FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Planeamento Simultâneo de Trajetórias para Múltiplos Robôs Autónomos num Ambiente Industrial

Daniel Filipe Barros Campos

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Doutor Paulo José Cerqueira Gomes da Costa

Co-orientador: Prof. Doutor Pedro Luís Cerqueira Gomes da Costa

31 de Julho de 2014

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES 2013/2014

A Dissertação intitulada

"Planeamento Simultâneo de Trajetórias para Múltiplos Robôs Autónomos num Ambiente Industrial"

foi aprovada em provas realizadas em 24-07-2014

o júri

Presidente Professor Doutor Aníbal Castilho Coimbra de Matos
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor José Luís Sousa de Magalhães Lima
Professor Adjunto do Departamento de Engenharia Eletrotécnica da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança

Professor Doutor Paulo José Cerqueira Gomes da Costa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - Daniel Filipe Barros Campos

Faculdade de Engenharia da Universidade do Porto

Resumo

Neste documento apresentam-se os diversos algoritmos desenvolvidos e implementados que procuram atuar em tempo real para planear trajetórias, permitindo o bom funcionamento de robôs em ambientes dinâmicos.

O planeamento de trajetórias é um problema transversal a múltiplas áreas de estudo, tendo sofrido diversos avanços nos últimos anos. No âmbito desta dissertação procura-se atuar sobre a área da robótica móvel ligada à indústria, visando controlar múltiplos robôs de forma simultânea.

Como tal é necessário ter presente que em ambientes industriais podem surgir obstáculos devido à presença de pessoas, outros robôs ou provocados pelo próprio robô. Assim prevê-se que para além do mapa conhecido e dos obstáculos estáticos, possam surgir objetos que não eram previstos e até mesmo dotados de movimento próprio. Desta forma o tempo de execução é crítico para poder obter as trajetórias em tempo real e replanear consoante a necessidade do momento, procurando sempre o caminho ótimo para demorar o menor tempo possível a deslocar-se do início para o destino.

Para tal nesta dissertação foram desenvolvidos e implementados algoritmos de planeamento de trajetórias e de controlo de movimentos rápidos e seguros, por forma a permitir a coordenação entre múltiplos robôs e replanear em tempo real consoante as necessidades. Desta forma foram criados processos de criação e expansão do mapa consoante um espaço bidimensional, tendo em conta as dimensões do robô a usar. Com isto obteve-se uma abordagem baseada em decomposição em células aproximadas com células fixas, a qual gera um grafo que é pesquisado recorrendo ao algoritmo A^*, com modificações para o tornar mais eficiente e garantir que continua perto da admissibilidade. Foi ainda desenvolvido um sistema de navegação baseado num controlo de eventos de forma hierárquica com recurso à aproximação dos caminhos por pequenas retas.

Além disso foi idealizado e implementado um algoritmo que permitisse o aumento da manobrabilidade entre múltiplos robôs com recurso a uma expansão do mapa consoante a orientação do robô, gerando um grafo de um espaço 3D para posterior pesquisa via A^*.

Por fim são apresentadas as plataformas de implementação desde o simulador usado ao robô prototipado desenvolvido, considerando sempre o ambiente da competição *Robot@Factory* como referência para trabalho.

Abstract

In this document are presented several algorithms developed and implemented looking towards solving a path planning problem, on the fly, allowing robots to move without collisions in dynamic work environments.

Path planning is a transversal problem to multiple areas of study, having suffered diverse advances in recent years. The scope of this dissertation is to act upon the subject of path planning for mobile robots in an industrial environment, aiming to control multiple robots simultaneously.

Industrial environments are full of obstacles due to presence of people, other robots or provoked by the robot itself. Thus besides the known map and static obstacles, moving objects that were not foreseen can appear. Therefore the execution time is critical to be able to get the trajectories on the fly and to change it according to the necessity of the moment, looking for the optimal path in order to reduce the time to reach the target.

For such, in this dissertation were developed and implemented quick and safe path planning and navigation control algorithms which allows changing trajectories according to the environment necessities and multiple robot coordination. To do this were created processes to generate the field map using a 2D space and expand the map according to the robot size. This way was developed an approximate cell decomposition based on fixed cells approach, which creates a graph of the map positions searched by a modified A^* algorithm with low processing time and granting the optimality of the path planning. For the navigation system was developed a hierarchical event control approaching the path to small line segments.

Moreover it was idealized and implemented an algorithm that allows a maneuverability increase between multiple robots resourcing to map expansion using the robot orientation to do it. This way was generated a graph to represent a 3D space posteriorly searched by A^*.

Finally are presented all the implementation platforms since the simulator used to the robot prototype built, always having in consideration the *Robot@Factory* competition as a work reference.

Agradecimentos

Ao Prof. Doutor Paulo Costa pela confiança, disponibilidade e apoio dados durante a realização da dissertação fulcrais para o sucesso da mesma. Ao Prof. Doutor Pedro Costa pelas contribuições dadas para o desenvolvimento do projeto.

Ao Senhor Jorge Barbosa por todo o suporte no trabalho, por todo o auxílio na adaptação a um ambiente diferente e pelo companheirismo que nos deu ao longo da dissertação.

A todos os amigos e companheiros de armas que estiveram comigo ao longo do meu percurso durante estes anos, por todos os momentos de diversão, de apoio, de trabalho e por todas as experiências que me proporcionaram. Pretendo deixar um especial bem haja aos meus irmãos emprestados Vitor Mota (Kanguru) e Diana Guimarães (Racer) por todo o tempo que passamos juntos, todos os momentos vividos, todas as brincadeiras na cadeira de rodas e por a união e laços criados que me seguiram desde o início e que de certeza continuarão. Ao Vitor Pinto com quem fiz o primeiro e o último trabalho do curso e muito me aturou, aos Andrés, Oliveira e Photogray, pelo "bullying"que me sujeitaram que múltiplas vezes me animou o dia e ao Nuno Moreira pois sem ti tinha despachado a parte escrita mais rápido, mas valeu a pena pelos momentos passados e pelo apoio que davas levo os teus (des)conselhos para a vida.

Aos meus pais, João Campos e Júlia Campos, um enorme agradecimento, sem eles eu não estaria aqui. Por garantirem o meu futuro, por todo o suporte e carinho dado agradeço do fundo do coração e tudo farei para fazer valer a pena. Por fim ao meu irmão, Hugo Campos, pelos momentos de distração e descanso que me cedeu que bem me ajudou a descomprimir.

Daniel Filipe Barros Campos

"If you do not know where you are going, any road will take you there."

Lewis Carroll, *Alice in Wonderland*

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

2D	Bidimensional
3D	Tridimensional
AGV	*Automated Guided Vehicle*
CAD	*Computer-Aided Design*
$C_{ostáculos}$	Configuração dos obstáculos
DC	*Direct current*
LTG	*Local Tangent Graph*
PSO	*Particle Swarm Optimization*
PRM	*Probabilistic Roadmap*
PWM	*Pulse-Width Modulation*
RGB	Sistema de cores aditivas formado por Vermelho, Verde e Azul

Capítulo 1

Introdução

Este documento traduz todo o trabalho realizado na unidade curricular de Dissertação e descreve todo o processo seguido ao longo do Semestre para a concretização do projeto de final de curso no âmbito do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto.

Ao longo deste primeiro capítulo serão introduzidos o contexto e a motivação por detrás do tópico desta dissertação, os objetivos da mesma e a estrutura deste documento.

1.1 Contexto e Motivação

Atualmente o planeamento de trajetórias não se encontra restrito a robôs móveis a desviarem-se de obstáculos, tendo expandido para outras aplicações, o que causou um maior desenvolvimento no estudo deste tema. Entre estas áreas de aplicação encontram-se animação digital, *software* de vídeo-jogos, procedimentos cirúrgicos, CAD para arquitetura, entre outros, sendo que algumas destas áreas já se encontrariam referenciadas como impulsionadoras do progresso do planeamento de trajetórias por *Latombe*, 1999 [1].

Dos exemplos anteriormente dados, os sistemas gráficos, como são o caso das animações e dos jogos de computador, constituem áreas em constante melhoria, devido à exigência do público que procura um ambiente visualmente apelativo e mais realista nas suas movimentações, levando a que a abordagem a toda esta problemática constitua uma ponte de desenvolvimento para as restantes. Assim diversos trabalhos de investigação têm sido realizados neste âmbito, por exemplo por *Fischer et al*, 2009 [2], para ambientes virtuais multi-agente, *Chang*, 2013 [3], planeamento de trajetórias híbrido otimizado, e no caso de animação, para definir o movimento das personagens, o trabalho de *Lau et al*, 2005 [4]. No caso dos procedimentos cirúrgicos hoje em dia já existem diversos robôs que auxiliam o trabalho dos cirurgiões, como os casos do Zeus (Computer Motion Inc., Santa Barbara, CA) e do Da Vinci (Intuitive Surgical Inc., Mountain View, CA) apresentados por *Lanfranco et al*, 2004 [5].

Todas estas áreas, jogos, animação, procedimentos cirúrgicos, etc., auxiliaram o desenvolvimento do conhecimento sobre planeamento de trajetórias, abordando novos ângulos, nomeadamente ambientes dinâmicos e com possibilidade de múltiplos agentes, que podem ser extrapolados para a robótica num ambiente industrial, por forma a automatizar e otimizar o funcionamento dos robôs.

Assim olhando agora para o ambiente industrial poder-se-á assumir que, tal como as áreas acima referidas, se trata de um meio dinâmico, em que constantemente surgirão diversos obstáculos, sejam eles criados por outros robôs, humanos ou materiais que possam eventualmente cair nos corredores ou estejam pontualmente lá armazenados, sendo que estes serão adicionados aos obstáculos que se encontram estáticos e são conhecidos. Isto obriga a uma operação em tempo real que permita planear um caminho e que, em caso de necessidade, possa sofrer um replaneamento, visando sempre resultados ótimos ou bastante próximos disso, tendo sempre em conta o tempo de resposta do sistema, o qual se torna crítico para o seu sucesso para chegar à meta.

Além disso este meio tende ao recurso crescente de veículos totalmente autónomos, tentando escapar aos AGVs usuais [6], procurando um mapa mais amplo e uma maior independência, não sendo necessário a inserção de trajetórias pré-programadas nem de marcadores externos, o que contudo conduz para a necessidade de coordenar os caminhos efetuados pelos vários robôs intervenientes nas missões das quais estão incumbidos.

Desta forma, na presente dissertação será realizado um estudo de um algoritmo que planeie trajetórias em tempo real num ambiente industrial com mapa conhecido e dinâmico, que permita o planeamento simultâneo para múltiplos robôs. Tendo ainda que evitar colisões com os obstáculos (dinâmicos e estáticos), permitindo replanear as trajetórias e procurando sempre a otimilidade ou um resultado bastante próximo.

1.2 Objetivos

O objetivo principal desta dissertação é o desenvolvimento e implementação de algoritmos que permitam o planeamento de trajetórias para robôs móveis em tempo real, num ambiente industrial dinâmico com mapa conhecido, evitando ao máximo colisões e procurando os resultados ótimos ou bastante próximos disso. Outro objetivo será o de efetuar este controlo para múltiplos robôs em simultâneo.

É de frisar que estas metodologias são aplicadas e testadas num robô móvel, desenvolvido com vista a ser passível de participar na competição *Robot@Factory*, que decorre no Festival Nacional de Robótica, Robótica 2014.

1.3 Estrutura da Dissertação

Para além da introdução, este documento contém mais oito capítulos.

No capítulo 2 é descrito o estado da arte relativo às metodologias atualmente existentes para o planeamento de trajetórias, desde a definição da representação dos obstáculos, passando por os algoritmos existentes para planeamento de trajetórias e a introdução de algoritmos de pesquisa.

No capítulo 3 é formulado o problema, enquadrando no âmbito industrial e da competição *Robot@Factory* e definindo ainda as metodologias implementadas para atuar sobre a problemática em causa.

No capítulo 4 encontra-se a descrição do ambiente de implementação, nomeadamente do simulador usado, do robô prototipado e construido e dos atuadores existentes no mesmo.

No capítulo 5 descrevem-se os métodos utilizados para calcular as trajetórias para o robô executar, abordando o algoritmo de pesquisa usado e os métodos de construção em grafo desenvolvidos.

No capítulo 6 identificam-se as alterações feitas ao planeamento de trajetórias, introduzindo melhorias e modos de funcionamento mais adaptáveis ao ambiente.

No capítulo 7 explica-se o sistema de navegação desenvolvido para realizar os percursos pretendidos entre pontos no campo.

No capítulo 8 apresentam-se os resultados obtidos e a respetiva discussão dos valores obtidos.

No capítulo 9 discute-se a satisfação dos objetivos propostos e trabalho futuro sobre o qual se pode atuar futuramente.

Capítulo 2

Estado da Arte

Neste capítulo serão apresentadas as metodologias existentes para planeamento de trajetórias. Inicialmente procede-se a uma breve introdução à temática, partindo para a introdução de conceitos sobre Configuração dos Obstáculos. Seguidamente expõem-se diversos algoritmos de planeamento de trajetórias, e, por fim, descreve-se algoritmos de pesquisa.

2.1 Introdução

Apesar de nesta dissertação ser usada a expressão planeamento trajetórias, devido a ser reconhecida mais facilmente no português corrente, na realidade a terminologia mais correta seria planeamento do caminho. Sendo que o planeamento de caminhos é definido como o percurso definido geometricamente ou matematicamente entre os pontos inicial e final que evita colisões. Já o planeamento de trajetórias representa esse caminho em função do tempo, ou seja determina qual o ponto em que se deve de encontrar para cada instante de tempo, o que no âmbito deste problema não é necessário.

Ao longo deste capítulo vai-se discutir métodos para atacar o problema desta dissertação. Para tal, ainda antes de executar a trajetória, tem-se de definir o ambiente e a sua configuração para termos a representação tanto do espaço livre como do espaço obstruído. De seguida procede-se à implementação de algoritmos de planeamento da trajetória a efetuar, sendo que alguns deles geram grafos que necessitam de algoritmos de pesquisa, por forma a obter a solução.

Assim para este processo existem diversos parâmetros a ter em conta sendo de seguida apresentados alguns, [7, 8]:

- Tipo de otimização a realizar, se se pretende otimizar o tamanho da trajetória, o tempo de execução ou a energia consumida
- A complexidade espacial, pode não haver memória suficiente para executar o algoritmo
- Se é um algoritmo completo, isto é retorna sempre um resultado
- Tipo de obstáculos, se são estáticos ou dinâmicos ou até se são deformáveis

- A complexidade computacional do algoritmo, deve ser o mais reduzida possível, usando, sempre que permitido, algoritmos simples, por exemplo com pouco recurso à biblioteca de matemática, já que esta tem um tempo de execução grande.

Além destes parâmetros deve-se ter em conta o facto do robô ser holonômico, é capaz de se mover em qualquer direção, ou não-holonômico, apresenta restrições de movimento. Isto afeta o percurso do robô, pois pode limitar os movimentos que o robô pode fazer. Assim deve-se ter em atenção o tipo de locomoção a usar, nomeadamente:

- Omnidirecional - podem ser utilizadas três ou quatro rodas para a tração e é holonômica, logo não tem restrições de movimento, contudo é pouco implementado em grande escala devido à complexidade de construção e aos custos associados [9].

- Diferencial - tem tração em duas das rodas e é não-holonômico, não pode deslocar-se na direção do eixo das rodas, isto implica ter restrições nas trajetórias quanto a movimentos laterais, por exemplo ter de rodar primeiro e só depois deslocar-se.

- *Ackerman* - recurso a quatro rodas com configuração tipo carro e é não-holonômico, não pode deslocar-se lateralmente e tem um raio de curvatura mínimo, não podendo, por isso, rodar sobre si próprio, implica restrições de trajetórias, como por exemplo para pontos que estejam a uma distância menor que o raio de curvatura terá de realizar diversas manobras de mudança de direção [9].

2.2 Configuração dos Obstáculos

A configuração dos obstáculos, $C_{obstáculos}$, representa configurações do robô, parâmetros dos graus de liberdade, que conduzem a colisões com obstáculos e é tipicamente calculada através de um dos sete métodos seguidamente apresentados, referenciado por *Hwang et al.* em 1992 [10].

2.2.1 Método do ponto de avaliação

Este método é o mais simples e ineficaz dos sete, baseando-se em colocar o robô na configuração e determinar se este interseta algum obstáculo. Caso haja interseção considera-se que essa configuração pertence a $C_{obstáculos}$.

2.2.2 Método das diferenças de *Minkowski*

Este método baseia-se na função $Mdiff(obstáculo, robô)$. Sendo A o conjunto de pontos que representam o obstáculo e B os pontos que representam o robô, as diferenças de *Minkowski* são calculadas através de $Mdiff(A,B) = \{a-b | a \in A, b \in B\}$, considerando-se o robô um corpo rígido sem rotação, este representa o $C_{obstáculos}$.

2.2.3 Método das equações de fronteira

Este método é de elevada complexidade e define o $C_{obstáculos}$ através das equações fronteira, sendo estas equações matemáticas que indicam quando há contacto entre robô e obstáculo.

2.2.4 Método da agulha

Neste método todos os parâmetros de configuração são dados como inalteráveis à exceção de um. Este parâmetro que se mantém variável é calculado de forma a obter todos os valores que levam a que o robô entre em contacto com o obstáculo. Tipicamente estes valores são calculados usando equações de fronteira.

Por fim o $C_{obstáculos}$ é representado como um conjunto de intervalos discretos.

2.2.5 Método de varrimento do volume

Este método constrói uma área em que o robô pode deslocar-se sem risco de colisão, recorrendo à fixação de um parâmetro de configuração do robô e variando os restantes, guardando o valores que não levam a colisão. Este processo é repetido para todos os parâmetros.

Por isso este método é computacionalmente pesado, sendo preferível para robôs com poucos graus de liberdade.

2.2.6 Método de *Template*

Este método determina o $C_{obstáculos}$ quanto a características simples do mundo, tipicamente pontos e linhas, sendo que estas característica e o $C_{obstáculos}$ constituem um *template* . A forma e a posição do *template* pode ser parametrizado pela posição do obstáculo.

O $C_{obstáculos}$ é representado como a união das formas simples que foram obtidas.

2.2.7 Método baseado em Jacobianas

O Jacobiano J do robô é a matriz que relaciona o deslocamento, dx, de um ponto do robô com a sua variação na configuração, dq. Assim será representado por $dx = J(q) * dq$. Tendo uma configuração q, o valor máximo de $|J(q)|$, sobre todos os pontos no robô, é denominado $B(q)$.

Se for definida a distância negativa D^- entre dois objetos sobrepostos como a distância mínima para os separar, então pode-se calcular a esfera de raio $D^-/B(q)$ que define o $C_{obstáculos}$.

2.3 Métodos de Planeamento de Trajetória

Ao longo dos anos diversos algoritmos de planeamento de trajetórias foram idealizados, tendo vários deles aplicações viáveis em diversos problemas, mas outros ou são demasiado complexos ou são inviáveis, pois são pouco eficazes.

Uma vez que se procuram soluções para ambientes industriais, logo dinâmicos, nesta secção procede-se à descrição dos seguintes métodos com utilização nestes meios, referindo algumas vantagens e desvantagens e, ainda, algumas aplicações dos mesmos:

- Algoritmos Bug
- Campo potencial
- Otimização de enxame de partículas
- Roadmap
- Decomposição em células

2.3.1 Algoritmo Bug

Os algoritmos Bug são formas de evitar obstáculos, que têm como ideia base seguir o contorno do obstáculo até este deixar de afetar a trajetória. Estes têm como objetivo deslocar do ponto partida (*start*, *S*) até ao ponto alvo (*goal*, *G*), em ambientes em que o mapa é desconhecido, [8], e com a menor quantidade de dados globais possíveis ou seja tem como recurso para se localizar e identificar o ambiente que o rodeia dados provenientes de sensores.

Os algoritmos Bug tem sofrido diversas modificações, sendo de seguida apresentados os algoritmos inicialmente produzidos, *Bug 1* e *Bug 2*, e o algoritmo *Tangent Bug*, que se trata de uma evolução com implementação prática.

2.3.1.1 Bug 1

O algoritmo *Bug 1*, é um algoritmo completo, isto é devolve sempre um resultado, logo sempre que encontra um percurso viável retorna o resultado ou, caso não encontre, retorna a impossibilidade de prosseguir para *G*. Este método não garante uma trajetória ótima entre dois pontos. Apesar disso este algoritmo garante que o robô não volta para o mesmo obstáculo.

Este algoritmo tem os seus passos descritos em [8, 11], tendo como base:

1. Inicializar deslocamento em direção ao destino, *G*
2. Se encontrado um obstáculo, contorna-se todo o obstáculo
3. Depois de contornar totalmente o obstáculo verificar se *G* é atingível, se for dirigir-se para o ponto mais perto do destino, L_i em que $i = 1, 2, 3, ..., n_{obstáculos}$, se não for possível chegar ao destino terminar processo
4. Recomeçar movimento para *G*.

Na Figura 2.1, retirada de [9], encontra-se um exemplo deste algoritmo, que como se pode ver é pouco eficaz uma vez que necessita de contornar o obstáculo por forma a descobrir o ponto de saída, L_i, do obstáculo.

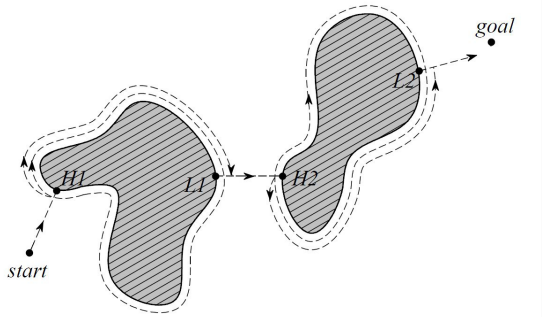


Figura 2.1: Exemplo de *Bug 1*

Este algoritmo, apesar de ser simples de desenvolver, não foi implementado, uma vez que é pouco eficiente devido aos percursos de grande tamanho que tem de realizar, pois tem de contornar totalmente cada obstáculo encontrado.

2.3.1.2 Bug 2

O algoritmo *Bug 2*, tal como o *Bug 1*, é um algoritmo completo e não ótimo, contudo constitui um melhoramento do apresentado em 2.3.1.1, uma vez que já não tem necessidade de contornar todo o obstáculo.

Este algoritmo tem os seus passos descritos em [8, 11], tendo como base:

1. Definir um segmento de reta imaginário, M, o entre os pontos S e G
2. Inicializar deslocamento em direção ao destino, G, seguindo o segmento de reta, M
3. Se encontrar obstáculo seguir o contorno do mesmo até:
   (a) Atingir o ponto alvo.
   (b) Encontrar um ponto pertencente a M, L_i, em que $i = 1,2,3,...,n_{obstáculos}$. Continuar deslocamento ao longo de M até chegar a G.
   (c) O robô voltar ao ponto onde estava, tornando o alvo inatingível. Terminar o processo.

Na Figura 2.2, retirada de [9], encontra-se um exemplo deste algoritmo, mostrando, tal como já foi referido, que não necessita de contornar o obstáculo antes de prosseguir.

Em 2012 *Al-Haddad et al.* [12], sugeriram a implementação de dois algoritmos para planear trajetórias numa cadeira de rodas o *Tangent Bug* que será exposto seguidamente e o *Bug2*. Relativamente ao segundo, tanto na simulação como na implementação no protótipo, concluíram que este não gera o caminho mais curto e rápido, logo não obtém o caminho ótimo, contudo tem a vantagem de ser de fácil implementação e eficaz na resolução de um problema que tem em conta um ambiente desconhecido e dinâmico.

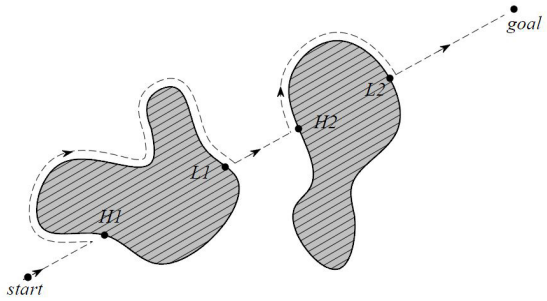


Figura 2.2: Exemplo de *Bug 2*

2.3.1.3 Tangent Bug

O algoritmo *Tangent Bug* foi apresentado e implementado em 1998 por *Kamon et al.* em [13], surgindo como uma melhoria relativamente ao dois previamente apresentados neste documento, utilizando os sensores de distância para construir um grafo com os vértices atingíveis do obstáculo, o LTG, procurando, posteriormente, o nó ótimo, ou seja o que tem a menor distância relativamente ao nó objetivo. Tal como afirmado em [11], por vezes podem ser encontrados mínimos locais que fazem com que o robô se desvie do alvo para ser possível atingi-lo.

Segundo *Siegwart et al.*, em [9], este algoritmo permite não só um movimento mais eficiente, como também permite seguir atalhos enquanto contorna obstáculos. Afirmando, também, que em ambientes simples este algoritmo aproxima caminhos globalmente ótimos.

Na Figura 2.3, retirada de [14], é apresentado um exemplo deste método de planeamento. Em que em A) o robô começa em x_0, executa o LTG e segue o caminho localmente ótimo, em B) O robô atinge x_1, volta a calcular o LTG e em C) O robô move-se à volta do obstáculo até x_3, até que o LTG veja o destino.

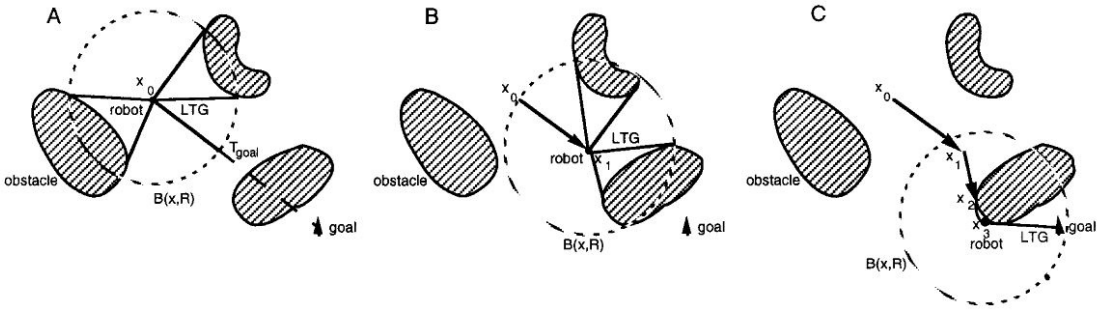


Figura 2.3: Exemplo de *Tangent Bug*.

Em 1998 *Laubach et al.* [14] expõem o recurso a uma extensão deste algoritmo para controlar a trajetória, baseando-se em sensorização, de um *microrover*, o *Rocky7*. Esta extensão deve-se

maioritariamente a que, apesar das boas propriedades locais para gerar caminhos localmente ótimos, nem os *rover* nem os seus sensores não são omnidirecionais, sendo necessário adaptar o algoritmo às restrições existentes.

Como referido em 2.3.1.2, em 2012 *Al-Haddad et al.* [12] sugeriram a implementação do *Tangent Bug* para planear trajetórias numa cadeira de rodas. Este algoritmo foi apenas simulado, não sendo testado no protótipo devido ao elevado custo do sensor laser, contudo os resultados obtidos apontam para uma maior rapidez a chegar ao destino, logo um caminho mais curto.

2.3.2 Campos de potencial

O método de campos de potencial para planeamento de trajetórias baseia-se na criação de um campo, ou gradiente, ao longo do mapa que direciona o robô de uma posição inicial, *S*, para a meta a atingir, *G*, [9]. Como se pode ver em [8], o robô é visto como um ponto que se desloca num ambiente com obstáculos, em que este é visto como uma carga positiva, que se desloca em direção a uma carga negativa, a meta *G*, que tem um efeito atrativo no veículo. Por forma a evitar os obstáculos, estes têm uma propriedade repulsiva, sendo por isso vistos como cargas positivas. Estas combinações de campos atrativos e repulsivos vão criar o campo de potencial artificial. Idealmente este tem um mínimo único no ponto de destino.

Apesar de ser amplamente usado devido à sua facilidade de implementação, [9], este algoritmo apresenta limitações devido ao formato e tamanho do obstáculo que podem gerar mínimos locais, por exemplo na Figura 2.4, pode surgir problemas com objetos côncavos. Isto pode conduzir a uma oscilação entre dois pontos perto do obstáculo, fazendo com que o algoritmo deixe de ser completo.

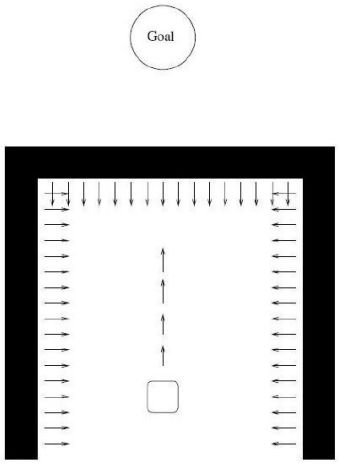


Figura 2.4: Mínimo local no algoritmo campo de potencial

Os modelos matemáticos por trás deste algoritmo, nomeadamente a de função potencial, de cálculo dos potenciais atrativos e repulsivos, assim como das forças dos mesmos podem ser consultados em [8, Secção 2.3.4 Campos de Potencial].

Na Figura 2.5, retirada de [15], encontra-se um exemplo deste algoritmo.

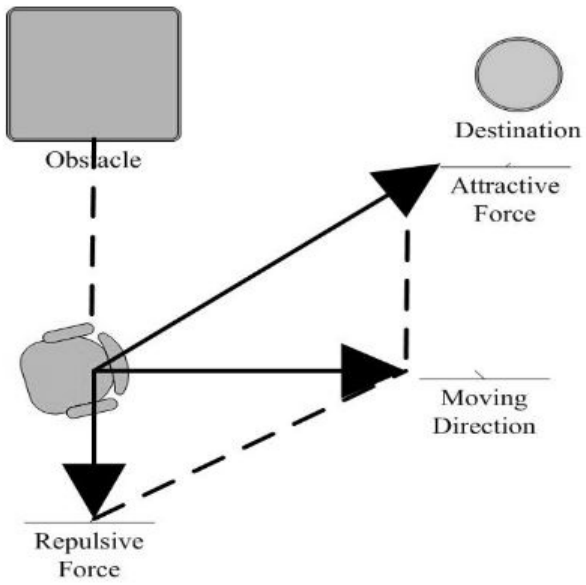


Figura 2.5: Exemplo de Campo de Potencial

Em 2002 *Ge et al.* [16] implementaram uma adaptação ao método de campos de potencial que permitisse o planeamento de trajetórias em ambientes com obstáculos estáticos e dinâmicos, tentando evitar problemas de mínimos locais. Este método foi implementado num robô que se movimentava num ambiente com alguns obstáculos fixos e outros móveis.

Em 2011 *Mohamed et al.* [17] apresentaram um algoritmo de planeamento de trajetórias para múltiplos robôs em ambiente dinâmico com recurso ao método de campos de potencial, utilizando também um melhoramento do *Tangent Bug*, apresentado em 2.3.1.3, para corrigir os problemas encontrados com os mínimos locais, contudo apresenta trajetórias não ótimas, uma vez que ao prender-se ao *Tangent Bug* para se desviar de alguns obstáculos, o robô vai seguir os seus contornos.

2.3.3 Otimização de enxame de partículas

Este algoritmo procura encontrar a melhor solução baseando-se no comportamento de enxames e cardumes [18].

Para tal rege-se num princípio básico em que quanto mais perto um agente se encontrar do destino maior valor de *fitness* vai produzir, caso contrário este valor é mais baixo.

Assim o algoritmo segue os seguintes passos [18]:

1. Iniciar a população com posições e velocidades aleatórias em torno da posição inicial do robô
2. Avaliar o *fitness* de cada partícula do enxame
3. Para cada partícula fazer:

(a) Encontrar o melhor valor de *fitness* para cada partícula (p_{best})

(b) Encontrar o melhor p_{best}, ou seja o maior valor de *fitness* de todas as partículas, g_{best}, logo o que se encontra mais perto do destino

(c) Atualizar a velocidade da partícula através da equação:
$V_i(t+1) = w \cdot V_i(t) + c_1 \cdot r_1 \cdot [\hat{x}_i(t) - x_i(t)] + c_2 \cdot r_2 \cdot [\hat{g}_i(t) - x_i(t)]$

(d) Atualizar a posição da partícula através da equação:
$x_i(t+1) = x_i(t) + V_i(t+1)$

Sendo os parâmetro das equações: w um coeficiente de inércia, $V_i(t+1)$ e $x_i(t+1)$ a velocidade e a posição atual da partícula, respetivamente, r_1 e r_2 valores aleatórios de uma distribuição uniforme entre [0;1], c_1 e c_2 constantes de aceleração, $\hat{x}_i(t)$ o p_{best} atual e $\hat{g}_i(t)$ o g_{best} atual.

Desta forma cada partícula é acelerada em direção às localizações do seu p_{best} e do g_{best}, procedendo-se os passos referidos até a localização do g_{best} coincidir com o destino.

Em 2006 *Wang te al.* [19] utilizaram o PSO para planeamento de trajetórias, com necessidade de desvio de obstáculos dinâmicos aplicado em futebol robótico, obtendo resultados que indiciam que o algoritmo é viável para a aplicação em ambientes dinâmicos, uma vez que é de baixa complexidade e converge rapidamente para a solução, permitindo ainda fugir às trajetórias localmente ótimas passando a ser globalmente ótimas.

Em 2010 *Masehian et al.* [20] sugeriram um algoritmo baseado em PSO para tentar otimizar múltiplos objetivos, nomeadamente procurar o caminho mais curto e mais suave, obtendo resultados que validam a sua proposta.

2.3.4 Decomposição em células

Este método representa o ambiente em células, às quais se associam valores indicativos se estão livres ou obstruídas, construindo com elas um grafo, em que cada célula é um nó. Assim para encontrar a solução é necessário posteriormente percorrer o grafo com um algoritmo de pesquisa.

Para realizar este algoritmo têm de inicialmente atribuir os pontos inicial e de destino às respetivas células e posteriormente encontrar as células que as unem. Esta decomposição de células pode ser realizada de forma exata, obtendo a representação do meio em conformidade com o ambiente real, ou de forma aproximada, isto é que não representam de forma exata o espaço real.

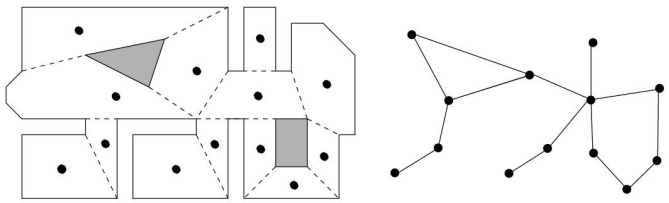
De seguida apresentam-se os métodos de decomposição em células exatas e em células aproximadas.

2.3.4.1 Decomposição em células exatas

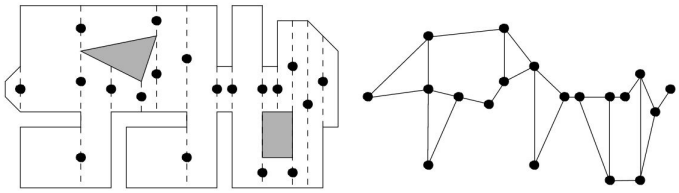
Na decomposição em células exatas o espaço é dividido em células que representam os locais livres ou obstruídos, sendo representações do meio real. Estas são feitas recorrendo a elementos de geometria simples, sendo os mais usados os polígonos convexos e os trapézios.

A divisão em polígonos convexos é efetuada definindo os vértices da célula, como os vértices dos obstáculos. A divisão em trapézios é realizada traçando linhas verticais nos vértices dos obstáculos, sendo estas as arestas das células.

Na Figura 2.6a, retirada de [21], encontra-se um exemplo de divisão em polígonos convexos e na Figura 2.6b, retirada de [21], encontra-se um exemplo de divisão em trapézios.



(a) Divisão em polígonos complexos (esquerda) e representação em grafo (direita)



(b) Divisão em trapézios (esquerda) e representação em grafo (direita)

Figura 2.6: Exemplos de decomposição em células exatas

2.3.4.2 Decomposição em células aproximadas

Na decomposição em células aproximadas o espaço é dividido em células que representam três estados livres, ocupadas ou parcialmente ocupadas. As células normalmente são representadas por formas geométricas simples, como quadrados. Neste método os tamanhos definidos têm de ser bem planeados, pois pode fazer com que existam caminhos que, por má escolha dos tamanhos, sejam ignorados.

Para este tipo de decomposição os mais métodos mais usados são o de células fixas e o de *quadtree*.

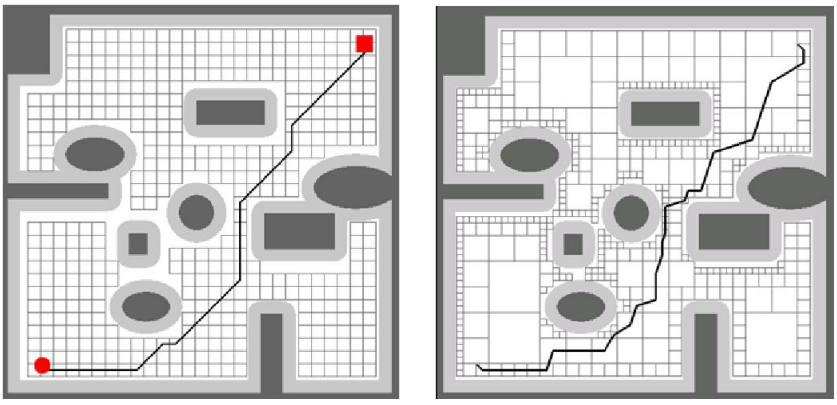
Na decomposição em células fixas as células têm um tamanho predefinido e o ambiente divide-se em células com esse tamanho.

Em 2011 *Costa* [8] apresenta uma solução para representação do meio baseado em células fixas, aplicado ao futebol robótico, ou seja um ambiente dinâmico com planeamento para vários robôs, procedendo a modificações posteriores aos pesos de cada célula por forma a otimizar os processos. Assim obteve resultados experimentais que permitiram planear a trajetória de forma segura, reduzindo o tempo de percurso e sem grandes custos nos tempos de execução.

Na decomposição em *quadtree* executa-se dividindo o espaço de configuração do robô em 4 células iguais e sempre que uma célula tiver um valor que não seja livre, voltar a dividir em 4 até se atingir o tamanho mínimo definido para uma célula.

Em 2007 *Arney* [22] apresenta um método de decomposição de células baseado em *quadtree* mais eficiente que reduz o risco de não encontrar solução onde esta existe e evitando um afunilamento associado a este método e funcionando para ambientes dinâmicos.

Na Figura 2.7a, retirada de [8], encontra-se um exemplo de divisão em células fixas e na Figura 2.7b, retirada de [8], encontra-se um exemplo de divisão em *quadtree*.



(a) Decomposição em células fixas (b) Decomposição em *quadtree*

Figura 2.7: Exemplos de decomposição em células aproximadas

2.3.5 *Roadmap*

Estes métodos, tal como o de decomposição em células, produzem um grafo que necessita de ser pesquisado usando algoritmos apropriados para tal. Os algoritmos de *Roadmap*, tal como referido em [23], representa as ligações no espaço livre do robô sobre a forma de uma rede de curvas uni-dimensionais, sendo assim o planeamento de trajetórias reduzido a um problema de ligações entre as configurações iniciais e finais do *Roadmap*.

De seguida serão apresentados métodos de *Roadmap*, usando *Visibility Graph*, diagramas de *Voronoi* e de *Roadmap* probabilísticos (PRM).

2.3.5.1 *Visibility Graph*

Em [9] afirma-se que este método é de uso comum na robótica móvel devido à sua simplicidade de implementação, consistindo na união de todos os vértices que estão visíveis entre si, estando incluídos os pontos inicial e de destino. Assim as retas que unem esses vértices, sem existir obstrução, são as menores distâncias entre eles, sendo necessário de seguida pesquisar o grafo produzido para encontrar os percursos que unem o início ao destino.

Este método é completo, contudo tem como principal problema o facto de que ao definir os caminhos mais curtos compromete a segurança do robô, passando muito perto do obstáculo, propiciando colisões. Na Figura 2.8, retirada de [9], encontra-se um exemplo deste método.

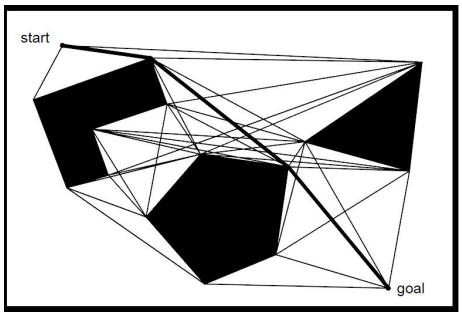


Figura 2.8: Exemplo de *Visibility Graph*

2.3.5.2 Diagrama de *Voronoi*

Este algoritmo é completo e, ao contrário do anteriormente apresentado, procura maximizar a distância aos obstáculos, isto é gera um conjunto de pontos equidistantes de dois ou mais obstáculos. Assim o espaço fica dividido em regiões com um único obstáculo.

Este método cria caminhos seguros, contudo têm elevado comprimento, isto é não fornece caminhos ótimos. Além disso, como se afasta o máximo dos obstáculos, pode causar perda de informação em sensores de curto alcance que estejam a ser usados no robô. Na Figura 2.9, retirada de [9], encontra-se um exemplo de um diagrama de *Voronoi*.

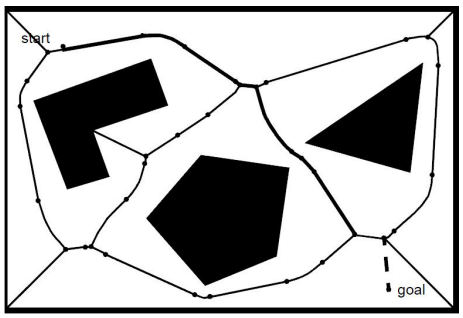


Figura 2.9: Exemplo de diagrama de *Voronoi*

Em 2013 *Ok et al.* [24] apresentaram um algoritmo com dois níveis para robôs a deslocarem-se em ambientes incertos, usando este método no nível mais alto para garantir que existe um caminho desde o inicio até ao destino, obtendo assim um algoritmo completo. Além disso conseguem fugir a mínimos locais e terminar o método quando não existe um caminho para o destino sem colisões.

2.3.5.3 *Roadmap* probabilísticos (PRM)

Neste método constrói-se o *roadmap* com métodos probabilísticos, sendo que este algoritmo encontra-se dividido em duas fases, uma de aprendizagem, onde se constrói o *roadmap* e de seguida uma fase de investigação, onde se procura o percurso no *roadmap* que une o início ao destino.

Na fase de aprendizagem realizam-se 3 processos. O primeiro é construir o *roadmap*, gerando uma configuração aleatória no espaço livre que é adicionada como nó, seguido de iterativamente selecionar os nós vizinhos (que se encontram dentro de uma distância máxima), verificando se há ligação entre eles, se existir juntar ao *roadmap*, repetindo até se ter um grafo suficientemente preenchido. No segundo expande-se o mapa nas regiões em que existem poucos espaços livres, logo com poucas ligações entre nós. No terceiro processo procede-se à remoção dos nós redundantes.

Na fase de investigação também decorrem 3 passos, primeiro encontrar um caminho livre do ponto inicial até a um nó do *roadmap*, de seguida fazer o mesmo para o ponto de destino e, por fim, encontrar um caminho que una o nó ligado ao ponto inicial ao nó ligado ao ponto de destino.

Este método não é completo, isto é pode nunca terminar. Para além disso em passagens estreitas é complicado encontrar caminhos.

Em 2004 *Clark* [25] sugeriu uma plataforma para coordenação de múltiplos robôs em ambientes desconhecidos e dinâmicos, usando um PRM modificado para planear as trajetórias, capaz de de reduzir o tempo de processamento do algoritmo para ser passível de ser executado em tempo real.

Em 2013 *Zhang et al.* [26] sugeriram um método baseado em PRM para robôs móveis em ambientes dinâmicos, procurando caminhos sem colisão obtendo resultados em que demonstram que encontra um percurso mais curto de forma rápida.

2.4 Algoritmos de Pesquisa

Por fim serão apresentados algoritmos de pesquisa em grafos ou em árvores. Estes algoritmos tornam-se necessários para complementar diversos dos métodos de planeamento de trajetórias anteriormente indicados, uma vez que estes apresentam como saída células ou outros componentes que representam nós num grafo, os quais necessitam ser pesquisados por forma a encontrar a solução.

Pode-se considerar que existem dois tipos de algoritmos de pesquisa os não informados, como são os casos das pesquisas em largura, profundidade e aprofundamento progressivo e os informados como o método guloso, o *Dijkstra* e o A*. A diferença entre ambos consiste em que os primeiros não têm em conta informações sobre o sistema, enquanto que os outros levam em consideração questões como custos de percurso ou heurísticas de distância entre dois pontos.

Na área de planeamento de trajetórias normalmente são usados os algoritmos informados, já que o tempo de execução e a memória usada são críticos para um planeamento eficiente, e por isso estes não são de busca tão exaustiva quanto os métodos não informados.

Assim, nesta secção abordados três métodos informados, nomeadamente o método guloso (subsecção 2.4.1), o *Dijkstra* (subsecção 2.4.2) e o A^* (subsecção 2.4.3). Estes serão avaliados relativamente a fatores de complexidade espacial[1] e temporal[2], à otimalidade e se é completo ou não. Além disso serão explicados, de forma sumária, os procedimentos para implementar o algoritmo e, também, serão apresentadas algumas aplicações de cada um.

A complexidade temporal e espacial são medidas quantos aos seguintes termos:

r - fator de ramificação, isto é número de máximo de ligações para cada nó

p - profundidade da solução

m - profundidade máxima do grafo

2.4.1 Pesquisa Gulosa

Estes algoritmos tentam expandir o nó que se encontra mais perto da solução, pois assume que este é o caminho que conduz à solução mais rapidamente, [27]. A pesquisa gulosa baseia-se em funções heurísticas, $h(n)$, que estimam quanto falta do nó atual para o nó destino, sendo definida pela função $f(n) = h(n)$.

Assim estes algoritmos são não ótimos, uma vez que vão expandindo os nós com menor distância a cada iteração, progressivamente até encontrar o destino, sendo que nem em todos os casos é a situação ideal, pois pode ter ignorado um percurso em que o primeiro nó fosse mais distante, mas compensa-se nos restantes casos. Além disso este algoritmo é completo (se r finito), no caso de pesquisa em grafos, e tanto a sua complexidade temporal como espacial são de $O(r^m)$, podendo ser melhorado consoante a qualidade da heurística.

Em 2012 *Shih et al.* [28] sugeriram a utilização da pesquisa gulosa para controlar a trajetória de um manipulador robótico, para tentar melhorar a complexidade temporal e espacial, enquanto aumentava a eficácia do planeamento.

Em 2013 *Shih et al.* [29] propuseram uma solução baseada em pesquisa gulosa para veículos autónomos em ambientes dinâmicos, sugerindo modificações por forma tornar o algoritmo mais perto da otimalidade.

2.4.2 Algoritmo *Dijkstra*

Este método é usado para determinar os caminhos mais curtos, recorrendo aos custos associados a ir de um nó para outro. Assim expande o nó inicial obtendo todos ligados a ele, coloca-os na pilha e reordena de acordo com o custo, $g(n)$, associado, desta forma o elemento com menor custo, que se encontra no topo da pilha, é extraído e expandido, continuando este processo até encontrar a solução. [9]

Este algoritmo é completo e ótimo, com custos positivos, [9], tem complexidade espacial de $O(r^p)$ e complexidade temporal $O(r^m)$.

[1] Complexidade espacial é o máximo número de nós guardados em memória

[2] Complexidade temporal é a quantidade de tempo necessário para realizar o algoritmo

Em 2003 *Kim et al.* [30] apresentam uma solução para extrair o caminho ótimo de grafos de *Roadmap* usando este algoritmo com algumas alterações, tendo apenas simulado e obtido resultados que apontam para a possibilidade de ser implementado este método num sistema real.

2.4.3 Algoritmo A^*

Este algoritmo combina o custo desde o nó inicial até ao atual, $g(n)$, com a heurística, $h(n)$, sendo assim baseado na equação $f(n) = g(n) + h(n)$.

Assim expande o nó inicial e insere numa pilha todos os vizinhos, ordenando-os de acordo com $f(n)$, ou seja o menor no topo. Este valor de menor custo é retirado e expandido. O processo repete-se até ser encontrada a solução. [9]

Este algoritmo é ótimo, completo e a sua complexidade espacial e temporal depende em grande parte da qualidade da heurística e dos custos utilizados.

Em 2000 *Yamashita et al.* [31] apresentaram uma solução para planeamento de movimentos para transporte de carga por vários robôs móveis, no qual recorria a este algoritmo para encontrar o caminho ótimo e com recurso a uma heurística que reduzia os tempos de transporte e que afastava os robôs dos obstáculos.

Em 2009 *Costa et al.* [32] apresentaram uma solução baseada em A^* para planear trajetórias em tempo real aplicada ao futebol robótico, que tenta responder a obstáculos em movimento de forma mais eficiente, reduzindo o tempo demorado a realizar a trajetória sem provocar grandes aumentos no tempo de processamento e diminuindo o número de colisões.

Em 2011 *Costa* [8] apresenta um método de planeamento cooperativo de tarefas e trajetórias com recurso a este algoritmo, também aplicado em futebol robótico, que com a realização de algumas modificações, permitiram reduzir os tempos de deslocamento entre pontos e reduzir as colisões, mantendo as trajetórias perto da otimalidade.

Capítulo 3

Formulação do Problema

Neste capítulo é exposta de forma sucinta a problemática abordada por esta dissertação tanto na área industrial como na abordagem adaptada para uma competição de robótica, o *Robot@Factory*. Além disso será apresentada a metodologia a implementar para o desenvolvimento do projeto.

3.1 Introdução

Hoje em dia as indústrias procuram criar fábricas inteligentes cada vez menos dependentes de operadores, nomeadamente na área de transporte de matéria prima entre postos de trabalho, usando AGVs, por forma a otimizar os tempos de transporte e reduzir os acidentes causados por erros humanos.

Estes transportadores atuam num ambiente de trabalho dinâmico em que podem surgir obstáculos, tal como os trabalhadores a cruzar com o robô, material caído ou armazenado no percurso e mesmo outros robôs a circularem em trabalho.

Em ambientes industriais os sistemas de veículos são constituídos por mais de 100 AGVs [33], desta forma o planeamento de trajetória tem de ser o mais rápido possível e otimizado de forma a possibilitar o replaneamento para permitir a coordenação entre eles. Contudo muitos dos sistemas têm percursos pré-definidos que restringem o movimento, nomeadamente o seguimento de linhas como é o caso do AGV na Figura 3.1.

Assim a problemática a enfrentar neste projeto é a implementação de algoritmos que permitam fazer um planeamento dinâmico que seja facilmente adaptável a diferentes ambientes e que não necessite de indicadores como linhas, carris ou marcos para definir o caminho, usando sistemas de localização autónomos, que para efeitos de dissertação é usado o sistema de localização implementado por um dos elementos da equipa do *Robot@Factory*, baseado em laser, fazendo a localização usando as paredes, e odometria.

Para além de ser dinâmico e adaptável, este também deve permitir responder à presença de vários robôs em atuação, procurando sempre o caminho ótimo e o mais seguro para assegurar que não existem colisões.

Figura 3.1: Exemplo de AGV de seguimento de linha

Uma vez que é pretendido testar o método num ambiente industrial foi escolhida a competição *Robot@Factory* como meio de simulação, tanto por existir um simulador disponível para testes, como também por ser uma representação fiel de uma indústria, pois consiste em implementar um sistema de transporte de peças entre armazéns e máquinas, onde se pode, inclusive, incluir um segundo robô em trabalho. Esta competição passa a ser descrita na secção seguinte.

3.2 *Robot@Factory*

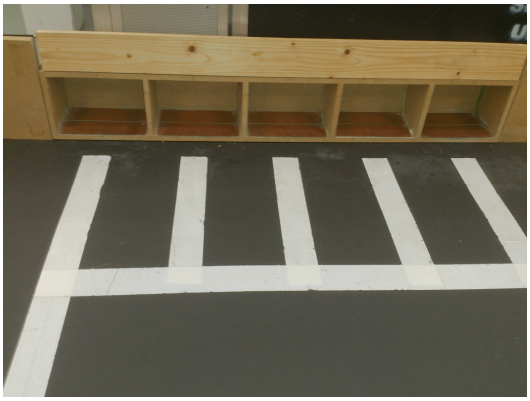
O *Robot@Factory* é uma competição, realizada no Festival Nacional de Robótica, Robótica 2014, que consiste na utilização de um robô autónomo num ambiente fabril, por forma a realizar ações de carga e descarga de materiais para armazenamento ou processamento das peças, devendo sempre evitar colisões com os obstáculos que encontrem, os quais neste caso são estáticos ao ambiente (máquinas e paredes).

Esta competição introduz nível de dificuldade intermédio que visa a transição das ligas juniores para as ligas seniores podendo ser implementadas soluções mais simplistas como o caso de seguimento de linhas brancas, passando para o seguimento de rotas pré-definidas usando sistemas de localização independentes das marcações no chão, podendo ainda se introduzir soluções mais robustas e dinâmicas que permitam ao robô ser totalmente autónomo e calcular a trajetória mais apropriada a usar sem depender de linhas ou caminhos que restringem movimentos.

O campo onde decorrem as provas pode ir até dimensões máximas de 3.5 x 2.5 m [34], sendo que este tamanho também se aplica ao chão no qual se encontram desenhadas as linhas que unem todos pontos de entrada em armazéns e máquinas por forma a poderem ser usadas para localização e seguimento de linhas. Além disso esta fábrica é constituída por oito máquinas onde são trabalhadas as peças e que se encontram agrupadas em duas caixas como se vê na Figura 3.2a. Também existem dois armazéns, um para fornecer a matéria prima e outro para armazenar o produto final,

constituídos por cinco *slots* cada, como se pode ver na Figura 3.2b. Com isto obtém-se uma fábrica onde os componentes são expostos simetricamente conforme se observa na Figura 3.2c.

(a) Conjunto de máquinas de processamento



(b) *Slots* constituintes do armazém

(c) Campo completo

Figura 3.2: Campo do *Robot@Factory* e elementos constituintes

A prova é dividida em três mangas, nas quais o objetivos aumentam de complexidade. Na primeira manga efetua-se o transporte das peças, que se pode ver na Figura 3.3, de um armazém para o outro no menor tempo possível.

Na segunda manga algumas das peças necessitam de ser levadas para a máquina para serem processadas encontrando-se representadas pelo LED a verde e quando mudar de cor para azul podem ser levadas para o armazém de saída, as restantes podem ser levadas diretamente para a saída.

Na terceira e última manga existem três tipos de peças, com a cor azul, podendo ser levada diretamente para a saída, com a cor verde que tem de passar por uma das máquinas e com a cor

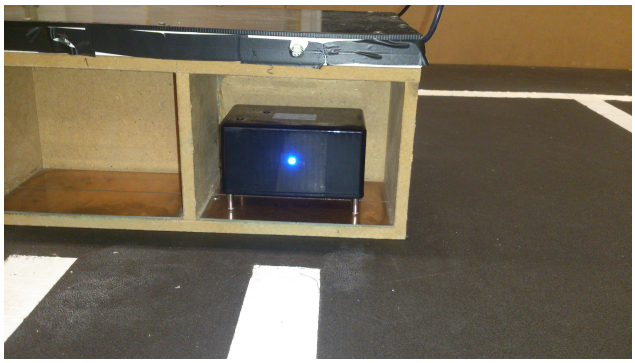


Figura 3.3: Peça usada na competição *Robot@Factory*

vermelha que precisa de sofrer duas maquinações. Nesta ronda existe a possibilidade das equipas recorrerem a dois robôs em funcionamento em campo e, ainda, pode ser colocado um obstáculo.

No final da prova ganha a equipa que transportou o maior número de peças e, em caso de empate, com o menor tempo. De frisar que o tempo sofre penalizações em caso de colisões com os obstáculos do ambiente.

3.3 Metodologia

Por forma a abordar a problemática desta dissertação após realizar o estado da arte, capítulo 2, dos métodos aplicados para planeamento de trajetórias foi efetuada a escolha dos algoritmos que iriam ser implementados neste projeto.

Assim, tendo em conta que é necessário implementar um método que obtenha o caminho ótimo e que ao mesmo tempo seja rápido para realizar o replaneamento, optou-se pelo algoritmo de pesquisa A^*, pois com as heurísticas apropriadas o algoritmo descobre a solução ótima rapidamente. Por forma a criar o grafo representativo do espaço é necessário implementar um método de planeamento que construa os nós. Por isso foi escolhido o algoritmo de decomposição em células aproximadas com células fixas, já que é um algoritmo simples de implementar e, como as células não mudam de tamanho nem implicam métodos de implementação complexos, a velocidade de execução é rápida.

Por fim, para o controlo de movimento foi escolhido implementar um sistema de navegação com uma estrutura hierárquica para definir os eventos que são atuados.

Capítulo 4

Plataformas Implementadas

Para poder validar os algoritmos é necessário utilizar plataformas para a implementação desde um simulador ao robô prototipado. Além disso uma das bases do projeto é a competição *Robot@Factory*, desta forma torna-se necessário escolher um simulador que possua as capacidades de implementar o ambiente e que o protótipo construido cumpra a devida regulamentação.

Assim neste capítulo começa-se por se apresentar o robô projetado, expondo os modelos em CAD criados, os atuadores usados, o resultado e a arquitetura funcional do mesmo. No final apresenta-se o simulador usado e as alterações a que se procedeu para aproximar a realidade.

4.1 Robô Projetado

O robô a utilizar para testes da implementação foi idealizado e prototipado pela equipa *Feup-Factory*, com vista à participação na competição *Robot@Factory*.

O protótipo para a prova não tem restrições ao nível da topologia de locomoção, sendo apenas restringido o tamanho, sem contabilizar as garras, a 45 x 40 cm e 35 cm de altura [34]. Como tal o foi escolhido recorrer a um robô omnidirecional com três rodas, Figura 4.1, desfasadas 120º entre elas, devido a não apresentar restrições de movimentos, permitindo translação e rotação simultâneas, o que se torna benéfico para a prova, e, ao contrário do de quatro rodas, não é necessário um sistema mecânico extra para realizar suspensão.

Figura 4.1: Roda omnidirecional usada da *Bot'n Roll*

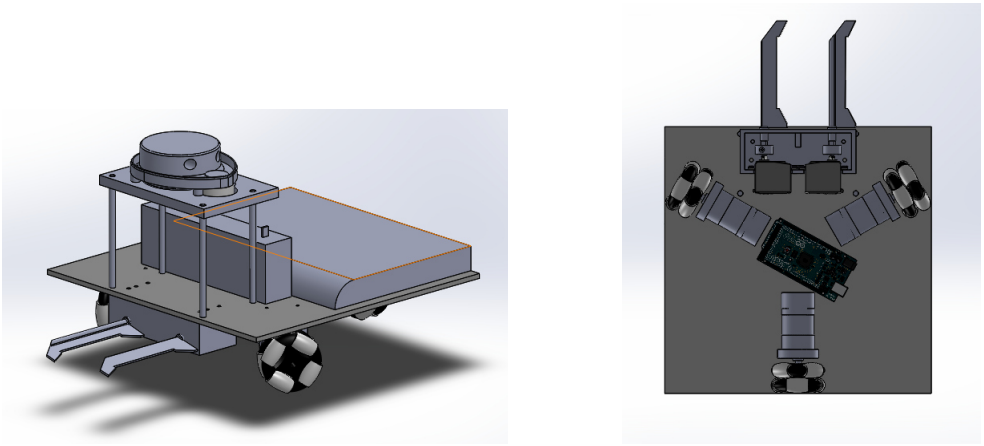
De seguida são apresentados os modelos em CAD desenhados para o robô, após isso referem-se os atuadores utilizados para o movimento do robô e por fim apresenta-se o resultado final da construção.

4.1.1 Projeto do robô em CAD

Antes da construção do protótipo foi necessário projetar o modelo 3D do robô para permitir uma melhor gestão de espaço de maneira a acomodar os componentes necessários em posições pré-definidas.

Com isto foi realizado o mapeamento de todas a furações e peças necessárias para tornar os componentes modulares, facilitando a troca de componentes, e ainda para garantir o menor erro possível nas distâncias das rodas ao centro do eixo e nos ângulos entre elas, visando, assim, reduzir efeitos nefastos derivados do *hardware*.

No conjunto de Figuras 4.2 encontram-se duas vistas do robô modelizado.



(a) Vista superior do CAD do robô (b) Vista inferior do CAD do robô

Figura 4.2: Protótipos em CAD do robô a construir

As dimensões do robô, contabilizando as garras, podem ser consultadas na tabela 4.1, em que a altura é ajustável consoante a necessidade para o laser.

Tabela 4.1: Dimensões projetadas para o robô

Parâmetro	**Dimensão (cm)**
Comprimento	30
Largura	48
Altura (máx.)	20

4.1.2 Atuadores utilizados

Uma vez que este projeto retrata métodos de planeamento de trajetórias e o controlo de movimento, torna-se uma necessidade definir e especificar os atuadores usados para o robô se deslocar e para o controlo do sistema de garras para empilhamento das peças através de servomotores.

Assim de seguida são apresentadas as tecnologias usadas no robô prototipado.

4.1.2.1 Servomotores

Para o controlo do sistema de empilhamento de peças, no caso do protótipo composto por duas garras, recorreu-se a dois servomotores standard, *Futaba S3003*, o qual pode ser visto na Figura 4.3 [35].

Figura 4.3: Servomotor usado para o controlo das garras

Estes servomotores são controlados através da largura dos pulsos da PWM, sendo necessário uma largura de 1.5ms para ir para a posição central (90^o), 2ms para rodar para o ângulo máximo (180^o), no sentido anti-horário, e 0.5ms para o ângulo mínimo (0^o).

Os atuadores das garras foram escolhidos devido ao baixo custo, à velocidade de rotação ser rápida ($0.19s/60^o$ sem carga) e o binário ser o suficiente para rodar as garras, o que compensa o facto de ter um consumo elevado [36], uma vez que só serão atuados esporadicamente.

No robô os dois servos irão ser dispostos de forma colinear, mas invertidos 180^o, tal como se pode observar na Figura 4.4.

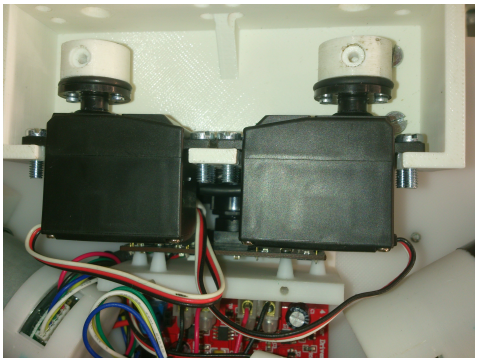


Figura 4.4: Disposição dos servomotores usados para o controlo das garras

4.1.2.2 Motor DC e driver de controlo

Os motores DC usados foram os *30:1 Metal Gearmotor 37Dx52L mm with 64 CPR Encoder* [37], da *Pololu*, e foram escolhidos devido a serem alimentados a 12V, os quais podem ser provenientes diretamente da bateria, atingem uma velocidade alta, 350 rpm, e têm um binário suficientemente elevado para movimentar o robô (8kg.cm), tendo contudo uma corrente de *stall*, corrente com o motor bloqueado, elevada, de 5A, mas a qual idealmente não será atingida. Para além disso já vêm providos de *encoders* o que traz mais uma fonte de informação para a localização do sistema. O motor encontra-se na Figura 4.5.

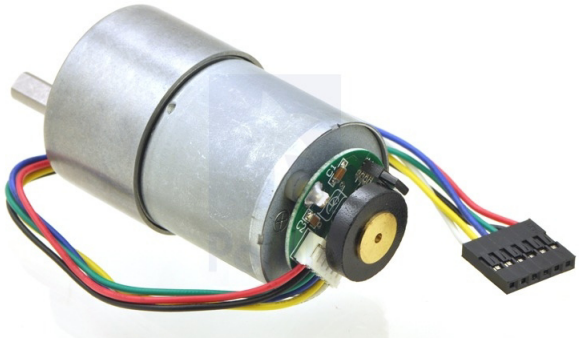


Figura 4.5: Motor DC *30:1 Metal Gearmotor 37Dx52L mm with 64 CPR Encoder* utilizado

Para o controlo dos motores DC foi utilizada a placa *Rover 5 Motor Driver Board* [38], como se pode ver na Figura 4.6, que permite o controlo de quatro motores DC, com alimentação máxima de 12V e corrente de *stall* de 4A, em simultâneo através de quatro pontes "H"de FET's com baixa resistência.

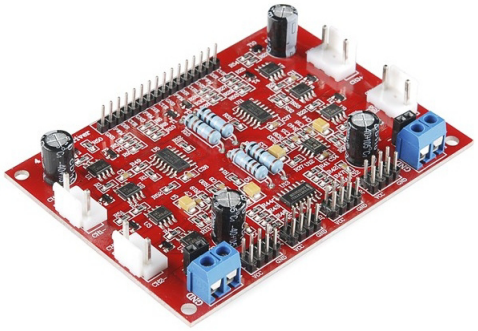
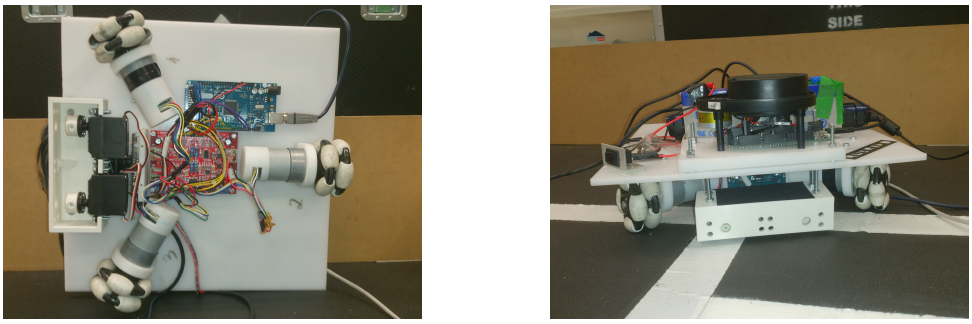


Figura 4.6: Driver *Rover 5 Motor Driver Board* usada para controlo dos motores DC

Esta driver comanda os motores através do fornecimento de um PWM para controlo de velocidade e uma *flag* lógica, 0 ou 1, para indicar a direção para cada canal. Além disto permite efetuar a leitura de ambas as inputs dos *encoders* em quadratura através de um pino de *interrupt* e também realizar a medição da corrente consumida por cada motor.

4.1.3 Robô prototipado

Após o projeto e escolha dos componentes foi construído um protótipo real para meio de implementação dos métodos propostos. Assim o robô final pode ser observado de duas vistas no grupo de Figuras 4.7.



(a) Vista Inferior (b) Vista superior

Figura 4.7: Robô prototipado

Na Figura 4.8 são representadas as ligações entre o *hardware* existente no robô, sendo que no PC são processados os métodos de alto nível, como é o caso do planeamento de trajetórias, e o *Arduino Mega* realiza o controlo dos motores DC e dos servomotores.

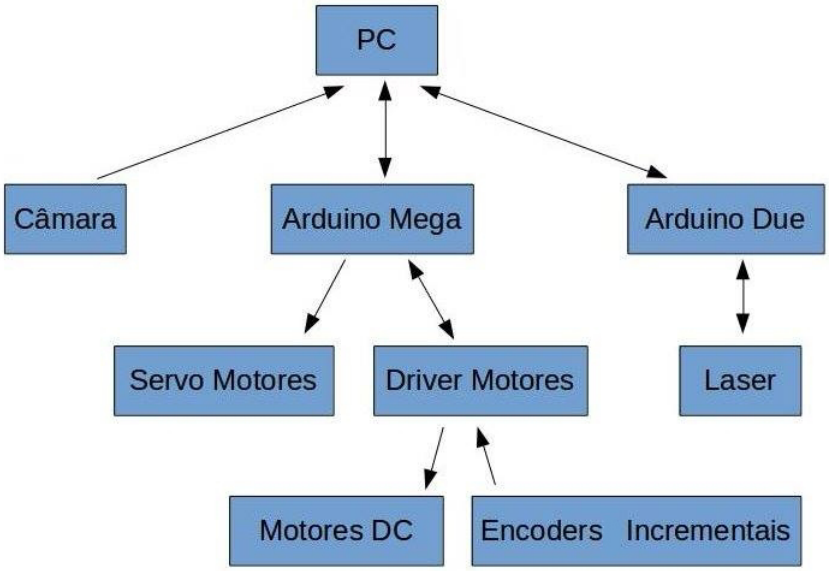


Figura 4.8: Diagrama de blocos representativo das ligações do *hardware* no robô prototipado

4.1.4 Arquitetura funcional

A arquitetura funcional do robô encontra-se distribuída por diversas camadas, sendo representadas na Figura 4.9.

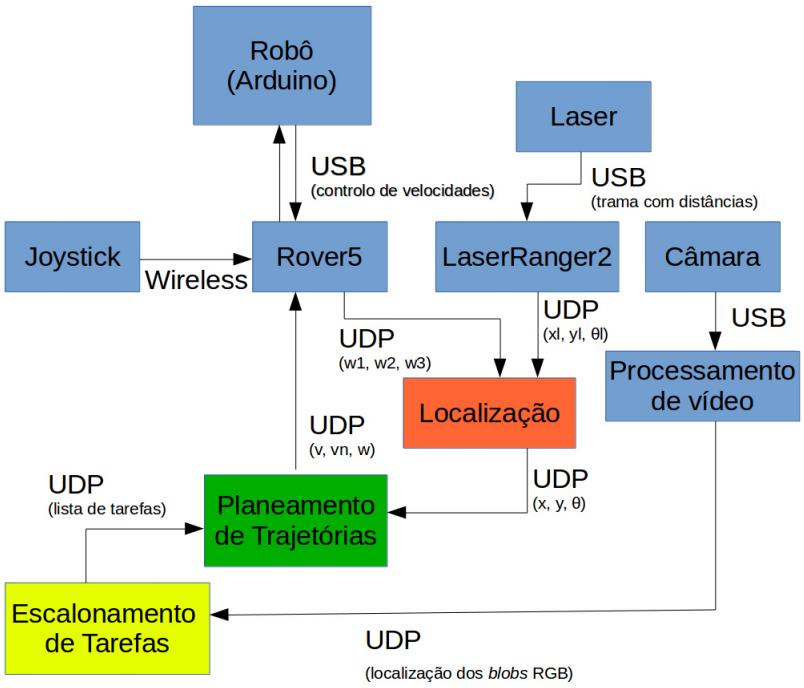


Figura 4.9: Diagrama de blocos representativo da arquitetura funcional do robô prototipado

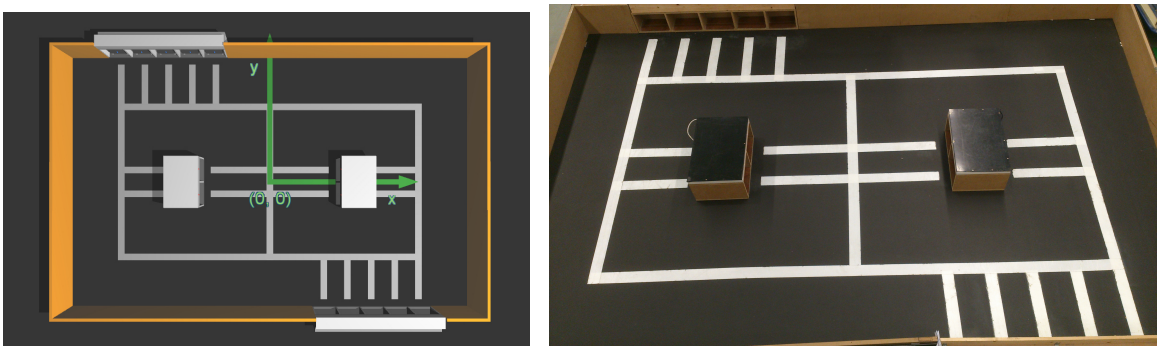
Com esta arquitetura pode-se observar um método distribuído, em que o que se encontra a azul são as camadas que fazem a interface do hardware do robô para o PC que executa os algoritmos de alto nível, como os de localização, planeamento de trajetórias e escalonamento de tarefas.

Neste projeto em questão realiza-se a componente a verde de planeamento de trajetórias a qual obtém a localização (X, Y, θ) provenientes do módulo laranja e também as tarefas que deve executar, isto é que postos de trabalho devem ser visitados. Com isto são calculados os percursos a ser efetuados entre os pontos e faz o controlo de velocidades para os realizar de forma adequada. Estas velocidades (V, V_n, W) são enviadas para o *Rover5* que realiza a interface entre a camada de baixo nível (*arduino*) e o PC enviando os comando para o *arduino* gerar as PWM de controlo da driver dos motores.

4.2 Simulador

O simulador utilizado para os testes do algoritmo foi o *SimTwo* [39], o qual se trata de um sistema que permite recriar vários ambientes onde se podem implementar diferentes tipos de robôs, nomeadamente com configurações omnidirecionais de três rodas, permitindo assim representar o robô prototipado. Além disso dispõe de uma aproximação realista, pois permite tomar considerações físicas como a forma, massa, atritos das superfícies, entre outros, e ainda dispõe de modelos que visam aproximar e capturar os elementos não lineares presentes nos motores que atuam nos robôs.

O *SimTwo* também surge como sugestão de simulador oficial da competição *Robot@Factory* [34], já que tem representado o campo e todos os elementos da competição de forma fiel e numa escala muito próxima da real, sendo apenas necessário modelar o robô para o utilizado, existindo inclusive diversos sensores, como por exemplo de linha branca, lidar, câmara e sensores infravermelhos. Tal como se pode ver nas Figuras 4.10a e 4.10b a representação do campo simulado aproxima-se do real.



(a) Campo representado no *SimTwo* (b) Campo real da competição

Figura 4.10: Comparação entre campo do simulador e o real

4.2.1 Modelo do robô

O robô modelizado para servir de apoio a esta dissertação foi criado com vista a manter as dimensões próximas das reais e que se encontram listadas na tabela 4.1, procurando também realizar uma distribuição do peso o mais semelhante à realidade possível.

Assim o modelo é constituído pela base maior, onde se encontram dispostas com o desfasamento de 120º as rodas omnidirecionais, representadas com as rodas equidistantes ao eixo de rotação do robô, o qual, tal como no robô real, se encontra com um desvio de 1cm em X relativamente ao centro da base. Na parte frontal do robô encontra-se ainda um lidar, modelizado para aproximar o real, e ainda as garras que no modelo apenas fazem a distância existente, não sendo dotadas de movimento para carregar as caixas. Desta forma o robô modelizado pode ser visto na Figura 4.11.

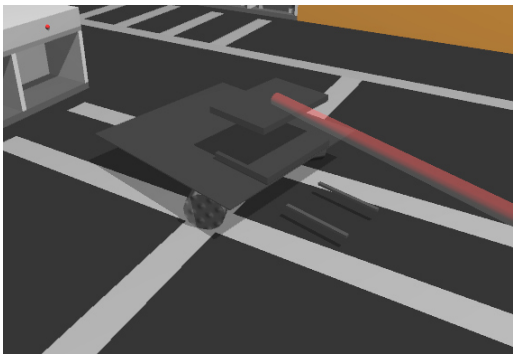


Figura 4.11: Robô modelizado em *SimTwo*

Ao nível da arquitetura funcional o *Simtwo* simula a componentes de localização e de controlo de baixo nível, isto é o *arduino*. Sendo que a localização é obtida através de uma rotina já existente que devolve a localização exata do robô, representada através da seta laranja na Figura 4.12, ou através do sistema de localização ligado com o *Simtwo*, desenvolvido por um membro da equipa *FeupFactory*.

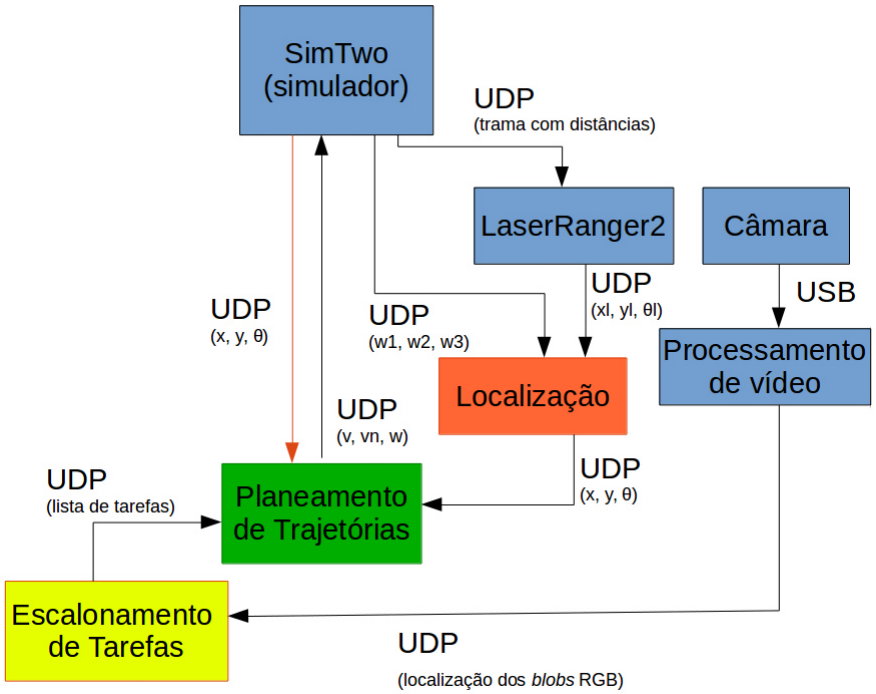


Figura 4.12: Diagrama de blocos representativo da arquitetura funcional alternativa para o *SimTwo*

Quanto ao controlo das velocidades, por forma a receber o mesmo que o *Rover5*, foi implementada uma rotina que através de (V, V_n, W) calcula a velocidade a aplicar a cada roda (V_1, V_2, V_3) através da equação 4.1 [40], em que d é distância das rodas ao eixo de rotação.

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} -sin(\pi/3) & cos(\pi/3) & d \\ 0 & -1 & d \\ sin(\pi/3) & cos(\pi/3) & d \end{bmatrix} \cdot \begin{bmatrix} V \\ V_n \\ W \end{bmatrix} \tag{4.1}$$

Por forma a comunicar entre os métodos desenvolvidos e o simulador é utilizado o protocolo de comunicação *UDP* que se encontra disponível no *SimTwo*.

Capítulo 5

Planeamento de Trajetórias

Este capítulo apresenta os algoritmos centrais ao cumprimento dos objetivos propostos nesta dissertação. Tal como já foi previamente referido o planeamento de trajetórias a idealizar tem de envolver métodos que para além de permitirem ser calculados em tempo real, funcionando de forma reativa a agentes externos, também tem de ser capaz de gerar percursos ótimos ou perto disso.

Assim neste capítulo procede-se à apresentação das metodologias adotadas, referidas no capítulo 3, começando pela apresentação do algoritmo de pesquisa e cálculo das trajetórias ótimas, o algoritmo A^*, fazendo uma descrição do seu algoritmo teórico e propriedades e de seguida apresenta-se as alterações efetuadas para o tornar mais eficiente e apto a ser implementado para aceitar replaneamento. Explicam-se ainda os métodos usados para representar o ambiente por forma a obter os nós necessários para o grafo, falando desde a construção do mapa, à expansão dos obstáculos e ainda as alterações necessárias para funcionar com múltiplos robôs na área de trabalho.

5.1 Algoritmo A^*

Tal como descrito no capítulo 3 os problemas a abordar nesta dissertação envolvem a necessidade de replanear as trajetórias consoante existam alterações no ambiente, o algoritmo A^* demonstra-se como sendo o ideal, pois para além de procurar a solução ótima para o grafo, também o faz de uma forma informada que o torna mais rápido que os restantes.

Este algoritmo é designado como o melhor primeiro, [27], e consiste na pesquisa num grafo do caminho mais curto que une os nós inicial e final. Uma vez que este se trata de um método informado utiliza uma função heurística, $f(n)$, para estimar o caminho com o menor custo estimado desde o nó n até ao final e é obtido através da soma de duas funções $g(n)$ e $h(n)$, isto é $f(n) = g(n) + h(n)$. A função $g(n)$ representa o custo desde o nó inicial até ao nó atual (n), podendo ou não ser heurística, e a $h(n)$ é a estimativa, através de uma heurística, do custo do percurso desde n até ao nó final, devendo esta ser subestimada. Na Figura 5.1 encontram-se representadas estas funções.

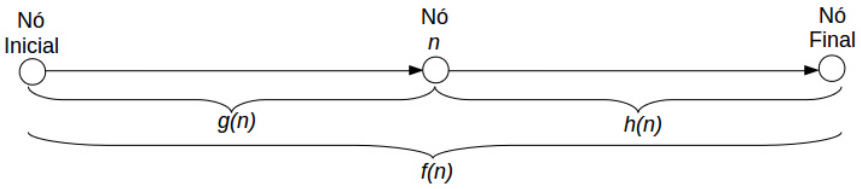


Figura 5.1: Funções $f(n)$, $g(n)$ e $h(n)$

Para o funcionamento deste método usam-se listas de nós, a lista aberta na qual se encontram os nós identificados e não processados e a lista fechada que armazena os nós processados.

Assim este algoritmo é definido através dos seguintes passos descrito em [8, 41, 42]:

1. Adicionar o nó inicial (n_{ini}), após calculo do $f(n_{ini})$, à lista aberta
2. Enquanto a lista aberta (O) não estiver vazia:
   (a) Escolher o nó com menor $f(n)$, n_{melhor}, presente em O, retirar de lá e colocar na lista fechada (C)
   (b) Caso n_{melhor} seja o final terminar o algoritmo
   (c) Senão visitam-se os nós vizinhos (nós ligados ao n_{melhor}) analisa-se o seu estado relativamente a um dos casos:
      i. Se o nó vizinho não se encontra nem em O nem em C adicionar a O
      ii. Se o nó vizinho pertencer à lista aberta, verificar se a função de custo, $f(n)$, é menor que a anteriormente calculada, caso seja altera-se o nó pai para o n_{melhor} e atualiza-se $f(n)$.
      iii. Se o nó vizinho pertencer à lista fechada, verificar se a função de custo, $f(n)$, é menor que a anteriormente calculada, caso seja altera-se o nó pai para o n_{melhor}, atualiza-se $f(n)$ e passa-se esse nó para O.

De notar que nos passos 2(c)ii e 2(c)iii para verificar se $f(n)$ é melhor, é apenas necessário ver $g(n)$, uma vez que apenas este é que se altera e se o seu valor for superior ao custo de ir do início até ao n_{melhor} e de este até ao vizinho a analisar, isto é $g(n_{melhor}) + c(n_{melhor}, n) < g(n)$, então a função de custo é melhor.

Para retirar a trajetória é necessário partir do nó final e ir analisando os nós pais até se chegar ao nó inicial.

Este método tem como propriedades [8] o facto de ser:

- Completo, isto é retorna sempre um resultado, ou chegou ao destino ou não existe nenhum caminho viável
- Admissível, retorna sempre o caminho ótimo, isto se a heurística também for admissível e não sobrestimar a distância

- Consistente, o valor que se estima para chegar do nó até ao destino é sempre menor que o custo de ir até um nó adjacente mais a heurística do mesmo

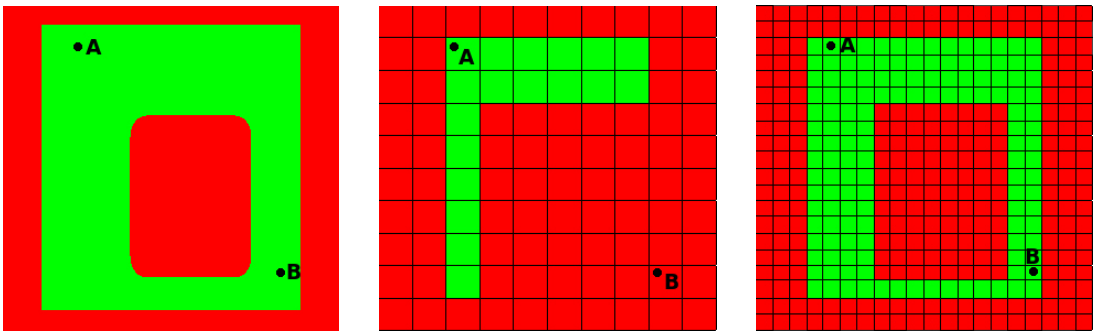
O método como descrito encontra algumas limitações relativamente à eficiência, que se devem ao facto de se ter de visitar várias vezes o mesmo nó e não se considerar algumas limitações.

5.1.1 Alterações ao algoritmo

Para adaptar o algoritmo A^* ao problema abordado neste projeto torna-se necessário definir todas as estruturas que possibilitam o funcionamento adequado do mesmo e a modificações no próprio algoritmo por forma a que este se torne mais eficiente e rápido para funcionamento em tempo real.

5.1.1.1 Representação dos nós

Para representação dos nós do grafo é utilizado o algoritmo de decomposição em células aproximadas com células fixas. Este método consiste em construir uma grelha composta de células quadradas, em que cada uma representa um nó e contém informação sobre o estado do espaço em que se inserem, isto é se está ocupado ou livre. Este algoritmo apresenta limitações quanto à otimilidade da trajetória, pois quanto maior a célula maior a possibilidade de perder um percurso que previamente era viável e também pode fazer com que os pontos ótimos não estejam posicionados no local correto, uma vez que a posição de cada nó é o ponto central da célula que pode estar deslocado em relação ao ideal. Contudo este problema é de fácil resolução através da escolha adequada do tamanho para acomodar todos os percursos viáveis e para reduzir o desvio entre as posições. Um exemplo disto encontra-se no conjunto de figuras 5.2, em que a Figura 5.2a mostra o mapa, a Figura 5.2b tem uma grelha grande em que bloqueia dois percursos e não dá para ir do ponto A para o B, já na Figura 5.2c com células quatro vezes menores já é possível passar.



(a) Mapa (b) Mapa representado com grelha grande (c) Mapa representado com grelha pequena

Figura 5.2: Exemplo de decomposição em células aproximadas com células fixas

Assim após construção do mapa, a ser descrita no capítulo seguinte, e da grelha, por forma a permitir ter os dados para a pesquisa, são associadas diversas informações à estrutura que representa o nó, nomeadamente:

1. As posições em XX e em YY dos pontos centrais das células
2. O custo de ir do melhor nó para o nó vizinho, $c(n_{melhor}, n)$.
3. O $g(n)$
4. O $h(n)$
5. O $f(n)$
6. O estado do nó se é possível passar ou não ($Free$)
7. Se pertence à lista aberta ou não
8. Se pertence à lista fechada ou não
9. Apontador para o nó pai

5.1.1.2 Construção dos nós vizinhos

Para a construção dos nós vizinhos foi realizada uma pesquisa com conetividade 8 em torno do n_{melhor}, ou seja vêm-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 5.3.

NE	N	NW
E	n_{melhor}	W
SE	S	SW

Figura 5.3: Vizinhos no n_{melhor}

Ao visitar cada nó atualizam-se os valores de X, Y, o custo de ir do n_{melhor} para lá, o apontador do nó pai para ser o n_{melhor} e se está livre ou não. Para os que se encontram em N, S, E e W o $c(n_{melhor}, n)$ será obtido através da equação 5.1 e para NW, SW, NE e SE o custo será dado por 5.2, em que $Sizec$ é o tamanho definido para as células e K_e é o custo extra que depende do valor que se encontra no mapa para definir as camadas de proteção ou se está totalmente livre e que se encontra descrito mais à frente.

$$c(n_{melhor}, n) = Sizec + K_e * Sizec \tag{5.1}$$

$$c(n_{melhor}, n) = Sizec * \sqrt{2} + K_e * Sizec * \sqrt{2} \tag{5.2}$$

Caso o nó a analisar tiver o valor de obstáculo no mapa então coloca-se $Free = False$, pois não é possível passar e o A^* não precisa de o considerar, caso contrário coloca-se $Free = True$

5.1.1.3 Heurística

Existem várias heurísticas que são viáveis para aplicar num mapa de células, estas para garantir a otimilidade do algoritmo A^* têm de ser subestimadas, como já foi referido anteriormente, contudo devem-se aproximar o mais possível da realidade, por forma a melhorar os resultados e ser necessário expandir menos nós.

A heurística escolhida para o problema foi a da distância euclidiana, uma vez que permite mover-se em qualquer sentido, ou seja permite calcular a distância em reta e em diagonal. Esta heurística, $h(n)$, é calculada através da equação 5.3, em que K é um coeficiente, utilizado por *Costa* em 2011 [8], que tem como objetivo dar maior peso aos nós que estão mais próximos do objetivo, D pode ser definido como o custo mínimo de ir de um ponto até ao adjacente, mas que pode ser definido como 1 [41], que é o caso neste projeto, n_x e n_y são, respetivamente, as coordenadas em XX e YY do nó atual e d_x e d_y são, respetivamente, as coordenadas em XX e YY do nó final.

$$h(n) = K * D * \sqrt{(n_x - d_x)^2 + (n_y - d_y)^2} \tag{5.3}$$

A Figura 5.4 representa um exemplo de um movimento na diagonal [8].

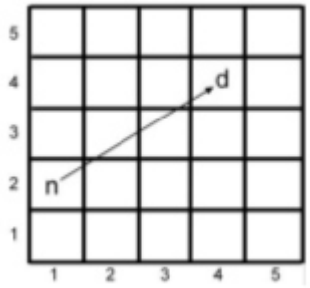


Figura 5.4: Distância euclidiana

5.1.1.4 *Binary heap*

O algoritmo A^* durante a sua execução despende uma parte considerável de tempo a adicionar valores por ordem e a remover o melhor nó das listas aberta e fechada, por isso é necessário escolher uma estrutura de dados com uma complexidade temporal baixa, isto é que seja rápida de executar principalmente para este tipo de atuações.

Tal como referido em [8, 41] as *binary heaps* são estruturas com uma complexidade temporal para os métodos de inserção e remoção baixa, O(log n), o que permite trabalhar facilmente com uma quantidade elevada de nós de forma mais eficaz. Sendo rentável para recorrer em casos em que o número de nós não ultrapassem os 10000 elementos [41].

O método de inserção de valores funciona através da introdução do valor na última posição da lista e realizar a função de *BubbleUp*, em que vê se a função de custo do nó no nível acima da árvore, posição $(Index-1)/2$, é superior, caso seja troca-se um nó com o outro, realizando-se este procedimento até atingir o índice 0 ou o custo atual ser superior ao do nível acima. Desta forma garante-se que o nó na posição 0 é o que tem menor valor de $f(n)$.

Os algoritmos de adicionar elementos e de *BubbleUp* [43, 44] são descritos de seguida:

Algoritmo Adicionar(Lista)

Variáveis:

Heap - Lista de nós a analisar

I_f - Índice da última posição da lista

Pseudocódigo:

1: Adicionar uma posição a Heap
2: Colocar nó em I_f
3: BubbleUp(Heap,I_f)

Algoritmo BubbleUp(Lista,Índice)

Variáveis:

Index - Índice do valor a analisar

Heap - Lista de nós a analisar

f - Função de custo do nó na lista

Pseudocódigo:

1: Enquanto Index > 0 e Heap[(Index-1)/2].f > Heap[Index].f
1.2: Trocar nó em Heap[(Index-1)/2] por Heap[Index]
1.3: Index := (Index-1)/2

O método de remoção de nós passa por inicialmente retornar o valor que se encontra na posição inicial da lista, ou seja o que tem menor valor de f(n), e de seguida colocar nessa posição o último nó da lista, com o maior f(n), depois reduzir o tamanho do array, retirando a última posição, e, se a lista tiver valores, ir descendo o nó na lista analisando os nós filhos, $Index*2+1$ e $Index*2+2$, trocando de posição com o menor dos dois, caso não haja mais nenhum menor termina o algoritmo.

Os algoritmos de remover elementos e de *BubbleDown* [44] são descritos de seguida:

Algoritmo Remover(Lista)

Variáveis:

Heap - Lista de nós a analisar

I_f - Índice da última posição da lista

Pseudocódigo:

1: Retornar Heap[0]
2: Heap[0]:=Heap[I_f]
3: Remover a última posição da lista
4: Se Size(Heap)>0 então BubbleDown(Heap)

Algoritmo BubbleDown(Lista)

Variáveis:
Index - Índice do valor a analisar
Heap - Lista de nós a analisar
SmallerChild - Nó filho mais pequeno
f - Função de custo do nó na lista

Pseudocódigo:

```
1:      Index:=0
2:      Enquanto Index*2+1 < Size(Heap)
2.1:        SmallerChild := Index*2+1
2.2:        Se Index*2+2 < Size(Heap) e Heap[Index*2+1].f > Heap[Index*2+2].f então
            SmallerChild:= Index*2+2
2.3:        Se Heap[Index].f > Heap[SmallerChild].f então
            trocar nó em Heap[Index] por Heap[SmallerChild]
2.4:        Senão sai do ciclo
2.5:        Index:=SmallerChild
```

5.1.1.5 Algoritmo A^* modificado

Por fim, de forma a adequar o algoritmo de pesquisa à problemática associada a esta dissertação procedeu-se a alterações que atuam sobre a eficiência e o tempo de processamento, uma vez que o algoritmo necessita de ser rápido para poder replanear a trajetória em tempo real, mantendo a sua admissibilidade.

Uma das alterações a realizar é retirar a condição de revisitar nós da lista fechada para verificar se sofrem alterações. Esta modificação foi implementada por *Costa* em 2011 [8], pois reduz o número de visitas aos nós sem afetar a admissibilidade, pois, tal como refere, utilizando uma heurística consistente a estimativa será sempre menor que a estimativa do nó adjacente mais o custo de se movimentar de um nó para o seguinte.

Outra das alterações implementadas foi a de ignorar os pontos em que não existe possibilidade de passar, reduzindo, assim, o número de nós a visitar, devido a não se expandir os mesmos, outra modificação é a utilização de *binary heaps* para a representação da lista aberta que permite, tal como foi referido anteriormente, reduzir o tempo de execução nas funções que este método usufrui maioritariamente.

Assim o algoritmo implementado pode ser descrito da seguinte forma:

Algoritmo A^* Modificado

Variáveis:

O - Lista aberta de nós representada por *binary heap*

C - Lista fechada de nós

n_{melhor} - Melhor nó da lista aberta

Neigh(n_{melhor}) - Vizinhos do melhor nó

$c(n_{melhor}, n)$ - custo de passar do nó n_{melhor} para n

Pseudocódigo:

1: Adicionar o nó inicial a O
2: Enquanto Size(O) > 0
2.1: n_{melhor} := Remover(O)
2.2: Adicionar n_{melhor} a C
2.3: Se n_{melhor} for o nó final então retornar a lista de nós que une os nós inicial e final
2.4: Para todos os n $\in$ Neigh(n_{melhor})
2.4.1: Se n ocupado então saltar para próximo vizinho
2.4.2: Se n $\notin$ O e n $\notin$ C então calcular g(n), h(n) e f(n) e Adicionar(O)
2.4.3: Se n $\in$ O e $g(n_{melhor}) + c(n_{melhor}, n) < g(n)$ então calcular novo g(n) e f(n), alterar nó pai para n_{melhor} e fazer BubbleUp(O,Index(n))

5.2 Representação do Ambiente

Na secção anterior foi apresentado o algoritmo do cálculo da trajetória, o qual realiza uma pesquisa em grafos. Para tal é necessário fazer o mapeamento do ambiente circundante, por forma a definir os nós, recorrendo para isso ao algoritmo de decomposição em células aproximadas com células fixas.

Desta forma é necessário inicialmente construir o mapa do campo, de seguida expandir os obstáculos em 2D (X, Y) e, por fim, alterar o mapa consoante a presença de múltiplos robôs.

5.2.1 Construção do mapa

O ambiente, tal como referido no capítulo 3, é conhecido e estático, pertencendo à competição *Robot@Factory*, o que faz com que seja de fácil representação, sendo as dimensões ilustradas na figura 5.5.

Para se realizar o mapeamento do espaço recorre-se à utilização de ferramentas de geração de *Bitmap*, atribuindo diferentes códigos de cor consoante a ocupação das áreas, encontrando-se listadas na Tabela 5.1.

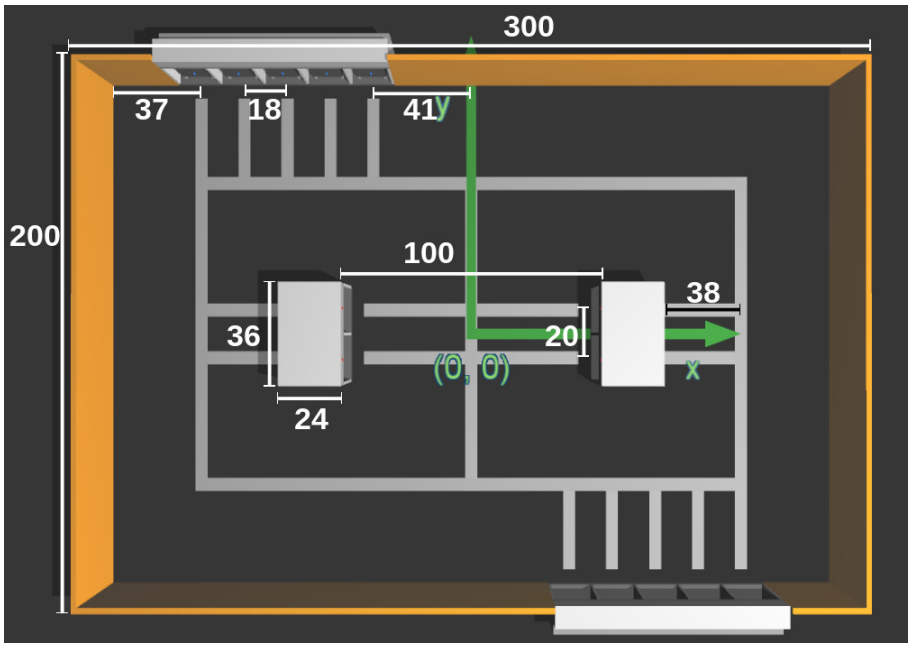


Figura 5.5: Medidas do campo em cm

Tabela 5.1: Codificação RGB do estado das zonas

Estado	**R**	**G**	**B**
Livre	0	255	0
Ocupado	255	0	0
Camada de Proteção Alta	255	165	0
Camada de Proteção Média	255	215	0
Camada de Proteção Baixa	255	255	0

Assim esta representação permite retirar informação para definir os nós a serem usados no algoritmo A^*, visando, com isto, a implementação do algoritmo de decomposição em células aproximadas de dimensões fixas, descrito anteriormente, definindo as células através de quadrados com dimensão escolhida pelo utilizador. Com isto a resolução obtida na representação será igual à dimensão das células.

Já que o campo é constituído por formas de fácil representação, retângulos, o mapa é construido definindo uma imagem com fundo indicativo de livre com o comprimento e largura do campo, reduzidos consoante a resolução, de seguida desenham-se as paredes como linhas que contornam o exterior e, por fim, inserem-se as máquinas como retângulos definidos por dois vértices diagonalmente opostos, ambas com cor de ocupado.

Desta forma o mapa fica representado com os obstáculos e paredes, tal como se pode ver na Figura 5.6, sendo posteriormente necessário expandir os obstáculos para permitir que o robô seja representado através de um ponto, facilitando o planeamento da trajetória.

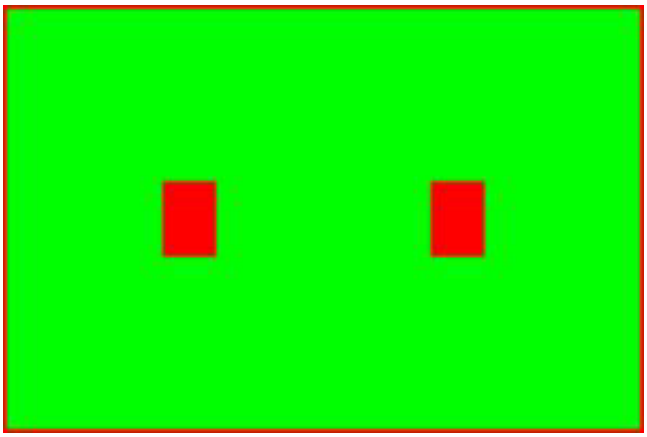


Figura 5.6: Bitmap do campo do *Robot@Factory*

5.2.2 Expansão dos obstáculos

Tal como referido em [8], a expansão da área dos obstáculos permite que o o robô seja considerado como um ponto no espaço, fazendo com que o planeamento da trajetória passe por encontrar o caminho do ponto ao longo do espaço livre, uma vez que ao aumentar os obstáculos garante-se que o robô não colide. Por isso o algoritmo desenvolvido visa garantir a segurança da trajetória, tendo um tempo de processamento rápido, contudo limitando a manobrabilidade do robô.

O robô, para ser usado para a expansão, foi visto como um círculo de raio (R) igual à maior distância do centro à extremidade, como se pode ver na Figura 5.7. O método desenvolvido, exemplificado nas Figuras 5.8a e 5.8b, consiste em considerar todas as poses (X e Y) que se encontrem a uma distância inferior a R como locais com o valor ocupado, desta forma ao efetuar a expansão garante-se que nunca haverá contacto independentemente da orientação que o agente tome, pois considera-se o pior caso. Na Figura 5.8b pode-se ver que, tendo o centro (O) como a localização, o robô (a) encontra-se numa situação de colisão, enquanto que o robô (b), embora muito perto, se encontra numa situação válida.

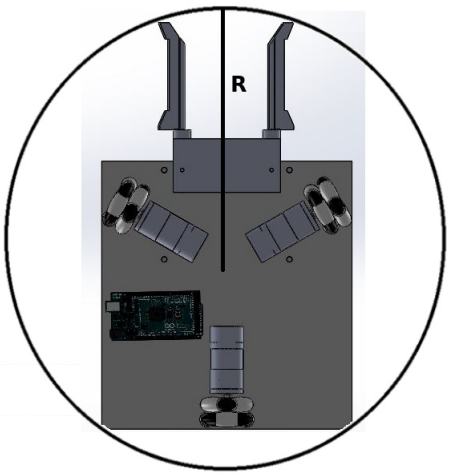
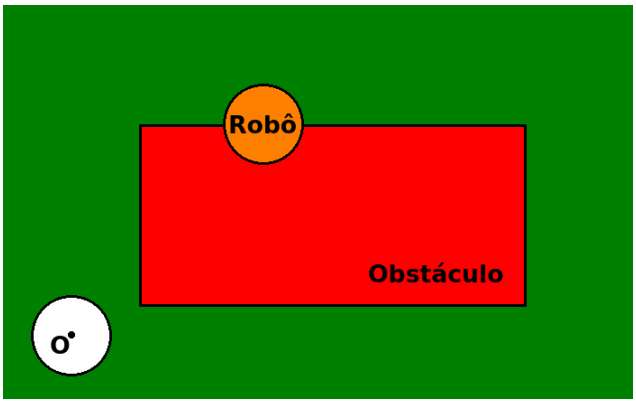
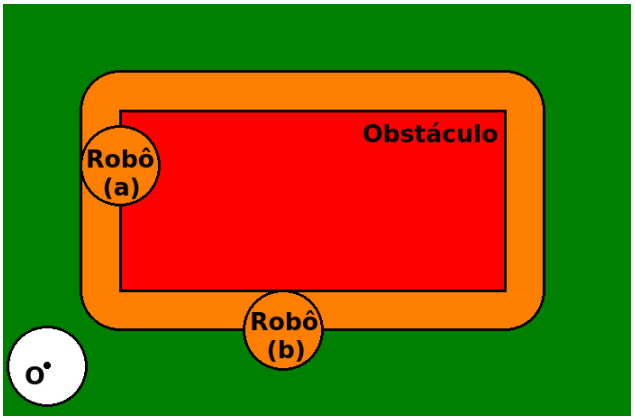


Figura 5.7: Representação do robô através de um círculo de raio R



(a) Antes da expansão



(b) Após expansão

Figura 5.8: Algoritmo de expansão de obstáculos

Este método consiste em três fases, primeiro converter o *Bitmap* em matriz, de seguida calcular a matriz de distância, por forma a obter a distância do robô aos obstáculos, e finalmente atribuir um estado às posições no mapa consoante as distâncias.

Inicialmente percorre-se o mapa construido em *Bitmap* e através do código de cores definido na Tabela 5.1 é criada uma matriz na qual os obstáculos são representados como 1 e o espaço livre como 0.

Com a matriz definida, efetua-se o cálculo da matriz de distâncias, o qual assentou no algoritmo exposto em 1984 por *Borgefors* [45], que consiste em realizar dois varrimentos, representados na Figura 5.9, um da esquerda para a direita e de cima para baixo e outro da direita para a esquerda e de baixo para cima nos quais de aplica uma transformada de distâncias, relativamente a pontos com o valor de obstáculo (1), sendo que no primeiro varrimento tudo o que não for obstáculo terá um peso infinito.

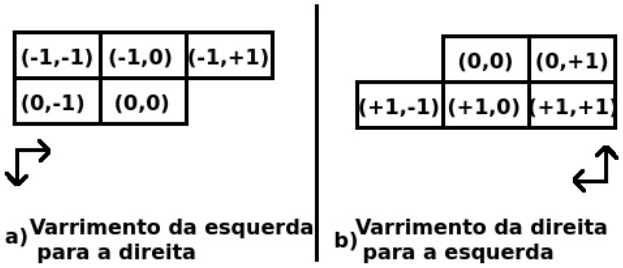


Figura 5.9: Transformadas dos varrimentos

O algoritmo baseado em [45] pode ser descrito da seguinte forma:

Algoritmo Transformação de distâncias com conetividade 8

Variáveis:

L - Número total de linhas

C - Número total de colunas

v(i,j) - Valor do pixel na linha i e coluna j

d1 - Distância horizontal e vertical

d2 - Distância diagonal

Pseudocódigo:

```
1:      Varrimento da esquerda para a direita
1.1:        for i := 1 até L-1
1.2:        for j := 1 até C-1
1.2.1:          if v(i,j) = 0 então v(i,j) := ∞;
1.2.2:          if v(i-1,j-1) = 0 então v(i-1,j-1) := ∞;
1.2.3:          if v(i-1,j) = 0 então v(i-1,j) := ∞;
1.2.4:          if v(i-1,j+1) = 0 então v(i-1,j+1) := ∞;
1.2.5:          if v(i,j-1) = 0 então v(i,j-1) := ∞;
1.2.6:          v(i,j) := min(v(i-1,j-1)+d2, v(i-1,j)+d1, v(i-1,j+1)+d2, v(i,j-1)+d1, v(i,j));
1.3:        end;
1.4:        end;

2:      Varrimento da direita para a esquerda
2.1:        for i := L-2 até 0
2.2:        for j := C-2 até 0
2.2.1:          v(i,j) := min(v(i+1,j+1)+d2, v(i+1,j)+d1, v(i+1,j-1)+d2, v(i,j+1)+d1, v(i,j));
2.3:        end;
2.4:        end;
```

Assim um exemplo dos varrimentos pode se ver na Figura 5.10, em que $d1 = 1$ e $d2 = 2$, (a) é o mapa em que 1 é obstáculo e 0 é caminho livre, (b) é o resultado após o varrimento da esquerda

para a direita, onde se pode ver que os valores dão a distância aos obstáculos que se encontram à esquerda ou acima, uma vez que não consideram as paredes inferior e a da direita, por fim em (c) as distâncias já foram consideradas em todas as direções após o varrimento da direita para a esquerda. De notar que os valores vêm acrescidos de um erro de 1 devido ao valor dos obstáculos.

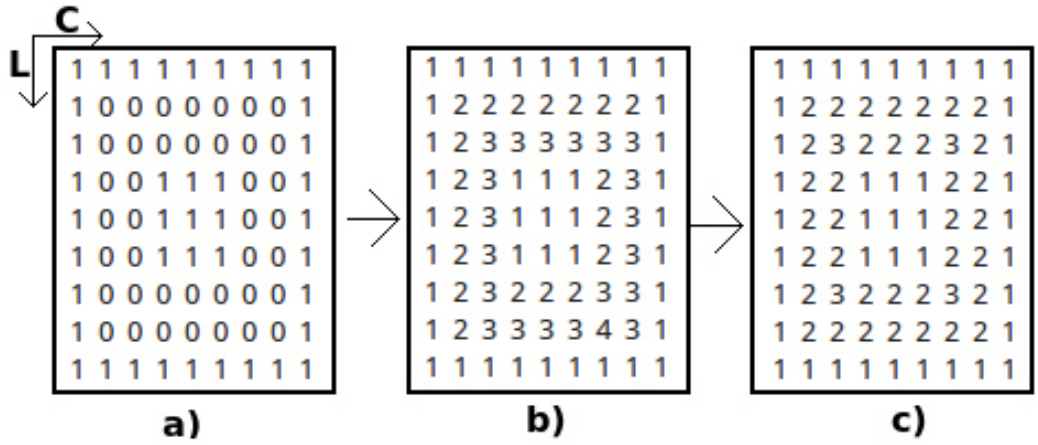


Figura 5.10: Exemplo do algoritmo de criação da matriz de distâncias

Por fim após obter a matriz de distância, o algoritmo percorre a matriz e, consoante o valor encontrado, define o estado nessa posição, ou seja se a distância for inferior ao raio do robô é tida como ocupada, caso esteja no intervalo de $]Raio + n * Tamanho\ célula, Raio + (n+1) * Tamanho\ célula]$, com $n = 0, 1, 2$, insere uma camada protetora que provoca um custo acrescido no planeamento do A^*, para garantir que a trajetória é segura e que só entra nessa área caso seja proveitoso. Estes valores são alterados na matriz que representa o mapa para depois poderem ser usados como custos adicionais (K_e) para a pesquisa, nomeadamente 255 para impossibilitar a passagem devido a obstáculo, 3, 2 ou 1 para custo extra devido à proximidade a um local de colisão e 0 para o caminho completamente livre.

Assim após fazer a expansão do mapa obtém-se uma representação na qual apenas se precisa de considerar o movimento do ponto, esquecendo outras restrições físicas. Com isto a expansão do mapa representado na Figura 5.6 tem como resultado o campo da Figura 5.11, em que as células têm dimensão 2.5cm e o raio do robô é de 25cm.

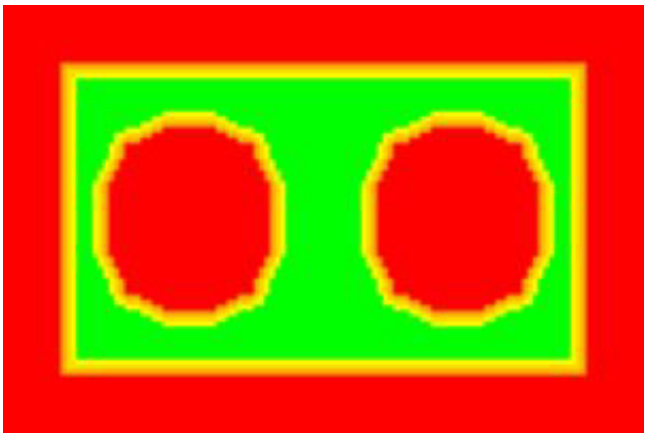


Figura 5.11: Mapa obtido após expansão

5.2.3 Alteração do mapa para múltiplos robôs

O ambiente de implementação desta dissertação apresenta um meio estático, em que os obstáculos se encontram imóveis, por isso o mapa apenas necessita ser criado uma vez, podendo depois voltar a ser chamado o mesmo para replanear a trajetória sem que existam problemas de colisão.

Contudo num ambiente industrial, e também na competição *Robot@Factory*, podem ser usados múltiplos robôs para atuar entre os postos de trabalho, os quais inserem um obstáculo dinâmico que causa mudanças no meio circundante, o que leva à possibilidade de reduzir a manobrabilidade em determinadas áreas e modificar as trajetórias para atingir a meta.

Assim recorrendo aos métodos anteriormente especificados neste capítulo procedeu-se a alterações para acomodar estas novas variáveis. As alterações passam primeiramente por iniciar uma comunicação onde se transmitem as posições dos outros robôs, para saber onde o inserir no mapa, de seguida calcula-se a distância entre eles, caso esta seja inferior a um *threshold* predefinido considera-se obstáculo, caso contrário não é tido em conta, já que se encontra demasiado afastado. O passo seguinte é de inserir um círculo com o raio do segundo robô no *Bitmap* semelhante ao obtido após a construção do mapa e, por fim proceder à expansão, usando o método descrito anteriormente.

A expansão do robô extra é o mesmo que aumentar o círculo de raio R2 para um com raio igual à soma dos raios de ambos os robôs, tal como representado na Figura 5.12.

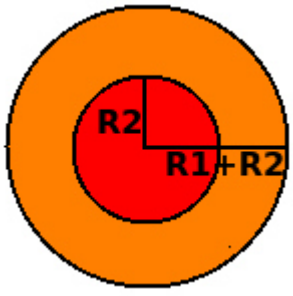


Figura 5.12: Expansão do segundo robô

Com isto o mapa, após a adição do agente extra, para o robô que se encontra a tomar a decisão vai ser variável consoante as posições de ambos, em que caso estejam perto se assemelha com o presente na Figura 5.13.

Figura 5.13: Mapa expandido com a presença do segundo robô na posição $(X,Y) = (0,0)$

Capítulo 6

Aperfeiçoamento do Planeamento de Trajetórias

Com o algoritmo a executar os cálculos para duas dimensões as trajetórias obtidas, principalmente no caso de existirem múltiplos robôs, nem sempre eram as ótimas devido ao facto de serem obtidas tendo em conta uma expansão do mapa pessimista em que se considera o robô como um círculo, bloqueando cruzamentos que de outra forma poderiam ser realizados.

Assim neste capítulo são apresentadas alterações aos métodos previamente implementados por forma a acomodar um novo modo de representação do mapa que possibilitasse a inclusão da orientação dos robôs para a respetiva expansão e para cálculo da trajetória, de maneira a permitir uma maior manobrabilidade, nomeadamente o cruzamento entre os veículos.

Desta forma inicialmente apresentam-se as alterações ao mapeamento, desde a expansão do mapa, à representação dos robôs secundários, e de seguida as alterações necessárias para implementação no algoritmo A^*.

6.1 Representação do Mapa Consoante a Orientação

O mapa como se encontrava expandido anteriormente, usando apenas duas dimensões (X,Y), apesar de garantir a segurança da trajetória, pois expandia os obstáculos considerando sempre o pior caso, e ser de implementação mais simples, tinha um entrave no planeamento de trajetórias, uma vez que no caso de um caminho estreito esse ficava completamente bloqueado se se encontrasse lá o outro veículo, contudo em múltiplas ocasiões com a combinação de orientações adequada seria possível cruzarem-se sem existir colisão.

Para tal procedeu-se à utilização de uma representação tridimensional do mapa considerando agora X, Y e θ [46], fazendo uma pilha de mapas em que cada nível expande os obstáculos com uma pose diferente para o robô, ficando a estrutura como representada na Figura 6.1.

A componente de construção do mapa manteve-se inalterada, uma vez que a representação do campo sem expansão permanece na mesma. Assim apenas se alterou o método de expansão dos obstáculos e a forma de representação dos robôs que se encontram em cooperação com o principal.

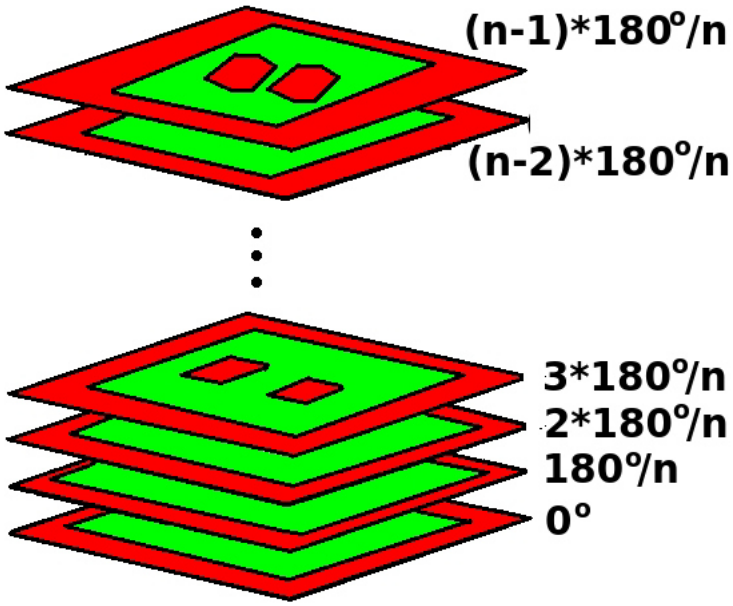


Figura 6.1: Estrutura de representação dos mapas

6.1.1 Expansão do mapa relativamente à orientação

Existem diferentes métodos de expansão de obstáculos, contudo, já que o polígono que representa o robô tem poucos vértices, foi escolhido um algoritmo de soma de *Minkowski* calculado analiticamente, pois os cálculos são de rápida execução, no caso de ser entre dois polígonos convexos $O(n+m)$ [46], em que n e m são o número de vértices de cada objeto, e são baseados na equação, semelhante à das diferenças de *Minkowski*, $Madd(A,B) = \{a+b|a \in A, b \in B\}$, sendo A o conjunto de pontos do robô e B é o conjunto de pontos do obstáculo. Assim para esta expansão o robô foi considerado como um retângulo, tal como representado na Figura 6.2.

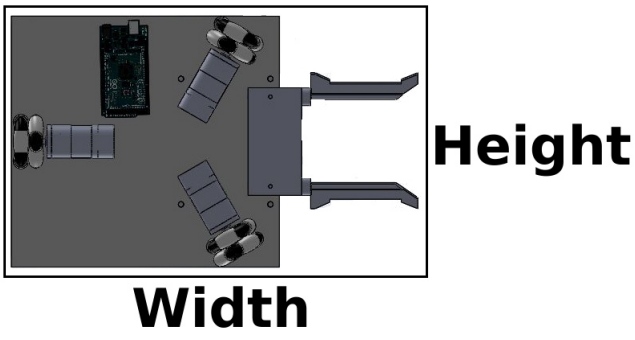


Figura 6.2: Retângulo que aproxima o robô com $\theta = 0^o$

Para facilitar o cálculo foram obtidos os vértices considerando que os objetos se encontravam centrados em $(X,Y) = (0,0)$. Para o cálculo dos vértices do robô foi executado um algoritmo que os define para uma orientação de 0^o e de seguida é aplicada a matriz de rotação descrita na

equação 6.1, por forma a obter as coordenadas após a rotação pretendida, fazendo o mesmo para os vértices dos obstáculos internos.

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix} * \begin{bmatrix} X \\ Y \end{bmatrix} \tag{6.1}$$

Após a obtenção dos vértices do robô a expansão do mapa foi realizada em duas etapas. Uma foi expandir os objetos no interior do campo usando somas de *Minkowski* e de seguida expandir os limites externos, paredes, uma vez que estes são sempre fixos e de fácil expansão recorrendo à distância entre o centro e os vértices com o maior X e o maior Y.

Começando por descrever o processo da expansão dos objetos no interior, após saber os vértices do robô e dos obstáculos realizam-se as somas entre as posições de todos eles, isto é todos os vértices do robô são somados com cada um dos vértices do obstáculo, como se pode observar um exemplo na Figura 6.3, em que os pontos a azul são o resultado da soma dos vértices e a preto os limites do obstáculo, obtendo um conjunto de pontos, devendo definir os contornos do obstáculo expandido com os que se encontram mais no exterior.

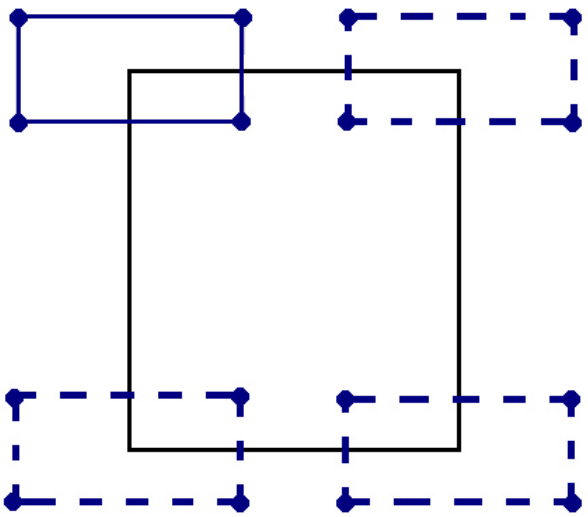


Figura 6.3: Exemplo de soma de *Minkowski*

Para calcular o contorno dos pontos utilizou-se um algoritmo de *Convex Hull* que retorna o polígono convexo de menor área que engloba todos os pontos. Entre os diversos existentes o algoritmo escolhido foi o *Jarvis Convex Hull*, pois para estruturas com um número pequeno de pontos é rápido, com complexidade temporal de $O(nh)$, em que h é o número de vértices do polígono [43]. Este método encontra inicialmente o ponto mais à esquerda e vai, contornando no sentido anti-horário os pontos que se encontram no exterior até formar o contorno completo. Por forma a garantir que os pontos se encontram no sentido anti-horário realiza-se o produto cruzado

entre dois vetores, calculado pela equação 6.2, referente à Figura 6.4, caso seja 0 o resultado são colineares, se for positivo está no sentido horário e se for negativo está no sentido anti-horário.

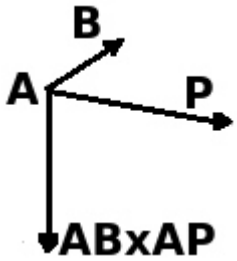


Figura 6.4: Vetores para o produto cruzado

$$AB\ x\ AP = (B.x - A.x) * (P.y - A.y) - (B.y - A.y) * (P.x - A.x) \tag{6.2}$$

Assim o algoritmo do *Jarvis Convex Hull* pode ser descrito da seguinte forma [47]:

Algoritmo *Jarvis Convex Hull*(Pontos,N_{Pontos})

Variáveis:

Next - Lista de pontos que definem o polígono convexo

l - posição mais à esquerda

p - ponto atual

q - ponto seguinte

prev_ q - lista do índices dos pontos já visitados

Pseudocódigo:

1: Se N_{Pontos} < 3 então terminar

2: Por todas as posições de prev_ q = -1

3: l:=0

4: De i:=1 até N_{Pontos}-1 fazer

4.1: Se Pontos[i].x < Pontos[l].x então l:=i

5: p:=l

6: Repetir

6.1: q := (p+1) % N_{Pontos}

6.2: De i:=0 até N_{Pontos}-1 fazer

6.2.1: Se pi x iq < 0 e prev_ q != 1 então q:=i

6.3: Adicionar Pontos[q] a Next

6.4: prev_ q[q] := 1

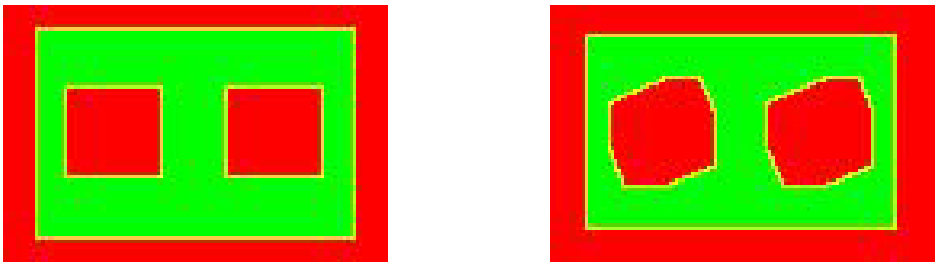
6.5: p:=q

7: Até p = l

Após obter as coordenadas do polígono convexo que aproxima o contorno efetua-se uma translação em (X,Y) para os pontos centrais dos obstáculos expandidos e imprime-se no *Bitmap* do campo, utilizando o código RGB que identifica ocupado.

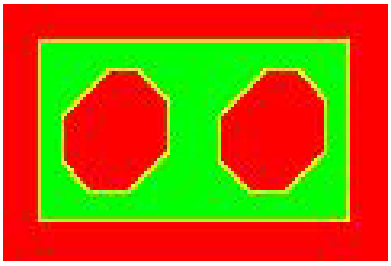
Abordando agora a expansão das paredes, o método baseia-se em, após saber os vértices do robô depois da rotação em torno de $(0,0)$, calcular o ponto com maior X, ou seja o mais à direita, e o com maior Y, o mais acima, e expandem-se as paredes da esquerda e da direita inserindo uma espessura igual a X e as superior e inferior inserindo uma espessura igual a Y.

Após expandir tudo percorre-se todo o *Bitmap* para adicionar uma camada protetora de alto custo, através da procura de todos os pontos livres que tenham pelo menos um vizinho ocupado. Após este passo converte-se o *Bitmap* para uma matriz e adiciona-se à pilha para cada ângulo desde 0^o até $\frac{180^o}{n_{ang}}$, em que n_{ang} é o número de ângulos que são pretendidos para a resolução em θ. Um exemplo com $n_{ang} = 8$ encontra-se no conjunto de Figuras 6.5

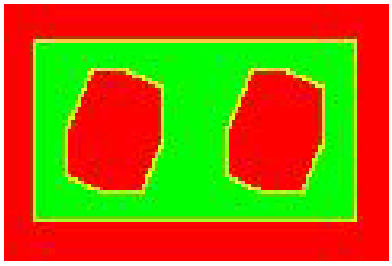


(a) Mapa com $\theta = 0^o$

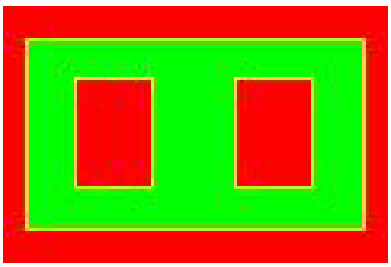
(b) Mapa com $\theta = 22.5^o$



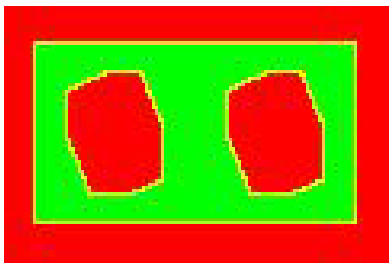
(c) Mapa com $\theta = 45^o$



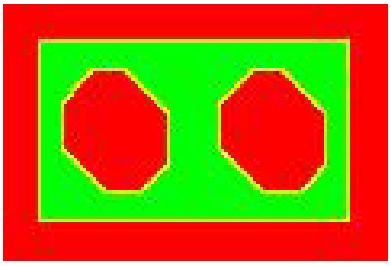
(d) Mapa com $\theta = 67.5^o$



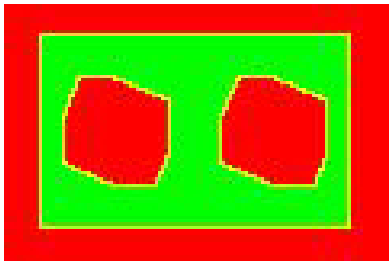
(e) Mapa com $\theta = 90^o$



(f) Mapa com $\theta = 112.5^o$



(g) Mapa com $\theta = 135^o$



(h) Mapa com $\theta = 157.5^o$

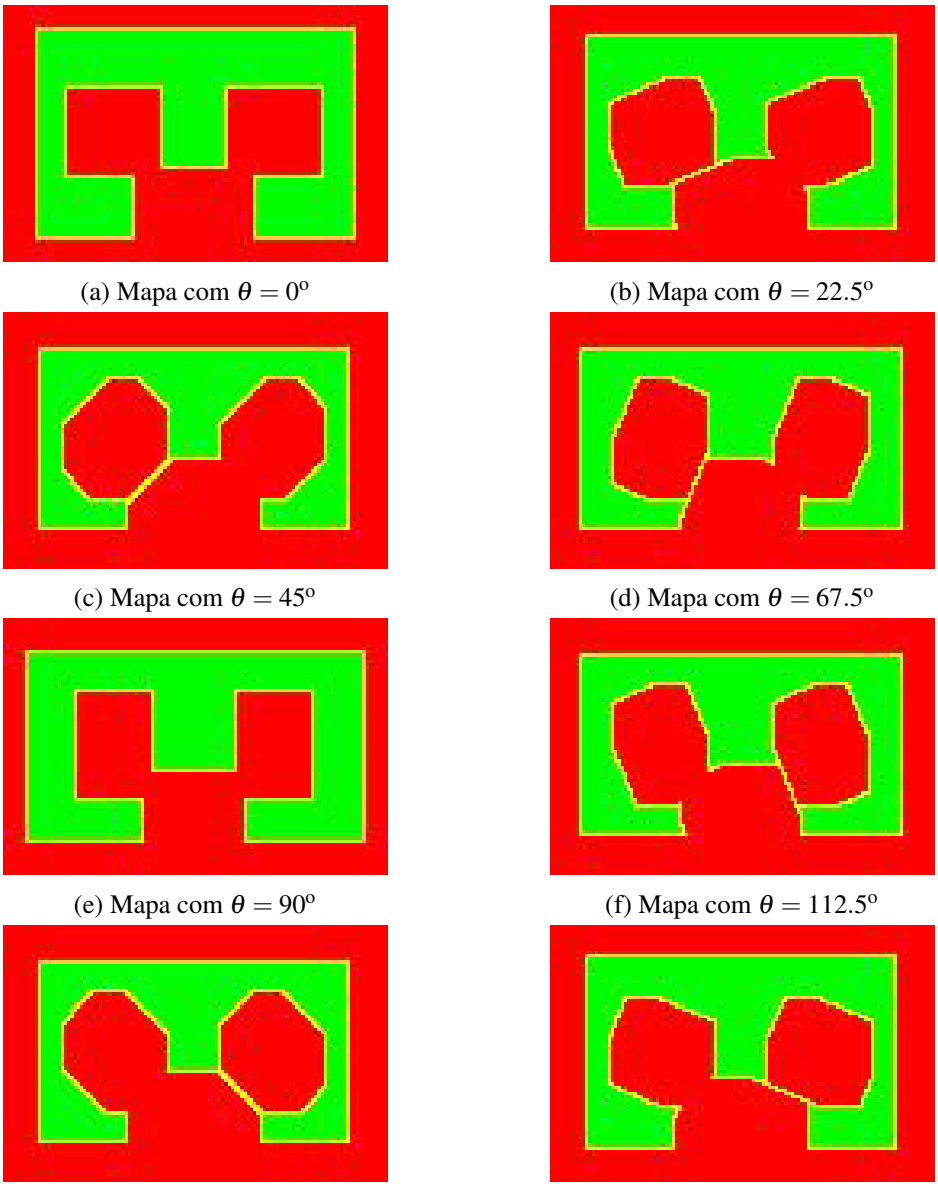
Figura 6.5: Exemplo de expansão do mapa com $n_{ang} = 8$

6.1.2 Alteração do mapa para múltiplos robôs consoante a orientação

As mudanças realizadas nesta parte foram maioritariamente na forma como os robôs externos são vistos, anteriormente era conhecida a sua posição em X e Y e representava-se o intruso através de um círculo. Após as alterações, com o intuito de acomodar as três dimensões associadas à pose do robô, passou ser conhecido, também, o seu θ, passando assim os agentes externos a serem representados através de um retângulo.

Desta forma expande-se o mapa como anteriormente descrito e de seguida procede-se a aplicar o algoritmo descrito para o enchimento de objetos no interior do mapa ao esboço do novo obstá-

culo, colocando-o depois com o centro na posição (X,Y) em que se encontra, assim o robô fica expandido e posicionado no devido local. No conjunto de Figuras 6.6 encontra-se um exemplo com $n_{ang} = 8$ e $(X, Y, \theta) = (0, -60, 0^o)$ desta expansão, onde X e Y se encontram em centímetros.



(a) Mapa com $\theta = 0^o$ (b) Mapa com $\theta = 22.5^o$

(c) Mapa com $\theta = 45^o$ (d) Mapa com $\theta = 67.5^o$

(e) Mapa com $\theta = 90^o$ (f) Mapa com $\theta = 112.5^o$

(g) Mapa com $\theta = 135^o$ (h) Mapa com $\theta = 157.5^o$

Figura 6.6: Exemplo de expansão do mapa com $n_{ang} = 8$ e segundo robô em $(X, Y, \theta) = (0, -60, 0^o)$

6.2 Algoritmo A^* para Espaço 3D

Uma vez que o novo método de construção de mapas possibilita um planeamento tridimensional (X,Y,θ) foram precisas algumas modificações no algoritmo para poder acomodar o novo

ambiente. As alterações executadas foram ao nível da estrutura dos nós, que passam a acomodar um novo parâmetro, a orientação que o robô necessita para respeitar a configuração que valida a passagem, θ.

Outra alteração foi realizada na representação dos nós vizinhos ao n_{melhor} que anteriormente eram apenas oito, passando agora a ser vinte e seis, tal como representado na Figura 6.7, onde R representa rotação sem translação, θ_{+1} e θ_{-1} representam rotações para a orientação anterior e seguinte e θ a orientação do nó pai. A existência deste incremento abrupto no número de nós vizinhos deve-se ao facto de agora haver uma pilha de mapas com diferentes possibilidades de passagem, o que conduz a que seja necessário analisar todas as configurações nos valores de θ anterior e seguinte, já que por vezes é possível existir cruzamento de robôs caso o estado de ambos seja favorável.

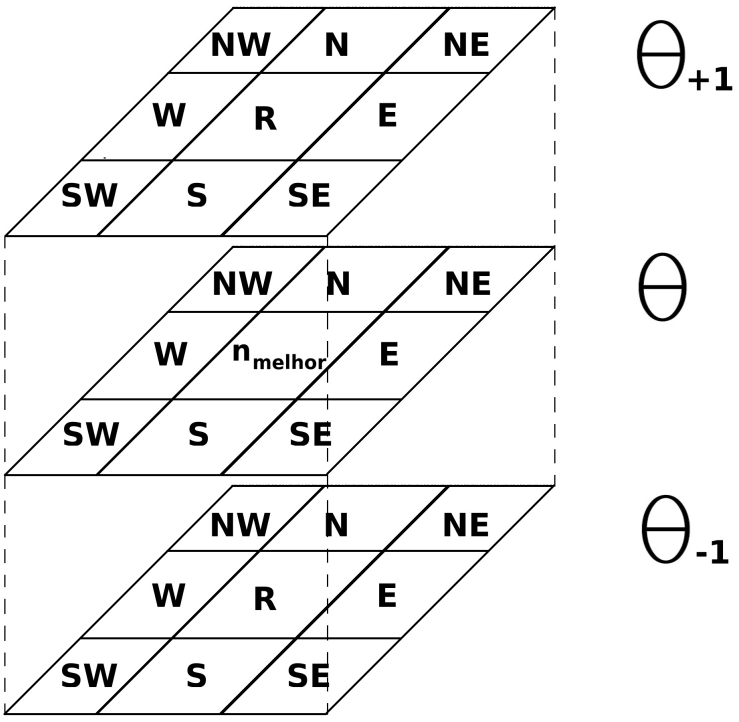


Figura 6.7: Representação dos nós vizinhos com mapa em (X,Y,θ)

Além do número de vizinhos também houve a alteração na atribuição dos custos que se encontram representados na equações 6.3, 6.4, 6.5, 6.6 e 6.7, sendo *Sizec* o tamanho das células e K_e os custos adicionais apresentados no capítulo 5.

$$c(n_{melhor}, n) = Sizec + K_e * Sizec \tag{6.3}$$

$$c(n_{melhor}, n) = Sizec * \sqrt{2} + K_e * Sizec \tag{6.4}$$

$$c(n_{melhor}, n) = Sizec + K_e * Sizec + Sizec/10 \tag{6.5}$$

$$c(n_{melhor}, n) = Sizec * \sqrt{2} + K_e * Sizec + Sizec/10 \tag{6.6}$$

$$c(n_{melhor}, n) = Sizec/10 \tag{6.7}$$

A equação 6.3 aplica-se aos custos para ir para os vizinhos N, S, E e W e equação 6.4 aplica-se aos custos para ir para os vizinhos NW, SW, SE e NE, ambas mantendo-se em θ, sem rotação.

A equação 6.5 aplica-se aos custos para ir para os vizinhos N, S, E e W e equação 6.6 aplica-se aos custos para ir para os vizinhos NW, SW, SE e NE, ambas rodando para θ_{+1} ou θ_{-1}. Como se pode ver as equações são semelhantes, mas adiciona-se um pequeno custo, pois senão seriam sempre adicionados à lista aberta os nós, mesmo quando não há vantagens em alterações dos ângulos.

Por fim a equação 6.7 é o custo para rodar, R, para θ_{+1} ou θ_{-1} sem mudar de posição.

Capítulo 7

Sistema de Navegação

Neste capítulo é apresentado o sistema de navegação que efetua as ações de coordenação do movimento do robô entre as máquinas e entre os robôs existentes no meio. É de referir que o processo de controlo se encontra otimizado para o funcionamento com o mapa construido em 2D.

Tal como se refere neste capítulo, este controlo é feito de forma hierárquica, começando por explicar os métodos de controlo de baixo nível, o controlo de movimentos, de seguida explica-se o controlo de ações, no qual se refere o algoritmo de mais alto nível, de seguida os modos de funcionamento e por fim é fornecido um exemplo simples de funcionamento.

7.1 Controlo de Movimentos

Para executar as trajetórias o robô necessita de ter um controlador que indique as velocidades (V, V_n, W) que são necessárias para o seu movimento seguir o pedido pelo planeamento efetuado. Uma vez que os movimentos realizados vão ser definidos por retas para entrar nas máquinas ou são de pequena dimensão devido ao facto das células serem de dimensões reduzidas, as manobras podem ser aproximadas por linhas retas mantendo na mesma uma certa suavidade nos movimentos, que se aproximam de curvas.

Assim torna-se necessário definir dois algoritmos, um de seguimento de linhas retas para realizar o movimento do ponto inicial para o final e um controlador de posição para quando a posição se encontrar muito perto da final, por forma a estabilizar em torno do ponto final.

7.1.1 Seguimento de linha

Para o controlador de seguimento de linhas foi implementado um método baseado no apresentado por *Conceição et al.* em 2006 [48]. Este método define os vetores de velocidade através da posição do robô e de um segmento de reta que une o ponto inicial (A) ao final (B). Na Figura 7.1 retirada de [48] encontra-se o esquemático do controlador, em que $P(X_r, Y_r)$ é a localização do robô, θ a orientação, *es* a distância do robô ao segmento de reta, α a inclinação do segmento $\overline{AB}$ e φ é a diferença entre o ângulo do segmento e a orientação do robô, isto é $\varphi = \theta - \alpha$.

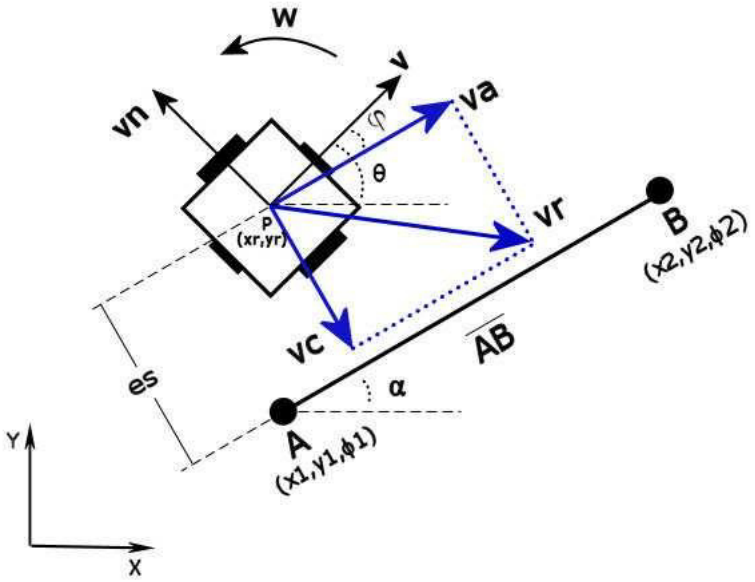


Figura 7.1: Esquema do controlo

Contudo foi aplicada uma mudança de referencial para facilitar a implementação, passando o segmento a estar alinhado com o eixo XX e o ponto final passar a ser o (0,0), tal como representado na Figura 7.2. Para realizar esta mudança de coordenadas aplicou-se uma matriz de rotação, equação 6.1 em que o θ representado na equação é igual a α, aos pontos A, B e $P(X_r, Y_r)$, obtendo, respetivamente, A', B' e $P'(X_r, Y_r)$. Seguidamente efetuou-se uma translação em (X,Y) de forma a colocar B' na posição (0,0), transladando A' e $P'(X_r, Y_r)$ de igual forma.

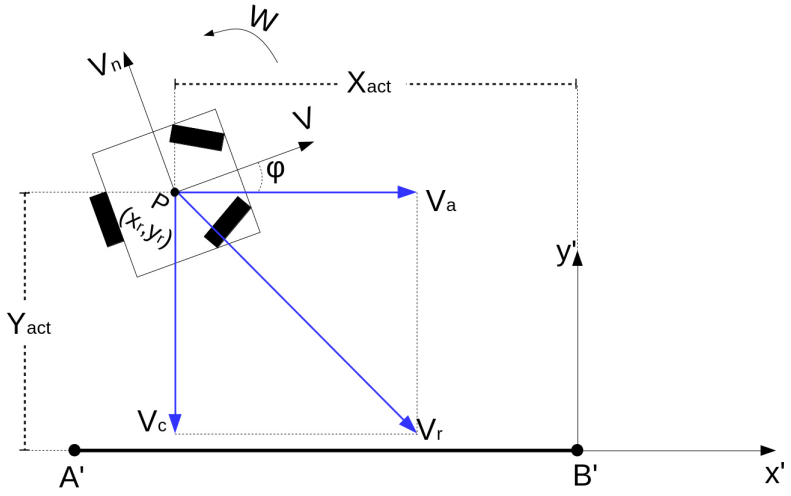


Figura 7.2: Esquema do controlo após mudança de coordenadas

Após efetuar a mudança de referencial, começa-se por calcular V_r que representa a velocidade linear desejada para o robô, e pode ser calculada atribuindo diferentes velocidades de referência uma V_{nom} que é a velocidade para o movimento normal e V_{fin} que é a velocidade para quando se aproxima da posição final. Assim a V_r pode ser calculado através da equação 7.1, onde X_{act}

é a distância em X' do robô ao ponto final, d_1 e d_2 são *thresholds* onde transitam as rampas de aceleração e de desaceleração e X_i é a posição inicial da reta, equivalente a A'.

$$V_r = \begin{cases} \frac{V_{nom} \cdot 0.8}{d_1 - X_i} \cdot X_{act} + V_{nom} \cdot \frac{0.2 \cdot d_1 - X_i}{d_1 - X_i}, & X_{act} < d_1 \\ V_{nom}, & d_1 \leq X_{act} \leq d_2 \\ \frac{V_{fin} - V_{nom}}{-d_2} \cdot X_{act} + V_{fin}, & X_{act} > d_2 \end{cases} \tag{7.1}$$

Assim a velocidade V_r fica representada pela curva que se encontra representada na Figura 7.3.

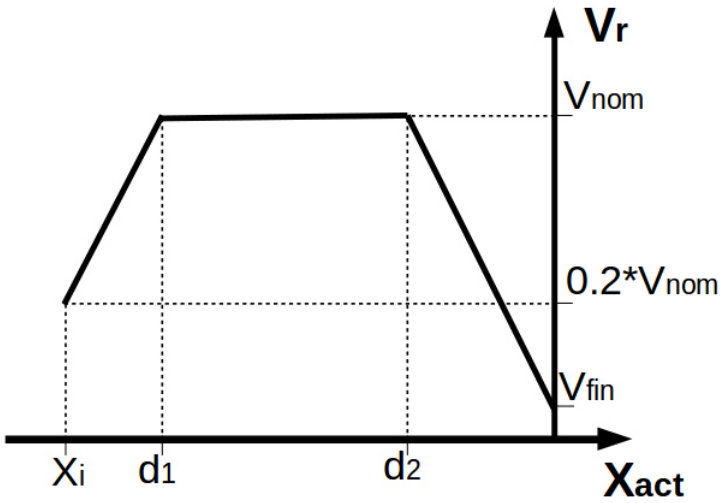


Figura 7.3: Curva representativa da velocidade de referência

De seguida calculam-se as velocidades V_c e V_a, em que V_c é a componente da velocidade perpendicular a $\overline{AB}$ ($V_a \perp \overline{AB}$), ou seja a velocidade de aproximação ao segmento e V_a é a componente da velocidade paralela a $\overline{AB}$ ($V_a \parallel \overline{AB}$), ou seja a velocidade de aproximação ao ponto final. Com isto as velocidades V_c e V_a são obtidas através das equações 7.2 e 7.3, respetivamente, sendo K_1 um ganho.

$$V_c = -Y_{act} * K_1 \tag{7.2}$$

$$V_a = \begin{cases} 0, & V_r^2 - V_c^2 \leq 0 \\ \sqrt{V_r^2 - V_c^2}, & V_r^2 - V_c^2 > 0 \end{cases} \tag{7.3}$$

Após obter estas velocidades calcula-se as velocidades do robô V e V_n usando uma matriz de rotação como a da equação 6.1, obtendo a equação 7.4.

$$\begin{bmatrix} V \\ V_n \end{bmatrix} = \begin{bmatrix} cos(\varphi) & sin(\varphi) \\ -sin(\varphi) & cos(\varphi) \end{bmatrix} * \begin{bmatrix} V_a \\ V_c \end{bmatrix} \tag{7.4}$$

Por fim para calcular a velocidade angular do robô (W) recorre-se à sua orientação (θ) e em orientações para o movimento normal e a aproximar-se do final, θ_{nom} e θ_{fin}, respetivamente. Assim calcula-se através destas duas orientações o θ_{ref}, equação 7.5, onde d é a projeção do ponto normalizada para o comprimento do segmento, de seguida calcula-se o erro entre o valor pretendido e o atual, equação 7.6, e, por fim, obtém-se W através da equação 7.7, em que K_2 é um ganho.

$$\theta_{ref} = \theta_{nom} * d + \theta_{fin} * (1 - d) \tag{7.5}$$

$$e_\theta = \theta_{ref} - \theta \tag{7.6}$$

$$W = e_\theta * K_2 \tag{7.7}$$

7.1.2 Controlo de posição

Ao terminar o percurso torna-se necessário aplicar um controlador que permita estabilizar o robô em torno do ponto final desejado. Para tal, da mesma forma que para o seguimento de linhas, é necessário realizar a mudança de referencial representada na Figura 7.2.

De seguida procede-se ao cálculo das velocidades V_a e V_c, para tal usam-se os erros na posição existentes em X e em Y, que neste referencial são dados por X_{act} e Y_{act}. As equações 7.8 e 7.9 representam o cálculo de V_a e V_c, em que K_3 e K_4 são ganhos.

$$V_a = -K_3 * X_{act} \tag{7.8}$$

$$V_c = -K_4 * Y_{act} \tag{7.9}$$

Após serem obtidos os valores de V_a e V_c, calcula-se o V e V_n através da equação anteriormente descrita, equação 7.4, e, por fim, aplicando as equações 7.6 e depois 7.7, com $\theta_{ref} = \theta_{fin}$, obtemos os valores de (V, V_n, W) para ajustar a posição final.

7.2 Controlo de Ações

Anteriormente foram descritos os controladores de baixo nível que permitem ao robô conhecer as velocidades (V, V_n, W), que são necessárias para seguir segmentos de reta e para estabilizar em torno de um ponto.

Agora torna-se imperativo introduzir uma máquina de estado de alto nível que execute o percurso de um posto de trabalho para o outro, seguindo as trajetórias necessárias para se deslocar no caminho ótimo e desviar-se de agentes externos que perturbam a passagem.

Contudo, uma vez que é utilizado um método de expansão do mapa que considera todo o comprimento do robô, incluindo o sistema de empilhamento de peças, existem situações em que é impossível calcular as trajetórias recorrendo ao método de planeamento sugerido devido a que

a posição da máquina se encontra numa zona ocupada, pois, como se referiu no capítulo 3, os armazéns estão embutidos nas paredes e as máquinas são inseridas como um obstáculo.

Assim é necessário implementar uma infraestrutura constituída por máquinas de estado hierárquicas que permitam decompor os percursos para funcionar em três modos:

- Entrada e saída das máquinas e armazéns
- Planeamento com recurso ao A^* para os percursos interiores
- Carga e descarga de peças

Com isto surgiu a necessidade de implementar um controlo de mais alto nível, que gere os modos de funcionamento para poder realizar as ações pretendidas. No nível hierárquico imediatamente abaixo surgem as máquinas de estado que controlam cada um dos modos de funcionamento.

De seguida são expostos cada uma destas estruturas de controlo, que se encontram otimizadas para o funcionamento com a construção do mapa em duas dimensões.

7.2.1 Controlo de deslocamento entre postos de trabalho

Como já foi referido os movimentos terão de ser controlados de forma modular e para isso recorreu-se a uma máquina de estados que indique para cada momento o modo de funcionamento a implementar para atingir o objetivo.

Assim para implementar este controlo de alto nível é necessário começar por definir os *waypoints* referentes a cada máquina e armazém, uma vez que é preciso dois modos de controlo de trajetórias, a representação de cada posto de trabalho será definido por dois pontos, um para indicar o local onde é obtida a entrada completa na máquina (A e C) e outro para indicar a posição em que já é possível efetuar o método de planeamento implementado (B e D), tal como representado na Figura 7.4, em que as retas a preto representam o modo de entrada na máquina e a azul o controlo em zonas internas.

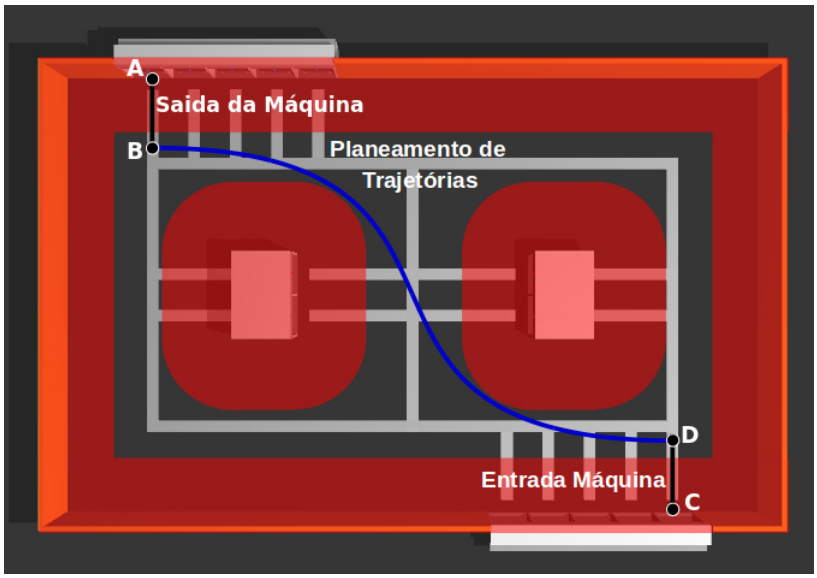


Figura 7.4: Decomposição dos percursos nos modos de funcionamento

Com isto o método implementado começa por realizar o movimento de saída do posto de trabalho, deslocando-se do ponto A para o B, quando estiver próximo da posição B alterna para o estado seguinte, começando o controlador com o planeamento de trajetória, para as zonas viáveis, começando de B para D e continuando a replanear a trajetória desde a localização atual, $P(X_r, Y_r)$, até ao objetivo, desta forma o algoritmo A^* está sempre a ser recalculado podendo atuar quando for representada uma mudança no meio circundante.

Quando for atingido o ponto final, D, realiza-se o controlo de posição por forma a garantir que o robô se encontra realmente no ponto idealizado e com a orientação devida para dar entrada na máquina.

Por fim, quando o erro for pequeno entre o ponto real e o pedido, efetua-se o algoritmo de entrada no posto de trabalho, igual ao de saída, mas em sentido contrário de D para C e atingindo a posição alvo efetua-se a estabilização nesse ponto, atuando as garras para carga ou descarga do material, segundo o estado pretendido (S).

A representação da máquina de estados que define este algoritmo encontra-se na Figura 7.5.

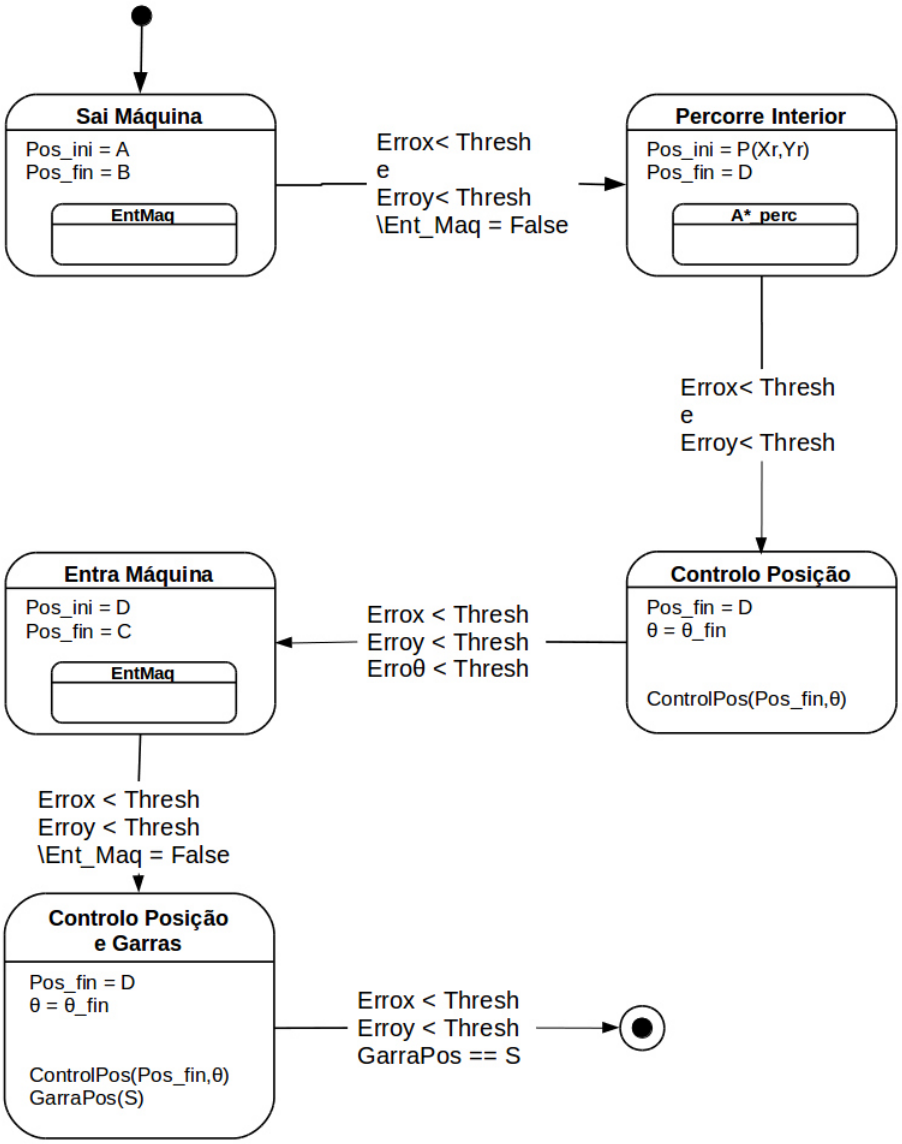
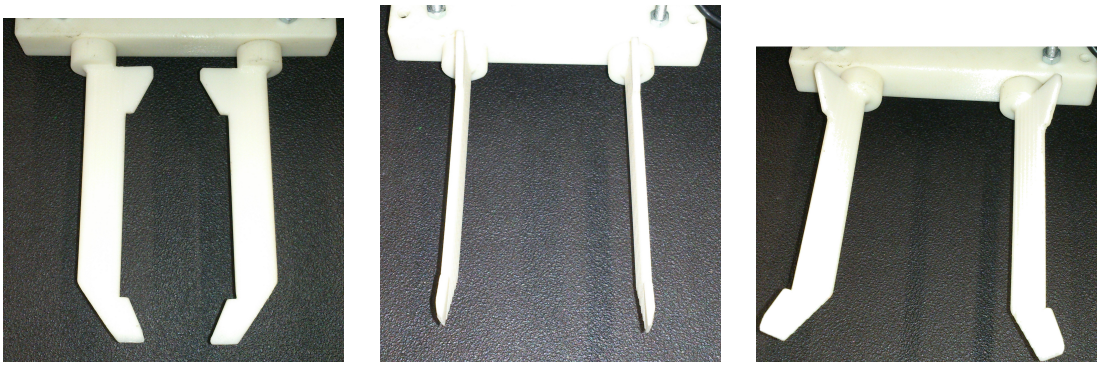


Figura 7.5: Máquina de estados para deslocamento entre postos de trabalho

7.2.2 Carga e descarga de peças

O controlo da carga e descarga efetua-se através de ordens para os servomotores, por forma a colocar as garras numa das três posições viáveis para atuar sobre as peças, em que a primeira é de desbloqueio das caixas, a segunda de elevação das caixas e a última de bloqueio do material, as quais se encontram expostas no conjunto de Figuras 7.6.

Este método é composto pela inserção da *flag* indicativa do estado pretendido, a qual é enviada de seguida para o controlador dos servomotores, a ordem é enviada durante um tempo, Δt, por forma a garantir que o sistema fica na posição correta.



(a) Posição de desbloqueio das caixas

(b) Posição de elevação das caixas

(c) Posição de bloqueio das caixas

Figura 7.6: Posições das garras para carga e descarga de materiais

7.2.3 Entrada e saída em postos de trabalho

Para efetuar as ações de aproximação dos postos de trabalho foi criada uma máquina de estados que transita de um método de movimentação calculado e dinâmico para um deslocamento pré-definido por forma a suplantar as limitações causadas pela expansão dos obstáculos, que retiram a possibilidade de introduzir o sistema de garras usado nas máquinas e armazéns existentes.

Com isto a máquina de estado que executa esta funcionalidade é constituída por dois estados:

- Inicial, onde se define o segmento de reta que realiza o trajeto entre dois pontos

- Final, em que se efetua o controlo de posição para estabilizar no ponto pretendido, corrigindo pequenos erros que possam surgir na posição

A entrada e a saída são diferenciadas através dos pontos iniciais e finais atribuídos. Sendo que para a entrada o ponto inicial definido encontra-se na zona marcada como livre no mapa, representado como B na Figura 7.4, e o final encontra-se no ponto de entrada no posto de trabalho, representado como A na Figura 7.4, sendo o sentido contrário a ação de saída.

Com isto a máquina de estados encontra-se representada na Figura 7.7, em que o M1 é o ponto inicial e o M2 é o ponto final.

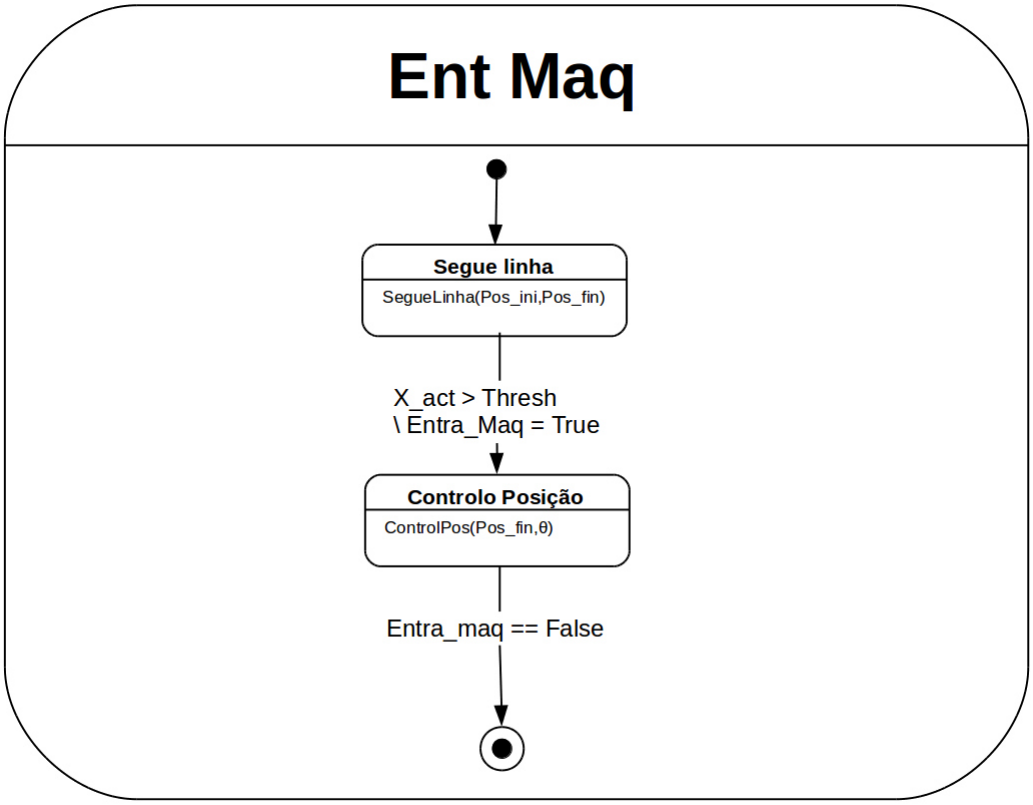


Figura 7.7: Máquina de estados para entrada e saída dos postos de trabalho

7.2.4 Movimentação usando planeamento de trajetória dinâmico

Ao contrário do modo de entrada e saída dos postos de trabalho, este modo de funcionamento necessita de cuidados extra devido à definição da trajetória ser feita de forma dinâmica, tendo como ponto inicial da trajetória a posição em que se encontra podendo a qualquer altura corrigir a trajetória a realizar consoante atuações externas, como por exemplo a presença de outro robô a movimentar-se, ou até mesmo deslizar para um ponto um pouco ao lado do planeado.

Assim o primeiro passo a tomar é verificar se o ponto final, que é o ponto da máquina na zona marcada como livre no mapa (B e D na Figura 7.4), está livre ou se existe lá algum obstáculo, nomeadamente outro robô. Caso não exista nenhuma impossibilidade o ponto definido como meta será mantido, senão se porventura houver um obstáculo nessa localização calcula-se o ponto mais próximo do final, usando métodos de projeção de pontos descrita em 7.2.4.1, e o robô desloca-se para lá até a posição final estar disponível.

Após estes estados de definição do destino efetua-se a geração da trajetória que o robô deverá seguir. Obtendo o caminho realiza-se o deslocamento entre nós através do algoritmo de seguimento de retas, que, uma vez que são distâncias curtas, aproximam facilmente a curva ideal. Contudo para suavizar a trajetória executada as retas foram traçadas de quatro em quatro nós presentes na lista obtida após o A^*, isto é invés de seguir a reta do nó 0 para o 1, seguia a reta do nó 0 para o 3 da lista, até atingir a posição final definida.

Em seguida encontra-se o pseudocódigo que define este controlo:

Algoritmo A^*_Perc

Variáveis:

Pos_{fin} - Posição final pretendida

Pos_{proj} - Posição da projeção do ponto final

Perc - Lista de nós que constituem o caminho

$A^*(A,B)$ - Cálculo da trajetória de A para B

$P(X_r,Y_r)$ - Posição atual do robô

SegueLinha(A,B) - Função de seguimento do segmento de reta de A para B

Pseudocódigo:

1: Se Pos_{fin} está desocupada então Perc:=$A^*(P(X_r,Y_r),Pos_{fin})$

2: Senão fazer projeção do ponto Pos_{fin}, Pos_{proj}, e fazer Perc:=$A^*(P(X_r,Y_r),Pos_{proj})$

3: Se size(Perc) > 0 então

3.1: Se size(Perc) > 3 então SegueLinha(Perc[0],Perc[3])

3.2: Senão SegueLinha(Perc[0],Pos_{fin})

De referir que este modo possibilita a atuação de múltiplos robôs em simultâneo, pois se realiza periodicamente o replaneamento enquanto está neste modo de funcionamento, tendo sempre em consideração a reconstrução do mapa consoante a posição do agente extra que se realiza em paralelo com o restante controlo.

7.2.4.1 Projeção da posição final

Como foi referido anteriormente no caso da posição final do planeamento se encontrar momentaneamente ocupada é necessário calcular o ponto livre mais próximo por forma a poder colocar o robô em espera até a posição final se encontrar livre.

Para tal começou-se por calcular o ângulo β formado entre a posição final, P_{fin}, e o ponto central do obstáculo, P_{obst}, neste caso considerado como um círculo, sendo calculado através da equação 7.10. Através deste ângulo calculam-se os valores em X e em Y através da projeção do vetor no círculo, sendo X_p obtido pela equação 7.11 e Y_p através de 7.12, em que R_{obst} é o raio do robô que constitui obstáculo e R_r o raio do próprio robô, sendo o resultado representado na Figura 7.8.

$$\beta = atan\left(\frac{P_{fin}.Y - P_{Obst}.Y}{P_{fin}.X - P_{Obst}.X}\right) \tag{7.10}$$

$$X_p = P_{Obst}.X + cos(\beta) * (R_{obst} + R_r) \tag{7.11}$$

$$Y_p = P_{Obst}.Y + sin(\beta) * (R_{obst} + R_r) \tag{7.12}$$

No caso de ser obtida uma posição válida esta é retornada, caso contrário efetua-se uma pesquisa em torno do círculo procurando as posições livres e comparando as distâncias de cada uma à posição do obstáculo, retornando a posição que se encontra mais próxima.

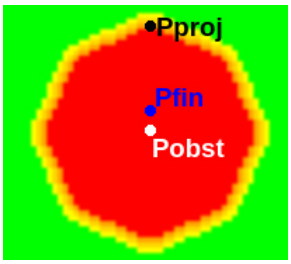
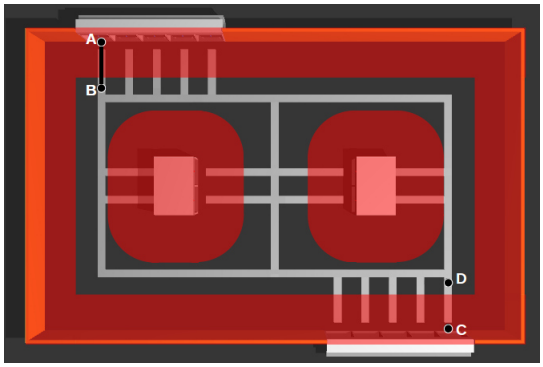


Figura 7.8: Exemplo de projeção de um ponto

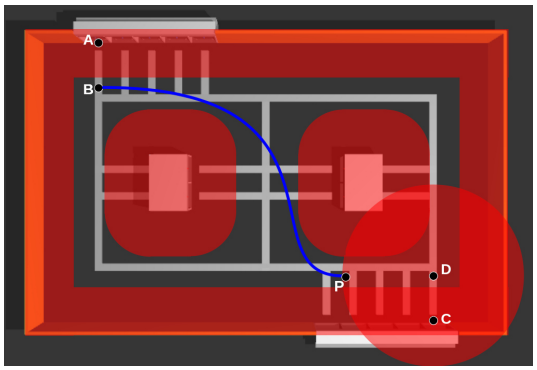
7.2.5 Exemplo de execução

De seguida procede-se à exemplificação de um percurso entre duas máquinas, representando os passos no conjunto de Figuras 7.9, em que A e B representam a máquina 1 e C e D a máquina 2.

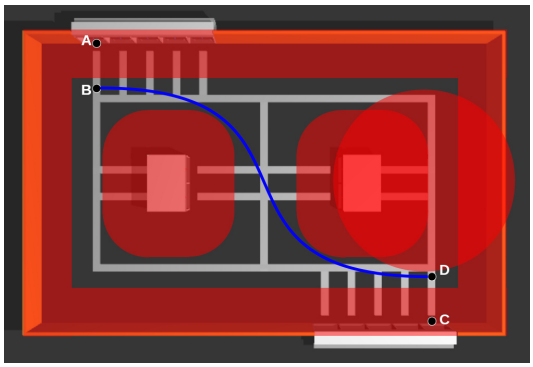
Como se pode ver na Figura 7.9a o robô inicia o percurso de saída da máquina realizado de A para B sem ter em conta o restante ambiente. Ao aproximar-se de B entra no modo que recorre ao planeamento de trajetórias e começa a executar a trajetória até ao ponto D, contudo como existe lá um robô a bloquear o acesso o agente planeia até ao ponto projetado, P, encontrando este estado representado na Figura 7.9b. Na Figura 7.9c o percurso altera uma vez que o robô secundário deixou de bloquear o ponto final, D, fazendo o planeamento para essa posição. Por fim é efetuado o movimento de entrada na máquina, de D para C, representado em 7.9d.



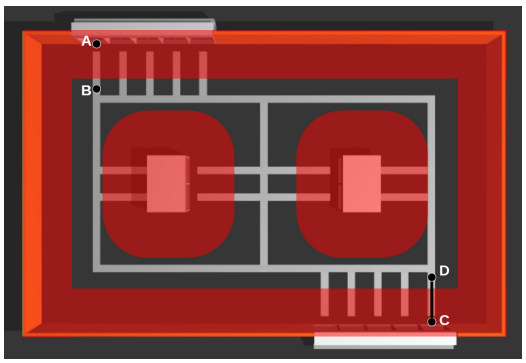
(a) Movimento de saída da máquina 1



(b) Movimento entre máquinas com posição final bloqueada



(c) Movimento entre máquinas com posição final livre



(d) Movimento de entrada na máquina 2

Figura 7.9: Exemplo de percurso entre a máquina 1 e a máquina 2

Capítulo 8

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) i7-4700MQ CPU @ 2.40GHz* e 8Gb de RAM.

Assim começa-se por apresentar os resultados dos métodos de construção de mapas implementados, passando para os algoritmos de pesquisa através do A^* e por fim os resultados obtidos com o sistema de navegação implementado.

8.1 Validação da Construção dos Mapas

Tanto a construção do mapa num espaço 2D, referenciado no capítulo 5, como a construção do mapa considerando a orientação, capítulo 6, passam pela representação do mapa sem expansão dos obstáculos, que é equivalente para ambos os métodos implementados.

Tabela 8.1: Dimensões do campo em cm

	Comprimento	**Largura**
Campo	300	200
Máquinas	24	36

Considerando um campo com as medidas apresentadas na tabela 8.1, em cm, obtém-se um mapa representado com duas cores, verde componente livre e vermelho ocupado. Contudo, tal como referido anteriormente, a resolução do mapa depende da dimensão das células usadas, assim quanto maior forem as células menor vai ser o mapa, com menos nós a representar. Na Figura 8.1 pode-se observar a diferença no tamanho dos mapas quando representados por células de 1cm, 2.5cm, 5cm e 7.5cm.

Na tabela 8.2 apresentam-se as dimensões do *Bitmap* obtido consoante a dimensão de células utilizada (número de colunas x número de linhas), assim como o número de nós representados.

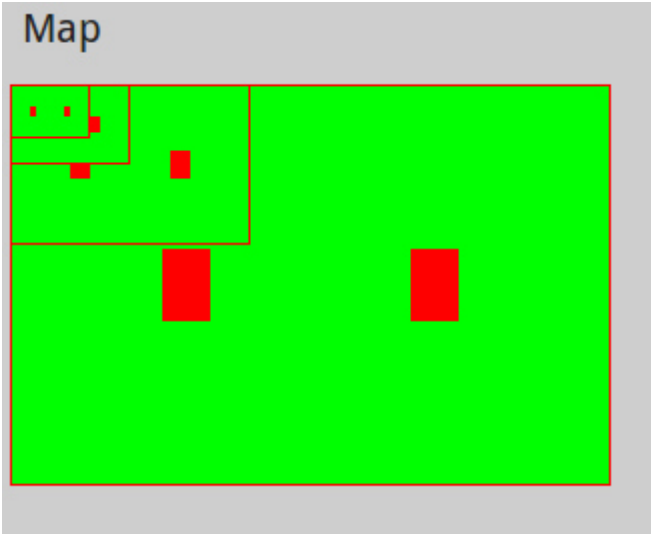


Figura 8.1: Variação no tamanho do mapa consoante o tamanho das células

Tabela 8.2: Variação nas dimensões e número de nós consoante o tamanho da célula

Tamanho da célula (cm)	**Dimensão** (NºColunas x NºLinhas)	**Número de nós**
1	300 x 200	60000
2.5	120 x 80	9600
5	60 x 40	2400
7.5	40 x 27	1080

Consoante a resolução usada existirão influências no tempo de execução, sendo que quanto menor for a resolução mais rápido será o algoritmo de expansão e a pesquisa usando A^*, contudo usando células de dimensões maiores que os obstáculos conduzem a perdas de informação, nomeadamente as máquinas centrais, como se pode ver na Figura 8.2, que se encontra aumentada para poder ser visível.

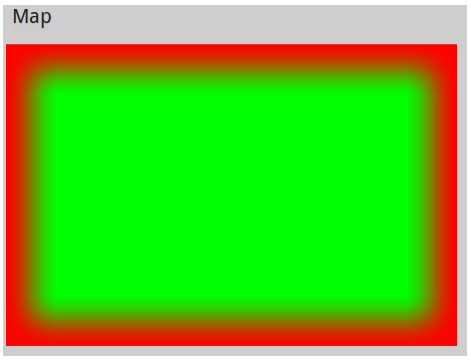


Figura 8.2: *Bitmap* construído com resolução de 30cm

De referir que o referencial utilizado para o campo se encontra representado na Figura 8.3, sendo as posições respetivas atribuídas em cm.

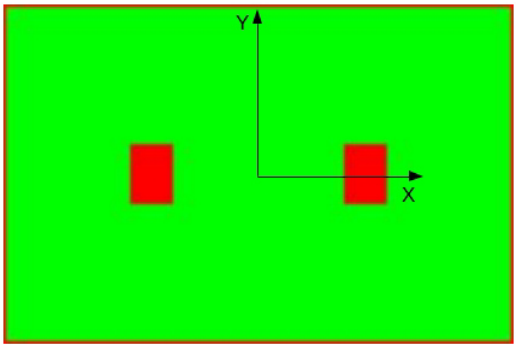
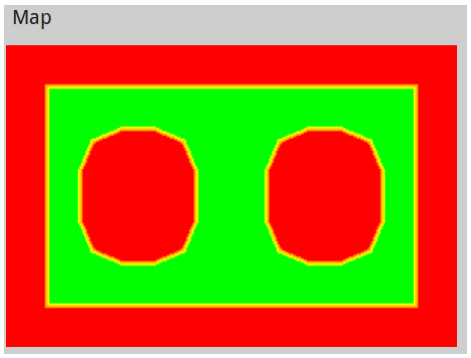


Figura 8.3: Sistema de eixos usados para representação do campo

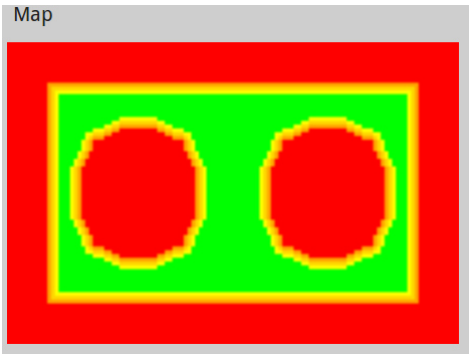
8.1.1 Expansão do mapa sem considerar orientação

A expansão varia consoante diversos parâmetros que se devem adaptar ao robô a usar e ao ambiente circundante, nomeadamente o raio do robô (R), que não deve ser menor que o raio do robô real, pois assim não garante que não existe colisão, e também o tamanho das células, que quanto maior forem menos espaço livre existe para o agente se movimentar, podendo causar a perda de percursos normalmente viáveis.

Nas Figuras 8.4a, 8.4b, 8.4c, 8.4d e 8.4e encontram-se representações de expansões com as células de dimensão 1cm, 2.5cm, 5cm, 7.5cm e 24cm, utilizando um raio do robô (R) de 25cm.



(a) Expansão do mapa com célula de 1cm



(b) Expansão do mapa com célula de 2.5cm

(c) Expansão do mapa com célula de 5cm

(d) Expansão do mapa com célula de 7.5cm

(e) Expansão do mapa com célula de 24cm

Figura 8.4: Expansão do mapa com diferentes tamanhos de célula com R=25cm

As Figuras anteriores sofreram uma mudança de escala por forma a tornarem-se visíveis e, tal como se pode observar, quanto maior forem as células, menor é a área onde o robô se pode movimentar, menor o número de nós livres, conduzindo a situações em que se perdem percursos viáveis, como é o caso da expansão com células de 24cm.

Também ao nível de camadas de proteção usadas deve-se adaptar consoante o tamanho de célula usada, pois quanto maior for a célula deve-se usar menos camadas de proteção, uma vez que a distância que aplicam no robô compensam o número de camadas, isto é se as células tiverem 7.5cm uma camada única salvaguarda uma distância equivalente a três camadas com 2.5cm.

Outros efeitos da variação do tamanho das células é a variação no tempo de processamento

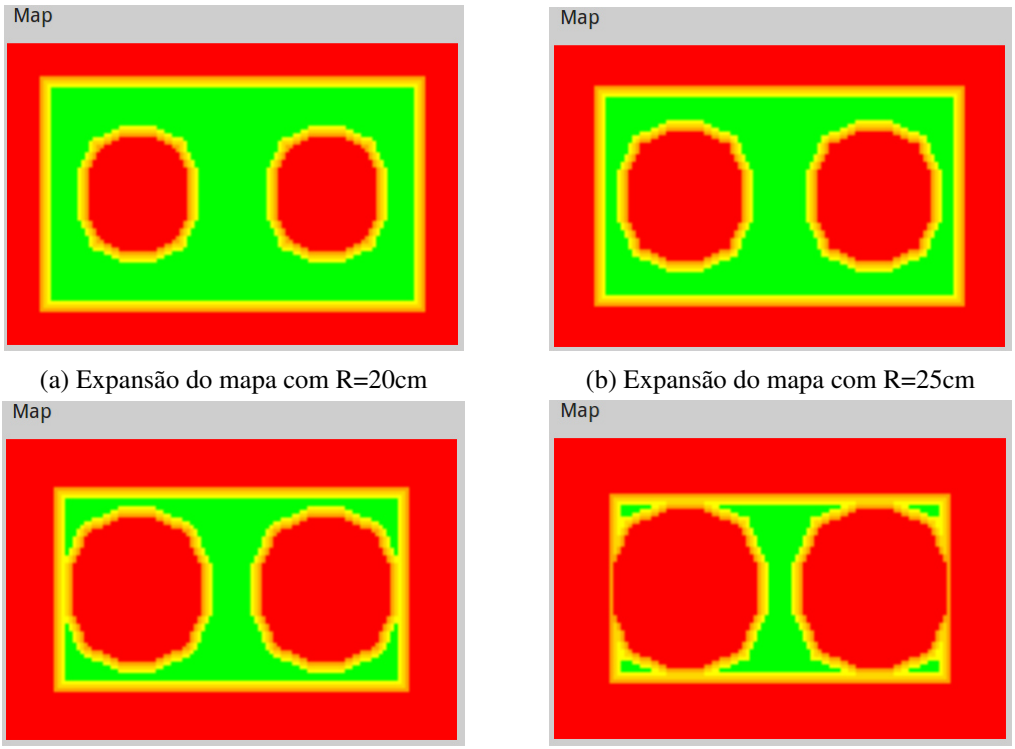
dos algoritmos e, consequentemente, no tempo para obter o mapa final expandido. Na tabela 8.3 apresentam-se o valores obtidos para estas características com os tamanhos de grelha referidos anteriormente.

Tabela 8.3: Expansão do mapa variando o tamanho das células para R=25cm

Tamanho da célula (cm)	**Número de Nós**	**Número de Nós Livres**	**Número de Camadas**	**Tempo** (ms)
1	60000	25640	5	32
2.5	9600	3956	3	8
5	2400	900	1	4
7.5	1000	370	1	2
24	96	16	0	-

Como se pode ver, quanto maior a célula, menor será o número de nós gerados, logo mais rápido será o processamento, mas a capacidade de gerar o caminho ótimo fica reduzida, devido à aproximação das posições reais para o nó em questão, representado pela posição central da célula, e causa, também, uma degradação na representação dos obstáculos, nomeadamente nas zonas arredondadas. Ainda o facto de ter um número inferior de nós livres conduz a um processamento mais rápido por parte do algoritmo A^*, já que o espaço de pesquisa é menor.

Variando o raio do robô, os tempos de processamento não são afetados, apenas variando o número de nós livres. Nas Figuras 8.5a, 8.5b, 8.5c e 8.5d expõe-se os resultados de variar o raio entre 20cm, 25cm, 30cm e 35cm respetivamente, com tamanho de células 2.5cm. Sendo na tabela 8.4 expostos o valores obtidos para o número de nós livres e o tempo de processamento.



(a) Expansão do mapa com R=20cm

(b) Expansão do mapa com R=25cm

(c) Expansão do mapa com R=30cm

(d) Expansão do mapa com R=35cm

Figura 8.5: Expansão do mapa com diferentes raios do robô com tamanho de células 2.5cm

Tabela 8.4: Expansão do mapa variando o tamanho do robô para células com 2.5cm

Raio do Robô (cm)	**Número de Nós**	**Número de Nós Livres**	**Tempo** (ms)
20	9600	4980	9
25	9600	3956	9
30	9600	2924	9
35	9600	1876	8

Tal como se pode observar aumentando o raio, os obstáculos ficam mais expandidos, reduzindo o número de nós livres sem, contudo, afetar o tempo para a criação do mapa. Desta forma o R escolhido deve ser sempre igual ou superior ao do robô, por forma a garantir a inexistência de colisões, sendo que para aumentar a segurança das trajetórias, apesar de sair da otimilidade, pode-se escolher um raio superior ao real. Apesar disto se o robô tiver um raio exageradamente elevado pode conduzir a percursos inviáveis.

8.1.1.1 Alteração do mapa com múltiplos robôs

No caso de existirem múltiplos robôs o mapa necessita ser reconstruido para representar o agente auxiliar na posição onde este se localiza, procedendo à expansão do mapa.

Tal como na expansão sem considerar os restantes robôs, os tamanhos de célula afetam o tempo de processamento, número de nós livres, o número de camadas a usar e a resolução, apresentando um comportamento igual, em que os tempos, uma vez que se procede à adição de apenas mais uma forma no *Bitmap*, não sofrem alterações visíveis.

A característica mais variante em relação ao método do robô isolado é o número de nós livres, a qual consoante a posição do robô pode variar, pois pode estar expandido sobrepondo-se a um maior ou menor número de nós normalmente vistos como ocupados.

Assim nas Figuras 8.6a, 8.6b, 8.6c e 8.6d apresentam-se as expansões com o segundo robô nas posições $(X,Y) = (0,0)$, $(X,Y) = (0,50)$, $(X,Y) = (50,50)$ e $(X,Y) = (113,50)$, respetivamente, em que o mapa tem a origem localizada no seu centro. O raio do robô principal é de 25cm e o tamanho das células é 2.5cm.

(a) Expansão do mapa com segundo robô em $(X,Y) = (0,0)$

(b) Expansão do mapa com segundo robô em $(X,Y) = (0,50)$

(c) Expansão do mapa com segundo robô em $(X,Y) = (50,50)$

(d) Expansão do mapa com segundo robô em $(X,Y) = (113,50)$

Figura 8.6: Expansão do mapa com segundo robô em diferentes posições, tamanho de células 2.5cm e R=25

Como se vê nas Figuras os mapas têm áreas a verde de dimensão variável, podendo ser consultado o número de nós livres na tabela 8.5.

Tabela 8.5: Expansão do mapa variando a localização do segundo robô

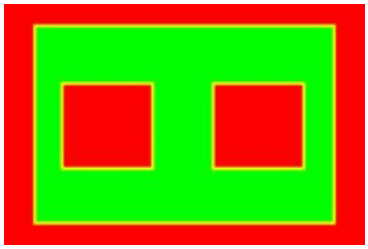
Posição do Segundo Robô (X,Y)(cm)	**Número de Nós Livres**
$(0,0)$	3171
$(0,50)$	3137
$(50,50)$	3388
$(113,50)$	3508

8.1.2 Expansão do mapa consoante a orientação

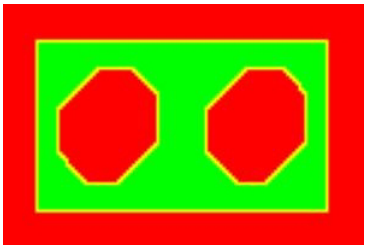
Na expansão tendo em consideração a orientação é definida uma pilha de mapas, em que cada um representa o campo expandido com o retângulo que representa o robô com a orientação θ. Este método em relação à expansão sem considerar o ângulo, apresenta a vantagem de ser menos restritiva nas manobras, uma vez que vê o robô pelo formato aproximado dele, em que permite a passagem num local onde outrora não passava desde que se desloque com a atitude correta.

Tal como no outro método anteriormente apresentado, este método sofre alterações consoante o tamanho de célula a usar, respondendo da mesma forma, isto é quanto menor a resolução menor o tempo de processamento, menor será o número de células livres e, com isto, maior será o risco de se perder caminhos válidos. No caso deste algoritmo apenas está a ser gerada uma camada de proteção, podendo-se compensar a perda na segurança com o aumento das dimensões do robô.

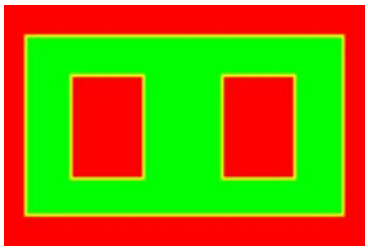
As Figuras 8.7, 8.8 e 8.9 representam esta expansão para variações de orientação de 45º, 22.5º e 10º respetivamente, sendo o tamanho das células de 2.5cm e o robô é definido como um retângulo de dimensões 45 x 30 cm.



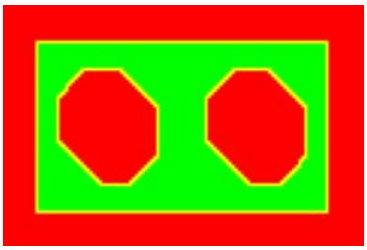
(a) Expansão do mapa com $\theta = 0^o$



(b) Expansão do mapa com $\theta = 45^o$

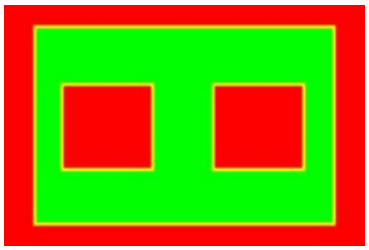


(c) Expansão do mapa com $\theta = 90^o$

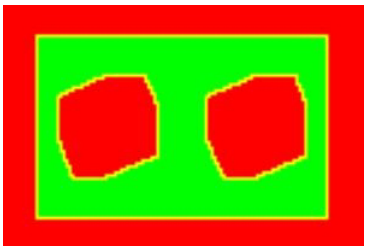


(d) Expansão do mapa com $\theta = 135^o$

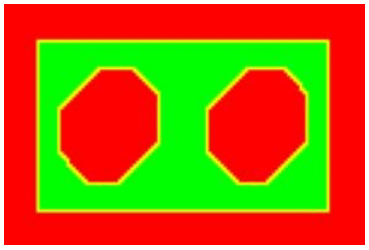
Figura 8.7: Expansão do mapa consoante variação de orientação de 45º



(a) Expansão do mapa com $\theta = 0^o$

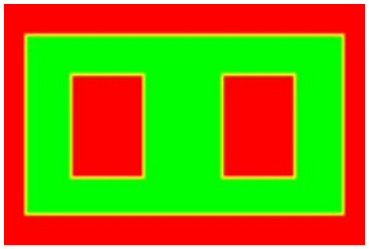


(b) Expansão do mapa com $\theta = 22.5^o$

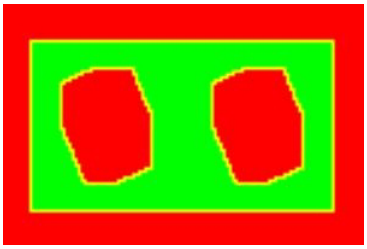


(c) Expansão do mapa com $\theta = 45^o$

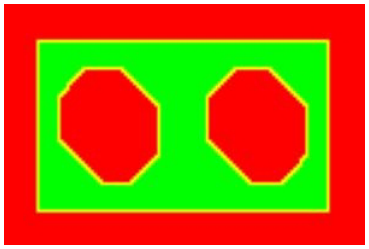
(d) Expansão do mapa com $\theta = 67.5^o$



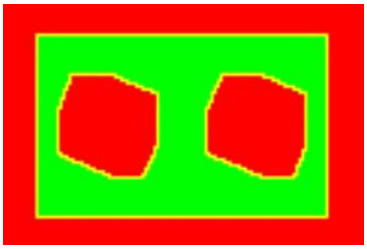
(e) Expansão do mapa com $\theta = 90^o$



(f) Expansão do mapa com $\theta = 112.5^o$

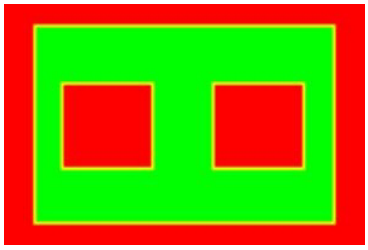


(g) Expansão do mapa com $\theta = 135^o$

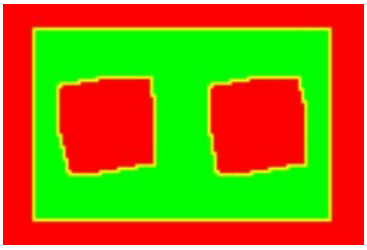


(h) Expansão do mapa com $\theta = 157.5^o$

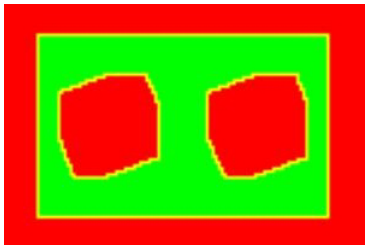
Figura 8.8: Expansão do mapa consoante variação de orientação de 22.5^o



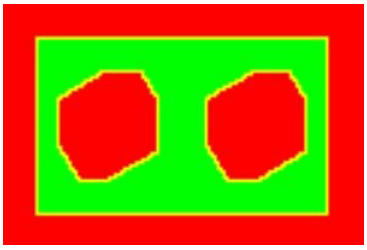
(a) Expansão do mapa com $\theta = 0^o$



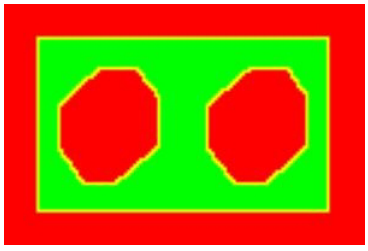
(b) Expansão do mapa com $\theta = 10^o$



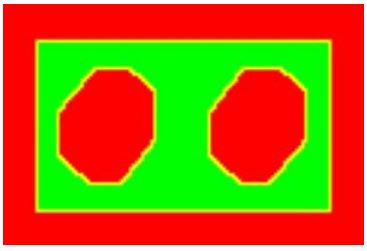
(c) Expansão do mapa com $\theta = 20^o$



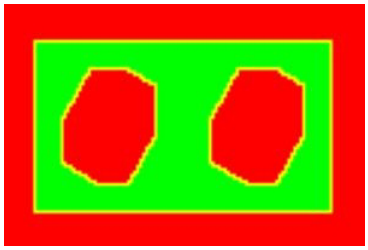
(d) Expansão do mapa com $\theta = 30^o$



(e) Expansão do mapa com $\theta = 40^o$

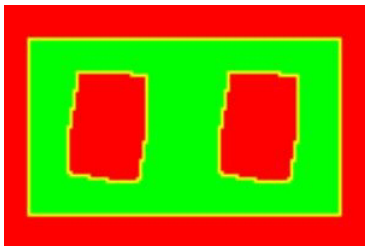


(f) Expansão do mapa com $\theta = 50^o$

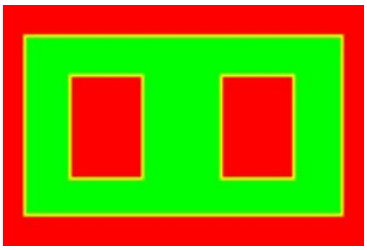


(g) Expansão do mapa com $\theta = 60^o$

(h) Expansão do mapa com $\theta = 70^o$



(i) Expansão do mapa com $\theta = 80^o$



(j) Expansão do mapa com $\theta = 90^o$

Figura 8.9: Expansão do mapa consoante variação de orientação de 10^o

Na Figura 8.9 é de notar que para a variação da orientação de 10º não se encontram representados todos os mapas, constituído por 18 ângulos, sendo os restantes semelhantes, mas espelhados em relação ao X.

Como se pode ver nas Figuras quanto maior for o número de ângulos maior vai ser o espaço de pesquisa do algoritmo de pesquisa o que trará custos adicionais de tempo, como será exposto posteriormente. Também se pode observar que, ao contrário dos resultados obtidos na subsecção 8.1.1, os obstáculos apresentam formas que variam com o θ do robô, o que altera as posições válidas, podendo passar em pontos mais diversificados. No processo de expansão também são adicionados custos temporais quanto maior for a resolução adotada para a orientação, isto é o número de ângulos a considerar, estes custos podem ser verificados na tabela 8.6.

Tabela 8.6: Expansão do mapa variando o número de orientações a considerar

Número de Orientações (cm)	**Número de Nós**	**Número de Nós Livres**	**Tempo** (ms)
4	38400	17264	6
8	76800	33524	11
18	172800	75054	21

Como se pode observar relativamente ao outro método existe um número muito superior de nós livres para visitar no algoritmo de pesquisa, contudo não existe uma discrepância acentuada nos tempos de processamento, uma vez que este algoritmo é realizado via cálculo analítico onde os objetos têm um número reduzido de vértices.

Ao alterar o tamanho do robô o efeito que terá sobre os mapas, tal como no outro método, é de diminuir o espaço livre para o robô se deslocar, levando a que um robô demasiado grande não possa passar. Apesar disso como o robô é visto como um retângulo poderá efetuar manobras caso apenas uma das dimensões seja elevada.

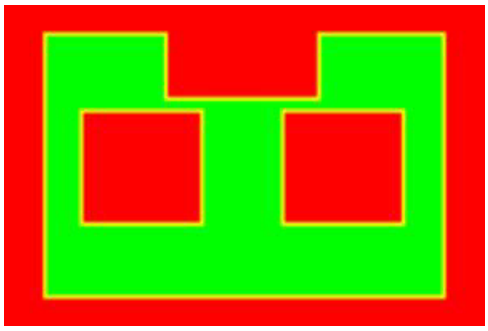
Assim trata-se de uma alternativa viável para a construção do ambiente, tratando-se de um método rápido, eficaz e, com a devida parametrização, seguro, mas que contudo causa uma penalização no grafo de pesquisa.

8.1.2.1 Alteração do mapa com múltiplos robôs considerando a orientação

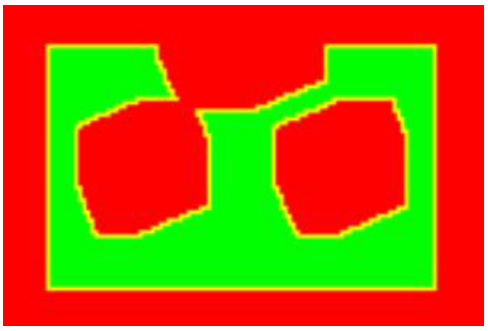
A alteração do mapa com o segundo robô vai necessitar reconstruir o mapa consoante a posição dele, representado-o como um retângulo com a respetiva orientação.

Desta forma a característica mais variante será o número de nós livres para a pesquisa, uma vez que se introduz um novo elemento no mapa que constitui um obstáculo que necessita ser contornado. Ao nível de tempo de processamento os tempos para fazer a reconstrução encontram-se dentro dos obtidos na tabela 8.6.

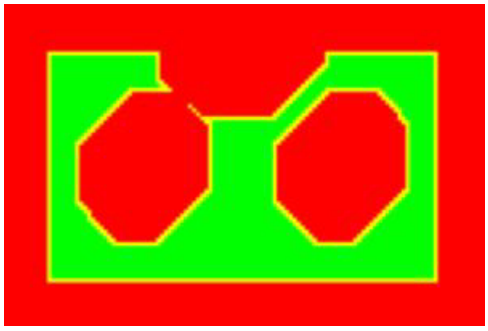
Nos conjuntos de Figuras 8.10, 8.11 e 8.12 encontra-se a expansão do mapa considerando a orientação com um segundo robô nas poses $(X,Y,\theta) = (0,74,0^{o})$, $(X,Y,\theta) = (0,74,30^{o})$ e $(X,Y,\theta) = (0,74,90^{o})$, respetivamente, sendo o tamanho das células de 2.5cm e ambos os robôs têm dimensões de 45 x 30 cm.



(a) Expansão do mapa com $\theta = 0^o$

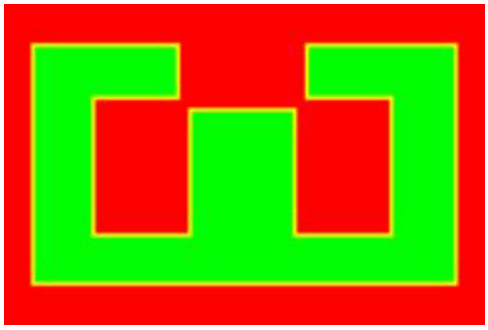


(b) Expansão do mapa com $\theta = 22.5^o$

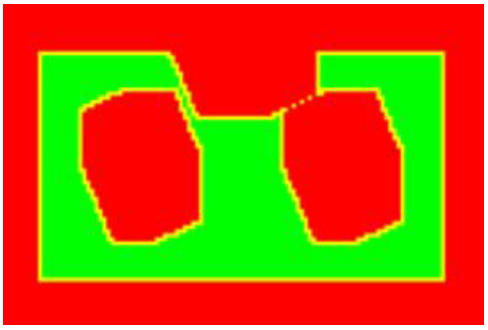


(c) Expansão do mapa com $\theta = 45^o$

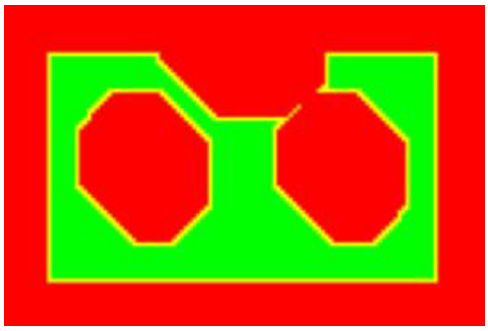
(d) Expansão do mapa com $\theta = 67.5^o$



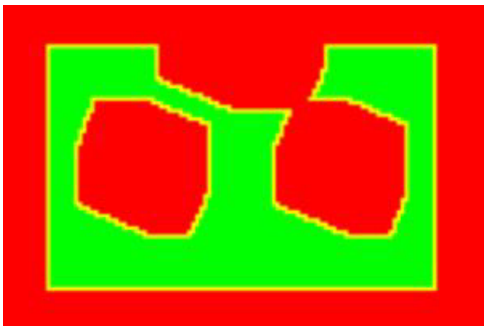
(e) Expansão do mapa com $\theta = 90^o$



(f) Expansão do mapa com $\theta = 112.5^o$

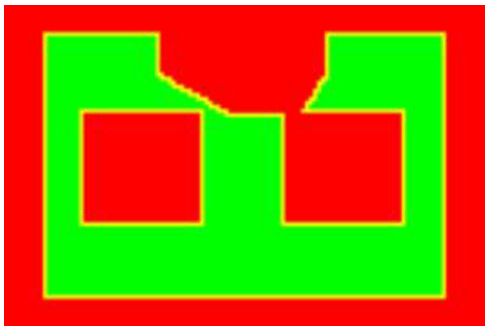


(g) Expansão do mapa com $\theta = 135^o$

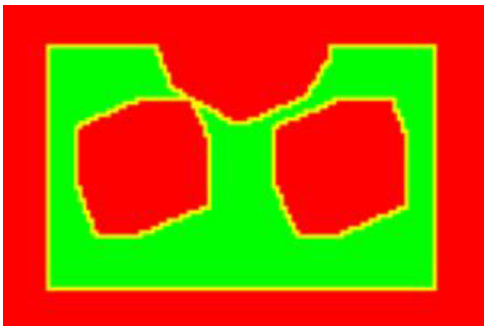


(h) Expansão do mapa com $\theta = 157.5^o$

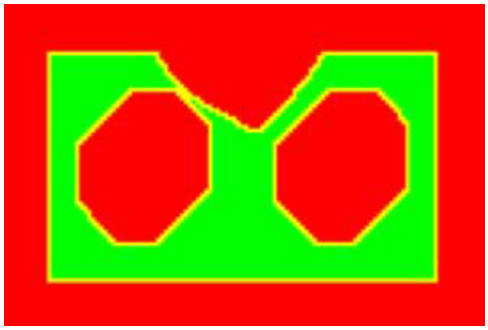
Figura 8.10: Expansão do mapa consoante variação de orientação com segundo robô na pose $(X, Y, \theta) = (0, 74, 0^o)$



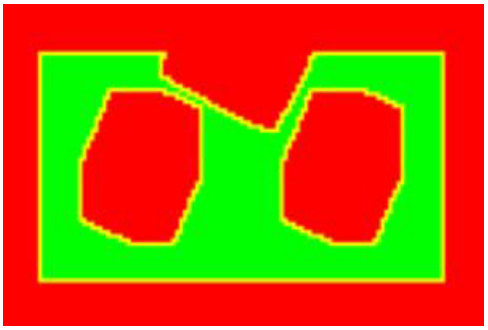
(a) Expansão do mapa com $\theta = 0^o$



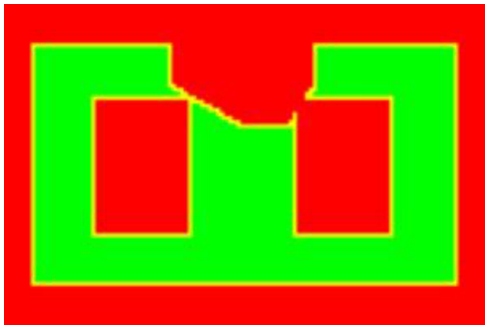
(b) Expansão do mapa com $\theta = 22.5^o$



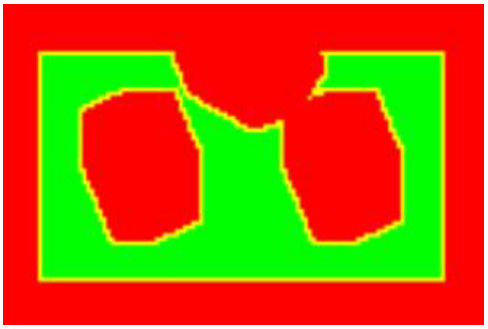
(c) Expansão do mapa com $\theta = 45^o$



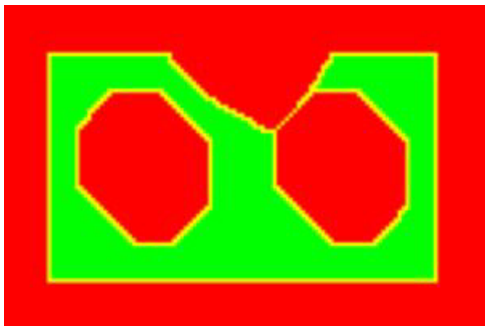
(d) Expansão do mapa com $\theta = 67.5^o$



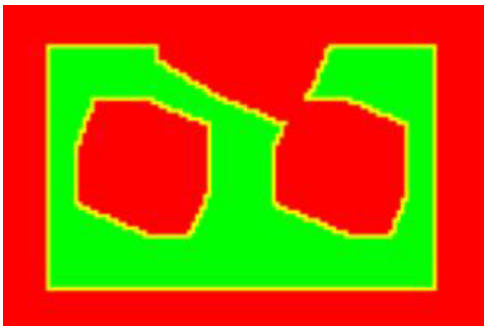
(e) Expansão do mapa com $\theta = 90^o$



(f) Expansão do mapa com $\theta = 112.5^o$

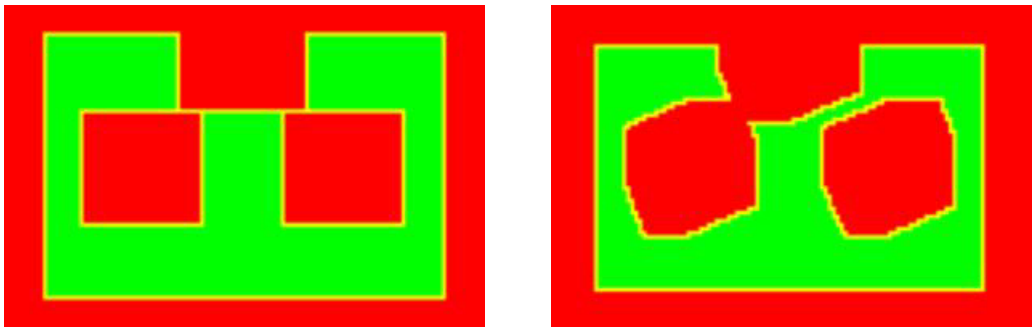


(g) Expansão do mapa com $\theta = 135^o$



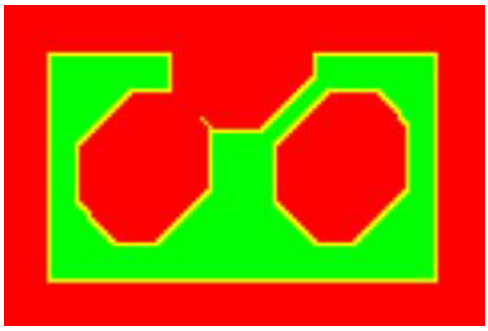
(h) Expansão do mapa com $\theta = 157.5^o$

Figura 8.11: Expansão do mapa consoante variação de orientação com segundo robô na pose $(X, Y, \theta) = (0, 74, 30^o)$

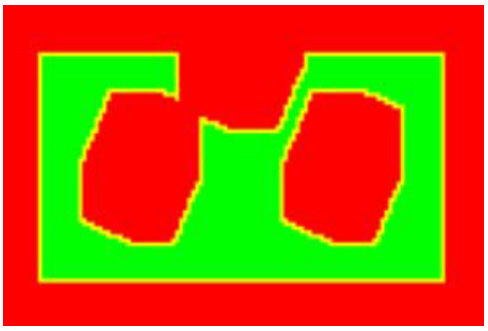


(a) Expansão do mapa com $\theta = 0^o$

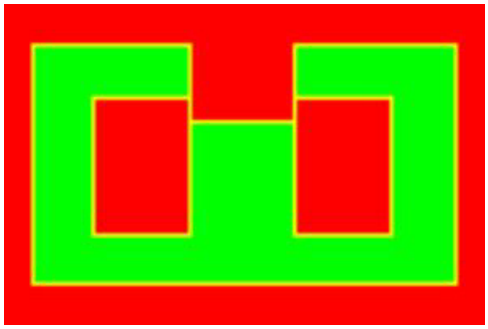
(b) Expansão do mapa com $\theta = 22.5^o$



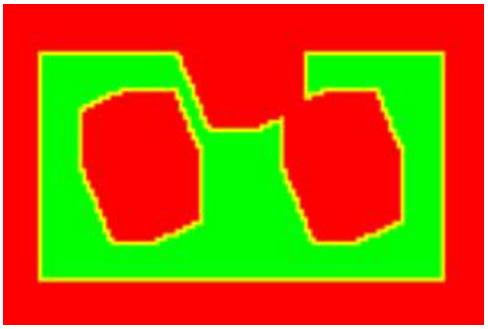
(c) Expansão do mapa com $\theta = 45^o$



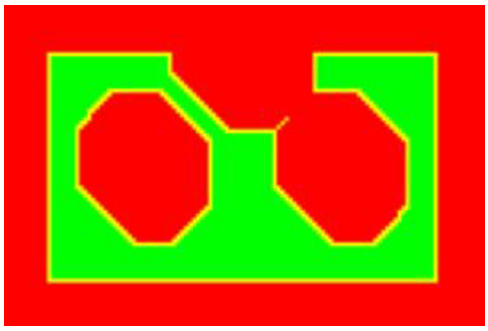
(d) Expansão do mapa com $\theta = 67.5^o$



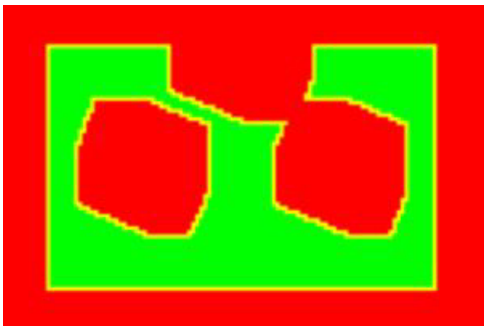
(e) Expansão do mapa com $\theta = 90^o$



(f) Expansão do mapa com $\theta = 112.5^o$



(g) Expansão do mapa com $\theta = 135^o$



(h) Expansão do mapa com $\theta = 157.5^o$

Figura 8.12: Expansão do mapa consoante variação de orientação com segundo robô na pose $(X, Y, \theta) = (0, 74, 90^o)$

Tal como se pode ver nas Figuras anteriores as representações das expansões variam consoante a orientação do segundo robô, isto é o conjunto de mapas para o que tem o segundo robô em

$(X,Y,\theta) = (0,74,0^o)$ são diferentes dos $(X,Y,\theta) = (0,74,30^o)$.

Comparativamente ao outro método apresentado a grande vantagem advém do facto de haver possibilidade de ir dos cantos superiores para o percurso do meio sem ter de contornar as máquinas, o que no outro não era permitido, pois era uma aproximação pessimista.

8.2 Validação do Algoritmo A^*

O algoritmo A^* executa a pesquisa dos nós gerados pela construção do mapa. Este método, por forma a permitir a execução em tempo real e o replaneamento da trajetória, necessita de ter um tempo de processamento rápido. Além disso é necessário gerar trajetórias ótimas, ou seja com o menor comprimento possível, sem comprometer a integridade do robô, sendo esta parte garantida na componente de construção do mapa.

As características do método de pesquisa para além de ser afetado pela construção do mapa, também varia consoante os parâmetros atribuídos para a heurística.

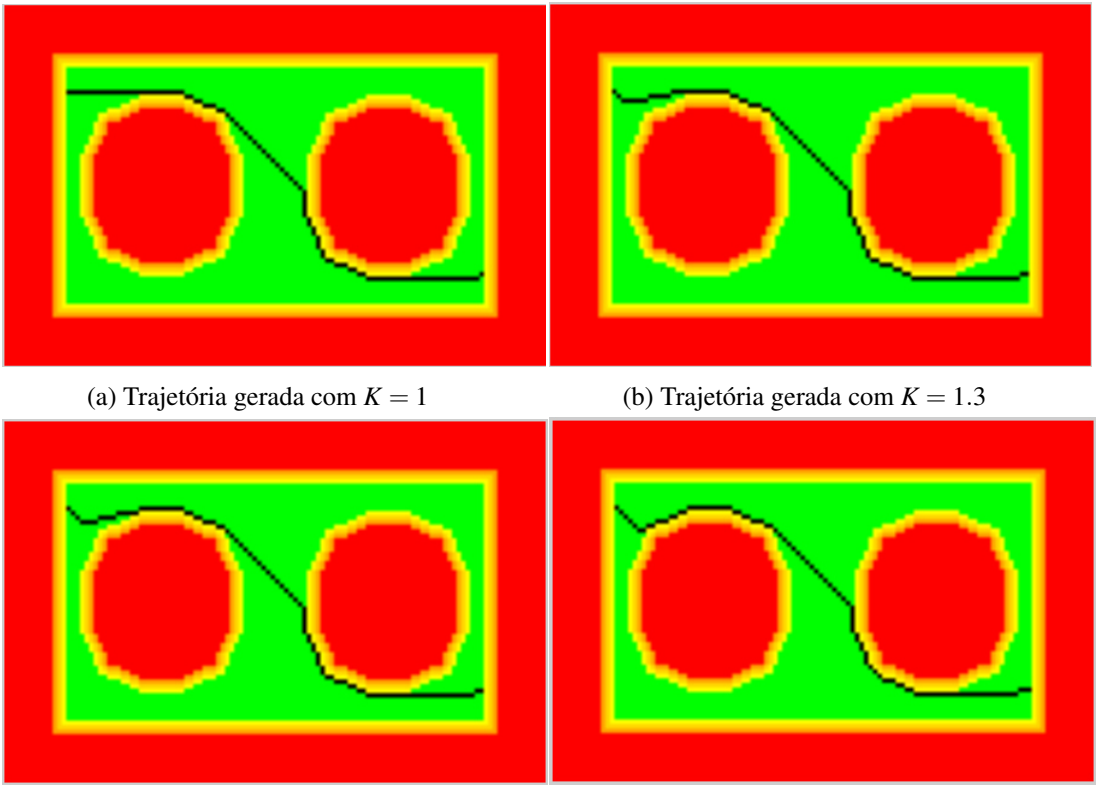
Este algoritmo foi implementado usando duas variantes, tal como já foram apresentadas nos capítulos 5 e 6, sendo de seguida discutidos os resultados obtidos para cada um dos casos.

8.2.1 Pesquisa em grafo representativo de um espaço bidimensional

Com o mapeamento sem ter considerado a orientação é gerado um grafo de um espaço bidimensional, (X,Y), com um número variável de nós consoante a resolução do mapa utilizada, o que afeta os tempos de processamento por parte do algoritmo e o comprimento da trajetória dimensionada. Para além disso, também a heurística utilizada trás repercussões sobre as características referidas.

A heurística utilizada foi a euclidiana com a introdução de um novo parâmetro, o coeficiente K, o qual permite alterar os tempos de processamento, afetando contudo a otimilidade do percurso, pois ao recorrer a um $K > 1$ a heurística deixa de ser admissível, pois sobrestima a distância, atribuindo uma maior importância aos nós mais próximos do destino.

Assim nas Figuras 8.13a, 8.13b, 8.13c e 8.13d, são apresentadas as trajetórias calculadas com $K = 1$, $K = 1.3$, $K = 1.4$ e $K = 1.9$, para se movimentar da posição $(X,Y) = (-113,50)$ para $(X,Y) = (113,-50)$, em que o robô tem raio de 25cm e as células usadas são de 2.5cm.



(a) Trajetória gerada com $K = 1$

(b) Trajetória gerada com $K = 1.3$

(c) Trajetória gerada com $K = 1.4$

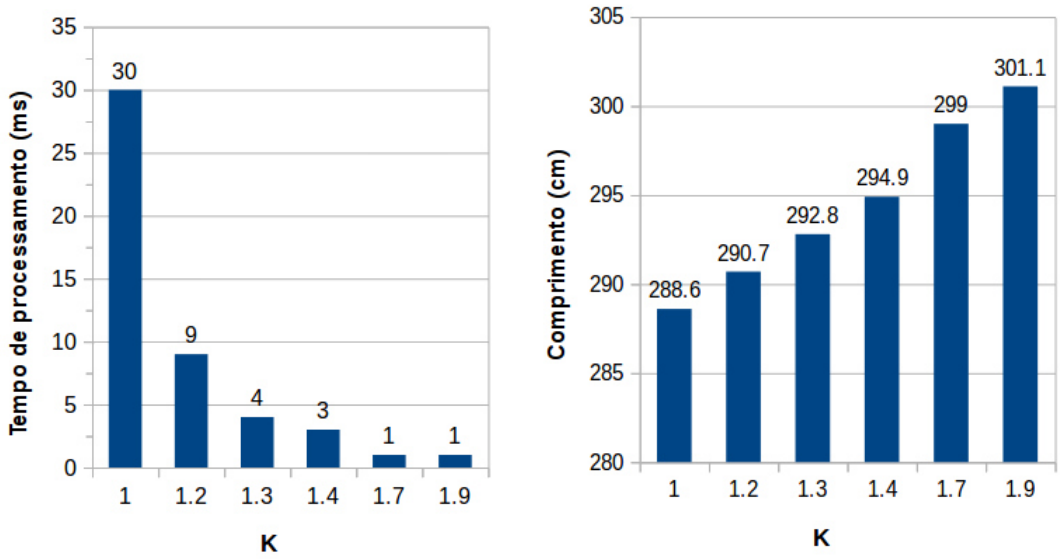
(d) Trajetória gerada com $K = 1.9$

Figura 8.13: Trajetórias planeadas usando diferentes valores de K para a heurística

Como se pode verificar pelas Figuras o aumento do coeficiente K leva a que a trajetória deixe de ser ótima percorrendo distâncias maiores, apresentando uma curvatura mais acentuada na aproximação do primeiro obstáculo, pois esses nós são vistos como mais próximos do destino, levando ao desvio do trajeto ideal que seria uma reta, suavizando a trajetória. Na tabela 8.7 encontram-se representados os efeitos que diferentes valores de K têm sobre o tempo de processamento, os deslocamentos e no número de nós processados, nas mesmas condições das trajetórias demonstradas no conjunto de Figuras 8.13.

Tabela 8.7: Efeitos causados pela variação do valor K no algoritmo A^*

	Número de Nós Processados	**Comprimento da Trajetória** (cm)	**Tempo de Processamento** (ms)
$K = 1$	12160	288.6	30
$K = 1.2$	6152	290.7	9
$K = 1.3$	3464	292.8	4
$K = 1.4$	2264	294.9	3
$K = 1.7$	1160	299.0	1
$K = 1.9$	984	301.1	1



(a) Tempos de processamento obtidos com diferentes valores de K

(b) Comprimentos obtidos com diferentes valores de K

Figura 8.14: Efeitos causados pela variação do valor K no tempo de processamento e comprimento da trajetória

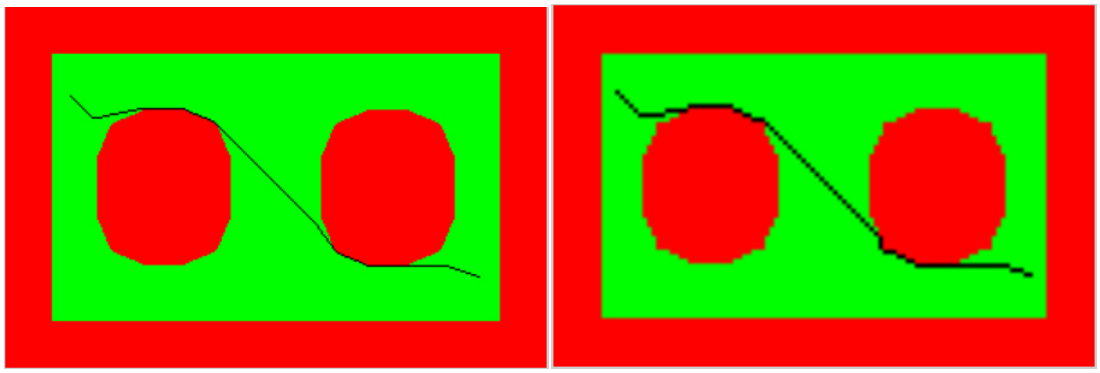
Tal como se pode verificar quanto maior for o valor de K, menor será o número de nós processados, pois, como já foi referido, a heurística passa a sobrestimar a distância, sendo que quanto menor for a distância para o destino, menor será o efeito desse coeficiente, logo esses nós são escolhidos. Com isto o tempo de processamento decresce de forma abrupta, como se vê na Figura 8.14a, passando de 30ms com $K = 1$, ou seja com estimativa admissível, para 9ms com $K = 1.2$.

Contudo esta variação tem repercussões ao nível do comprimento da trajetória a realizar, como se vê na Figura 8.14b, o qual incrementa o erro relativamente ao resultado ótimo com o aumento do valor do coeficiente. Assim, passa a realizar percursos com desvios como se viu no conjunto de Figuras 8.13.

Desta forma na escolha do K a usar é necessário escolher o *tradeoff* entre otimização e tempo de processamento que se adequa melhor à problemática. Por exemplo, neste projeto, uma vez que se necessita que a resposta seja rápida, podendo sofrer uma ligeira discrepância para o caminho mais curto, foi escolhido um valor de $K = 1.3$, sendo que a variação de comprimento relativamente ao ótimo é de 4.2cm, que se trata de um valor reduzido quando comparado com o tamanho da trajetória.

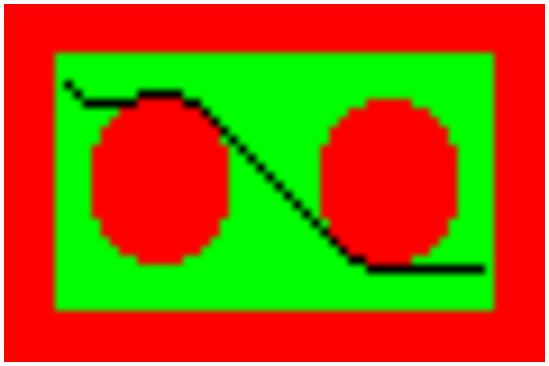
O tamanho das células tem um efeito no tempo de processamento do algoritmo de pesquisa, reduzindo quanto maior for o tamanho de célula usado. Contudo o caminho passa a ser menos suave fazendo movimentos mais bruscos quando as células crescem, aumentando a probabilidade de existir colisões no recurso ao sistema de navegação devido à sobreatuação dos motores no sistema real e à degradação da representação. No conjunto de Figuras 8.15 pode-se observar os percursos obtidos consoante a variação da resolução, onde as trajetórias foram calculadas sem

camadas de proteção, pois como já foi referido seria necessário adaptar consoante o tamanho das células, e utilizou-se um $K = 1.3$, planeando o percurso de $(X,Y) = (-113,50)$ para $(X,Y) = (113,-50)$.



(a) Trajetória gerada com tamanho de célula 1cm

(b) Trajetória gerada com tamanho de célula 2.5cm



(c) Trajetória gerada com tamanho de célula 5cm

Figura 8.15: Trajetórias planeadas usando diferentes resoluções do mapa

Os caminhos, como se podem ver, são semelhantes, mas é necessário considerar que o deslocamento entre células é maior consoante o seu tamanho, na tabela 8.8 apresenta-se o tempo de processamento com a alteração do tamanho das células, em que como se vê quanto maior a dimensão menor o tempo de processamento.

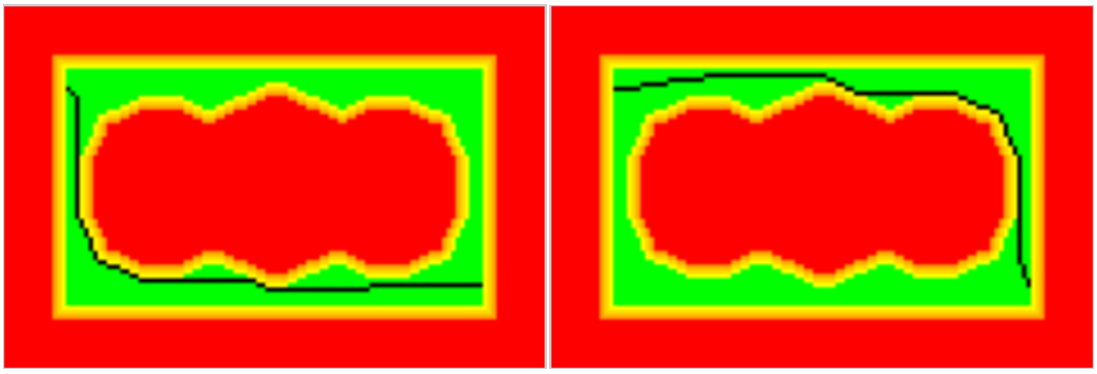
Tabela 8.8: Efeitos causados pela variação do tamanho da célula no algoritmo A^*

Tamanho da Célula (cm)	**Tempo de Processamento** (ms)
1	4
2.5	2
5	1

8.2.1.1 Algoritmo A^* em espaço bidimensional com múltiplos robôs

Este algoritmo com múltiplos robôs sofre o mesmo efeito quando se varia o tamanho de células e o valor de K como quando se aplica para apenas um robô, contudo um estudo a fazer é o efeito do valor de K na trajetória o qual provoca variações no caminho a seguir consoante a heurística seja admissível ou não.

Tal como se pode observar no conjunto de Figuras 8.16 o robô que executa o algoritmo A^* responde de formas diferentes à presença de outro veículo na posição central do mapa consoante o valor atribuído ao ganho da estimativa, considerando ambos os robôs iguais (R=25cm), com células de 2.5cm e planeando o percurso de $(X,Y) = (-113,53)$ para $(X,Y) = (113,-53)$.



(a) Trajetória gerada com $K = 1$ (b) Trajetória gerada com $K = 1.3$

Figura 8.16: Trajetórias planeadas com múltiplos robôs usando diferentes valores de K para a heurística

Como se pode verificar com a estimativa admissível o percurso escolhido foi pelo caminho de baixo, uma vez que tem o comprimento menor e torna o caminho mais suave, já com o $K = 1.3$ o percurso é o superior. Contudo, como se pode retirar da tabela 8.9, quanto maior for o coeficiente maior será o comprimento e reduz o tempo de processamento. Contudo verifica-se que os tempos são superiores ao com o robô isolado, pois expande mais nós devido à ocupação do percurso ideal.

Tabela 8.9: Efeitos causados pela variação do valor K no algoritmo A^* com múltiplos robôs

	Número de Nós Processados	**Comprimento da Trajetória** (cm)	**Tempo de Processamento** (ms)
$K = 1$	12736	322.5	33
$K = 1.3$	7680	326.6	10

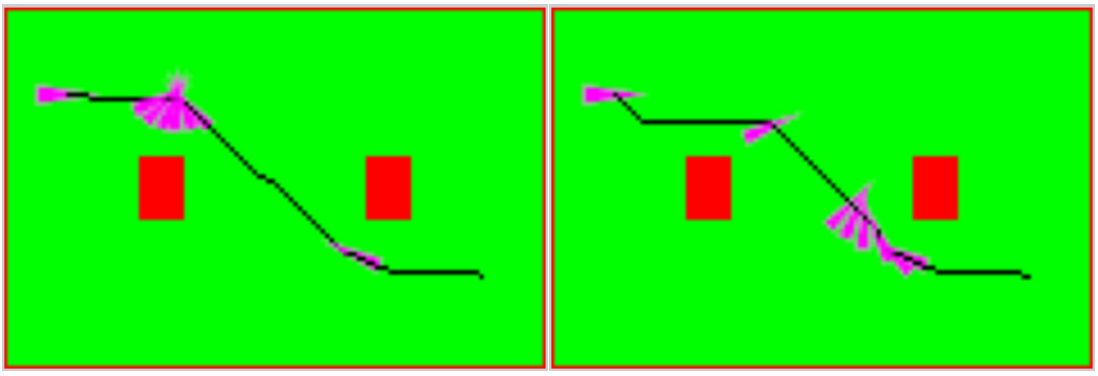
8.2.2 Pesquisa em grafo representativo de um espaço tridimensional

Com o mapeamento considerando a orientação é gerado um grafo de um mapa tridimensional, (X,Y,θ), que tal como no método para o grafo do espaço 2D, depende da resolução das células

usada e para além disso do número de orientações a usar, que aumentam o número de mapas e, consequentemente, a quantidade de nós a considerar pelo A^*.

Para efeitos de testes deste algoritmo o robô a considerar terá dimensões de 40 x 30 cm, e serão usadas células de 2.5cm.

Assim neste método de pesquisa implementado os comportamentos derivados das variações do coeficiente K tem os mesmos efeitos que no algoritmo anteriormente apresentado, isto é quanto maior for o seu valor menor será o tempo de processamento e maior será o comprimento, sofrendo ainda uma penalização devido às variações do ângulo a considerar, tal como se vê no conjunto de Figuras 8.17, onde os triângulos a rosa indicam a orientação do robô.



(a) Trajetória gerada com $K = 1$ (b) Trajetória gerada com $K = 1.3$

Figura 8.17: Trajetórias planeadas para pesquisa em grafos de espaço 3D com diferentes valores de K

Como se pode observar nas Figuras anteriores a grande diferença nas trajetórias encontra-se no local onde se altera a orientação, o que leva, também, a diferentes valores no comprimento do percurso a realizar. Nas tabelas 8.10a e 8.10b encontram-se os pontos onde existe mudança de orientação, representados pela posição central da célula, consoante o K utilizado, sendo que com a estimativa sobrestimada as variações de atitude são mais graduais tornando o movimento mais suave.

Tabela 8.10: Nós de variação das orientações das trajetórias do conjunto de Figuras 8.17

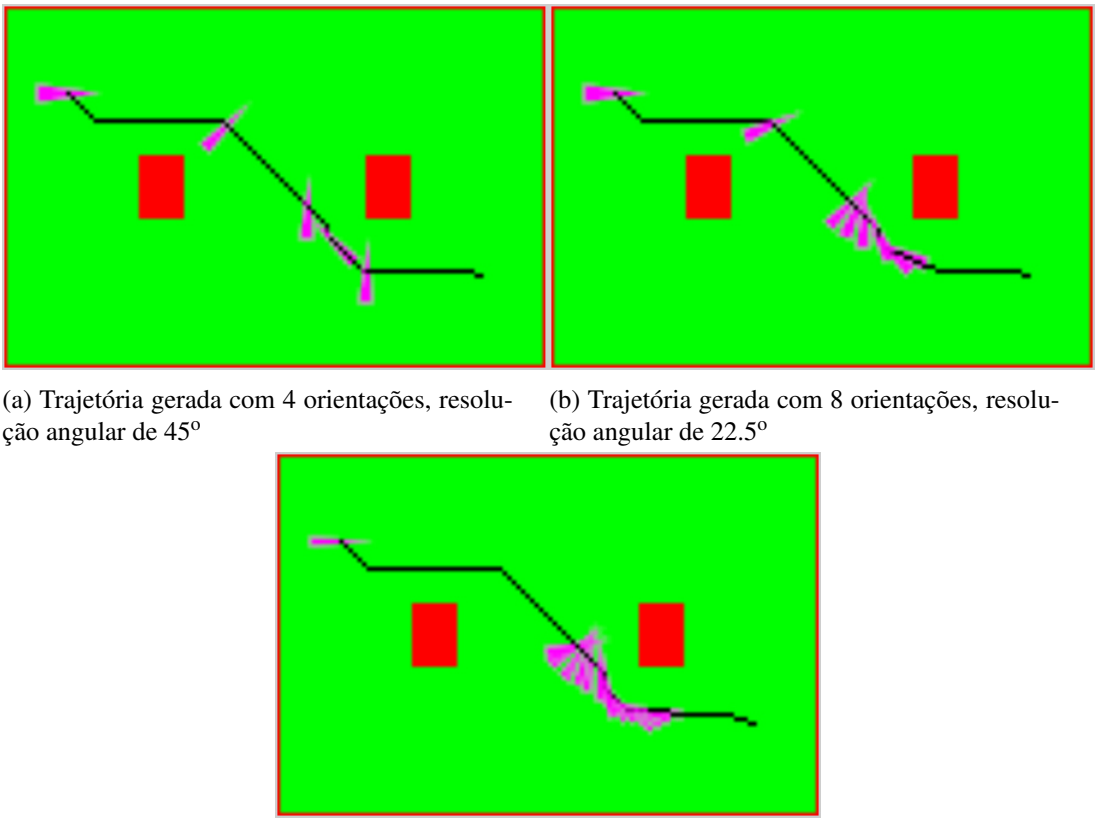
(a) Trajetória da Figura 8.17a com $K = 1$

Nó	**X** (cm)	**Y** (cm)	θ (^{o})
0	-113.75	51.25	0
⋮	⋮	⋮	⋮
21	-61.25	48.75	22.5
22	-58.75	48.75	45
23	56.25	48.75	67.5
24	-53.75	48.75	90
25	51.25	48.75	112.5
26	-48.75	-46.25	135
⋮	⋮	⋮	⋮
62	41.25	-36.25	157.5

(b) Trajetória da Figura 8.17b com $K = 1.3$

Nó	**X** (cm)	**Y** (cm)	θ (^{o})
0	-113.75	51.25	0
⋮	⋮	⋮	⋮
35	-26.25	33.75	22.5
⋮	⋮	⋮	⋮
52	16.25	-8.75	45
53	18.75	-11.25	67.5
⋮	⋮	⋮	⋮
55	23.75	-16.25	90
⋮	⋮	⋮	⋮
58	31.25	-23.75	112.5
⋮	⋮	⋮	⋮
62	36.25	-33.75	135
⋮	⋮	⋮	⋮
64	41.25	-36.25	157.5

Outro fator que provoca variações no tempo de processamento é a resolução angular, isto é o número de orientações a usar para construir o mapa. Quanto maior for a resolução, maior vai ser o número de nós para processar e consequentemente maior o tempo de processamento. No conjunto de Figuras 8.18 demonstra-se os resultados obtidos considerando diferentes resoluções angulares.



(a) Trajetória gerada com 4 orientações, resolução angular de 45º

(b) Trajetória gerada com 8 orientações, resolução angular de 22.5º

(c) Trajetória gerada com 18 orientações, resolução angular de 10º

Figura 8.18: Trajetórias planeadas para pesquisa em grafos de espaço 3D com diferentes resoluções angulares

Analisando as Figuras 8.18 pode se observar que quanto maior for o número de orientações a usar, mais liberdade de movimento rotacional o robô dispõem. Por exemplo para reduzir a trajetória entre os nós inicial e de destino o robô efetua quatro rotações, pois ao adaptar a rotação para os 45º a primeira vez pode começar a realizar a diagonal mais cedo, corroborado com a expansão em 8.7, de seguida coloca-se a 90º para se aproximar do obstáculo o máximo possível, rodando para 135º de maneira a realizar uma nova diagonal, por forma a diminuir o comprimento do caminho, por fim recoloca-se a 90º para poder seguir uma reta até à posição final.

Para as restantes o comportamento segue a mesma ideia de funcionamento, sendo na tabela 8.11 apresentados os tempos de processamento e comprimentos da trajetória calculada.

Tabela 8.11: Efeitos causados pela resolução angular usada no algoritmo A^*

Número de Orientações	**Comprimento da Trajetória** (cm)	**Tempo de Processamento** (ms)
4	274.9	3
8	274.6	5
18	276.1	32

Como se pode verificar os tempos de processamento são tanto maiores quanto o número de orientações a usar, sendo o comprimento também dependente dos ângulos obtidos, nomeadamente com 4 e 8 ângulos pode-se atingir os 45º, logo faz-se uma diagonal mais próxima dos obstáculos relativamente ao de 18 ângulos, que roda de 10º em 10º. Contudo comparando com uma situação equiparável com o método no espaço bidimensional, considerando apenas uma camada de proteção, os comprimentos são sempre menores, sendo obtido um comprimento de 281.9cm com o outro algoritmo.

8.2.2.1 Algoritmo A^* em espaço tridimensional com múltiplos robôs

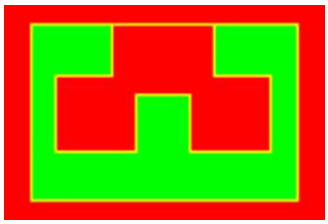
A grande vantagem deste algoritmo recorrendo a espaços que consideram as orientações ocorre na existência de múltiplos robôs no ambiente de trabalho, já que aumenta a manobrabilidade do robô o que permite o cruzamento dos robôs em múltiplas ocasiões.

O comportamento com a presença de múltiplos robôs perante os parâmetros da estimativa, K, e do tamanho de células é de todo semelhante à implementação com um único robô, havendo também ocasiões em que caso $K > 1$ envia o robô por percursos que se fosse usada a heurística admissível o robô não iria, não sendo, contudo, muito elevada a diferença de comprimento para o caso de se recorrer a $K = 1.3$.

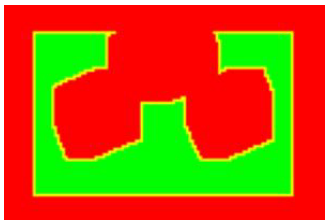
Para demonstrar a vantagem inerente a este método apresentam-se duas situações em que o segundo robô se encontra em $(X,Y) = (0,50)$ com duas orientações distintas, 0º e 45º, realizando uma trajetória de $(X,Y) = (-113,50)$ para $(X,Y) = (113,-50)$.

Como se pode ver na Figura 8.6b sem considerar as orientações dos robôs a passagem do robô principal pelo percurso do meio, que habitualmente é o trajeto ótimo, é impossível, pois considera o pior dos casos para ambos os robôs.

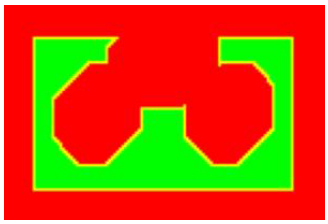
Esse caso é o do robô, que constitui o obstáculo, se encontrar a 0º bloqueando todas as passagens possíveis, como se verifica no conjunto de Figuras 8.19.



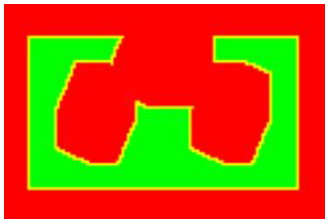
(a) Expansão do mapa com $\theta = 0^o$



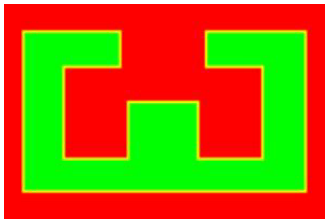
(b) Expansão do mapa com $\theta = 22.5^o$



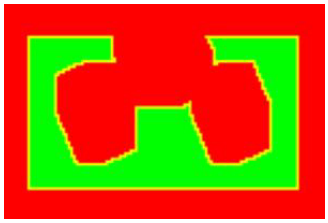
(c) Expansão do mapa com $\theta = 45^o$



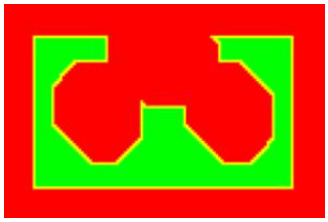
(d) Expansão do mapa com $\theta = 67.5^o$



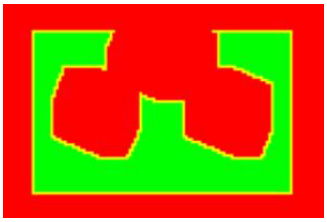
(e) Expansão do mapa com $\theta = 90^o$



(f) Expansão do mapa com $\theta = 112.5^o$



(g) Expansão do mapa com $\theta = 135^o$



(h) Expansão do mapa com $\theta = 157.5^o$

Figura 8.19: Expansão do mapa consoante variação de orientação de 22.5^o e segundo robô em $(X, Y, \theta) = (0, 50, 0^o)$

Desta forma o robô necessita de gerar uma trajetória alternativa pelo percurso lateral, obtendo a trajetória representada na Figura 8.20. Nesta pode-se observar que o robô segue uma trajetória similar à definida com o algoritmo anterior, contudo mais curta devido a poder aproximar-se mais dos obstáculos ao cortar caminho com as diagonais.

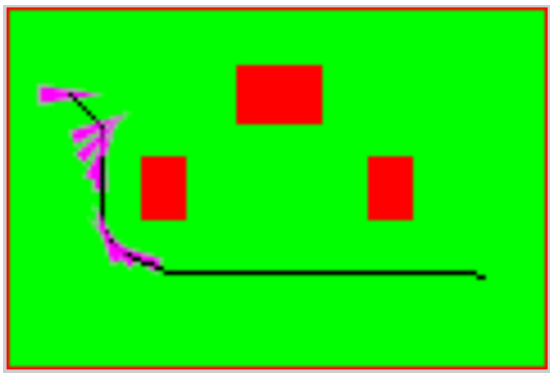
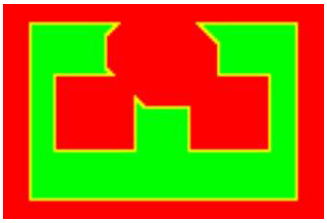
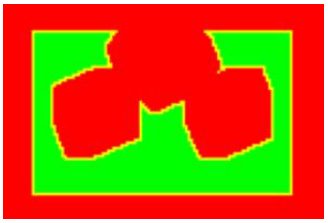


Figura 8.20: Trajetória gerada com segundo robô em $(X,Y,\theta) = (0,50,0^o)$

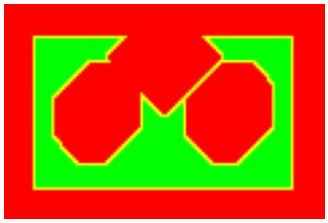
Contudo se o obstáculo se encontrar rodado a 45^o obtém-se a expansão representada no conjunto de Figuras 8.21.



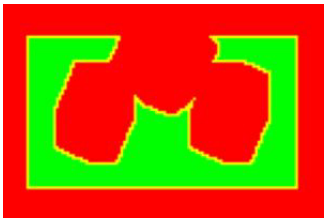
(a) Expansão do mapa com $\theta = 0^o$



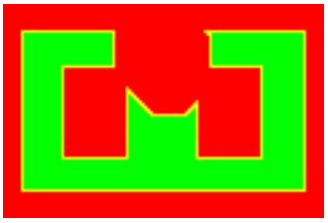
(b) Expansão do mapa com $\theta = 22.5^o$



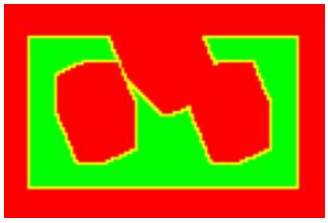
(c) Expansão do mapa com $\theta = 45^o$



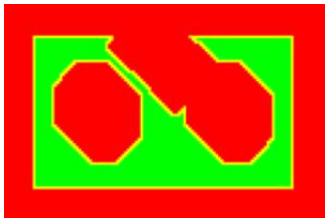
(d) Expansão do mapa com $\theta = 67.5^o$



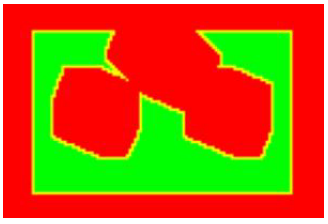
(e) Expansão do mapa com $\theta = 90^o$



(f) Expansão do mapa com $\theta = 112.5^o$



(g) Expansão do mapa com $\theta = 135^o$



(h) Expansão do mapa com $\theta = 157.5^o$

Figura 8.21: Expansão do mapa consoante variação de orientação de 22.5^o e segundo robô em $(X,Y,\theta) = (0,50,45^o)$

Como se pode observar existe a possibilidade de, com esta orientação do segundo robô, existir o cruzamento, nomeadamente com o veículo principal a 112.5^o e a 135^o, como se vê nas Figuras

8.21f e 8.21g respetivamente, sendo que com 112.5º pagaria um custo extra devido a estar demasiado próximo dos obstáculos. Assim a trajetória gerada para efetuar encontra-se na Figura 8.22, podendo-se ver os nós que existe a rotação na tabela 8.12, representados pela sua posição central.

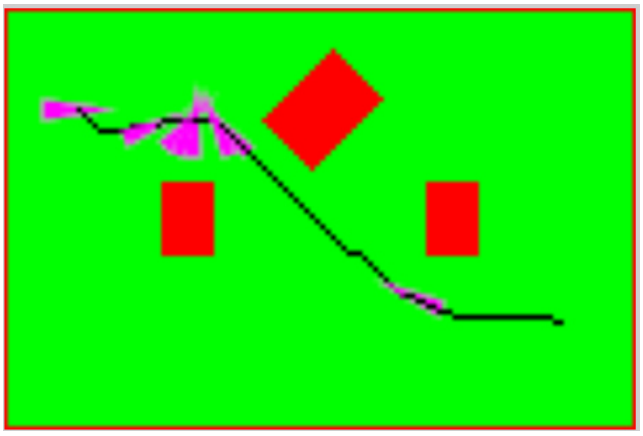


Figura 8.22: Trajetória gerada com segundo robô em $(X, Y, \theta) = (0, 50, 45^{o})$

Tabela 8.12: Nós de variação das orientações da trajetória da Figura 8.22

Nó	**X** (cm)	**Y** (cm)	θ (^{o})
0	-113.75	51.25	0
⋮	⋮	⋮	⋮
15	-76.25	43.75	22.5
⋮	⋮	⋮	⋮
22	-61.25	46.25	45
23	-61.25	46.25	67.5
24	-58.75	46.25	90
⋮	⋮	⋮	⋮
29	-48.75	46.25	112.5
30	-46.25	43.75	135
⋮	⋮	⋮	⋮
65	41.25	-36.25	157.5

Desta forma é possível verificar que existe a possibilidade de cruzamento entre robôs otimizando a trajetória de ambos, favorecendo os cruzamentos entre eles e facilitando o processo de cooperação que necessitarem de realizar.

8.3 Validação do Sistema de Navegação

O sistema de navegação desenvolvido tem diferentes modos de funcionamento que permitem a deslocação entre postos de trabalho. Esses modos de funcionamento estabelecem os movimentos

usando o seguimento de segmentos de reta apresentado no capítulo 7 e estabilizando nas posições usando o controlo de posição.

Por forma a fazer o seguimento de linha e o controlo de posição torna-se necessário definir os valores das variáveis de ganho K_1, K_2, K_3 e K_4, para a implementação no ambiente de simulação e no real é necessário alternar os valores, uma vez que o simulador permite movimentos mais rápidos e com ganhos mais elevados, pois a localização é ideal e não considera todos os erros existentes no sistema físico, sendo o valor dos ganhos apresentados na tabela 8.13.

Tabela 8.13: Ganhos utilizados nas duas plataformas

	Ganhos Utilizados			
Plataforma	K_1	K_2	K_3	K_4
SimTwo	3.8	2.5	1.5	1.5
Real	3.8	1.3	2.5	2.5

O variar dos ganhos causam diferentes comportamentos no movimento do robô, no caso do seguimento de reta o valor de K_1 quanto mais baixo for mais lenta será a aproximação à reta, contudo se for elevado causa oscilações em torno dela. O ganho K_2 aumenta a velocidade de rotação por forma a obter a orientação desejada, aumentando a oscilação em torno de θ_{fin}. Os valores de K_3 e K_4 têm o mesmo efeito que o K_2, mas relativamente a X e a Y, respetivamente.

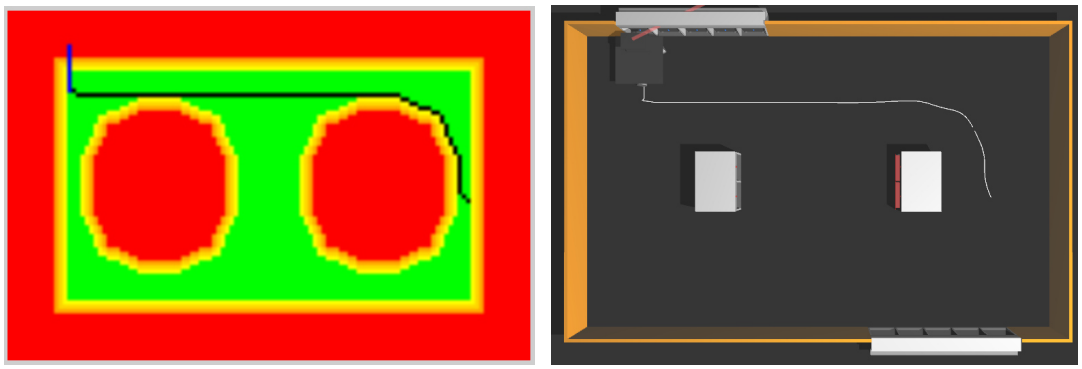
Com isto de seguida apresenta-se os resultados obtidos com o sistema de navegação com um e com dois robôs.

8.3.1 Sistema de navegação para um robô

O sistema de navegação com apenas um robô simplesmente tem em consideração o campo como era anteriormente, assim uma alternativa a usar poderia ser após efetuar os movimentos especiais de entrada ou saída de postos de trabalho calcular uma única vez o percurso e de seguida movimentar-se em torno dele.

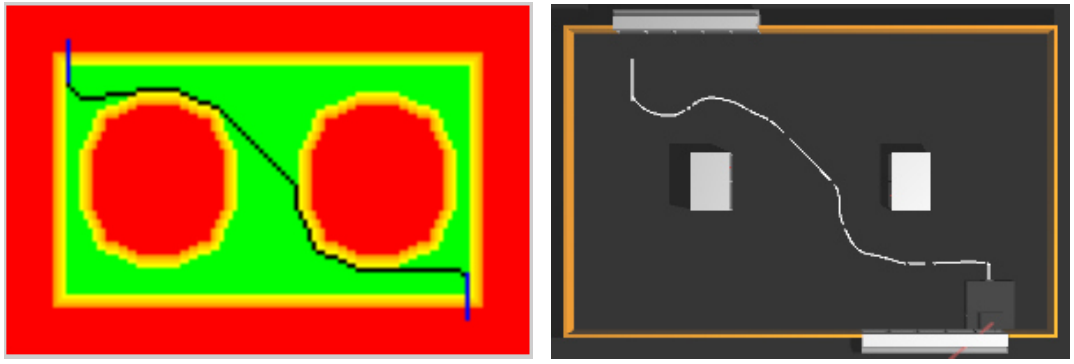
Contudo como foi referido no capítulo 7 o método de deslocamento entre máquinas e armazéns passou por replanear a trajetória, no modo de funcionamento com recurso ao A^*, sempre que fosse obtida uma medição da localização.

Nas Figuras 8.23, 8.24 e 8.25 apresentam-se três exemplos de percursos entre duas máquinas (trilhos a branco), em que se compara o deslocamento planeado pelos algoritmos desenvolvidos, em que a azul se encontram os movimentos de entrada e saída de postos de trabalho e a preto a primeira trajetória calculada pelo A^*, e o percurso realizado no *SimTwo* replaneando ao longo do caminho.



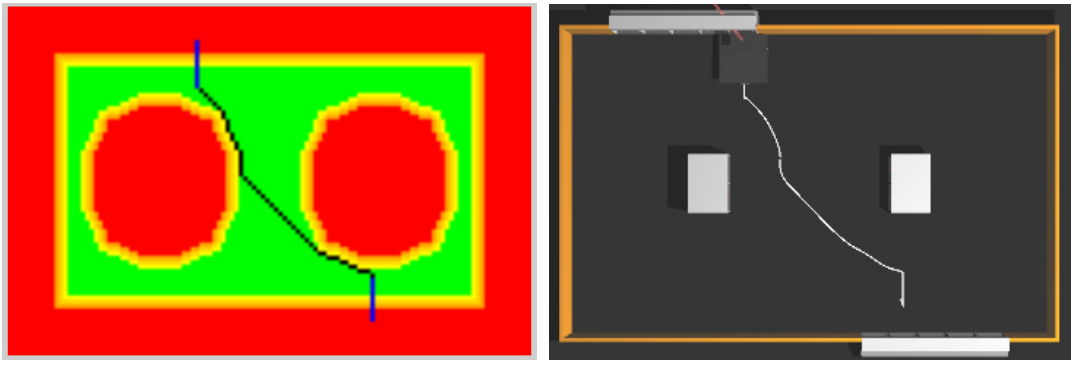
(a) Navegação calculada (b) Navegação realizada no *SimTwo*

Figura 8.23: Comparação entre as trajetórias calculadas e realizadas do ponto inicial para o armazém superior



(a) Navegação calculada (b) Navegação realizada no *SimTwo*

Figura 8.24: Comparação entre as trajetórias calculadas e realizadas do armazém superior para o inferior



(a) Navegação calculada (b) Navegação realizada no *SimTwo*

Figura 8.25: Comparação entre as trajetórias calculadas e realizadas do armazém inferior para o superior

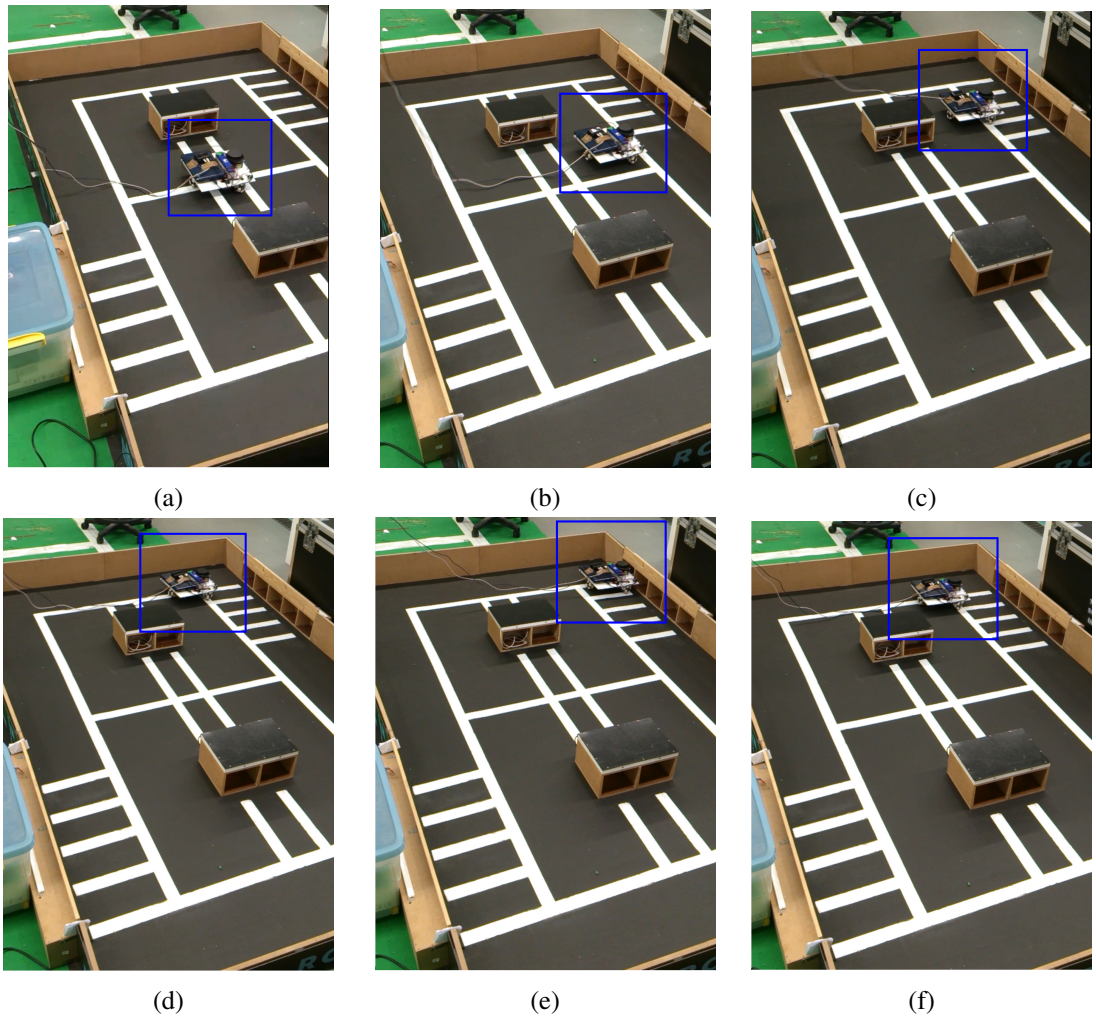
Na Figura 8.23 apresenta-se o robô na posição inicial em $(X,Y) = (113,-10)$ a deslocar-se para o armazém superior e como se pode verificar os movimentos são respeitados e aproximam-se das previamente calculadas, isto é não foram efetuados replaneamentos.

Como se pode ver na Figura 8.24 o caminho seguido está muito próximo do planeado inicialmente, contudo como efetua a curva, causada pelo valor $K = 1.3$ a que a heurística está sujeita, o deslocamento por vezes sai da posição devida, conduzindo a um reajuste no cálculo da trajetória via A^*, podendo-se ver que seguidamente a trajetória se assemelha bastante ao planeado, neste caso o replaneamento foi visível corrigindo o caminho do robô.

Na Figura 8.25 pode-se observar que o movimento realizado do armazém de baixo para o de cima é realizado de acordo com o expectável, não sofrendo reajustes visíveis.

Com este sistema foi testada a realização da primeira manga da competição *Robot@Factory*, recorrendo ao *SimTwo*, na qual se transita as peças de um armazém para o outro e transpareceu resultados viáveis no deslocamento entre armazéns, sem contudo ter sido testado com garras nesta plataforma.

Com o sistema real foram efetuados testes para a realização da primeira manga encontrando-se nos conjuntos de Figuras 8.26 e 8.27 o percurso controlado através do sistema de navegação desenvolvido.

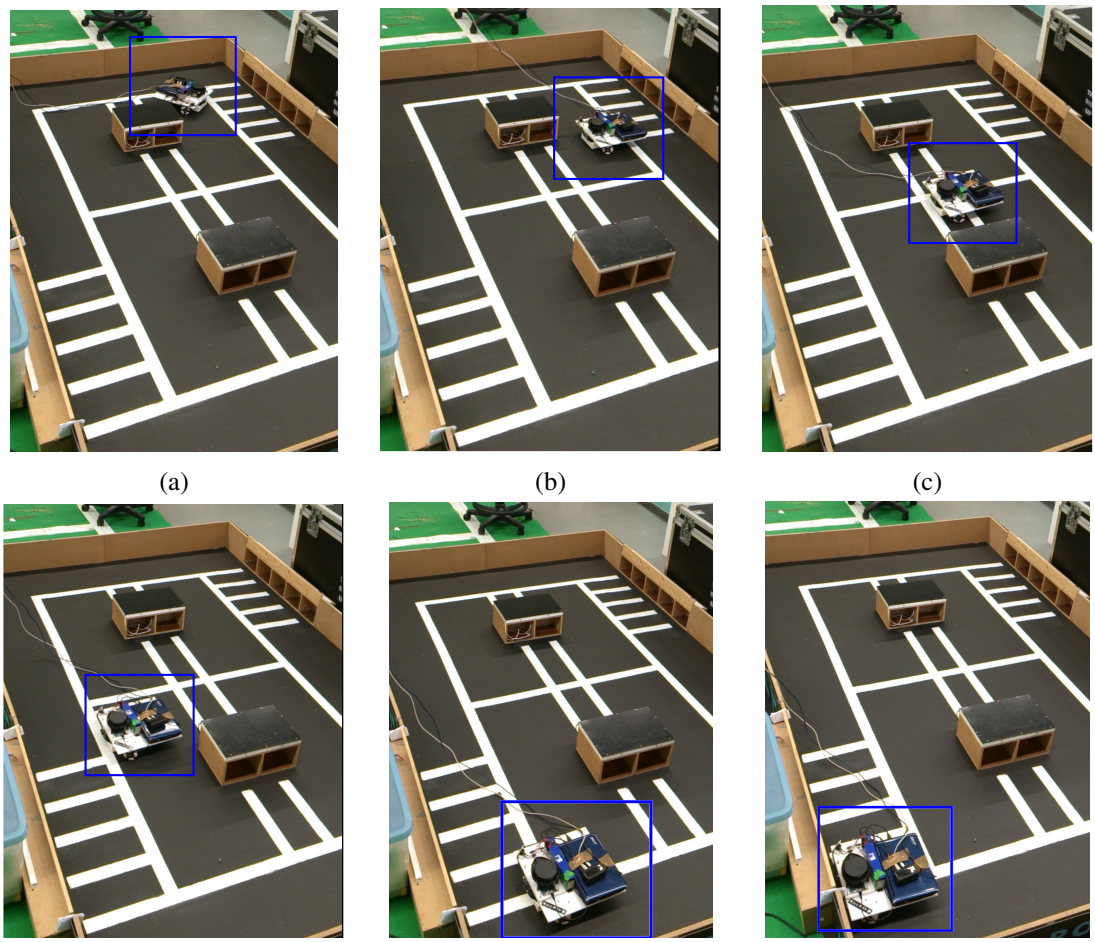


(a) (b) (c)

(d) (e) (f)

Figura 8.26: Primeira parte da trajetória desde a posição central para o armazém da direita

No conjunto de Figuras 8.26 vê-se a trajetória da posição central até ao armazém da direita na qual o robô vai até ao ponto de entrada na máquina, onde se posiciona e segue a reta até entrar no armazém e de seguida sai do armazém para se dirigir até ao armazém da esquerda, Figura 8.27. Ao deslocar-se o robô, pode-se observar as limitações mecânicas impostas, nomeadamente na altura de realizar o controlo de posição, por exemplo para rodar ângulos perto de 1^{o}, a velocidade requisitada é demasiado pequena, logo os motores não atuam devidamente ou então rodam em demasia, outra das causas é o facto de a travagem não ser imediata o que conduz a um erro na translação que necessita de acerto.



(a) (b) (c)

(d) (e) (f)

Figura 8.27: Segunda parte da trajetória desde o armazém da direita para o armazém da esquerda

8.3.2 Sistema de navegação para dois robôs

O sistema de navegação para dois robôs está otimizado para o mapeamento sem considerar orientação, tendo sido testado apenas no simulador, já que não se encontrava desenvolvido um segundo protótipo para aplicar testes reais.

Assim, começando por usar um dos robôs estático, pode-se observar nas Figuras 8.28 e 8.29 dois casos distintos a considerar.

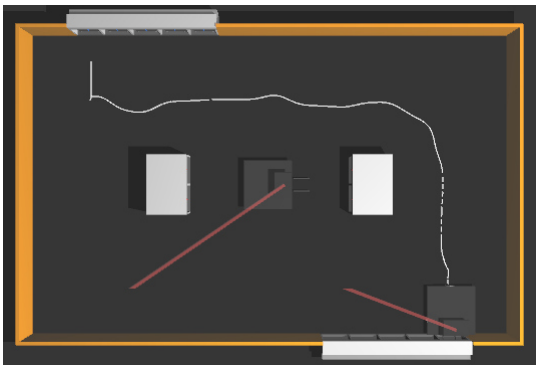
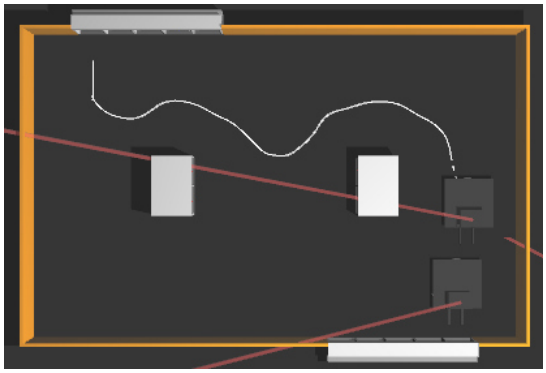
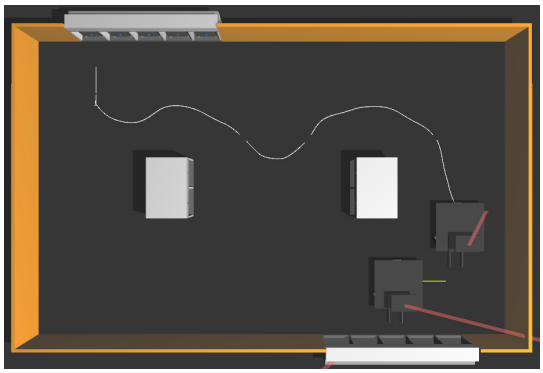


Figura 8.28: Navegação com segundo robô estático em $(X,Y) = (0,0)$

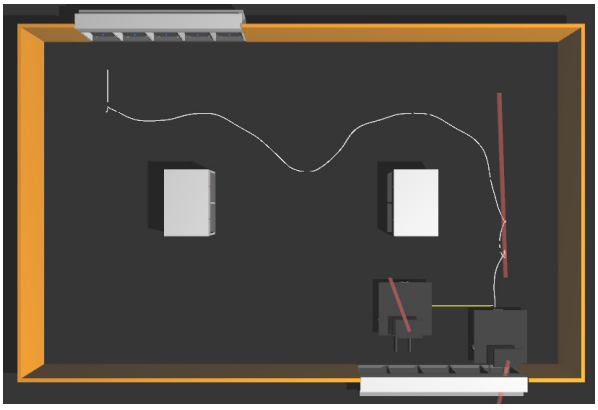
Na Figura 8.28 o robô auxiliar encontra-se estático na posição central do mapa e como se pode ver, com o trilho branco no chão, o agente principal gera uma trajetória alternativa para movimentar-se entre armazéns, que idealmente era pelo caminho central. Tal como se pode observar numa zona perto do segundo robô é realizada uma pequena curvatura para garantir a distância de segurança ao obstáculo, chegando contudo à posição final sem entraves.



(a) Segundo robô posicionado na posição final



(b) Segundo robô posicionado perto da posição final



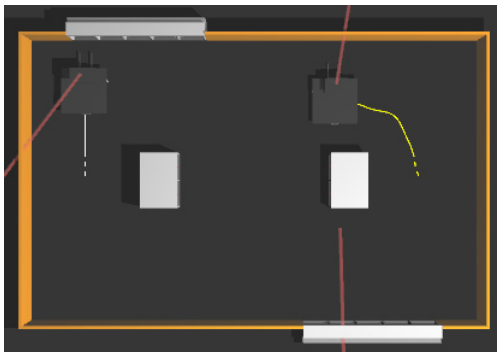
(c) Segundo robô posicionado longe da posição final

Figura 8.29: Efeito da proximidade do segundo robô na posição final do planeamento de trajetória dinâmico

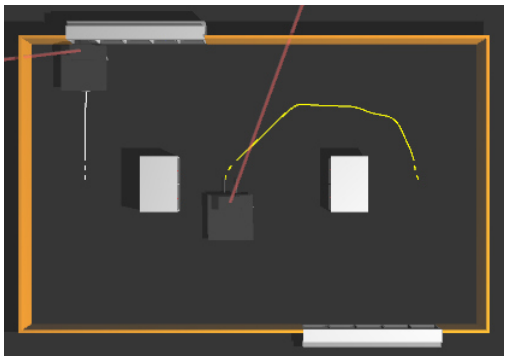
No conjunto de Figuras 8.29 encontram-se ilustrados diferentes estados do sistema de navegação quando se encontra um obstáculo no ponto final do modo de funcionamento com planeamento de trajetórias dinâmico.

Como se pode ver na Figura 8.29a o robô inicialmente não conhecia o posicionamento do segundo robô o que conduziu a uma entrada no percurso central, contudo ao aproximar-se o suficiente para o detetar como obstáculo, foi efetuado o replaneamento e, como a posição final era momentaneamente inválida, foi projetado um novo ponto, que servirá como local de espera até ser desimpedido o nó final.

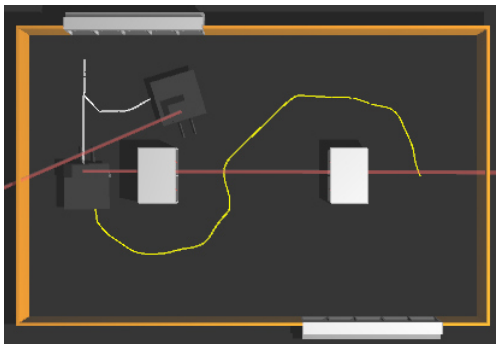
Na Figura 8.29b como se pode ver ao deslocar o robô auxiliar para o lado numa distância onde, apesar de não estar a bloquear a posição final, esteja a ser visto como ponto de possível colisão, o planeador recalcula um ponto projetado e desloca-se para lá, entrando na máquina quando não tiver esse nó como bloqueado, Figura 8.29c.



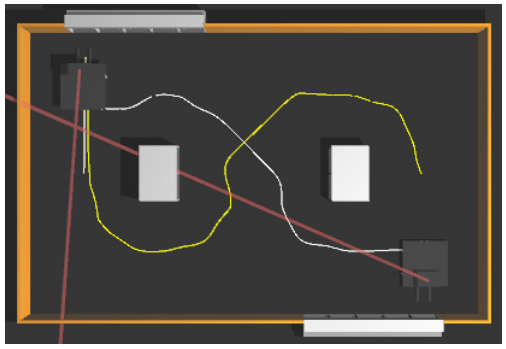
(a) Arranque de ambos os robôs da posição inicial



(b) Deslocamento do segundo robô para o ponto alternativo



(c) Partida do primeiro robô para o outro armazém



(d) Entrada do primeiro robô no armazém de baixo e saída do segundo robô

Figura 8.30: Sistema de navegação com ambos os robôs em funcionamento coordenado

No conjunto de Figuras 8.30 está representado um exemplo do movimento de dois robôs em simultâneo com ordens concorrentes, isto é ambos vão para o mesmo armazém, sendo o trilho branco do primeiro robô e o amarelo do segundo. Na Figura 8.30a os robôs partem do ponto inicial para se deslocarem para o armazém superior. Na Figura 8.30b como o primeiro robô atingiu primeiro a máquina o segundo altera a trajetória para o ponto de espera, local livre mais próximo

da posição final. Na Figura 8.30c a máquina encontra-se livre podendo o robô secundário dirigir-se para lá e o principal dirigir-se para o armazém inferior seguindo o percurso ótimo, Figura 8.30d.

O algoritmo desenvolvido apesar de ser seguro, não existem colisões nos testes efetuados, conduz à perda da otimilidade das trajetórias, já que por diversas situações os robôs poderiam-se cruzar e, contudo, não o fazem, caso que poderia ser resolvido adaptando o sistema de navegação para recorrer ao planeamento de trajetórias num espaço tridimensional. Para além disso, ainda existem situações em que o robô quando está a chegar ao ponto desejado pode ter de recuar devido ao aparecimento do outro e fazer um caminho maior que contorna a máquina.

Apesar destas limitações, com o devido escalonamento de tarefas, onde não se recorre a dois robôs para efetuar exatamente a mesma tarefa ao mesmo tempo, este método de planeamento simultâneo apresenta bons resultados, pois é um método seguro e fiável.

Capítulo 9

Conclusões e Trabalho Futuro

Nesta dissertação foi abordada a problemática do planeamento de trajetórias num ambiente industrial, recorrendo à competição *Robot@Factory* para simular o âmbito de teste tomando ainda considerações que permitissem o recurso a múltiplos robôs. Com isto foi permitido definir algoritmos dinâmicos com necessidade de replaneamento que funcionem em tempo real.

Por forma a apresentar as metodologias utilizadas foram expostas nos capítulos iniciais as descrições dos algoritmos implementados e os processos em que se baseiam. Sendo no fim demonstrados os resultados obtidos para cada um dos métodos desenvolvidos.

Ao nível dos objetivos inicialmente propostos pode-se afirmar que foram todos alcançados de forma satisfatória, sendo inclusive implementados métodos extra para a abordagem do problema.

Tal como referido no capítulo 1 o objetivo principal desta dissertação era implementar um algoritmo que permitisse planear trajetórias em tempo real num ambiente conhecido e dinâmico, evitando colisões e obter caminhos ótimos ou perto disso. Foi ainda proposto um objetivo secundário que retratava o planeamento de trajetórias em simultâneo para múltiplos robôs no ambiente.

Para realizar esta parte foram implementados diversos processos e metodologias. Primeiramente garantiu-se a segurança da trajetória a efetuar, através da representação do mapa realizada, recorrendo inicialmente a uma expansão sem considerar a orientação do robô, em que ao considerar o robô como um círculo de raio igual ou superior à pior situação de rotação, garante que não existe colisões. Para além disto foi implementado um sistema que altera o mapa quando surge um novo obstáculo, nomeadamente um segundo robô, para poder ser considerado no algoritmo de pesquisa.

De seguida foi desenvolvido um algoritmo A^* modificado, no qual através dos nós representativos do mapa é gerada a trajetória, sendo necessário ter em atenção a heurística a usar, nomeadamente devido ao valor atribuído ao K, pois conduzem à perda da otimilidade. Contudo através dos testes realizados foi possível ver que o valor de $K = 1.3$ tem um ligeiro custo no comprimento da trajetória a realizar, mas compensa na redução do tempo de processamento, permitindo realizar o método em menos de 10ms, logo é passível de utilização em tempo real.

Ainda foi desenvolvido um sistema de navegação baseado no seguimento de segmentos de reta e controlo de posição, organizado em máquinas de estado hierárquicas, atuando os motores para

seguir as trajetórias definidas e, ainda, recalcular os percursos, com alterações momentâneas nos pontos finais, para acomodar o uso de múltiplos robôs.

Para além destes métodos foi desenvolvido um objetivo extra no qual se procura aumentar a manobrabilidade dos robôs através do mapeamento considerando as orientações do veículo e a respetiva pesquisa A^* do grafo do espaço 3D gerado, tendo sido obtidos resultados que demonstram a viabilidade do recurso deste método em tempo real.

Por fim é de referir que ao longo deste projeto para além dos algoritmos para atacar a problemática, foi desenvolvido e construido o robô prototipado para realizar testes num ambiente real, por forma a comparar com o ambiente simulado.

9.1 Trabalho Futuro

Com vista a dar continuidade ao trabalho desenvolvido ao longo da dissertação e testar os algoritmos em meios alternativos são sugeridos como trabalhos futuros os seguintes pontos:

- Desenvolver o segundo robô e testar o sistema de navegação com múltiplos robôs
- Implementar um sistema de navegação para otimizar o controlo das trajetórias geradas através do espaço considerando o valor de θ
- Introduzir os métodos num ambiente com o mapa desconhecido, reconstruindo o mapa com dados provenientes de sensores, para testar a resposta dos algoritmos de expansão do mapa e de pesquisa do A^*
- Adicionar parâmetros à construção do mapa por forma a suavizar a trajetória, considerando por exemplo a velocidade dos movimentos dos obstáculos, e ainda alterar o sistema de navegação para representação da trajetória através de *splines*

Anexo A

Ficheiro XML do SimTwo

A.1 Ficheiro de descrição do robô omnidirecional

```
<?xml version="1.0" ?>

<robot>
  <defines>

  <!-- Robot Dimensions -->
    <const name='RobotWidth' value='0.30'/>
    <const name='RobotLength' value='0.30'/>
    <const name='RobotThickness' value='0.002'/>
    <const name='RobotHeight' value='0.11'/>
    <const name='RobotMass' value='3'/>


    <const name='WheelToCenter' value='RobotWidth/2'/>
    <const name='MotorPosX' value='0.05'/>
    <const name='CenterMotorToFront' value='RobotLength/2 - MotorPosX'/>
    <const name='CasterToBack' value='0.07'/>

    <const name='ForkWidth' value='0.06'/>
    <const name='ForkLength' value='0.1'/>
    <const name='ForkThickness' value='0.002'/>
    <const name='ForkMass' value='0.1'/>

    <const name='ForkMin' value='0.005'/>
    <const name='ForkMax' value='0.05'/>
```

```
<!-- Motor Contants -->
  <const name='MotorDiameter' value='0.028'/>
  <const name='MotorLength' value='0.075'/>
  <const name='MotorMass' value='0.027'/>

<!-- Wheel Contants -->
  <const name='WheelDiameter' value='0.098'/>
  <const name='WheelThickness' value='0.028'/>
  <const name='WheelMass' value='0.5'/>

<!--Caster Contants -->
  <const name='CasterWheelDiameter' value='0.05'/>
  <const name='CasterWheelThickness' value='0.028'/>
  <const name='CasterToWheel' value='0.05'/>
  <const name='CasterWheelMass' value='0.2'/>
  <const name='CasterMass' value='0.2'/>


<!--Calculated Measures  -->
  <const name='BracketHeight' value='RobotHeight-RobotThickness
   -(WheelDiameter/2+MotorDiameter/2)'/>
  <const name='CasterPosY' value='-RobotLength/2
   -(RobotLength/2-CenterMotorToFront)+CasterToBack'/>
  <const name='CasterToBase' value='RobotHeight
   -RobotThickness-CasterWheelDiameter/2'/>

</defines>


<solids>

  <cuboid>
    <ID value='BasePlate'/>
    <mass value='RobotMass'/>
    <size x='RobotLength' y='RobotWidth' z='RobotThickness'/>
    <pos x='-0.01' y='0' z='0.07'/>
    <rot_deg x='0' y='0' z='0'/>
    <color_rgb r='20' g='20' b='20'/>
  </cuboid>
  <cuboid>
```

```
      <ID value='BaseLaser'/>
      <mass value='RobotMass/3+0.2'/>
      <size x='0.1' y='0.15' z='0.01'/>
      <pos x='0.09' y='0' z='0.09'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='20' g='20' b='20'/>
    </cuboid>

    <cuboid>
      <ID value='ForkE'/>
      <mass value='ForkMass'/>
      <size x='ForkLength' y='ForkWidth/10' z='ForkThickness*2'/>
      <pos x='RobotLength/2 + ForkLength/2' y='RobotWidth/6' z='0.025'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='40' g='40' b='40'/>
    </cuboid>
    <cuboid>
      <ID value='ForkD'/>
      <mass value='ForkMass'/>
      <size x='ForkLength' y='ForkWidth/10' z='ForkThickness*2'/>
      <pos x='RobotLength/2 + ForkLength/2' y='-RobotWidth/6'
       z='0.025'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='40' g='40' b='40'/>
    </cuboid>
  </solids>
  <shells>
    <cuboid>
      <ID value='BaseLaserShell'/>
      <size x='0.1' y='0.15' z='0.01'/>
      <pos x='0.09' y='0' z='0.08'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='20' g='20' b='20'/>
    </cuboid>


  </shells>

  <wheels>
                <default>
```

```
                <omni/>
                <pos x='0' y='0.1' z='0'/>
                <tyre mass="0.13" radius="0.03" width="0.033"
                 centerdist="0.144"/>
                <axis angle="0"/>
                <motor ri="2.853" ki="0.0289" vmax="12" imax="2"
                 active="1"/>
                <gear ratio="51/10"/>
                <friction bv="0" fc="0.00283" coulomblimit="1e-2"/>
                <encoder ppr="216" mean="0" stdev="0"/>
                <controller mode="pidspeed" kp="0.2" ki="0" kd="0.01"
                kf="0.05" active="1" period="10"/>
                <color_rgb r="128" g="0" b="128"/>
                </default>

                <wheel>
                <axis angle="300"/>
                </wheel>

                <wheel>
                <axis angle="60"/>
                </wheel>
                <wheel>
        <axis angle="180"/>
        </wheel>
        </wheels>
  <shells>

  </shells>

  <articulations>

<default>
  <motor ri='3.6' li='4.88e-3' ki='0.01025' vmax='12' imax='1'
   active='1'/>
  <gear ratio='0'/>
  <friction bv='2.63e-5' fc='3.558e-4' />
  <encoder ppr='400' mean='0' stdev='0'/>
<controller mode='pidspeed' kp='1' ki='0' kd='0' kf='0.05' active='1'
 period='10'/>
```

```
</default>


 <joint>
    <ID value='ForksE'/>
    <connect B1='ForkE' B2='BasePlate'/>
    <pos x='(RobotLength/2 + ForkLength/2)/2' y='RobotWidth/6'
     z='0.025'/>
    <axis x='1' y='0' z='0'/>
    <!--limits Min='0' Max='ForkMax-ForkMin'/-->
    <type value='Hinge'/>
  </joint>

  <joint>
    <ID value='ForksD'/>
    <connect B1='ForkD' B2='BasePlate'/>
    <pos x='(RobotLength/2 + ForkLength/2)/2' y='-RobotWidth/6'
     z='0.025'/>
    <axis x='1' y='0' z='0'/>
    <!--limits Min='0' Max='ForkMax-ForkMin'/-->
    <type value='Hinge'/>
  </joint>

  <joint>
    <ID value='PesoLaser'/>
    <connect B1='BaseLaser' B2='BasePlate'/>
    <pos x='0.1' y='0' z='0.08'/>
    <axis x='0' y='0' z='1'/>
    <limits Min='-0.012' Max='-0.011'/>
    <type value='Slider'/>
  </joint>


   <joint>
    <ID value='Forkssss'/>
    <connect B1='BackFork' B2='BasePlate'/>
    <pos x='CenterMotorToFront + ForkWidth/2' y='0'
     z='0.005+(RobotHeight-ForkMIn-RobotThickness)/2'/>
    <axis x='0' y='0' z='1'/>
    <!--<limits Min='0' Max='ForkMax-ForkMin'/>    -->
```

```
    <type value='Slider'/>
    <gear ratio='1000'/>
    <friction bv='5e-3' fc='3.558e-4' />
    <controller mode='pidposition' kp='100' ki='0.2' kd='0'
     kf='0.05' active='1' period='10'/>
  </joint>


  </articulations>

  <defines>
    <!-- Sensor "dimensions" -->
    <const name='IRSharpOffsetX' value='-0.03'/>
    <const name='IRSharpOffsetY' value='-0.1'/>
    <const name='IRSharpOffsetZ' value='-0.02'/>

    <const name='LineSensorOffsetX' value='0.05'/>
    <const name='LineSensorYSpace' value='0.015'/>
  </defines>

  <sensors>

    <ranger2d>
          <ID value='ranger2d'/>
          <period value='0.1'/>
          <beam length='4' initial_width='0.01' final_width='0.015'/>
         <!-- <pos x='1' y='0' z='0.49'/> -->
          <pos x='0.1' y='0' z='0.09'/>
          <rot_deg x='0' y='0' z='180'/>
          <tag value='00'/>
          <beam angle='360' rays='360'/>
          <noise stdev='0.0' stdev_p='0' offset='0' gain='1'/>
          <color_rgb r='255' g='0' b='0'/>
    </ranger2d>

  </sensors>

</robot>
```

Referências

[1] Jean-Claude Latombe. Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts. *International Journal of Robotics Research*, 18:1119–1128, 1999.

[2] L G Fischer, R Silveira, e L Nedel. GPU Accelerated Path-Planning for Multi-agents in Virtual Environments. Em *Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on*, páginas 101–110, 2009. doi:10.1109/SBGAMES.2009.20.

[3] Sung June Chang. The hybrid optimized path finding in MMOG. Em *Proceedings of the 3rd International Planning in Games Workshop*, páginas 15–18, 2013.

[4] Manfred Lau e James J Kuffner. Behavior Planning for Character Animation. Em *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, páginas 271–280, New York, NY, USA, 2005. ACM.

[5] Anthony R Lanfranco, Andres E Castellanos, Jaydev P Desai, e William C Meyers. Robotic surgery: a current perspective. *Annals of surgery*, 239:14–21, 2004.

[6] J Petereit, T Emter, e C W Frey. Safe mobile robot motion planning for waypoint sequences in a dynamic environment. Em *Industrial Technology (ICIT), 2013 IEEE International Conference on*, páginas 181–186, 2013.

[7] Bhushan Mahajan e Punam Marbate. Literature Review on Path planning in Dynamic Environment. *International Journal of Computer Science and Network*, 2(1):115–118, 2013.

[8] Pedro Luís Cerqueira Gomes da Costa. *Planeamento Cooperativo de Tarefas e Trajectórias em Múltiplos Robôs*. Tese de doutoramento, Faculdade de Engenharia da Universidade do Porto, 2011.

[9] Roland Siegwart, Illah R Nourbakhsh, e Davide Scaramuzza. Introduction to Autonomous Mobile Robots second edition. *MIT Press*, 2011.

[10] Yong K. Hwang e Narendra Ahuja. Gross motion planning—a survey, 1992.

[11] James Ng. *An Analysis of Mobile Robot Navigation Algorithms in Unknown Environments*. Tese de doutoramento, School of Electrical, Electronic and Computer Engineering, 2010.

[12] A A Al-Haddad, R Sudirman, C Omar, e S Z M Tumari. Wheelchair motion control guide using eye gaze and blinks based on Bug algorithms. Em *Biomedical Engineering and Sciences (IECBES), 2012 IEEE EMBS Conference on*, páginas 398–403, 2012.

[13] I. Kamon, E. Rimon, e E. Rivlin. TangentBug: A Range-Sensor-Based Navigation Algorithm, 1998.

[14] S.L. Laubach, J. Burdick, e L. Matthies. An autonomous path planner implemented on the Rocky 7 prototype microrover. *Proceedings. 1998 IEEE International Conference on Robotics and Automation*, 1, 1998.

[15] Hui Miao. Robot Path Planning in Dynamic Environments using a Simulated Annealing Based Approach, 2009.

[16] S. S. Ge e Y. J. Cui. Dynamic Motion Planning for Mobile Robots Using Potential Field Method. *Autonomous Robots*, 13:207–222, 2002.

[17] E F Mohamed, K El-Metwally, e A R Hanafy. An improved Tangent Bug method integrated with artificial potential field for multi-robot path planning. Em *Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on*, páginas 555–559, 2011.

[18] Abdullah Zawawi Mohamed, Sang Heon Lee, Hung Yao Hsu, e Namrata Nath. A faster path planner using accelerated particle swarm optimization, 2012.

[19] Li Wang, Yushu Liu, Hongbin Deng, e Yuanqing Xu. Obstacle-avoidance Path Planning for Soccer Robots Using Particle Swarm Optimization. Em *2006. ROBIO '06. IEEE International Conference on Robotics and Biomimetics*, páginas 1233–1238, 2006.

[20] E. Masehian e D. Sedighizadeh. A multi-objective PSO-based algorithm for robot path planning. *Industrial Technology (ICIT), 2010 IEEE International Conference on*, 2010.

[21] Nora Sleumer e Nadine Tschichold-Gürmann. Exact cell decomposition of arrangements used for path planning in robotics. 1999.

[22] Timothy Arney. An efficient solution to autonomous path planning by Approximate Cell Decomposition. *2007 Third International Conference on Information and Automation for Sustainability*, 2007.

[23] J Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[24] K Ok, S Ansari, B Gallagher, W Sica, F Dellaert, e M Stilman. Path planning with uncertainty: Voronoi Uncertainty Fields. Em *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, páginas 4596–4601, 2013.

[25] Christopher M Clark. Dynamic robot networks: a coordination platform for multi-robot systems. *Computer Science and Software Engineering*, página 77, 2004.

[26] Yunfei Zhang, N Fattahi, e Weilin Li. Probabilistic roadmap with self-learning for path planning of a mobile robot in a dynamic and unstructured environment. Em *Mechatronics and Automation (ICMA), 2013 IEEE International Conference on*, páginas 1074–1079, 2013.

[27] Stuart Russell e Peter Norvig. Artificial Intelligence: A Modern Approach, 3rd edition. *Prentice Hall*, 2009.

[28] Bih-Yaw Shih, Chen-Yuan Chen, Hsiang Chang, e Jia-ming Ma. Dynamics and control for robotic manipulators using a greedy algorithm approach. *Journal of Vibration and Control*, 18(6):859–866, 2012.

[29] Bih-Yaw Shih, Hsiang Chang, e Chen-Yuan Chen. Path planning for autonomous robots - a comprehensive analysis by a greedy algorithm. *Journal of Vibration and Control*, 19(1):130–142, 2013.

[30] Jinsuck Kim, R.A. Pearce, e N.M. Amato. Extracting optimal paths from roadmaps for motion planning. *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, 2, 2003.

[31] A. Yamashita, M. Fukuchi, J. Ota, T. Arai, e H. Asama. Motion planning for cooperative transportation of a large object by multiple mobile robots in a 3D environment. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 4, 2000.

[32] Pedro Costa, A. Paulo Moreira, e Paulo Costa. Real-time path planning using a modified A* algorithm. Em *ROBOTICA 2009 -Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, páginas 147–152, 2009.

[33] Lothar Schulze, Sebastian Behling, e Stefan Buhrs. Automated Guided Vehicle Systems: a Driver for Increased Business Performance. *Proceedings of the International MultiConference of Engineers and Computer Scientist*, 2, 2008.

[34] Comissão Especializada para o Festival Nacional de Robótica. Robot@Factory - Competition rules and technical specifications, 2013.

[35] Servocity. S3003 Servo Standard, Junho de 2014. URL: http://www.servocity.com/html/s3003_servo_standard.html#.U6GYMd_HlpQ.

[36] Futaba. Futaba S3003 Servo Standard, Junho de 2014. URL: http://www.gpdealera.com/cgi-bin/wgainf100p.pgm?I=FUTM0031.

[37] Pololu. 30:1 Metal Gearmotor 37Dx52L mm with 64 CPR Encoder, Junho de 2014. URL: http://www.pololu.com/product/1443.

[38] Cool Components. Rover 5 Motor Driver Board, Junho de 2014. URL: http://www.coolcomponents.co.uk/rover-5-motor-driver-board.html.

[39] Paulo Costa. paco/Wiki|Main/SimTwo, Junho de 2014. URL: http://paginas.fe.up.pt/~paco/wiki/index.php?n=Main.SimTwo.

[40] Helder P. Oliveira, Armando J. Sousa, António P. Moreira, e Paulo Costa. Modeling and Assessing of Omni-directional Robots with Three and Four Wheels. Em *CONTEMPORARY ROBOTICS - Challenges and Solutions*, páginas 109–138. 2009.

[41] Amit Patel. Amit's A* Pages, Junho de 2014. URL: http://theory.stanford.edu/~amitp/GameProgramming/.

[42] Felipe Haro e Miguel Torres. A Comparison of Path Planning Algorithms for Omni-Directional Robots in Dynamic Environments. *2006 IEEE 3rd Latin American Robotics Symposium*, páginas 18–25, 2006.

[43] CE Charles E Leiserson, RL Ronald L Rivest, Clifford Stein, e Thomas H Cormen. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

[44] Marijn Haverbeke. Eloquent JavaScript - A Modern Introduction to Programming, Junho de 2014. URL: http://eloquentjavascript.net/contents.html.

[45] Gunilla Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing*, 26:270, 1984.

[46] Mark De Berg, Otfried Cheong, Marc Van Kreveld, e Mark Overmars. *Computational Geometry: Algorithms and Applications*, volume 17. 2008.

[47] Jeff Erickson. Computational Geometry -Lecture 1: Convex Hulls, Junho de 2014. URL: http://www.cs.uiuc.edu/~jeffe/teaching/compgeom/notes/01-convexhull.pdf.

[48] André S. Conceição, António P. Moreira, e Paulo Costa. Controller optimization and modelling of an omni-directional mobile robot. 2006.