FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Implementação de Algoritmos de Planeamento de Trajectórias de Robôs Móveis

Luís Miguel da Silva Ferreira Leça

PARA APRECIAÇÃO POR JÚRI

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Dr. Pedro Luis Cerqueira Gomes da Costa

Co-orientador: Prof. Dr. José Luis Sousa de Magalhães Lima

29 de Junho de 2015

Resumo

O planeamento de trajetórias tem um papel importante em diferentes áreas e aplicações entre as quais a biologia molecular, a animação por computador e a robótica. Com a introdução deste conceito em diferentes áreas existe cada vez mais um maior desenvolvimento de diferentes abordagens para diversos fins que resultam num maior avanço tecnológico.

O planeamento de trajetórias é um dos aspetos mais importantes na robótica móvel. Nesta, o planeamento de trajetórias consiste em encontrar um caminho livre de colisões através de um ambiente específico com obstáculos desde o seu ponto inicial até ao seu destino, seguindo um critério de otimização.

Esta dissertação teve como objetivo comparar diferentes métodos em ambientes dinâmicos de forma a verificar qual dos métodos se comporta melhor em diferentes situações. Para cumprir este objetivo implementaram-se três métodos de planeamento de trajetória, um método baseado em aproximação probabilística, a decomposição em células aproximadas com heurística da família A* e o método Velocity Obstacles.

Após implementados, foram testados e comparados em diferentes situações, com muitos obstáculos dinâmicos ou estáticos, em passagens estreitas, diferentes situações de colisão iminente, etc..

Como plataforma de desenvolvimento e de integração foi usado o ROS (*Robot Operating System*), uma ferramenta que permite a modularidade e a integração fácil de diferente software e hardware.

Abstract

Path planning has an important role in many specific areas and applications like molecular biology, computer animation and in the context of this dissertation robotics. With the introduction of this concept in different areas it was possible to further develop the diverse existent algorithms as well as developing and studying new approaches to this problem, which helped the development of this area.

Path planning is one of the most important aspects of mobile robotics. In this area, path planning represents finding a collision-free path through a specific environment with obstacles since its initial position to its target, following a certain criterion to optimize it.

This dissertation's objective is to compare different path planning methods in dynamic environments to check which one behaves better in different situations. To fulfill this objective three methods were implemented, one based on probabilistic approximation (RRT), another on cell decomposition, using approximated cells, with A* heuristics and the Velocity Obstacles method.

After implemented, all algorithms were tested and compared in different scenarios, with many or few dynamic and/or static obstacles, narrow paths, imminent collision scenarios, ...

ROS (Robot Operating System) was used as a development and integration platform, ROS is an interesting tool that allows modularity and easy integration of different software and hardware.

Agradecimentos

Ao meu orientador, Prof. Dr. Pedro Costa e ao meu co-orientador Prof. Dr. José Lima pelo apoio, disponibilidade e orientação prestados durante a realização desta dissertação.

Aos meus pais e restantes familiares pelo incentivo, inspiração e ânimo que me deram ao longo deste meu percurso.

Aos meus amigos e colegas, pelo companheirismo e ajuda prestados ao longo do meu percurso académico.

Obrigado.

Luís Leça

"The shortest answer is doing the thing. "

Ernest Hemingway

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

A*	Algoritmo A*
D*	*Dynamic* A*
D(x, r)	Disco com raio r na posição x
DRRT	*Dynamic* RRT
ERRT	*Extended* RRT
HRVO	*Hybrid Reciprocal Velocity Obstacles*
MSL	*Middle Sized League*
OMPL	*Open Motion Planning Library*
P2P	*Peer-to-peer*
PRM	*Probabistic Roadmap*
r	Raio
ROS	*Robot Operating System*
RVO	*Reciprocal Velocity Obstacles*
RRT	*Rapidly-exploring Random Trees*
v	Velocidade
VO	*Velocity Obstacle*
x	Posição

Capítulo 1

Introdução

Neste primeiro capítulo é introduzido o tema da dissertação bem como os seus objetivos e a motivação para a realizar.

É também apresentada a metodologia seguida bem como a estrutura deste documento.

1.1 Objetivos

O objetivo deste projeto é implementar e comparar diferentes métodos de planeamento de trajetórias num ambiente de tempo real com obstáculos dinâmicos, recorrendo ao ROS como plataforma de implementação e cujo objetivo final é implementar no futebol robótico. Pretende-se comparar os diversos algoritmos implementados de forma a verificar qual se adapta melhor a diferentes cenários, enumerando as suas vantagens e desvantagens.

1.2 Motivação

Desde a introdução do conceito de robots autónomos que o planeamento de trajetórias é uma das suas mais importantes áreas de desenvolvimento. Este permite a um robot encontrar o caminho óptimo entre dois pontos. Esse caminho pode representar o caminho mínimo mas também outros critérios, como por exemplo o caminho com menos curvas ou o caminho que resulta num menor consumo energético para o robot, dependendo da sua aplicação.

Vários algoritmos foram desenvolvidos, implementados e melhorados de acordo com o avanço da tecnologia e da maior capacidade de processamento mas também com as características do ambiente e do robot.

A introdução deste conceito noutras áreas, como por exemplo na biologia computacional [1], e também na inteligência artificial usada em videojogos e em simulações dinâmicas [2, 3] permitiu a evolução dos métodos. Também a introdução de conceitos de outras áreas como algoritmos genéticos e redes neuronais [4] auxiliou a evolução do planeamento de trajetórias, permitindo uma maior eficiência.

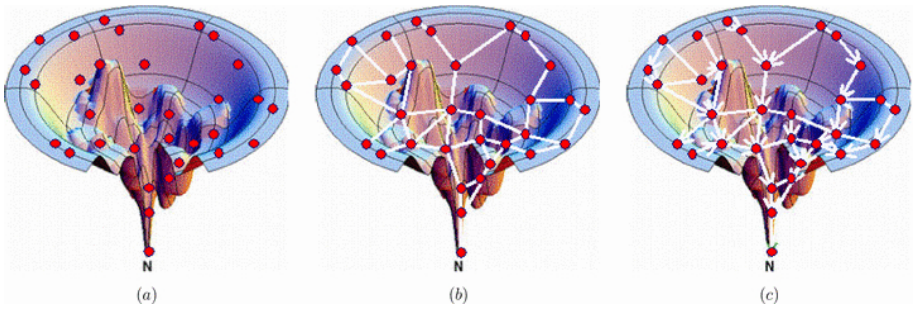


Figura 1.1: Abordagem PRM para o enovelamento de proteínas [1]

De forma a optimizar a trajetória num ambiente com obstáculos dinâmicos, pretende-se neste projecto explorar três dos métodos mais implementados neste caso, de forma a perceber qual se comporta melhor em diferentes aplicações como por exemplo num ambiente com mais ou menos obstáculos, passagens estreitas, ambientes muito populados, grandes ou pequenos, *indoor* ou *outdoor*, etc..

A plataforma de teste será o futebol robótico devido à sua versatilidade mas também complexidade, onde é possível testar diversas situações, podendo posteriormente ser testado num outro tipo de ambiente, como por exemplo industrial ou até no exterior.

1.3 Metodologia

Para realizar esta dissertação seguiu-se a seguinte metodologia:

- Estudo da ferramenta ROS e de simuladores associados;
- Estudo do estado da arte;
- Estudo dos métodos a implementar no âmbito desta dissertação;
- Implementação da navegação de robots;
- Implementação dos métodos de planeamento de trajetória;
- Testes e comparações entre os diferentes métodos de planeamento de trajetória implementados;
- Escrita do documento da dissertação.

1.4 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 5 capítulos. No capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados. No capítulo 3 é descrita a plataforma

ROS bem como as características do problema e a abordagem ao mesmo. No capítulo 4 são apresentados e explicados os métodos implementados. No capítulo 5 são apresentados os resultados obtidos e são comparados os algoritmos em diferentes situações. No capítulo 6 são apresentadas as conclusões e possível trabalho futuro.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é apresentado um levantamento teórico sobre métodos usados no planeamento de trajetórias e são apresentados conceitos relevantes à escolha dos mesmos bem como algoritmos de pesquisa em grafos.

2.1 Planeamento de trajetórias

No contexto desta dissertação, o planeamento de trajetórias referir-se-á à descrição em termos geométricos ou matemáticos dos caminhos desde o ponto inicial até ao destino, evitando colisões.

Inicialmente e de acordo com as configurações possíveis do robot, neste caso as suas posições e orientação, é preciso definir um espaço de configuração que representa as configurações possíveis do sistema. *Hwang* e *Ahuja* [5] enumeraram diferentes métodos de definição do espaço de configuração de forma a poder definir o espaço livre e os obstáculos.

Após definir este espaço é possível planear a trajetória de acordo com diferentes estratégias.

As estratégias de planeamento de trajetória podem ser classificadas como:

Roadmap Identificar caminhos dentro do espaço livre divindo-o em nós e ligações entre eles.

Decomposição em células Aproximar o mapa por células e discriminar células livres e ocupadas.

Campo Potencial Função matemática no espaço de acordo com a presença de obstáculos.

Algoritmo Bug Ambiente desconhecido, evitar obstáculos.

Velocity Obstacles Evitar obstáculos de acordo com a sua velocidade, criação de um conjunto de velocidades restritas para evitar colisões.

Outras abordagens Comportamento animal, Lógica Fuzzy, etc..

2.1.1 Roadmap

Neste algoritmo são criados nós e ligações entre eles, no caso da robótica os nós representam uma certa posição e as ligações representam o caminho entre essas posições. Pode-se assim considerar este método como uma representação baseada em grafos, que é uma das estratégias mais usadas no planeamento de trajetórias. A trajetória do robot é planeada de acordo com esse grafo de forma a ligar a posição inicial do robot ao seu destino.

Algumas das técnicas de construção do *roadmap* são apresentadas de seguida.

2.1.1.1 *Visibility Graph*

No *Visibility Graph* os nós representam o ponto inicial e final e uma certa características de objectos, que neste caso são os vértices dos obstáculos e as ligações são os segmentos de recta que conectam um nó a outro no seu campo visível, ou seja que não passam por obstáculos.

O objetivo do planeamento de trajetórias é encontrar o caminho mínimo através destas ligações. Um dos problemas deste método é o uso dos limites do espaço livre para criação do mapa, que pode resultar em colisões na ocorrência de erros no controlo.

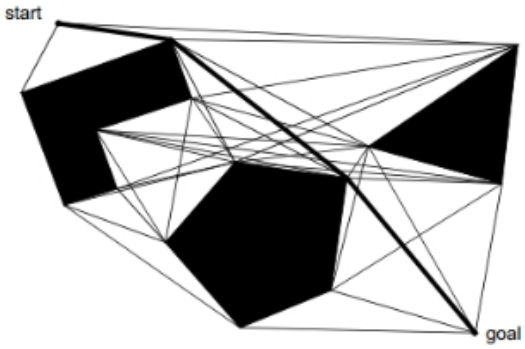


Figura 2.1: Visibility Graph [6]

2.1.1.2 Diagrama *Voronoi*

O Diagrama *Voronoi* é formado por um conjunto de pontos equidistantes entre dois ou mais obstáculos e pelas ligações entre eles, sendo estas segmentos de recta ou curvas.

O caminho a encontrar é constituído pela ligação do ponto inicial ao diagrama, o melhor caminho dentro do diagrama e a ligação deste ao destino. A tendência a maximizar a distância entre o robot e o obstáculo deste método tem como vantagem evitar colisões com obstáculos mas como desvantagem poder não resultar no caminho mínimo.

Uma vantagem interessante do uso deste tipo de diagrama está no controlo do robot, que neste caso e com o recurso aos valores dos sensores pode ser considerado bastante simples.

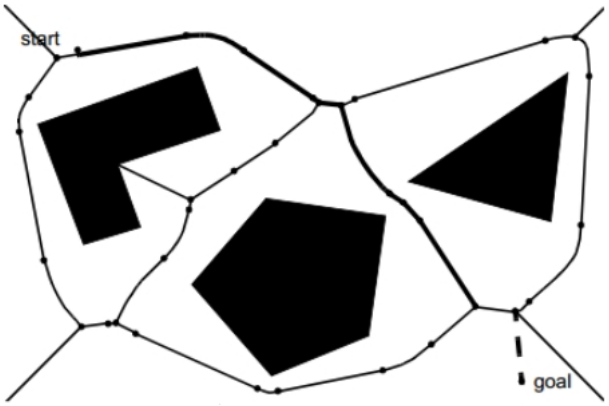


Figura 2.2: Diagrama *Voronoi* [6]

2.1.1.3 *Roadmap* Probabilístico (PRM)

Este método consiste em tirar amostras aleatórias pertencentes ao espaço e verificar se pertencem ao espaço livre ou não. Divide-se em duas fases segundo [7, 8]:

- Fase de aprendizagem, onde se constrói o *Roadmap*, de acordo com os seguintes passos:

  1. Inicializar o grafo.
  2. Gerar um nó, ou seja uma posição, aleatoriamente dentro do espaço.
  3. Verificar se cada nó pertence ao espaço livre, se sim adicioná-lo ao grafo.
  4. Verificar se cada nó pertence ao espaço livre, se sim adicioná-lo ao grafo.
  5. Repetir até ter um número adequado de nós.
  6. Para cada nó seleccionar os nós vizinhos(aqueles cuja distância é menor a uma distância máxima pré-definida).
  7. Testar se existe uma ligação através de um planeamento local, se sim acrescentar essa ligação ao grafo.
  8. Repetir passos até se considerar *Roadmap* suficientemente preenchido de forma a gerar um caminho.

- Fase de investigação, na qual se constrói o caminho entre o ponto inicial e o destino que consiste em:

  1. Encontrar caminho livre entre ponto inicial e um dos nós do roadmap.
  2. Encontrar caminho livre entre destino e um dos nós do roadmap.
  3. Encontrar caminho entre os nós definidos nos passos 1 e 2 através de um algoritmo de pesquisa em grafo explicado na secção 2.2, como por exemplo *Dijkstra*.

Este método é probabilisticamente completo, ou seja a probabilidade de a trajetória ser definida tende para 1 quando o tempo de processamento tende para infinito, podendo resultar numa pesquisa infinita caso não haja solução.

Não é completo devido à probabilidade de se definir nós em passagens estreitas ser bastante inferior à de outros locais com mais espaço, podendo também resultar na omissão da única solução, caso faça parte do caminho único entre o destino e o ponto inicial.

Para contornar este problema foram apresentadas diferentes propostas como por exemplo aumentar a probabilidade de haver nós perto dos obstáculos [9].

De forma a reduzir o tempo de execução foram desenvolvidos algoritmos como o *Lazy* PRM [10] que visa reduzir o número de verificações de colisões durante a fase de aprendizagem, procurando qual o caminho mínimo e verificar se está livre de colisões.

2.1.1.4 *Rapidly-exploring Random Trees* (RRT)

Este algoritmo, desenvolvido por *Kuffner* e *LaValle* [11], é uma variação do PRM onde os nós são construídos a partir da árvore, dos nós gerados aleatoriamente mas também recorrendo a uma distância pré-definida para incluir o nó seguinte no grafo.

Neste algoritmo podem ser usadas duas árvores, começando uma no ponto inicial e consiste nos seguintes passos [7]:

1. Inicializar as duas árvores, uma com origem no ponto inicial e a outra no destino.
2. Expandir as árvores através de:
   (a) Gerar nó q_{rand} de forma aleatória de acordo com uma distribuição normal.
   (b) Encontrar nó pertencente à árvore (q_{near}) mais próximo do nó gerado em 2a .
   (c) Verificar se um movimento com o tamanho pré-definido para as ramificações a partir de q_{near} na direção de q_{rand} resulta em colisão. Se não, acrescentar esse ramo à árvore como um novo nó q_{new}.
3. Se as árvores se uniram o caminho está definido.

Este método permite eliminar caminhos redundantes e reduz as dispersões dos nós, o que resulta numa execução mais rápida quando comparado com o PRM.

Uma análise a este método está presente foi feita por *LaValle* [12] para diferentes tipos de robots e tipos de movimentos.

De forma a poder ser usado em ambientes com obstáculos em movimento, *Bruce* e *Veloso* [13] desenvolveram o ERRT, que está constantemente a calcular novos caminhos para cada instante visto que o espaço livre está em constante mudança mas que estes não variam muito dos calculados anteriormente. Assim há um armazenamento de informação sobre o caminho para ser usado na construção seguinte.

Outra derivação do RRT para ambientes dinâmicos é o DRRT [14] que verifica se existem nós ou ligações que deixaram de pertencer ao espaço livre, eliminando-os e a partir dos nós anteriores reconstruir o grafo.

O RRT é um dos métodos implementados no âmbito desta tese.

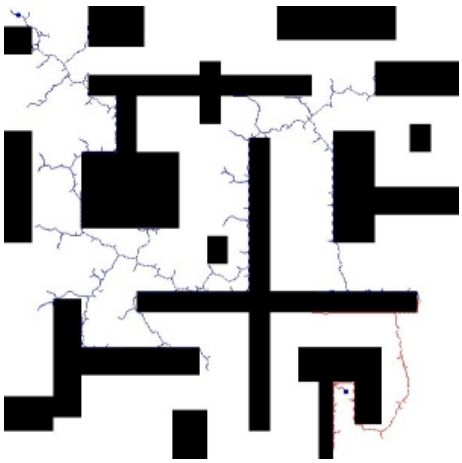


Figura 2.3: Exemplo de RRT [15]

2.1.2 Decomposição em células

Através da decomposição em células do espaço de configuração é construído um grafo, calculando se a célula está livre ou não e a sua ligação às células vizinhas. Ou seja, para planear o caminho é pesquisado o grafo e recorre-se a algoritmos de pesquisa em grafos, como por exemplo *Dijkstra*, explicados no subcapítulo 2.2.

Este tipo de método divide-se em duas fases:

1. Determinar células que contêm o ponto final e o destino.
2. Procurar caminho através de um grafo ligado pelas células adjacentes.

Existem duas variantes deste método:

- Decomposição em células exatas.
- Decomposição em células aproximadas.

2.1.2.1 Decomposição em células exatas

Neste tipo de decomposição, as células representam ou espaços completamente livres ou espaços completamente ocupados. As células são formas geométricas simples, sendo maioritariamente decompostas em polígonos convexos e trapézios.

Nesta decomposição a eficiência computacional do planeamento do caminho depende maioritariamente da complexidade dos objectos no ambiente, sendo esta a principal vantagem quando

o ambiente é largo e espaçoso mas também uma desvantagem quando o ambiente tem bastantes obstáculos de formas e complexidades diferentes.

Este tipo de métodos, de acordo com [16] não é muito implementado em robots móveis devido à sua complexidade de implementação.

2.1.2.2 Decomposição em células aproximadas

Neste tipo de decomposição as células podem ser consideradas ocupadas, livres ou parcialmente ocupadas e têm geralmente formas geométricas simples.

A forma mais popular de implementar este método, o qual vai ser implementado no projeto, é a decomposição em células fixas, na qual as células têm sempre um tamanho pré- definido.

Neste tipo de método o tamanho das células não depende dos objetos no ambiente, e a principal vantagem é a baixa complexidade computacional. Este método vai ser implementado no âmbito desta tese em conjunto com o algoritmo A*, explicado em 2.2.5.

2.1.3 Campo potencial

Este tipo de métodos usa um campo potencial artificial para encontrar a trajectória, ou seja o robot segue um campo de forças definido em cada ponto pelo gradiente do campo potencial. Este campo potencial é criado pelo destino e pelos obstáculos, em que os obstáculos criam um campo repulsivo e o destino gera um campo atrativo. A soma desses campos resulta no campo potencial. Assim o robot tenderá a ir para o destino.

Neste método o robot pode não chegar ao destino devido ao problema de mínimos locais como por exemplo quando a força resultante dos campos atrativos e repulsivo for nula há um mínimo local e o robot fica parado nesse ponto. Para evitar este problema desenvolveram-se métodos para resolução desses mínimos. Outra forma de evitar este problema consiste em criar um campo potencial onde o único mínimo local é o destino.

Para além disto foram feitas combinações deste método com outros métodos, como por exemplo algoritmos genéticos [17].

De acordo com *Hwang* e *Ahuja* [18] a abordagem para encontrar o caminho é a seguinte:

1. O espaço livre é representado por um grafo consistido por um número finito de nós e arestas, correspondentes a pontos formados pelo campo potencial.

2. Atribuir custo a cada nó de acordo com a largura do espaço livre à sua volta.

3. Encontrar caminho candidato que minimiza o percurso mas também a hipótese de colisões.

4. Com recurso a um planeamento local, o caminho é modificado de forma a derivar um caminho final livre de colisões.

5. Se caminho livre de colisões não for encontrado retirar aresta em que ocorre colisão e voltar a 3.

6. Repetir 3 a 5 até solução ser encontrada ou caminho não existir.

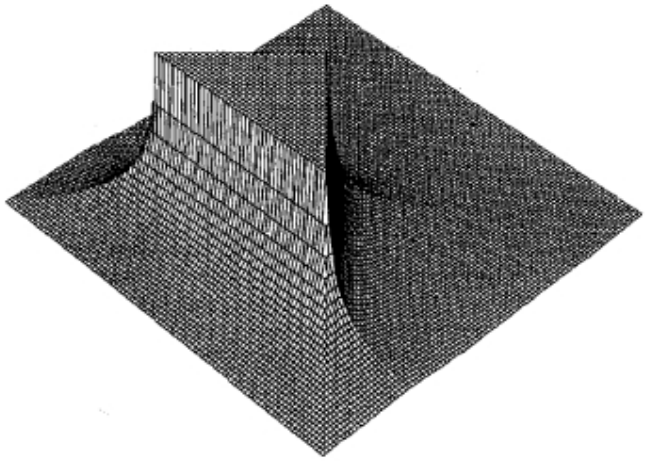


Figura 2.4: Campo potencial gerado por um obstáculo 2D triangular [18]

2.1.4 Algoritmos *Bug*

Este é considerado o algoritmo mais simples usado para evitar obstáculos.

É usado em casos em que o ambiente é desconhecido, baseando-se apenas nos seus sensores para se localizar e chegar ao destino. Este tipo de algoritmos assume que o robô sabe sempre a sua localização, tendo como referência o seu ponto de origem.

Alguns dos algoritmos Bug são apresentados de seguida.

O algoritmo *Bug1* foi o primeiro algoritmo na família *Bug*, tendo sido criado por *Lumelsky* e *Stepanov* [19]. Neste algoritmo o robot segue desde o ponto inicial até ao destino e se encontrar um obstáculo no seu percurso contorna-o até regressar ao ponto mais próximo do destino no obstáculo e segue para o destino.

O algoritmo *Bug2* é um melhoramento do *Bug1* onde é traçado um segmento de recta virtual desde o início até ao destino que é seguida até chegar ao destino ou até encontrar um obstáculo, neste caso contorna-o até encontrar um novo ponto pertencente ao segmento de recta e volta a segui-lo.

O algoritmo *Bug2+* [20] acrescenta uma condição ao *Bug2* de apenas seguir a recta se ainda não passou por pontos na mesma mais próxima do destino, evitando assim que ocorram casos em que se repetem caminhos.

2.1.5 *Velocity Obstacles*

Este método foi introduzido na robótica por *Fiorini* [21]. Tem como base definir o conjunto de velocidades que resultam em colisão com um obstáculo em movimento num dado instante, assumindo que a velocidade deste se mantém. Assim, este conjunto contém as velocidades rejeitadas.

Ao escolher uma velocidade fora desse conjunto, ou seja fora do *velocity obstacle*, garante-se que essa colisão não ocorre. Para planear a trajectória é necessário ter em conta todos os *velocity obstacles* e considerar uma velocidade admissível que não esteja dentro desses conjuntos. O conjunto de velocidades rejeitadas é definido pela equação 2.1.

$$VO_{A|B} = \{v|\exists t > 0 : (v - v_b)t \in D(x_B - x_A, r_A + r_B)\} \tag{2.1}$$

Na qual A representa o robot e B um obstáculo dinâmico, D(x, r) representa o disco na posição x de raio r e v e v_b representam, respetivamente, a velocidade do robot e do obstáculo.

Existem diferentes variações deste método tais como o *Hybrid Reciprocal Velocity Obstacles* (HRVO), que evita oscilações do robot pois tem em conta que os outros robots também mudam o seu comportamento de acordo com o ambiente em seu redor [22].

O método *Velocity Obstacles* é abordado no âmbito desta tese devido à sua constante evolução e à sua interessante abordagem relativa a ambientes dinâmicos.

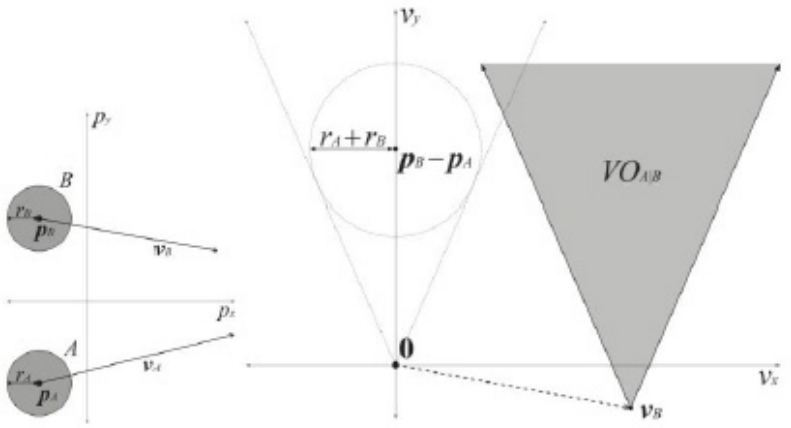


Figura 2.5: *Velocity Obstacle* induzido por B em A [23]

2.1.6 Outras abordagens

Para além dos métodos referidos anteriormente, foram implementados no planeamento de trajetórias conceitos como redes neuronais [24], algoritmos genéticos [25], lógica *fuzzy* [26, 27] e conceitos relacionados com comportamentos animais, usando como exemplo o comportamento de uma colónia de formigas [28].

2.2 Algoritmos de pesquisa

Após construir o mapa e a sua representação num grafo através de nós e ligações é necessário escolher um algoritmo que permite realizar a pesquisa desde o nó final até ao nó inicial, de forma a encontrar o melhor caminho. Deve-se escolher o algoritmo de forma a ser o mais eficiente possível.

Estes dividem-se em algoritmos com e sem heurística. Os mais relevantes dentro dos algoritmos sem informação são a Pesquisa em Largura, Pesquisa em Profundidade e suas variações e dentro dos algoritmos com heurística os mais relevantes são o algoritmo *Dijkstra's*, o algoritmo guloso e o A* e suas variantes.

Para medir a eficiência e a eficácia de cada algoritmo são considerados factores como o tempo necessário para encontrar a solução, se encontra sempre a solução caso exista, ou seja se é completo e se é óptimo, no caso de encontrar a melhor solução.

2.2.1 Pesquisa em largura

O algoritmo explora todas as ligações de um nó e assim sucessivamente até encontrar o destino. É óptimo caso o custo de todas as ligações seja igual, caso contrário a solução pode não ser óptima.

2.2.2 Pesquisa em profundidade

A pesquisa é feita até ao último nó de cada ramo, ou seja escolhe a primeira ligação para cada um dos nós até encontrar o destino ou um nó sem ligações. Este método é completo caso o número máximo de ligações seja finito. Não é óptimo e explora nós já visitados, sendo necessário guardar cada um deles em memória.

2.2.3 Algoritmo *Dijkstra*

Este algoritmo é semelhante à pesquisa em largura, explora todos os nós ligados ao inicial, escolhendo após isso o nó em que a ligação tem o menor custo, ou seja o nó mais próximo do inicial. O nó seguinte será esse e assim sucessivamente até chegar ao ponto final. Tal como no caso da pesquisa em largura é completo caso o número de ligações máximo dos nós seja finito, e não é óptimo excepto nos casos em que o custo de todas as ligações seja igual.

2.2.4 Algoritmo do tipo guloso

Os algoritmos do tipo guloso têm em consideração o quanto falta para alcançar o destino, e escolhem os nós que estão mais próximos desse destino a partir do nó inicial e assim sucessivamente. Tal como no caso anterior o algoritmo é completo caso o custo associado às ligações ser constante.

2.2.5 Algoritmo A*

O algoritmo A* é semelhante ao *Dijkstra's* (2.2.3), mas usa também a informação de quanto falta para alcançar o destino, procurando primeiro nos nós com condições mais favoráveis. Este algoritmo é óptimo e completo e a sua complexidade de espaço e de tempo é dependente da heurística adoptada para calcular a distância ao destino.

De acordo com *Costa* [29] o algoritmo corresponde aos seguintes passos:

Algoritmo 1: A*

1. Adicionar o nó de partida a O.
2. Repetir os seguintes passos até que O esteja vazio.
   (a) Escolher o n melhor (nó melhor) de O tal que f(n_{melhor}) $\leq$ f(n) $\forall$ n $\in$O.
   (b) Remover o n_{melhor} de O e adicionar a C.
   (c) Se n_{melhor} = nó final, terminar o algoritmo.
   (d) Para todos n $\in$ *Star* (n_{melhor}) fazer o seguinte:
      i. Se (n $\notin$ O) e (n $\notin$ C) então adiciona o nó n a O.
      ii. Se (n $\in$ O) então se g(n_{melhor}) + c(n_{melhor} , n) < g(n) então alterar o pai do nó n para n melhor.
      iii. Se (n $\in$ C) então se g(n_{melhor}) + c(n_{melhor}, n) < g(n) então alterar o pai do nó n para n_{melhor} e passar n para a lista aberta (O).

No qual são consideradas as seguintes definições:

- n: nó.
- g(n): função custo desde o início até nó n.
- h(n): função heurística de custo que estima a distância do nó n ao destino.
- f(n): função custo (g(n)+h(n)).
- O: lista aberta que contém os nós ainda não analisados e que poderão vir a ser escolhidos.
- C: lista fechada que contém os nós já processados.
- *Star*(n): conjunto de nós vizinhos ao nó n.
- c(n1, n2): custo de ir desde o nó n1 ao nó n2.

Este algoritmo é aplicado sobre um mapa dividido em células podendo a heurística usada para calcular o percurso que falta até chegar ao destino ser a distância mínima entre a célula em estudo e a célula correspondente ao destino.

Este planeia o caminho entre a posição inicial do robot e o destino, o qual vai ser seguido pelo robot até encontrar o destino ou uma discrepância entre o mapa gerado e o ambiente, quando este caso ocorre há uma reconstrução do mapa e uma nova escolha de caminho.

Este método pode ser considerado pouco eficiente em ambientes com obstáculos dinâmicos, por isso foram introduzidas variantes que permitem lidar com obstáculos dinâmicos de uma forma mais eficiente.

Uma destas variantes foi proposta e introduzida por *Moreira e Colaboradores* [30] que introduz na construção do mapa um método que calcula obstáculos estáticos a partir dos dinâmicos de acordo com características como a distância e a direcção incluindo-as também na função custo.

Outra variante é o D* (*Dynamic* A*) [31] que altera os caminhos à medida que o robot vai recebendo novas informações do ambiente.

O A* é um dos métodos abordados nesta dissertação por ser um método considerado óptimo e completo.

Capítulo 3

Arquitectura

Neste capítulo é apresentada a framework usada no âmbito desta tese para implementar os diversos métodos, bem como a abordagem adotada para o problema. São também apresentadas as características do meio e dos robots, são introduzidos os conceitos de plano global e local e é ilustrada a navegação dos robots.

3.1 *Robot Operating System* (ROS)

Os métodos apresentados no capítulo 4 foram implementados na framework do *Robot Operating System* (ROS). Este foi desenhado como interface entre hardware e software em plataformas robóticas. Um dos seus principais objetivos é possibilitar a modularidade e a reutilização de recursos e código em pesquisa e desenvolvimento.

Este possui várias ferramentas, serviços e um sistema baseado em nós que permite troca de mensagens. Estes nós representam módulos que funcionam de forma independente e comunicam entre eles através de tópicos através de um modelo publicação-subscrição e o protocolo TCP/IP.

Isto permite uma fácil integração de novos componentes, sensores e de software num ou mais robots. É assim possível separar e melhorar o desempenho no processamento de dados dos sensores, localização e navegação do robot bem como melhorar o desempenho dos controladores.

Cada sistema tem um mestre que publica a informação sobre quais os tópicos que cada nó publica, desde que estes estejam registados.

Para os nós subscreverem um certo tópico faz um pedido ao mestre para o endereço desse tópico e quando o obtém liga-se diretamente ao mesmo, resultando numa ligação P2P.

De forma a melhorar ainda mais a sua funcionalidade são desenvolvidas *packages* e *stacks* que integram diferentes nós específicos para uma certa tarefa. Estas estão em constante desenvolvimento por serem *open-source*.

A modularidade do ROS permite criar várias configurações para diferentes situações e configurações. Os módulos podem ser usados em conjunto ou separados, o código usado em simulação pode ser o mesmo que no caso real, sendo necessário apenas alterar os módulos relativos a *drivers* para sensores, *encoders*, etc. Para correr o sistema apresentado em robots adaptados ao ROS,

é apenas necessário alterar as características do robot como o modelo do sensor,o seu tipo de movimento e outros parâmetros.

Um exemplo dos nós e tópicos usados para um robot estão presentes na figura 3.1 Para os visualizar recorreu-se à ferramenta *rqt*.

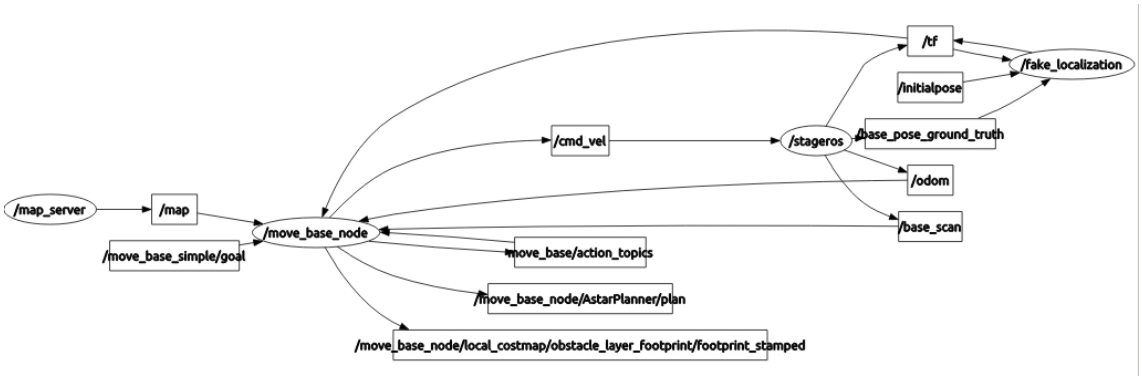


Figura 3.1: Exemplo de nós e tópicos publicados para um robot

3.2 Abordagem

Para implementar um sistema multi-robots optou-se por usar uma abordagem distribuída.

Neste tipo de abordagem as decisões dos vários robots são feitas pelos mesmos. Cada robot é independente e toma decisões de acordo com as informações dadas pelos seus sensores. Têm também conhecimento sobre o mapa e cada robot comunica dados sobre a sua posição. Comparando com uma arquitetura centralizada, onde as decisões são dadas por um mestre e todo o tipo de informações é guardado e fornecido pelo mesmo, este tipo de abordagem é mais robusta e necessita de menor poder computacional. Está também mais preparado para situações em que existem muitos robots.

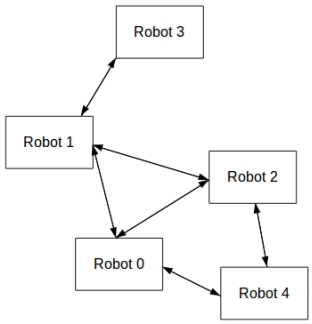


Figura 3.2: Exemplo de arquitectura distribuída

3.2.1 Características do robot e do mapa

As características dos robots foram implementadas de acordo com a *RoboCup Middle Size League* do futebol robótico, foram usadas como dimensões do robot dimensões perto da máxima permitida (em largura e comprimento) nesta liga, 50 cm x 50 cm x 20 cm (largura, comprimento, altura). Foi considerada como velocidade máxima 0,5 m/s. Os robots móveis implementados são omnidireccionais, ou seja podem-se movimentar em qualquer direção num espaço 2D (x,y,θ), os graus de liberdade controláveis são iguais aos graus de liberdade totais.

Cada um deles está equipado com um sensor de forma a detectar obstáculos desconhecidos, neste caso foi implementado em simulação um sensor laser. O seu deslocamento e posição é dado por odometria.

Foram implementados dois tipos de mapa, um está de acordo com as dimensões da MSL, 12 por 18 metros sendo apenas considerados como obstáculos os restantes robots. A outra situação representa um mapa com obstáculos estáticos mais complexos, podendo representar um meio industrial.

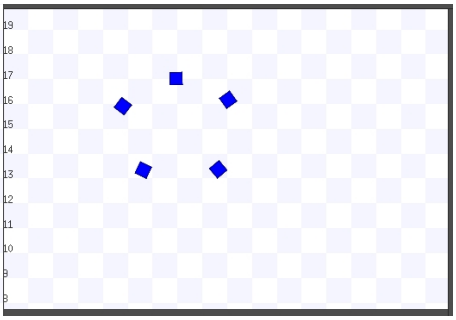


Figura 3.3: Simulação num mapa com dimensões da *MSL* e exemplos de robots

3.2.2 Plano Global e Plano Local

De forma a construir e seguir a trajetória é necessário definir um plano global e um local.

O planeador global define um caminho de alto-nível e de média resolução entre o ponto inicial do robot e o seu destino.

O planeador local ou controlador local foca-se numa região bastante inferior do mapa, apenas parte do caminho gerado pelo planeador global e rege as velocidades linear e angular do robot. Informa também o planeador global quando são encontrados obstáculos não presentes no mapa originalmente usado no planeamento global. Este recebe e depende de informação dos sensores (mapa local), da estimativa da sua localização e também do destino proveniente do caminho global.

Focando nos métodos implementados o Velocity Obstacle (VO) representa um plano local, controla as velocidades, não foca a elaboração de um caminho mas sim evitar obstáculos, enquanto que os algoritmos A* e RRT são planos globais, focam a construção do caminho.

O plano local usado em conjunto com o A* e com o RRT teve como objetivo dar velocidades linear e angular seguras tendo em conta as características do robot e do meio e também seguir o

caminho traçado. Ao encontrar um obstáculo desconhecido, há um replaneamento por parte do planeador global atualizando o mapa.

O plano global usado para testes em conjunto com o VOs teve apenas como objetivo fazer um caminho direto entre a posição inicial e o destino.

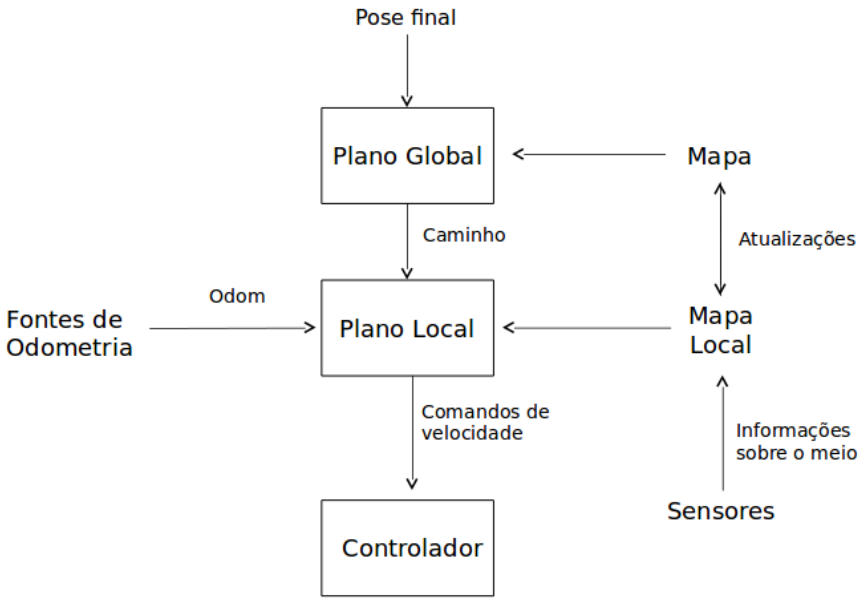


Figura 3.4: Abordagem global e local

3.2.3 Navegação

A navegação de um robot foi feita adaptando a *Stack* de navegação do ROS [32] e o esquema das mensagens trocadas entre o nó principal e o simulador estão presentes na figura 3.5 .

Esta combina um plano global e um local de forma a seguir uma trajetória de acordo com o mapa estático mas também evitar obstáculos dinâmicos e desconhecidos (não presentes no mapa).

Recebe também informações provenientes de sensores e de odometria bem como a pose inicial e final do robot e envia comandos de velocidade (mensagens do tipo *geometry_msgs/Twist*) para a base.

De acordo com a figura 3.5 as mensagens transmitidas pelos diferentes tópicos e nós são explicadas de seguida:

- Pose atual: Pose atual do robot (x,y,θ) é fornecida pelo simulador e subscrita pelo plano global. O seu tipo de dados é geometry_msgs::PoseStamped.

- Destino: Pose pretendida para o robot (x, y, θ). Destinada ao plano global. Tipo de dados: geometry_msgs::PoseStamped.

- *Path*: Conjunto de pontos (x, y) fornecido pelo plano global ao plano local que representam a trajetória a seguir. Tipo de dados: nav_msgs::Path, que é constituído por um header e por um vetor de poses (geometry_msgs::PoseStamped).
- Odom: Mensagem proveniente do simulador que contém informações sobre velocidades e posição do robot. Tipo de dados: geometry_msgs::Pose (pose) + geometry_msgs::Twist (velocidades).
- cmd_vel: Proveniente do plano local é usado pelo simulador. Envia para o simulador a velocidade angular e linear a usar pelo robot. Tipo de dados: geometry_msgs::Twist.

São também recebidas mensagens provenientes dos sensores que permitem atualizar tanto o mapa local como o global.

O nó principal de cada robot, onde estão contidos os tópicos especificados recebe também informações das transformadas dos sensores de forma a poder se localizar com recurso aos mesmos.

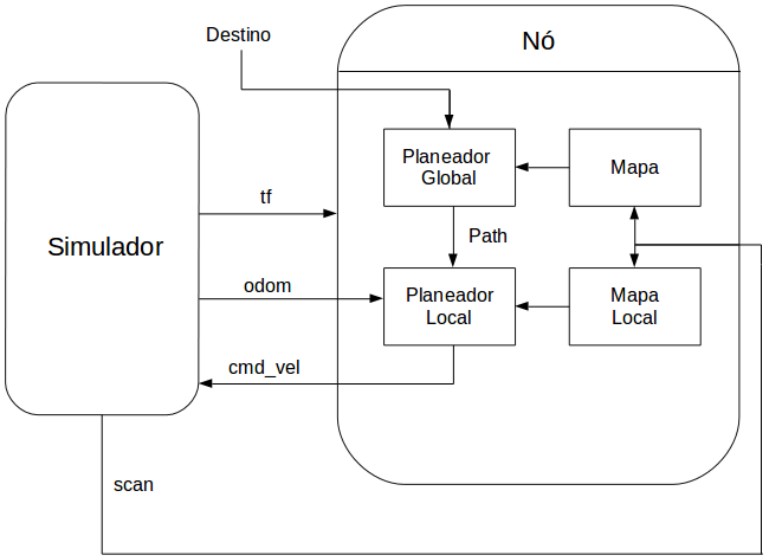


Figura 3.5: Mensagens entre o simulador e o nó principal

3.3 Resumo

Neste capítulo foram apresentadas as abordagens tomadas para implementar os métodos. Foram descritas as características dos ambientes de teste bem como do robot e foi apresentada a arquitectura dos sistemas.

Foi também introduzida a ferramenta ROS bem como as mensagens e tópicos associados ao mesmo.

No capítulo seguinte são apresentados e explicados os métodos implementados bem como a sua implementação.

Capítulo 4

Métodos implementados

Neste capítulo são apresentados os algoritmos implementados, suas características e a razão para terem sido escolhidos estes métodos. Na secção 4.1 é apresentado o método RRT, um algoritmo baseado em amostragem, probabilístico e são explicadas e justificadas as suas variantes implementadas.

Na secção 4.2 é apresentado o método de decomposição em células com heurística A*. É explicada a decomposição em células fixas e a sua implementação bem como a do algoritmo A* com heurística baseada em distância euclidiana.

Na secção 4.3 é apresentado o método *Velocity Obstacles*, um método específico para evitar colisões. Este recorre às velocidades do robot e dos obstáculos para classificar obstáculos no espaço de velocidades que restringem o robot de forma a evitar colisões. São também mostradas duas variantes deste algoritmo que demonstram melhores resultados em ambientes multi-robot.

4.1 *Rapidly-exploring Random Trees*

4.1.1 Descrição

O RRT é um método de planeamento de trajetória baseado em amostragem que tem em conta as restrições do mapa (obstáculos) mas também restrições relativas ao próprio robot (dinâmica do robot). Tem como objectivo dar prioridade na exploração às zonas ainda não exploradas do mapa.

Duas variantes deste método foram implementadas, o RRT* (*Optimal* RRT) e o pRRT (*Parallel* RRT). A primeira foi escolhida pois é uma versão que é assintoticamente óptima, ou seja tende a encontrar não só um caminho entre o início e o destino mas também a encontrar a melhor trajetória de acordo com as restrições do meio e do robot e a otimizar o caminho. O algoritmo do método RRT*, implementado de acordo com [33] é apresentado no algoritmo 2.

Algoritmo 2: RRT*

Inicializar árvore T;
Inserir nó x_{init} na árvore T;
enquanto $i < N$ **faça**
  $x_{rand} \leftarrow Amostrar(i)$; $i \leftarrow i+1$;
  $x_{nearest} \leftarrow MaisProx(T, x)$;
  $(x_{new}, u_{new}, T_{new}) \leftarrow Aproxima(x_{nearest}, x_{rand})$;
  se *LivreObstáculos*(x_{new}) **então**
    $X_{near} \leftarrow Prox(T, x_{new}, |V|)$;
    $x_{min} \leftarrow x_{nearest}$;
    $x_{min} \leftarrow EscolherPai(X_{near}, x_{nearest}, x_{new})$;
    $T \leftarrow InserirNo(x_{min}, x_{new}, T)$;
    $T \leftarrow Reconstruir(T, X_{near}, x_{min}, \text{traj}_{new})$;
  fim
fim
Retornar árvore T;

No qual foram usadas as seguintes funções:

- *Amostrar* - Escolher novo nó de forma aleatória dentro do espaço.
- *MaisProx*- Encontrar nó mais próximo do nó x na árvore T.
- *Aproxima* - Encontrar nó seguinte através do nó amostrado x_{rand} e do mais próximo dentro da árvore.
- *InserirNo* - Inserir novo nó na árvore.
- *LivreObstáculos* - Verifica se caminho entre nós não contém obstáculos.
- *Prox* - Encontra e liga nós na vizinhança $|V|$ do novo nó a este. Neste caso foram considerados os três nós vizinhos mais próximos.
- *EscolherPai* - Escolhe nó com melhor ligação ao nó x_{new}.
- *Reconstruir* - Remove ligações a nós que não fazem parte do caminho mínimo entre nó inicial e o nó em análise.

Este algoritmo difere do original pois tenta ligar os nós dentro de um certo raio à volta do novo nó criado e não só do anterior ou do mais próximo acrescentando novas ligações. Após esta construção remove as ligações redundantes, as que não resultam num caminho mínimo entre o ponto inicial e o vértice em análise.

O segundo caso (pRRT) foi escolhido pois revelou ter um tempo de processamento significativamente menor que o caso anterior mas também para casos em que é necessário encontrar um caminho num ambiente mais complexo, como um labirinto no qual o RRT* tem menor sucesso

que o RRT. Para reduzir o tempo de processamento recorre a várias *threads* que juntam simultaneamente estados à mesma árvore, resultando num maior número de nós no mesmo intervalo de tempo. Para o mesmo caso obteve-se os resultados presentes na tabela com as duas variantes.

Ambos os métodos são probabilisticamente completos, a probabilidade de encontrar a solução tende para 1 à medida que o número de estados máximo permitido tende para infinito.

4.1.2 Bibliotecas usadas

Para implementar os dois métodos referidos (RRT* e pRRT) recorreu-se às bibliotecas OMPL (*Open Motion Planning Library*) [34]. Estas consistem num conjunto de métodos de planeamento de trajetória baseados em amostragem.

Para as usar é necessário converter os dados do mapa e da configuração do robot para um espaço de estados, que representa todas as configurações possíveis do robot no espaço em que o mesmo opera, sendo assim o caminho é um conjunto de estados no espaço de estados.

Após definir o espaço de estados e obter informações sobre o mesmo é necessário definir o número de passos mínimo e máximo e o tamanho de cada um deles. De forma a garantir um bom resultado tanto em distância como em tempo definiu-se múltiplos de 10 centímetros como o possível tamanho de cada passo (cada ligação entre nós) e um máximo de 500 passos entre o início e o destino.

Após esta definição os estados propagam-se de acordo com a dinâmica e com a configuração do robot explicada em 3.2.1 e também de acordo com o método escolhido.

Esta solução permitiu encontrar soluções na maioria dos casos, contudo o tempo de processamento foi consideravelmente maior que nos outros métodos implementados, razão pela qual se comparou com a variante pRRT. Uma comparação entre as duas variantes está presente na tabela 4.1 e nas figuras 4.1 e 4.2. Esta comparação foi feita através do *software rviz* numa situação com obstáculos estáticos.

Tabela 4.1: Comparação entre variantes do RRT

	RRT*	**pRRT**
t_{proc} (ms)	1119.8	10.16
d (m)	11.45	21.59

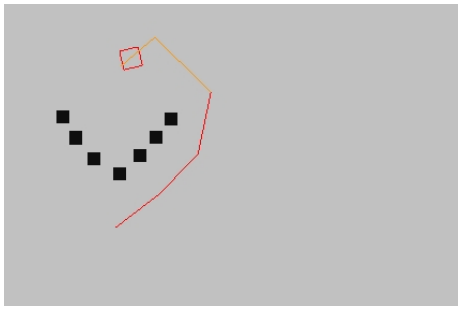


Figura 4.1: RRT*

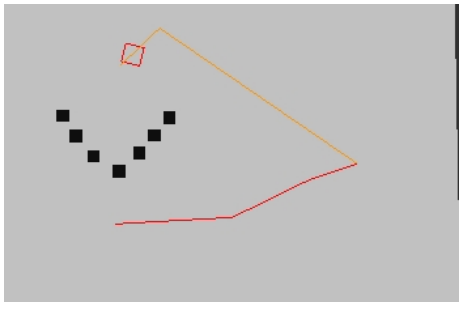


Figura 4.2: pRRT

Tal como é possível verificar pelas figuras 4.1 e 4.2 a trajetória do RRT* é bastante menor que no caso do pRRT.

4.1.3 Obstáculos

Nos casos em que existem obstáculos estáticos, este método exclui os estados que estão num obstáculo e as ligações que passam num obstáculo considerando apenas os caminhos livres e optando pelos de menor custo, obtido de acordo com os parâmetros que o definem, como por exemplo número de curvas e distância entre nós.

Quando o ambiente é desconhecido ou na existência de obstáculos dinâmicos, a árvore é re-calculada a partir do ponto em que encontrou um obstáculo, resultando num novo caminho.

4.2 Decomposição por células com heurísticas da família A*

4.2.1 Descrição

Como já foi apresentado em 2.2.5, o algoritmo A* é um método de pesquisa em grafo que retorna o caminho mínimo de acordo com as informações do mapa, ou seja faz o seu planeamento de forma informada, sendo implementada uma heurística de acordo com esta.

Esta heurística, $f(n)$ é usada para estimar caminho com o custo mínimo desde o nó n até ao destino, podendo-se dividir em duas outras funções $g(n)$ e $h(n)$, *g(n)* representa o custo desde o nó inicial até ao nó atual através das células. *h(n)*, considerada a heurística representa o custo desde o nó atual até ao nó correspondente ao destino, onde este custo é definido de acordo com uma distância definida entre os dois nós. Estas funções estão representadas na figura 4.3.

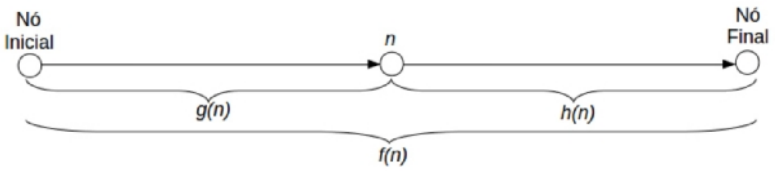


Figura 4.3: Funções *f(n)*, *g(n)* e *h(n)*

Para o implementar é necessário primeiro dividir o mapa em células de forma a obter um conjunto de nós onde o A* possa pesquisar. Neste caso o mapa foi decomposto em células fixas de 10 centímetros, que tendo em conta as características do meio e do robot é um valor adequado que não compromete significativamente o tempo de processamento. Posteriormente cada célula é analisada e classificada como livre ou ocupada de acordo com o mapa.

Cada uma das células representa um nó e tem ligação directa a outras 8 células, ou seja conectividade 8, sendo a pesquisa feita nas células adjacentes. O custo para cada uma destas ligações é dado pela distância ao ponto atual.

Os custos das ligações da célula X aos vizinhos estão de acordo com a figura 4.4.

√2	1	√2
1	x	1
√2	1	√2

Figura 4.4: Custos da vizinhança relativos à célula x

4.2.2 Implementação

Visto o A* ser um método de pesquisa em grafo o próprio robot vai ser considerado do tamanho de uma célula na pesquisa. Para evitar colisões é necessário inflacionar tanto os obstáculos dinâmicos como os estáticos. Assim, os obstáculos foram inflacionados de acordo com as características do robot. Um exemplo de inflação de um obstáculo está presente nas figuras 4.5 e 4.6.

Na figura 4.7 é demonstrada através de um código de cinzentos a mesma inflação, na qual se assume o seguinte:

- Branco - Espaço livre;
- Preto - Obstáculo;
- Cinzento escuro - Espaço a evitar pelo centro do robot de forma a evitar colisões (inflação do obstáculo);
- Cinzento claro - Zona de segurança a evitar mas permitida.

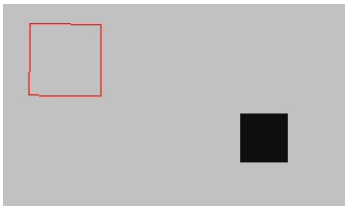


Figura 4.5: Obstáculo antes de expansão

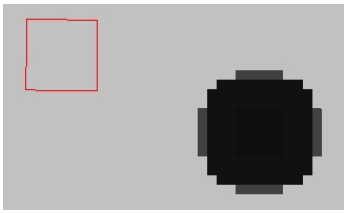


Figura 4.6: Obstáculo após expansão

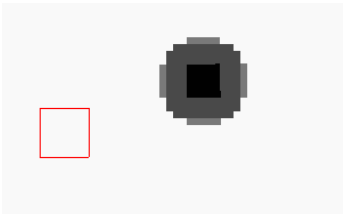


Figura 4.7: Obstáculo após expansão com esquema de cores

O pseudo-algoritmo implementado está presente no algoritmo 3.

Algoritmo 3: Pseudo-Algoritmo A*

1. Inserir nó inicial na lista aberta;
2. Escolher nó da lista aberta com menor função custo f;
3. Retirar melhor nó da lista aberta e introduzi-lo na lista fechada;
4. Se nó actual igual ao nó destino finalizar;
5. Pesquisar em todos os vizinhos do melhor nó que não pertencem à lista fechada;
   (a) Se não pertence à lista aberta, insere na lista aberta;
   (b) Se pertence, verificar se custo até chegar ao nó vizinho é menor que o anteriormente atribuído, se for mudar pai do nó analisado para o melhor nó atual;
6. Repetir 2 a 5 até lista aberta estar vazia;

A lista aberta é constituída pelos nós identificados mas não processados enquanto que a lista fechada armazena os nós processados.

À medida que são visitados os nós são guardadas as seguintes informações:

- Índice convertível em coordenadas x e y;
- Custo g;
- Custo h;
- Custo f;
- Índice do pai.
- Lista a que pertence.

Após este algoritmo ser realizado a trajetória é definida a partir do nó correspondente ao destino escolhendo os pais de cada nó analisado.

No caso em que a lista aberta fique vazia sem que a célula correspondente ao destino seja encontrada é retornado que não existe caminho possível entre a posição inicial e o destino.

4.2.3 Heurística

Existem várias heurísticas possíveis para aplicar num mapa de células. Para garantir que o algoritmo é óptimo esta deve-se aproximar o máximo possível da realidade mas também subestimar os custos reais de cada nó.

A heurística implementada foi a distância euclidiana ou seja a função *h(n)* foi implementada de acordo com a equação 4.1.

Implementou-se esta heurística dado que o robot se pode mover em qualquer direção.

$$h(n) = K * D * \sqrt{(x_{destino} - x_n)^2 + (y_{destino} - y_n)^2} \tag{4.1}$$

D representa o peso da mudança de célula que pode ser considerado 1 e K é um valor usado para dar mais prioridade ou não às células mais próximas do destino. Neste caso usou-se K = 1 de forma a garantir a otimalidade.

4.2.4 Estrutura de dados

Grande parte do tempo de processamento do algoritmo A* está relacionado com as tarefas de inserção e remoção de nós das listas bem como na procura do nó com a menor função custo. Assim, de forma a reduzir o tempo de processamento é necessário considerar a estrutura de dados usada para a lista aberta.

Tendo em conta que as tarefas necessárias para esta lista são inserir, procurar e eliminar é necessário implementar uma estrutura cujo tempo despendido nestas seja o mínimo possível. Optou-se por implementar a lista através de uma estrutura *multiset* (ou *heap*), cuja complexidade para a pesquisa, inserção e remoção são O(log(n)). Visto que o objetivo é encontrar o nó com o menor custo pode-se considerar a complexidade para a pesquisa do mesmo O(1) visto este tipo de estrutura se encontrar ordenada.

4.2.5 Obstáculos

No caso em que os obstáculos são conhecidos, ou seja fazem parte do mapa, o algoritmo considera as células ocupadas evitando-as no momento em que é feita a procura, considera só as células livres na pesquisa.

Ao identificar obstáculos desconhecidos na sua trajetória é feito um novo cálculo do caminho e as células correspondentes aos mesmos são consideradas ocupadas, resultando numa nova trajetória.

4.3 *Velocity Obstacles*

4.3.1 Descrição

Este método, ao contrário dos anteriores não visa desenhar uma trajetória e segui-la mas sim evitar obstáculos à medida que os encontra, ou seja o plano inicial a seguir é o caminho directo desde a posição atual até ao destino. Não considera informação do mapa mas sim os vectores correspondentes às velocidades dos obstáculos que encontra. É assim considerado um algoritmo reativo tal como o *Bug* apresentado em 2.1.4.

Um *Velocity Obstacle* corresponde a uma representação geométrica de todas as velocidades que podem resultar em colisão com um obstáculo dinâmico com velocidade constante.

Tem como objetivo definir a velocidade do robot de forma a evitar o *Velocity Obstacle* cuja definição está presente na equação 4.2.

$$VO_{A|B} = \{v|\lambda(p_A, v - v_B) \cap B \oplus -A \neq \emptyset\} \tag{4.2}$$

Na qual A representa o robot, p_A a sua posição e v a sua velocidade, B representa um obstáculo dinâmico, v_B a sua velocidade e A⊕B = $\{a+b|a \in A, b \in B\}$ e $\lambda(p,v)$ uma semi-recta com início na posição p e direção de v.

Se a velocidade do robot estiver incluída neste conjunto existe risco de colisão e é necessário alterar a mesma de forma a evitar colisões.

Foi implementado de acordo com [35, 36] tendo sido implementado o algoritmo original mas também duas variantes que permitem um melhor desempenho quando os obstáculos dinâmicos são robots que seguem o mesmo algoritmo. Nestes casos o *VOs* resulta em oscilações pois os robots estão constantemente a adaptar a sua velocidade de acordo com a dos outros, alterando os *VOs*.

As duas variantes implementadas foram o RVO (*Reciprocal Velocity Obstacles*) na qual o robot considera que o obstáculo dinâmico também evita colisões, apenas tendo em conta metade do problema e o HRVO (*Hybrid Reciprocal Velocity Obstacles*).

Este último estabelece prioridades consoante a direção da velocidade dos robots, dando prioridade a direções para a direita ou esquerda do RVO e aumenta os VOs criados do lado oposto de acordo com o algoritmo original.

Um exemplo que mostra os obstáculos gerados por cada uma das três variantes está presente na figura 4.8.

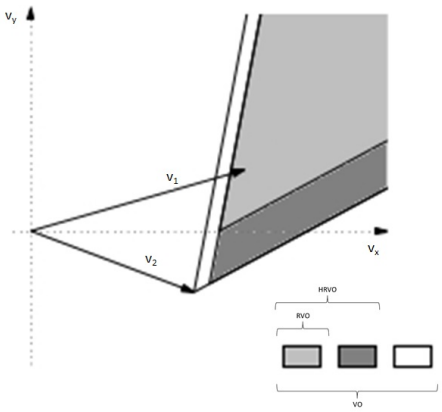


Figura 4.8: VOs criados pelas diferentes variantes do método VO

Para definir a direção das linhas que delimitam o VO recorre-se à equação 4.3.

$$\theta_{left} = -\theta_{right} = arcsin(\frac{r_A + r_B}{|p_{rel}|}) \tag{4.3}$$

Na qual r_A representa o raio do robot A, e p_{rel} é a posição relativa dos dois robots (posição de A subtraída da posição de B).

4.3.2 Obstáculos

Em ambientes com obstáculos dinâmicos o robot evita colisões da forma como já foi explicada em 4.3.1, é considerada a velocidade do robot e dos obstáculos e são criados conjuntos de velocidades que o robot tem de evitar.

O algoritmo original é o mais capaz de evitar obstáculos estáticos, visto que as suas variantes envolvem reciprocidade. Para obstáculos estáticos os vetores de velocidade do mesmo ficam na origem do espaço, resultando numa imobilização do robot quando rodeado por vários. Na existência de obstáculos com velocidade muito reduzida, por vezes, estes são considerados estáticos, pois os vértices do VO obtido para esse obstáculo estão próximos da origem no espaço das velocidades.

Por forma a evitar obstáculos estáticos e casos com velocidade reduzida recorre-se ao método da truncatura. Através deste certas velocidades são consideradas seguras num dado número de amostras temporais. Através deste método o VO é cortado resultando numa maior permissão. Os obstáculos são recalculados a uma frequência de 10 Hz correspondente também à frequência de atualização do mapa.

4.3.3 Escolha das velocidades

Após os obstáculos serem definidos, ou seja o conjunto de velocidades que evita colisões, é necessário escolher a melhor velocidade não pertencente a este conjunto.

De forma a escolher a velocidade adequada são geradas diferentes amostras que são avaliadas de acordo com certas propriedades como a distância à velocidade atual e ao VO mais próximo.

A cada uma dessas propriedades é dada um peso e a velocidade com melhor classificação resultante desses pesos é escolhida.

Para garantir que não são escolhidas velocidades dentro do VO é atribuído um custo muito alto às que se encontram dentro do mesmo. Fora desse grupo de velocidades, a velocidade a usar é escolhida de acordo com a sua distância à velocidade atual de forma a evitar grandes acelerações e desacelerações.

4.4 Resumo

Neste capítulo foram apresentados os métodos implementados no âmbito desta dissertação bem como a sua implementação.

O RRT é um método probabilístico que consiste em construir uma árvore desde a pose inicial à pose final através de amostragem. Implementou-se uma variante que tenta se aproximar da otimalidade, o RRT*.

A decomposição em células com heurísticas da família A* consiste em dividir o mapa em células livres ou ocupadas, neste caso de tamanho fixo e posteriormente procurar o caminho mínimo entre o ponto inicial e o destino através dessas células.

O VOs é um método de evitar obstáculos através da sua velocidade criando obstáculos no espaço de velocidades.

No capítulo seguinte são apresentados e comparados os resultados obtidos para cada um dos métodos em diferentes situações e cenários.

Capítulo 5

Resultados

Os algoritmos foram testados em simulação com recurso ao simulador *Stage* [37]. Foram testadas diversas situações para avaliar os métodos e verificar quais os que obtêm os melhores resultados nas mesmas. Para realizar estes testes recorreu-se a um computador com um processador Intel® Core™ i7-2630QM CPU @ 2.00 GHz e 8GB de RAM, com sistema operativo Ubuntu 14.04.2 LTS e distribuição *Indigo* do ROS.

Foram abordados ambientes com obstáculos estáticos, dinâmicos e com outros robots. Os obstáculos dinâmicos diferem dos robots pois não tentam evitar colisões, apenas seguem a sua trajetória.

As medidas usadas para avaliar e classificar os métodos foram o tempo de processamento, o tempo que demora a percorrer o caminho (representado por t) e a distância percorrida entre o ponto inicial e o destino (representado por d).

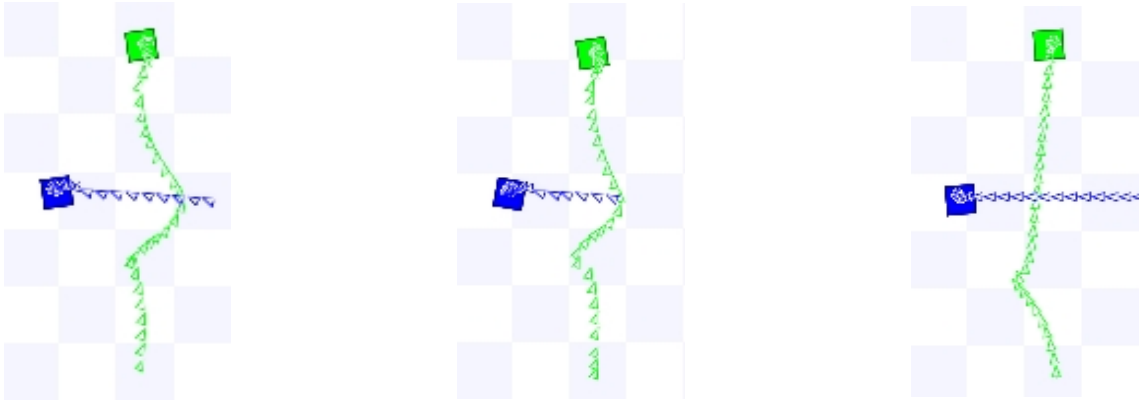
O tempo de processamento representa o tempo que demorou a definir a trajetória para o A* e para o RRT enquanto que para o VO representa a média dos tempos obtidos em cada ciclo para processar obstáculos e suas velocidades (representado por t_{proc}).

5.1 Cruzamento de robots ou obstáculos dinâmicos

Os resultados obtidos para o cruzamento de dois robots estão presentes nas figuras 5.1b, 5.1a, 5.1c e na tabela 5.1.

Tabela 5.1: Resultados cruzamento

	RRT	**A***	**VOs**
t_{proc} (ms)	1050.5	1.93	30.58
t (s)	(1) 13.0, (2) 21.78	(1) 11.0, (2) 18.6	(1) 8.4, (2) 16.4
d (m)	7.29	7.18	7.11

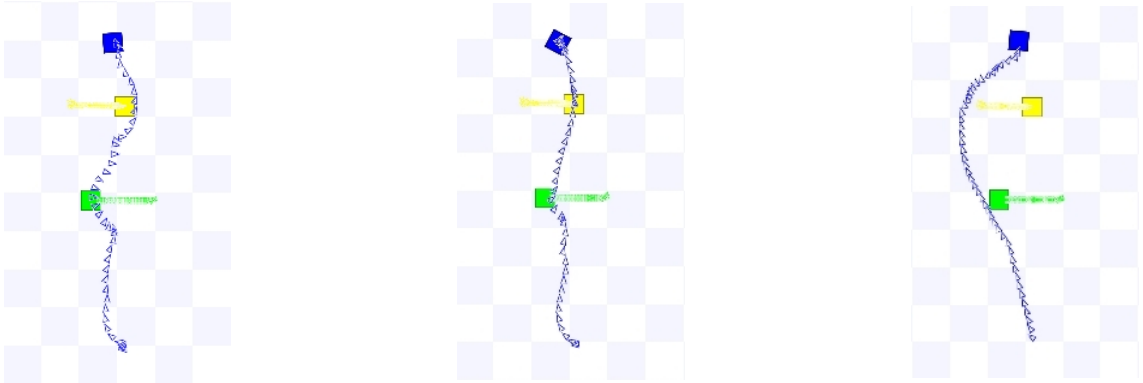


(a) RRT (b) A* (c) VO

Figura 5.1: Resultado: Cruzamento

Neste caso o método que obteve melhores resultados foi o VOs pois as velocidades foram consideradas de imediato.

Noutro caso semelhante mas com obstáculos dinâmicos, que não têm em conta os outros obstáculos, os resultados estão presentes nas seguintes imagens e tabela.



(a) RRT (b) A* (c) VO

Figura 5.2: Resultado: Situação de duplo cruzamento

Tabela 5.2: Resultados para cruzamento com dois obstáculos dinâmicos

	RRT	**A***	**VOs**
t_{proc} (ms)	1079.78	1.69	33.53
t (s)	23.999	21.59	24.80
d (m)	8.86	8.57	9.21

Neste caso o algoritmo A* foi o que obteve o melhor resultado. Noutro exemplo em que os obstáculos não têm direções perpendiculares à do robot os resultados foram semelhantes. Este exemplo é apresentado na tabela 5.3.

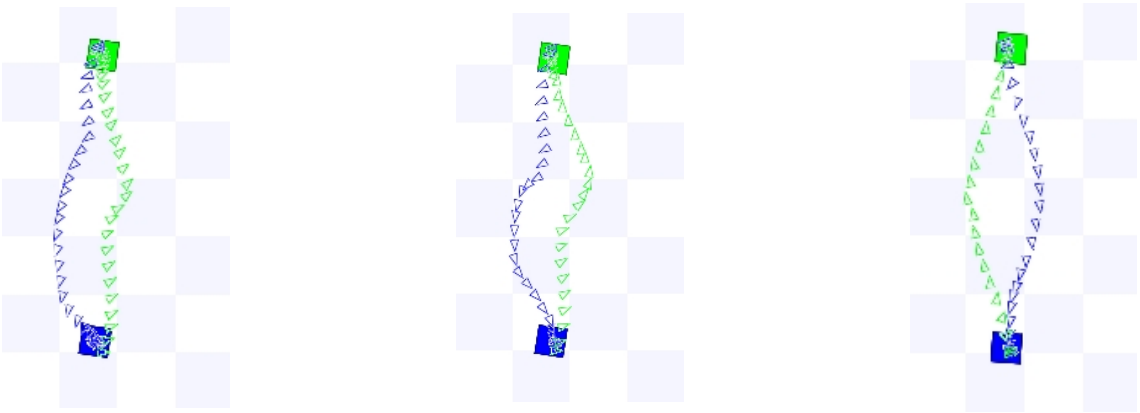
Tabela 5.3: Resultados para outro caso de cruzamento com obstáculos dinâmicos

	RRT	**A***	**VO**
t_{proc} (ms)	1087.13	2.19	28.6
t (s)	23.0	21.0	23.19
d(m)	8.80	8.5	9.08

5.2 Situação de Colisão frontal

Outro cenário testado foi um caso em que podia ocorrer colisão frontal com um obstáculo dinâmico. Para o testar foram usados dois casos, um representa o cruzamento entre dois robots e o outro entre um robot e um obstáculo dinâmico de velocidade reduzida.

Os resultados obtidos para o caso em que são considerados dois robots estão presentes na tabela 5.4 e são ilustrados na figura 5.3.



(a) RRT (b) A* (c) VO

Figura 5.3: Resultados: Situação de colisão frontal

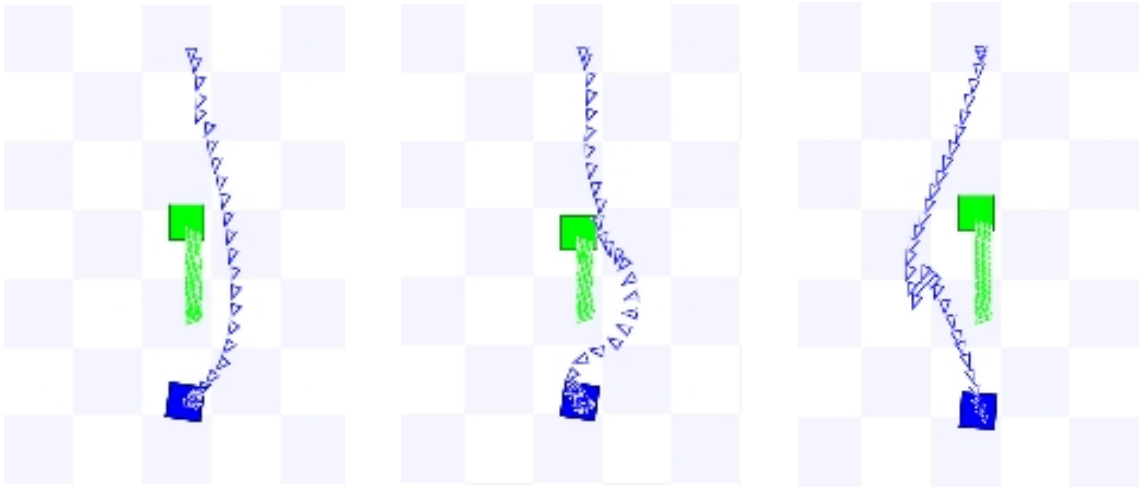
Tabela 5.4: Resultados para situação de colisão frontal

	RRT	**A***	**VOs**
t_{proc} (ms)	1050.5	0.353774	55.1915
t (s)	(1) 14.4, (2) 14.3	(1) 12.8 (2) 14.2	(1) 12 (2) 12.4
d (m)	5.59	5.48	5.42

Através da análise da figura 5.3 e da tabela 5.4 o método que revelou melhores resultados foi o VO pois ambos os robots tentam evitar a colisão desde o início, enquanto que nos outros casos é evitado mais próximo do local de colisão.

No caso em que há encontro entre um obstáculo dinâmico e um robot, ou seja o obstáculo não tenta evitar colisões, o A* foi o que revelou melhores resultados. Com o método VOs ocorreram oscilações por parte do robot que resultaram numa distância e num tempo bastante superior aos

dois outros métodos (tempo com A* - 16,47 segundos, tempo com VOs - 21,15 segundos). Os resultados deste exemplo estão presentes na tabela 5.5 e são ilustrados na figura 5.4.



(a) RRT (b) A* (c) VO

Figura 5.4: Situação de colisão frontal com obstáculos dinâmico

Tabela 5.5: Resultado: Situação de Colisão frontal com obstáculo dinâmico

	RRT	**A***	**VO**
t_{proc} (ms)	1084.08	0.58	28.52
t (s)	16.59	16.47	21.15
d(m)	5.53	5.43	12.68

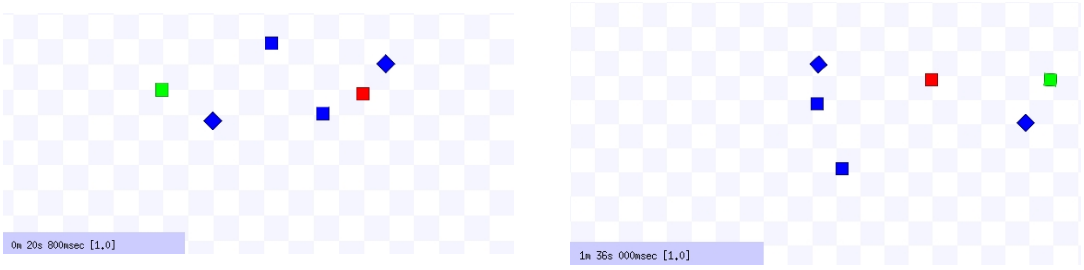
5.3 Múltiplos Obstáculos dinâmicos

Os três métodos foram testados em ambientes com vários obstáculos dinâmicos de forma a verificar qual se comporta melhor nos mesmos.

Para a situação apresentada onde as poses inicial e final estão representadas, respetivamente, nas figuras 5.5a e 5.5b, os resultados obtidos estão presentes na tabela 5.6 e no fotograma 5.6. Nestas figuras o robot é representado pela cor verde, os obstáculos dinâmicos pela cor azul e os estáticos pela cor vermelha.

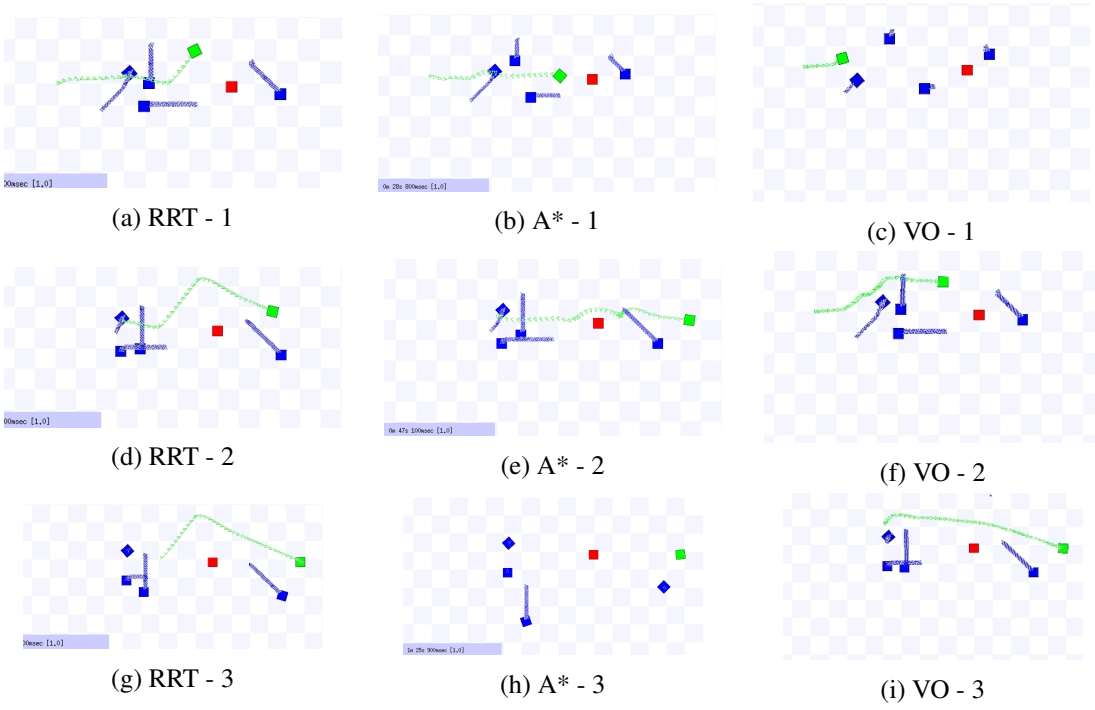
Tabela 5.6: Resultados: Múltiplos obstáculos dinâmicos

	RRT	**A***	**VO**
t_{proc} (ms)	3288.43	2.69	25.80
t (s)	39.77	33.83	35.32
d(m)	14.55	13.44	13.70



(a) Poses iniciais (b) Poses finais

Figura 5.5: Pose inicial e final do ambiente e do robot



(a) RRT - 1 (b) A* - 1 (c) VO - 1

(d) RRT - 2 (e) A* - 2 (f) VO - 2

(g) RRT - 3 (h) A* - 3 (i) VO - 3

Figura 5.6: Fotogramas com resultados para a situação apresentada

No exemplo apresentado na figura 5.6, o A* obteve o melhor desempenho segundo os critérios seguidos. O VOs apresentou o caminho com manobras menos bruscas, com uma trajetória mais suave, podendo representar um melhor desempenho em robots que tenham restrições de movimento, que não foi o caso abrangido neste projeto.

Os tempos de processamento apresentados para os métodos A* e RRT representam a soma do planeamento inicial e dos replaneamentos realizados ao encontrar um novo obstáculo.

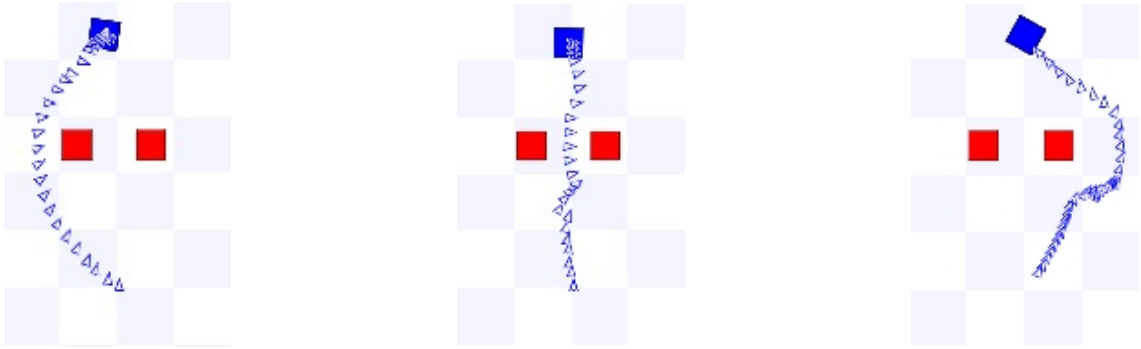
5.4 Obstáculos estáticos

Foram efetuados diferentes testes para verificar o desempenho de cada um dos métodos em ambientes com obstáculos estáticos.

Um dos testes foi a passagem de um robot por um caminho estreito.

Este teste tem como objetivo verificar se o robot consegue passar entre dois obstáculos estáticos cuja distância é um pouco maior que as dimensões do robot.

A trajetória obtida por cada método está presente nas figuras 5.7a, 5.7b, e 5.7c, e os resultados estão presentes na tabela 5.7.



(a) RRT (b) A* (c) VO

Figura 5.7: Resultados: Passagem estreito

Tabela 5.7: Resultados para passagem estreita

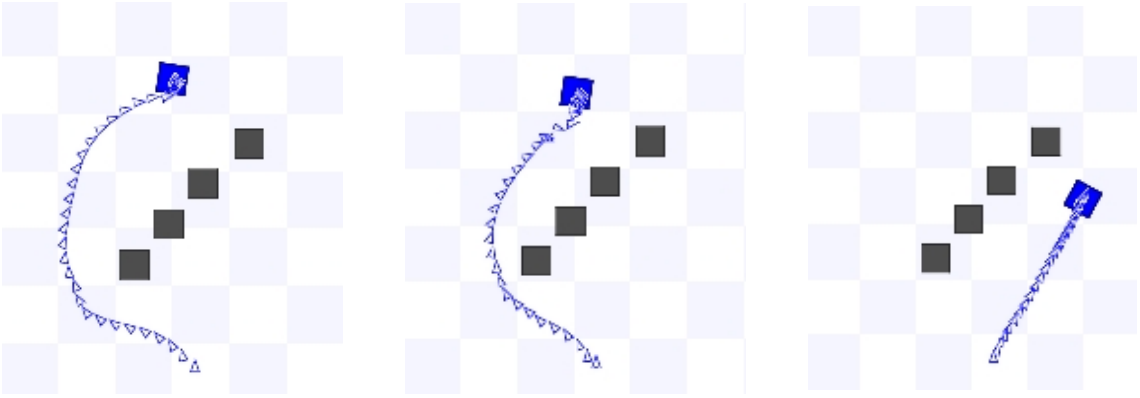
	RRT	**A***	**VOs**
t_{proc} (ms)	1093.6	0.287	38.28
t (s)	13.0	12.0	24.4
d (m)	5.65	4.425	6.05

Tal como é possível comprovar pelas figuras presentes em 5.7, o A* foi o único a definir a trajectória por entre os dois obstáculos enquanto que a trajetória resultante dos outros métodos contornou os obstáculos. O robot oscilou ao encontrar um obstáculo estático quando usado o método VOs. No caso do RRT, a distância percorrida e o tempo para a percorrer variaram pois este é um método probabilístico, os nós são resultados de amostragem no espaço.

Outro exemplo testado foi um caso em que os obstáculos estão dispostos de acordo com a figura 5.8 e com a tabela 5.8.

Tabela 5.8: Resultados: Caso de estudo

	RRT	**A***	**VO**
t_{proc} (ms)	1097.47	7.427	107.984
t (s)	17.607	16.0	-
d(m)	8.810	7.108	-



(a) RRT (b) A* (c) VO

Figura 5.8: Resultados: Caso de Estudo

Neste caso o A* obteve o melhor desempenho enquanto que o VO não conseguiu chegar ao destino sem o auxílio de um planeamento global. Isto porque vários obstáculos estáticos influenciam o robot, provocam um VO com vértices na origem do espaço de velocidades o que resulta na imobilização do robot. No caso do método RRT, os resultados variam, tendo sido ilustrado um bom caso.

5.5 Navegação num mapa estático

Para testar a navegação noutro tipo de ambientes recorreu-se a um mapa com obstáculos estáticos de forma a verificar qual o que melhor se comporta nesse tipo de mapa.

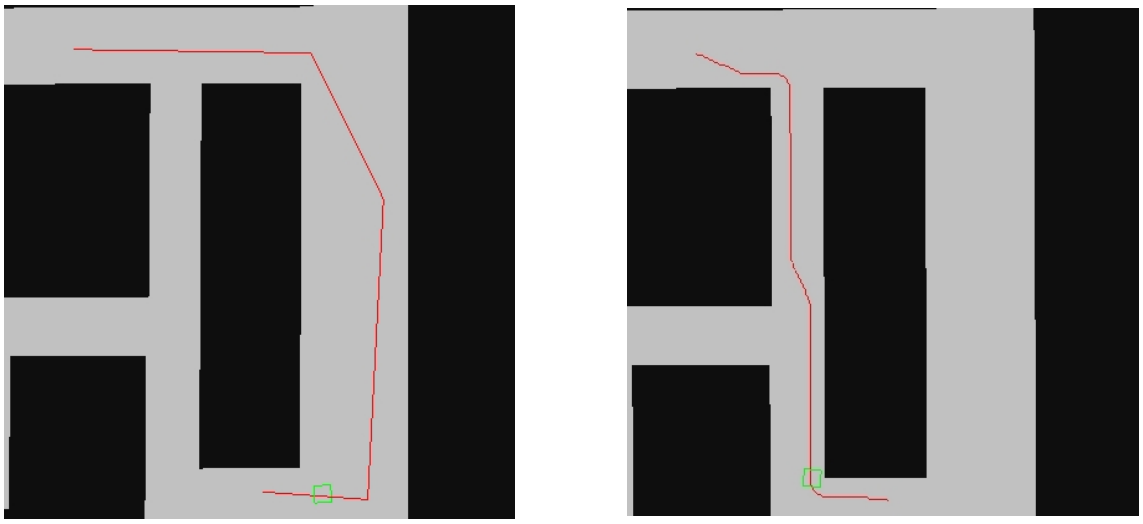
No caso presente nas figura 5.9 é ilustrado um dos pontos fracos do método RRT, visto a passagem que faz parte do caminho mínimo ser menor que a tomada pelo robot quando usado este método há uma menor probabilidade de serem amostrados nós nessa passagem (de caminho mínimo).

Tabela 5.9: Resultados para o caso apresentado na figura 5.9

	RRT	**A***
$\mathbf{t}_{proc}$ **(ms)**	1123.04	30.83
d(m)	34.62	25.52

Um dos critérios interessantes de analisar é o tempo de processamento do algoritmo A*, visto o mapa ser maior o tempo de processamento do mesmo aumentou consideravelmente, tal como se pode ver na tabela 5.9.

Nos casos com apenas um mapa estático, o VO não obteve resultados relevantes visto que é apenas um plano local, oscilando quando encontra obstáculos como paredes. De forma a este poder ser usado em mapas estáticos complexos é necessário combiná-lo com um plano global como



(a) RRT (b) A*

Figura 5.9: Resultados: Comparação entre o método RRT e A* num mapa

o RRT ou o A*. Comparando os outros dois algoritmos o A* foi o que revelou um melhor comportamento neste tipo de mapa. Uma das razões para este caso acontecer é a maior probabilidade de serem encontrados e escolhidos nós em espaços mais livres no caso do RRT. O A* considera este tipo de caminho se houverem células livres nesse corredor (considerando os obstáculos inflacionados).

5.6 Resumo e Conclusões

Neste capítulo foram apresentados resultados dos diferentes métodos para diversos casos tanto com obstáculos dinâmicos como estáticos.

Tal como foi possível observar pelos resultados apresentados, o VO apresenta os melhores resultados quando todos os robots estão preparados para evitar obstáculos, quando são recíprocos. Na existência de obstáculos dinâmicos que não evitam colisões tanto o A* como o VOs apresentam bons resultados, sendo o A* que revela melhores resultados, principalmente quando a velocidade dos obstáculos é relativamente baixa.

Para obstáculos estáticos e em ambientes com obstáculos estáticos e dinâmicos o método que revelou melhores resultados foi o A* pois o VOs não obtém bons resultados na existência de obstáculos estáticos, resultando em oscilações do robot e consequentemente num maior tempo para alcançar o destino.

O método RRT obteve, por vezes, bons resultados dependendo dos nós gerados, apesar de os resultados variarem. De forma a obter melhores resultados em termos de distância percorrida e tempo para a percorrer gerou-se vários nós que resultou num tempo de processamento bastante maior que nos dois outros métodos. Este tem uma zona de segurança maior que nos dois outros casos devido a ser probabilístico, a probabilidade de criar nós perto de obstáculos é menor que

em zonas bastante afastadas de obstáculos. Apesar disso os resultados obtidos para a distância e tempo para a percorrer foram satisfatórios.

No capítulo seguinte são apresentadas as conclusões bem como o possível trabalho futuro a realizar.

Capítulo 6

Conclusões e Trabalho Futuro

O objetivo principal desta dissertação foi implementar e comparar diferentes algoritmos de planeamento de trajetória de robôs móveis, mais especificamente métodos de aproximação probabilística, métodos de decomposição por células com heurística A* e o método Velocity Obstacles.

Para alcançar este objetivo, começou-se por fazer um estudo dos vários métodos e correspondentes fases necessárias para o planeamento de trajetórias. Foram analisados as diferentes formas de construir o mapa (como por exemplo decomposição por células) bem como algoritmos de pesquisa, para obter um caminho quando o mapa está dividido em células.

Deu-se ênfase no estudo dos três métodos a implementar, o RRT, um método de aproximação probabilística, a decomposição em células aproximadas com heurística da família A* e o VOs.

Para os implementar recorreu-se à ferramenta ROS e a um simulador (*Stage*) de forma a testar a sua navegação bem como os desempenhos de cada um dos métodos. Apresentou-se também a abordagem seguida, recorrendo ao ROS para a implementar.

Foram testadas diversas situações, como por exemplo situações de possível colisão, quer com obstáculos dinâmicos como estáticos de forma a verificar qual dos métodos revelou melhores resultados em cada uma dessas situações.

Os três métodos, bem como as suas variantes implementadas, foram descritos e comparados de acordo com a distância percorrida, o tempo que a demorou a percorrer e o tempo de processamento.

Analisou-se os resultados de cada método de forma a verificar as vantagens e as desvantagens de cada um.

Foram implementadas diferentes variantes dos métodos também como forma de testar com a melhor variante para cada caso.

Falando dos resultados em si foi possível concluir que o A* revelou melhores resultados em ambientes com obstáculos estáticos onde encontrou o caminho mínimo e também em ambientes com obstáculos estáticos e dinâmicos.

Tanto o A* como o VO obtiveram bons resultados em situações com obstáculos dinâmicos sendo que o A* apresentou os melhores resultados quando os obstáculos não evitam colisões e quando a velocidade destes é reduzida.

O VO apresentou os melhores resultados quando todos os robots tentam evitar obstáculos, quando são recíprocos e obteve bons resultados na presença de obstáculos dinâmicos. Na presença de obstáculos estáticos com alguma complexidade obteve os piores resultados,nestes casos este método resulta em oscilações por parte do robot devido a ser um plano local podendo não alcançar o destino.

O método RRT obteve resultados satisfatórios, apesar de geralmente pior e mais variáveis que nos outros dois casos. Tal como foi apresentado num dos resultados (mapa estático), um dos seus pontos fracos é seguir caminhos em espaços mais amplos e não o caminho mínimo caso este inclua situações como passagens estreitas.

Em termos de tempo de processamento o mais rápido é o A* cujo tempo aumenta com o número de células exploradas. No caso do RRT o tempo de processamento é elevado devido ao número de passos permitido e ao tamanho dos mesmos, bem como ao facto de ser usado um critério de otimização de forma a tentar minimizar o caminho.

6.1 Satisfação dos Objectivos

Os objetivos principais desta dissertação podem ser considerado cumpridos pois foram implementados, testados e comparados.

Foram implementados com recurso à linguagem C++, à *framework* ROS e ao simulador *Stage* e comparados em diferentes cenários.

Apesar de os principais objetivos serem considerados cumpridos é possível existem melhorias e possível trabalho futuro para um maior desenvolvimento deste tema.

De seguida são apresentado possíveis rumos para um trabalho futuro.

6.2 Trabalho Futuro

Como possíveis melhorias e trabalho futuro para dar continuidade ao realizado no âmbito desta tese podem ser consideradas as seguintes abordagens:

- Validar os métodos em diferentes plataformas robóticas reais.
- Testar os algoritmos em ambientes 3D de forma a compará-los no mesmo.
- Desenvolver variantes e combinar cada um dos métodos de forma a colmatar os pontos fracos de cada um com os pontos fortes do outro, por exemplo combinar RRT e VO de forma a permitir ao VOs melhores resultados para obstáculos estáticos.

Bibliografia

[1] S. Thomas, G. Song, and N. M. Amato, "Protein folding by motion planning," *Physical biology*, vol. 2, no. 4, p. S148, 2005.

[2] R. Silveira, L. Fischer, J. A. S. Ferreira, E. Prestes, and L. Nedel, "Path-planning for rts games based on potential fields," in *Motion in Games*. Springer, 2010, pp. 410–421.

[3] M. H. Overmars, "Path planning for games," in *Proc. 3rd Int. Game Design and Technology Workshop*, 2005, pp. 29–33.

[4] N. Noguchi and H. Terao, "Path planning of an agricultural mobile robot by neural network and genetic algorithm," *Computers and electronics in agriculture*, vol. 18, no. 2, pp. 187–204, 1997.

[5] Y. K. Hwang and N. Ahuja, "Gross motion planning—a survey," *ACM Computing Surveys (CSUR)*, vol. 24, no. 3, pp. 219–291, 1992.

[6] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Bradford Company, 2004.

[7] H. M. Choset, "Principles of robot motion: theory, algorithms, and implementation," 2005.

[8] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[9] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2. IEEE, 1999, pp. 1024–1031.

[10] R. Bohlin and E. Kavraki, "Path planning using lazy prm," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 521–528.

[11] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[12] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects," 2000.

[13] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3. IEEE, 2002, pp. 2383–2388.

[14] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 1243–1248.

[15] S. M. LaValle, "Rrt." [Online]. Available: http://msl.cs.uiuc.edu/rrt/point1.jpg [Accessed: 2015-02-14]

[16] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[17] O. Montiel, R. Sepúlveda, and U. Orozco-Rosas, "Optimal path planning generation for mobile robots using parallel evolutionary artificial potential field," *Journal of Intelligent & Robotic Systems*, pp. 1–21, 2014.

[18] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 1, pp. 23–32, 1992.

[19] V. J. Lumelsky and A. A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, vol. 2, no. 1-4, pp. 403–430, 1987.

[20] J. Antich, A. Ortiz, and J. Minguez, "Bug2+: Details and formal proofs," Technical Report A-1, University of the Balearic Islands, Tech. Rep., 2009.

[21] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[22] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *Robotics, IEEE Transactions on*, vol. 27, no. 4, pp. 696–706, 2011.

[23] J. Snape, S. J. Guy, D. Vembar, A. Lake, M. C. Lin, and D. Manocha, "Reciprocal collision avoidance and navigation for video games," in *Game Developers Conf., San Francisco*, 2012, Conference Proceedings.

[24] R. Glasius, A. Komoda, and S. C. Gielen, "Neural network dynamics for path planning and obstacle avoidance," *Neural Networks*, vol. 8, no. 1, pp. 125–133, 1995.

[25] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and H.-M. Tai, "Autonomous local path planning for a mobile robot using a genetic algorithm," in *Evolutionary Computation. CEC2004. Congress on*, vol. 2. IEEE, 2004, Conference Proceedings, pp. 1338–1345.

[26] H. R. Beom and K. S. Cho, "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 25, no. 3, pp. 464–477, 1995.

[27] C.-C. Wong, M.-F. Chou, C.-P. Hwang, C.-H. Tsai, and S.-R. Shyu, "A method for obstacle avoidance and shooting action of the robot soccer," in *ICRA*, 2001, Conference Proceedings, pp. 3778–3782.

[28] M. Brand, M. Masuda, N. Wehner, and X.-H. Yu, "Ant colony optimization algorithm for robot path planning," in *Computer Design and Applications (ICCDA), International Conference on*, vol. 3. IEEE, 2010, Conference Proceedings, pp. V3–436–V3–440.

[29] P. L. C. G. d. Costa, "Planeamento cooperativo de tarefas e trajectórias em múltiplos robôs," Ph.D. dissertation, Faculdade de Engenharia da Universidade do Porto, 2012.

[30] A. P. Moreira, P. J. Costa, and P. Costa, "Real-time path planning using a modified a* algorithm," 2009.

[31] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE, 1994, pp. 3310–3317.

[32] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *International Conference on Robotics and Automation*, 2010.

[33] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[34] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[35] J. Snape, J. Van den Berg, S. J. Guy, and D. Manocha, "Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 5917–5922.

[36] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen, "Collision avoidance under bounded localization uncertainty," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 1192–1198.

[37] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.