua orientação traduz-se num problema de otimização no que diz respeito ao planeamento de trajetórias destes robôs. A motivação desta dissertação assenta no desenvolvimento de um algoritmo de planeamento de trajetórias, adaptado a um ambiente dinâmico, que permita a deslocação do robô para um determinado local, previamente definido, de forma rápida, eficaz e segura. Neste contexto a trajetória selecionada terá em conta os vários graus de liberdade associados ao movimento do robô. De forma a permitir ao robô seguir a trajetória

desejada, esta dissertação pretende o desenvolvimento de um controlador de trajetórias adapatado ao robô omnidirecional presente nesta dissertação. Este deve permitir o seguimento da trajetória pretendida com o menor erro possível.

1.3 Objetivo

Nesta dissertação pretende-se contribuir para o desenvolvimento de uma plataforma móvel associada a um robô omnidirecional presente no laboratório de Robótica. O robô omnidirecional deverá ser capaz de se deslocar o mais rápido possível, sem pôr em causa os padrões de segurança.

O principal objetivo é o desenvolvimento e implementação de um algoritmo de planeamento de trajetórias, que visa encontrar uma sequência contínua de posições válidas, entre uma localização inicial e uma localização final, que permita ao robô mover-se em direção ao alvo final, tendo em consideração a capacidade deste se movimentar em todas as direções. Neste sentido será implementado o algoritmo A$\star$ aplicado a mapas representados por decomposição em células. Pretende-se que o resultado gerado demonstre um bom compromisso entre a trajetória definida e o tempo de processamento associado à mesma.

O segundo objetivo visa a implementação de um controlador de trajetórias aplicado a um robô omnidirecional, que permita o seguimento de diversos caminhos com um erro pequeno. O controlador deverá permitir um movimento fluido, com poucas oscilações.

Capítulo 2

Revisão bibliográfica

Neste capitulo serão mencionados métodos usados no planeamento de trajetórias, algoritmos de pesquisa de grafos e controladores de trajectórias aplicados a robôs omnidirecionais.

2.1 Planeamento de Trajetórias

Dado a localização atual e um ponto de destino, o planeamento de trajetórias consiste na identificação de um caminho a percorrer pelo robô, de forma a alcançar a posição pretendida. Durante a execução da trajetória definida, o robô deve reagir a eventos imprevistos, tais como obstáculos, redefinindo a sua trajetória de modo a evitar colisões.

Inicialmente, é necessário definir o espaço de configuração denominado $C_{espaço}$, que traduz todas as configurações possíveis do sistema. O $C_{espaço}$ é composto por duas partes, a parte livre de obstáculos, que consequentemente, podem pertencer à trajetória a definir, denominada espaço livre C_{livre}, e a parte ocupada por obstáculos, denominada $C_{obstáculos}$. Frequentemente, é aplicada uma redução do espaço de configuração, reduzindo o robô a um único ponto [2]. Para tal, todos os obstáculos devem sofrer uma expansão, de modo a que a trajetória a ser definida não permita a colisão do robô com os obstáculos.

Após a definição do espaço de configuração recorre-se a técnicas de planeamento de trajetórias de modo a obter o caminho que permite o robô deslocar-se para o ponto pretendido.

2.1.1 *Roadmap*

Este algoritmo baseia-se na criação de nós que representam localizações, e ligações entre os mesmos que representam possíveis caminhos entre os nós. *Roadmap* são ligações dos pontos inicial e final efetuadas no espaço livre através de uma rede de curvas uni-dimensionais. Esta representação permite que o problema de planeamento de trajetórias se reduza a um problema de ligações entre as configurações inicias e finais do *Roadmap*. Neste método são produzidos grafos que necessitam posteriormente de ser pesquisados através de algoritmos apropriados para obter a trajetória a percorrer pelo robô.

De seguida, são apresentadas algumas das técnicas de construção de *Roadmap.*

2.1.1.1 *Visibility Graph* (VG)

Visibility graph é um gráfico a duas dimensões constituído por nós que representam os pontos inicial, final e todos os vértices do espaço de configuração dos obstáculos, e por arestas que representam caminhos que unem os nós entre si. As arestas são segmentos de retas formadas entre dois nós que pertencem ao campo visível um do outro. Todos os nós e arestas estão definidos no espaço livre, pelo que o caminho determinado não intercetará obstáculos. Desta forma são produzidos os grafos que contêm as arestas que representam as menores distâncias entre os nós. De salientar que as arestas formadas pelos obstáculos também pertencem ao caminho que o robô poderá percorrer, pelo que, embora o método seja completo, possui a desvantagem de o robô caminhar muito próximo dos obstáculos podendo ocasionar colisões.

A Figura 2.1 ilustra um exemplo para o método descrito, onde está representado a tracejado a trajetória definida após a execução do algoritmo de pesquisa de grafos, onde se pode observar que o caminho determinado se aproxima muito dos obstáculos.

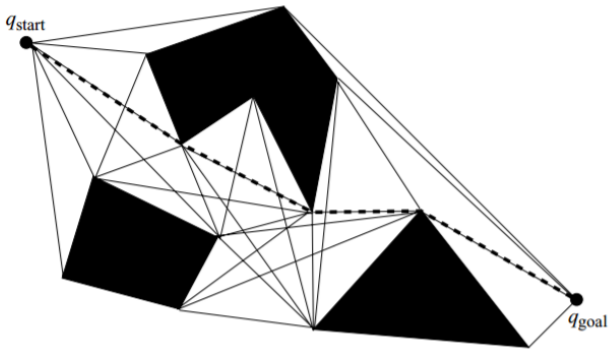


Figura 2.1: Exemplo do Algoritmo *Visibility Graph*[1]

Este método foi melhorado, retirando ao VG as arestas que nunca serão utilizadas. Para tal são usadas linhas de suporte que são tangentes a dois obstáculos, de tal modo que ambos os obstáculos estão do mesmo lado da linha, e linhas de separação que também são tangentes a dois obstáculos, mas os obstáculos estão de lados opostos da linha, tal como ilustra a Figura 2.2.

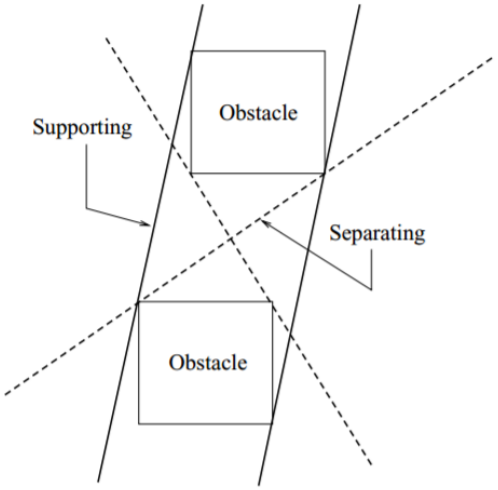


Figura 2.2: Linhas de suporte e de separação usadas para construção do *Visibility Graph*[1].

O *Reduced Visibility Graph* (RVG) é constituído a partir das linhas de suporte e de separação, pelo que todas as arestas do VG original que não são linhas de suporte ou de separação são removidas. Apesar de o caminho com menor distância não alterar, o tempo para o determinar diminui consideravelmente. A Figura 2.3 ilustra o gráfico obtido através do RVG.

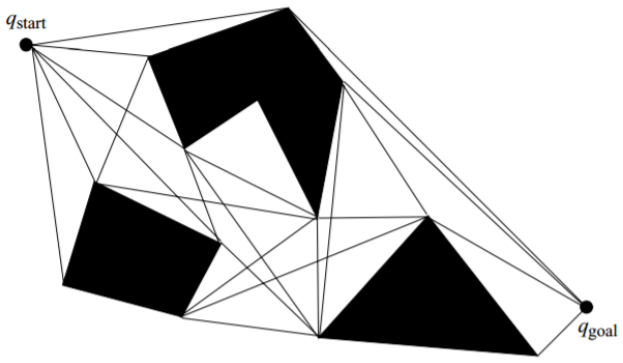


Figura 2.3: Exemplo do Algoritmo *Reduced visibility graph*[1].

2.1.1.2 Diagrama *Voronoi* (DV)

O diagrama *Voronoi* é constituído por um conjunto de pontos equidistantes entre dois obstáculos. O trajeto a definir pelo robô será do ponto inicial até ao DV, o melhor trajeto pertencente ao diagrama para alcançar o ponto do DV mais próximo de ponto destino e posteriormente deixará o DV e alcançará o ponto destino. Uma vez que o DV é definido em termos de distância, um robô equipado com sensores de distância pode construir incrementalmente o DV de um espaço desconhecido.

Uma característica a realçar deste método é que, como se pode observar na Figura 2.4, este maximiza a distância entre o robô e os obstáculos, pelo que dificilmente ocorrerá uma colisão entre os mesmos. No entanto, a trajetória definida, na maioria dos casos não se traduz no menor trajeto possível. Para além disto, o DV tem uma fraqueza considerável aquando da utilização de sensores de localização de alcance limitado. Como o algoritmo de planeamento de trajetória maximiza a distância entre o robô e os objetos no ambiente, os sensores de curto alcance no robô poderão não ter perceção do ambiente, pelo que, se estes forem usados para localização, esta terá um maior erro associado.

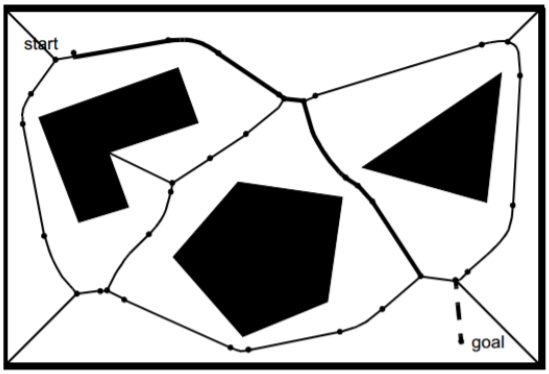


Figura 2.4: Exemplo do Algoritmo Voronoi Diagram [2]

2.1.1.3 *Roadmap* Probabilístico (PRM)

Nesta abordagem o roadmap é construido através de métodos probabilísticos. O planeamento de trajetórias é dividido em duas fases: a fase de aprendizagem, durante a qual é construido um roadmap no C_{livre}, e a fase de investigação na qual é feita a pesquisa de um trajeto desde o ponto inicial até ao ponto final, percorrendo o roadmap construido na fase anterior.

A fase de aprendizagem deste método consiste na distribuição aleatória de um conjunto de pontos, designados por nós, no espaço livre, entre os quais são estabelecidas ligações. Estas ligações são feitas através de linhas retas e apenas ocorrem entre nós vizinhos se as mesmas não intercetarem o $C_{obstáculo}$. A fase de investigação visa a ligação do ponto inicial e do ponto desejado ao roadmap construido na fase anterior através do C_{livre} e posterior determinação da trajetória a percorrer entre o nó inicial até ao destino, através do uso de algoritmos de pesquisa em grafos. A Figura 2.5 ilustra uma trajetória determinada pelo presente método.

No PRM o processamento é concentrado na fase de aprendizagem, de forma a tornar a fase de investigação mais rápida. Um dos problemas associados a este método é a dificuldade em encontrar uma passagem quando o C_{livre} é estreito.

Para melhorar o desempenho deste método e diminuir o tempo de execução aquando da pesquisa do caminho, foram desenvolvidas algumas melhorias tais como o *Rapidly-exploration Random Tree* (RRT) que difere do PRM no nó que guarda. Como mencionado anteriormente, o PRM

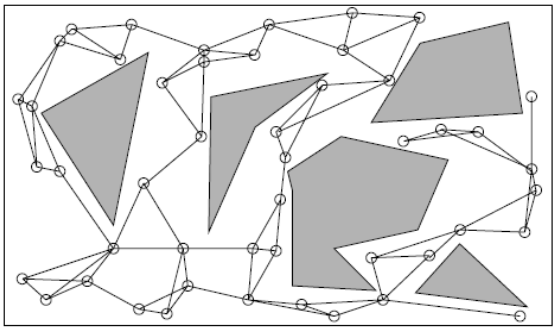


Figura 2.5: Exemplo do Algoritmo PRM [1].

guarda o nó gerado aleatoriamente, já no RRT o nó guardado é um nó que se situa na direção do nó gerado aleatoriamente e do nó já existente mais próximo, a uma distância previamente definida. Este método permite uma pesquisa mais rápida, na medida em que os nós ficam mais concentrados e não existem caminhos redundantes.

2.1.2 Decomposição em Células

Este método consiste na decomposição do $C_{espaço}$ em células, e posterior cálculo se as mesmas se encontram ocupadas ou livres de obstáculos. Inicialmente é determinada a célula que contém o ponto inicial e final, e posteriormente é efetuada a pesquisa de qual a sequência de células a percorrer para alcançar o ponto desejado.

A decomposição em células pode ser feita através de células exatas ou células aproximadas. Estas duas técnicas diferem na forma como representam o mundo real.

2.1.2.1 Decomposição em Células Exatas

A decomposição em células exatas apenas possui células totalmente livres ou totalmente ocupadas. Estas possuem formas geométricas simples, tais como polígonos convexos e trapézios. A decomposição em polígonos convexos forma células com esta geometria, onde os vértices das mesmas são os vértices dos obstáculos. Os vértices das células na decomposição em trapézios tem o mesmo significado físico, no entanto existem linhas na vertical associado a estes vértices que representam as arestas das células, tal como ilustra a Figura 2.6.

A desvantagem da decomposição em células exatas incide no número de células, que afeta a eficiência computacional que depende da complexidade dos objetos no ambiente. No entanto, em ambientes espaçosos, o número de células é pequeno e consequentemente a eficiência computacional não será afetada.

Esta técnica é uma boa escolha quando se pretende uma representação sem perdas [2].

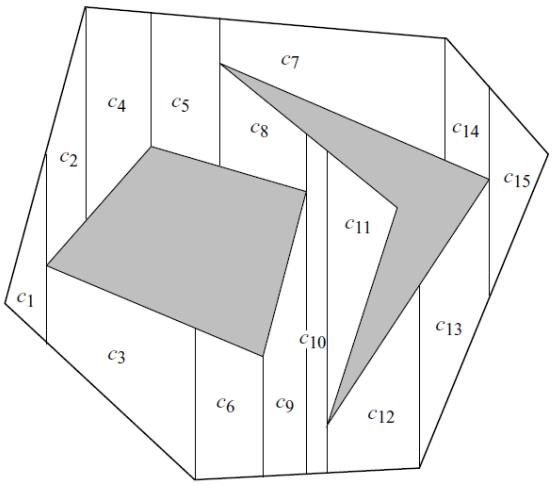


Figura 2.6: Exemplo da Decomposição em Células em Forma de Trapézio [1].

2.1.2.2 Decomposição em Células Aproximadas

Nesta decomposição cada célula poderá estar livre, ocupada ou parcialmente livre de obstáculos, e possuir formas geométricas simples, como quadrados, de forma a simplificar a implementação desta técnica.

Existem várias formas de implementar este método, sendo estas a decomposição em células fixas, onde todas as células possuem o mesmo tamanho, e decomposição em *Quadtree*, onde primeiramente é dividido o espaço em 4 células com o mesmo tamanho, e posteriormente as células que não se encontram no espaço livre são de novo dividas. Este processo é repetido sucessivamente até uma célula alcançar o limite mínimo de tamanho definido previamente. O algoritmo *Quadtree* requer uma menor capacidade computacional quando comparado com os restantes algoritmos de decomposição em células já apresentado anteriormente, no entanto a trajetória gerada pelo mesmo está restrita a segmentos formados entre os centros de células, o que na maioria dos casos não se traduz numa trajetória ótima.

Para resolver o problema acima identificado aquando da implementação do algoritmo *Quadtree*, foi desenvolvido em [3] uma melhoria do método denominada *Framed Quadtrees*, que consiste na adição de células de maior resolução nas proximidades do perímetro de cada região *quadtree*. Além disso, a trajetória pode ser definida entre duas células distantes. A Figura 2.7 ilustra trajetórias definidas através do algoritmo *Quadtree*, representada na imagem superior, e através do algoritmo *Framed Quadtree*, representada na imagem inferior. De notar que o uso do algoritmo *Framed Quadtree* permite a definição de uma trajetória mais próxima da trajetória ótima.

A decomposição em células aproximadas é uma das técnicas mais usadas no planeamento de trajetórias de robôs móveis. No entanto, este algoritmo nem sempre retorna uma trajetória mesmo que esta seja possível. Um dos fatores que influência o desempenho desta técnica é o tamanho da célula, sendo que um tamanho menor se traduz num melhor desempenho.

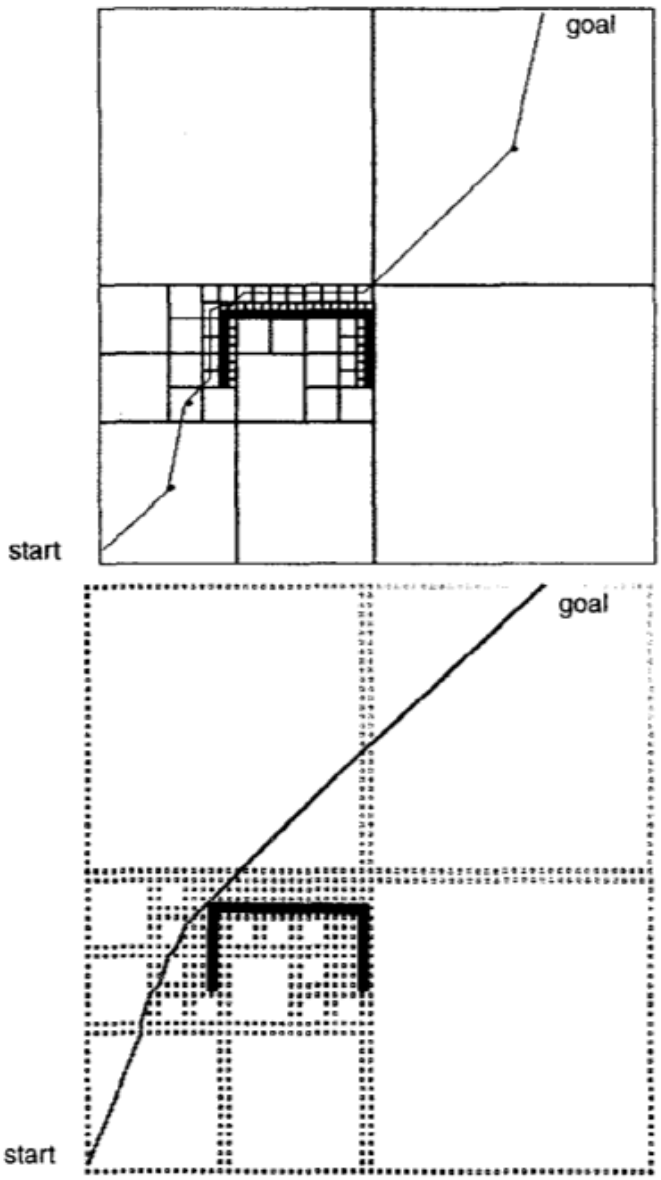


Figura 2.7: Exemplo de trajetórias definidas através do algoritmo *Quadtree* (a) e o algoritmo *Framed Quadtree* (b) [3]

2.1.3 Algoritmo Bug

O algoritmo Bug é o algoritmo de maior simplicidade em termos de implementação. Este é usado quando não se tem conhecimento do ambiente, baseando-se nos sensores usados para a sua localização, permitindo assim evitar obstáculos.

Esta técnica baseia-se no movimento do robô em linha reta e no seguimento de contornos dos obstáculos. Inicialmente o robô inicia o movimento da sua posição atual para a posição destino em linha reta até encontrar um obstáculo. Quando se depara com um obstáculo este contorna-o.

De seguida são abordados alguns destes algoritmos.

2.1.3.1 Bug1

No algoritmo Bug1 o robô quando se depara com um obstáculo contorna totalmente o mesmo e determina qual o ponto em que se encontra mais próximo do destino, dirigindo-se para esse ponto para seguir em linha reta para o ponto destino. Esta metodologia está representada na Figura 2.8.

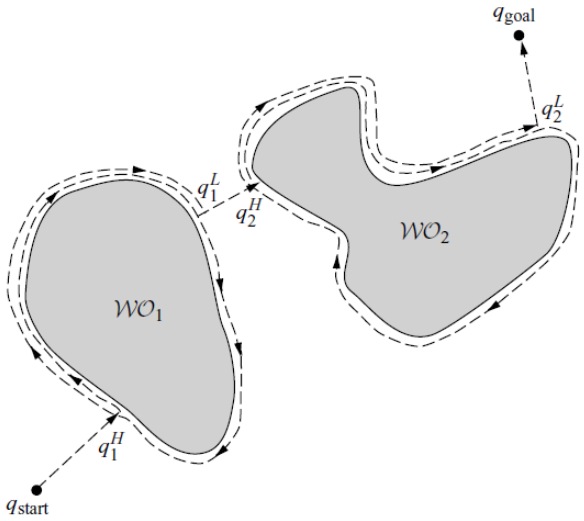


Figura 2.8: Exemplo do Algoritmo Bug1 [1]

Esta técnica garante que o robô atinge o ponto desejado desde que o mesmo seja alcançável, no entanto é muito ineficiente.

2.1.3.2 Bug2

Como se pode observar pela Figura 2.9, aquando do contorno do obstáculo, este método permite ao robô partir imediatamente em linha reta quando encontra de novo o segmento de reta.

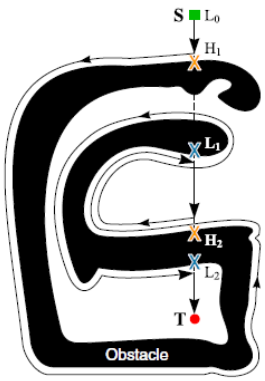


Figura 2.9: Exemplo do Algoritmo Bug2. Ponto inicial S e ponto destino T [4].

Comparando a Figura 2.9 com a Figura 2.8 é notória a melhoria no desempenho, no entanto este método não é ótimo.

2.1.3.3 Bug2+

O algoritmo *Bug2+* é uma melhoria do algoritmo *Bug2*. Estes diferem no momento de abandonar o contorno do obstáculo, sendo que no algoritmo Bug2+ o robô só abandona o contorno do obstáculo para seguir o segmento de reta se o mesmo não tiver passado por um ponto mais próximo do ponto destino.

A Figura 2.10 ilustra o caminho percorrido pelo robô através do algoritmo Bug2+. A configuração do espaço, o ponto inicial e o ponto destino são os mesmo que os apresentados no algoritmo Bug2.

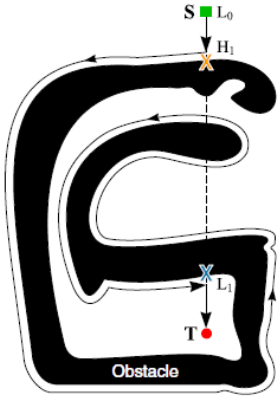


Figura 2.10: Exemplo do Algoritmo Bug2+. Ponto inicial S e ponto destino T [4].

2.1.3.4 Tangent Bug

O uso do algoritmo Tangente Bug permite ao robô comportar-se da seguinte forma: o robô inicia o seu movimento em linha reta para o ponto destino, até que este deteta um obstáculo no raio de alcance do seu sensor. Nesta situação, o circulo com o raio do alcance do sensor torna-se tangente ao obstáculo. Este ponto tangente divide-se em dois pontos, que são os pontos finais do intervalo. Se o intervalo interceta o segmento que conecta o robô e o ponto destino, este move-se para o ponto final que diminui a distância ao ponto destino. Esta metodologia ocorre sucessivamente até atingir o ponto destino. A Figura 2.11 ilustra o comportamento de um robô com a aplicação deste método.

Tangent Bug determina um caminho mais curto para a ponto destino, do que os métodos bug mencionados anteriormente. De realçar que quanto maior o alcance do sensor melhor será o desempenho do algoritmo, podendo mesmo este aproximar-se de trajetos ótimos aquando da presença de ambientes simples.

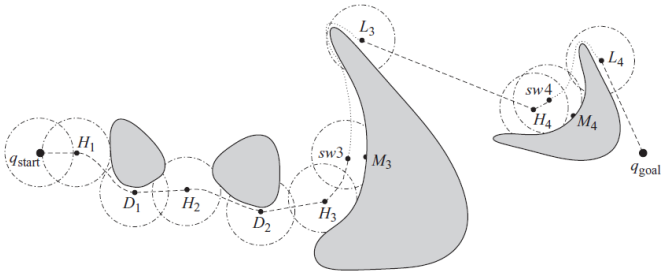


Figura 2.11: Exemplo do Algoritmo Tangent Bug [1].

2.1.4 Campo Potencial

A utilização de campos potencias no planeamento de trajetórias baseia-se no uso dos potenciais elétricos da física como heurística para encontrar a trajetória. Este método consiste na representação do robô através de um ponto com carga positiva no $C_{espaço}$, que se desloca num espaço com obstáculos sob a influência de um campo potencial artificial. O ponto destino comporta-se como uma partícula de carga negativa e os obstáculos comportam-se como partículas de carga positiva, atuando assim como forças atrativas e repulsivas respetivamente. O campo potencial artificial corresponde ao somatório de todos os campos formados pelo destino, sendo este atrativo, e pelos obstáculos, que geram campo repulsivo. Desta forma o robô será guiado até ao ponto destino evitando obstáculos previamente conhecidos. Se o robô estiver inserido num ambiente dinâmico, é possível a atualização do campo potencial de modo a integrar novas informações à cerca do mapa.

Sendo q uma posição no mapa, o campo potencial $U(q)$ que atua no robô resulta do somatório do campo atrativo $U_{att}(q)$ gerado pelo ponto destino e do campo repulsivo gerado pelos obstáculos $U_{rep}(q)$.

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{2.1}$$

Este potencial atrativo é diferenciável, pelo que a força $F(q)$ que atua no robô é o gradiente.

$$F(q) = -\bigtriangledown U(q) = -\bigtriangledown U_{att}(q) - \bigtriangledown U_{rep}(q) \tag{2.2}$$

Um potencial atrativo pode ser definido como uma função parabólica,

$$U_{att}(q) = \frac{1}{2} k_{att} \parallel q - q_{goal} \parallel^2 \tag{2.3}$$

onde k_{att} é um fator escalar positivo, q é a posição atual do robô e q_{goal} é a posição desejada. Desta forma a força atrativa $F_{att}(q)$ pode ser definida através da equação 2.4.

$$F_{att}(q) = -k_{att} \times (q - q_{goal}) \tag{2.4}$$

O potencial repulsivo deve ser muito forte quando o robô está perto do obstáculos, mas não deve influenciar o seu movimento quando o robô se encontra distante do obstáculo. Esta relação

pode ser representada pela seguinte equação, onde k_{rep} é um fator de escala, $\varphi(q)$ é a mínima distância entre o a posição do robô q e o obstáculo e φ_0 é a distância máxima a que o obstáculo pode influenciar no movimento do robô.

$$U_{rep}(q) = \begin{cases} \frac{1}{2} \times k_{rep}(\frac{1}{\varphi(q)} - \frac{1}{\varphi_0}) & \text{if } \varphi(q) \leq \varphi_0 \\ 0 & \text{if } \varphi(q) \geq \varphi_0 \end{cases}$$

Se o limite do obstáculo é convexo e as partes constituintes da função são diferenciáveis, $\varphi(q)$ é diferenciável em qualquer C_{livre}, pelo que a força repulsiva é calculada pela seguinte equação:

$$F_{rep}(q) = \begin{cases} \frac{1}{2} \times k_{rep}(\frac{1}{\varphi(q)} - \frac{1}{\varphi_0}) \frac{1}{\varphi(q)^2} \frac{q - q_{obstacle}}{\varphi(q)} & \text{if } \varphi(q) \leq \varphi_0 \\ 0 & \text{if } \varphi(q) \geq \varphi_0 \end{cases}$$

A força resultante, determinada pela equação 2.5 que atua sobre o robô exposto às forças atraentes e repulsivas move o robô para longe dos obstáculos e em direção à posição desejada.

$$F(q) = F_{att}(q) + F_{rep}(q) \tag{2.5}$$

De notar que o campo resultante neste método é também uma lei de controlo para o robô. Assumindo que o robô sabe a sua localização em relação ao mapa e ao campo potencial, pode-se determinar qual a próxima ação necessária com base no campo potencial. Em condições ideais, ao configurar o vetor velocidade proporcional à força de campo, o robô movimentar-se-á suavemente na direção da posição desejada.

No entanto, este método tem algumas limitações tais como os mínimos locais, que quando ocorrem impedem o robô de chegar ao destino. Isto ocorre quando o somatório de todas as forças resultantes se anulam, mesmo quando o robô não está ainda no destino, tal como ilustra a Figura 2.12. Outro problema que pode ocorrer aquando da utilização deste método, é a existência de várias distâncias mínimas que resultam em oscilações entre os pontos mais próximos do destino. Esta limitação ocorre quando o robô está perante obstáculos côncavos [2].

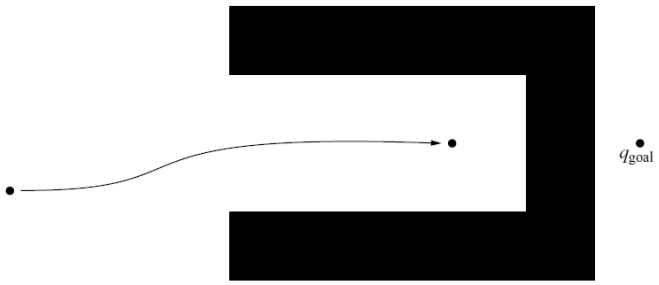


Figura 2.12: Exemplo do problema de mínimo local [1].

O método de campo potencial estendido visa um melhor acompanhamento de parede, e desta forma obtém uma trajetória que permite ao robô atingir o destino percorrendo uma menor distância.

Esta abordagem, tal como no método de campo potencial, utiliza forças atrativas e repulsivas que têm como origem o campo potencial artificial. No entanto são adicionados ao campo potencial o campo potencial de rotação e o campo potencial de tarefa. O campo potencial de rotação infere que a força repulsiva é uma função da distância do obstáculo e da orientação com o que o robô se desloca em relação ao obstáculo. Para tal recorre-se a um fator de ganho que reduz a força de repulsão quando um obstáculo se encontra paralelamente à direção de deslocamento do robô. O campo potencial de tarefa considera a velocidade atual do robô, e através da mesma filtra os obstáculos que não devem afetar o potencial de curto prazo.

A Figura 2.13 relaciona o desempenho do campo potencial clássico e do campo potencial estendido. Verifica-se uma significante melhoria na eficácia da trajetória.

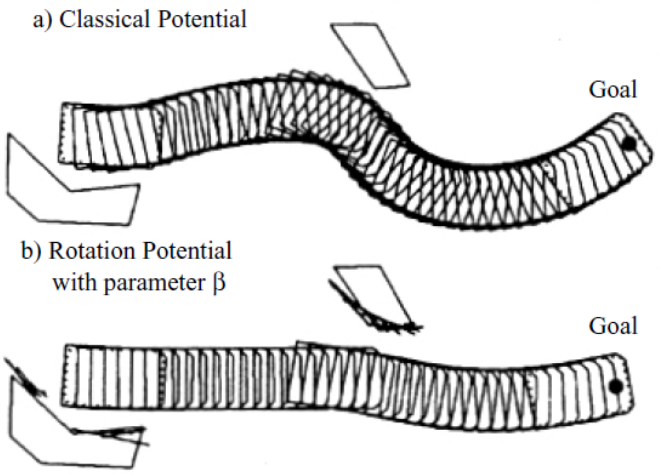


Figura 2.13: Comparação entre o campo potencial clássico e o campo potencial estendido [2].

2.1.4.1 *Fast Marching*

O método *Fast Marching* (FM) surge com a necessidade da existência de um único mínimo local possível, sendo este o ponto destino [8]. A construção do campo potencial assemelha-se à expansão de uma onda na água, sendo construida uma onda artificial que não é circular devido à presença de obstáculos. Este método permite gerar o caminho mais curto, criando uma trajetória que tem em atenção a velocidade com que a onda artificial percorreu cada ponto.

Os resultados produzidos pelo método original não são os ideais devido à proximidade da trajetória final com os obstáculos e ao facto de a velocidade ser muito baixa nos pontos próximos a espaços do mapa ocupados [8]. De forma a contornar estes resultados, foram criadas algumas variantes do FM, tal como a sua utilização em conjunto com diagramas de *Voronoi* [8]. Tal como referido anteriormente, os diagramas de *Voronoi* permitem criar um conjunto de pontos equidistantes em relação aos obstáculos. A partir dessa representação do mapa, o método FM pode ser usado para criar campos potenciais até ao ponto final. Esta variante reduz consideravelmente o tempo necessário para encontrar uma trajetória.

Outra variante do método original é o *Fast Marching Square* (FM2) [8]. Este método, em vez de criar uma onda artificial a partir do ponto inicial, cria um conjunto de ondas artificiais que têm como origem os obstáculos. O método FM2 permite obter uma trajetória o mais afastada possível dos obstáculos, sendo o resultado final é semelhante ao obtido utilizando o FM com diagramas de Voronoi. No entanto, quando não se pretende obter uma trajetória tão afastada dos obstáculos, pode ser utilizada uma outra versão do método: o FM2 com saturação. Este consiste na alteração do mapa resultante do FM2 de acordo com a distância máxima de segurança permitida em relação aos obstáculos e com a velocidade máxima permitida.

O *Fast Marching Square Star* (FM2*) é uma outra variante que utiliza custos heurísticos para orientar a procura de um caminho ao ponto final [8]. Ao contrário das variantes anteriores em que a onda artificial se expande em todas as direções da mesma forma, o FM2* reduz o número de pontos expandidos pois o ponto final é tido em consideração e, desta forma, os pontos mais próximos do ponto final são percorridos em primeiro lugar. Este método reduz até quatro vezes o tempo de computação de um caminho em relação ao FM2, produzindo um caminho final semelhante.

2.1.5 *Velocity Obstacle*

Este método consiste em determinar um conjunto de velocidades que resultam em colisões com obstáculos, tendo em consideração as velocidades dos mesmos, assumindo que estas se mantêm. Desta forma deve ser determinada uma velocidade que não permita a colisão do robô [9].

Existem algumas variantes deste método, tais como *ellipse-based VO* (EBVO) [10]. Esta abordagem difere da original na forma como são representados o robô e os obstáculos presentes no ambiente. No método VO o robô e os obstáculos são representados em círculos, sendo assim difícil considerar movimentos rotativos em robôs holonômicos. Esta dificuldade é superada substituindo a representação do robô e dos obstáculos por uma elipse. Resultados apresentados em [10] mostram que este dá origem a uma trajetória mais curta e o tempo que o robô demora a percorre-la é menor que o método original, no entanto o tempo computacional é maior.

2.1.6 Outras abordagens

Muitos outros métodos são usados para planeamento de trajetórias. Foram implementados no planeamento de trajetórias conceitos como lógica difusa [11], redes neuronais [12], algoritmos genéticos [13] e *Ant Colony Algorithm* [14].

Em [5] foi desenvolvido um novo método de planeamento de trajetórias probabilística, que consiste na sobreposição de subconjuntos convexos, denominado bolhas no C_{livre}. O caminho da posição inicial para a posição final desejada é definido através de uma árvore de pesquisa construída por propagação aleatória das bolhas no C_{livre}. A Figura 2.14 ilustra uma trajetória definida por este método. O método não é completo, pois nem sempre encontra uma trajetória mesma que esta exista, e o mapa não é completamente coberto pelas bolhas.

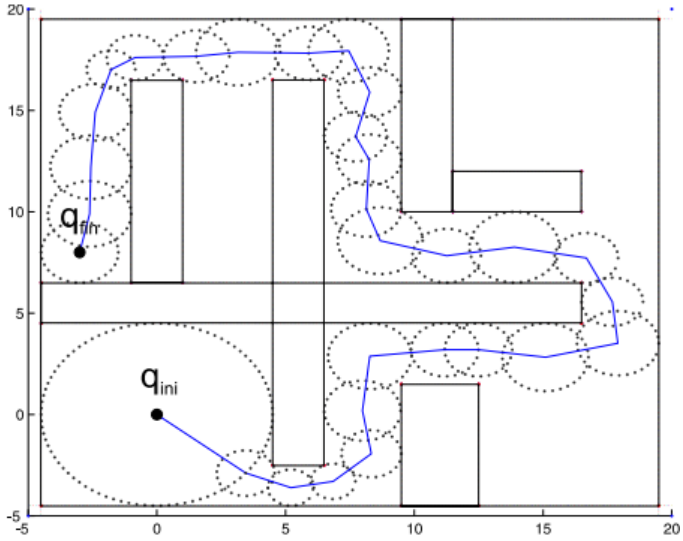


Figura 2.14: Exemplo de uma trajetória definida pelo método apresentado em [5].

2.2 Algoritmos de Pesquisa

Na secção anterior foram abordados alguns métodos de planeamentos de trajetórias. Nas abordagens já mencionadas, roadmap e decomposição em células, são gerados grafos que dão origem a trajetórias a percorrer pelo robô recorrendo a algoritmos de pesquisa de grafos. Assim, nesta secção serão apresentados alguns algoritmos de pesquisa.

Estes podem ser divididos em algoritmos sem heurística, que são algoritmos que não possuem qualquer informação que lhes permita obter a solução mais rapidamente, sendo feita uma pesquisa exaustiva, e algoritmos com heurística, onde são usadas algumas informações para escolher a sequência de pesquisa, e desta forma determinar a trajetória com um menor tempo de execução.

De forma a verificar a eficácia dos algoritmos é mencionado em cada um deles se os mesmos são completos, encontrando sempre a solução caso esta exista, e se são ótimos, sendo que a solução encontrada corresponde à solução de menor custo.

O algoritmo de **pesquisa por profundidade** é um algoritmo sem heurística, onde é feita uma pesquisa exaustiva de forma a explorar os nós o mais longe possível sem retroceder. O algoritmo apenas retrocede para o nó anterior se encontrar um nó sem ligações e o mesmo não for o destino desejado. Este algoritmo é completo, no entanto não é ótimo, uma vez que o algoritmo dá por

detetada a trajetória quando o nó que coincide com o ponto destino é encontrado, podendo este não ser o trajeto mais curto.

O algoritmo de **pesquisa por largura** é um algoritmo sem heurística, onde são exploradas todas as ligações de um nó antes de se seguir para outro nó. Este algoritmo encontra sempre a solução caso exista, no entanto não é ótimo, exceto quando todas as ligações têm tamanhos iguais.

No método de **pesquisa por aprofundamento limitado**, a abordagem é semelhante à pesquisa por profundidade, diferindo na medida que neste é imposto um limite à profundidade a que se pode pesquisar. O algoritmo é completo se o nó que corresponde ao destino se encontra dentro do limite imposto, no entanto este também não é ótimo.

Na **pesquisa por aprofundamento iterativo**, a abordagem é semelhante à pesquisa por aprofundamento limitado. É executado a pesquisa por aprofundamento limitado sucessivamente, aumentando o limite imposto até a solução ser encontrada. É um algoritmo completo, no entanto não é ótimo.

O algoritmo **Dijkstra's** é um algoritmo com heurística. São explorados todos os nós ligados ao nó atual, e segue para o nó que apresenta uma ligação com menor custo. Isto processa-se sucessivamente até que a solução é encontrada [15]. É um algoritmo completo e ótimo.

O **algoritmo do tipo guloso** é um algoritmo com heurística. Em cada nó é determinado o nó que está mais próximo do destino, sendo este o nó a ser explorado a seguir. É um algoritmo completo e não ótimo.

O **algoritmo A-*star* (A$\star$)** é um algoritmo com heurística, pois este tem em conta qual o nó que está mais perto do nó atual e qual o nó que se encontra mais próximo do destino. À medida que A* percorre o mapa, este segue o caminho com o custo mais baixo enquanto mantém nós alternativos em fila de prioridades classificadas. Se o nó a ser percorrido tiver um custo maior do que outro nó encontrado em qualquer ponto, o nó com o custo mais alto será descartado e percorrerá o nó de custo mais baixo. Esse processo continua até que a meta seja alcançada [16]. Desta forma o algoritmo explora primeiro os nós que considera mais promissores.

O A$\star$ é um algoritmo completo e ótimo, no entanto quando existem alterações no ambiente e o caminho para atingir o ponto final tem de ser reajustado, este executa de novo o processo a partir do inicio, tornando assim o tempo de processamento demasiado elevado. De forma a reduzir o tempo de processamento foram desenvolvidos algoritmos que permitem um reajuste da trajetória, sem replanear a trajetória desde o ponto inicial. Surgem então algoritmos que permitem reajustar a trajetória sem a necessidade de iniciar todo o processo, tal como o algoritmo D* tendo o nome surgido a partir do termo "*Dynamic A**"[17]. Este algoritmo é normalmente utilizado em ambientes dinâmicos, em situações que o robô encontra um obstáculo e tem de rapidamente replanear o seu caminho. A utilização do D* é normalmente mais eficiente do que a utilização de várias pesquisas A* consecutivas, principalmente nos casos em que o ponto final não muda. A principal diferença em relação ao A* é o facto de, quando ocorre um alteração no ambiente, apenas os nós situados na zona onde surgem as alterações são revisitados. Dessa forma, a trajetória não é totalmente reconstruida de cada vez que existe uma necessidade de alterar a mesma.

2.3 Controladores de Trajetórias

Os controladores de trajetórias visam determinar a velocidade do robô para o mesmo seguir a trajetória desejada. No caso particular de robôs omnidirecionais, o controlador determinará a velocidade linear, normal e angular que o robô deverá ter para percorrer a trajetória.

Nesta secção serão abordados controladores normalmente aplicados a robôs móveis omnidirecionas.

2.3.1 Controladores PID

O controlador *proportional–integral–derivative* (PID) é um controlador em malha fechada muito utilizado em sistemas de controlo industrial.

Este tipo de controladores consiste no calculo do erro $e(t)$ associado ao seguimento da trajetória, sendo este a diferença entre a localização desejada e a localização real do robô, e posterior correção baseada nos termos proporcionais, integrais e derivativos, denotados P, I e D respetivamente.

É possível a utilização de apenas alguns dos modos, tendo como controladores convencionais P, PI, PD e PID.

O controlo proporcional consiste essencialmente num amplificador com ganho ajustável que atua sobre o erro. O aumento de k_p permite minimizar o erro estacionário, no entanto este provoca um aumento do tempo de estabelecimento. De forma a limitar o produto entre o k_p e o erro da posição, pode ser usado a tangente hiperbólica que tornará o controlador mais robusto [6].

O termo integrador (I), permite eliminar o erro de posição em regime permanente para valores de referencia constantes, no entanto esta também provoca um aumento no tempo de estabelecimento e tende a piorar a estabilidade do sistema. A aplicação desta ação em controladores de trajetórias permite que o robô se movimente de uma forma suave, sem reações abruptas.

O termo derivativo (D), é responsável pela diminuição da sobre-elevação e do tempo de estabelecimento, melhorando a estabilidade. Quando aplicado em controladores de trajetórias, o robô tende a reagir mais rapidamente, tendo um movimento mais abrupto.

Sendo $u(t)$ a saída do controlador, $e(t)$ o erro de posição, T_i o tempo integral e T_d o tempo derivativo, o controlador PID é descrito pela seguinte equação.

$$u(t) = k_p(e(t) + \frac{1}{T_i}\int e(\Gamma)\mathrm{d}\Gamma + T_d\frac{\mathrm{d}e(t)}{\mathrm{d}t}) \tag{2.6}$$

Em [18], erro de posição é convertido para um vetor de erro nas coordenadas do ângulo da roda através da cinemática inversa, ao qual é aplicado um controlador.

Controladores baseados no método de controlo linear necessitam de ajustar o ganho de forma a atribuir diferentes velocidades para diferentes tipos de trajetórias. Para o primeiro e o último intervalo deve ser adicionado o tempo necessário para acelerar e desacelerar o veículo, respetivamente. A velocidade do movimento não deve exceder o valor máximo. A velocidade de rotação

do veículo deve ser considerada, pois esta não deve exceder a velocidade de rotação máxima admissível. No caso de a velocidade de rotação exceder o seu valor máximo, a velocidade máxima admitida ao movimento do robô deve diminuir.

2.3.2 Controladores Feedforward

Os controladores Feedforward consistem na ação antecipada, ou seja, este prevê as saídas do controlador desejadas. A Figura 2.15 ilustra um controlador Feedforward PD utilizado para controlo de trajetórias de um robô omnidirecional.

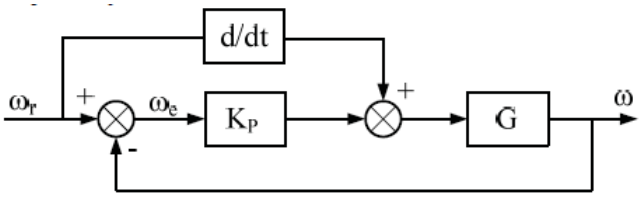


Figura 2.15: Exemplo de um controlador Feedforward PD [6].

O controlador apresentado na Figura 2.15 prevê a saída do controlador através da derivada da posição desejada, sendo w_r e w o vetor da posição desejada e real do robô, respetivamente. A saída do controlador é multiplicada pela cinemática inversa do robô de forma a obter o erro de velocidade associado a cada roda.

Uma outra forma de calcular a saída prevista é através da cinemática inversa do robô [19, 20]. Em [20] a saída do controlador correspondente à velocidade linear, normal e angular do robô é determinada através do mínimo entre as velocidades obtidas através da cinemática inversa e as velocidades obtidas através de uma lei de controlo, sendo neste artigo referido a linearização da dinâmica do erro de seguimento ao longo da posição desejada. Este método de controlo é referido em seguida.

Controladores *Feedforward* permitem uma correção rápida quando a referência é alterada, o que se traduz num menor erro da posição associado.

2.3.3 *Trajectory Linearization Control* (TLC)

O TLC é um controlador não linear, baseado na linearização ao longo de uma trajetória nominal que permite o seguimento de qualquer trajetória. Este fornece estabilidade ao longo da trajetória sem interpolação dos ganhos do controlador [19].

Neste controlador, é feita uma linearização do modelo cinemático do robô, ao longo das trajetórias nominais desejadas e das velocidades linear, normal e angular do robô desejadas, calculadas através da cinemática inversa. Assim é determinada a dinâmica do erro associada à posição do robô.

Sendo $f(x,u)$ o modelo cinemático do robô, $\widehat{x}$ e $\widehat{u}$ a posição e velocidades desejadas respetivamente, a linearização é dada pela seguinte equação.

$$\triangle \dot{x} = \frac{\partial f}{\partial x}(\widehat{x},\widehat{u}) \triangle x + \frac{\partial f}{\partial u}(\widehat{x},\widehat{u}) \triangle u \tag{2.7}$$

Posteriormente o erro de seguimento é estabilizado através de uma lei de controlo linear. Em [19] foi usado um controlador PI com realimentação para a estabilização do erro. Os coeficientes associado ao controlador PI estão relacionados com o polinómio característico da equação de estado do erro que se pretende obter. A saída deste controlador é a soma das velocidades desejadas calculadas a partir da cinemática inversa com o erro a estas associado.

Com o TLC, o robô pode ser controlado para seguir trajetórias diferentes com precisão, sem alterar os ganhos do controlador.

2.3.4 Sliding Mode Control

O controlo de modo deslizante é um método de controlo não linear que altera a dinâmica de um sistema não linear através da aplicação de um sinal de controlo descontínuo permitindo ao sistema permanecer ao longo de uma superfície deslizante.

Este método de controlo é dividido em duas fases. A primeira fase consiste na definição da superfície deslizante. Esta função, é frequentemente definida em função do erro de seguimento assim como das suas derivadas. A superfície de deslizamento deve ser definida de tal forma que quando esta se anula dê origem a uma equação diferencial estável. Usualmente, o coletor deslizante é definido como uma combinação linear do seguinte tipo:

$$s = e^k + \sum_{i=0}^{N} c_i e^i \tag{2.8}$$

Sendo s a função de deslizamento, e o erro de seguimento e k o número de derivadas a ser incluídas na função. O coeficiente k deve ser $k = r - 1$ onde r é o grau relativo entre a entrada e saída do sistema.

A segunda fase consiste em definir uma ação de controlo que direciona o sistema para a superfície deslizante. Neste contexto existem várias abordagens baseadas em controlo de modo deslizante de primeira ordem e de ordem maior.

O controlo de modo deslizante de primeira ordem consiste num controlo descontínuo onde $u = -Usgn(s)$, sendo u a entrada de controlo. Desta forma, a variável de controlo u comuta a uma frequência elevada entre os valores $u = U$ e $u = -U$, o que origina oscilações. Para resolver este fenómeno de vibração, é possível implementações aproximadas, onde o termo descontinuo $sign(s)$ é substituído por uma aproximação contínua [21, 22]. Seguem-se duas possíveis aproximações, onde $\mathscr{E}$ é positivo e aproximadamente zero.

$$u = -Usat(s;\mathscr{E}) = -U\frac{s}{|s| + \mathscr{E}} \tag{2.9}$$

$$u = -Utanh(\frac{s}{\mathscr{E}}) \tag{2.10}$$

Com $\mathscr{E}$ pequeno o efeito de suavização na entrada do controlo é limitada, mas é mantida a precisão de controlo, enquanto que com um $\mathscr{E}$ maior é notável a suavização mas existe uma perda de precisão. Desta forma deve ser encontrada uma boa relação entre a suavização da entrada e a perda de precisão. Esta abordagem apenas é eficaz em casos específicos, e embora alguns problemas sejam atenuados, existe perda de robustez.

Uma forma alternativa para resolver o fenómeno de vibração é a utilização de algoritmos de controlo de modo deslizante de segunda ordem. Com esta abordagem o fenómeno de vibração é completamente resolvido sem perda de robustez, uma vez que a lei de controlo é uma função contínua no tempo.

Em [21] foi implementado um controlo de modo deslizante integral de primeira ordem para controlo de seguimento de trajetórias de um robô móvel omnidirecional de três rodas. A função de deslizamento foi definida pela equação 2.11, onde $s_0(x)$ é uma combinação linear dos estados do sistema dinâmico do erro e a s_z é uma função integral que deve ser definida de forma a que os estados do sistema permaneçam na superfície deslizante.

$$s = s_0(x) + z_s \tag{2.11}$$

A entrada de controlo u é definida como a soma de uma entrada de controlo ideal u_0 obtida através da lei de controlo linear PID, e uma entrada de controlo descontínua que permitirá restringir os estados à superfície deslizante. u_1 é defina como uma aproximação já mencionada anteriormente 2.9 , onde foi usado $sat(s)$ para que não ocorra o fenómeno de vibração.

A principal questão na conceção destes controladores é a estimativa dos parâmetros a estes associados. Para resolver este problema foram implementados controlos de modo deslizante adaptativos a robôs omnidirecionais [23, 24], que consiste na introdução de uma lei adaptativa para estimar os valores desses parâmetros.

Resultados experimentais demonstrados em [21, 23] afirmam que o método de controlo de modo deslizante apresenta melhores resultados comparado com controladores PID aplicados ao controlo do mesmo sistema, assim como o controlo de modo deslizante adaptativo apresenta maior precisão do que o controlo de modo deslizante convencional.

2.3.5 Outras abordagens

Muitos outros métodos são usados para o controlo das trajetórias a serem percorridas pelo robô. Para além das técnicas já mencionadas, foram implementados no controlo de trajetórias conceitos como lógica difusa [25, 26], que permite um ajuste dos parâmetros do controlador em tempo real e métodos baseados no comportamento de colónia de formigas [27].

É usada frequentemente uma abordagem em que o caminho predeterminado é parametrizado usando segmentos de reta e arcos de circunferências. O controlo é realizado por um controlador híbrido, incluindo os algoritmos do movimento em linha reta, ao longo de uma circunferência e

a lógica de comutação. O uso de caminhos mais elementares fornece uma parametrização mais flexível da trajetória, o que melhora o desempenho [28]. De forma a definir a lógica de comutação foram desenvolvidos alguns algoritmos. Nesta secção serão dados apenas alguns exemplos baseados em [28].

O algoritmo de comutação definido pela equação 2.12 é um algoritmo simplicista, onde (x,y) corresponde à posição do robô, (x_{wp},y_{wp}) são as coordenadas do ponto de interseção de duas retas e $\mathscr{E}$ representa a área de comutação.

$$((x-x_{wp})^2+(y-y_{wp})^2)<\mathscr{E} \tag{2.12}$$

Um outro método de algoritmo de comutação consiste em ligar os pontos em linha reta e posteriormente introduzir áreas de comutação especificas usando circunferências de raio fixo de forma a suavizar a comutação. No entanto, o movimento real do robô entre uma linha reta e uma curva circular contém a espiral de *Cornus*, onde ocorre um decrescimento linear do raio de curvatura com o caminho percorrido ao longo do seu desenvolvimento, proporcionando assim uma variação gradual da curvatura. De forma a reduzir o erro entre a transição de uma trajetória em linha reta para uma trajetória circular, deve-se selecionar uma curvatura menor, ou como alternativa, recorrer a curvas polinomiais para os segmentos de transição, proporcionando assim comutações suaves.

2.4 Discussão do estado de arte

Os algoritmos de planeamento de trajetórias que necessitam da representação do ambiente em que o robô está inserido permitem, na maioria dos casos, obter trajetórias próximas do ideal. Um exemplo disso é a utilização de algoritmos de pesquisa a partir de métodos como *roadmap* e decomposição em células. Os métodos probabilísticos, têm como vantagem um menor período de tempo de processamento necessário, no entanto, na presença de passagens estreitas, o caminho pode não ser encontrado.

O *Visibility Graph*, método de construção do *roadmap* dá origem a trajetórias onde o robô tem de se movimentar muito próximo dos obstáculos, aumentando assim o risco de colisão. Já o Diagrama *Voronoi* maximiza a distância entre o robô e os obstáculos, pelo que, na maioria dos casos não se traduz no menor trajeto possível.

Os métodos de decomposição em células apresentam como vantagem a sua simplicidade de implementação e a fácil atualização do mapa em caso de alterações no ambiente. A principal desvantagem incide no número de células, que afeta a eficiência computacional que depende da complexidade dos objetos no ambiente. Existem vários algoritmos de pesquisa para gerar caminhos a partir dessa representação do ambiente. O algoritmo de *Dijkstra* permite encontrar o caminho mais curto entre dois pontos, mas não é orientado à célula final e por isso necessita de um maior período de tempo de processamento em relação a outros algoritmos, tal como o algoritmo A* que

utiliza custos heurísticos para estimar a distância à célula final e desta forma determina o caminho mais rapidamente. É um dos algoritmos de pesquisa de grafos mais utilizados devido à sua simplicidade e eficiência.

Devido às vantagens e desvantagens enunciadas anteriormente dos vários métodos de planeamento de trajetórias e de pesquisa de grafos, serão analisados nesta dissertação com maior detalhe a decomposição em células aproximadas, nomeadamente decomposição em células fixas, tendo estas a forma de quadrados, o método *Quadtree*, o método *Framed Quadtree* e ainda será abordado um novo método que tem como procedimentos base os métodos *Quadtree* e *Framed Quadtree*. De forma a encontrar um caminho a partir dessas representações do ambiente será usado como algoritmo de pesquisa de garfos o A*.

De forma a controlar o movimento do robô para o mesmo percorrer a trajetória desejada, será abordado um controlador denominado *Trajectory Linearization Control* (TLC), que consiste na linearização do modelo cinemático ao longo de uma trajetória nominal e na estabilização do erro de seguimento através de uma lei de controlo PI, o que proporcionará um movimento mais suavizado do robô. A escolha do controlador incide no fato de este permitir um ajuste dos ganhos do controlador para diferentes trajetórias.

Capítulo 3

Métodos de Planeamento de Trajetórias Implementados

Neste capítulo são apresentados os algoritmos de planeamento de trajetórias centrais ao cumprimento dos objetivos propostos nesta dissertação. Tal como já foi previamente referido o planeamento de trajetórias a idealizar tem de ser capaz de gerar percursos ótimos ou perto disso. Assim neste capítulo procede-se à apresentação das metodologias adotadas, referidas no capítulo anterior, começando pela descrição da expansão dos obstáculos no mapa. Nas secções 3.2.1, 3.2.2, 3.2.3 e 3.3 são apresentados a decomposição em células fixas, *Quadtrees*, *Framed Quadtrees* e o *k-Framed Quadtrees*, respetivamente, sendo este último o método sugerido nesta dissertação. Por fim é abordado o algoritmo de pesquisa A-*star* (A $\star$), onde é feita uma descrição da teoria associada ao algoritmo e de como este é aplicado às várias representações do mapa.

3.1 Expansão dos obstáculos

A expansão da área dos obstáculos permite considerar o robô como um ponto no espaço. De forma a garantir que a trajetória permita uma navegação do robô livre de colisões, é necessária a expansão dos obstáculos com as dimensões do robô ao qual será aplicado o algoritmo de planeamento de trajetórias.

No caso de o robô ser circular, os obstáculos sofrem uma expansão de raio igual ao raio do robô. No caso do mesmo ter uma geometria diferente, os obstáculos são expandidos com a forma geométrica do robô dependendo da sua orientação. Assim, são necessários vários mapas de expansão com as diferentes possíveis orientações do robô, o que por vezes afeta a eficiência computacional, tornando os algoritmos de planeamento de trajetória mais lentos. De forma a contornar esta situação, é feita uma aproximação da geometria do robô a um circulo de raio igual à maior distância do centro do robô à extremidade.

O robô omnidirecional presente nesta dissertação não é circular, pelo que foi feita a aproximação, tal como ilustra a Figura 3.1.

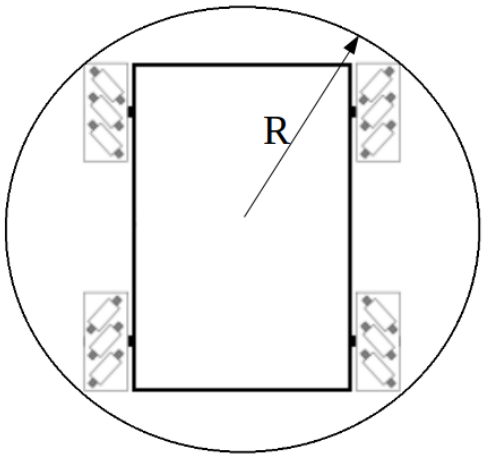
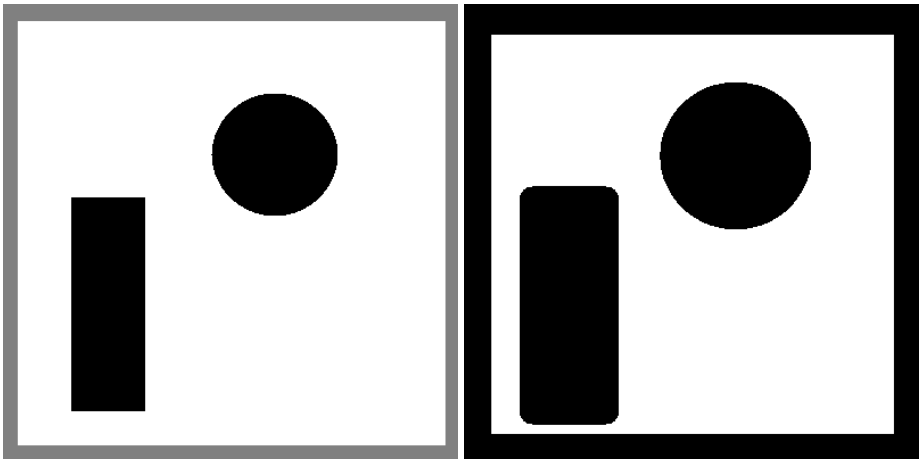


Figura 3.1: Aproximação da geometria do robô a um circulo de raio R.

Para a expansão dos obstáculos recorre-se à biblioteca *(Open Source Computer Vision Library)* OpenCV. É aplicada a função *Dilate* ao mapa original, que permite a alargamento dos obstáculos, tal como ilustra a figura 3.2. Esta, utiliza uma estrutura, denominada *kernel* com uma forma e tamanho específico. Neste caso a estrutura (kernel) usada foi o círculo com as dimensões do robô, uma vez que a forma geométrica do robô foi aproximada a um círculo como mencionado anteriormente.



(a) (b)

Figura 3.2: Expansão do mapa. Em 3.2a está representado o mapa original. Em 3.2b está representado o mapa com a expansão dos obstáculos.

3.2 Decomposição em células

Nesta secção são descritos os métodos implementados de decomposição em células fixas, *quadtrees* e *framed quadtrees*.

3.2.1 Células Fixas

A decomposição em células fixas consiste na divisão do mapa em células de igual tamanho, em que cada uma representa um nó que contém informação sobre o estado do espaço em que se inserem, isto é, se está livre ou ocupada. Esse tamanho varia de acordo com o tipo de ambiente e com os resultados pretendidos.

Nesta dissertação são usadas células quadradas para a decomposição do mapa. O pseudo-código do presente método está presente no Algoritmo 1, onde T é o tamanho pretendido de cada célula, l e c o numero de linhas e colunas do mapa respetivamente, M a matriz com informação da ocupação do mapa e o *estado* é o vetor que contém a informação sobre o estado da célula.

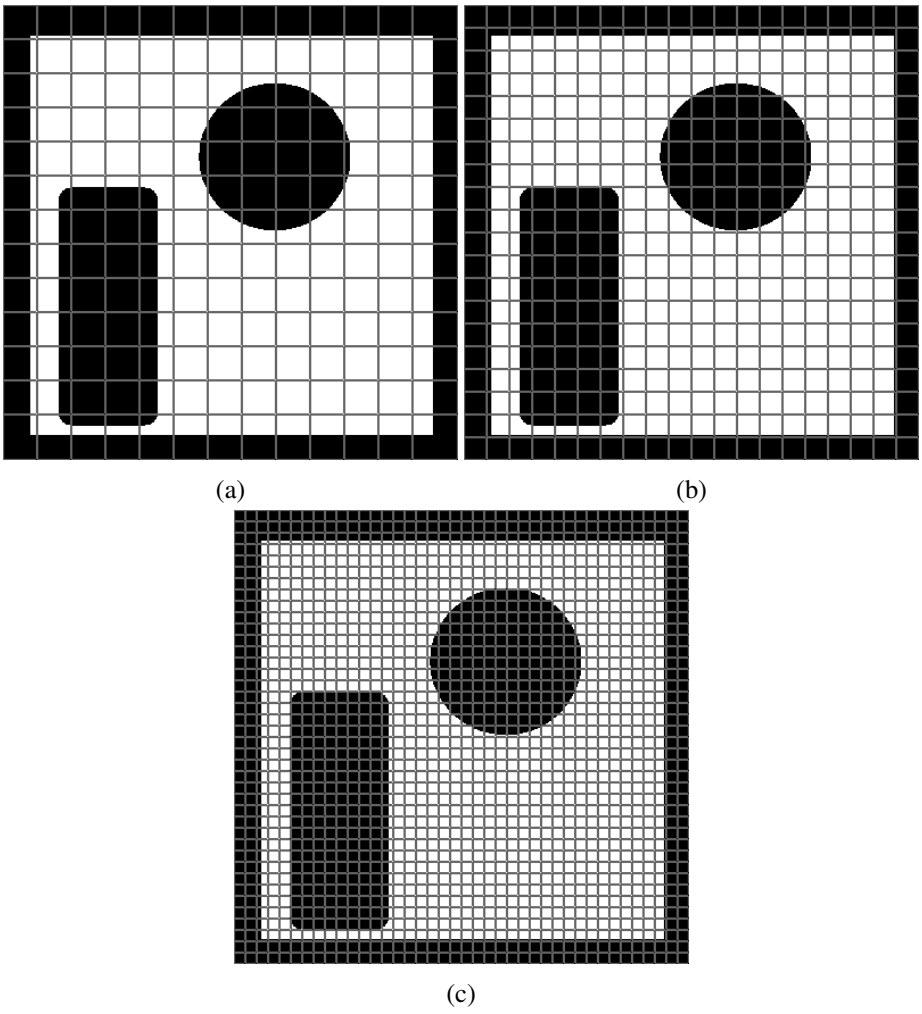
Algoritmo 1: DECOMPOSIÇÃO EM CÉLULAS FIXAS

Entrada: T, l, c, M
Saída: *estado*

1. Calcular o número de células por linha e por coluna
2. Para cada célula:
   (a) Percorrer todos os pontos que pertencem à célula
   (b) Se um ou mais pontos estão ocupados, a célula é considerada ocupada
   (c) Uma célula é considerada livre se todos os pontos estão livres
3. Guardar a informação do estado da célula no vetor *estado*, de forma a que a posição da informação da célula no vetor seja $indice_x + indice_y \times$ *número de células por linha*.

No exemplo da Figura 3.3 o ambiente foi representado com células de dimensões 75×75 cm, 50×50 cm e 25×25 cm. No primeiro caso a resolução do ambiente não permite ao robô planear caminhos entre todos os locais possíveis do mapa, pois existem locais que são erradamente considerados como estando obstruídos, tal como a passagem estreita, onde com esta decomposição, não é considerada espaço livre. Isto deve-se ao facto de bastar que uma pequena porção de um obstáculo seja intersetada por uma célula, para toda a célula ser dada como ocupada. O segundo caso já considera a passagem estreita como livre, e o último caso permite ao robô alcançar quase todas as zonas do mapa, embora um maior número de células tenha de ser expandido para ser encontrado um caminho.

O tamanho das células influencia o desempenho do algoritmo de pesquisa de grafos, posteriormente aplicado, uma vez que quanto menor o tamanho da célula, maior é a quantidade destas para analisar (ver tabela 3.1) o que origina um maior tempo de processamento, no entanto a trajetória definida será próxima do ideal. O tamanho de células escolhido deve ser o suficiente para maximizar a rapidez dos algoritmos de pesquisa, ao mesmo tempo que permite atingir grande parte das zonas do mapa.



(a) (b)

(c)

Figura 3.3: Decomposição do mapa em células de igual tamanho. Na Figura 3.3a o ambiente está representado por células de dimensão 75×75 cm, na Figura 3.3b por células de 50×50 cm e na Figura 3.3b por células de 25×25 cm.

Tamanho das células	Número total de células
75×75 cm	169
50×50 cm	400
25×25 cm	1600

Tabela 3.1: Número de células correspondente ao tamanho de cada célula, associadas à decomposição em células fixas.

As células vizinhas são determinadas através de uma pesquisa com conetividade 8 em torno da célula atual, ou seja consideram-se os vizinhos a norte (N), noroeste (NW), oeste (W), sudoeste (SW), sul (S), sudeste (SE), este (E) e nordeste (NE), como representado na Figura 3.4.

Considerando que a célula atual tem índice x em XX e índice y em YY, são considerados vizinhos células com índices $(x-1, y-1)$, $(x, y-1)$, $(x+1, y-1)$, $(x-1, y)$, $(x+1, y)$, $(x-1, y+1)$, $(x, y+1)$ e $(x+1, y+1)$, onde o estado de cada uma, isto é, se está ocupada ou livre de

NW	*N*	*NE*
W	*O*	*E*
SW	*S*	*SE*

Figura 3.4: Vizinhos da célula *O*.

obstáculos obtêm-se consultando o vetor estado na posição $x + y \times$ *número de células por linha*.

3.2.2 *Quadtrees* (DQ)

O método de decomposição em *quadtrees* utiliza células de tamanho variável para representar o ambiente. As células são decompostas sucessivamente de uma forma recursiva em quatro células-filhas até que uma célula esteja situada numa zona totalmente ocupada ou livre de obstáculos, ou até que uma resolução limite seja atingida.

A *quadtree* é uma estrutura de dados em árvore em que cada nó interno tem exatamente quatro nós filhos, onde cada um representa um quadrante (noroeste - NW, nordeste - NE, sudoeste -SW e sudeste - SE), tal como ilustra a Figura 3.5.

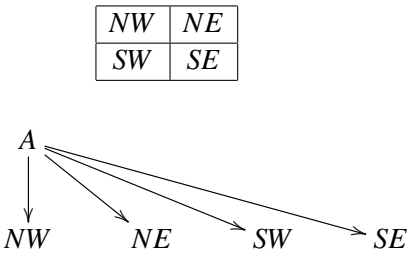


Figura 3.5: Quadrantes correspondente da decomposição em *quadtrees* .

O pseudo-código do algoritmo implementado encontra-se descrito no Algoritmo 2, onde T_l representa a resolução limite das células, l e c o número de linhas e de colunas do mapa respetivamente, M a matriz com informação da ocupação do mapa, e *quadtrees* a lista com a árvore que origina esta decomposição. Cada célula é representada por uma estrutura que contém as seguintes informações:

- Posição inicial da célula em XX e em YY
- Tamanho da célula em XX e em YY
- Estado da célula, se a mesma se encontra totalmente ocupada, totalmente livre, ou se deve ser subdividida
- Nível de divisão a que a célula pertence

- Apontadores para as 4 células-filhas (NW, NE, SW e SE)

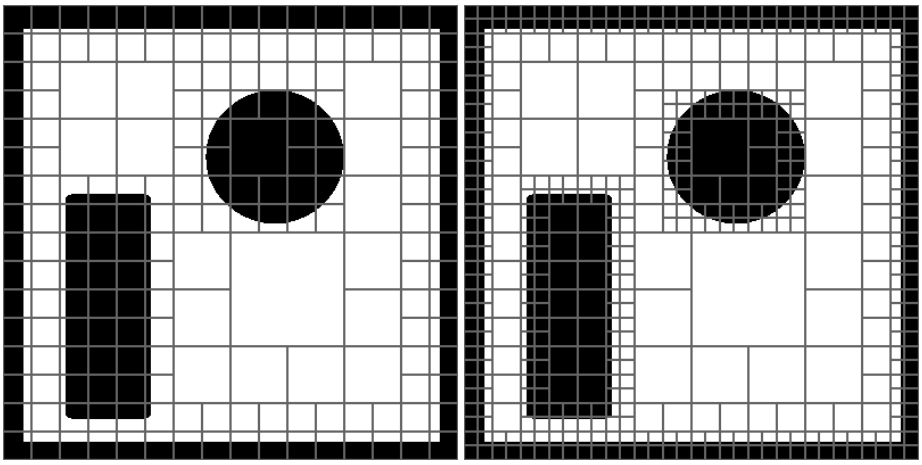
Algoritmo 2: DECOMPOSIÇÃO EM QUADTREES

Entrada: T_l, l, c, M
Saída: *quadtree*

1. Calcular o número máximo de divisões, tendo em consideração a resolução limite (T_l)
2. Repetir
   (a) Dividir em quatros células-filhas todas as células com estado não determinado, isto é, não estão totalmente livres, nem totalmente ocupadas, e que ainda não sofreram divisão.
   (b) Verificar o estado das células-filhas
   (c) Guardar informações referentes a cada célula-filha
3. Até todas as células que não sofreram divisão terem como estado livres ou ocupadas, ou o limite máximo de divisões ser atingido

---

Como as células são sucessivamente decompostas até estarem situadas numa zona totalmente ocupada ou livre de obstáculos, ou até atingir um limite máximo de resolução definido previamente, este método permite obter uma grande precisão nas zonas próximas dos obstáculos. A Figura 3.6 ilustra a decomposição do mapa em *quadtrees*, sendo considerados diferentes resoluções limites. No primeiro caso o limite considerado é de células com dimensões 62×62 cm (Figura 3.6a), e no segundo caso (Figura 3.6b), com dimensões de 30×30 cm. Ambas as representações, permitem a passagem pela zona mais estreita do mapa, no entanto no segundo caso é possível acesso a muitas mais zonas, que o primeiro caso não permite.



(a) (b)

Figura 3.6: Decomposição do mapa em *quadtrees*. Na Figura 3.6a foi considerado um limite 62×62 cm para o tamanho da célula, na Figura 3.6b foi considerado um limite de 30×30 cm.

Na Tabela 3.2 estão representados os números de células correspondente ao limite de resolução imposto. De notar que o número de células indicado corresponde ao número de nós a considerar pelo algoritmo de pesquisa de grafos, pelo que, são consideradas apenas células que não sofreram divisão. Esta solução tem como principal vantagem o número reduzido de células o que origina um menor tempo de processamento aquando da aplicação do algoritmo de pesquisa de grafos.

Limite de tamanho das células	Número total de células
62×62 cm	193
30×30 cm	487

Tabela 3.2: Número de células presentes na decomposição em *quadtrees*.

A resolução limite determina a perda de informação do mapa, isto é, uma maior resolução limite permite células de menor tamanho, pelo que serão possíveis aceder a mais zonas do mapa, em relação a uma resolução limite inferior.

As células são consideradas células vizinhas se estas partilharem uma aresta e/ou um vértice. Assim, se uma célula vizinha for livre de obstáculos, a união entre as mesmas pode pertencer a um caminho, posteriormente determinado por um algoritmo de pesquisa de grafos. A Figura 3.7 representa a ligação entre células vizinhas e livres de obstáculo.

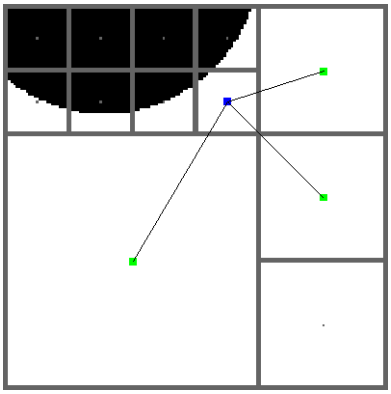


Figura 3.7: Ligação entre células vizinhas na DQ. O centro da célula atual está representado a azul, e o centro das células vizinhas estão representadas a verde

3.2.3 *Framed Quadtrees* (DFQ)

A decomposição em células recorrendo ao uso de *framed quadtrees*, resulta de uma melhoria do método anteriormente descrito na secção 3.2.2. Este algoritmo consiste na adição de células de maior resolução nas proximidades do perímetro de cada região *quadtree*.

Uma vez executado o algoritmo descrito no Algoritmo 2 segue-se a DFQ descrito pelo Algoritmo 3 onde está presente o pseudo-código do mesmo. Neste, o parâmetro de entrada *quadtree* corresponde à lista que contém a árvore construída pela DQ, T_f é o tamanho das células a construir no perímetro das células geradas pela DQ, e *FramedQuadtree* é a lista onde são guardadas as informações das células geradas pelo presente algoritmo.

Algoritmo 3: DECOMPOSIÇÃO EM FRAMED QUADTREES

Entrada: $quadtree, T_f$
Saída: $FramedQuadtree$

1. Para todas as células presentes na lista *quadtree* que não sofreram divisão, e o seu estado é livre:
   (a) Construir as células de tamanho T_f no perímetro da célula
   (b) Guardar as informações das células criadas neste algoritmo na lista $framedquadtree$

Cada célula resultante é representada por uma estrutura que contém as seguintes informações:

- Posição em XX e em YY do ponto central da célula
- Apontador para a célula *quadtree* a qual a célula pertence

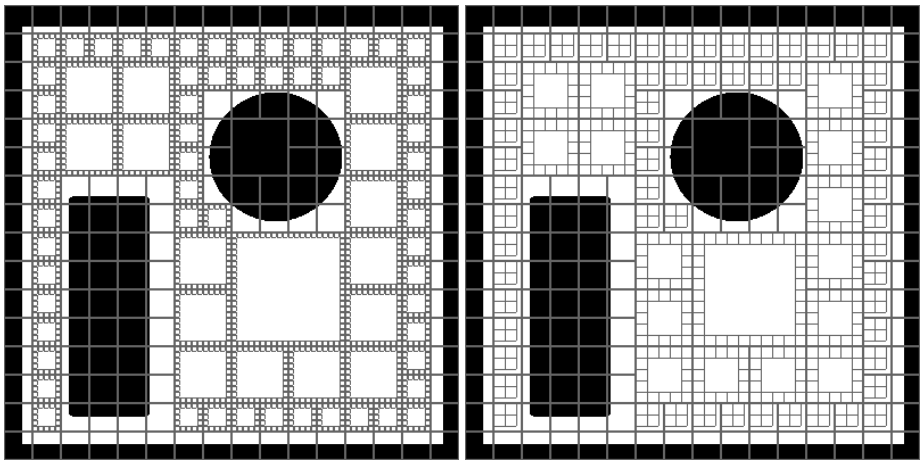
Nesta estrutura são guardados os pontos centrais das células, uma vez que a ligação entre estas é feita através dos mesmos. Apenas as células originadas pelo método descrito no Algoritmo 3 são consideradas pelo algoritmo de pesquisa de grafos.

No exemplo da Figura 3.8 o ambiente foi representado com um limite de resolução de células de 62×62 cm e com células de maior resolução com 13×13 cm de dimensão no primeiro caso (Figura 3.8a) e 25×25 cm no segundo caso (Figura 3.8b). É possível, através da análise das mesmas verificar que os caminhos determinados através desta decomposição podem ser muito próximos dos ideais, no entanto, analisando os valores representados na Tabela 3.3, onde estão indicados o número de células correspondentes às células originadas pela DFQ, verifica-se que o mesmo é elevado, o que origina um maior tempo de processamento quando aplicado o algoritmo de pesquisa de grafos. O número de células representado é o número de células geradas através da DFQ, isto é, as células originadas na DQ não são contabilizadas, uma vez que estas não são consideradas pelo algoritmo de pesquisa de grafos para determinar o caminho.

Figura	Número total de células
3.8a	1576
3.8b	516

Tabela 3.3: Número de células presentes na decomposição em *framed quadtrees*.

Neste algoritmo, quando se determina os vizinhos das células geradas através da DQ, é necessário guardar a informação do lado a que a célula vizinha pertence, isto é, se a célula vizinha está situada a norte (N), Sul (S), oeste (W), este (E), nordeste (NE), noroeste (NW), sudeste (SE) ou a sudoeste (SW) da célula atual. Uma vez determinado os vizinhos de uma célula-mãe (célula gerada pela DQ), os vizinhos de uma célula-filha (célula gerada pela DFQ), são as células-filhas pertencentes à mesma célula-mãe, e ainda as células-filhas das células-mãe vizinhas da célula atual, se estas se encontrarem dentro de certos limites. Estes limites são enumerados a seguir, onde *v* representa a célula-mãe vizinha e *n* representa a célula-filha atual:



(a) (b)

Figura 3.8: Decomposição do mapa em *framed quadtrees*. Foi considerado um limite 62×62 cm para o tamanho da célula, e as células de maior resolução têm dimensões de 13×13 cm e 25×25 cm nas figuras 3.8a e 3.8b, respetivamente.

- Se v é uma célula vizinha a Norte ou a Sul
  1. são consideradas vizinhas as células-filhas (células originadas pela DFQ) de v se a sua posição em XX estiver entre os limites da célula-mãe em XX de n e se a posição x de n estiver nos limites de XX de v
  2. se não existirem falhas ao lado que pertencem (N ou S), e v não contém nos seus limites em XX o x da célula n, são consideradas vizinhas as células-filhas de v que que não ultrapassam os limites em YY de n +/- tamanho das células-filhas
- Se v é vizinha a Oeste ou a Este
  1. são consideradas vizinhas as células-filhas de v se a sua posição em YY estiver entre os limites da célula-mãe em YY de n e se a posição y de n estiver nos limites de yy de v
  2. se não existirem falhas ao lado que pertencem (W ou E), e v não contém nos seus limites em YY o y da célula n, são consideradas vizinhas as células-filhas de v que que não ultrapassam os limites em XX de n +/- tamanho das células-filhas
- Se v é vizinha a NW, NE, SW ou SE, apenas é considerada vizinha a célula-filha mais próxima da célula atual, se nos lados a que pertence não existirem falhas.

A Figura 3.9 ilustra as células-filhas consideradas vizinhas, representadas pelo ponto central a vermelho, de uma célula que se encontra representada a verde. Quando estamos na presença de uma célula vizinha a NW, NE, SW ou SE, apenas é considerada a célula-filha mais próxima da atual, se não existirem falhas dos lados pertencentes à vizinhança, ou seja, considerando que estamos perante uma célula vizinha a NW, se não existirem falhas nos vizinhos a N e a W, isto

significa, se não existirem vizinhos a N e a W ocupados, então é considerada a célula-filha da célula a NW mais próxima da célula atual. Na Figura 3.9 está representado o caso descrito, em que a célula atual, representada a verde, tem vizinhos a NW, NE, SW e SE, no entanto existem falhas na vizinhança a N, pelo que nenhuma das células-filhas de NW e NE são consideradas. Em relação às células a SW e SE, são consideradas as células-filhas mais próximas, pois não existem falhas a S e W nem a S e E, respetivamente.

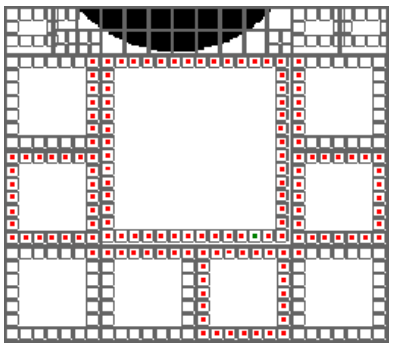


Figura 3.9: Células consideradas vizinhas, representadas pelo ponto central a vermelho, da célula atual, representada a verde, no método de deomposição em *framed quadtrees*.

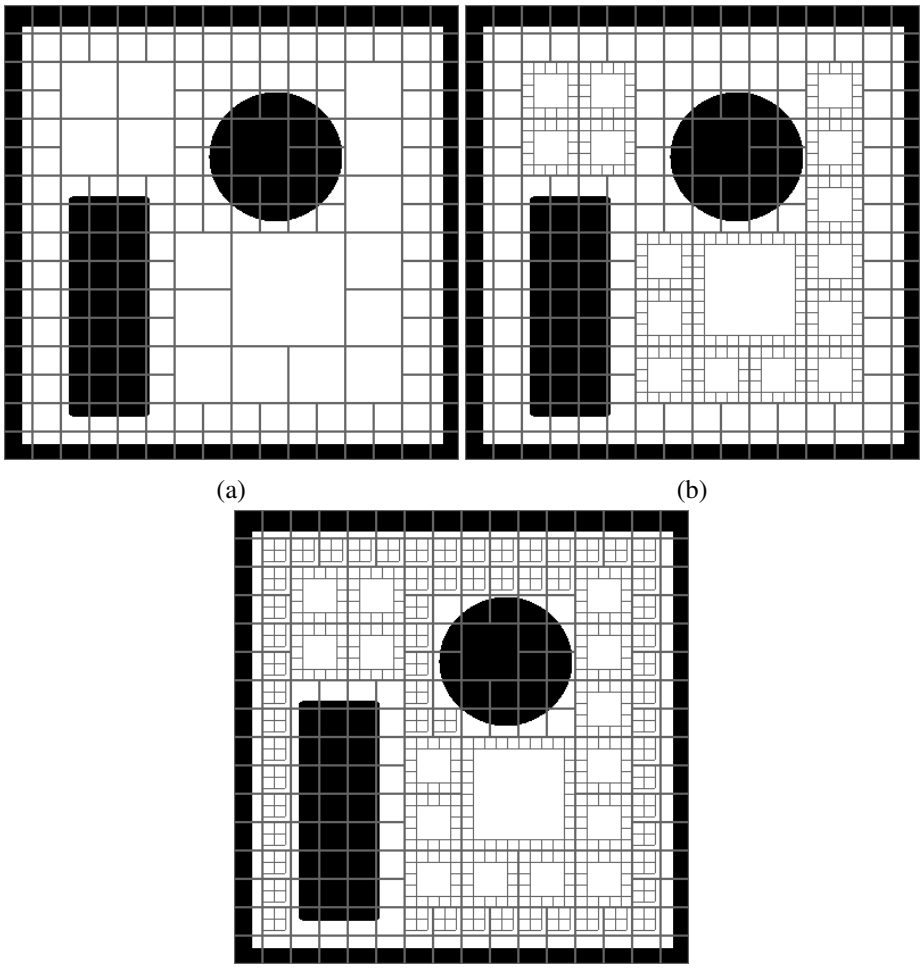
3.3 Solução Proposta - K-*Framed Quadtrees*

Esta decomposição surge com o intuito de compor um algoritmo que resulte numa trajetória melhor que a resultante através da decomposição em *quadtrees* e com um tempo de processamento menor que o associado à decomposição em *framed quadtree*, através da junção dos mesmos.

Este algoritmo é muito semelhante à decomposição em *framed quadtrees* (DFQ), sendo executada numa primeira fase a decomposição em *quadtrees* (DQ) descrita no Algoritmo 2. Posteriormente são adicionadas células de maior resolução nas proximidades do perímetro de cada célula que tenha um tamanho superior ao limite inferior estabelecido previamente, k. O pseudo-código de DFQ é idêntico ao descrito no Algoritmo 3. Isto permite um ajuste entre os dois algoritmos em que nos dois pontos extremos é possível ter uma decomposição muito semelhante à DQ e à DFQ.

A Figura 3.10 ilustra o mapa decomposto através do presente algoritmo onde é considerada uma resolução limite de células com dimensões de 62×62 cm, as células de maior resolução têm dimensões de 25×25 cm e um k de 2.5×2.5 m, 1×1m e 25×25 cm, nas figuras 3.10a, 3.10b e 3.10c, respetivamente. No primeiro caso, a decomposição em célula com o coeficiente $k = 2.5 \times 2.5$ m resulta numa decomposição idêntica à DQ, pois não existem células com tamanho superior ao coeficiente k, já o terceiro caso resulta numa decomposição idêntica à DFQ, pois o coeficiente considerado foi inferior ao tamanho das menores células. O segundo caso é um intermédio entre a DQ e a DFQ.

No caso de o coeficiente k escolhido ser elevado, e dessa forma existirem células que não sofreram DFQ, se alguma dessas células pertencerem ao ponto inicial ou ao ponto destino, a DFQ é executada e posteriormente apagada. A razão pela qual estas células-filhas devem ser



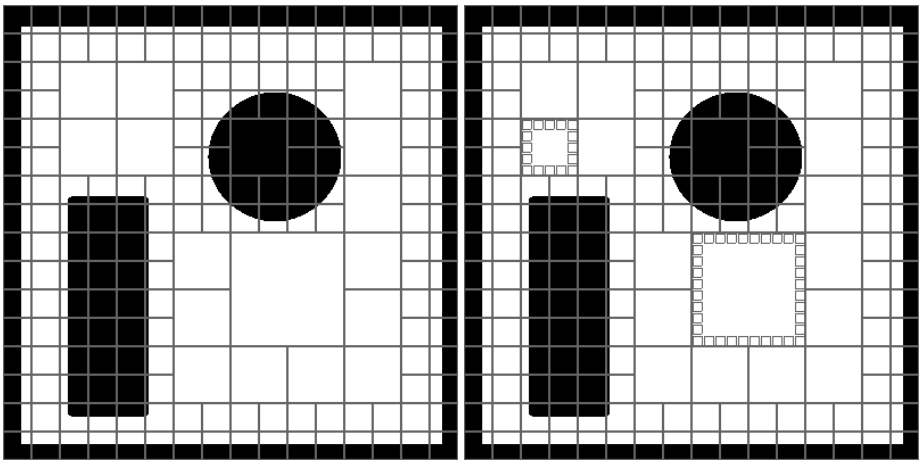
(a) (b)

(c)

Figura 3.10: Aplicação do K-*Framed Quadtrees* à decomposição do mapa. Foi considerado um limite 62×62 cm para o tamanho da célula, as células de maior resolução têm dimensões de 25×25 cm, sendo estas aplicadas a células com dimensões superiores a 2.5×2.5 m (Figura 3.10a), a 1×1 m (Figura 3.10b) e a 25×25 cm (Figura3.10c).

eliminadas, deve-se ao facto de se pretender manter as configurações escolhidas previamente. Se as mesmas não fossem eliminadas, após a execução do algoritmo de pesquisa de grafos para determinar várias trajetórias, a representação do mapa seria cada vez mais semelhante à DFQ, mesmo que a configuração inicial escolhida resultasse numa representação mais semelhante à DQ. Assim, as mesmas são eliminadas posteriormente à trajetória ser determinada, mantendo assim a configuração escolhida. Na Figura 3.11a está uma representação do mapa em que nenhuma das células sofreu DFQ, no entanto aquando da execução do algoritmo de pesquisa de grafos são determinadas as células que contém o ponto inicial e final, sendo então executada a DFQ nessas mesmas células, representada na Figura 3.11b.

As células a considerar para a pesquisa de grafos são as células obtidas através da DFQ em células de dimensões superior a k e células que pertencem à DQ se estas tiverem dimensões inferiores a k. A Tabela 3.4 descreve o número de células a considerar pelo algoritmo de pesquisa de



(a) (b)

Figura 3.11: Representação do mapa através do método K-*Framed Quadtrees* (Figura 3.11a). Alteração da representação do mapa aquando da execução do algoritmo de pesquisa de grafos nas células pertencentes ao ponto inicial e ao ponto destino (Figura 3.11b).

grafos para diferentes configuração da decomposição K-*Framed Quadtrees*.

Figura	Número total de células
3.10a	193
3.10b	439
3.10c	516

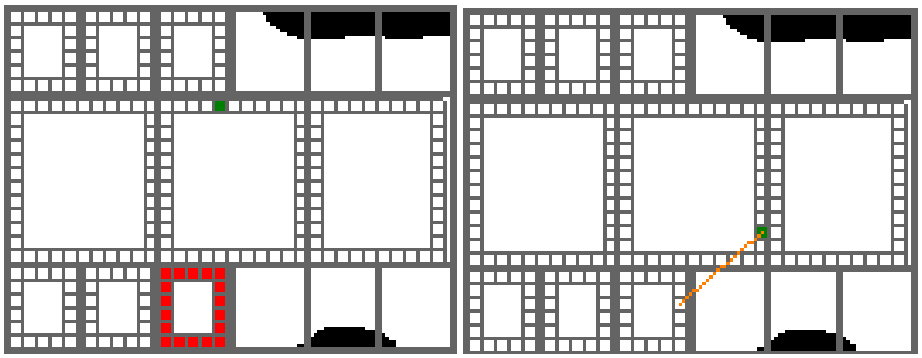
Tabela 3.4: Número de células presentes no algoritmo K-*Framed Quadtrees* com diferentes configurações.

Quando são determinadas as células vizinhas originadas pela DQ (células-mãe), é determinado se existe alguma falha na vizinha a Norte(N), Sul (S), Oeste (W) e a Este (E), isto é, se alguma das células vizinha está ocupada. A seguir são mencionadas todas as restrições para uma célula ser considerada vizinha da célula atual *n*.

1. Se a célula *v* é vizinha a S da célula *n*

   (a) Se *v* tem células-filha (células com origem na DFQ)

      i. São consideradas células vizinhas as células-filhas de *v* se a célula *v* tiver nos seus limites de *XX* o x da célula *n* (Figura 3.12a)



(a) (b)

Figura 3.12: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho 3.12a. A Figura 3.12b ilustra a situação em que, se a presente restrição não existisse a ligação entre as células podia ser de forma insegura (reta a laranja).

ii. Se não existirem falhas a S da célula *n* e se *v* não contém nos limites de XX o x da célula *n*

A. São consideradas células vizinhas, as células filhas de *v* que não ultrapassam os limites da célula-mãe de *n* em XX, e que não ultrapassam em YY o limite superior de YY da célula-mãe de *n* mais o tamanho de cada célula-filha (Figura 3.13)

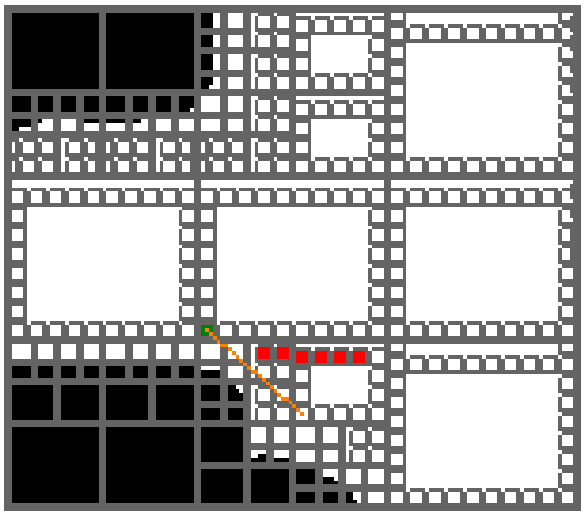


Figura 3.13: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representa. A reta a laranja representa a situação em que, se a presente restrição não existisse a ligação entre as células podia ser de forma insegura.

(b) Se *v* não tem células-filha

i. Se *n* for uma célula-mãe (Figura 3.14)

A. A célula *v* é considerada uma célula vizinha

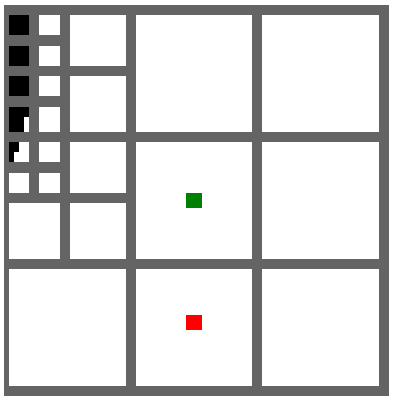


Figura 3.14: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

ii. Se *n* é uma célula-filha

A. *v* é considerada célula vizinha se contiver nos seus limites de *XX* o x da célula *n* (Figura 3.15)

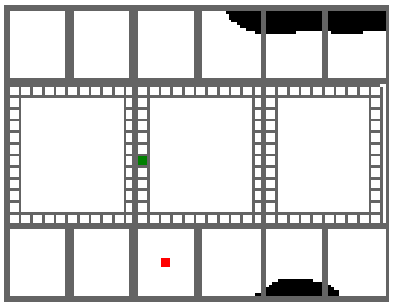
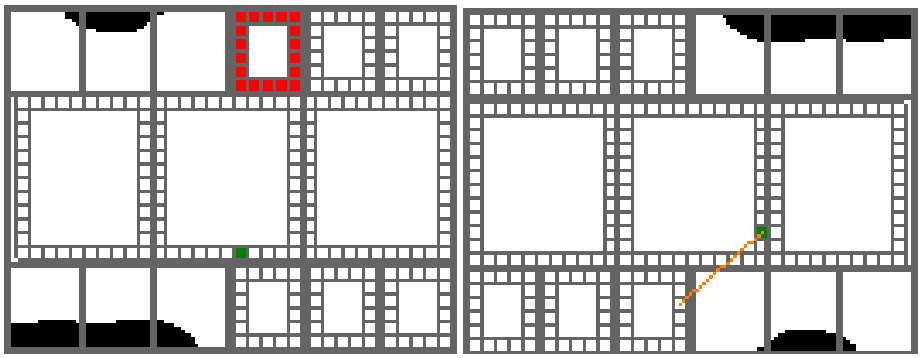


Figura 3.15: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

2. Se a célula *v* é vizinha a N da célula *n*

(a) Se *v* tem células-filha

i. São consideradas células vizinhas as células-filhas de *v* se a célula *v* tiver nos seus limites de *XX* o x da célula *n*



(a) (b)

Figura 3.16: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho 3.16a. A Figura 3.16b ilustra a situação em que, se a presente restrição não existisse a ligação entre as células podia ser de forma insegura (reta a laranja).

ii. Se não existirem falhas a N da célula *n* e se *v* não contém nos limites de XX o x da célula *n*

A. São consideradas células vizinhas, as células filhas de *v* que não ultrapassam os limites da célula-mão de *n* em XX, e que não ultrapassam em YY o limite inferior em YY da célula-mãe de *n* menos o tamanho da célula-filha (Figura 3.17)

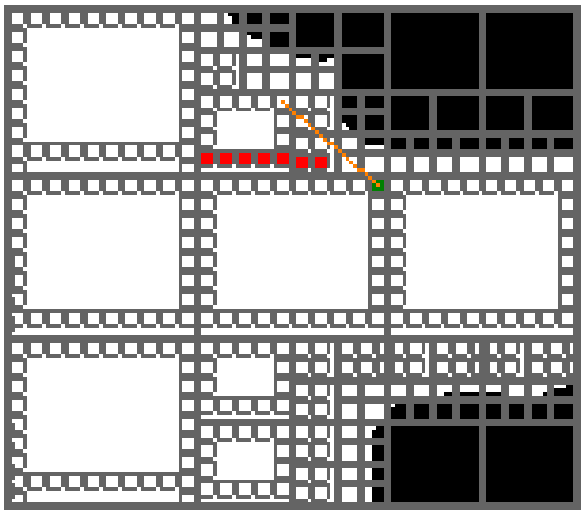


Figura 3.17: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representa. A reta a laranja representa a situação em que, se a presente restrição não existisse a ligação entre as células podia ser de forma insegura.

(b) Se *v* não tem células-filha

i. Se *n* for uma célula-mãe (Figura 3.18)

A. A célula *v* é considerada uma célula vizinha

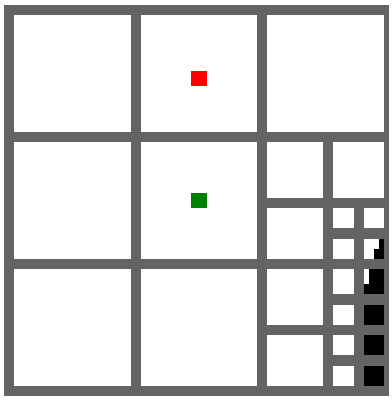


Figura 3.18: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

ii. Se *n* é uma célula-filha

A. *v* é considerada célula vizinha se contiver nos seus limites de XX o x da célula *n* (Figura 3.19)

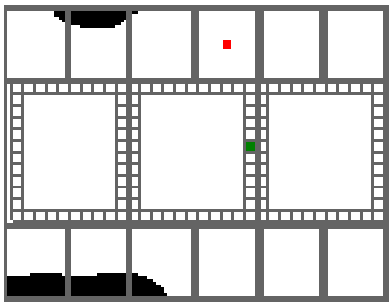


Figura 3.19: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

3. Se a célula *v* é vizinha a W da célula *n*

(a) Se *v* tem células-filha

i. São consideradas células vizinhas as células-filhas de *v* se a célula *v* tiver nos seus limites de YY o y da célula *n* (Figura 3.20)

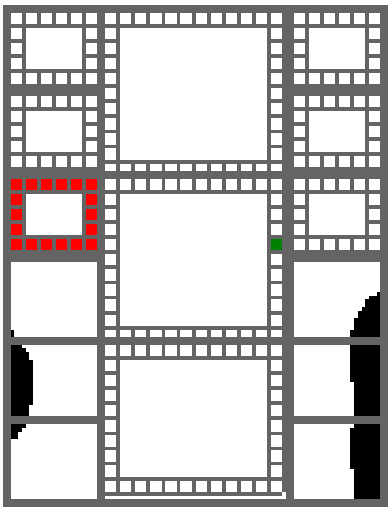


Figura 3.20: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

ii. Se não existirem falhas a W da célula *n* e se *v* não contém nos limites de YY o y da célula *n*

A. São consideradas células vizinhas as células-filhas de *v* que não ultrapassam os limites da célula-mãe de *n* em YY, e que não ultrapassam em XX o limite inferior em XX da célula mãe de *n* menos o tamanho de cada célula-filha (Figura 3.21)

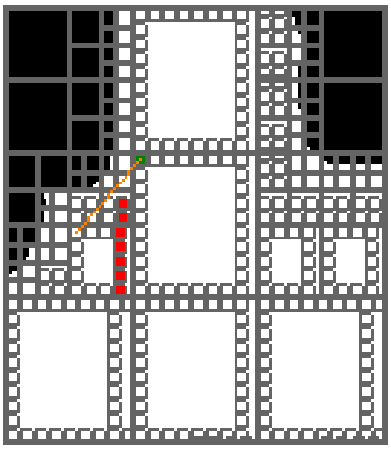


Figura 3.21: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

(b) Se *v* não tem células-filha

i. Se *n* for uma célula-mãe (Figura 3.22)

A. A célula *v* é considerada uma célula vizinha

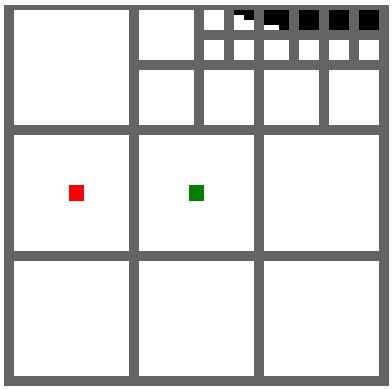


Figura 3.22: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

ii. Se *n* é uma célula-filha

A. *v* é considerada célula vizinha se contiver nos seus limites de *YY* o y da célula *n* (Figura 3.23)

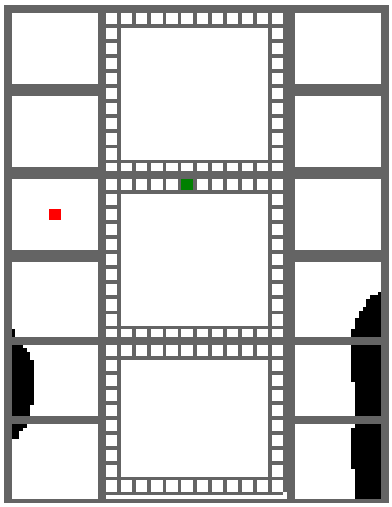


Figura 3.23: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

4. Se a célula *v* é vizinha a E da célula *n*

(a) Se *v* tem células-filha

i. São consideradas células vizinhas as células-filhas de *v* se a célula *v* tiver nos seus limites de *YY* o y da célula *n* (Figura 3.24)

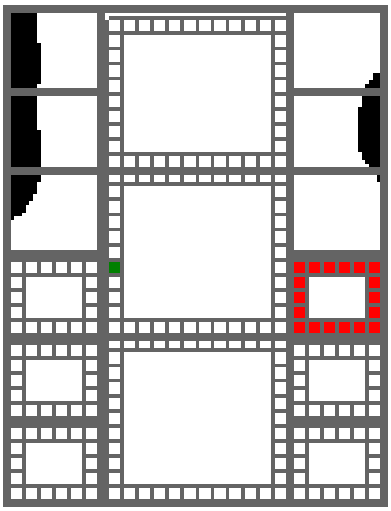


Figura 3.24: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

ii. Se não existirem falhas a E da célula *n* e se *v* não contém em YY o y da célula *n*

A. São consideradas células vizinhas as células filhas de *v* que não ultrapassam os limites da célula-mãe de *n* em YY, e que não ultrapassam em XX o limite superior de XX da célul-mãe de *n* mais o tamanho de cada célula-filha (Figura 3.25)

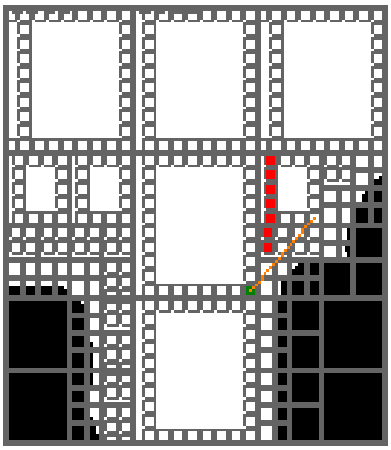


Figura 3.25: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

(b) Se *v* não tem células-filha

i. Se *n* for uma célula-mãe (Figura 3.26)

A. A célula *v* é considerada uma célula vizinha

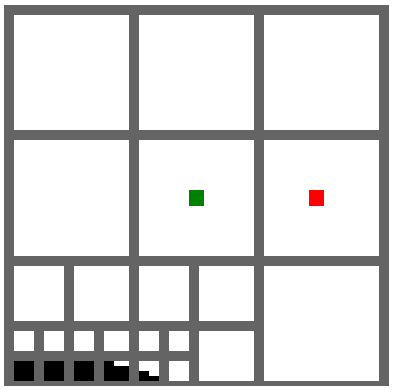


Figura 3.26: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

ii. Se *n* é uma célula-filha

A. *v* é considerada célula vizinha se contiver nos seus limites de *YY* o y da célula *n* (Figura 3.27)

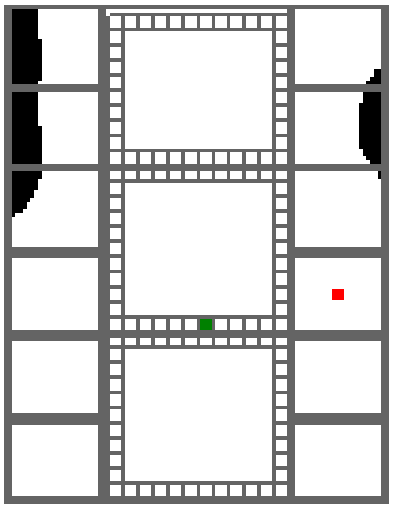


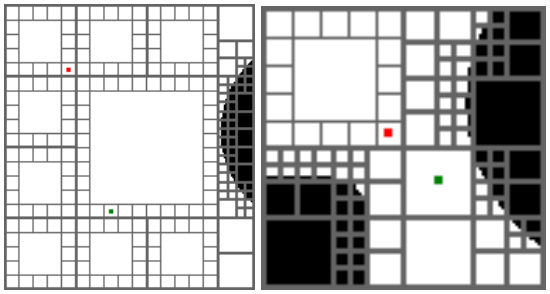
Figura 3.27: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

5. Se a célula *v* é vizinha a NW da célula *n*

(a) Se *v* tem células-filha

i. Se não existem falhas a N e a W de *n* (Figura 3.28a) ou se *n* é uma célula-mãe (Figura 3.28b

A. É considerada vizinha apenas a célula-filha de *v* mais próxima da célula *n*

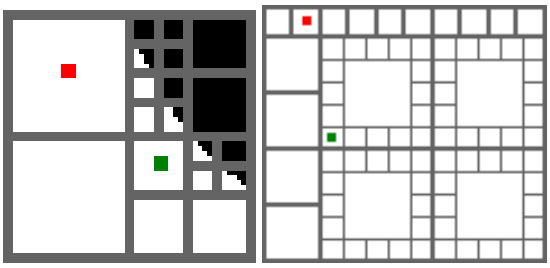


(a) (b)

Figura 3.28: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

(b) Se *v* não tem células-filha

i. *v* é considerada célula vizinha se *n* é uma célula-mãe (Figura 3.29a) ou se não existem falhas a N e W de *n* (Figura 3.29b)



(a) (b)

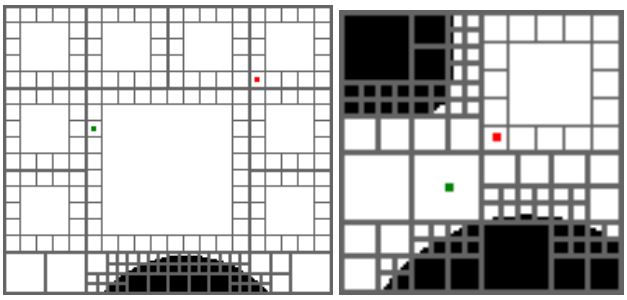
Figura 3.29: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

6. Se a célula *v* é vizinha a NE da célula *n*

(a) Se *v* tem células-filha

i. Se não existem falhas a N e a E de *n* (Figura 3.30a) ou se *n* é uma célula-mãe (Figura 3.30b)

A. É considerada vizinha apenas a célula-filha de *v* mais próxima da célula *n*

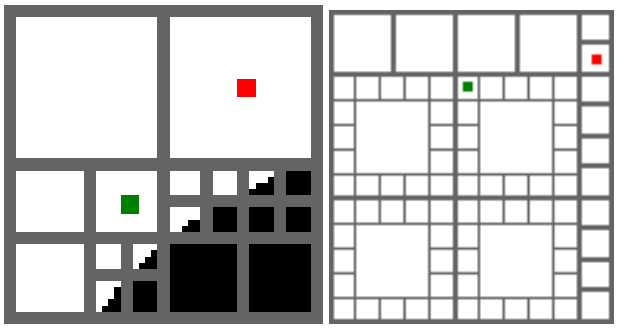


(a) (b)

Figura 3.30: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

(b) Se *v* não tem células-filha

i. *v* é considerada célula vizinha se *n* é uma célula-mãe (Figura 3.31a) ou se não existem falhas a N e E de *n* (Figura 3.31b)



(a) (b)

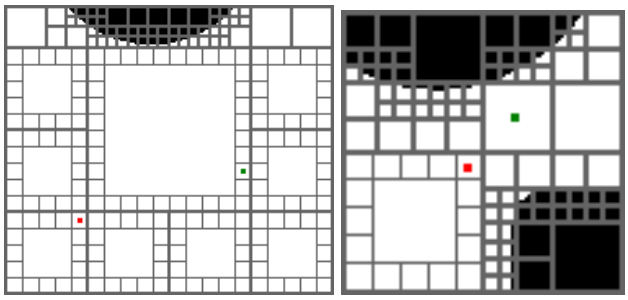
Figura 3.31: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

7. Se a célula *v* é vizinha a SW da célula *n*

(a) Se *v* tem células-filha

i. Se não existem falhas a S e a W de *n* (Figura 3.32a) ou se *n* é uma célula-mãe (Figura 3.32b)

A. É considerada vizinha apenas a célula-filha de *v* mais próxima da célula *n*

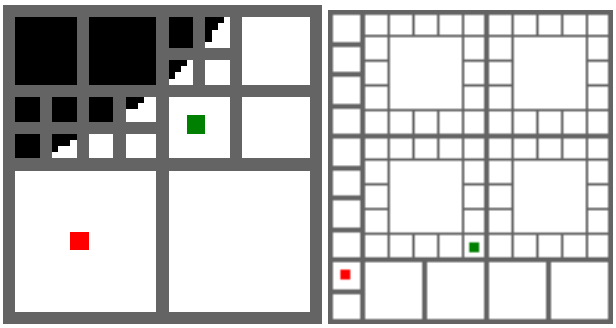


(a) (b)

Figura 3.32: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

(b) Se *v* não tem células-filha

i. *v* é considerada célula vizinha se *n* (Figura 3.33a) é uma célula-mãe ou se não existem falhas a S e W de *n* (Figura 3.33b)



(a) (b)

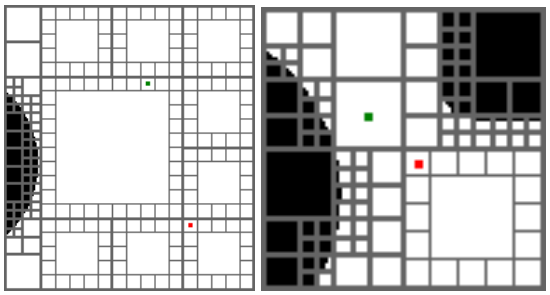
Figura 3.33: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

8. Se a célula *v* é vizinha a SE da célula *n*

(a) Se *v* tem células-filha

i. Se não existem falhas a S e a E de *n* (Figura 3.34a) ou se *n* é uma célula-mãe (Figura 3.34b)

A. É considerada vizinha apenas a célula-filha de *v* mais próxima da célula *n*

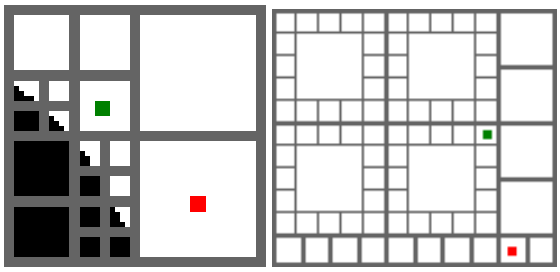


(a) (b)

Figura 3.34: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

(b) Se *v* não tem células-filha

i. *v* é considerada célula vizinha se *n* é uma célula-mãe (Figura 3.35a) ou se não existem falhas a S e E de *n* (Figura 3.35b)



(a) (b)

Figura 3.35: Célula *n* representada pelo ponto central a verde. As células determinadas vizinhas pela presente restrição estão representadas pelo ponto central a vermelho.

Este algoritmo é de maior complexidade devido à forma como se determina as células vizinhas, pois, são várias as restrições a considerar. Na Figura 3.36 estão representadas as células vizinhas da célula atual *n*.

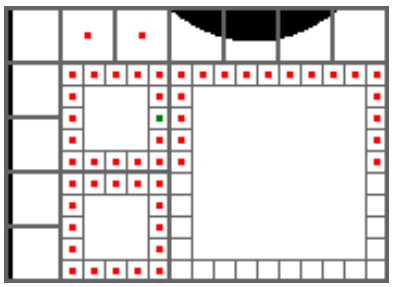


Figura 3.36: Células consideradas vizinhas , representadas pelo ponto cnetral a vermelho, da célula *n* representada a verde.

Com a aplicação deste algoritmo é assim possível determinar um coeficiente que minimize o tempo de processamento, considerando o trajeto que se pretende.

3.4 Algoritmo de pesquisa de grafos A-*star* (A⋆)

Nas secções 3.2.1, 3.2.2, 3.2.3 e 3.3 foram descritos os algoritmos de decomposição em células fixas, em *quadtrees*, em *framed quadtree* e um novo método denominado K-*Framed Quadtrees*, que resultam na representação do mapa por células, onde o ponto central de cada célula corresponde a nós que o algoritmo de pesquisa de grafos utilizará para determinar as trajetórias.

O algoritmo A* é um método de pesquisa de grafos que utiliza como informação a distância entre o nó inicial e o nó atual, e o custo heurístico estimado entre o nó atual e o nó final, tornando-se desta forma um algoritmo eficiente. Este, utiliza uma função de custo $f(n) = g(n) + h(n)$, em que $g(n)$ representa o custo atual desde do nó inicial até ao nó n, e $h(n)$ representa o custo heurístico estimado entre um nó n e o nó que contém o ponto destino. Se o valor de $h(n)$ for inferior ao custo do movimento entre o ponto atual e o ponto final, o algoritmo determinará o caminho mais curto. A ordem pela qual o algoritmo deve procurar os nós é definido pela função $f(n)$, sendo explorado o nó com menor custo. Com o uso do custo heurístico $h(n)$, é possível determinar o caminho de uma forma mais rápida, pois são explorados inicialmente os nós mais promissores.

A implementação deste método consiste no uso de duas listas, a lista aberta e a lista fechada onde são armazenados os nós já identificados mas não explorados e os nós já explorados, respetivamente. A lista aberta é uma lista ordenada de forma crescente relativamente ao custo $f(n)$, sendo desta forma facilitada a procura do nó mais promissor. O pseudo-código está presente no Algoritmo 4 , onde O representa a lista aberta, C a lista fechada, $grafos$ é a lista com todos os nós a considerar e o *caminho* é um vetor com os pontos que definem a trajetória. O algoritmo inicia a sua pesquisa pelo nó que contém o ponto inicial da trajetória, seguindo a sua pesquisa pelos nós vizinhos ao mesmo. Neste momento é explorado o nó vizinho mais promissor, movendo este nó para a lista fechada. São determinados os vizinhos do mesmo e colocados na lista aberta para posterior exploração. Este processo repete-se até que o nó que contém o ponto destino seja determinado, ou quando O está vazia, o que significa que não existe nenhum nó para ser explorado, isto é, não existe trajetória possível entre os dois pontos pretendidos.

O valor a atribuir ao custo heurístico $h(n)$, correspondente à distância entre o nó atual e o nó final, que pode ser estimado de diferentes formas. A distância de *Manhattan*, a distância diagonal e a distância euclidiana são os métodos usualmente utilizados para determinar o valor de $f(n)$. A distância de *Manhattan* é normalmente utilizada quando são permitidos movimentos em apenas quatro direções, movimentos na horizontal e vertical. Este método consiste na distância que é necessário percorrer para atingir a célula final, utilizando apenas movimentos horizontais e verticais, independentemente da ocupação das células, de acordo com a seguinte equação:

$$h(n) = \Delta x + \Delta y \tag{3.1}$$

A distância diagonal é utilizada quando o movimento pode ser feito em 8 direções, ou seja, quando são permitidos movimentos na diagonal. O custo $h(n)$ é calculado através das seguintes

Algoritmo 4: A $\star$

Entrada: $grafo$
Saída: *caminho*

1. Adicionar o nó inicial n a O e guardar os custos g, h e f
2. Repetir
   (a) Procurar o nó de menor custo f em O
   (b) Mover esse nó para C
   (c) Para cada vizinho livre de obstáculos
      i. Se o nó vizinho já estiver presente em C, este não é considerado
      ii. Se o nó vizinho não estiver presente em C nem em O, adicionar o nó a O. n é o nó pai do nó a acrescentar. Guardar os custos g, h e f.
      iii. Se o nó vizinho estiver presente em O, verificar se o caminho atual é melhor para esse nó, isto é, tem um menor custo g. Neste caso, o nó pai é alterado para n.
3. Até o nó destino ser acrescentado na lista C ou até O estar vazia, o que significa que o caminho não foi encontrado
4. Se o caminho for encontrado, guardar o mesmo, começando pelo nó destino, e a partir deste verifica-se qual é o seu nó pai. Em seguida verifica-se nesse nó o seu Pai e assim sucessivamente até se chegar ao nó inicial n.

equações:

$$h_{diagonal}(n) = min(abs(n_x - destino_x), abs(n_y - destino_y)) \tag{3.2}$$

$$h_{N4}(n) = max(abs(n_x - destino_x), abs(n_y - destino_y)) \tag{3.3}$$

$$h(n) = D_d * h_{diagonal}(n) + D_{N4} * h_{N4} \tag{3.4}$$

A distância euclidiana é utilizada quando o movimento pode ser feito em qualquer direção. Trata-se da distância em linha reta entre o ponto n e o ponto destino, calculada a partir da seguinte equação.

$$h(n) = \sqrt{(\Delta x)^2) + (\Delta y)^2)} \tag{3.5}$$

De forma a garantir que o algoritmo A$\star$ determina o caminhos mais curto, a heurística tem de ser subestimadas, no entanto esta deve-se aproximar o mais possível da realidade, por forma a melhorar os resultados. Nesta dissertação são consideradas várias decomposições do mapa em células, já descritas anteriormente. Quando são aplicadas decomposição em *quadtrees*, em *framed quadtrees*, ou quando é aplicado o método K-*Framed Quadtrees*, apenas a distância euclidiana

pode ser aplicar, pelo que foi este o método implementado.

O custo $g(n)$ é considerado, aquando da decomposição em células fixas, como $1 \times T$ quando o movimento ocorre na horizontal e na vertical, e de $1.4 \times T$, quando o movimento ocorre na diagonal, onde T representa o tamanho das células. Quando o algoritmo A⋆ é aplicado aos restantes métodos de decomposição em células descritos anteriormente, o custo $g(n)$ não é fixo, uma vez que as células têm dimensões diferentes, pelo que é usada a distância euclidiana.

As estruturas utilizadas que guardam a informação referente a cada nó têm algumas variações em conformidade com a decomposição em que o algoritmo A⋆ é aplicado, pelo que, são descritas a seguir estas pequenas diferenças.

1. Decomposição em Células Fixas
    - Posição em XX e em YY
    - Posição do nó pai em XX e em YY
    - Custos g, h e f
2. Decomposição em *quadtrees*
    - Apontador para a célula correspondente gerada pela DQ
    - Posição em XX e em YY
    - Posição do no pai em XX e em YY
    - Custos g, h e f
3. Decomposição em *framed quadtrees*
    - Apontador para a célula correspondente gerada pela DFQ
    - Posição em XX e em YY
    - Posição do no pai em XX e em YY
    - Custos g, h e f
4. K-*Framed Quadtrees*
    - Apontador para a célula correspondente gerada pela DQ (se esta for uma célula gerada DQ, caso contrário este é NULL)
    - Apontador para a célula correspondente gerada pela DFQ (se esta for uma célula gerada DFQ, caso contrário este é NULL)
    - Posição em XX e em YY
    - Posição do no pai em XX e em YY
    - Custos g, h e f

Capítulo 4

Controlador

Neste capítulo é descrita a implementação do seguidor de trajetórias omnidirecional. Para tal é desenvolvido um controlador que tem como objetivo fazer com que o robô siga um determinado percurso minimizando o erro entre o trajeto efetuado pelo robô e o trajeto de referência previamente definido. O controlador tem como entradas caminho a percorrer e a posição atual do robô, e como saída as velocidades a serem aplicadas no robô, sendo estas a velocidade linear V_l, a velocidade normal V_n e a velocidade angular V_ω. Na Figura 4.1 estão representadas as entradas e saídas do controlador.

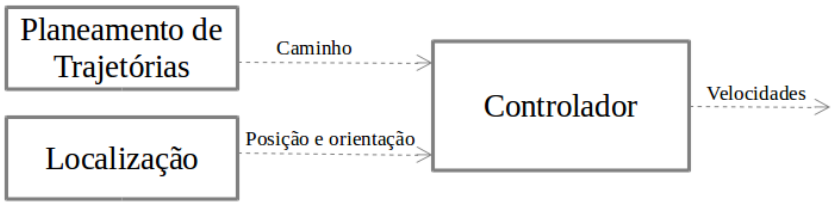


Figura 4.1: Entradas e saídas do controlador.

O robô irá movimentar-se num espaço bidimensional pelo que a sua posição é dada pelas coordenadas x, y e a sua orientação pelo ângulo Θ dadas em função de um referencial global externo ao robô e fixo no espaço.

Nesta dissertação é implementado um método de controlo não linear, denominado *Trajectory Linearization Control* (TLC), baseado em [19], que consiste na linearização ao longo de uma trajetória nominal. O controlo não linear de seguimento por linearização da trajetória pode ser visto como o controlador ideal, uma vez que o ganho é determinado em todos os pontos da trajetória.

São considerados dois referências, o referencial fixo global, definido previamente (X,Y) e o referencial do robô (X_r, Y_r), estando este fixo no centro do mesmo, sendo alterado pelo movimento do robô. A posição do robô em relação ao referencial (X,Y) é designada por (x, y, Θ), e as velocidades linear, normal e angular por V_l, V_n e V_ω respetivamente, tal como ilustra a Figura 4.2.

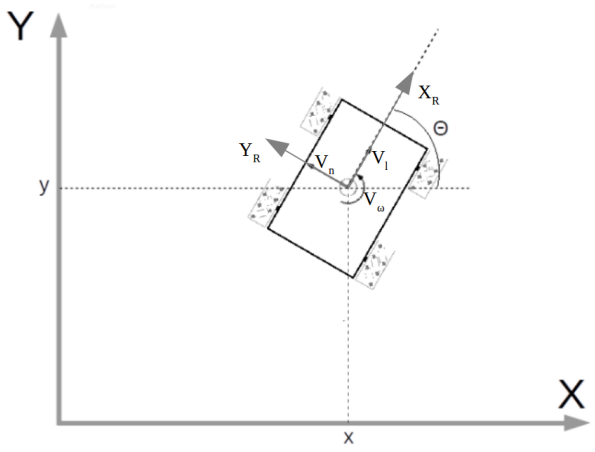


Figura 4.2: Referenciais global e do robô.

A transformação de coordenadas do o referencial (X_r, Y_r) para referencial global (X, Y) é dada pela seguinte equação:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Theta} \end{bmatrix} = A \cdot \begin{bmatrix} V_l \\ V_n \\ V_\omega \end{bmatrix}, onde$$
$$A = \begin{bmatrix} \cos(\Theta) & -\sin(\Theta) & 0 \\ \sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

O diagrama de blocos do controlador implementado encontra-se representado na Figura 4.3, onde é feita uma linearização do modelo cinemático do robô ao longo da trajetória desejada e das velocidades desejadas, calculadas através da cinemática inversa. Desta forma é determinada a dinâmica do erro, sendo, o erro posteriormente estabilizado através de uma lei de controlo PI. De seguida são descritos todos os passos para a implementação deste controlador.

As velocidades nominais desejadas são calculadas pela seguinte equação:

$$\begin{bmatrix} \widehat{V_l} \\ \widehat{V_n} \\ \widehat{V_\omega} \end{bmatrix} = A^{-1} \cdot \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{y}} \\ \dot{\hat{\Theta}} \end{bmatrix} \tag{4.2}$$

Definindo:

$$\begin{bmatrix} e_x \\ e_y \\ e_\Theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \omega \end{bmatrix} - \begin{bmatrix} \hat{x} \\ \hat{y} \\ \widehat{\Theta} \end{bmatrix} e \begin{bmatrix} \delta V_l \\ \delta V_n \\ \delta V_\omega \end{bmatrix} = \begin{bmatrix} V_l \\ V_n \\ V_\omega \end{bmatrix} - \begin{bmatrix} \widehat{V_l} \\ \widehat{V_n} \\ \widehat{V_\omega} \end{bmatrix} \tag{4.3}$$

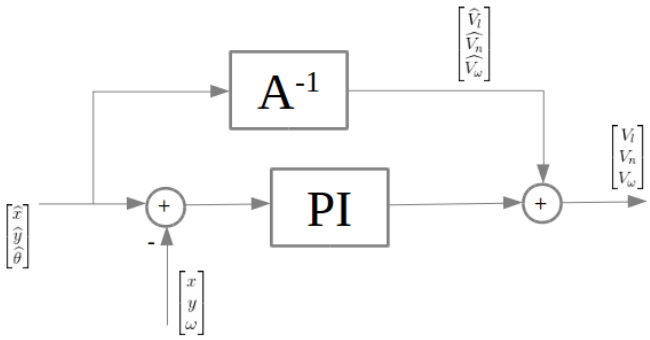


Figura 4.3: Diagrama de blocos do controlador.

Para obter a dinâmica do erro, é feita a linearização do modelo cinemático do robô descrito em 4.1 em torno da trajetória $[\widehat{x},\widehat{y},\widehat{\Theta}]^T$ e das velocidades desejadas $[\widehat{V_l},\widehat{V_n},\widehat{V_\omega}]^T$. A dinâmica do erro é obtida através:

$$\begin{bmatrix} \dot{e_x} \\ \dot{e_y} \\ \dot{e_\Theta} \end{bmatrix} = B \cdot \begin{bmatrix} e_x \\ e_y \\ e_\omega \end{bmatrix} + C \cdot \begin{bmatrix} \delta V_l \\ \delta V_n \\ \delta V_\omega \end{bmatrix} \tag{4.4}$$

Sendo:

$$f = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} \cos(\Theta) * V_l - \sin(\Theta) * V_n \\ \sin(\Theta) * V_l + \cos(\Theta) * V_n \\ V_\omega \end{bmatrix} \tag{4.5}$$

B e C são definidos por:

$$B = \begin{bmatrix} \frac{\partial A1}{\partial x} & \frac{\partial A1}{\partial y} & \frac{\partial A1}{\partial \Theta} \\ \frac{\partial A2}{\partial x} & \frac{\partial A2}{\partial y} & \frac{\partial A2}{\partial \Theta} \\ \frac{\partial A3}{\partial x} & \frac{\partial A3}{\partial y} & \frac{\partial A3}{\partial \Theta} \end{bmatrix}$$

$$C = \begin{bmatrix} \frac{\partial A1}{\partial V_l} & \frac{\partial A1}{\partial V_n} & \frac{\partial A1}{\partial V_\omega} \\ \frac{\partial A2}{\partial V_l} & \frac{\partial A2}{\partial V_n} & \frac{\partial A2}{\partial V_\omega} \\ \frac{\partial A3}{\partial V_l} & \frac{\partial A3}{\partial V_n} & \frac{\partial A3}{\partial V_\omega} \end{bmatrix} \tag{4.6}$$

$$, com \begin{bmatrix} x \\ y \\ \Theta \end{bmatrix} = \begin{bmatrix} \widehat{x} \\ \widehat{y} \\ \widehat{\Theta} \end{bmatrix} e \begin{bmatrix} V_l \\ V_n \\ V_\omega \end{bmatrix} = \begin{bmatrix} \widehat{V_l} \\ \widehat{V_n} \\ \widehat{V_\omega} \end{bmatrix}$$

Assim:

$$B = \begin{bmatrix} 0 & 0 & -\sin(\widehat{\Theta}) * \widehat{V}_l - \cos(\widehat{\Theta}) * \widehat{V}_n \\ 0 & 0 & \cos(\widehat{\Theta}) * \widehat{V}_l - \sin(\widehat{\Theta}) * \widehat{V}_n \\ 0 & 0 & 0 \end{bmatrix} \tag{4.7}$$

$$C = \begin{bmatrix} \cos(\widehat{\Theta}) & -\sin(\widehat{\Theta}) & 0 \\ \sin(\widehat{\Theta}) & \cos(\widehat{\Theta}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

O erro de seguimento foi estabilizado através de uma lei de controlo proporcional e integral (PI).

$$\begin{bmatrix} \delta V_l \\ \delta V_n \\ \delta V_\omega \end{bmatrix} = -K_P \cdot \begin{bmatrix} e_x \\ e_y \\ e_\Theta \end{bmatrix} - K_I \cdot \begin{bmatrix} \int e_x(t)dt \\ \int e_t(t)dt \\ \int e_\Theta(t)dt \end{bmatrix} \tag{4.9}$$

Definindo o vetor alargado do erro de seguimento como:

$$\gamma = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \\ \gamma_6 \end{bmatrix} = \begin{bmatrix} \int e_x(t)dt \\ \int e_y(t)dt \\ \int e_\Theta(t)dt \\ e_x \\ e_y \\ e_\Theta \end{bmatrix} \tag{4.10}$$

A equação do estado de erro é:

$$\dot{\gamma} = D\gamma$$

$$\begin{bmatrix} e_x(t) \\ e_y(t) \\ e_\Theta(t) \\ \dot{e_x} \\ \dot{e_y} \\ \dot{e_\Theta} \end{bmatrix} = D \begin{bmatrix} \int e_x(t)dt \\ \int e_y(t)dt \\ \int e_\Theta(t)dt \\ e_x \\ e_y \\ e_\Theta \end{bmatrix} \tag{4.11}$$

onde, a partir das equações 4.4 e 4.9 obtemos

$$\begin{aligned}\begin{bmatrix} \dot{e_x} \\ \dot{e_y} \\ \dot{e_\Theta} \end{bmatrix} &= B \begin{bmatrix} e_x \\ e_y \\ e_\Theta \end{bmatrix} + C \begin{bmatrix} \delta V_l \\ \delta V_n \\ \delta V_\omega \end{bmatrix} \\ &= B \begin{bmatrix} e_x \\ e_y \\ e_\Theta \end{bmatrix} + C \left(-K_P \begin{bmatrix} e_x \\ e_y \\ e_\Theta \end{bmatrix} - K_I \begin{bmatrix} \int e_x(t)dt \\ \int e_y(t)dt \\ \int e_\Theta(t)dt \end{bmatrix} \right)\end{aligned} \tag{4.12}$$

$$D = \left[\begin{array}{c|c} O_3 & I_3 \\ \hline -CK_I & B - CK_P \end{array} \right] \tag{4.13}$$

O_3 representa uma matriz de zeros 3×3 e I_3 representa uma matriz identidade 3×3. Assim é possível determinar K_P e K_I de forma a obter a dinâmica do erro desejada, onde $a_{1j} > 0$ e a_{2j}>0, presentes na equação 4.14, são os coeficientes do polinómio característico da dinâmica do erro,

$$\lambda^2 + a_{1j}\lambda + a_{2j}$$

.

$$D = \left[\begin{array}{c|c} O_3 & I_3 \\ \hline diag{-}a_{11} - a_{12} - a_{13} & diag{-}a_{21} - a_{22} - a_{23} \end{array} \right] \tag{4.14}$$

Desta forma, $K_P = C^{-1}(B - diag{-}a_{21} - a_{22} - a_{23})$ e $K_I = C^{-1} diaga_{11}a_{12}a_{13}$, onde:

$$C^{-1} = \begin{bmatrix} \frac{\cos(2\widehat{\Theta})+1}{2\cos(\widehat{\Theta})} & \sin(\widehat{\Theta}) & 0 \\ -\sin(\widehat{\Theta}) & \cos(\widehat{\Theta}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

Assim, K_P e K_I são determinados da seguinte forma:

$$K_P = \begin{bmatrix} a_{21}\frac{2\cos(\widehat{\Theta})+1}{2\cos(\widehat{\Theta})} & a_{22}\sin(\widehat{\Theta}) & \frac{\cos(2\widehat{\Theta})+1}{2\cos(\widehat{\Theta})}(-\sin(\widehat{\Theta})\widehat{V_l} - \cos(\widehat{\Theta})\widehat{V_n}) + \sin(\widehat{\Theta})(\cos(\widehat{\Theta})\widehat{V_l} - \sin(\widehat{\Theta})\widehat{V_n}) \\ -a_{21}\sin(\widehat{\Theta}) & a_{22}\cos(\widehat{\Theta}) & \sin(\widehat{\Theta})(\sin(\widehat{\Theta})\widehat{V_l} + \cos(\widehat{\Theta})\widehat{V_n}) + \cos(\widehat{\Theta})(\cos(\widehat{\Theta})\widehat{V_l} - \sin(\widehat{\Theta}\widehat{V_n})) \\ 0 & 0 & a_{23} \end{bmatrix} \tag{4.16}$$

$$K_I = \begin{bmatrix} a_{11}\frac{\cos(2\widehat{\Theta})+1}{2\cos(\widehat{\Theta})} & a_{12}\sin(\widehat{\Theta}) & 0 \\ -a_{11}\sin(\widehat{\Theta}) & a_{12}\cos(\widehat{\Theta}) & 0 \\ 0 & 0 & a_{13} \end{bmatrix} \tag{4.17}$$

A saída do controlador é calculada pela equação 4.18.

$$\begin{bmatrix} V_l \\ V_n \\ v_\omega \end{bmatrix} = \begin{bmatrix} \widehat{V_l} \\ \widehat{V_n} \\ \widehat{V_\omega} \end{bmatrix} + \begin{bmatrix} \delta V_l \\ \delta V_n \\ \delta V_\omega \end{bmatrix} \tag{4.18}$$

A trajetória desejada é definida com um conjunto de pontos (x,y) interligados através de retas. A referência dada ao controlador, são pontos sucessivos dessa reta, com uma distância definida previamente. Isto permite estabelecer a forma como se pretende que o robô execute o trajeto, isto é, é possível definir o ângulo com que o robô deve efetuar a trajetória. Se o objetivo é usar a particularidade de os robôs omnidirecionais se poderem movimentar em todas as direções, define-se que o robô percorre o caminho sempre com o mesmo ângulo θ, ou por outro lado, é possível definir que o robô deve percorrer o caminho com um ângulo θ igual ao ângulo da trajetória a percorrer.

De forma a permitir ao robô um movimento seguro, e com poucas oscilações abruptas na velocidade com que este se movimenta, é estabelecida uma velocidade máxima limite para o movimento de rotação e translação. Sem esta condição, a velocidade do robô estaria relacionada com a distância aplicada aos pontos sucessivos de referência da reta aplicada ao controlador. Uma distância muito elevada, levaria o robô a movimentar-se a grande velocidade, e pequenas distância resultariam numa velocidade menor, não sendo esta abordagem segura para o robô, nem para o meio onde este se insere. Quanto maior a velocidade a que o robô se movimenta, maior deverá ser o erro admissível em cada ponto, de forma a que a o robô se movimente de uma forma suave sem grande oscilações. Menores velocidades permitem que o robô se movimente uma uma forma mais precisa.

Nesta dissertação, o polinómio característico foi definido para todos os canais x, y e θ como

$$\lambda^2 + 2 \times \lambda + 2$$

Os pontos sucessivos da reta que são a entrada de referencia do controlador distam de 10cm e de 0.1 rad se a trajetória for circular. O erro permitido para passar o ponto referencia para o ponto seguinte é de 5cm ou 0.05 rad.

Capítulo 5

Testes e Resultados

Neste capítulo serão apresentados os testes e resultados obtidos para cada um dos métodos implementados para a resolução da problemática associada a esta dissertação. Estes testes foram realizados num PC com um processador Intel(R) Core(TM) i5-6200MQ CPU @ 2.30 GHz e 16 Gb de RAM.

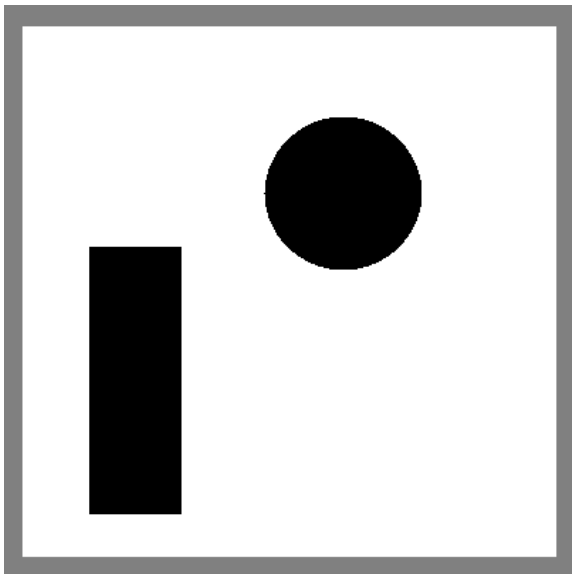
Inicialmente são apresentados os resultados dos métodos de construção de mapas implementados, com aplicação do algoritmo de pesquisa de grafos A$\star$, posteriormente é apresentada uma comparação entre os vários algoritmos, e por fim são demonstrados resultados referentes ao controlador implementado.

5.1 Algoritmos de Planeamento de Trajetórias

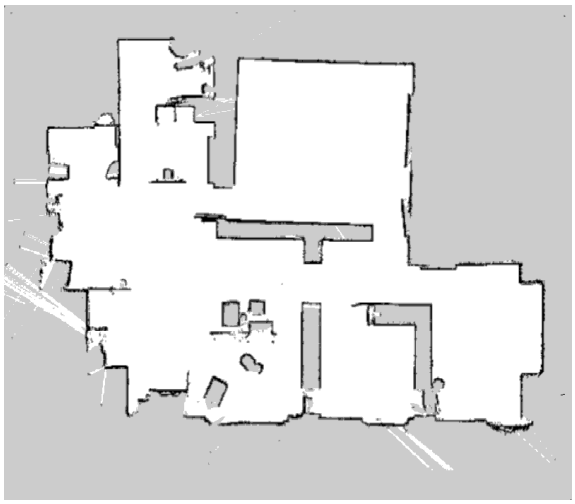
O algoritmo A$\star$ executa a pesquisa dos nós gerados pela construção do mapa. Este método necessita de ter um tempo de processamento rápido, sendo também necessário que as trajetórias geradas se aproximem do ideal, ou seja, devem ter associado a estas o menor comprimento possível com pequenas mudanças de direção, sem comprometer a integridade do robô, sendo esta parte garantida na componente de construção do mapa, isto é, através da expansão e decomposição em células do mapa.

As medidas usadas para avaliar e classificar os métodos de planeamento de trajetórias foram o tempo de processamento (t) e a distância associada ao caminho definido entre o ponto inicial e o destino (D).

Para os testes dos algoritmos de planeamento de trajetórias foram utilizados dois mapas distintos, tendo estes, dimensões de 10×10 m e de 20.1×17.5 m, representados na Figura 5.1a e 5.1b, respetivamente.



(a)



(b)

Figura 5.1: Mapas utilizados nos testes.

5.1.1 A⋆ Aplicado à Decomposição em Células Fixas

Na Figura 5.2 estão representados os caminhos obtidos através do algoritmo de pesquisa de grafos A⋆ aplicado à decomposição em células fixas, com dimensões de 25×25 cm (Figura 5.2a e 5.2d), 50×50 cm (Figura 5.2b e 5.2e), e 75×75 (Figura 5.2c), desde o ponto inicial, representado a verde, até ao ponto destino, representado a vermelho. Na Tabela 5.1 encontram-se representados os efeitos que diferentes tamanhos de células, representado por T, e consequentemente diferentes número de células têm sobre o tempo de processamento e sobre o caminho determinado, analisando assim o tempo de processamento t do algoritmo para determinar o caminho e a distância do mesmo D, nas mesmas condições das trajetórias demonstradas na Figura 5.2.

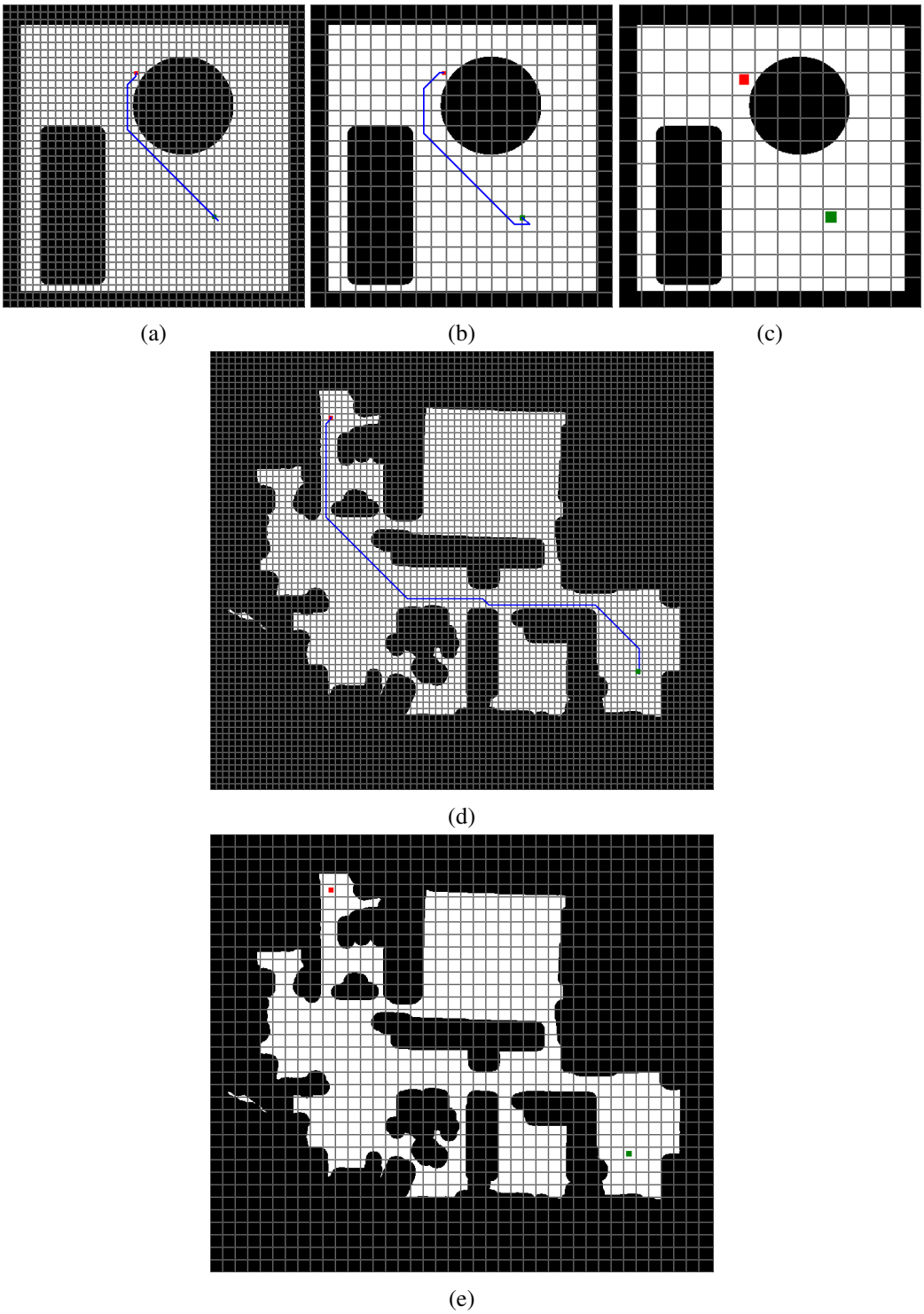


Figura 5.2: Caminho determinado (azul) desde o ponto inicial (verde) até ao ponto destino (vermelho), através da decomposição em células fixas, com diferentes tamanhos de células (ver Tabela 5.1).

Analisando as trajetórias ilustradas na Figura 5.2 e a Tabela 5.1 é possível concluir que quanto menor a resolução das células, maior é a distância do caminho determinado, e maior é a perda

Figura	T [cm]	Número de nós	t [ms]	D [m]
5.2a	25	1600	28	6.05
5.2b	50	400	5	6.9
5.2c	75	169	≈ 0	-
5.2d	25	5600	104	19.7
5.2e	50	1400	≈ 0	-

Tabela 5.1: Resultados para diferentes mapas e diferentes tamanho de células.

de informação, sendo alguns pontos considerados ocupados erradamente. A Figura 5.2a ilustra a trajetória determinada com uma decomposição em células de 25×25 cm, sendo esta considerada a trajetória ideal, com uma menor distância associada. Já o caminho ilustrado na Figura 5.2b tem um custo adicional de distância a percorrer de 0.85 m, no entano o tempo de processamento associado é 23 ms inferior. A Figura 5.2c ilustra o caso em que o destino é considerado ocupado erradamente e dessa forma, o trajeto não é determinado. O caso ilustrado na Figura 5.2e é referente a uma situação em que o destino não é considerado ocupado, no entanto a resolução das células é demasiado pequena, não permitindo assim a passagem do robô por zonas estreitas. A decomposição em células fixas aplicado ao segundo mapa, deve ser executada com células de grande resolução, caso contrário o robô não poderá navegar pela maioria dos locais. No entanto, células com grandes resoluções requerem um maior tempo de processamento, pois existe um maior número de células.

Comparando o desempenho do algoritmo aplicado aos dois mapas distintos, verifica-se que com um mapa de grandes dimensões e passagens estreitas, em relação a um mapa com pequenas dimensões, o número de células e consequentemente o tempo de processamento aumenta consideravelmente.

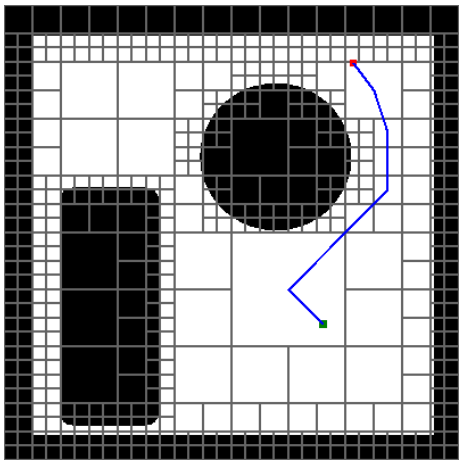
5.1.2 A$\star$ Aplicado à Decomposição em *Quadtrees*

Na Figura 5.3 estão representados os caminhos obtidos através do algoritmo de pesquisa de grafos A$\star$ aplicado à decomposição em *quadtrees*, onde o limite de resolução aplicado corresponde a células com dimensões mínimas de 30×30 cm. Na Tabela 5.2 estão apresentados alguns detalhes referentes ao desempenho do algoritmo ao gerar os caminhos, onde t representa o tempo de processamento e D a distância da trajetória gerada.

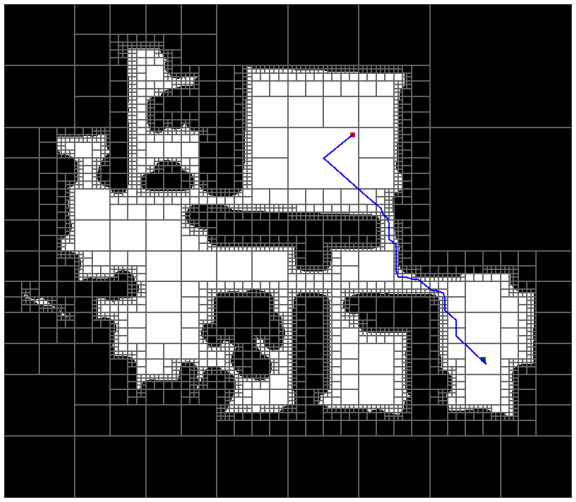
Figura	Limite resolução [cm]	Número de nós	t [ms]	D [m]
5.3a	30×30	499	1	6.9
5.3b	15×12.5	2713	39	11.6
5.3c	15×12.5	2713	94	19.49

Tabela 5.2: Resultados para diferentes mapas e diferentes resoluções limite das células.

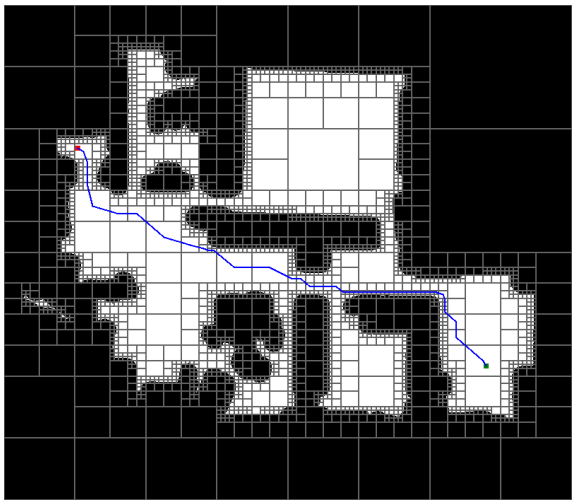
O limite de resolução escolhido para o segundo mapa deve-se ao facto da existência de passagens estreitas, sendo que, apenas este limite ou inferiores permite que o robô se movimente em todas as zonas livres do mapa. Uma maior resolução limite permite ao robô alcançar mais zonas do mapa.



(a)



(b)



(c)

Figura 5.3: Caminho determinado (azul), desde o ponto inicial (verde) até ao ponto destino (vermelho), pelo algoritmo A$\star$ aplicado à DQ.

Este método de planeamento de trajetórias gera caminhos longe do que seria ideal, tal como se verifica nas figuras 5.3a e 5.3b, tendo a trajetória determinada associada distâncias superiores ao que seria esperado, e com mudanças de direção abruptas. No entanto este algoritmo tem períodos de tempo de processamento pequenos, mesmo quando a trajetória a determinar é entre pontos muito distantes e o mapa utilizado é de grandes dimensões, como o caso representado na Figura 5.3c.

A decomposição em *quadtrees* é um método que deve ser utilizado quando se pretende um pequeno período de tempo de processamento, e não se pretende obter caminhos próximos do ideal.

5.1.3 A⋆ Aplicado à Decomposição em *Framed Quadtrees*

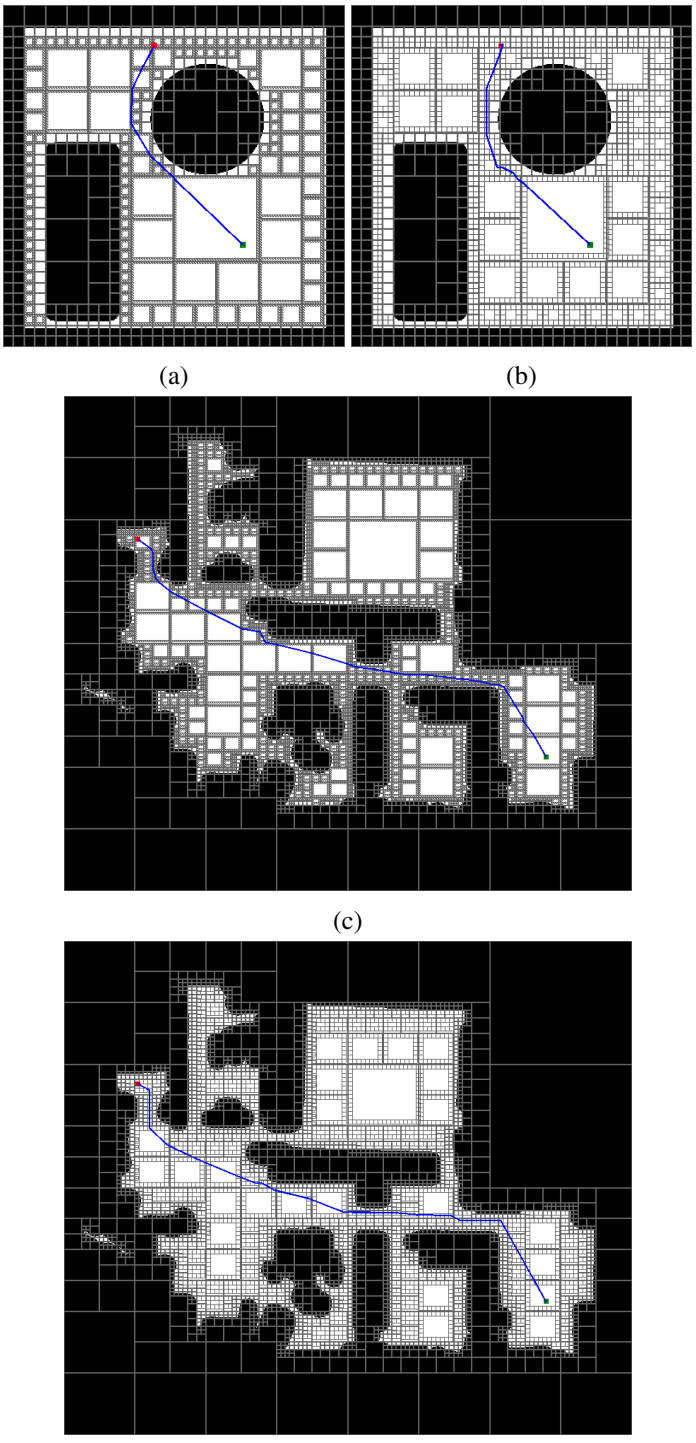
A Figura 5.4 ilustra as trajetórias geradas pelo algoritmo A⋆ aplicado à decomposição em *framed quadtrees*, com diferentes configurações do métodos. As figuras 5.4a e 5.4b têm como resolução limite células de dimensões 30×30 cm, e as células de maior resolução têm dimensões de 7.5×7.5 cm e 15×15 cm, respetivamente. As figuras que ilustram trajetórias determinadas no segundo mapa, 5.4c e 5.4c, têm como resolução limite, células de 15×15 cm e células de maior resolução de dimensões 7.5×7.5 cm e 15×15 cm, respetivamente. A Tabela 5.4 representa alguns detalhes, referentes às trajetórias determinadas, onde l representa a resolução limite e r representa as dimensões das células de maior resolução (células-filha), t o tempo de processamento e D a distância da trajetória gerada.

Figura	l [cm]	r [cm]	Número de nós	t [ms]	D [m]
5.4a	30×30	7.5×7.5	3048	7389	6.66
5.4b	30×30	15×15	1218	717	7.26
5.4c	15×12.5	7.5×7.5	8174	48112	17.68
5.4d	15×12.5	15×15	2177	1372	28.25

Tabela 5.3: Resultados para diferentes mapas com diferentes configurações do algoritmo DFQ.

Analisando as trajetórias representas na Figura 5.4 e a Tabela 5.4 verifica-se que o método gera trajetórias muito próximas do ideal independente do mapa em que se inserem. O coeficiente r é responsável pela distância da trajetória determinada, onde, quanto menor for r, menor é a distância associada ao caminho e mais próximo do ideal este se torna. A perda de informação, tal como no algoritmo de decomposição em *quadtrees*, está relacionado com o coeficiente l, onde um menor l permite uma menor perda de informação, e consequentemente, são permitidos acessos a mais zonas do mapa. No entanto este algoritmo tem associado um grande número de nós, o que se traduz em tempos de processamento muito elevados.

Este algoritmo não deve ser selecionado para aplicações que se pretende uma resposta rápida por parte do algoritmo, devido ao elevado tempo de processamento.



(a) (b)

(c)

(d)

Figura 5.4: Caminho determinado (azul) desde o ponto inicial (verde) até ao ponto destino (vermelho), através da DFQ, com diferentes configurações.

5.1.4 A⋆ Aplicado ao K-*Framed Quadtrees*

De forma a validar e analisar o novo método proposto foram elaborados testes, representados na Figura 5.5 e na Tabela 5.4, onde l representa o limite imposto para a resolução das células, r representa as dimensões das células filhas, k representa o limite máximo de dimensão das células que não sofrem a decomposição em *framed quadtrees*, t o tempo de processamento associado à trajetória e D a distância da trajetória gerada. As figuras seguem o mesmo código de cores apresentados anteriormente.

Figura	l [cm]	r [cm]	k	Número de nós	t [ms]	D [m]
5.5a	30×30	12.5×12.5	25×25 cm	1574	1464	7.34
5.5b	30×30	25×25	62.5×62.5 cm	725	106	7.53
5.5c	30×30	12.5×12.5	5×5 m	499	15	7.71
5.5d	15×12.5 cm	12.5×12.5	12.5×10 cm	4036	8896	17.86
5.5e	15×12.5 cm	37.5×37.5	65×62.5 cm	2845	249	18.25
5.5f	15×12.5 cm	37.5×37.5	5×4.4 m	2713	163	18.8

Tabela 5.4: Resultados para diferentes mapas com diferentes configurações do algoritmo K-*Framed Quadtree*.

Este método permite, através da escolha apropriada do coeficiente r e k, resultados que se situam entre os limites dos resultados gerados através da decomposição em *quadtrees* e da decomposição em *framed quadtrees*.

Para os dois mapas foram feitos testes em que os coeficientes tornavam o algoritmo muito semelhante à DFQ, Figuras 5.5a e 5.5d, semelhantes à DQ, Figuras 5.5a e 5.5f e coeficientes que aproximam o algoritmo a um intermédio dessas duas representações, Figuras 5.5b e 5.5e.

As Figuras 5.5a e 5.5d representam a trajetória ideal, tendo distâncias de $7.34m$ e de $17.84m$, respetivamente paras as trajetórias entre dois pontos em mapas diferentes. No entanto esta configuração, tem associado um elevado tempo de processamento, $1.464s$ e $8.896s$.

Nas Figuras 5.5a e 5.5d, o coeficiente k é considerado muito elevado, pelo que, não existe células de dimensões superiores a k, e consequentemente a DFQ não é executada. Assim, o algoritmo comporta-se de forma muito semelhante à DQ, com a diferença que as células que contém o ponto inicial e o ponto destino, sofrem a DFQ, mesmo não tendo dimensões suficientes, o que resulta numa melhoria da trajetória, em relação à DQ, na partida do ponto inicial e na chegada ao ponto destino. Os tempos de processamento são muito melhores em comparação aos resultados das Figuras 5.5a e 5.5d, sendo estes tempos de $15ms$ e $163ms$ para o primeiro e segundo mapa, respetivamente. Embora os custos temporais sejam bastante satisfatórios, a trajetória gerada tem um custo adicional na distância a percorrer, sendo esta superior em aproximadamente $0.5m$ e $1m$, respetivamente.

Nas Figuras 5.5b e 5.5e são ilustradas trajetórias onde foram selecionados os coeficientes de forma a obter uma trajetória, não muito distante do considerado ideal, com tempos de processamento relativamente baixos. Para os coeficientes considerados o custo adicional na distância da

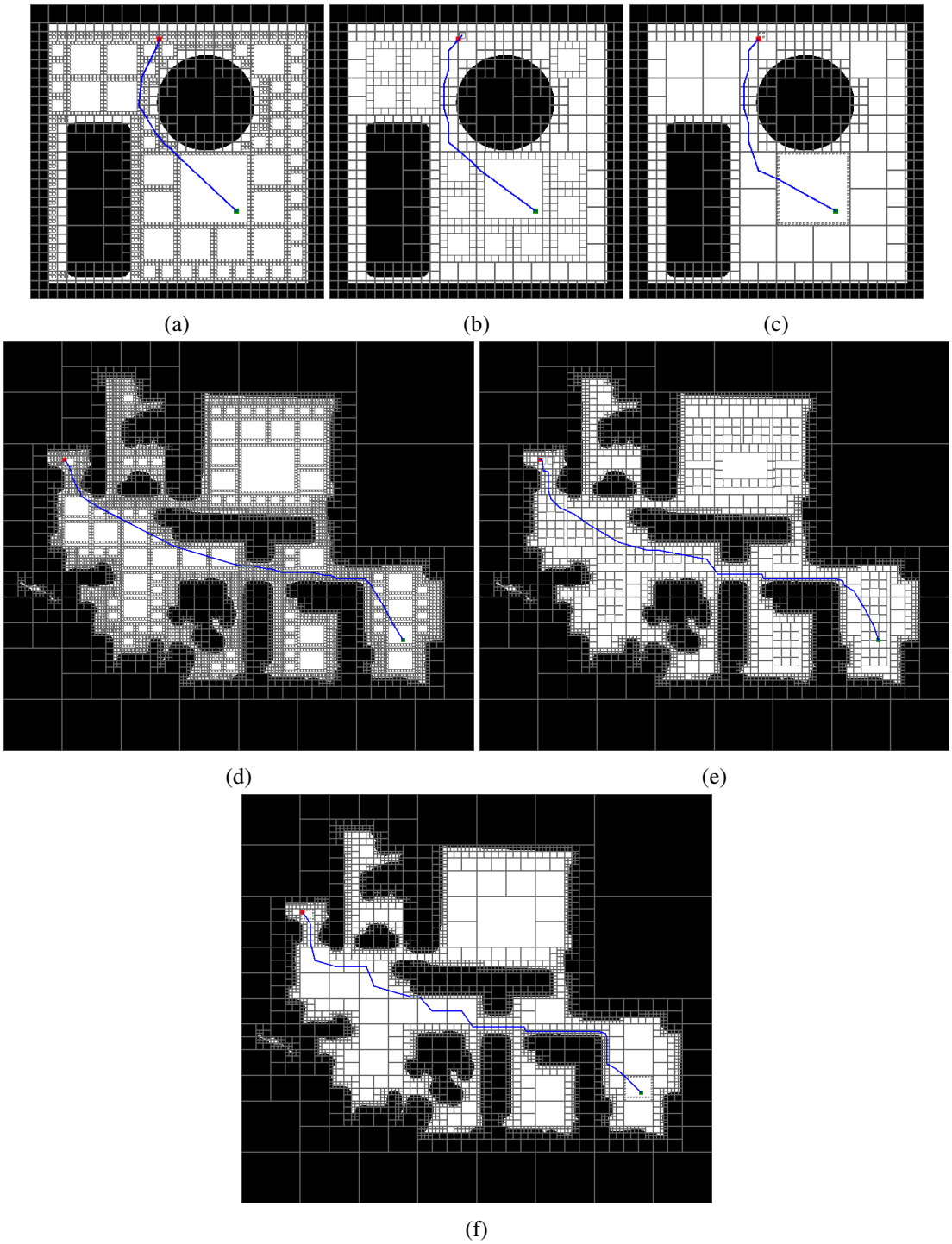


Figura 5.5: Caminho determinado (azul) com diferentes configurações do algoritmo K-*Framed Quadtree* desde o ponto inicial representado a verde, até ao ponto destino, representado a vermelhelho.

trajetória, relativamente ao ideal, é de aproximadamente $0.2m$ e $0.4m$, com tempos de processamento de $106ms$ e $249ms$, respetivamente.

O coeficiente l permite definir a que zonas do mapa pretendemos ter acesso, sendo que, um

valor elevado de l traduz-se numa maior perda de informação. Os coeficientes r e k permitem definir o quão se aproxima do ideal a trajetória gerada. Valores pequenos destes coeficientes, permitem obter trajetórias mais próximas do ideal, no entanto quanto mais pequenos estes forem considerados, maior é o tempo de processamento associado.

5.1.5 Comparação entre os Algoritmos Implementados

Nesta secção é ilustrada os resultados de uma análise comparativa entre os quatro algoritmos de planeamento de trajetórias que foram abordados até ao momento: decomposição em células fixas, decomposição em *quadtrees*, decomposição em *framed quadtrees* e K-*Framed Quadtrees*.

Para os testes comparativos foram utilizados os mesmos dois mapas que nas secções anteriores, sendo um dos mapas de pequenas dimensões, $10 \times 10m$, sem obstáculos muito complexos, e um segundo mapa de grandes dimensões, $20.1 \times 17.5m$ que possuí passagens muito estreitas. É assim analisado o comportamento dos algoritmos aplicados aos diferentes mapas, verificando-se a distância das trajetórias geradas D e o tempo de processamento t das mesmas. Estes testes foram feitos em ambiente de simulação, e foi estabelecido como critério que todos os métodos devem dar acesso às mesmas zonas, ou seja, a perda de informação associada deve ser igual para todos os métodos.

Na Figura 5.6 estão representadas as trajetórias geradas pelos diferentes algoritmos implementados, desde o ponto inicial, representado a verde escuro, até ao ponto destino, representado a vermelho. As trajetórias referente à decomposição em células fixas, *quadtrees*, *framed quadtrees* e ao novo método K-*Framed Quadtrees*, encontram-se representadas a cor-de-rosa, verde claro, azul e amarelo, respetivamente.

Algoritmo	T ou l [cm]	r [cm]	k [cm]	t [ms]	D [m]
Células Fixas	$T = 30 \times 30$	-	-	40	5.5
Quadtrees	$l = 30 \times 30$	-	-	9	6.33
Framed Quadtrees	$l = 30 \times 30$	30×30	-	126	5.4
K-*Framed Quadtrees*	$l = 30 \times 30$	30×30	62.5×62.5	61	5.42

Tabela 5.5: Resultados no primeiro mapa do A$\star$ aplicado à decomposição em células fixas, *quadtrees*, *framed quadtrees* e k-*framed quadtrees*.

Analisando a Figura 5.6 e a Tabela 5.5, verifica-se que a trajetória com uma menor distância associada é a gerada através da decomposição em *framed quadtrees* (DFQ), seguindo-se o método K-*Framed Quadtrees*, a decomposição em células fixas e por fim a decomposição em *quadtrees* (DQ), tendo as três últimas um custo adicional em relação à distância da trajetória gerada através da DFQ, de 0.3%, 1.8% e 17.2%, respetivamente, correspondendo a $2cm$, $10cm$ e $93cm$. Os caminhos determinados através do método K-*Framed Quadtrees* e da DFQ possuem menores distâncias que a decomposição em células fixas, porque esta apenas determina vizinhos das células com conectividade 8, o que apenas possibilita a mobilidade em 8 direções distintas, o que não acontece com os restantes métodos apresentados, onde são permitidos movimentos em diversas direções.

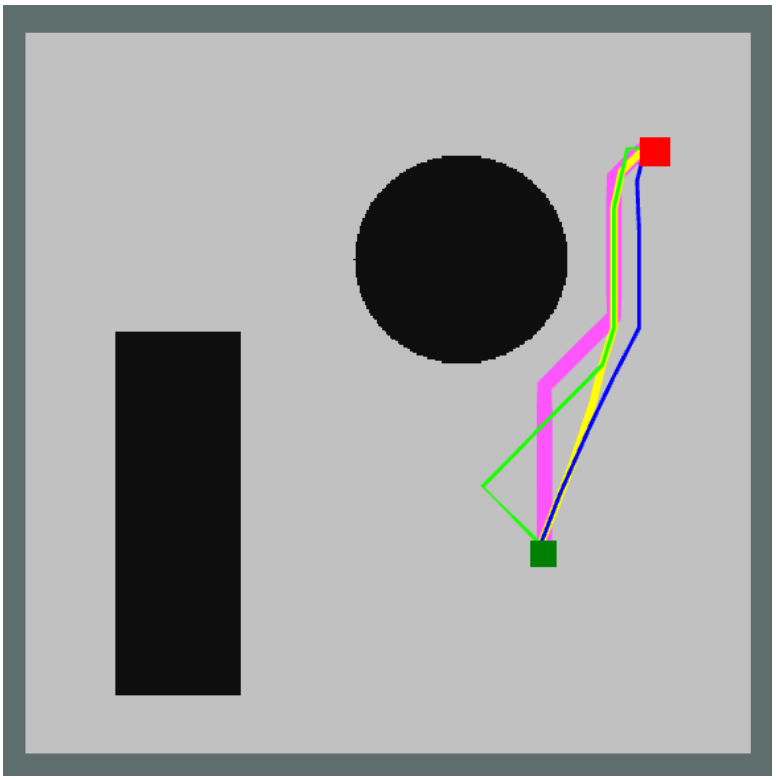


Figura 5.6: Trajetórias geradas no primeiro mapa pelo A$\star$ aplicado aos métodos de decomposição em células fixas (cor-de-rosa), *quadtrees* (verde), *framed quadtrees* (azul) e k-*framed quadtrees* (amarelo).

Relativamente aos tempos de processamento apresentados, todos são inferiores a 200ms, no entanto, o tempo de processamento que se destaca é referente à DQ sendo este apenas de 9ms. As restantes métodos têm um custo adicional de tempo de processamento de 31ms, 52ms e 117ms, referentes à decomposição em células fixas, ao método K-*Framed Quadtrees*, e à DFQ, respetivamente.

Perante um mapa de pequenas dimensões, como o que foi utilizado para o teste descrito, o algoritmo que demonstrou melhores resultados foi a decomposição em células fixas, embora o tempo de processamento seja significativamente melhor na decomposição em *quadtrees*, a trajetória associada a este algoritmo afasta-se do ideal, tem um elevado custo adicional na distância, e, perante os restantes algoritmos, o algoritmo de decomposição em células fixas apresenta um tempo de processamento melhor e uma trajetória muito semelhante, tendo esta um custo adicional de cerca de 10cm relativamente à trajetória considerar ideal, gerada pela DFQ.

O segundo mapa, é um mapa de grandes dimensões com passagens estreitas, onde, quando aplicada a decomposição em *quadtrees*, ou os seus derivados, decomposição em *framed quadtrees* e K-*Framed Quadtrees*, é necessário uma resolução de células limite de $15 \times 12.5cm$ para que o robô se possa movimentar entre passagens estreitas, e assim ter acesso a diversas zonas do mapa. Tal como referido anteriormente, todos os métodos devem ter a mesma perda de informação, e permitir acesso às mesmas zonas, pelo que, foram utilizadas células com dimensões de $15 \times 15cm$

na decomposição em células fixas.

De forma a comparar o comportamento dos métodos implementados, são consideradas três caminhos diferentes, onde o ponto de partida é igual para as três trajetórias, e o ponto destino encontra-se em três zonas diferentes do mapa, P_1, P_2 e P_3. Estes estão representados na Figura 5.7 a verde escuro e a vermelho, respetivamente. As trajetórias referente à decomposição em células fixas, *quadtrees*, *framed quadtrees* e K-*Framed Qaudtrees*, encontram-se representadas a cor-de-rosa, verde claro, azul e amarelo, respetivamente.

Figura 5.7: Trajetórias geradas no mapa de grandes dimensões, pelo A$\star$ aplicado aos métodos de decomposição em células fixas (cor-de-rosa), *quadtrees* (verde), *framed quadtrees* (azul) e k-*framed quadtrees* (amarelo), para o ponto destino P_1.

Algoritmo	T ou l [cm]	r [cm]	k [cm]	t [ms]	D [m]
Células Fixas	$T = 15 \times 15$	-	-	1068	18.05
Quadtrees	$l = 15 \times 12.5$	-	-	190	18.69
Framed Quadtrees	$l = 15 \times 12.5$	12.5×12.5	-	10184	17.80
K-*Framed Quadtrees*	$l = 15 \times 12.5$	30×30	65×62.5	411	18.41

Tabela 5.6: Resultados no segundo mapa do A$\star$ aos métodos de decomposição em células apresentados, para o ponto destino P_1.

Tendo em consideração os valores apresentados na Tabela 5.6, verifica-se que a trajetória ideal é gerada através da DFQ, tendo esta uma distância de 17.80 m. As restantes trajetórias geradas através dos métodos de decomposição em células fixas, K-*framed quadtrees* e DQ, têm um custo adicional na distância do caminho gerado de 1.4%, 3.4% e 5%, respetivamente.

A DQ resulta na trajetória com maior distância, sendo esta aproximadamente 0.7 m superior, no entanto do tempo de processamento associado é o mais satisfatório, sendo este apenas de 190 ms. O caminho determinado através da DFQ é o mais otimizado, dos caminhos apresentados, no entanto este tem associado um tempo de processamento demasiado elevado, sendo este de aproximadamente 10 s. A decomposição em células fixas originou uma trajetória com uma distância 25 cm superior à trajetória considerada ideal e um tempo de processamento de 1 s. O novo método implementado, com as configurações descritas na Tabela 5.6, resultou numa trajetória de 18.41 m, 41 cm superior ao considerado ideal e um tempo de processamento de 411 ms, pelo que se verifica uma melhoria significativa, em termos temporais, em relação à decomposição em células fixas e DFQ.

Figura 5.8: Trajetórias geradas no mapa de grandes dimensões, pelo A$\star$ aplicado aos métodos de decomposição em células fixas (cor-de-rosa), *quadtrees* (verde), *framed quadtrees* (azul) e k-*framed quadtrees* (amarelo), para o ponto destino P_2.

Algoritmo	T ou l [cm]	r [cm]	k [cm]	t [ms]	D [m]
Células Fixas	$T = 15 \times 15$	-	-	496	12.00
Quadtrees	$l = 15 \times 12.5$	-	-	74	12.69
Framed Quadtrees	$l = 15 \times 12.5$	12.5×12.5	-	2903	12.25
K-*Framed Quadtrees*	$l = 15 \times 12.5$	30×30	65×62.5	186	12.39

Tabela 5.7: Resultados no segundo mapa do A$\star$ aos métodos de decomposição em células apresentados, para o ponto destino P_2.

A Figura 5.8 e a Tabela 5.7 demonstram resultados das trajetórias geradas do ponto inicial até

ao ponto destino P_2, pelos diferentes métodos implementados. Neste teste o algoritmo que resultou no caminho ideal foi a decomposição em células fixas tendo este trajeto uma distância de 12 m. Os restantes métodos, DFQ, k-*framed quadtrees* e DQ, tiveram um custo de distância adicional de 2.1%, 3.3% e 5.8%, respetivamente.

Os métodos que demonstraram melhor desempenho referente ao tempo de processamento foi a DQ e o novo método, tendo estes valos inferiores a 200 ms. A DFQ tem associado um tempo de processamento demasiado elevado, face ao comprimento da trajetória, enquanto que a decomposição em células fixas, embora não ultrapasse os 500 ms, é um tempo demasiado elevado, dado que a trajetória é apenas de 12m.

Figura 5.9: Trajetórias geradas no mapa de grandes dimensões, pelo A$\star$ aplicado aos métodos de decomposição em células fixas (cor-de-rosa), *quadtrees* (verde), *framed quadtrees* (azul) e k-*framed quadtrees* (amarelo), para o ponto destino P_3.

Algoritmo	T ou l [cm]	r [cm]	k [cm]	t [ms]	D [m]
Células Fixas	$T = 15 \times 15$	-	-	705	14.60
Quadtrees	$l = 15 \times 12.5$	-	-	124	14.78
Framed Quadtrees	$l = 15 \times 12.5$	12.5×12.5	-	4776	14.46
K-*Framed Quadtrees*	$l = 15 \times 12.5$	30×30	65×62.5	234	14.54

Tabela 5.8: Resultados no segundo mapa do A$\star$ aos métodos de decomposição em células apresentados, para o ponto destino P_3.

A Figura 5.9 e a Tabela 5.8 demonstram resultados das trajetórias geradas do ponto inicial até ao ponto destino P_3, pelos diferentes métodos implementados. Com este ponto destino, a trajetória

mais curta é de 14.46 m e resulta da DFQ, sendo este considerado o caminho ideal. Os restantes métodos, em relação à trajetória ideal tem um custo adicional de distância de 0.6%, 1.0% e 2.2%, sendo estes valores correspondentes aos métodos k-*framed quadtrees*, decomposição em células fixas e DQ, respetivamente. Os tempos de processamento referentes à decomposição em células fixas e à DFQ são mais elevados que os associados aos restantes algoritmos. A DQ possui o melhor tempo de processamento, seguindo-se o método sugerido.

5.1.5.1 Conclusões

Os testes elaborados permitem concluir, que na presença de um mapa com pequenas dimensões, o método que apresenta resultados mais satisfatórios é a decomposição em células fixas, tendo este um bom compromisso entre distância da trajetória gerada e tempo de processamento. Apenas a DQ apresenta um tempo de processamento menor, no entanto, tem associado a trajetória mais distante do que a considerada ideal. No teste referente ao primeiro mapa, esta apresenta um custo adicional de distância de 17%, não sendo assim aplicada quando se pretende uma trajetória ideal. De uma forma totalmente oposta, a DFQ permite uma trajetória muito próxima de ideal, no entanto o tempo de processamento é muito superior aos restantes métodos, não sendo por isso o melhor método a aplicar. O método k*Framed Quadtrees* apresenta uma trajetória muito próxima do ideal, e um tempo de processamento não muito maior que o apresentado pela decomposição em células fixas, no entanto, face a um aumento de 21 ms por uma diminuição de 8 cm na trajetória, este não foi considerado o melhor método.

Para o mapa de maiores dimensões, foram avaliadas as trajetórias geradas pelos métodos de planeamento de algoritmos implementados, para três pontos destinos P_1, P_2 e P_3. A decomposição em células fixas, método que é considerado mais apropriado na presença de mapas de pequenas dimensões, obtém um tempo de processamento bastante mais elevado aquando de mapas de grandes dimensões. Isto torna o algoritmo inadequado a aplicações onde se pretende uma resposta rápida. Em todas as testes elaborados, a DQ apresenta valores muito satisfatórios de tempo de processamento, no entanto este origina trajetórias mais afastadas do considerado ideal, tendo esta como característica a presença de mudanças de direção abruptas, face aos restantes algoritmos. Nesta dissertação pretende-se obter um bom compromisso entre o tempo de processamento e a distância da trajetória, desta forma a DQ torna-se desapropriada, assim como a DFQ, que apresenta na maioria dos testes a trajetória mais curta, mas um tempo de processamento muito elevado, relativamente aos restantes métodos implementados.

Face a estes resultados, surge uma nova decomposição, que visa, através de uma escolha apropriada de coeficientes um compromisso entre a distância da trajetória gerada e o tempo de processamento. Nos testes elaborados no mapa de maiores dimensões, este método apresenta bons resultados em relação a tempos de processamento, sendo que, apenas o método DQ apresenta melhores resultados. Relativamente às distâncias dos caminhos gerados, este apresentou um custo adicional de distância máxima de 81 cm, correspondente a 3.4% da trajetória considerada ideal para o ponto P_1. No entanto, no teste referido, o tempo de processamento associado é de apenas 411 ms, face a 1 s e 10 s correspondentes à decomposição em células fixas e DFQ. A DQ apresenta

tempos de processamento melhores, mas como já foi referido anteriormente, os custos adicionais de distâncias das trajetórias são elevados. Desta forma, conclui-se que, aquando de mapas de grandes dimensões com passagens estreitas, com uma escolha adequada dos coeficientes associados ao novo método, este permite obter melhores resultados que os restantes, se o objetivo for obter um bom compromisso entre tempo de processamento e a distância da trajetória gerada.

De notar que, quando se pretende que o tempo de processamento seja bastante pequeno, o método K-*Framed Quadtrees* é uma boa abordagem, pois com a devida escolha dos coeficientes, permite uma decomposição muito semelhante à DQ, obtendo também resultados parecidos, com uma melhoria da trajetória aquando do inicio da trajetória e da chegada ao destino.

5.2 Controlador

Para classificar o controlador, são tido em consideração o erro máximo com que o robô se movimenta, ou seja a distância máxima a que o robô se afasta da trajetória.

Testes em ambiente de simulação apenas permite validar o funcionamento do controlador, sendo que, o facto do simulador não considerar a dinâmica do robô resulta em erros de seguimento demasiado pequenos que os esperados em ambiente real. Desta forma, em simulação, os resultados do controlador implementado demonstraram ser bastante satisfatórios como seria de esperar, percorrendo a trajetória idealmente.

De forma a analisar o comportamento do controlador aquando de uma situação real, o mesmo foi testado em ambiente real. O controlador foi aplicado a um robô omnidirecional de quatro rodas Mecanum, sendo este um protótipo desenvolvido pelo INESC TEC. Inicialmente foram utilizados os sensores do robô para mapear o local, para posteriormente testar o sistema de navegação que inclui o controlador de trajetórias desenvolvido, um sistema de localização e um planeamento de trajetórias. A Figura 5.10 representa os tópicos e nós ativos durante o teste deste sistema, onde se observa a utilização do nó "localization_perfect_match"para calcular a localização do robô, desenvolvido no INESC TEC, a utilização do nó "mapa_omni_quadrados", sendo este o nó do planeamento de trajetórias baseado na decomposição em células quadradas, e a utilização do nó "controlador_omni", sendo este o nó referente ao controlador apresentado, ambos desenvolvidos nesta dissertação.

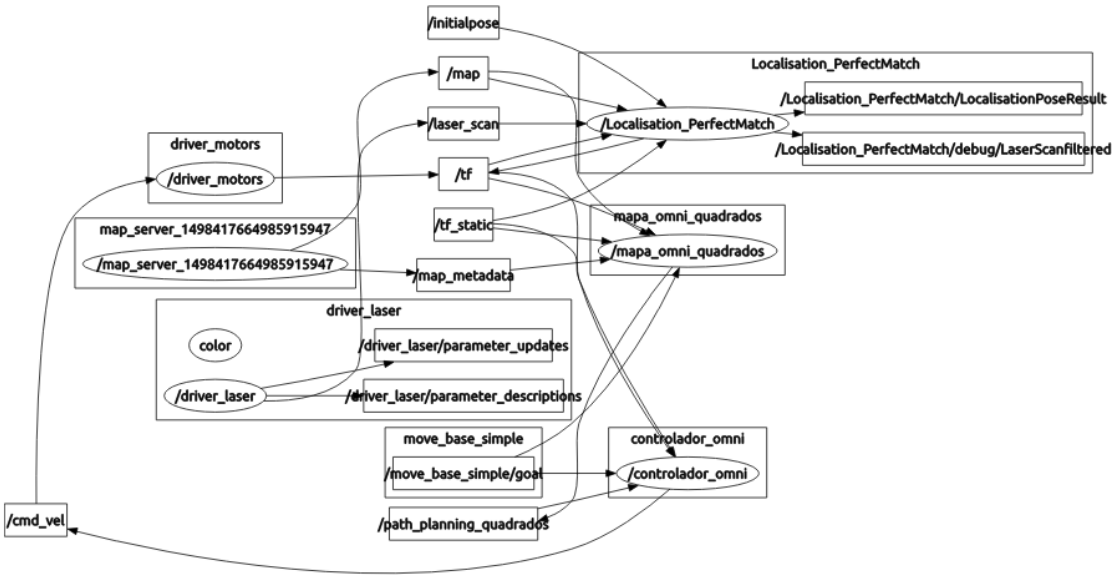


Figura 5.10: Nós e tópicos ativos durante o teste do controlador de trajetórias

A Figura 5.11 representa a trajetória pretendida e o caminho efetuado pelo robô, com uma velocidade máxima linear admissível de 0.3 m/s e velocidade de rotação máxima de $4°/s$. O erro associado à mesma, está ilustrado no gráfico da Figura 5.12, onde está representado o distância do robô ao ponto mais próximo da reta, sendo o erro máximo de 5.5cm. O robô efetuou a trajetória de aproximadamente 6 m em 1 minuto e 10 segundos.

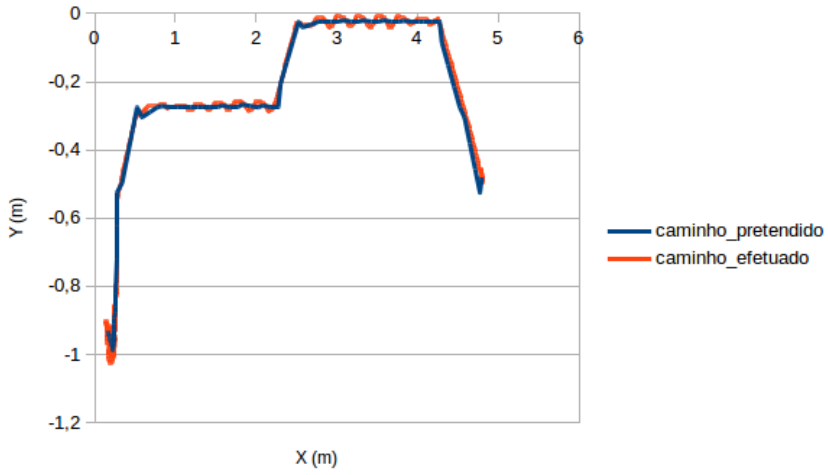


Figura 5.11: Posição x, y.

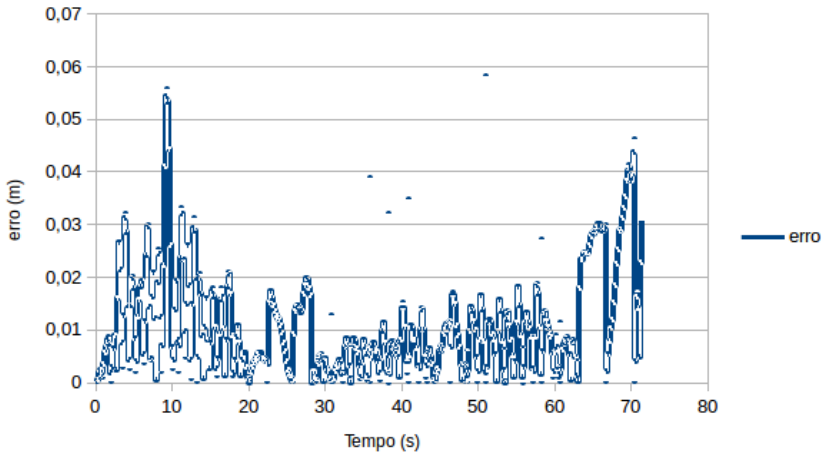


Figura 5.12: Erro da trajetória.

O controlador implementado foi validado através de testes em ambiente simulado, o que permitiu validar o seu funcionamento, mostrando este, valores de erro muito pequenos, o que resulta num seguimento de trajetórias muito próxima do ideal do robô em ambiente simulado. Quando analisado o comportamento do controlador em ambientes reais, este mostra-se eficaz para o robô seguir a trajetória desejada, com um erro máximo associado de aproximadamente 5.5 cm, quando o robô não necessita de rodar sobre si durante a trajetória. Quando este movimento rotacional é pretendido, o erro associado ao movimento do robô é maior.

Capítulo 6

Conclusão

Esta dissertação tem como foco principal a análise e estudo de métodos de planeamento de trajetórias para aplicação de um robô omnidirecional. Nesta vertente analisaram-se e compararam-se algumas metodologias que permitem a um robô omnidirecional planear caminhos em ambientes conhecidos, onde se pretende uma boa relação entre tempo de processamento e caminho definido. De forma a controlar o robô para este se moviemntar ao longo da trajetória definida é abordado um controlador não linear, denominado *Trajectory Linearization Control* (TLC).

Inicialmente, descreveu-se a forma como o robô e os obstáculos são definidos no espaço de trabalho durante o planeamento de um caminho. A geometria do robô é aproximada a um círculo, e como se trata de um robô omnidirecional, que tem a particularidade de efetuar movimento em todas as direções e a capacidade de rodar sobre si mesmo, o planeamento de trajetórias fica assim reduzido a um problema a 2 dimensões (x, y) . Uma vez que se assume que o robô é redondo, os obstáculos são aumentados de acordo com o seu tamanho, e o robô é tratado como um ponto que se pode mover em qualquer direção.

De seguida, foram analisados o comportamento da utilização de diferentes métodos de planeamento de caminhos. Foram assim implementados vários métodos de decomposição em células, aos quais é aplicado o algoritmo de pesquisa de grafos A-*star*. Os métodos de decomposição em células estudados nesta dissertação são: a decomposição em células fixas, decomposição em *quadtrees*, decomposição em *framed quadtrees* e, foi ainda sugerido um novo método de decomposição em células, denominada k-$FramedQuadtrees$, que resulta de uma fusão entre a decomposição em *quadtrees* e *framed quadtrees*. A análise comparativa entre os métodos implementados efetuada, permite concluir em que aplicações devem ser usados. Quando estamos perante um mapa de pequenas dimensões, a decomposição em células fixas demonstra bons resultados com um pequeno tempo de processamento e caminhos definidos muito próximos do considerado ideal. Esta decomposição tem ainda a vantagem da sua simplicidade de implementação, relativamente aos restantes métodos de decomposição em células abordados. A decomposição em *quadtrees*, tanto em mapas com pequenas dimensões, como em mapas de grandes dimensões e passagens estreitas, resultou sempre em tempos de processamentos inferiores aos restantes métodos, no entanto este resulta em caminhos mais longos, e com mudanças de direção abruptas, o que para aplicações na pre-

sença de humanos, pode transmitir pouca confiança. A decomposição em *framed quadtrees* gerou caminhos muito próximos do ideal, em ambos os mapas, no entanto o tempo de processamento associado é bastante mais elevado que os tempos associados aos restantes métodos. Para o método proposto, os coeficientes utilizados são selecionados em conformidade com o objetivo de obter um bom compromisso entre tempo de processamento e caminho determinado. Aquando de mapas de pequenas dimensões, os caminhos resultantes não se afastem do ideal, no entanto o tempo de processamento é ligeiramente mais elevado comparativamente à decomposição em células fixas. O método K-*Framed Quadtrees* tem a desvantagem, relativamente aos restantes métodos abordados, a elevada complexidade de implementação devido principalmente ao grande número de restrições para determinar os vizinhos de cada célula, pelo que, em mapas de pequenas dimensões, o método a utilizar deverá ser a decomposição em células fixas. Na presença de mapas de grandes dimensões e com passagens estreitas, este tem associado tempos de processamentos elevados, pelo que, o novo método torna-se a melhor abordagens nestes casos, permitindo obter caminhos próximos do ideal, com um menor tempo de processamento.

Posteriormente foi analisado os resultados associados ao controlador implementado TLC. Este foi analisado através de testes em ambiente simulado, o que permitiu validar o seu funcionamento, mostrando este, valores de erro muito pequenos, o que resulta num seguimento de trajetórias muito próxima do ideal por parte do robô em ambiente simulado. Quando analisado o comportamento do controlador em ambientes reais, este mostra-se eficaz para o robô seguir a trajetória desejada.

6.1 Trabalho Futuro

Para trabalho futuro, relativamente ao planeamento de trajetórias seria importante a implementação de um algoritmo de desvio de obstáculos de forma a alcançar uma navegação segura, sem a necessidade de o robô esperar que o obstáculo se afaste da trajetória pretendida.

Seria ainda interessante no planeamento de trajetórias ter em consideração a orientação do robô, uma vez que o mesmo não é circular, e assim tirar um maior partido da particularidade dos robôs omnidirecionais terem a capacidade de se movimentarem em todas as direções e ainda rodarem sobre si próprios. Assim, o robô poderia aceder a mais zonas estreitas, que com a aproximação feita nesta dissertação não seria possível.

Ao nível do controlo das trajetórias, depois do trabalho realizado neste projeto, é ainda necessário aperfeiçoar e realizar mais testes ao desempenho do mesmo, aplicando diferentes configurações e analisando o seu desempenho em trajetórias distintas, com o intuito de analisar a sua viabilidade em diversos tipo de trajetórias.

Referências

[1] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kentor, Wolfram Burgard, Lydia E. Kavraki, e Sebastian Thrun. *Principles of Robot Motion Theory, Algorithms, and Implementations.* Bradford Book, 2005.

[2] Roland Siegwart e Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Book, 2004.

[3] A. Yahja, A. Stentz, S. Singh, e B. L. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. Em *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 1, páginas 650–655 vol.1, Maio 1998. doi:10.1109/ROBOT.1998.677046.

[4] J. Antich, A. Ortiz, e J. Mínguez. A bug-inspired algorithm for efficient anytime path planning. Em *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, páginas 5407–5413, Oct 2009. doi:10.1109/IROS.2009.5354182.

[5] Y. S. Silveira e P. J. Alsina. A New Robot Path Planning Method Based on Probabilistic Foam. Em *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, páginas 217–222, Outubro 2016. doi:10.1109/LARS-SBR.2016.43.

[6] C. Ye, J. Chen, M. Chen, e L. Liu. A control approach of an omnidirectional mobile robot with differential wheels. Em *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*, páginas 1211–1216, Agosto 2015. doi:10.1109/ICMA.2015.7237658.

[7] www.euroc-project.eu. URL: http://www.euroc-project.eu/.

[8] A. Valero-Gomez, J. V. Gomez, S. Garrido, e L. Moreno. The path to efficiency: Fast marching method for safer, more efficient mobile robot trajectories. *IEEE Robotics Automation Magazine*, 20(4):111–120, Dec 2013. doi:10.1109/MRA.2013.2248309.

[9] P. Fiorini e Z. Shiller. Motion planning in dynamic environments using the relative velocity paradigm. Em *[1993] Proceedings IEEE International Conference on Robotics and Automation*, páginas 560–565 vol.1, May 1993. doi:10.1109/ROBOT.1993.292038.

[10] J. D. Jeon e B. H. Lee. Ellipse-based velocity obstacles for local navigation of holonomic mobile robot. *Electronics Letters*, 50(18):1279–1281, August 2014. doi:10.1049/el.2014.1592.

[11] B. S. Sandeep e P. Supriya. Analysis of fuzzy rules for robot path planning. Em *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, páginas 309–314, Setembro 2016. doi:10.1109/ICACCI.2016.7732065.

[12] Z. Lv e J. Cao. Path planning methods of mobile robot based on new neural network. Em *Proceedings of the 32nd Chinese Control Conference*, páginas 3222–3226, Julho 2013.

[13] N. Lu, Y. Gong, e J. Pan. Path planning of mobile robot with path rule mining based on GA. Em *2016 Chinese Control and Decision Conference (CCDC)*, páginas 1600–1604, Maio 2016. doi:10.1109/CCDC.2016.7531239.

[14] Jiang Zhao, Dingding Cheng, e Chongqing Hao. An Improved Ant Colony Algorithm for Solving the Path Planning Problem of the Omnidirectional Mobile Vehicle. *Mathematical Problems in Engineering*, 2016:e7672839, Setembro 2016. URL: https://www.hindawi.com/journals/mpe/2016/7672839/abs/, doi:10.1155/2016/7672839.

[15] S. A. Fadzli, S. I. Abdulkadir, M. Makhtar, e A. A. Jamal. Robotic Indoor Path Planning Using Dijkstra's Algorithm with Multi-Layer Dictionaries. Em *2015 2nd International Conference on Information Science and Security (ICISS)*, páginas 1–4, Dezembro 2015. doi:10.1109/ICISSEC.2015.7371031.

[16] W. Y. Loong, L. Z. Long, e L. C. Hun. A star path following mobile robot. Em *2011 4th International Conference on Mechatronics (ICOM)*, páginas 1–7, Maio 2011. doi:10.1109/ICOM.2011.5937169.

[17] A. Stentz. Optimal and efficient path planning for partially-known environments. Em *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, páginas 3310–3317 vol.4, May 1994. doi:10.1109/ROBOT.1994.351061.

[18] I. E. Paromtchik e U. Rembold. A practical approach to motion generation and control for an omnidirectional mobile robot. Em *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, páginas 2790–2795 vol.4, Maio 1994. doi:10.1109/ROBOT.1994.350916.

[19] Yong Liu, Xiaofei Wu, J. Jim Zhu, e Jae Lew. Omni-directional mobile robot controller design by trajectory linearization. Em *Proceedings of the 2003 American Control Conference, 2003.*, volume 4, páginas 3423–3428 vol.4, Junho 2003. doi:10.1109/ACC.2003.1244061.

[20] K. Kanjanawanishkul e A. Zell. Path following for an omnidirectional mobile robot based on model predictive control. Em *2009 IEEE International Conference on Robotics and Automation*, páginas 3341–3346, Maio 2009. doi:10.1109/ROBOT.2009.5152217.

[21] S. T. Kao, W. J. Chiou, e M. T. Ho. Integral sliding mode control for trajectory tracking control of an omnidirectional mobile robot. Em *2011 8th Asian Control Conference (ASCC)*, páginas 765–770, Maio 2011.

[22] P. H. Kim Khanh, Nguyen Thanh Trung, Phuc Thinh Doan, e Nguyen Hung. Trajectory tracking control of omnidirectional mobile robot using sliding mode controller. Em *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, páginas 1170–1175, Outubro 2013. doi:10.1109/ICCAS.2013.6704095.

[23] Veer Alakshendra, Shital S. Chiddarwar, e Abhishek Jha. Trajectory Tracking Control of Three-Wheeled Omnidirectional Mobile Robot: Adaptive Sliding Mode Approach. Em Dipak Kumar Mandal e Chanan Singh Syan, editores, *CAD/CAM, Robotics and Factories of the Future*, Lecture Notes in Mechanical Engineering, páginas 275–286. Springer India, 2016. DOI: 10.1007/978-81-322-2740-3_27. URL: http://link.springer.com/chapter/10.1007/978-81-322-2740-3_27.

[24] V. Alakshendra e S. S. Chiddarwar. A robust adaptive control of mecanum wheel mobile robot: simulation and experimental validation. Em *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, páginas 5606–5611, Outubro 2016. doi:10.1109/IROS.2016.7759824.

[25] A. Sheikhlar, A. Fakharian, H. Beik-Mohammadi, e A. Adhami-Mirhosseini. Design and Implementation of Self-Adaptive PD Controller Based on Fuzzy Logic Algorithm for Omni-Directional Fast Robots in Presence of Model Uncertainties. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 24(05):761–780, Outubro 2016. URL: http://www.worldscientific.com/doi/abs/10.1142/S0218488516500343, doi:10.1142/S0218488516500343.

[26] H. C. Huang, T. F. Wu, C. H. Yu, e H. S. Hsu. Intelligent fuzzy motion control of three-wheeled omnidirectional mobile robots for trajectory tracking and stabilization. Em *2012 International conference on Fuzzy Theory and Its Applications (iFUZZY2012)*, páginas 107–112, Novembro 2012. doi:10.1109/iFUZZY.2012.6409684.

[27] Hsu-Chih Huang. Inligent Motion Control for Omnidirectional Mobile Robots Using Ant Colony Optimization. *Applied Artificial Intelligence*, 27(3):151–169, Março 2013. URL: http://dx.doi.org/10.1080/08839514.2013.768877, doi:10.1080/08839514.2013.768877.

[28] J. Wang, S. A. Chepinskiy, A. J. Krasnov, B. Zhang, H. Liu, Y. Chen, e D. A. Khvostov. Geometric path following control for an omnidirectional mobile robot. Em *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, páginas 1063–1068, Agosto 2016. doi:10.1109/MMAR.2016.7575285.