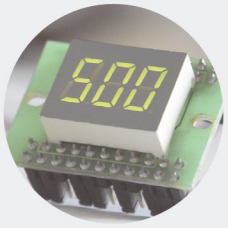
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Planeamento de Trajetória para Operações de Busca e Salvamento com UAVs

TIAGO SANTOS FERNANDES
março de 2016

POLITÉCNICO
DO PORTO

Planeamento de Trajetória para Operações de Busca e Salvamento com UAVs

Tiago Santos Fernandes
Nº 1100500

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Sistemas Autónomos
Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2016

Dissertação, para satisfação parcial dos requisitos do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Tiago Santos Fernandes

Nº 1100500

Orientador: André Miguel Pinheiro Dias

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização de Sistemas Autónomos

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

9 de Março de 2016

Agradecimentos

A realização desta dissertação contou com o apoio e incentivo das pessoas que se encontram em meu redor.

Gostaria de em primeiro lugar agradecer ao meu orientador, Engº André Dias por me ter proporcionado esta oportunidade e pelo suporte e ajuda inigualável que me forneceu ao longo deste percurso.

Quero também agradecer a todos os meus amigos e colegas de curso no laboratório de sistemas autónomos (LSA) por todo o apoio e ajuda prestada.

Por último gostava de agradecer à minha família em especial aos meus pais pela educação que me proporcionaram e pelos esforços que realizaram durante a minha caminhada académica.

Esta página foi intencionalmente deixada em branco.

Resumo

Os sistemas autónomos trazem como mais valia aos cenários de busca e salvamento a possibilidade de minimizar a presença de Humanos em situações de perigo e a capacidade de aceder a locais de difícil acesso.

Na dissertação propõe-se endereçar novos métodos para perceção e navegação de veículos aéreos não tripulados (UAV), tendo como foco principal o planeamento de trajetórias e deteção de obstáculos. No que respeita à perceção foi desenvolvido um método para gerar *clusters* tendo por base os *voxels* gerados pelo Octomap. Na área de navegação, foram desenvolvidos dois novos métodos de planeamento de trajetórias, GPRM (Grid Probabilistic Roadmap) e PPRM (Particle Probabilistic Roadmap), que tem como método base para o seu desenvolvimento o PRM. O primeiro método desenvolvido, GPRM, espalha as partículas numa *grid* pré-definida, construindo posteriormente o *roadmap* na área determinada pela *grid* e com isto estima o trajeto mais curto até ao ponto destino. O segundo método desenvolvido, PPRM, espalha as partículas pelo cenário de aplicação, gera o *roadmap* considerando o mapa total e atribui uma probabilidade que irá permitir definir a trajetória otimizada.

Para analisar a performance de cada método em comparação com o PRM, efetua-se a sua avaliação em três cenários distintos com recurso ao simulador MORSE.

Palavras-Chave: Planeamento de Trajetórias, Deteção Obstáculos, UAV, *Clustering*, Octomap, GPRM, PPRM

Esta página foi intencionalmente deixada em branco.

Abstract

In the last years, autonomous vehicles have contribute to search and rescue scenarios by allowing to minimize the presence of Humans in dangerous situations and also in the capability to support operations in unstructured environments.

The present document propose to address new methods in the area of perception and navigation to Unmanned Aerial Vehicles (UAV), having as main focus the path planning and obstacle detection. As regards to perception was developed a new method to generate clusters based on the voxels provided by Octomap. In the navigation area, were developed two new methods for path planning, GPRM (Grid Probabilistic Roadmap) and PPRM (Particle Probabilistic Roadmap), that arise from the PRM method. The first one, GPRM propose a method that will spread particles in a pre-defined grid in order to be able to estimate a roadmap with the shortest path to the target position. The second one, denote by PPRM, propose a technique to spread randomly particles by the scenario and assigns a weight for each one, based on probability of collision with obstacles, in order to define the optimized path.

To evaluate the performance of each developed method we perform a benchmark related to the well know path planning PRM in three different challenger scenarios through the MORSE simulator.

Keywords: Path Planning, Obstacle Detection, UAV, Clustering, Octomap, GPRM, PPRM

Esta página foi intencionalmente deixada em branco.

Conteúdo

Esta página foi intencionalmente deixada em branco.

Lista de Figuras

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Lista de Siglas e Acrónimos

AUV Autonomous Underwater Vehicle

BFS Breadth-first search

BVS Boundary Value Solver

EKF Extended Kalman Filter

EKF SLAM Extended Kalman Filter for Simultaneous Localization and Mapping

FOV Field of View

GNSS Global Navigation Satellite System

GPRM Grid Probabilistic Roadmap

GPS Global Positioning System

GUI Graphic User Interface

HRI Human-Robot Interaction

ICP Iterative Closest Point

IEKF Iterative Extended Kalman Filter

IMU Inertial Measurement Unit

ISEP Instituto Superior de Engenharia do Porto

KF Kalman Filter

LKH Lin-Kernighan-Helsgaun Heuristic

LSA Laboratório de Sistemas Autónomos

ODE Open Dynamics Engine

PID Proportional - Integral - Derivative

PPRM Particle Probabilistic Roadmap

PRM Probabilistic Roadmap

PTAM Parallel Tracking and Mapping

ROS Robot Operating System

RRT Rapidly-exploring Random Trees

SLAM Simultaneous Localization and Mapping

UAV Unmanned Aerial Vehicle

UGV Unmanned Ground Vehicle

VISTA Visual Threat Awareness

V-REP Virtual Robot Experimentation Plataform

VSLAM Visual Simultaneous Localisation and Mapping

Capítulo 1

Introdução

Nos últimos anos, a robótica tem emergido como uma área de investigação de enorme importância para o nosso quotidiano em áreas como a Indústria, Transporte, Medicina, Aplicações Militares, entre outras.

Uma das áreas de investigação emergentes, prende-se com a utilização de veículos autónomos em operações de busca de salvamento [1], quer seja em cenário terrestres (com *Unmanned Ground Vehicle* (UGVs)), aéreos (com *Unmanned Aerial Vehicles* (UAVs)) ou aquáticos (com *Autonomous Underwater Vehicles* (AUVs)). Em todos estes cenários, a capacidade operacional de um veículo autónomo para executar este tipo de operações requer capacidade de perceção a bordo e de planeamento da trajetória. Os veículos autónomos podem também ser aplicados em cenários de vigilância [2], desastres [1] e inspeção[3], sendo que em muitos destes cenários o acesso encontra-se limitado somente a veículos aéreos.

Na dissertação propõe-se endereçar o problema de planeamento de trajetória em cenários de busca e salvamento com UAVs através do desenvolvimento de estratégias de planeamento trajetórias que permitam evitar colisões num trajeto de um ponto A para o ponto B, sendo necessário mapear o ambiente e planear a trajetória que o veículo necessita de seguir. Na figura 1.1 encontra-se representado um possível ambiente onde o *UAV* será utilizado, podendo-se verificar que o veículo para efetuar a navegação no ambiente de desastre demonstrado necessita de evitar os obstáculos que se encontram no seu interior e com isso planear a sua trajetória.

Atualmente a comunidade científica propõe algumas soluções de forma a ser possível resolver os problemas gerados pelo planeamento de trajetórias nos cenários já descritos. Na área da perceção de obstáculos, necessária para proceder-se ao planeamento de trajetórias, uma das soluções é a apresentada por Nathan Michael em [1], que utiliza o

método de *3D Iterative Closest Point* (*ICP*) e o método *SLAM* (*Simultaneous Localization and Mapping*) para gerar o mapa 3D do ambiente do cenário de busca e salvamento e desastres. Já no planeamento de trajetórias e para os cenários de vigilância é sugerido por Jose Acevedo, [2], uma abordagem que utiliza múltiplos veículos aéreos em interação entre si. No caso dos cenários de inspeção e como é verificado em [3], é sugerido o uso de uma implementação rápida do algoritmo *Lin-Kernighan-Helsgaun Heuristic* (LKH) e do *Boundary Value Solver* (BVS) para determinar a trajetórias para inspeção.

Neste documento irá ser endereçado o uso de UAVs em operações de busca e salvamento devido à sua elevada mobilidade e acessibilidade que possui neste tipo de cenários. Sendo necessário também ter em consideração o *payload* de sensores que este tipo de veículos suporta e o facto de existir ainda muito por onde investigar neste tipo de aplicação.

Figura 1.1: Ambiente de aplicação na competição *euRathlon 2015* [4].

1.1 Motivação

No Laboratório de Sistemas Autónomos (LSA), do Instituto Superior de Engenharia do Porto (ISEP), são desenvolvidos sistemas autónomos para operar em diferentes ambientes e aplicações como monitorização, segurança, busca e salvamento, entre outros. Desta forma são desenvolvidos e analisados sistemas sensoriais que permitam obter informação que será necessária para a perceção, navegação e controlo dos diversos sistemas.

O trabalho desenvolvido enquadra-se na área de veículos autónomos aéreos onde o LSA encontra-se a desenvolver um veículo aéreo autónomo com capacidade de efetuar inspeções, busca e salvamento e vigilância, sendo desta forma necessário desenvolver novos métodos e sistemas que permitam obter uma melhor perceção do ambiente em torno do veículo, bem como o processo de planeamento de trajetória para navegação.

1.2 Objetivos

O tema da dissertação aborda o problema de planeamento de trajetória de um UAV num cenário desestruturado como é o caso do cenário de aplicação de busca e salvamento apresentado na figura 1.1.

O objetivo principal deste trabalho consiste em estudar o estado da arte de algoritmos para planeamento de trajetórias e explorar a sua aplicabilidade em tempo real em cenários desestruturados.

Para a sua concretização , existem desafios ao nível da perceção e ao nível do planeamento da trajetória pelo que os objetivos para a sua concretização passam por:

- Análise de métodos de perceção e navegação que permita endereçar o cenário de busca e salvamento;
- Desenvolvimento de um método de *Clustering* que permita a definição em tempo-real da posição dos obstáculos com recurso a laser;
- Desenvolvimento de métodos de planeamento de trajetória capaz de endereçar o cenário de busca e salvamento com veículos aéreos.
- Efetuar a validação dos métodos desenvolvidos e comparar com as soluções identificadas no estado de arte.

1.3 Estrutura

Esta dissertação encontra-se organizada em sete capítulos. No Capítulo 1 é apresentada uma pequena introdução onde é realizado o enquadramento do tema da dissertação e detalhado as motivações e objetivos da mesma.

No Capítulo 2 realiza-se um estudo das abordagens, métodos e processos já desenvolvidos pela comunidade científica sobre perceção de obstáculos e planeamento de trajetórias, para veículos aéreos.

No Capítulo 3, irão ser abordados os fundamentos teóricos que servirão de suporte aos objetivos definidos na dissertação, tendo como base os métodos de planeamento de trajetórias analisados no capítulo do estado da arte. Será também efetuada uma pequena descrição dos possíveis simuladores a utilizar.

De seguida, no Capítulo 4, é referida a arquitetura geral do projeto que servirá de base para o sistema a desenvolver, bem como uma breve explicação de cada uma das camadas constituintes.

No capítulo seguinte, Capítulo 5, é descrito o algoritmo desenvolvido para deteção de obstáculos, sendo também apresentado os resultados obtidos.

O Capítulo 6, apresenta uma análise de um dos métodos do estado da arte e de dois métodos desenvolvidos para aplicações em tempo real, sendo apresentados os respetivos resultados e comparações que permitam uma melhor observação.

Por último é divulgado algumas conclusões, Capítulo 7, sobre os diversos métodos analisados e respetivos resultados, bem como o trabalho futuro a realizar.

Capítulo 2

Estado da Arte

Neste capítulo é endereçado métodos e algoritmos que se enquadrem em cenários de busca e salvamento com recurso a veículos autónomos aéreos.

O facto do cenário de aplicação ser busca e salvamento implica que o veículo seja capaz de percecionar o ambiente que o rodeia de forma a poder planear manobras e navegar de modo autónomo evitando os obstáculos. No decorrer deste capítulo iremos endereçar o problema de navegação e de perceção tendo por base a sua aplicabilidade. Como estes métodos serão aplicados num UAV será necessário ter em consideração o custo computacional de cada método que deve ser reduzido e que permita uma rápida resolução do problema.

2.1 Planeamento de Trajetórias

Nesta secção irão ser abordados alguns métodos de navegação aplicados a veículos aéreos e que permitam que estes naveguem de forma autónoma em ambientes 3D. Nestes ambientes o objetivo dos algoritmos de planeamento de trajetórias não é só encontrar o trajeto livre de colisão mas também minimizar a distância a percorrer e a energia consumida pelo veículo. Neste documento considera-se que os métodos de planeamento de trajetórias encontram-se divididos em cinco categorias, tal como é apresentado por Yang em [5], *sampling-based algorithms*, *node-based algorithms*, *mathematical model based algorithms*, *Bio-inspired algorithms* e *multi-fusion based algorithms*. Sendo o fator divisório as características que cada método possui.

2.1.1 *Sampling-based* Métodos

Os algoritmos *Sampling-based* necessitam de possuir um conhecimento prévio da informação do ambiente onde vão ser aplicados os robôs. Outra característica, é o facto de normalmente ser efetuado uma amostra do ambiente com um conjunto de *nodes*, sendo que de seguida pode efetuar uma amostragem do ambiente o ambiente ou então procurar de uma forma aleatória um trajeto ótimo.

Mesmo os algoritmos que possuam estas características podem ser divididos em duas sub-categorias, passivos e ativos. O algoritmos passivos são aqueles que conseguem gerar um *road net map* desde o *node* de inicio até ao *node target* mas como existe um conjunto de possíveis trajetos necessita de algoritmos de procura para determinar o menor trajeto ou o trajeto ótimo, ou seja, todos os algoritmos que não conseguem por si só encontrar um único trajeto são determinados como passivos. Os algoritmos ativos conseguem gerar um esqueleto até ao *target*, utilizando apenas o seu próprio procedimento de processamento.

Nos algoritmos passivos encontram-se métodos como o Probabilistic Roadmap (PRM) [6], K-PRM, S-PRM, 3D Voronoi [7], Rapidly-exploring Random Graph [8], Visibility Graphs, Corridor Map entre outros. Já nos algoritmos ativos encontram-se os métodos Rapidly-exploring Random Trees (RRT) [9], Dynamic Domain RRT (DDRRT), RRT-Star (RRT*), Artificial Potential Field [10] entre outros.

Os algoritmos *Sampling-based* são de fácil implementação e possuem estruturas simples, sendo apropriados para condições de planeamento estáticas e dinâmicas, permitindo ser implementado em tempo real.

O método sugerido por Kavraki em [6], PRM, é constituído por duas fases *learning phase* e *query phase*. Na primeira fase, *learning phase*, constrói-se um roadmap com os trajetos livres de colisão, e na segunda fase, *query phase*, determina-se o trajeto desde o ponto inicial até ao final.

Shen em [9] sugere o uso do método RRT para efetuar planeamento de trajetórias. Este método constrói uma *tree* de forma incremental usando as amostras espalhadas pelo espaço, sendo que cada ligação possível entre duas amostras serão um novo ramo da *tree*.

Outra abordagem é a sugerida por Cho em [7], onde é utilizado o método 3D Voronoi, que é uma variação da estrutura de dados de *edge* radial, capaz de lidar com as características topológicas do diagrama Voronoi Euclidiano para esferas.

2.1.2 *Node-based* Métodos

Os algoritmos *Node-based*, como o próprio nome indica, geram trajetos com base num conjunto de *nodes* e tal como os *Sampling-Based Algorithms* partilha a mesma propriedade de procurar num conjunto de *nodes* num gráfico ou mapa onde a pré-informação sensorial e processos de processamento já se encontram executados.

Nestes algoritmos é possível encontrar-se algoritmos como o algoritmo de Dijkstra [11], A*, Lifelong Planning A* (LPA), Theta* [12], Lazy Theta*, Dynamic A* (D*), D* Lite [13], Harmony Search entre outros métodos.

Este tipo de algoritmos podem ser combinados com outros para alcançar uma melhoria global, podendo ser utilizados em aplicações em tempo real.

O algoritmo de Dijkstra utilizado por Musliman em [11], permite determinar o caminho mais curto de um determinado grafo. Após se determinar qual o estado do grafo que será considerado como ponto inicial esta abordagem determina o menor trajeto para todos os estados constituinte do grafo. Assim basta saber qual será o estado final para conseguir-se conhecer o menor trajeto até ele.

Outra abordagem é o Theta*, que foi sugerido por Filippis em [12], conseguindo refinar a procura no grafo permitindo obter trajetos com qualquer atitude. Este método pode ser aplicado a ambientes 3D e foi baseado no método A*.

Um abordagem muito utilizada é a apresentada por Grzonka em [13] onde refere o método D* Lite, que permite determinar o trajeto da atual posição do robô até à posição que pretende chegar. Na situação apresentada em [13] o algoritmo é utilizado para calcular a trajetória no espaço X-Y para cada camada do mapa de múltiplos níveis, sendo posteriormente gerado uma trajetória 2,5D com a inclusão da componente *yaw*.

2.1.3 *Mathematical model* Métodos

Estes algoritmos permitem efetuar um modelo do ambiente como do corpo, considerando as limitações cinemáticas e dinâmicas sendo posteriormente calculada a função de custo tendo em consideração todas as desigualdades ou equações para determinar uma solução ótima, sendo apropriado para uso em modo offline devido ao custo computacional que possuem. Os *Mathematical model based algorithms* podem ser divididos em duas sub-categorias, *Linear Programming* e *Optimal Control*, onde o *Linear Programming* contêm os métodos Mixed-Integer Linear Programming (MILP), Binary Linear Programming, Nonlinear Programming, EKF, etc.

O método EKF pode ser considerado um *Mathematical model based algorithm* pois como se pode verificar em [14], Huh utiliza o método em conjunto com o método SLAM

para navegação do robô baseada no conjunto de sensores câmara e laser, em cenários *indoor*. Este algoritmo numa primeira fase combina a informação dos sensores resultando uma nuvem de pontos 3D das *landmarks* e de seguida estas são adicionadas como estados do EKF, onde o objetivo deste filtro é estimar os estados do veículo e das *landmarks*. Para a previsão do filtro EKF necessita-se de identificar que este é um modelo dinâmico, ou seja modelo cinemático não linear, do veículo. Ainda segundo esta abordagem a matriz de covariância é atualizada no passo de previsão do filtro EKF e os estados desta matriz é composto pelas seguintes secções: robô para robô, robô para *landmark*, *landmark* para robô e *landmark* para *landmark*. Para permitir uma melhor gestão de marcas só são consideradas as *landmarks* que se encontrem no *field of view* (FOV) da câmara. Para o modelo de correção do EKF todos os dados medidos através do laser e da câmara permitem corrigir os estados e covariância do veículo e *landmarks*.

2.1.4 *Bio-inspired* Métodos

Os Bio-inspired algorithms foram desenvolvidos para imitar o comportamento biológico de forma a lidar com os problemas. Estes métodos deixam de fora o processo de construção de modelos de ambientes complexos e propõe um forte método de procura para convergir para o objetivo de forma estável, podendo ser divididos em duas sub categorias, Evolutionary Algorithm (EA) e Neural Networks (NN). Na primeira sub categoria é possível encontrar-se algoritmos como *genetic algorithm*, *memetic algorithm* [15], *particle swarm optimization*, *ant colony optimization* e *shuffled frog leaping algorithm*. Estes algoritmos começam por selecionar de forma aleatória soluções viáveis como primeira geração. De seguida toma em consideração o ambiente, capacidade do robô, objetivo, e outras restrições que o planeador avalia individualmente. Posteriormente é escolhido um conjunto de individuais como parentes da próxima geração de acordo com a sua aptidão. Por último é a etapa de mutação e passagem, sendo que o processo termina quando atinge um valor pré-definido. A sub categoria Neural Network tem como objetivo gerar uma paisagem dinâmica para atividades neurais, bem como o método Potencial Field.

Tal como os algoritmos *Mathematical model based*, os *Bio-inspired* requerem um tempo computacional por iteração demasiadamente elevado, sendo que só podem funcionar em modo offline.

Um destes métodos, e como já referido é *Memetic algorithm* apresentado por Shahidi em [15]. Esta abordagem permite encontrar o trajeto ótimo livre de colisão de um robô usando um reduzido número de população e recorrendo a poucas gerações.

2.1.5 *Multi-fusion based* Métodos

Os algoritmos *Multi-fusion based* são utilizados em planeamento de trajetórias de ambientes 3D e são constituídos por múltiplos algoritmos simples com o objetivo de obter uma trajetória ótima. Estes manejam com problemas que um algoritmo simples não consegue alcançar individualmente um resultado ótimo. Tal como nas outras secções os *Multi-fusion based algorithms* podem ser divididos em duas categorias, *Integration of Algorithms* e *Algorithms ranking*. O primeiro, *Integration of Algorithms*, representa os algoritmos que são formados por integração de vários algoritmos de planeamento de trajetórias de forma a trabalharem em conjunto para encontrarem um trajeto ótimo. A segunda categoria, *Algorithms ranking*, o conjunto de métodos que constituem os algoritmos *Multi-fusion based algorithms* funcionam normalmente uns de seguida dos outros, ou seja, quando um termina a sua parte outro começa de seguida e assim sucessivamente.

Na tabela 2.1 e de acordo com [5] encontram-se representados os diversos métodos de planeamento de trajetórias, divididos pelas diversas secções e como os respetivos tempos e informação sobre os ambiente a aplicar. Nesta tabela E corresponde a Estático e D a Dinâmico.

Tabela 2.1: Propriedades de cada tipo de método [5].

Método	Elementos do método	Tempo de Complexidade	Tipo de Ambiente	Tempo Real
Sampling Based	Voronoi, RRT, PRM, Vor K-PRM, S-PRM, Visibility Graphs, Corridor Map, DDRRT, RRT*	$0(n\ log\ n) \leq T \leq 0(n^2)$	S e D	Sim
Node Based	Dijkstra's Algorithms, A*, D*, LPA, Theta*, Lazy Theta*, D*-Lite, Harmony Search	$0(m\ log\ n) \leq T \leq 0(n^2)$	S e D	Sim
Mathematic Model Based	Optimal Control, Mixed-Integer Linear Programming, Binary Linear Programming, Non-linear Programming, EKF	Depende da equação polinomial	S e D	Não
Bio-inspired	NN, genetic algorithm, memetic algorithm, particle swarm optimization, ant colony optimization, shuffled frog leaping algorithm	$T \geq 0(n^2)$	S	Não
Multi-fusion Based	VVP, PRM Node based optimal algorithms, GIS-MCDA algorithms, visibility graph Node based optimal algorithms, visibility graph Geodesics algorithm	$0(n\ log\ n) \leq T$	Depende do algoritmo	Sim

2.2 Métodos de percepção de obstáculos

De forma a se poder efetuar o planeamento de trajetórias e percecionar obstáculos é necessário conseguir perceber o ambiente em redor do veículo. Nesta secção são expostos alguns métodos de perceção para o cenário de busca e salvamento com UAVs em que os possíveis sensores serão cameras de espetro visivel ou LIDAR, podendo ser analisados de seguida.

2.2.1 *Optical Flow*

Esta abordagem é utilizada por Antoine Beyeler em [16] onde utiliza um conjunto de cameras onde aplica a técnica de visão *optical flow* para tentar se desviar de obstáculos. Ao método desenvolvido foi dado o nome de *optiPilot* e é uma estratégia de controlo. Esta estratégia foi baseada nos insetos voadores e permite interpretar diretamente as medidas da técnica *optical flow* devido à propriedade do movimento translacional se encontrar alinhado com o eixo principal do veículo. Antoine divide a estratégia de controlo baseado em visão em três etapas, extração da informação do sistema de visão, estimação translacional do *optical flow* e uma terceira etapa que efetua a combinação da informação previamente retirada no sistema de controlo do veículo. Com a deteção de *optical flow* é possível estimar a velocidade, que é independente da frequência de contraste e da intensidade da imagem. Este processo é obtido pela técnica de estimação denominada de *optical flow* simples.

Jean-Christophe Zufferey em [17] utiliza a técnica *optical flow* para UAVs em ambientes *indoor*, estimando a distância para os objetos envolventes do veículo, permitindo desta forma evitar os objetos. Antoine e Jean-Christophe utilizam também a abordagem de *Optical Flow* para o levantar e aterrar do veículo, [18].

2.2.2 *Monocular Vision*

Para o *collision avoidance* muitas vezes é utilizado em veículos aéreos apenas uma camera para perceção de obstáculos, devido ao limite de *payload* e capacidade de processamento destes veículos.

Desta forma a abordagem seguida por Stephane em [20] surge como uma referência para este projeto. A abordagem consiste numa primeira fase em retirar as *features* das imagens recolhidas pela camera e de seguida utilizar estas *features* no controlo do UAV. Possui também uma etapa de aprendizagem, a qual consiste em ter o UAV a fazer o planeamento da trajetória do veículo e sempre que existir atuação humana no *joystick* os

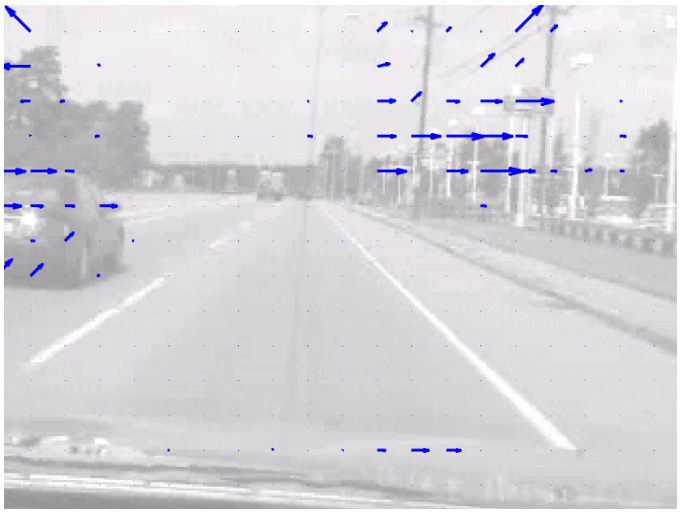


Figura 2.1: Aplicação da técnica Optical Flow [19].

dados referentes aos sensores e trajetória serão guardados de forma a que numa próxima vez que exista uma situação semelhante o veículo corresponda desta forma, para tal recorrem ao algoritmo *DAgger*, [20], que permite múltiplas iterações de aprendizagem.

Outra abordagem proposta é a de Inkyu Sa em [21], onde utilizam as imagens fornecidas pela camera para gerar um mapa utilizando o método *Simultaneous Localisation and Mapping* (SLAM). Como a camera utilizada não se encontra num referencial fixo, é necessário saber sempre a sua posição, para tal é utilizado o método *PTAM*, [22], figura 2.3. Como é utilizado somente uma camera é necessário conhecer a posição desta face ao referencial do UAV e retirar o valor de *Scale* da *feature* detetada. Na etapa seguinte procede-se à estimação do estado onde é aplicado o filtro Kalman Filter (KF), para proceder à estimação da posição e velocidade do veículo. Por fim, na etapa de controlo, utiliza um controlo PID tendo em consideração a posição previamente estimada.

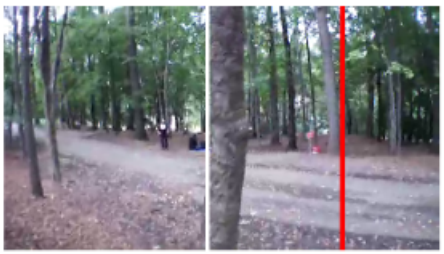


Figura 2.2: Aplicação método DAgger na visão monocular [20].

A solução apresentada por Markus Achtelik em [23] utiliza um método muito parecido ao de Inkyu, utilizando desta forma o Visual *Simultaneous Localisation and Mapping* (VSLAM) para gerar um mapa. De forma a reduzir erros do uso de uma única camera utiliza também sensores de pressão de ar e acelerómetros. Os dados destes sensores serão conjugados com os dados fornecidos pela camera para o VSLAM, sendo posteriormente utilizado um *Kalman Filter* para estimar a posição do veículo. Esta solução para o controlo de posição do UAV utilizou uma estrutura de controlo de dois graus.

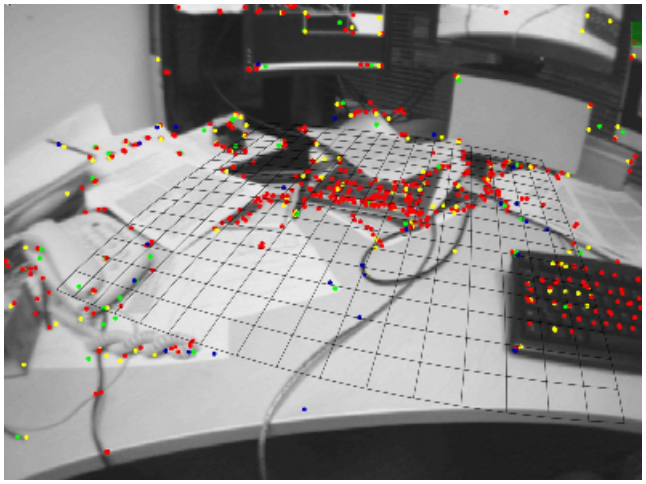


Figura 2.3: Aplicação método PTAM na visão monocular [22].

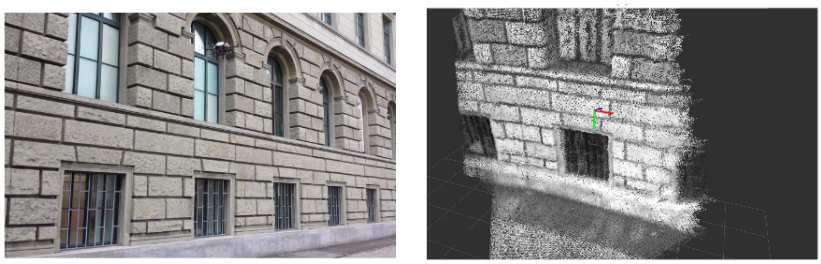
2.2.3 *Stereo Vision*

Como se constata na secção anterior, o facto de se estar a utilizar somente uma camera monocular, não permite efetuar a estimação de profundidade, conseguindo-se apenas obter um fator *Scale* de forma recursiva quando se tem informação e associação de *feactures* entre imagens.

A utilização de um par de cameras com *overlap*, irá permitir ultrapassar essa limitação contudo aumenta o custo do processamento a bordo.

O método sugerido por Omari, [24], estima a profundidade da imagem usando o algoritmo *area-correlation-based block-matching*, gerando uma nuvem de pontos 3D que será guardada num *OctoMap*, podendo posteriormente ser utilizado para estimação da posição do *UAV*. O método de navegação utilizado baseia-se na estimação da posição do UAV, utilizando para tal o mapa 3D obtido anteriormente. O *UAV* referente a esta abordagem é tele-operado, permitindo ao utilizador controlar diretamente o mesmo

usando um controlo remoto. Neste método o objetivo consiste em ser capaz de desviar dos obstáculos através da alteração da velocidade do *setpoint* do veículo de forma a evitar os obstáculos que o rodeiam, recorrendo à introdução de uma velocidade virtual contrária à originada nos objetos com alcance próprio, sendo que este valor pode aumentar ou diminuir conforme a distância seja respetivamente menor ou maior.



(a) Imagem Adquirida (b) Mapa Gerado

Figura 2.4: Exemplo de Mapeamento por stereo [24].

A solução exibida em [25] por Jeffrey Byrne utiliza a abordagem de *collision avoidance Visual Threat Awareness* (VISTA), que é baseada em *real time Stereo*. Este sistema é dividido em quatro etapas, Correspondência *Stereo*, *Foveation*, Segmentação e Deteção de Obstáculos e *Tracking*. Na primeira etapa é retirada a informação das cameras e efetua-se o cálculo *stereo*. O *Foveation*, imagem variante do espaço, é uma representação da imagem com elevada resolução no centro da imagem e menor resolução na periferia, podendo ser implementada usando o *log-polar mapping*. Na terceira etapa, *Segmentation*, procede-se à "rotulagem das *features*"eliminando as imagens que não encontra correspondente. Na última etapa efetua-se a deteção dos obstáculos nas imagens, atribuindo uma *bounding box* do tipo elipse, sendo que a cada objeto detetado será aplicado o *tracking* independentemente, usando um *Kalman Filter*, para estimação da posição de cada obstáculo.

C. De Wagter em [26], utiliza duas cameras no *Micro UAV* e para processar os dados das imagens utiliza o algoritmo *LongSeq*. Este algoritmo é computacionalmente leve, permitindo efetuar o *Stereo* das imagens num curto período de tempo. Após ter utilizado o algoritmo *LongSeq*, que fornece uma imagem com pixels a preto e branco, é verificado qual o conjunto de pixels com uma disparidade superior a um dado valor para as imagens da esquerda e da direita. Posteriormente verifica-se se este conjunto de pixels é inferior a um *threshold* e caso se verifique, o veículo continua o seu percurso seguindo em frente, caso contrário verifica qual a imagem que tem menor valor destes pixels e deste modo o veículo efetua o seu controlo virando para esse mesmo lado.

2.2.4 *Laser Scanner*

Alguns UAVs utilizam somente um *Laser Scanner* para efetuar a perceção de obstáculos, podendo-se verificar pela abordagem seguida por Grzonka em [13], onde utiliza os dados fornecidos pelo laser scanner e guarda-os num mapa de múltiplos níveis, conseguindo com isto localizar o *UAV* num ambiente *indoor* e definir a trajetória de um ponto A para um ponto B sem que ocorra uma colisão. O método proposto por Grzonka é o SLAM (*Simultaneous Localisation and Mapping*) com o objetivo de navegar e efetuar mapeamento com base em informação do LIDAR. No caso do mapa ser previamente conhecido, o método de Grzonka estima a posição 2D do robô para um dado *grid-map* utilizado o método *Monte Carlo Localization*.

As abordagens de Droeschel em [27] e [28] guardam os dados do laser scanner, figura 2.5, que se encontram numa gimbal de dois eixos, num *grid-map* de multi resolução 3D, tendo como centro deste mapa o veículo aéreo. No primeiro caso além da informação da medida 3D do sensor é também guardado a informação da ocupação de cada célula, as medidas de *scans* consecutivos são armazenados em *buffers* circulares de tamanho fixo permitindo o processamento de base de dados e facilidade no acesso da consulta dos vizinhos mais próximos. No segundo caso os pontos 3D obtidos do sensor são guardados numa célula sendo esta marcada como ocupada, podendo posteriormente utilizar o mapa para controlo de desvio de obstáculos, utilizando para tal o método de previsão Potencial Field de forma a evitar as células ocupadas. Os pontos 3D contidos em cada célula podem ser usados também na base de pontos para processamento do *scan*. Com isto consegue-se de uma forma eficiente o mapa de obstáculos.

A abordagem apresentada por Wang em [29] divide a posição do UAV em duas partes de forma a separar a localização fornecida pelos 2 laser scanners. A primeira parte, denominada de posição planar utiliza o algoritmo *planar localization* que vai usar os dados do laser que se encontra em cima do UAV tendo como resultado a estimação do plano horizontal do veículo. A segunda parte é denominada por medida de altura que vai permitir obter a altitude do veículo, usando para tal um segundo laser scanner montado ortogonalmente ao primeiro.

2.2.5 Combinação de *Laser Scanner* com Sistema de Visão

Como se pode verificar nas secções anteriores, o facto de se utilizar somente um sensor para efetuar a perceção de obstáculos, como exemplo uso do laser scanner ou somente de uma camera de espetro visível, faz com que não se consiga recolher toda a informação disponibilizada por estes. De forma a ser possível resolver este outros problemas que

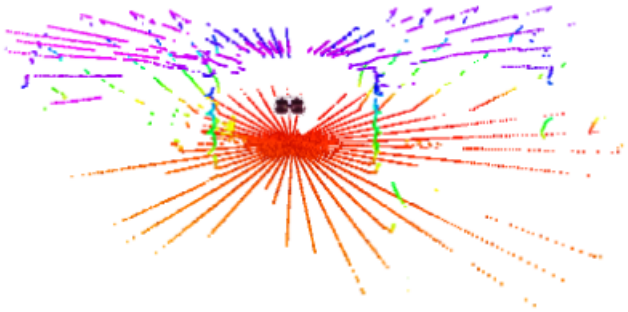


Figura 2.5: Dados recolhidos do Laser Scanner [27].

o uso de um só sensor possui, existem abordagens que efetuam combinações do Laser Scanner com o Sistema de Visão.

No caso da solução anunciada por Nieuwenhuisen em [30], necessita-se de construir um mapa dos obstáculos detetados pelos diversos sensores do robô, câmaras e laser scanner, figura 2.6, estando a reconstrução do mapa dividido em três secções que são descritas de seguida.

- ***Local Multiresolution Map*:** Este mapa de multi resolução terá uma maior resolução nas distâncias mais próximas do veículo e menor nas mais afastadas. A nova informação recebida será guardada em *buffers* circulares que estarão inseridos em cada célula do *grid-map*, permitindo desta forma reduzir o custo computacional. O tamanho dos *buffers* circulares dependerá do tamanho e resolução do mapa. O entrelaçamento de diversos *buffers* permitem obter um mapa 3D.

- ***Scan Registration*:** De forma a compensar o movimento do laser scanner durante a aquisição de medidas é tomada em consideração a estimação da odometria visual. Os dados obtidos pelos laser serão de seguida combinados com os pontos de estatísticos gaussianos e guarda-se no *grid-map* de multi resolução, utiliza-se também o modelo GMM (*Gaussian Mixture Model*). Cada *scan* 3D é adicionado no *grid-map* substituindo todas as medidas anteriores.

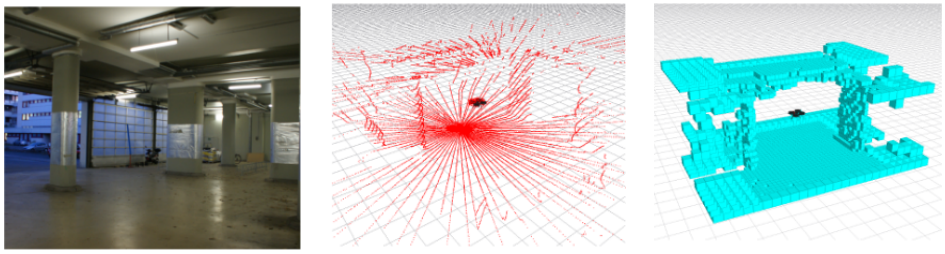
- ***Occupancy Mapping*:** Devido a todos os sensores fornecerem informações complementares, com diferentes exatidões existe a necessidade de fundir as diversas medidas de forma a poder-se gerar um único mapa detalhado, para tal a informação deve ser guardada numa grelha de ocupação onde detêm também a probabilidade de uma dada célula da grelha se encontrar ocupada.

Esta abordagem de Nieuwenhuisen guarda os dados dos sensores num *OctoMap* conseguindo desta forma um mapeamento dos objetos que o rodeiam. O *OctoMap* permite ter uma perceção do mundo em formato de *voxels*, permitindo saber quais os pontos do mundo em que se encontram ocupados e fornece as probabilidades dos espaços se encontrarem ou não ocupados, [31].

A abordagem seguida por Ferrick em [32] utiliza duas câmaras e um laser scanner, sendo que as câmaras têm como principal objetivo vídeo *streaming*, já o laser scanner é utilizado para criar vários mapas, denominados por *bitmaps*, sendo que um destes é o *bitmap* dos obstáculos. Cada pixel de uma imagem *bitmap* indica a probabilidade do espaço se encontrar livre. O método para mapear o ambiente em redor, utilizado por Ferrick é o algoritmo "*occupancy grid*". Este algoritmo começa por uma imagem do mundo, representada por *bitmap*, que usa duas imagens geradas pelo laser scanner. Na primeira imagem é desenhado um círculo por cada feixe do laser e representa a incerteza da distância do objeto, na segunda imagem é indicado o espaço livre encontrado desde o *scan* do laser. Com esta abordagem consegue-se obter um mapa probabilístico do espaço vazio e dos objetos gerados pela informação de cada sensor.

As soluções apresentadas por Huh, [14] e Jutzi, [33], também utilizam um método convencional de deteção *features* que posteriormente poderá ser utilizado para estimação de posição ou *tracking* do UAV, sendo que Jutzi utiliza o método SLAM para mapeamento do ambiente em redor. Em [10] a deteção de obstáculos visuais é baseado nos pontos de interesse obtidos após utilização da ferramenta *OpenCV*. Com esta informação e juntando os dados das medidas obtidas pelo *laser scanner* pode-se gerar um mapa de obstáculos, para tal necessita-se de guardar esta informação numa *grid* 3D, onde cada célula com uma dada medida será marcada como ocupada.

A abordagem seguida por Shen em [9] utiliza o algoritmo SLAM (Simultaneous Localization and Mapping) para mapeamento em tempo real do ambiente que rodeia o veículo, utiliza também uma otimização baseada no *Iterative Extended Kalman Filter* (IEKF) para poder criar uma *grid* 3D consistente com a posição global do veículo.



(a) Imagem Adquirida (b) Dados Laser Scanner (c) Mapa Gerado

Figura 2.6: Exemplo da Combinação *Laser Scanner* com Sistema de Visão [30].

2.3 Análise do Estado da Arte

Para uma rápida análise dos métodos propostos para se conseguir efetuar o planeamento de trajetórias em veículos aéreos dividiu-se a análise em duas partes planeamento de trajetórias e perceção de obstáculos.

No que se refere a métodos de planeamento de trajetórias é possível verificar que estes se podem dividir em cinco secções diferentes. Destas secções as que maior importância possuem por poderem ser aplicados em tempo real são *Sampling-based algorithms*, *Node-based algorithms* e *Multi-fusion based algorithms*. Destas secções os algoritmos mais utilizados são PRM, RRT, algoritmo de Dijkstra, D* Lite, entre outros. Como também é possível verificar existe a possibilidade de integrar diversos métodos de forma a se poder obter o trajeto ótimo livre de obstáculos.

Do que se pode constatar destes métodos apresentados, é necessário ter um conhecimento prévio do mapa completo do ambiente em redor do veículo, sendo para tal necessário desenvolver um método que permita resolver este problema. Outra adversidade é o custo computacional dos métodos apresentados que pode vir a ser reduzido caso se possua um vetor de direção para a posição de destino.

A informação que se pode retirar dos trabalhos existentes no métodos de perceção de obstáculos, é o uso de sistemas de visão individualmente, ou uso de Laser Scanners ou então uma junção destes dois sistemas de forma a permitir perceber e mapear o ambiente em redor do veículo. Uma conclusão que se retira do estado da arte é o facto de o método que mais se utiliza para mapeamento e localização é o SLAM. Com este algoritmo consegue-se localizar e mapear em simultâneo, conseguindo localizar o UAV no ambiente, mesmo sem se ter um pré-mapa. Outras formas encontradas para se obter uma melhor perceção dos obstáculos num ambiente 3D foi o uso de mapas com múltiplos níveis e uso de *grid-maps*. Uma possível solução para gerar o mapa e que foi

utilizada em algumas abordagens foi o uso de *Octomaps*. Algumas abordagens utilizam versões modificadas do *SLAM*, tal como *VSLAM* (Visual Simulataneous Localisation and Mapping), permitindo corrigir possíveis erros, outros métodos usam o *SLAM* em conjunto com outro método, tal como o *Monte Carlo Localization*.

Esta página foi intencionalmente deixada em branco.

Capítulo 3

Fundamentos Teóricos

Neste capítulo será efetuado um estudo teórico de algumas técnicas de planeamento de trajetória assim como uma descrição da ferramenta de mapeamento Octomap e da *framework* ROS (Robot Operating System) para implementação dos algoritmos.

Uma outra linha de trabalho que irá ser analisado será a análise de características técnicas dos simuladores utilizados em robótica com um foco para uma solução de simulador 3D.

3.1 Técnicas de Planeamento de Trajetória

Existem algumas técnicas de planeamento de trajetória sendo que de seguida será dado uma pequena introdução teórica a algumas destas técnicas.

3.1.1 *Potencial Field*

O método de *Potencial Field* é baseado na navegação utilizando um campo magnético em cada obstáculo. Este método começa por calcular o Potencial Atrativo, U_{goal}, relativo à posição do *target*, *goal*, tendo em consideração o atual estado do veículo, ignorando os obstáculos existentes sendo o potencial de atração obtido pela equação 3.1 onde é calculada a distância entre a posição atual do veículo e o ponto que se pretende atingir, sendo efetuado para cada estado do veículo, q.

$$U_{goal} = \bigtriangledown dist(q, goal)^2 \tag{3.1}$$

No que se refere ao cálculo do potencial repulsivo, $U_{obstaculos}$, que é aplicado a todos os obstáculos existentes no percurso, podendo ser obtido pela equação 3.2, em que neste

caso é efetuado o cálculo da distância para o obstáculo. Com o somatório do potencial atrativo ao potencial repulsivo, $U(q)$, equação 3.3, obtém-se um mapa com as direções dos potenciais, como demonstrado na figura 3.1.

$$U_{obstaculos} = \bigtriangledown dist(q, obstaculos)^{-1} \tag{3.2}$$

$$U(q) = U_{goal}(q) + \sum U_{obstaculos}(q) \tag{3.3}$$

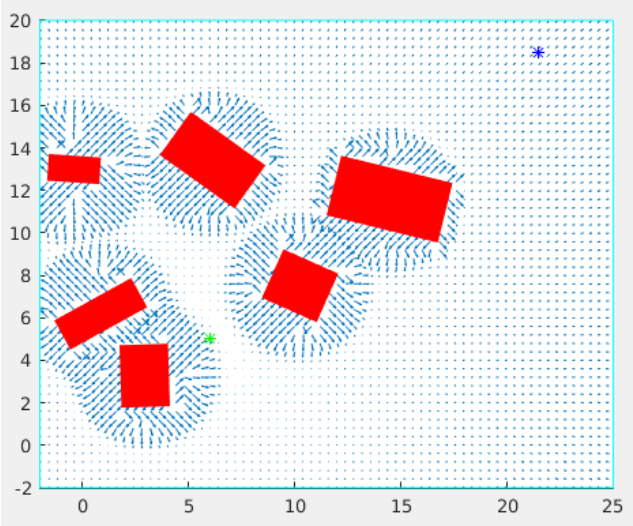


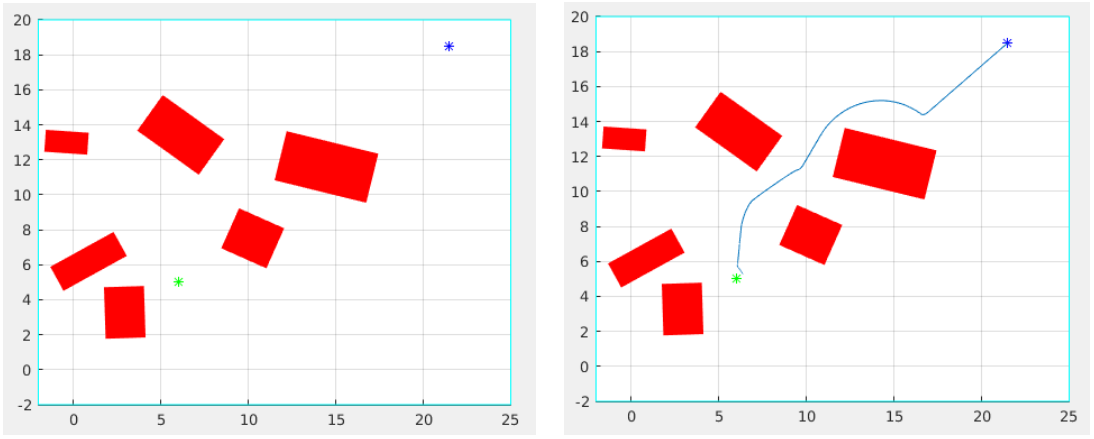
Figura 3.1: *Potencial Field* - Conjunto das forças atractivas e repulsivas. (* azul - atual posição do veículo; * verde - posição do target)

Com base nesta informação é calculada a trajetória que o veículo deve efetuar para atingir a posição final desejada, para tal tem em consideração o mapa previamente obtido repetindo os seguintes passos enquanto não chegou ao ponto final.

- Calcula os vetores da força para todos os *potential fields*, usando a equação 3.3.
- Efetua a navegação ao longo dos campos com uma velocidade proporcional à força do campo em que se encontra, $U(q)$.

Seguindo estas etapas é possível obter uma trajetória similar à da figura 3.2(b) quando aplicado ao cenário da figura 3.2(a).

Existem outros métodos de calcular a força repulsiva de cada obstáculo, tal com a definida em [34][35][36] e [37].



(a) Ambiente com obstáculos (* azul - atual posição do veículo; * verde - posição do target)

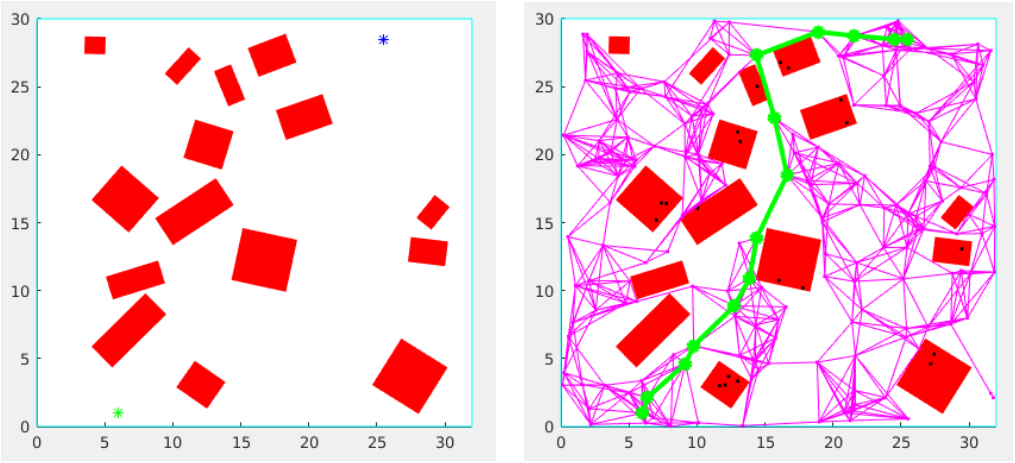
(b) Trajetória obtida (* azul - atual posição do veículo; * verde - posição do target)

Figura 3.2: *Potencial Field* - Ambiente para simulação.

3.1.2 *Probabilistic Roadmap*

Outra técnica de planeamento de trajetória é o *Probabilistic Roadmap* (PRM) que permite através de amostras aleatórias encontrar o trajeto para o *target*. Este algoritmo tem como parâmetros de entrada os obstáculos, a posição atual e a posição desejada. O método começa por espalhar pelo mapa o número de amostras definidas, verifica quais destas amostras é que se encontram dentro dos obstáculos e retira-as, adicionando posteriormente a posição inicial e a final às amostras. De seguida, cada amostra tenta conectar-se a um máximo de amostras vizinhas com menor distância (criação do *roadmap*), posteriormente a este passo e caso exista algum tipo de conexão entre todas as amostras é calculado o caminho mais curto desde a posição inicial até à final, sendo que os métodos mais utilizados para esta etapa o método A* e de Dijkstra, [38]. No método PRM é possível indicar o número máximo de amostras a serem espalhadas pelo mapa e indicar o número máximo de amostras que podem ser consideradas vizinhas, [6] e [39]. Na figura 3.3 apresenta-se o ambiente com os obstáculos e a trajetória calculada quando utilizado este método.

O método PRM é fundamentado pelo algoritmo 4, o qual necessita de ter como parâmetros de entrada a posição atual do veículo, x_{init}, a posição do *target*, x_{goal}, o número de amostras a espalhar, nS, e os obstáculos que se encontram no ambiente, ${}^{W}\mathrm{C}$. Este algoritmo retorna o trajeto mais curto, P, desde o ponto inicial até ao *target*.



(a) Ambiente com obstáculos (b) Trajetória obtida

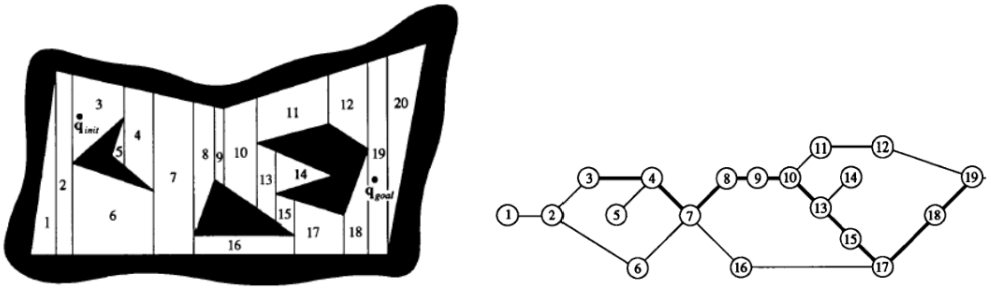
Figura 3.3: PRM - Ambiente de simulação.

3.1.3 Trapezoidal

O método Trapezoidal é outro método possível para planeamento de trajetórias, para tal esta abordagem começa por decompor os espaços que não possuam obstáculos, espaço livre, em células trapezoidais e triangulares como pode ser verificado na figura 3.4(a). Esta divisão do espaço livre gera uma nova célula a cada novo vértice encontrado, ou no eixo do X, como no exemplo, ou no eixo do Y. Com base nessa informação é gerado um gráfico representativo das células adjacentes, figura 3.4(b), obtendo-se posteriormente, os centroides de cada célula, designado por *roadmap*.

Algorithm 1 $(P) \leftarrow \mathrm{PRM}(x_{init}, x_{goal}, nS, {}^{W}\mathrm{C})$

eA $\leftarrow$ Environment_area(x_{init}, x_{goal})
S $\leftarrow$ Sampling$(nS, eA, {}^{W}\mathrm{C})$
{RoadMap$(S, {}^{W}\mathrm{C})$};
for each $s \in S$ **do**
  Nearest$(R = (V, eA, s) := argmin_{v \in V} \parallel s - v \parallel)$
  Near$(R = (V, eA), s, r) := \{v \in V : v \in ß_{s,r}\}$
  $CollisionFree(s, s^{j}, {}^{W}\mathrm{C}) \in eA$
end for
P $\leftarrow$ Shortestpath(x_{init}, x_{goal}, R)
  {Dijkstra Algorithm};
return (P)



(a) Divisão do Ambiente em trapézios e (b) Gráfico representativo das células adjacentes triângulos

Figura 3.4: Método Trapezoidal - Ambiente exemplo [40].

Tendo em consideração o *roadmap* é determinado o trajeto mais curto entre as células inicial e final, local onde se encontram os pontos inicial e final respetivamente, recorrendo para isso do algoritmo *Breadth-first search* (BFS) que é um algoritmo de pesquisa em grafos ou em dados do tipo árvore, [41] e [40] Na figura 3.5 é apresentado o resultado obtido para o exemplo detalhado.

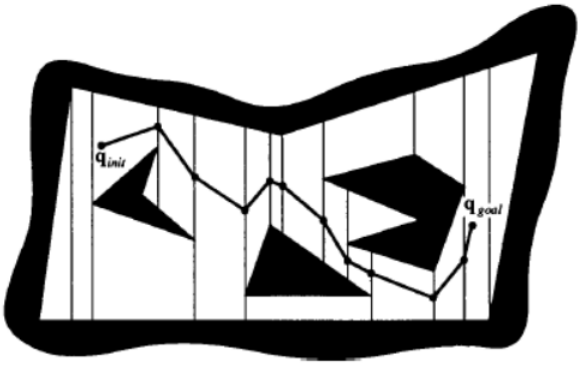


Figura 3.5: Método Trapezoidal - Trajetória Final [40].

3.1.4 Voronoi

O Diagrama de Voronoi é um método desenvolvido para a criação de *roadmaps*, [42]. Este método utiliza um conjunto de n pontos, S, (denominados de *sites*) num plano, para dois *sites* distintos $p, q \in S$ a dominância de p sobre q é definida como um subconjunto do plano tendo uma distância equidistante entre p e q. Formalmente é descrito na equação 3.4, onde δ é a função da distância euclidiana, $dom(p, q)$ é o semiplano fechado delimitado

pela bissetriz perpendicular à reta que une p a q.

$$dom(p,q) = x \in R^2 \mid \delta(x,p) \leq \delta(x,q) \tag{3.4}$$

A região de um *site* $p \in S$ é a porção de plano de assentamento em todas as dominâncias de p nos restantes *sites* em S, podendo ser descrito pela equação 3.5

$$reg(p) = \bigcap_{q \in S - \{p\}} dom(p,q) \tag{3.5}$$

O facto de as regiões serem geradas pela interseção de $n-1$ semiplanos estas tomam a forma de polígonos convexos. Com isto, os pontos que se encontrem nas arestas ou nos vértices destas regiões encontram-se equidistantes entre dois *sites*. Como consequência das regiões serem aresta/ aresta e vértice/ vértice, diz-se que formam uma partição poligonal do plano, denominado de Diagrama de Voronoi, $V(S)$ para um conjunto de pontos, S, finitos.

Na figura 3.6 é demonstrado um exemplo do Diagrama de Voronoi com um conjunto de pontos, S, e as respetivas regiões.

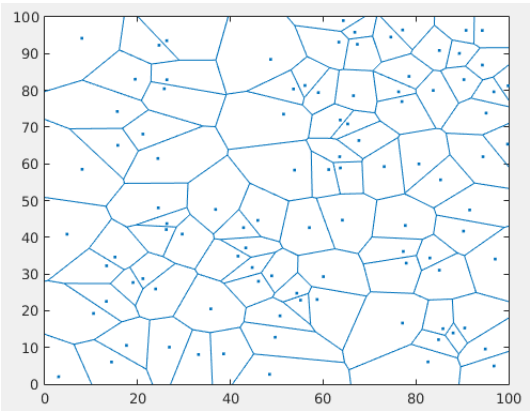


Figura 3.6: Método *Voronoi*.

3.1.5 WaveFront

A abordagem seguida pelo método WaveFront, [43] e [44], pretende ser uma das possíveis soluções no planeamento de trajetórias de um veículo que pretenda ir de um ponto inicial até a um ponto final. De modo a conseguir realizar tal objetivo esta abordagem baseia-se no método *Cell Decomposition*, sendo que inicialmente divide o

mapa que o veículo gerou numa *grid* identificando o espaço livre, a posição onde o veículo se encontra e a posição que pretende atingir assim como os obstáculos que se encontram no ambiente. Para tal, atribui-se o valor zero aos espaços livres, o valor um às células onde se encontram os obstáculos, à posição atual e de destino do veículo é atribuído os valores de zero e dois respetivamente. Na figura 3.7 encontra-se representado um exemplo desta inicialização do algoritmo.

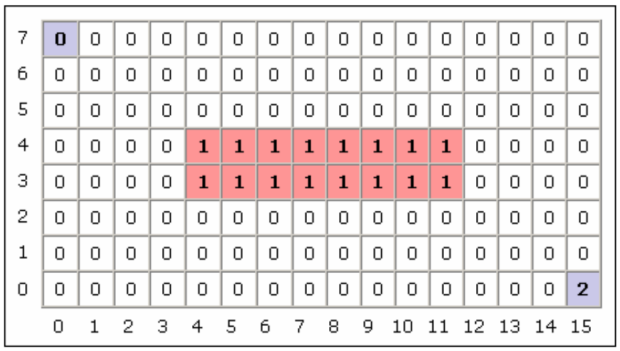
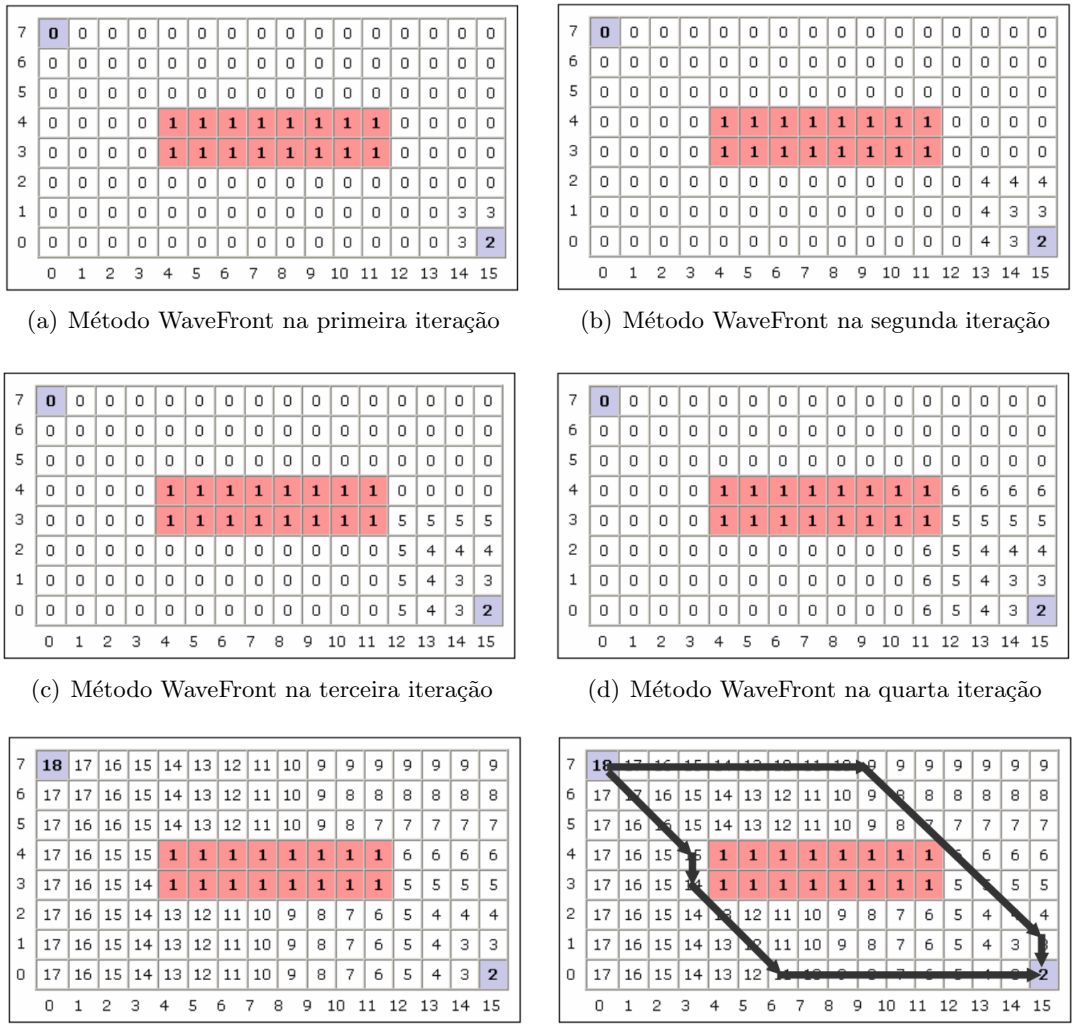


Figura 3.7: WaveFront - Exemplo inicial [44].

A etapa seguinte do método implica selecionar a célula do ponto final que o veículo pretende atingir e nas suas células adjacentes livres inserir um valor que será igual ao valor da atual célula final incrementado de um, figura 3.8(a).

De seguida será repetido o processo às células que anteriormente foram modificadas, como se pode verificar na sequência de imagens da figura 3.8, até que se chegue à célula do ponto inicial.

O método seguido pelo algoritmo WaveFront para encontrar o menor trajeto entre o ponto inicial e o final é descrito como um simples movimento no sentido decrescente dos valores das células, ou seja, o algoritmo começa no ponto inicial, célula que contém o valor mais elevado, de seguida é escolhida a célula adjacente com menor valor, sendo indiferente à direção que a célula possua. Posteriormente é aplicado o mesmo método à célula anteriormente escolhida, sendo este processo repetido enquanto não se atingir a célula do ponto final, célula com o valor mais pequeno, pode-se verificar esta descrição na figura 3.8(f), onde neste exemplo se possui dois caminhos possíveis.



(a) Método WaveFront na primeira iteração

(b) Método WaveFront na segunda iteração

(c) Método WaveFront na terceira iteração

(d) Método WaveFront na quarta iteração

(e) Método WaveFront na iteração n

(f) Método WaveFront - Determinação do trajeto.

Figura 3.8: Sequência do processo do método WaveFront [44].

3.2 Octomap

A *framework* Octomap foi desenvolvida por Armin Hornung na Universidade de Freiburg na Alemanha. Esta usa uma representação baseada numa *tree* de forma a oferecer melhor resolução de uma área mapeada, recorrendo a uma estimativa probabilística da ocupação garantindo *update* das medidas e permitindo suportar o ruído dos sensores. O Octomap assenta em cinco partes, *Octrees*, Fusão Sensorial Probabilística, Pesquisa Multi-Resolução, Compressão do *Octree Map* e extensões.

- ***Octrees*:** Uma *octree* é uma estrutura de dados hierárquica para subdivisão espacial 3D, onde cada *node* de uma *octree* representa um dado volume do espaço e é denominado por *voxel*. Estes nodes podem ser consecutivamente subdivididos em outros oito novos *voxels*, até se obter o *voxel* com o tamanho mínimo pré-definido pelo utilizador, denominado por resolução da *octree*. A *octree* permite cortar a *tree* em qualquer nível podendo-se com a mesma resolução máxima definida obter mapas com diferentes resoluções. Aplicando as *octrees* na sua forma básica, é utilizada como um modelo de propriedade Booleana, que no contexto do mapeamento na robótica é definido como ocupado num dado volume. Sempre que um determinado volume é definido como ocupado, o *node* correspondente na *octree* é inicializado, sendo que os novos *nodes* serão inicializados como livres ou ocupados. Na figura 3.9 encontra-se representado um exemplo de uma *octree* com uma hierarquia de três níveis, obtendo-se assim a máxima resolução da *octree*, onde em cada nível é possível verificar os *voxels* que se encontram totalmente ocupados (representados com a cor preta), os que estão vazios (representados com a cor branca) e os *voxels* mistos, ou seja que não estão totalmente ocupados (representados com a cor cinzenta).

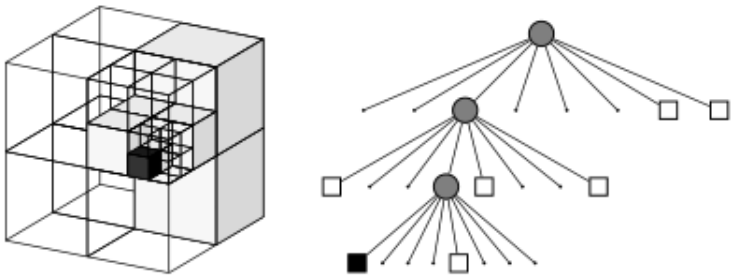


Figura 3.9: *Octree* - Exemplo de uma Octree (branco - voxels vazios; preto - voxels completamente cheios; cinzento - voxels mistos) [31].

- **Fusão Sensorial Probabilística:** A abordagem utilizada para a integração dos dados dos sensores utiliza o *occupancy grid mapping* de Moravec e Elfes onde a probabilidade de um dado *node* n se encontrar ocupado, $P(n|z_{1:t})$, dada uma medida do sensor, $z_{1:t}$, pode ser obtida através da equação 3.6, onde o *update* depende da atual medida do sensor z_t, da probabilidade anterior $P(n)$ e da estimação anterior $P(n|z_{1:t-1})$. O termo $P(n|z_t)$ corresponde à probabilidade do voxel n se encontrar

ocupado tendo em consideração a medida z_t.

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{3.6}$$

Considerando uma probabilidade prévia uniforme, então $P(n) = 0.5$ e usando a notação logaritmica pode-se reescrever a equação 3.6, como a apresentada na equação 3.7.

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \tag{3.7}$$

com,

$$L(n) = log\left[\frac{P(n)}{1 - P(n)}\right] \tag{3.8}$$

Quando se utiliza o mapa 3D para navegação é necessário definir um *threshold* na probabilidade de ocupação. Um *voxel* é considerado como ocupado quando é atingido o *thresold* e é considerado livre quando não atingiu o limite, sendo desta forma definido dois estados discretos. Desta forma é apresentado na equação 3.9 uma formulação de *update* tendo em consideração os limites máximos, l_max, e mínimos, l_min, da estimação.

$$L(n|z_{1:t}) = max(min(L(n|z_{1:t-1}) + L(n|z_t), l_max), l_min) \tag{3.9}$$

Desta forma é possível verificar duas vantagens, assegura-se que a confiança do mapa mantêm-se entre limites e como consequência o modelo pode adaptar-se rapidamente às mudanças do ambiente.

- **Pesquisa Multi-Resolução:** A pesquisa em multi-resolução é efetuada através dos *nodes* interiores da *tree*. Para determinar a probabilidade de um *node* interior é necessário agregar as probabilidades dos seus filhos, desta forma e dependendo aplicação a ocupação média e a ocupação máxima podem ser obtidas pelas equações 3.10 e 3.11, onde n corresponde ao *node* interior e n_i aos seus oito sub-volumes.

$$\bar{l} = \frac{1}{8} \sum_{i=1}^{8} L\left(n_i\right) \tag{3.10}$$

$$\hat{l} = \max_i L\left(n_i\right) \tag{3.11}$$

Assumindo que o volume se encontra ocupado se alguma parte possui uma medida ocupada, pode-se calcular trajetos livres de colisões, sendo esta a principal razão

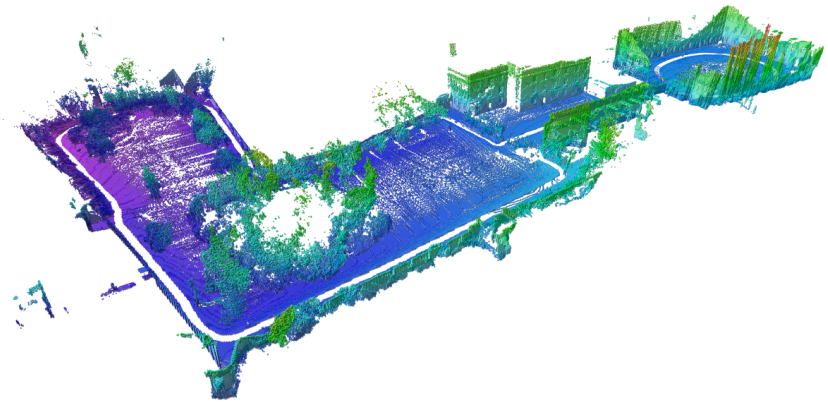
para o uso do *update* máxima de ocupação.

- **Compressão do *Octree Map*:** A compressão do mapa é dada pela política de *update* dada pela equação 3.9. Com esta aplicação consegue-se obter bons resultados devido a ser relativamente estável o valor da probabilidade.

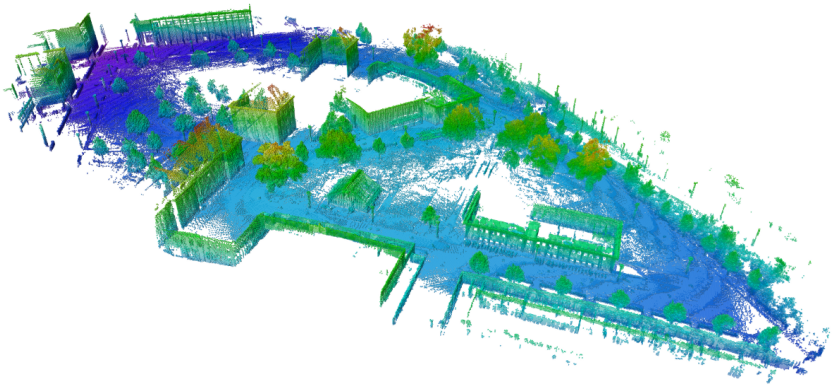
- **Extensões:** Os *nodes* da *Octree* podem conter informações adicionais além da distância, tais como temperatura ou cor do ambiente, permitindo gerar mapas mais completos. Outra extensão possível desta *framework* é a possibilidade de gerar sub-mapas hierárquicos.

A ferramenta *Octomap* possui a capacidade de ser integrada com a *framework* ROS permitindo gerar mapas face a um referencial determinado pelo utilizador, podendo se gerar o mapa com múltiplos sensores no mesmo instante, desde que se consiga fornecer uma *Point Cloud* dos pontos em 3D, sendo esta uma das grandes vantagens do *Octomap*. Como output o *Octomap* fornece o mapa num formato com tamanho reduzido, podendo este ser no formato binário ou então num formato completo, formato este que fornece além da máxima *likelihood* do mapa ocupado, contempla também as probabilidades completas de um dado *voxel* se encontrar ou não ocupado e outros dados adicionais. Além destes outputs, o *Octomap* fornece somente as células que se encontram ocupadas, ou seja, as células que correspondem ao local onde existem obstáculos, fornece também uma *Point Cloud* com os centroides destas células ocupadas. O facto de o *Octomap* ser uma ferramenta *Open Source*, permite que o utilizador possa personalizar os inputs e outputs como bem entender. Na figura 3.10, encontra-se representados alguns mapas gerados pelo *Octomap*, podendo também se verificar a sua diferencia para diferentes valores de resolução da *octree*, [31].

.



(a) *Octomap* - *Dataset* New College



(b) *Octomap* - *Dataset* Exterior Freiburg

Figura 3.10: Exemplos de *Datasets* usando *Octomaps* [31]

3.3 ROS - Robot Operating System

O ROS (Robot Operating System), [45] e [46], é uma *framework Open Source* concebida para aplicações em robótica. Foi desenvolvido de forma a ser modular, inclui um *middleware* de comunicações para diversas plataformas de hardware, existindo a possibilidade de incorporar módulos (*packages*) de projetos já existentes. Esta *framework* possui uma grande comunidade científica para apoio e desenvolvimento de novos *packages* e um conjunto de recursos online incluindo documentação destes. Esta aplicação também permite a interface com múltiplos sensores e atuadores utilizados pelos diversos utilizadores da comunidade cientifica e pode ser programado em diferentes linguagens, tais como C++, Python, Octave e LISP.

Um sistema de ROS normalmente inclui um número de processos independentes, de-

nominados por *nodes*, podendo estes comunicar entre si utilizando o mecanismo *publish - subscriber*, passando entre eles tipos de mensagens, *topics*. A comunicação entre *nodes* é baseada no tipo de comunicação *peer-to-peer*. Este *middleware* também providencia interrupções remotas síncronas e assíncronas, denominadas de *services* e *actions* respetivamente. De forma a que os *nodes* possam funcionar é necessário correr o *roscore*, que estabelece a conectividade entre os *nodes* que estão a ser executados. Outra funcionalidade importante do ROS é o agrupamento em *packages* dos *nodes* individuais, sendo que por sua vez os *packages* estão agrupados em *meta-packages*. Esta *framework* permite também gerar mensagens personalizadas para cada *node*, sendo que as mensagens são do tipo linguagem-neutra, ou seja, são capazes de serem interpretadas pelas diversas linguagens de programação.

3.4 Simuladores

Nos dias de hoje muitos laboratórios de investigação possuem múltiplos robôs de diversas áreas. No caso particular dos robôs aéreos, muitos dos laboratórios podem não possuir os recursos necessários para que se efetuem os testes necessários utilizando veículos reais. De forma a combater esta necessidade, existem alguns simuladores 3D que permitem efetuar os testes sem que seja necessário possuírem um robô. Alguns desses simuladores serão abordados de seguida.

3.4.1 Gazebo

O Gazebo, [47], [48], [49], [50], [51], [52], é um simulador tridimensional que permite efetuar simulações multi-robôs em ambientes *indoor* e *outdoors* em real-time. Foi desenvolvido pela University of Southern California para o DARPA e possui alguns robôs já desenvolvidos tais como PR2, Care-O-bot, TurtleBot entre outros. Os robôs integrados neste simulador são desenvolvidos com dinâmica, o ambiente onde vai ser efetuada a simulação tem em consideração a gravidade, forças de contacto e de fricção. Nestes robôs podem ser incluídos diversos sensores que tentam simular um sensor real, para tal tem um erro associado de primeira ordem *Gauss Markov*. De referir que o Gazebo diferencia os elementos Colisão dos Visíveis, tratando apenas os elementos Colisão como Visíveis para os laser scanners e na verificação de colisão. Os robôs a serem inseridos no ambiente de simulação, bem como os sensores neles incluídos podem ser desenvolvidos na ferramenta *Blender* ou então em ODE (Open Dynamics Engine), permitindo que o renderização dos ambientes possam ser o mais realísticos possíveis. Além desta facilidade

de uso de múltiplas ferramentas para gerar ambientes, o simulador também permite a integração com *middlewares*, tais como ROS (*Robot Operating System*) e YARP. Este simulador partilha de muitas funcionalidades que outros simuladores, ex. Simulador MORSE. Na figura 3.11 encontra-se representado o exemplo de um veículo no Gazebo.

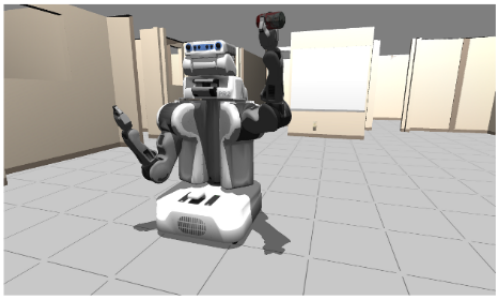


Figura 3.11: Simulador Gazebo [47].

3.4.2 USARSim

O simulador USARSim (Urban Search and Rescue Simulation), [53], [54], [55], [56], [57], é um simulador 3D Open-Source desenvolvido pela University of Pittsburgh para simulação de cenários busca e salvamento destinados à investigação e à educação, tendo sido já utilizado na competição de salvamento do RoboCup. Este simulador é multi-plataforma e baseia-se num motor de jogo comercial designado de Unreal Engine, o que se torna uma vantagem para o simulador, pois sempre que existe uma atualização do motor de jogo, o simulador também irá melhorar. O USARSim permite o uso de multi-robôs ou então o uso de um único robô, tais como veículos terrestres, *underwater*, aéreos e humanoides, em ambientes *indoor* e *outdoor*. Os ambientes de simulação podem ser desenvolvidos com o uso da aplicação UnrealEd, já os robôs e os sensores são desenvolvidos na linguagem C++ ou JavaScript. Estes veículos, tais como os respetivos sensores, podem ser facilmente adicionados aos ambientes de simulação usando para tal o Unrealscript. De referir que os sensores e o sistema de simulação fornecem dados semelhantes aos reais, permite também testar e desenvolver interfaces de interação Humano-Robô (*HRI - Human-Robot Interaction*). Comparativamente a outros simuladores o facto de não conseguir interagir com a *framework* ROS, neste projeto é uma grande desvantagem. Na figura 3.12 encontra-se representado um ambiente desenvolvido utilizando o simulador USARSim.

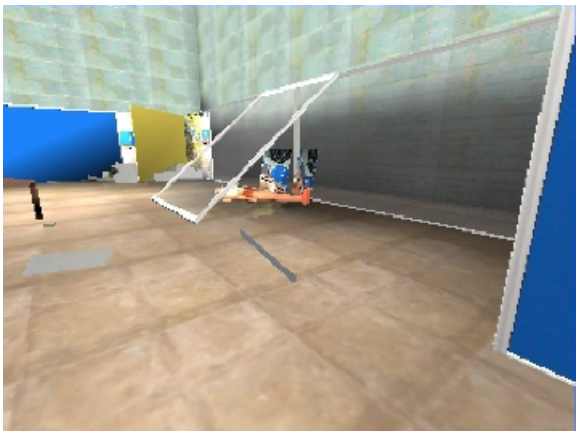


Figura 3.12: Simulador USARSim [53].

3.4.3 Webots

Outro simulador 3D é o Webots, [58], [59], [60], [61] e [62]. Este é um simulador comercial criado pela Cyberbotics, *spin-off* da Swiss Federal Institute of Technology in Lausanne (EPFL), com o intuito de ser utilizado por investigadores e professores interessados em robôs móveis. O Webots é multi-plataforma, permite simular robôs com rodas, pernas e robôs aéreos e detém um GUI (Graphic User Interface) para as instruções de movimentos dos robôs. A construção de novos modelos de robôs neste simulador é simples e relativamente rápida, sendo que a estes robôs é possível adicionar os sensores que se pretender com base nos sensores e atuadores já existentes. Os sensores que são adicionados aos robôs podem ser individualmente configurados, como por exemplo no seu alcance, ruído, resposta, campo de visão, etc. Além destas características o simulador recorre ao ODE para uma simulação precisa da física, permite criar filmes das simulações nos formatos AVI e MPEG e pode usar-se para programação a linguagem C, C++, Java, Matlab, Python ou Urbi. Como desvantagem face aos simuladores anteriormente enumerados é o facto não poder existir interação do simulador com nenhuma *framework*. Na figura 3.13 encontra-se demonstrado um robô no ambiente de simulação Webots.

3.4.4 V-REP

O V-REP (Virtual Robot Experimentation Plataform), [63], [49], é um simulador 3D que foi desenvolvido pela Coppelia Robotics e possui dois tipos de licença, uma comer-

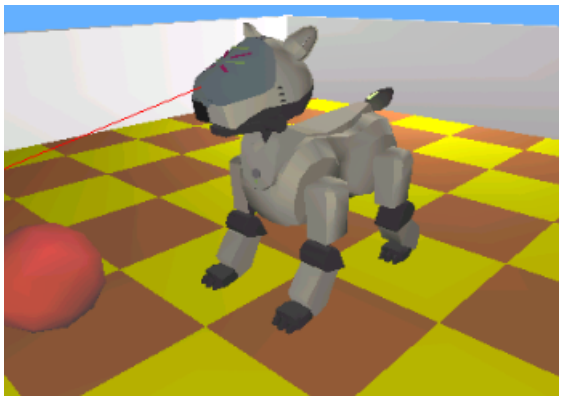


Figura 3.13: Simulador Webots [58].

cial e uma educacional, sendo ambas multi-plataforma. Este simulador faz uso de um motor de renderização personalizado e oferece apoio para três motores de física (Bullet, ODE e Vortex), sendo estes comutáveis durante o tempo de execução. O V-REP fornece uma API para C++ e Lua, no entanto pode-se programar os controladores em C/ C++, Python, Java, Lua, Matlab, Octave ou Urbi. A nível de simulação é possível simular utilizando *plugins*, *scripts* incorporados ou *nodes* ROS, sendo também possível ter ao mesmo tempo múltiplos robôs num único ambiente de simulação. Os robôs que podem ser adicionados ao ambiente de simulação podem ser veículos terrestres, aéreos ou *underwater*, no entanto para estes últimos é necessário estar num ambiente apropriado e adicionar *plugins* ao V-REP. Estes robôs podem possuir múltiplos sensores no entanto o apoio fornecido pelo simulador é reduzido. Na figura 3.14 é apresentado um cenário usando o simulador V-REP.

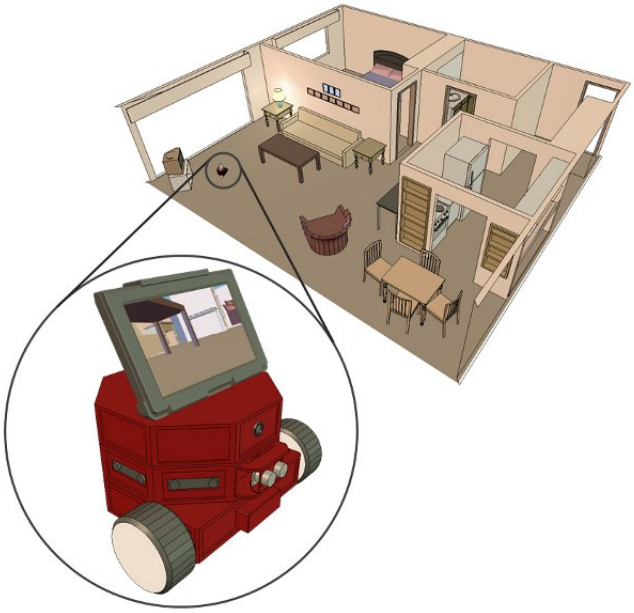


Figura 3.14: Simulador V-REP [63].

3.4.5 MORSE

O MORSE (*Modular Open Robots Simulation Engine*), [64], [65], [66], [67], foi desenvolvido em conjunto entre os laboratórios de pesquisa LAAS e ONERA e é uma ferramenta Open-Source, multi-plataforma e completamente modular. Todos os componentes de uma simulação podem ser integrados com outros componentes sendo desta forma possível para uma simulação escolher um ambiente que possuirá um ou mais robôs e esses robôs possuirão sensores que serão independentes e só pertencerão a um dado robô. A isto acresce o facto do simulador permitir a interação com diversos *middlewares*, entre eles YARP, ROS, Pocolibs, etc., o que permite que em conjunto com o facto de ser um simulador modular que num ambiente de simulação um robô esteja a ser controlado por um *middleware*, exemplo YARP, ao mesmo tempo um dos sensores está a enviar dados para o socket e um outro robô se encontra a ser manipulado através do *middleware* ROS. Toda estas vantagens só conseguem ser realizadas pois o simulador encontra-se desenvolvido em cima da aplicação *Blender Game Engine* (BGE). O BGE é uma aplicação de modulação e renderização 3D *open-source*, conseguindo a vantagem de possuir um detalhe a nível gráfico muito elevado em situações de *real-time*. Permite também o uso de cameras para verificação da evolução da simulação e pode ser programado em Python ou C++. O simulador permite testar simulações em ambientes *indoor* e *outdoor* que podem

ser personalizados ou gerados pelo utilizador, podendo ser adicionados aos cenários robôs aéreos, terrestres e aquáticos. A estes robôs é possível adicionar os múltiplos sensores e atuadores já desenvolvidos e que possuem características semelhantes aos homónimos reais. Estes sensores e atuadores são minimalistas na sua funcionalidade e fornecem dados similares aos respetivos sensores/ atuadores reais, sendo que alguns destes tem múltiplas variantes podendo funcionar com diferentes níveis de realismo e abstração. É também adicionado a estes sensores e atuadores um ruído, ou então é alterados os dados conforme necessário de forma a que os dados sejam os mais reais possíveis. Com estas características estes componentes moduladores conseguem ser versáteis e configuráveis para testes de *software-in-the-loop* de software de robótica. Na figura 3.15 é apresentado um exemplo de um cenário do simulador MORSE.

Figura 3.15: Simulador Morse [67].

Capítulo 4

Projeto

Um dos grandes problemas que um veículo aéreo enfrenta em cenários de desastres, semelhante ao apresentado na figura 4.1, é a enorme quantidade de obstáculos presentes, o que faz com estes veículos necessitem de ter dimensões reduzidas de forma a poderem passar as estreitas passagens existentes, o que de uma forma indireta vai diminuir o limite de peso que os veículos podem suportar. Este condicionamento implica que a quantidade de sensores a serem aplicados tem de ser o mínimo necessária ao funcionamento e de dimensões reduzidas. Um outro entrave que os obstáculos criam nos UAVs é na navegação autónoma, o que implica que o veículo tenha uma boa perceção dos obstáculos que o rodeiam, pelo que é necessário que o veículo precisa de ter uma boa capacidade computacional de forma a conseguir gerar um mapa dos obstáculos.

No veículo que se encontra em desenvolvimento no laboratório, pretende-se utilizar um sensor laser scanner, que permita detetar corretamente o ambiente em redor do UAV, e que permita ser utilizado para gerar um mapa de forma a que o UAV saiba a sua posição face a cada obstáculo. Após se ter conhecimento da posição dos obstáculos é necessário proceder ao planeamento da trajetória de modo a ser capaz de navegar pelo ambiente de forma autónoma.

Neste capítulo é detalhada a arquitetura geral do sistema que irá permitir endereçar todos os requisitos previamente enumerados tendo por base o estado da arte e os fundamentos teóricos apresentados anteriormente.

Neste projeto o sensor a aplicar no UAV será o laser scanner, pois permite obter os dados suficientes relativamente aos obstáculos que se encontram no ambiente, como exemplo o cenário apresentado na figura 4.1, além de também possuir um baixo peso, o que é uma característica importante para os UAVs. De modo a poder-se aplicar os métodos desenvolvidos num simulador foi escolhida a *framework* ROS, pois permite

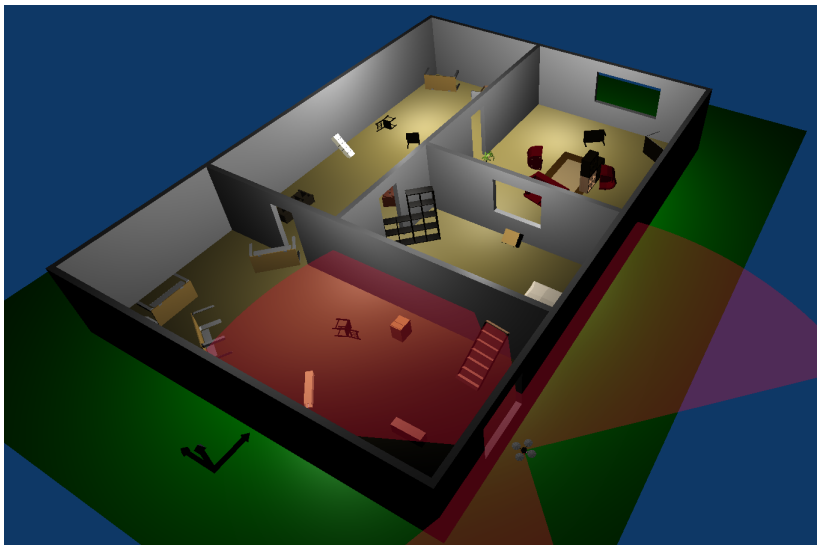


Figura 4.1: Ambiente de aplicação simulado.

interligar a aplicação desenvolvida com diversos simuladores além de permitir utilizar um *package* já desenvolvido para laser scanners. De forma a ser possível realizar simulações das aplicações desenvolvidas foi utilizado o simulador MORSE, que permite utilizar uma vasta gamas de sensores em diferentes cenários, sendo que estes cenários podem ser gerados de raiz pelo utilizador. Outra vantagem em recorrer ao MORSE é a possibilidade de integração com a *framework* ROS a qual será também ela utilizada.

4.1 Arquitetura do Sistema

De forma a ser possível resolver o problema proposto de planeamento de trajetória 3D para *UAV*s, onde necessita de se deslocar de um ponto A para um B evitando colisões, estabeleceu-se uma arquitetura que é dividida em três camadas Perceção, Mapeamento e Planeamento de Trajetórias, como detalhado na figura 4.2.

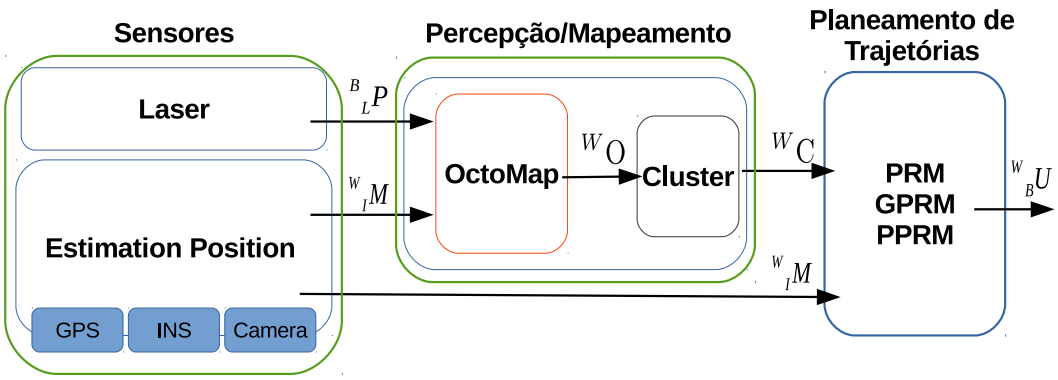


Figura 4.2: Arquitetura geral do sistema.

De forma a se percecionar melhor os referenciais apresentados é exposto de seguida um esquema detalhado e pormenorizado, figura 4.3. Neste esquema observa-se os referenciais entre os sensores e o veículo relativo ao referencial global, podendo-se verificar que $\{W\}$ corresponde ao referencial global, $\{B\}$ corresponde ao referencial do veículo, neste caso do UAV, $\{I\}$ corresponde ao referencial do sensor de posição e atitude do veículo e $\{L\}$ corresponde ao referencial do laser scanner.

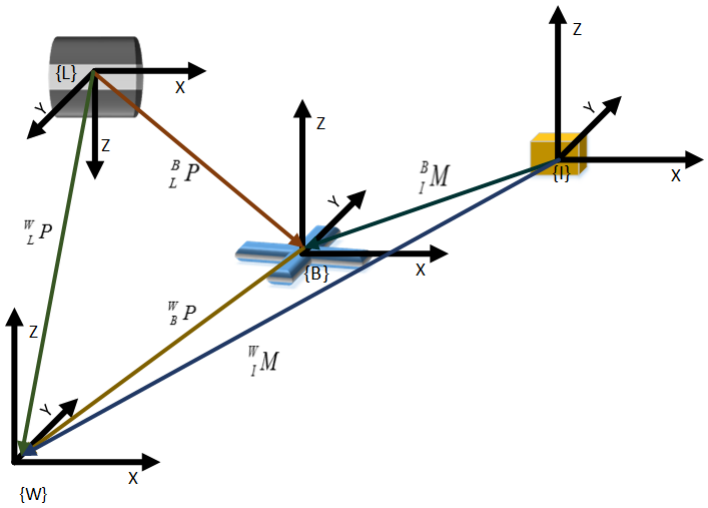


Figura 4.3: Referenciais 3D do veículo e sensores.

Serão estes os referenciais que se terá como base para os subsequentes capítulos e secções.

4.1.1 Sensores

Na primeira camada do sistema desenvolvido encontra-se o bloco de Sensores. Nesta secção obtém-se a informação recolhida pelo *laser scanner 2D*, ${}^{B}_{L}\mathrm{P}$, que posteriormente será utilizada para se gerar um mapa dos obstáculos do ambiente e posteriormente se conseguir definir uma trajetória que evite os obstáculos. De referir que o laser scanner utilizado é um *Hokuyo* e que se encontra fixo à *frame* do veículo, sem qualquer *gimbal* como processo intermédio. Nesta camada também se recolhe a informação relativa à posição e atitude atual do veículo, ${}^{W}_{I}\mathrm{M}$, dados que serão importantes para as etapas seguintes. A posição e atitude do veículo encontra-se a ser estimada pelo *autopilot PX4* que fornece mensagens *MAVLink*, com posições e atitudes. De modo a ser possível integrar estas mensagens na *framework* ROS é utilizada a ferramenta *MAVROS* que permite interpretar as mensagens *MAVLink* e transforma-las em *nodes* ROS. Por sua vez o *MAVLink* é uma biblioteca, bastante leve, de mensagens header-only para pequenos veículos aéreos. A posição também é estimada recorrendo à abordagem de *Visual Odometry*, [68], permitindo ter posição do veiculo em ambientes *indoor* onde o sinal de GPS é inexistente, através das *feactures* retiradas das imagens.

4.1.2 Percepção e Mapeamento

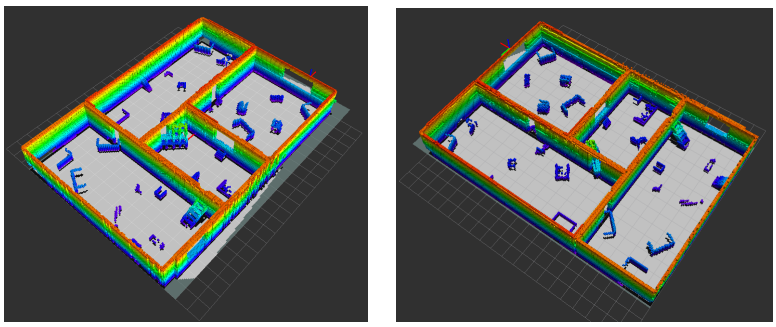
A camada de mapeamento será responsável pela construção do mapa do ambiente em redor do veículo. Para este bloco são necessários os dados da posição e atitude do veículo e os dados recolhidos do Hokuyo, tendo como outputs um conjunto de pontos que representam os obstáculos que se encontram no ambiente, ${}^{W}\mathrm{O}$.

Internamente encontra-se dividida em dois módulos, um responsável pela geração de uma mapa dos obstáculos com *voxels* e outro responsável pela criação de *clusters*, denominados de Octomap e *Clustering* respetivamente.

O primeiro módulo, Octomap, é uma *framework* que pode ser integrada com a *framework* ROS e que permite gerar um mapa 3D do ambiente em tempo real quando combinadas. O Octomap quando combinado com o ROS, tem como *inputs* a nuvem de pontos do varrimento atual do Hokuyo, ${}^{B}_{L}\mathrm{P}$, a posição atual do veículo, ${}^{W}_{I}\mathrm{M}$, e a resolução máxima a atribuir à *Octree*. O primeiro parâmetro de entrada fornece uma nuvem de pontos 3D do varrimento efetuado ao ambiente em redor do veículo no atual instante de tempo. O segundo parâmetro de entrada, posição atual do veículo, é dada pelo *MAVROS* e pela transformação que existe entre o referencial do veículo e o referencial determinado pelo utilizador como sendo o referencial global. Este referencial pode ser a posição em que o veículo iniciou a manobra no cenário de busca e salvamento ou

então um ponto no referencial das coordenadas globais. Esta necessidade de definir um referencial permite construir e atualizar o mapa 3D com base na informação retirada a cada instante dos sensores. A resolução máxima que a *Octree* deve possuir permite obter um mapa com maior ou menor resolução e deve ser ajustada conforme a aplicação onde será implementada a framework Octomap. Na figura 4.4 é possível verificar um exemplo do output gerado pelo *Octomap* tendo em consideração as configurações anteriormente descritas e aplicado num cenário de busca e salvamento apresentado na figura 4.1.

O segundo módulo, *Clustering*, será apresentado no Capítulo 5 e tem como objetivo gerar *clusters*, ${}^{W}\mathrm{C}$, dos *voxels* provenientes do Octomap, ${}^{W}\mathrm{O}$, que indicam os obstáculos do ambiente em redor do veículo. A existência deste módulo deve-se à necessidade que a secção planeamento de trajetórias tem em receber os pontos extremos dos obstáculos e não um conjunto de pontos que indicam os centroides dos diversos *voxels* constituintes destes obstáculos. Desta forma será gerado um *cluster* por cada face de um dado obstáculo detetado no Octomap.



(a) (b)

Figura 4.4: Representação do Octomap para o ambiente de busca e salvamento.

4.1.3 Planeamento de Trajetórias

A terceira camada, denominada por Planeamento de Trajetória, consiste no planeamento da trajetória do veículo de forma a conseguir ir do ponto A para o B evitando os obstáculos. Nesta camada são analisados três métodos de planeamento de trajetórias, o primeiro método é o PRM (*Probabilistic Roadmap*) e foi descrito no Capítulo 2, os outros dois métodos foram desenvolvidos no âmbito da dissertação e encontram-se descritos no Capítulo 6 e denominam-se por GPRM (*Grid Probabilistic Roadmap*) e PPRM (*Particle Probabilistic Roadmap*).

O primeiro método a ser analisado é o PRM e como referido anteriormente neste

documento necessita de ter conhecimento da posição atual do veículo e da posição do *target*, sendo também necessária a informação relativa aos obstáculos e ao número de amostras a espalhar. Desta forma é possível determinar o trajeto mais curto desde a atual posição do veículo até à do *target*. Este método servirá de base para os dois outros métodos desenvolvidos de forma a ser possível retirar algumas conclusões das abordagens implementadas.

Posteriormente é explorado o método GPRM que foi desenvolvido nesta dissertação. Esta abordagem é baseada no método PRM e recorre a uma *grid* para espalhar as amostras, ao contrário do PRM em que efetua a amostragem em toda a área do mapa. Desta forma esta abordagem necessita como parâmetros de entrada a posição atual do veículo, a posição final, o número de células no eixo X-Y, o tamanho de cada célula e o número de amostras por célula para determinar um trajeto possível.

O terceiro método desenvolvido é o PPRM e também ele é baseado no método PRM no entanto ao contrário dos métodos expostos até ao momento, determina mais do que um trajeto possível sendo de seguida atribuído uma probabilidade a cada um dos trajetos. Posteriormente é verificado quais os trajetos que se encontram acima de uma dada probabilidade e desta forma retira-se o número de amostras que constituem estes trajetos, sendo estas amostras as que serão na próxima iteração espalhadas. Caso o número de amostras seja demasiadamente reduzido ou não se encontre nenhum trajeto com este número de amostras é então efetuado um *resample*. O método PPRM de planeamento de trajetórias tem um funcionamento similar ao filtro de partículas.

Estes métodos tem como parâmetros de entrada comuns a posição e atitude atual do veículo, ${}^{W}_{I}\mathrm{M}$, posição final e conjuntos dos pontos extremos dos *clusters*, ${}^{W}\mathrm{C}$ sendo o output desta camada a posição e atitude para onde o veículo se deve deslocar, ${}^{W}_{B}\mathrm{U}$. O output produzido, posição e atitude do próxima *waypoint* do *UAV*, deve-se ao facto de o sistema ser efetuado em tempo real, não necessitando de enviar para o veículo o conjunto de *waypoints* calculados para um determinado instante, permitindo desta forma que o *UAV* se desvie de qualquer obstáculo, estático ou móvel, ou até mesmo de obstáculos que até ao momento não tenha sido detetado.

Capítulo 5

Deteção e Clustering de Obstáculos

Neste capítulo irá ser detalhada a camada de Mapeamento apresentada no Capítulo 4 e encontra-se subdividida em dois módulos, Octomap e *Clustering*.

O módulo Octomap, e como já explicado previamente, é onde se gera o mapa do ambiente em redor do veículo, tendo como dados de saída os centroides dos *voxels* da *octree* que indicam os espaços ocupados, ou seja, os centroides dos diversos *voxels* que constituem cada obstáculo detetado no ambiente. Esta informação em conjunto com a posição do UAV é por si só suficiente para se proceder ao planeamento de trajetórias, no entanto o métodos de planeamento de trajetórias a serem analisados neste projeto requerem como informação de entrada os vértices de cada obstáculo, e não o conjunto de pontos constituintes desses obstáculos sendo necessário proceder à criação de *clusters*. Desta forma foi desenvolvido um algoritmo que permitisse gerar um *cluster* por cada face de um dado obstáculo, correspondendo ao segundo módulo, *Clustering*, da camada Mapeamento.

5.1 Método de *Clustering*

Como se pretende resolver o problema endereçado pela camada de Planeamento de Trajetórias onde é necessário receber os pontos extremos dos obstáculos e visto que o *output* gerado pelo módulo Octomap da camada Mapeamento é um conjunto de centroides dos *voxels* constituintes dos obstáculos torna-se necessário proceder à criação de *clusters* dos mesmos.

Para se gerar os *clusters* foi necessário desenvolver um algoritmo que permita recolher

a informação fornecida pelo módulo Octomap e de seguida identificar os obstáculos. Este algoritmo pretende identificar as faces dos objetos de forma independente, como consequência cada face de um objeto corresponderá a um obstáculo.
Os principais parâmetros que o algoritmo possui são a distância máxima que um dado *voxel* pode ser associado a um outro *voxel*, e o ângulo máximo permitido de forma a que os *voxels* possam ser considerados somente um *cluster*, ou seja se pertence à mesma face do obstáculo.

O método numa primeira fase vai analisar o vetor de pontos, referentes aos centroides dos *voxels* do *Octomap*, ${}^{W}\mathrm{O}$, verificando se estes centroides se encontram dentro do limites da distância, *dist_thresh*, e ângulo, *angle_thresh*, considerados para serem associados ao mesmo *cluster*, C. Este limites são sempre calculados face ao ponto seguinte do vetor, tal como é demonstrado na figura 5.1.

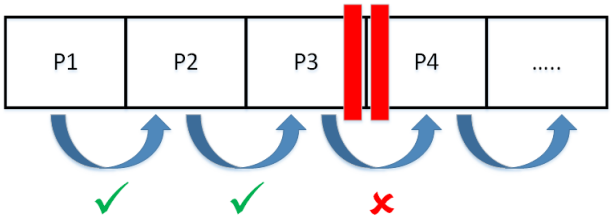


Figura 5.1: Análise do vetor de pontos do output do Octomap.

Na situação em que todos os pontos do vetor de pontos do output do Octomap, ${}^{W}\mathrm{O}$, se encontrem dentro dos limites, então todos os pontos do vetor serão considerados como um único *cluster*. No caso em que os limites de associação deixem de ser respeitados, então todos os pontos até ao ponto em que não foi respeitado os limites, serão considerados como um *cluster*, sendo os restantes pontos analisados numa próxima fase onde também serão verificados os limites de associação para serem constituintes de um novo *cluster*. Esta validação dos limites é efetuada para a criação dos primeiros dois *clusters* que o algoritmo gerar, ou seja é efetuado esta etapa duas vezes.

No fim destes dois ciclos de verificação e se ainda existir pontos no vetor ${}^{W}\mathrm{O}$, por atribuir a um *cluster*, então é necessário voltar a percorrer o vetor. No entanto a partir deste instante é importante saber se a interrupção gerada por não se encontrar nos limites, foi efetuada pela distância ou pelo ângulo, pois conforme seja um ou outro ter-se-á diferentes formas de abordar a criação de *clusters*.

Caso a interrupção anterior tenha sido efetuada pela distância, é verificado se o ponto

que ainda não se encontra num *cluster*, pode ou não ser adicionado a um já existente. Se for esse o caso é verificado se os próximos pontos do vetor também se encontram dentro dos limites face a este ponto do vetor e adiciona os pontos que se encontram dentro dos limites ao *cluster* já existente. Caso o primeiro ponto do vetor de pontos, ${}^{W}\mathrm{O}$, não possa ser adicionado a um *cluster* já existente é gerado um novo *cluster* onde este ponto e todos os próximos que se encontrem dentro dos limites, serão adicionados.

Na situação em que interrupção anterior da validação dos limites de associação tenha sido provocada pelo ângulo se encontrar fora dos limites, então verifica se os próximos pontos e adiciona todos os que se encontrarem dentro dos limites ao novo *cluster*. Este processo de verificar se a interrupção prévia foi gerada pela distância ou pelo ângulo e consequentes ações são efetuadas enquanto existir pontos do ${}^{W}\mathrm{O}$ que não tenha sido atribuído a um *cluster*.

De seguida elimina-se todos os *clusters* cujo o número de pontos constituintes seja inferior a um valor pré-definido, sendo posteriormente determinado para cada *cluster* o respetivo centroide, C_C, e pontos extremos, ${}^{W}\mathrm{C}$, sendo este último o *output* do algoritmo desenvolvido.

De forma a possibilitar uma forma mais fácil de interpretação do método desenvolvido é apresentado no algoritmo 3 um fluxograma relativo a este método, onde *prev_pts_dist* corresponde à verificação se a interrupção anterior se deveu à Distância por esta se encontrar fora dos limites, *prev_pts_angle* corresponde à verificação da interrupção anterior foi gerada pelo ângulo se encontra fora dos limites de associação e *next_pts_add* é a verificação se os próximos pontos do vetor de pontos, ${}^{W}\mathrm{O}$, podem ser adicionados ao atual *cluster*.

Algorithm 2 $[{}^{W}\mathrm{O}, C, num\_clust] \leftarrow \text{init\_Clustering}({}^{W}\mathrm{O}, dist\_thresh, angle\_thresh)$

```
for num_clust ← 1 to 2 do
  for ᵂo ∈ ᵂO do
    if Dist < dist_thresh && Angle < angle_thresh then
      C(num_clust) ← ᵂo
    else
      Sai do ciclo dos pontos
    end if
  end for
  num_clust ← num_clust + 1
end for
return [ᵂO, C, num_clust]
```

Algorithm 3 $({}^{W}\mathrm{C}) \leftarrow \mathrm{Clustering}\big({}^{W}\mathrm{O}, dist\_thresh, angle\_thresh\big)$

$[{}^{W}\mathrm{O}, C, num\_clust] \leftarrow \mathrm{init\_Clustering}\big({}^{W}\mathrm{O}, dist\_thresh, angle\_thresh\big)$
while ${}^{W}\mathrm{O} \neq \emptyset$ **do**
  for ${}^{W}\mathrm{o} \in {}^{W}\mathrm{O}$ **do**
    if prev_pts_dist **then**
      $add\_clust \leftarrow vC({}^{W}\mathrm{o}, C)$
      if $add\_clust == 1$ **then**
        $C(num\_clust) \leftarrow {}^{W}\mathrm{o}$
      else
        $num\_clust \leftarrow num\_clust + 1$
        $C(num\_clust) \leftarrow {}^{W}\mathrm{o}$
      end if
      if next_pts_add **then**
        $C \leftarrow add\_points(C(num\_clust))$
      else
        Sai do ciclo dos pontos
      end if
    else
      if prev_pts_angle **then**
        $num\_clust \leftarrow num\_clust + 1$
        $C(num\_clust) \leftarrow {}^{W}\mathrm{o}$
        if next_pts_add **then**
          $C \leftarrow add\_points(C(num\_clust))$
        else
          Sai do ciclo dos pontos
        end if
      end if
    end if
  end for
end while
$[C_C, {}^{W}\mathrm{C}] = Centroids\_ExtremePoints(C)$
return $({}^{W}\mathrm{C})$

5.1.1 Deteção de Cantos

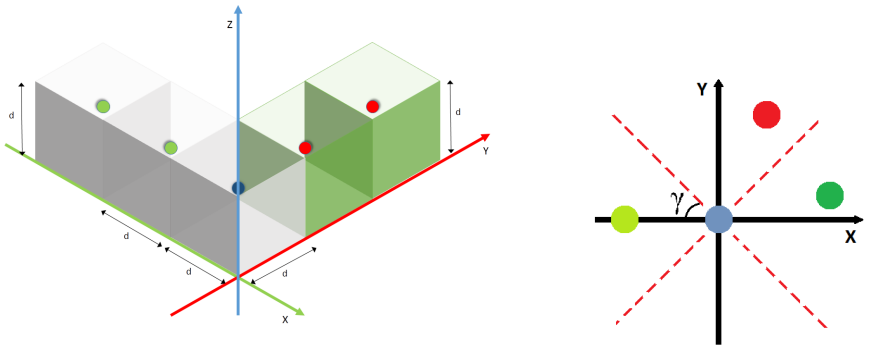
Como referido anteriormente para os pontos serem considerados do mesmo *cluster* necessitam de estar de acordo com os limites de distância e ângulo impostos. Para verificar o limite da distância necessita-se somente de verificar se esta se encontra abaixo de um dado *threshold* de forma a poder se encontrar dentro do limite da distância. No caso do limite do ângulo é utilizado os pontos anterior, $Point_{i-1}$, e seguinte, $Point_{i+1}$, ao ponto que se vai analisar, $Point_i$. Para o cálculo dos ângulos pode-se aplicar a equação 5.1, onde $Point_a$ toma o valor de $Point_{i-1}$ quando se efetua a validação para o ponto anterior. Quando se pretende efetuar o calculo relativamente ao ponto seguinte $Point_a$ toma o valor de $Point_{i+1}$.

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \arctan\left(\dfrac{Point_a(Y) - Point_i(Y)}{Point_a(Z) - Point_i(Z)}\right) \\ \arctan\left(\dfrac{Point_a(Z) - Point_i(Z)}{Point_a(X) - Point_i(X)}\right) \\ \arctan\left(\dfrac{Point_a(Y) - Point_i(Y)}{Point_a(X) - Point_i(X)}\right) \end{bmatrix} \tag{5.1}$$

Uma vez obtido o ângulo relativo ao ponto anterior e ao seguinte, é validado se o próximo ponto se encontra nos limites definidos na equação 5.2, onde *angle_thresh* é o valor máximo que se considera o ângulo nos limites, para ser considerado da mesma face, este valor pode ser definido pelo utilizador.

$$\gamma_{Point_{i-1}} - angle\_thresh <= \gamma_{Point_{i+1}} <= \gamma_{Point_{i-1}} + angle\_thresh \tag{5.2}$$

Na figura 5.2(b) é possível verificar uma explicação gráfica, onde o ponto amarelo corresponde ao ponto anterior, o ponto azul corresponde ao ponto atual, os pontos vermelhos e os verdes correspondem ao ponto seguinte. Como é possível verificar na figura 5.2(b) e respeitando a equação 5.2, o ponto vermelho encontra-se fora dos limites e os pontos verdes dentro dos limites. Na figura 5.2(a) encontra-se demonstrado os dados de entrada no método *Clustering*, ${}^{W}\mathrm{O}$.



(a) Exemplo de Deteção de Cantos.

(b) Deteção de cantos (ângulo).

Figura 5.2: Deteção de Cantos.

5.2 Casos de Estudo

Nesta secção irão ser apresentados os casos de estudo que foram implementados para a validação do módulo de mapeamento de obstáculos.

Para a sua implementação será utilizado o simulador MORSE, onde se irá escolher o cenário de aplicação e o veículo. O módulo de *clustering* foi desenvolvido e aplicado em *Matlab* com recurso à *framework* ROS permitindo assim que fosse implementado em tempo real e integrado com o simulador.

Para se proceder à demonstração prática do algoritmo desenvolvido foi implementada a arquitetura apresentada na figura 5.3.

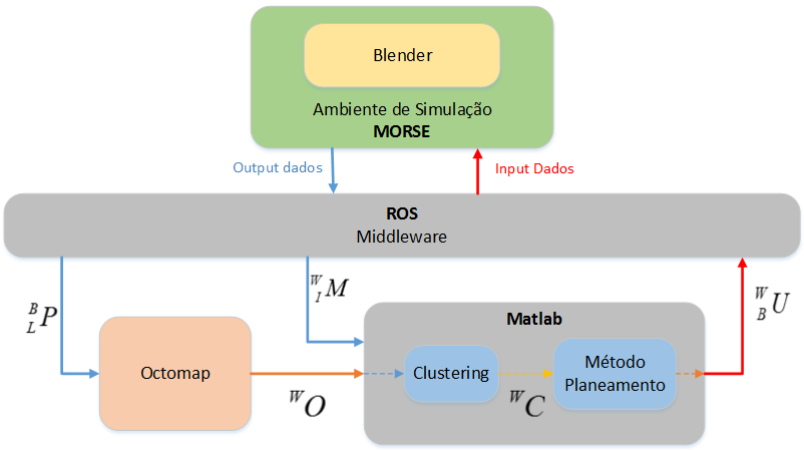


Figura 5.3: Arquitetura implementada para simulação do bloco de *Clustering* de obstáculos.

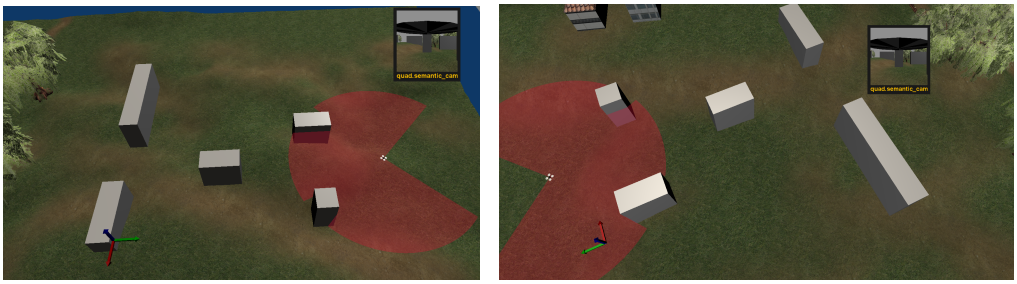


Figura 5.4: Ambiente Simulado no MORSE.

Como se pode averiguar foi utilizado o simulador MORSE, onde foi desenvolvido um ambiente de simulação na aplicação Blender, figura 5.4. Neste ambiente é possível constatar-se a existência de polígonos a servirem de obstáculos.

No ambiente de simulação foi adicionado o veículo aéreo quadrotor, que se encontra equipado com um sensor laser scanner Hokuyo, um IMU e um GPS. De forma a ser possível utilizar os dados dos sensores em tempo real, recorreu-se ao *middleware* ROS como interface entre o simulador e o bloco de processamento desenvolvido em *Matlab*. Numa primeira fase do processamento é utilizado a ferramenta Octomap, que possui interface com o ROS permitindo desta forma obter um mapa tridimensional tendo como entrada os dados do *Hokuyo*, a posição e atitude atual do veículo, valores estes obtidos no simulador. Outro parâmetro de entrada é o valor que define a máxima resolução da *Octree*, que foi establecido em 10 cm. Para realizar o *clustering* dos objetos recolhidos do ambiente de simulação é indispensável ter como *input* os dados de saída da *framework* Octomap, ${}^{W}\mathrm{O}$, podendo-se comprovar na figura 5.3. Esta informação concede os centroides referente aos *voxels* que se encontram ocupados, assim como as suas dimensões.

Na figura 5.5 encontra-se representado nas figuras da esquerda o mapa 3D gerado pelo Octomap usando os dados dos sensores do UAV em diversas iterações. Na direita é apresentado os resultados da implementação do algoritmo de *clustering* e na esquerda os dados do Octomap correspondentes. Para melhor perceção dos *clusters* gerados foi atribuído uma cor a cada um deles (até cinco *clusters*, cyan, vermelho, magenta, amarelo e verde), no entanto alguns casos, o número de *clusters* é elevado o que para esses casos é só apresentado cinco *clusters* e os restantes correspondem à cor azul escuro.

Como se pode comprovar através da figura 5.5, foi-se capaz de encontrar os diversos *clusters* para os obstáculos que se localizam no ambiente em redor do veiculo. Nas figuras da direita é possível constatar-se que os *clusters* obtidos conseguem identificar as diversas

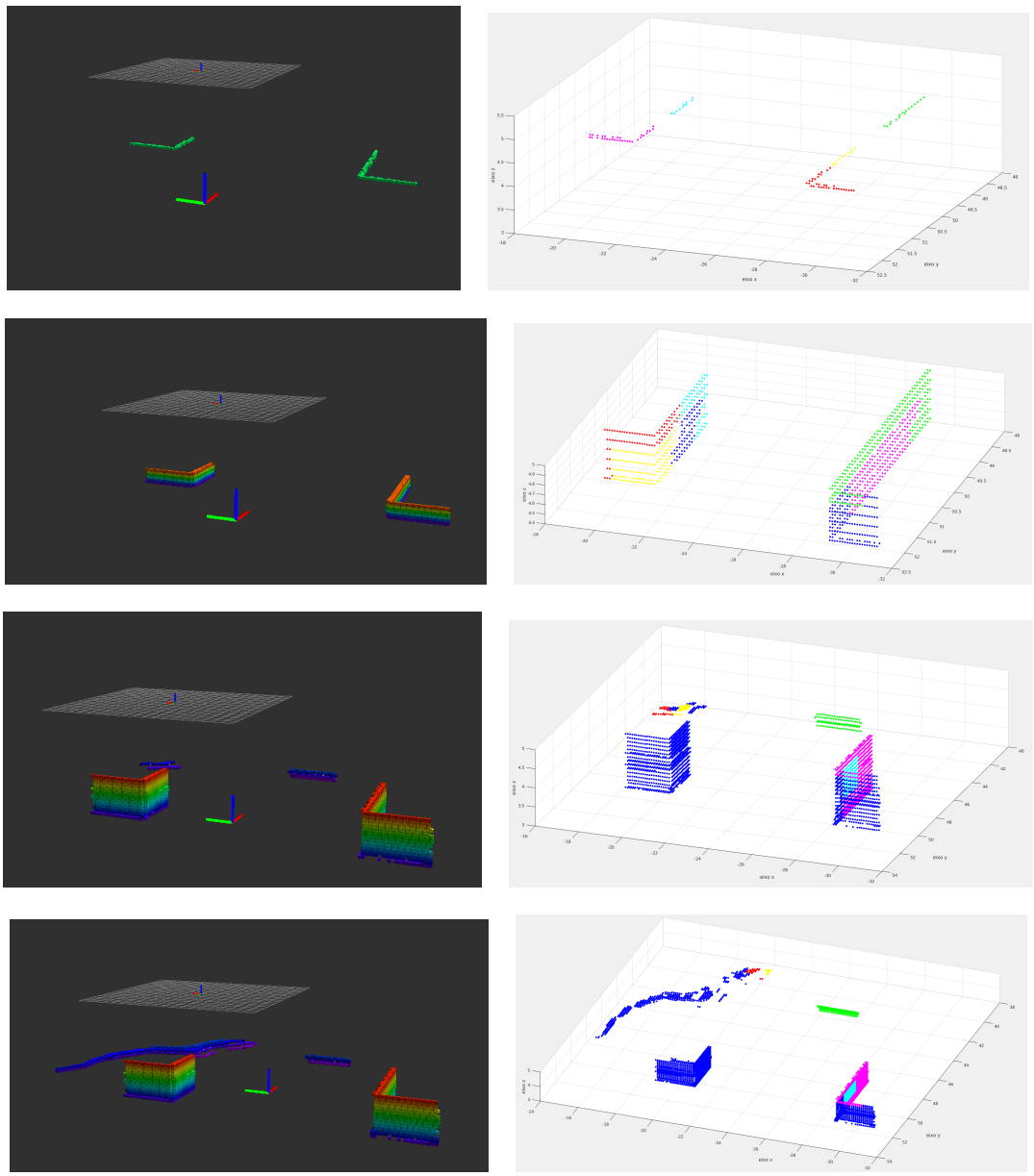


Figura 5.5: Mapa 3D gerado no Octomap e respetivos *clusters* para diferentes instantes da posição do veículo.

faces dos obstáculos correspondentes nas figuras da esquerda. Isto deve-se ao facto de existir agregação dos pontos disponibilizados pela *framework* Octomap. Tornando-se assim viável fornecer os dados necessários para a camada planeamento de trajetórias.

Capítulo 6

Planeamento de Trajetórias

Neste capítulo é descrito o bloco de planeamento de trajetórias responsável por gerar as trajetórias que um UAV deve efetuar de forma a conseguir ir de um ponto A até um ponto B evitando os obstáculos que se encontrem no seu redor. O veículo deve efetuar o planeamento de trajetórias em cenários de busca e salvamento, onde os veículos terrestres ou os humanos não conseguem ter acesso. Neste capítulo além das abordagens utilizadas serão também apresentados os resultados obtidos para os diversos métodos.

6.1 Métodos de Planeamento de Trajetórias

O método de planeamento de trajetórias PRM (*Probabilistic Roadmap*), que foi apresentado no estado da arte é detalhado nesta secção, sendo também descrito de uma forma aprofundada os métodos com que esta dissertação contribui, o GPRM (Grid Probabilistic Roadmap) e o PPRM (Particle Probabilistic Roadmap).

6.1.1 PRM - Probabilistic Roadmap

O método PRM (Probabilistic RoadMap) foi o método escolhido das diversas abordagens apresentadas no estado da arte, visto ser um método que permitia uma fácil alteração, apresentando maiores potencialidades de melhoria.

Este método é capaz de receber objetos tridimensionais e com isto fornecer uma trajetória que possibilite o movimento do ponto A para o B evitando colisões. Para tal o método, que pode ser descrito pelo Algoritmo 4, necessita num primeiro instante de determinar a área em que o veículo vai efetuar a navegação, eA, desta forma precisa de conhecer a posição atual em que o UAV se encontra e o ponto que pretende alcançar, x_{init} e x_{goal} respetivamente.

De seguida o método efetua um *sampling* das partículas, onde são espalhadas de forma aleatória o número de partículas que foram definidas, nS, pela área anteriormente calculada, eA. Posteriormente são eliminadas as partículas que se encontrem dentro dos obstáculos, ${}^{W}\mathrm{C}$, obtendo-se assim as *samples* válidas para se proceder ao planeamento de trajetórias, S.

Posteriormente o algoritmo constrói um *roadmap*, R, com as *samples* anteriormente obtidas, S, com a informação dos obstáculos, ${}^{W}\mathrm{C}$ de forma a poder verificar-se se existe possibilidade de colisão quando se tenta validar o trajeto entre duas *samples* aleatórias, sendo que uma *sample* pode ter n *samples* vizinhas, onde este valor é definido previamente pelo utilizador. Ainda no algoritmo 4, o V é um conjunto de vértices, onde $v \in V$ e $V \subset S$, e s é uma dada *sample* em que $s, s^{j} \in S$ e r é o raio para se poder considerar os vértices como vizinhos de uma *sample*, como apresentado em [69].

A última etapa consiste em determinar a trajetória mais curta, P, entre o ponto inicial e o final, sendo também necessário para este cálculo o *roadmap* previamente calculado, onde o segundo ponto do trajeto determinado, $P(2)$, será a próxima posição do veículo, ${}^{W}_{B}\mathrm{U}$.

Algorithm 4 $(P) \leftarrow \mathrm{PRM}\big(x_{init}, x_{goal}, nS,\ {}^{W}\mathrm{C}\big)$

eA $\leftarrow$ Environment_area(x_{init}, x_{goal})
S $\leftarrow$ Sampling$(nS, eA,\ {}^{W}\mathrm{C})$
{RoadMap$(S,\ {}^{W}\mathrm{C})$};
for each $s \in S$ **do**
  Nearest$(R = (V, eA, s) := argmin_{v \in V} \parallel s - v \parallel)$
  Near$(R = (V, eA), s, r) := \{v \in V : v \in ß_{s,r}\}$
  $CollisionFree(s, s^{j},\ {}^{W}\mathrm{C}) \in eA$
end for
P $\leftarrow$ Shortestpath(x_{init}, x_{goal}, R)
  {Dijkstra Algorithm};
return (P)

Este método é computacionalmente muito pesado para ser implementado em tempo real e não se encontra suficientemente otimizado. Necessita de conhecer previamente todo o mapa do ambiente de forma a obter o melhor trajeto até ao ponto destino, requerendo para tal de conhecimento prévio da posição atual do veículo e da posição final para a qual pretende deslocar. Esta abordagem não permite obter bons resultados em cenários como o de busca e salvamento, visto não permitir uma fácil adaptabilidade ao ambiente de aplicação. Nesse sentido é detalhado nas próximas secções os métodos desenvolvidos contributivos desta dissertação, GPRM e PPRM.

6.1.2 GPRM - Grid Probabilistic Roadmap

O método detalhado nesta secção é uma contribuição da dissertação e denomina-se por GPRM (Grid Probabilistic Roadmap). Este método foi desenvolvido tendo como base o algoritmo PRM e permite resolver alguns dos problemas que este algoritmo base possui. O algoritmo desenvolvido recorre a *grid* para poder gerar a sua trajetória não necessitando de conhecer previamente todo o mapa do ambiente em redor do veículo.

Este método, que se encontra representado no Algoritmo 5, parte da mesma base que o PRM, [69], e apresenta-se dividido também em quatro etapas:

- Determinação de área de navegação ($Environment\_area()$);
- *Sampling* ($Grid\_Sampling()$);
- Construção do *roadmap* ($RoadMap()$);
- Determinação do trajeto mais curto ($Shortestpath()$).

Algorithm 5 $(P) \leftarrow$ Grid PRM$(x_{init}, x_{goal}, nS, sizeG, numG, {}^{W}\mathrm{C})$

eA $\leftarrow$ Environment_area(x_{init}, x_{goal})
[S, eG] $\leftarrow$ Grid_Sampling$(nS, sizeG, numG, x_{init}, eA, {}^{W}\mathrm{C})$
{R $\leftarrow$ RoadMap$(S, {}^{W}\mathrm{C}, eG)$};
for each $s \in S$ **do**
  Nearest$(R = (V, eG, s) := argmin_{v \in V} \parallel s - v \parallel)$
  Near$(R = (V, eG), s, r) := \{v \in V : v \in ß_{s,r}\}$
  $CollisionFree(s, s^{j}, {}^{W}\mathrm{C}) \in eG$
end for
P $\leftarrow$ Shortestpath(x_{init}, x_{goal}, R)
  {Dijkstra Algorithm};
return (P)

A primeira etapa, que consiste na determinação da área de navegação, o algoritmo necessita de saber a posição atual e de destino do veículo, respetivamente x_{init} e x_{goal}, obtendo-se uma área do ambiente, eA, sendo esta função exatamente igual à do método PRM.

Na segunda etapa, para efetuar o *sampling* das partículas, é necessário recorrer a uma *grid* sendo fundamental definir o seu tamanho. Para que se possa gerar esta *grid* é indispensável definir o tamanho de cada célula da *grid*, $sizeG$, e o número de células que a *grid* vai ter nos eixos X e Y, $numG$, sendo representado na figura 6.1(a) por n e m respetivamente. Por último é essencial saber a posição atual do veículo de forma para

o centro da *grid* corresponda com esta posição. Definida a grelha é possível espalhar nas células o número de partículas definidas para cada célula, nS, sendo de seguida eliminadas as partículas que se encontrarem no interior de obstáculos, obtendo-se tal como no método PRM as *samples* válidas para o próximo passo, S. Além destas *samples*, esta etapa fornece também a área em que a *grid* se encontra no ambiente inicialmente determinado, eG, pode-se verificar a explicação desta etapa no Algoritmo 6.

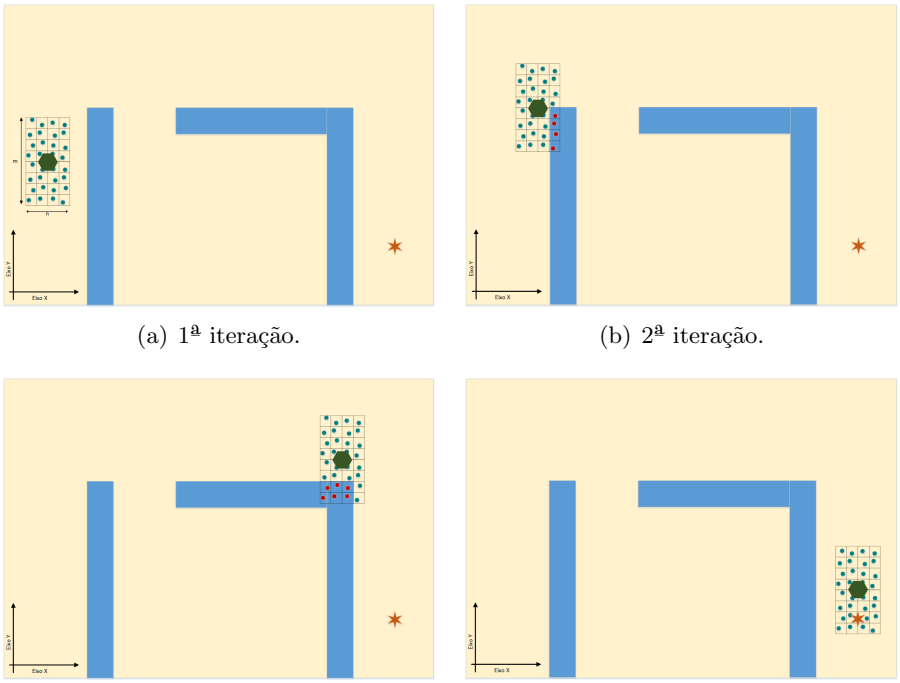
Algorithm 6 $[S, nG] \leftarrow$ Grid_Sampling$(nS, sizeG, numG, x_{init}, eA, {}^{W}\mathrm{C})$

```
eG ← Get_numCell(eA, sizeG, numG, x_init)
for eg ∈ eG(x) do
  for eg ∈ eG(y) do
    S ← Sampling_Cell(nS, eG, ^W C)
  end for
end for
return (S, eG)
```

Na terceira etapa, construção do *roadmap*, R, efetua-se os mesmos passos que no método PRM, ou seja, obter um mapa das *samples* que podem ser associadas sem que ocorra colisões, sendo o número de ligações possíveis pré-definidas, ou seja, o número de vizinhos que cada *sample* pode ter com conexões válidas é determinado previamente. No entanto os obstáculos só interferem na construção do mapa quando se encontram na área delimitada pela *grid*, ao contrário do que acontece no método PRM, em que os obstáculos afetam em toda a área de navegação. No algoritmo, V representa um conjunto de vértices, onde $v \in V$ e $V \subset S$, e s é uma dada *sample* em que $s, s^j \in S$ e r é o raio para se poder considerar os vértices como vizinhos de uma *sample*.

A última etapa, encontrar a trajetória mais curta, P, é efetuada da mesma forma que no PRM, necessitando para tal de recorrer ao algoritmo de Dijkstra, que precisa de conhecer o *roadmap* e as posições atual e final do veículo. De referir que o segundo ponto do trajeto determinado, $P(2)$, será a próxima posição do veículo, ${}^{W}_{B}\mathrm{U}$.

Na figura 6.1 é possível verificar uma demonstração gráfica do funcionamento do método ao longo das iterações até chegar ao ponto destino.



(a) 1ª iteração. (b) 2ª iteração.

(c) 3ª iteração. (d) 4ª iteração.

Figura 6.1: Exemplificação das etapas do GPRM.

6.1.3 PPRM - Particle Probabilistic Roadmap

A segunda contribuição desta dissertação é o método PPRM (Particle Probabilistic Roadmap) que se encontra detalhado nesta secção, este método foi desenvolvido tendo como base o algoritmo PRM, permitindo resolver alguns dos problemas que este último possui. O PPRM foi desenvolvido tendo em consideração a aplicação em tempo real para cenários que não se obtinha bons resultados usando o PRM. De uma forma geral este método pode ser descrito como uma mistura entre o método PRM e o filtro de Partículas.

Esta abordagem no primeiro instante de tempo funciona da mesma forma que o método PRM, no entanto ao invés de determinar somente o caminho mais curto, determina também n outros caminhos, P, mesmo não sendo ótimos. A cada um destes trajetos será atribuída uma probabilidade, P_p, sendo que o trajeto mais curto terá a maior probabilidade e o trajeto mais longo a menor probabilidade, sendo esta determinada através da equação 6.1.

$$P_p = 1 - \frac{err}{sum(err)} \tag{6.1}$$

onde err é a matriz dos erros, que pode ser obtida pela equação 6.2

$$err = dist(P_n) - dist_{min} \tag{6.2}$$

em que P_n é o trajeto n, $dist(P_n)$ é a distância total do trajeto n e pode ser obtida através da equação 6.3 e $dist_{min}$ é a mínima distância entre o ponto atual do veículo e o ponto destino sendo obtida através da equação 6.4.

$$dist(P_n) = sum(dist(P_{nj})) \tag{6.3}$$

onde j é um ponto dos que constituem o trajeto n.

$$dist_{min} = x_{goal} - x_{init} \tag{6.4}$$

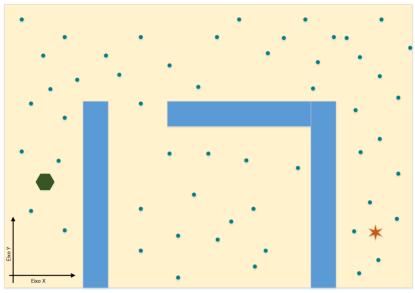
Após ser atribuída a probabilidade aos trajetos, é retirado o número de pontos total dos trajetos, nS_P, que possuam probabilidade superior a um dado *threshold*, $prob_{acept}$, sendo estes pontos o próximo número de partículas a ser considerado para a próxima iteração do método. Caso o número de partículas retirado seja inferior a um dado valor definido pelo utilizador é efetuado um *resample*, ou seja o número de partículas considerados para o método será o inicialmente atribuído, nS_{init}.

Com exceção da primeira iteração, se numa primeira fase, utilizando o número de partículas anteriores, não encontrar nenhum trajeto possível entre a posição atual do veículo e a de destino, é efetuado um *resample* onde irá novamente tentar encontrar pelo menos um trajeto possível.

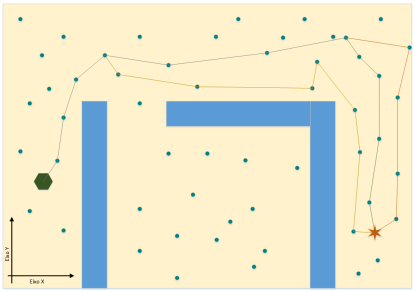
Neste método a posição para a qual o veículo se deve deslocar na iteração seguinte, ${}^{W}_{B}\mathrm{U}$, é dada pelo segundo ponto da menor trajetória calculada, $P_p(2)$.

No exemplo apresentado na figura 6.2 é possível verificar uma sequencia de iterações do método PPRM, onde na primeira iteração é espalhado o número de partículas máximas e calcula-se os possíveis trajetos para essa mesma iteração, figuras 6.2(a) e 6.2(b) respetivamente. Na segunda iteração, figura 6.2(c), é espalhado somente as partículas que na iteração anterior se encontravam nos trajetos calculados, sendo posteriormente determinado novamente os trajetos para esta nova iteração, figura 6.2(d). Na figura 6.2(e) encontra-se representada a situação em que o número de partículas espalhadas não permite determinar no mínimo um trajeto possível, devido ao seu reduzido número de partículas, necessitando assim de na iteração seguinte ser efetuado um

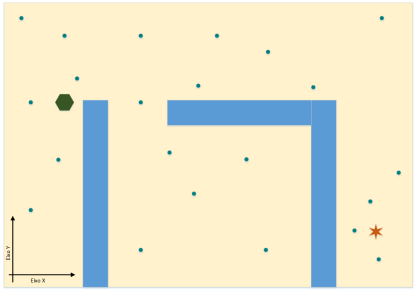
resample, recomeçando a sequencia exemplificada nestas figuras.



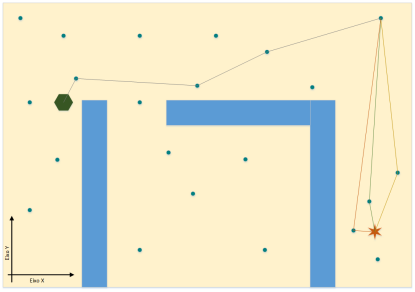
(a) 1ª iteração.



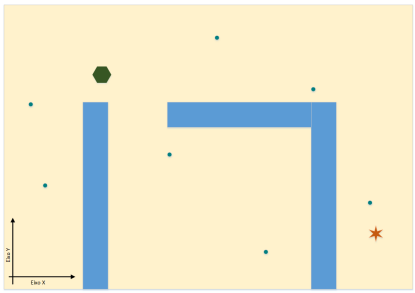
(b) 1ª iteração - Paths.



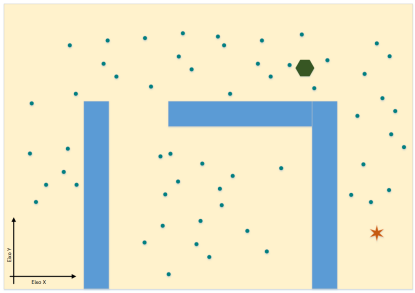
(c) 2ª iteração.



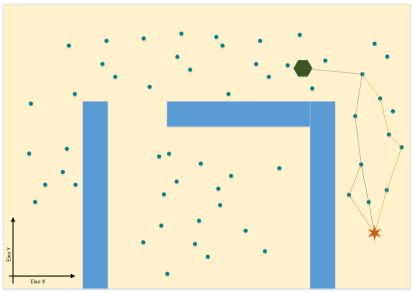
(d) 2ª iteração - Paths.



(e) 3ª iteração.



(f) nª iteração.



(g) nª iteração - Paths.

Figura 6.2: Etapas do PPRM.

Algorithm 7 $(P_p) \leftarrow \text{PPRM}(x_{init}, x_{goal}, nS_{init}, {}^{W}\text{C}, n\_Paths, prob_{acept})$

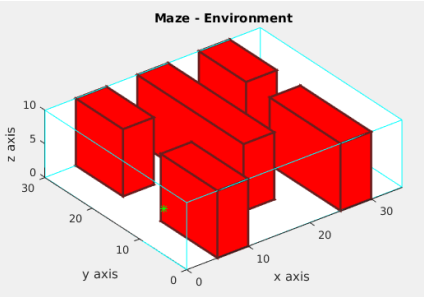
```
while x_init ≠ x_goal do
  eA ← Environment_area(x_init, x_goal)
  if 1ª iteração then
    S ← Sampling(nS_init, eA)
    R ← RoadMap(S, ^W C)
    P ← n_Shorthestpath(x_init, x_goal, R, S, n_Paths)
    P_p ← Add_probability(x_init, x_goal, P)
    nS_P ← Get_numParticles(P_p, prob_acept)
  else
    nS ← nS_P
    if nS < 1 then
      nS ← nS_init
    end if
    S ← Sampling(nS, eA)
    R ← RoadMap(S, ^W C)
    P ← n_Shorthestpath(x_init, x_goal, R, S, n_Paths)
    if P = ∅ then
      [nS_P, P_p] ← Resampling(nS ← nS_init)
    else
      P_p ← Add_probability(x_init, x_goal, P)
      nS_P ← Get_numParticles(P_p, prob_acept)
    end if
  end if
  x_init ← P_p(2)
end while
return P_p
```

6.2 Casos de Estudo

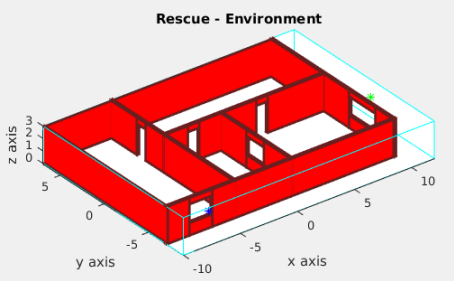
De forma a avaliar os dois métodos desenvolvidos na dissertação, foram gerados três cenários de teste, figura 6.3. Numa primeira fase procede-se à avaliação do método clássico de planeamento de trajetórias PRM no sentido de servir de base para comparação com os métodos desenvolvidos. Todas as simulações são efetuadas recorrendo ao simulador MORSE sendo adicionado ao ambiente de simulação um quadrotor equipado com um laser scanner Hokuyo, um IMU e um GPS. Estas serão executadas num computador com CPU Intel I7 4700HQ - 2.4Ghz, 8 Gb de RAM e com sistema operativo Linux Ubuntu 14.04 LTS 64 bits. Os resultados obtidos destas simulações são avaliados através da ferramenta *Matlab*.

Para esta simulação em tempo real de cada método apresentado e de forma a ser o mais semelhante à realidade, será usado somente o segundo ponto do conjunto de pontos da trajetória determinada pelos métodos, permitindo desta forma evitar possíveis novos obstáculos que venham a ser detetados pelo UAV. Será também apresentado para cada método os resultados das trajetórias obtidas, tempos e outras características que sejam importantes referentes a cada um.

(a) Window.



(b) Maze.



(c) Rescue.

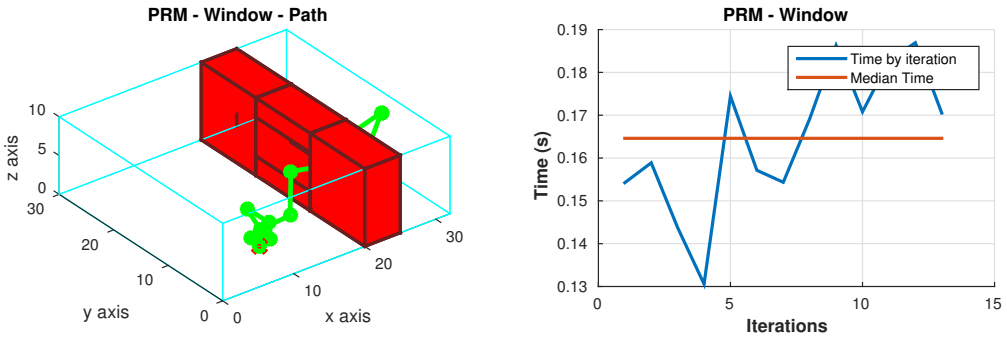
Figura 6.3: Cenários de aplicação utilizados para analisar a performance dos métodos de planeamento.

6.2.1 PRM - Probabilistic Roadmap

Na simulação do método PRM, e tendo em consideração a parte teórica previamente apresentada, necessita-se de ter em atenção os parâmetros referentes ao número de partículas, nS, e número de vizinhos que cada *sample* pode ter na criação do *roadmap*, pois estes possuem uma grande influência na forma como o método vai funcionar, bem como o tempo que vai necessitar para gerar uma trajetória.

Cenário *Window*

Para o ambiente de simulação *Window*, o método PRM foi simulado com um número de *samples* de 50 ($nS = 50$) e 10 vizinhos máximos por *sample* para a criação do *roadmap*. Com estes valores e com a posição inicial e final do veículo definidas, a simulação necessitou de 13 iterações para que o UAV chegasse do ponto inicial até ao final, tendo em média cada iteração demorado 0.164605 segundos, perfazendo um total de 2.13987 segundos necessários para este ambiente. Na figura 6.4(a) é apresentada a trajetória final que o veículo efetuou e na figura 6.4(b) os tempos que cada iteração necessitou para calcular as trajetórias, sendo também apresentado o tempo médio por iteração.



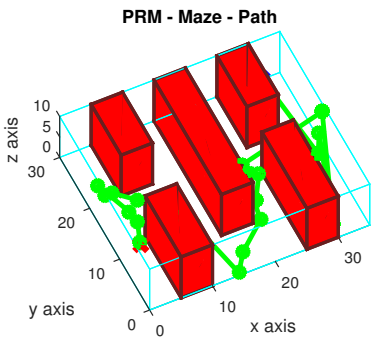
(a) PRM - Window - Path

(b) PRM - Window - Tempos

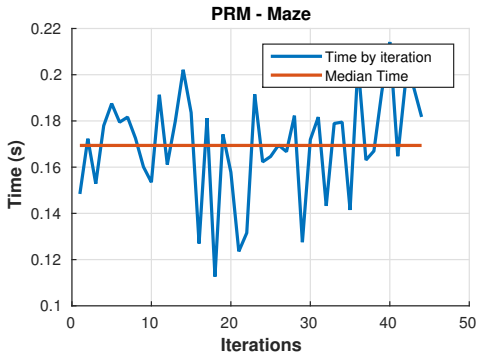
Figura 6.4: PRM - Simulação em Ambiente Window.

Cenário *Maze*

O método PRM quando aplicado no ambiente de simulação *Maze*, necessitou de um número de *samples* $nS = 50$ e um número de vizinhos por *sample* para a construção do *roadmap* de 15. Utilizando estes dados, obteve-se uma trajetória ao fim de 44 iterações num total de tempo de 7.453550 segundos gastando cada iteração uma média de 0.1693989 segundos. Na figura 6.5(a) é apresentado a trajetória efetuada pelo veículo para se deslocar do ponto inicial até ao final, enquanto que na figura 6.5(b) é apresentado os tempos que cada iteração necessitou para o cálculo da trajetória bem como o valor médio das mesmas.



(a) PRM - Maze - Path

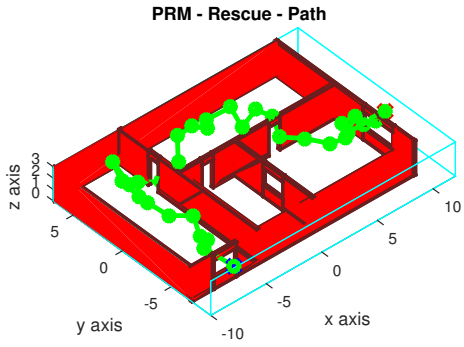


(b) PRM - Maze - Tempos

Figura 6.5: PRM - Simulação em Ambiente Maze.

Cenário *Rescue*

No ambiente de simulação *Rescue* e de forma a ser possível encontrar um trajeto que permitisse ao veículo ir do ponto inicial até ao final, foi definido um número total de *samples*, nS, igual a 400 e um número de vizinhos por *sample* de 15. Conseguindo assim o UAV chegar ao ponto final ao fim de 32 iterações necessitando de um total de 294.9778 segundos a uma média de 9.218056 segundos por iteração. Na figura 6.6 encontra-se representado o trajeto efetuado pelo veículo bem como os tempos de cada iteração.



(a) PRM - Rescue - Path

(b) PRM - Rescue - Tempos

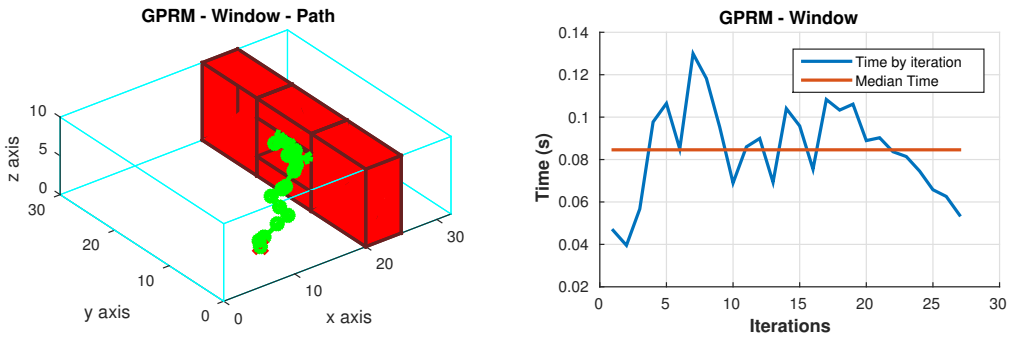
Figura 6.6: PRM - Simulação em Ambiente Rescue.

6.2.2 GPRM - Grid Probabilistic Roadmap

Para o método desenvolvido, GPRM (Grid Probabilistic Roadmap), é necessário ter em consideração o tamanho da grid onde vai ser aplicado o planeamento. Desta forma é necessário definir o mais corretamente possível o número de células nos eixos X e Y, bem como a dimensão que cada célula deve ter. Nas simulações efetuadas o valor atribuído à dimensão de cada célula foi sempre 1 metro, pois este valor corresponde ao diâmetro do UAV utilizado. É também necessário definir o número de *samples* a ser atribuído a cada célula. Tal como no método PRM, o parâmetro de definição do número de vizinhos que cada *sample* pode ter, é importante pois influencia no tempo de processamento do método.

Cenário *Window*

Para efetuar a simulação do método Grid PRM no ambiente *Window* foi considerado uma grid com 6 células no eixo X e 8 no eixo Y, sendo o tamanho de cada célula, como referido previamente, de 1 metro e o número de *samples* por célula de 1. O número de vizinhos considerado foi de 10. Desta forma o veículo para chegar ao ponto de destino, utilizando o método Grid PRM necessitou de 2.283838 segundos e 27 iterações a um valor médio por iteração de 0.08458659 segundos. Na figura 6.7(a) é apresentado o trajeto que o veículo efetuou e na figura 6.7(b) os tempos por cada iteração.



(a) Grid PRM - Window - Path

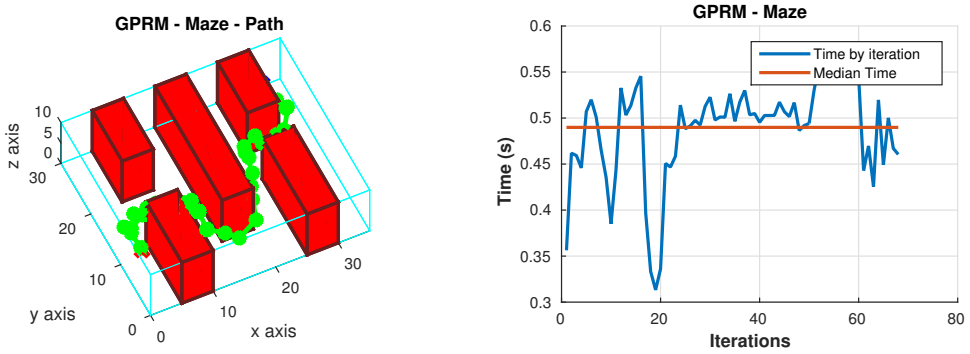
(b) Grid PRM - Window - Tempos

Figura 6.7: Grid PRM - Simulação em Ambiente Window.

Cenário *Maze*

No ambiente *Maze* foi considerado uma grid com 8 células no eixo X e 24 células no eixo Y, sendo atribuída a cada célula a quantidade de *samples* de 1. O número de vizinhos

considerados para a criação do *roadmap* foi de 15. Assim para percorrer o trajeto, figura 6.8(a), foram necessárias 68 iterações com um tempo total de 33.31085 segundos a uma média de 0.4898655 segundos por iteração, sendo possível verificar estes dados na figura 6.8(b).

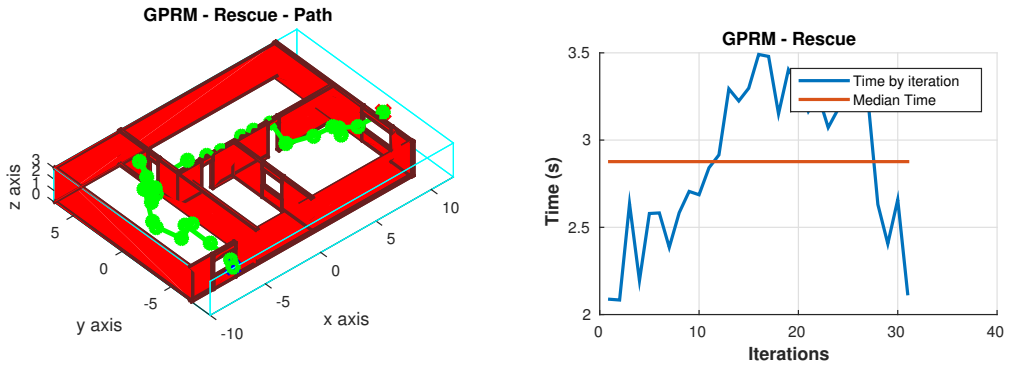


(a) Grid PRM - Maze - Path (b) Grid PRM - Maze - Tempos

Figura 6.8: Grid PRM - Simulação em Ambiente Maze.

Cenário *Rescue*

A trajetória obtida, figura 6.9(a), no ambiente de simulação *Rescue* foi conseguida ao fim de 31 iterações e de um tempo total de 89.17216 segundos a uma média de 2.876521 segundos por iteração, figura 6.9(b). Para tal contribuiu o facto de ter uma grid com 12 células no eixo X e 24 no eixo Y, sendo que cada célula possuía uma *sample* que por sua vez podia ter até 15 *samples* vizinhas.



(a) Grid PRM - Rescue - Path (b) Grid PRM - Rescue - Tempos

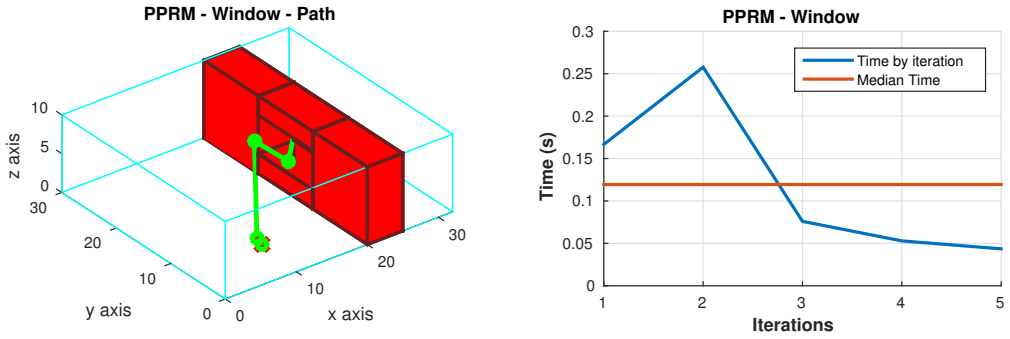
Figura 6.9: Grid PRM - Simulação em Ambiente Rescue.

6.2.3 PPRM - Particle Probabilistic Roadmap

O método PPRM foi desenvolvido de forma a poder colmatar alguns dos problemas que existem no método tradicional PRM. Para a utilização do método PPRM, é necessário ter em consideração um conjunto de parâmetros que carecem de uma inicialização, sendo estes o número de partículas iniciais a espalhar, o número de vizinhos máximo que cada *sample* pode possuir, o número de trajetos a calcular e a probabilidade que um trajeto necessita de deter de forma a poder ser aceite como uma boa opção.

Cenário *Window*

Tendo em consideração os parâmetros necessários para o método PPRM, no ambiente de simulação *Window*, foi definido um número de *samples* iniciais igual a 50 e o número máximo de vizinhos igual a 10. Pretende-se que o método calcule 2 trajetos e que para esses trajetos serem aceites tem de possuir uma probabilidade de 50%. Tendo em consideração estas configurações, o tempo necessário para o veículo chegar ao destino é de 0.5967020 segundos e de 5 iterações com uma média de tempo de 0.1193404 segundos cada. Na figura 6.10 é apresentado o trajeto final do UAV e os tempos necessários por cada iteração.



(a) PPRM - Window - Path

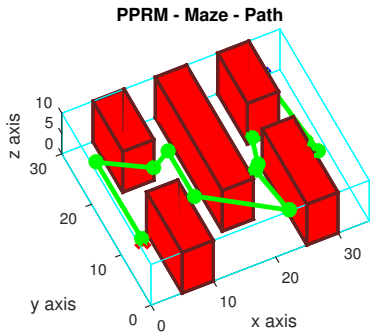
(b) PPRM - Window - Tempos

Figura 6.10: PPRM - Simulação em Ambiente Window.

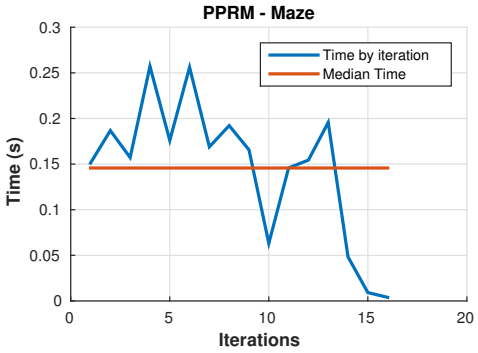
Cenário *Maze*

Para o ambiente Maze foi definido que o número de samples inicial é de 50, que o máximo de vizinhos por sample é de 15, que o método necessita de determinar até 2 trajetos que liguem do ponto inicial até ao ponto final e que a probabilidade deles seres aceites é de 50%. Desta forma obteve-se o trajeto final, figura 6.11(a), ao fim de 16

iterações cujo tempo total é de 2.331394 segundos com uma média de tempo por iteração de 0.1457121 segundos, figura 6.11(b).



(a) PPRM - Maze - Path

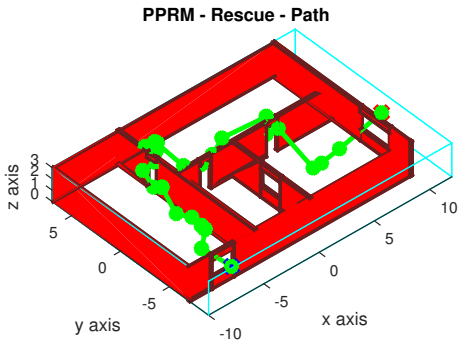


(b) PPRM - Maze - Tempos

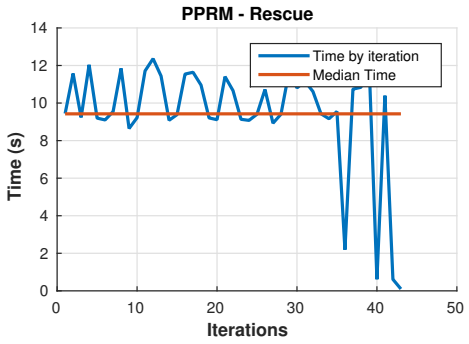
Figura 6.11: PPRM - Simulação em Ambiente Maze.

Cenário *Rescue*

Para o ambiente *Rescue* e forma a ser possível encontrar pelo menos um trajeto até ao ponto de destino foi utilizado uma quantidade de *samples* inicial de 400, com um numero de vizinhos máximos de 15, sendo que o método necessita de determinar até 2 trajetos e a probabilidade para serem aceites é de 25%. Desta forma consegue-se determinar um trajeto final, figura 6.12(a), ao fim de 43 iterações, com o tempo total de 405.2791 segundos com uma média de tempo por iteração de 9.425096 segundos, figura 6.12(b).



(a) PPRM - Rescue - Path



(b) PPRM - Rescue - Tempos

Figura 6.12: PPRM - Simulação em Ambiente Rescue.

6.3 Comparação de Métodos

Para uma melhor perceção dos resultados obtidos por cada método para os diversos cenários será apresentado um conjunto de comparações tendo como referência o método PRM, visto que os outros métodos foram desenvolvidos baseados neste. Nestas comparações será dado ênfase aos parâmetros configurados relativos aos diversos métodos.

6.3.1 Comparação PRM - GPRM

A primeira comparação a ser efetuada é entre o método já existente PRM e o método desenvolvido Grid PRM para os diversos cenários.

No cenário *Window* e inicializando a comparação entre métodos pode-se verificar na figura 6.13 e na tabela 6.1 que o método PRM necessitou de um menor tempo para que o veículo conseguisse ir do ponto inicial até ao final e consequentemente um menor número de iterações, mesmo possuindo um número de *samples* semelhante, 50 *samples* contra as 48 do método GPRM, sendo este valor obtido através da número total de células da grid multiplicado pelo numero de *samples* de cada célula. No entanto como se pode verificar na tabela, o método GPRM necessita por iteração de um tempo médio inferior ao método PRM, o que faz com que o seu custo computacional seja inferior, outra vantagem deste método face ao PRM é o facto de produzir uma trajetória com maior detalhe e mais direcional para o ponto destino, figura 6.7(a), ou seja, o veículo vai-se mover uma menor distância entre cada *waypoint*, o que num cenário em que possua múltiplos obstáculos poderá permitir efetuar um contorno mais perfeito. Na situação em que o valor do tamanho da célula é maior que o atual e mantendo os restantes parâmetros, o número de iterações necessárias para calcular o trajeto final será menor, consequentemente o tempo final necessário para efetuar o calculo também será inferior.

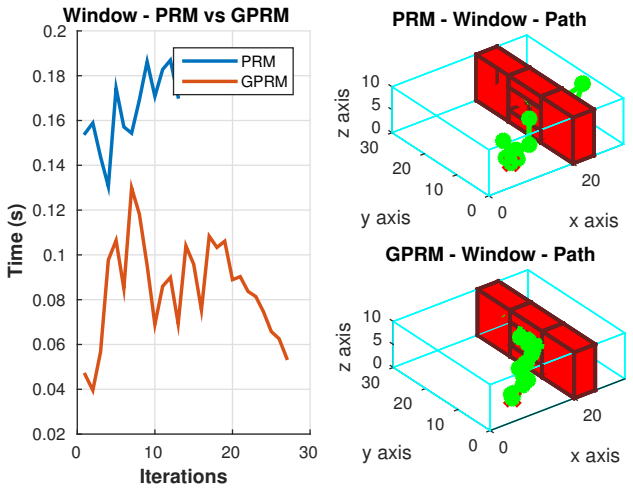


Figura 6.13: Comparação PRM - GPRM tempos e trajetos no ambiente window

No ambiente *Maze*, o método GPRM possui um tempo muito superior ao método PRM, figura 6.14 e tabela 6.1, no entanto para este cenário necessita de 192 *samples* ao contrário do PRM que só necessita de 50 samples, no entanto como pode ser comprovado pelas figuras 6.14, 6.8(a) e 6.5(a) o trajeto efetuado pelo método Grid PRM muito melhor que o efetuado pelo método PRM. Nesta situação e caso a dimensão atribuída à célula fosse maior o tamanho da grid poderia ser reduzido, ou seja, o número de células a considerar para os eixos X - Y poderia ser inferior.

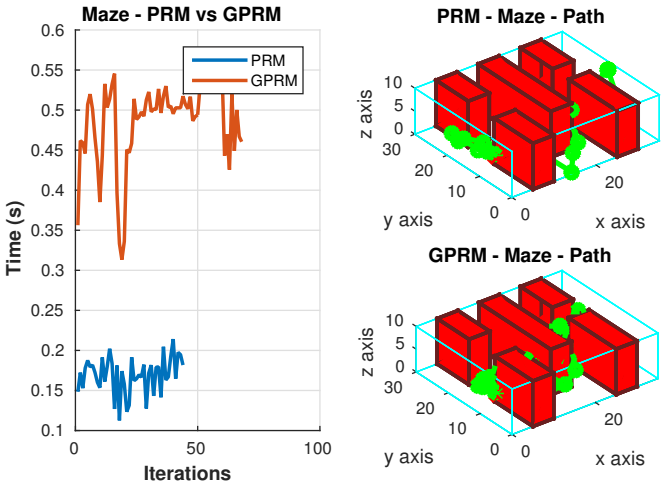


Figura 6.14: Comparação PRM - GPRM tempos e trajetos no ambiente maze

Comparando os métodos no cenário *Rescue* pode-se verificar através da figura 6.15 e da tabela 6.1 que o método desenvolvido, GPRM, é de longe melhor que o método PRM, pois necessita de um tempo total para o trajeto final do veículo de apenas 89.17216 segundos o que para semelhante número de iterações verificado implica um tempo médio por iteração de 2.876521, o que a isto se junta o facto de necessitar de menor número de samples. Além destes resultados e se verificar o trajeto final determinado com o uso do método Grid PRM para o atual cenário, figura 6.15, pode-se verificar que a característica de produzir trajetos de elevados pormenores e sem grande dispersão no sentido do ponto destino.

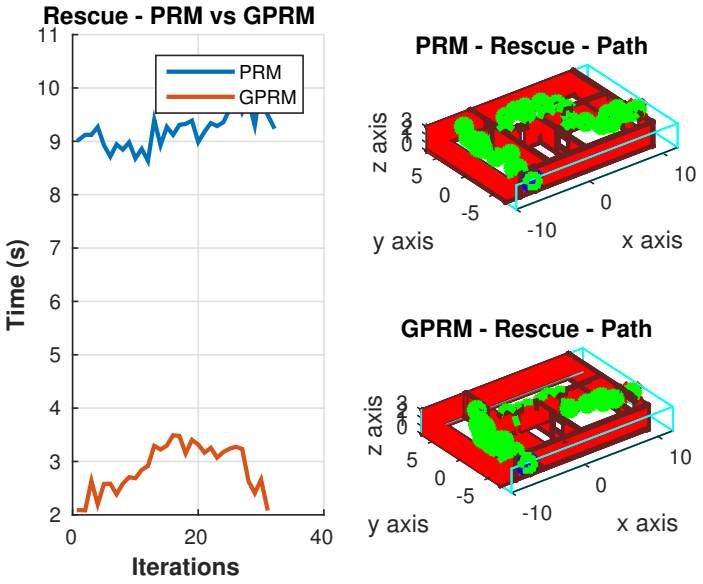


Figura 6.15: Comparação PRM - GPRM tempos e trajetos no ambiente rescue

Desta forma pode-se concluir que o método GPRM para cenários com uma maior complexidade necessita de um menor tempo para determinar o trajeto final, isto numa aplicação em tempo real, e como consequência o tempo médio por iteração será menor, aliando-se a isto o número de *samples* necessárias são mais menores que o método PRM, no entanto em cenários com menor complexidade necessita de um maior número de *samples* total que o metodo PRM o que vai implicar um tempo maior para processamento. Outro aspeto importante a reter é o facto de ser necessário definir o tamanho da *grid* conforme o cenário a aplicar, sendo que uma boa definição pode implicar um menor tempo de processamento, a este parâmetro deve-se também ter em atenção número de

Tabela 6.1: Comparação PRM - GPRM

	PRM			GPRM		
	Window	Maze	Rescue	Window	Maze	Rescue
Tempo Médio (s)	0.1646054	0.1693989	9.218056	0.08458659	0.4898655	2.87651
Tempo Total (s)	2.139870	7.453550	294.9778	2.283838	33.31085	89.17216
Nº Iterações	13	44	32	27	68	31
Nº Samples	50	50	400	1	1	1
Nº Vizinhos	10	15	15	10	15	15
Tam. da Grid (X*Y)	-	-	-	6*8	8*24	12*24
Tamanho Célula (m)	-	-		1	1	1
Dist. Percorrida (m)	64.96613	143.1229	47.62949	51.73965	77.71683	44.23538

samples a aplicar a cada célula, estando este valor também interligado com o tamanho da célula. Outra vantagem que este método possui é o facto de para qualquer cenário calcular trajetos de elevada qualidade e sem grande dispersão no sentido do ponto destino e percorrer um menor de metros que o método PRM.

6.3.2 Comparação PRM - PPRM

Para se conseguir encontrar e compreender de uma melhor forma os pontos fortes e fracos do método desenvolvido PPRM é necessário proceder à comparação do mesmo com o método já desenvolvido PRM para os três cenários de simulação anteriormente apresentados.

No ambiente de simulação *Window* é possível verificar através da figura 6.16 e da tabela 6.2 que o método desenvolvido, PPRM, necessita de um menor tempo e de um menor número de iterações para determinar o trajeto final do veículo, como consequência o tempo médio das iterações será inferior comparando com o método PRM. De salientar que estes resultados se obtém considerando o mesmo número de *samples* e de vizinhos, sendo que o método desenvolvido só vai procurar por 2 trajetos possíveis e a probabilidade de eles serem aceites tem de ser superior a 50%.

Quando aplicado os métodos PPRM e PRM no ambiente de simulação *Maze*, verifica-se que o tempo de processamento necessário para que o veículo chegue ao ponto final é de 2.331394 segundos quando aplicado o método PPRM sendo este valor muito inferior ao apresentado pelo método PRM, figura 6.17 e tabela 6.2. Além do tempo total gasto, verifica-se que o número de iterações necessárias para encontrar o trajeto apresentadas pelo PPRM é muito inferior ao valor apresentado pelo método PRM. Tal como a simulação no ambiente *Window*, o número de trajetos que o método PPRM necessita de

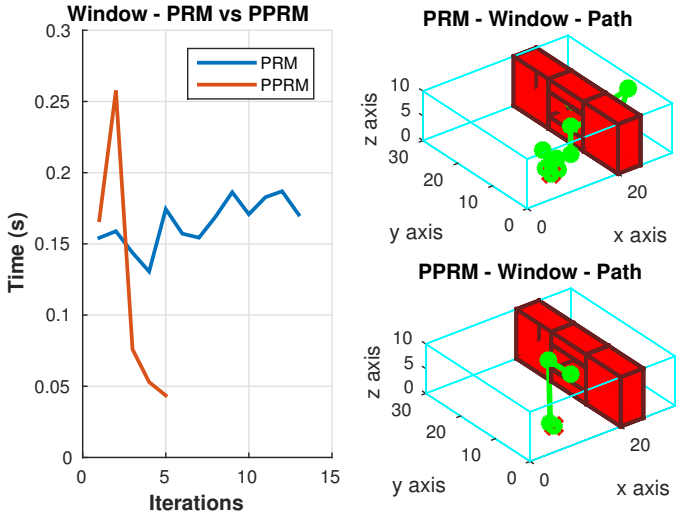


Figura 6.16: Comparação PRM - PPRM tempos e trajetos no ambiente window

determinar é dois, sendo que a probabilidade deles serem aceites é superior a 50%.

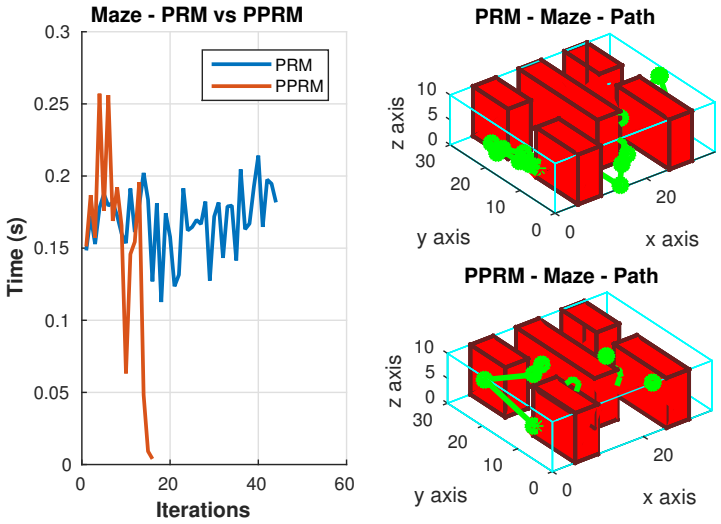


Figura 6.17: Comparação PRM - PPRM tempos e trajetos no ambiente maze

No último cenário de simulação apresentado, ambiente *Rescue*, o tempo total gasto pelo método desenvolvido PPRM, é muito superior ao apresentado pelo método PRM, isto deve-se ao facto de o método necessitar de efetuar diversas vezes *resample* devido

ao facto de ser mais difícil de encontrar caminhos, mesmo após se ter diminuído a probabilidade de um dado trajeto ser aceite. Além de possuir um tempo superior ao PRM, o método PPRM necessita de mais iterações para que o veículo chegue ao destino tendo os dois métodos definido o mesmo número de partículas, 400. Estes dados podem ser verificados na figura 6.18 e na tabela 6.2.

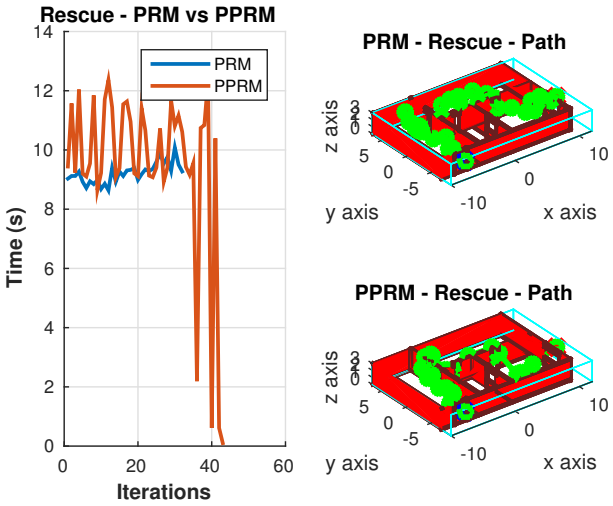


Figura 6.18: Comparação PRM - PPRM tempos e trajetos no ambiente Rescue

Assim é possível concluir que o método desenvolvido tem um melhor comportamento para ambientes com menor complexidades. Verifica-se também que o custo de ser efetuado o *resample* pode compensar dependendo dos cenários em que o método é aplicado, sendo que para cada cenário deve ser adaptado o número de trajetos que o método deve calcular bem como o limite mínimo da probabilidade que cada um desses trajetos deve possuir para que possa ser aceite. O método desenvolvido, PPRM em todos os cenários apresenta um valor da distância percorrida inferior ao método PRM.

Tabela 6.2: Comparação PRM - PPRM

	PRM			PPRM		
	Window	Maze	Rescue	Window	Maze	Rescue
Tempo Médio(s)	0.1646054	0.1693989	9.218056	0.1193404	0.1457121	9.425096
Tempo Total (s)	2.139870	7.453550	294.9778	0.596702	2.331394	405.2791
Nº Iterações	13	44	32	5	16	43
Nº Samples	50	50	400	50	50	400
Nº Vizinhos	10	15	15	10	15	15
Nº Trajetos	-	-	-	2	2	2
Probabilidade Aceitar	-	-		50%	50%	25%
Dist. Percorrida (m)	64.96613	143.1229	47.62949	40.76210	97.30443	40.90566

6.3.3 Comparação PRM - GPRM - PPRM

A última comparação apresentada neste documento é relativa aos dois métodos desenvolvidos, GPRM e PPRM e o método onde estes foram baseados, PRM. Estes métodos vão ser comparados entre si para os três cenários previamente apresentados.

Efetuando uma comparação entre os três métodos para o ambiente de simulação *Window* é possível verificar através dos dados apresentados na figura 6.19 e na tabela 6.3 que o método que melhor resultados apresenta para este cenário é o método desenvolvido PPRM. Esta conclusão tem como base os tempos totais necessários para o cálculo da trajetória em conjunto com os parâmetros que necessitam de ser configurados.

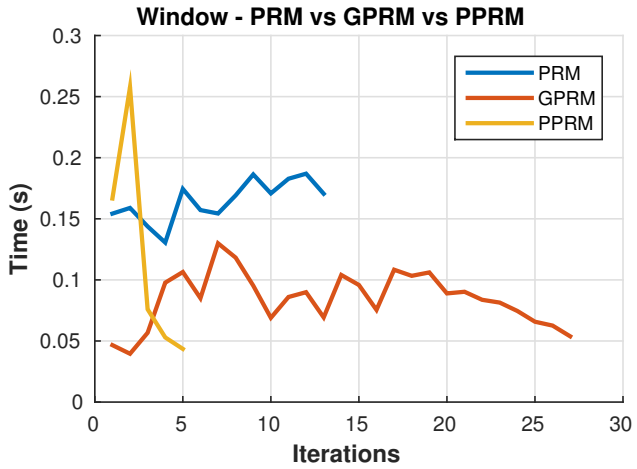


Figura 6.19: Comparação PRM - GPRM - PPRM no ambiente Window.

Tabela 6.3: Window - Comparação entre métodos PRM, GPRM e PPRM

	PRM	GPRM	PPRM
Tempo Médio (s)	0.1646054	0.08458659	0.1193404
Tempo Total (s)	2.139870	2.283838	0.596702
Nº Iterações	13	27	5
Nº Samples	50	1	50
Nº de Vizinhos	10	10	10
Tamanho Grid (X*Y)	-	6*8	-
Tamanho Célula (m)	-	1	-
Nº Trajetos	1	1	2
Probabilidade Aceitar	-	-	50%
Distancia Percorrida (m)	64.96613	51.73965	40.76210

Quando se efetua uma comparação dos três métodos no cenário Maze é possível verificar que o método PPRM volta a ser o mais rápido a determinar uma trajetória final em tempo real para o veículo, tal como se pode comprovar a partir da figura 6.20 e da tabela 6.4. No entanto o método que percorre uma menor distância é o Grid PRM, podendo-se dizer que o seu cálculo de planeamento de trajetória é mais eficaz que os restantes.

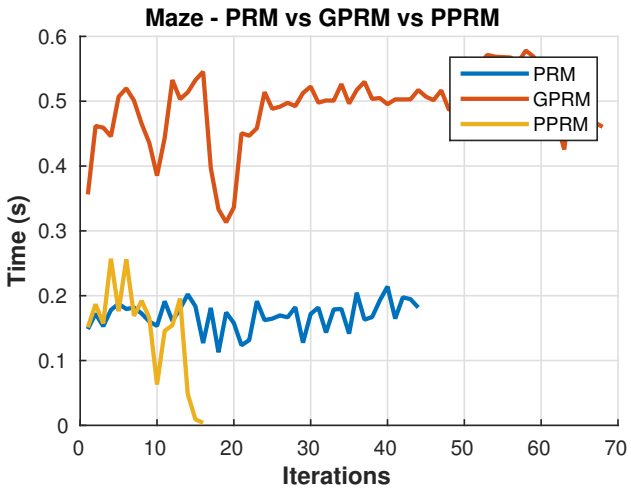


Figura 6.20: Comparação PRM - GPRM - PPRM no ambiente Maze.

Tabela 6.4: Maze - Comparação entre métodos PRM, GPRM e PPRM

	PRM	GPRM	PPRM
Tempo Médio (s)	0.1693989	0.4898655	0.1457121
Tempo Total (s)	7.453550	33.31085	2.331394
Nº Iterações	44	68	16
Nº Samples	50	1	50
Nº de Vizinhos	15	15	15
Tamanho Grid (X*Y)	-	8*24	-
Tamanho Célula (m)	-	1	-
Nº Trajetos	1	1	2
Probabilidade Aceitar	-	-	50%
Distancia Percorrida (m)	143.1229	77.71683	97.30443

No último cenário de simulação e tendo em consideração os dados da tabela 6.5 e da figura 6.21 o método que mais rapidamente conseguiu determinar um trajeto em tempo real foi o Grid PRM gerando um trajeto sem muita dispersão.

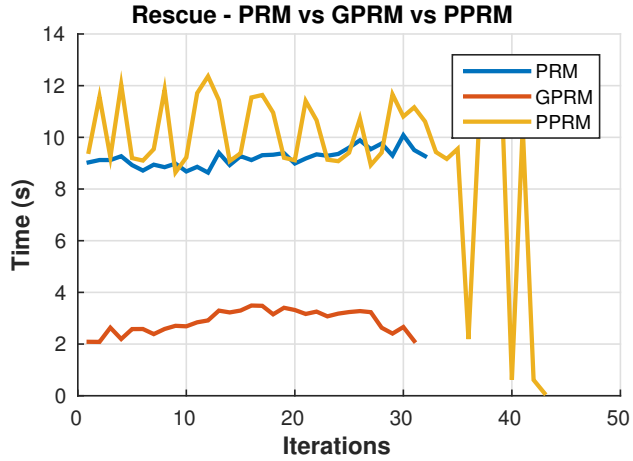


Figura 6.21: Comparação PRM - GPRM - PPRM no ambiente Rescue.

Tabela 6.5: Rescue - Comparação entre métodos PRM, GPRM e PPRM

	PRM	GPRM	PPRM
Tempo Médio (s)	9.218056	2.87651	9.425096
Tempo Total (s)	294.9778	89.17216	405.2791
Nº Iterações	32	31	43
Nº Samples	400	1	400
Nº de Vizinhos	15	15	15
Tamanho Grid (X*Y)	-	12*24	-
Tamanho Célula (m)	-	1	-
Nº Trajetos	1	1	2
Probabilidade Aceitar	-	-	25%
Distancia Percorrida (m)	47.62949	44.23538	40.90566

Como conclusões finais entre os métodos desenvolvidos, GPRM e PPRM, e o método em que estes foram baseados, PRM, pode-se verificar que o método PRM pode ser substituído pelos métodos desenvolvidos, conseguindo-se melhores resultados e performances que o tradicional método PRM. Pela análise dos resultados constata-se que o método que melhor otimiza o trajeto é o GPRM, sendo também o método que melhor se adequa aos cenários mais complicados, que é o caso do *Rescue*. Para cenários em que a sua complexidade seja menor ou que possua grandes espaços sem obstáculos o método com melhor performance é o PPRM. Também ficou claro que os métodos são sensíveis à sua parametrização o que deverá ser objeto de estudo no futuro, sendo que os parâmetros de cada método desenvolvido podem e devem ser ajustados a cada cenário permitindo obter os melhores resultados. Também se pode verificar que os métodos desenvolvidos são mais eficazes no cálculo da trajetória pois ambos conseguem ir de um ponto para outro percorrendo uma menor distância.

Esta página foi intencionalmente deixada em branco.

Capítulo 7

Conclusão e Trabalho Futuro

Esta dissertação abordou as áreas de perceção e navegação para veículos aéreos autónomos em cenários de busca e salvamento.

Inicialmente foi efetuado um estudo com objetivo de analisar os métodos e abordagens existentes para a perceção e navegação de veículos aéreos que pudesse ser aplicados em cenários de busca e salvamento. Esta análise permitiu conhecer a melhor abordagem para cada área e definir a arquitetura do projeto, conseguindo desta forma atingir o primeiro objetivo definido.

De forma a ser possível aplicar a arquitetura geral do sistema foi necessário desenvolver um método *Clustering* que permitisse obter os obstáculos que se encontram no cenário de aplicação. Este método tinha como requisito o facto de necessitar ser aplicado em tempo-real, pelo que o custo computacional deveria ser reduzido, agregando obstáculos em *clusters*.

Para que o veículo aéreo conseguisse navegar num cenário de busca e salvamento recorreu-se ao método PRM como base de estudo e a partir daí foram desenvolvidos dois novos métodos que permitiram obter melhores resultados que método PRM.

Desenvolvidos os métodos, foi necessário efetuar uma comparação e validação destes para diferentes cenários de simulação, tais como ambiente *Window*, onde o veículo necessita de passar pelo interior de uma janela, ambiente *Maze*, onde o veículo deve encontrar o trajeto até à saída num labirinto e por fim o ambiente *Res*cue, semelhante ao que se pode encontrar numa situação de busca e salvamento onde o veículo terá de navegar no interior de um edifício em ruínas.

O trabalho desenvolvido na dissertação resultou na publicação de um artigo cientifico "Three Dimensional Probabilistic Path Planning with Aerial Vehicles for Unstructured Environments"na conferência 47th International Symposium on Robotics (ISR 2016).

Encontra-se ainda submetido para aprovação o artigo "Grid-Based Three Dimensional Path Planning for Aerial Vehicles"na conferência IEEE Multi-Conference on Systems and Control 2016.

Como trabalho futuro pretende-se efetuar portabilidade do atual código, que se encontra desenvolvido em Matlab, para a *framework* ROS, assim como avaliar o desempenho do UAV Octos quando aplicado os dois métodos de planeamento de trajetória desenvolvidos para os cenários simulados.

Uma outra linha de trabalho consiste no desenvolvimento de um bloco dinâmico de redefinição dos parâmetros utilizados no GPRM e no PPRM no sentido de permitir uma maior adaptabilidade ao cenário de aplicação.

Bibliografia

[1] Nathan Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *J. Field Robot*, pages 832–841, 2012.

[2] Jose J. Acevedo, Begoña C. Arrue, Ivan Maza, and Anibal Ollero. A decentralized algorithm for area surveillance missions using a team of aerial robots with different sensing capabilities. *Robotics and Automation (ICRA), 2014 IEEE International Conference*, pages 4735–4740, 2014.

[3] A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart. Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics. *Robotics and Automation (ICRA), 2015 IEEE International Conference*, pages 6423–6430, 2015.

[4] ISEP/ INESC TEC Aerial Robotics Team. Dataset eurathlon. 2015.

[5] Jizhong Xiao Liang Yang, Junton Qi and Xia Yong. A literature review of uav 3d path planning. *11th World Congress on Intelligent Control and Automation*, 2014.

[6] Ludia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, 1996 IEEE Transactions*, 1996.

[7] Donguk Kim Youngsong Cho and Deok-Soo Kim. Topology representation for the voronoi diagram of 3d spheres. *International Journal of CAD/CAM*, 2009.

[8] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *arXiv:1005.0416*, 2010.

[9] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous multi-floor indoor navigation with a computationally constrained MAV. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 20–25, 2011.

[10] Matthias Nieuwenhuisen, David Droeschel, Johannes Schneider, Dirk Holz, Thomas Läbe, and Sven Behnke. Multimodal Obstacle Detection and Collision Avoidance for Micro Aerial Vehicles. *6th European Conference on Mobile Robots (ECMR)*, 2013.

[11] Alias Abdul Rahman Ivin Amri Musliman and Volker Coors. Implementing 3d network analysis in 3d-gis. *International archives of ISPRS*, 2008.

[12] Giorgio Guglieri Luca De Filippis and Fulvia Quagliotti. Path planning strategies for uavs in 3d environments. *Journal of Intelligent and Robotic Systems*, 2012.

[13] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. A fully autonomous indoor quadrotor. *IEEE Transactions on Robotics*, 28(1):90–100, 2012.

[14] Sungsik Huh, David Hyunchul Shim, and Jonghyuk Kim. Integrated navigation system using camera and gimbaled laser scanner for indoor and outdoor autonomous flight of UAVs. *IEEE International Conference on Intelligent Robots and Systems*, pages 3158–3163, 2013.

[15] Marziye Abdollahi Neda Shahidi, Hadi Esmaeilzadeh and Caro Lucas. Memetic algorithm based path planning for a mobile robot. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 2007.

[16] Antoine Beyeler, Jean-Christophe Zufferey, and Dario Floreano. Vision-based control of near-obstacle flight. *Autonomous Robots*, 2009.

[17] Jean-Christophe Zufferey, Antoine Beyeler, and Dario Floreano. Near-obstacle flight with small UAVs*. 2008.

[18] Antoine Beyeler, Jean-Christophe Zufferey, and Dario Floreano. optiPilot: control of take-off and landing using optic flow. *European Micro Air Vehicle Conference*, 2009.

[19] N. Andrew Browning, Stephen Grossberg, and Ennio Mingolla. Cortical dynamics of navigation and steering in natural scenes: Motion-based object segmentation, heading, and obstacle avoidance. 2008.

[20] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J. Andrew Bagnell, and Martial Hebert. Learning Monocular Reactive UAV Control in Cluttered Natural Environments. 2012.

[21] Inkyu Sa, Hu He, Van Huynh, and Peter Corke. Monocular Vision based Autonomous Navigation for a Cost-Effective MAV in GPS-denied Environments. *Advanced Intelligent Mechatronics (AIM)*, 2013.

[22] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. *International Symposium on Mixed and Augmented Reality (ISMAR'07)*, 2007.

[23] Markus Achtelik, Michael Achtelik, Stephan Weiss, and Roland Siegwart. Onboard IMU and Monocular Vision Based COntrol for MAVs in Unknown In- and Outdoor Environments. *Robotics and Automation (ICRA)*, 2011.

[24] Sammy Omari, Pascal Gohl, Michael Burri, and Roland Siegwart. Visual Industrial Inspection Using Aerial Robots. *International Conference on Applied Robotics for the Power Industry (CARPI)*, 2014.

[25] Jeffrey Byrne, Martin Cosgrove, and Raman Mehra. Stereo Based Obstacle Detection for an Unmanned Air Vehicle. *Robotics and Automation*, 2006.

[26] C. De Wagter, S. Tijmons, B.D.W. Remes, and G.C.H.E. Croon. Autonomous Flight of a 20-gram Flapping Wing MAV with a 4-gram Onboard Stereo Vision System. *Robotics and Automation (ICRA)*, 2014.

[27] David Droeschel, Jörg; Stückler, and Sven Behnke. Local Multi-Resolution Representation for 6D Motion Estimation and Mapping with a Continuously Rotating 3D Laser Scanner. *Robotics and Automation (ICRA), IEEE International Conference on*, (May), 2014.

[28] David Droeschel, Dirk Holz, and Sven Behnke. Omnidirectional Perception for Lightweight MAVs using a Continuously Rotating 3D Laser Scanner. *Photogrammetrie Fernerkundung Geoinformation*, XL(September):323–332, 2014.

[29] Fei Wang, Kangli Wang, Shupeng Lai, Swee King Phang, Ben M Chen, and Tong H Lee. An Efficient UAV Navigation Solution for Confined but Partially Known Indoor Environments. *Control and Automation (ICCA)*, pages 1351–1356, 2014.

[30] Matthias Nieuwenhuisen, David Droeschel, Marius Beul, and Sven Behnke. Obstacle Detection and Navigation Planning for Autonomous Micro Aerial Vehicles. *International Conference on Unmanned Aircraft Systems (ICUAS)*, (May), 2014.

[31] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[32] Allen Ferrick, Jesse Fish, Edward Venator, and Gregory S Lee. UAV obstacle avoidance using image processing techniques. *2012 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 73–78, 2012.

[33] Boris Jutzi, Martin Weinmann, and J Meidow. Improved Uav-Borne 3D Mapping By Fusing Optical and Laserscanner Data. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL(September):4–6, 2013.

[34] Hani Safadi. Local path planning using virtual potential field. 2007.

[35] DING Fu-guang, JIAO Peng, BIAN Xin-qian, and WANG Hong-jian. Auv local path planning based on virtual potential field. *International Conference on Mechatronics and Automation*, 2005.

[36] Thomas Hellström. Robot navigation with potential fields. *Autonomous Robots*, 2011.

[37] Howie Choset, Ji Yeong Lee, G.D. Hager, and Z. Dodds. Robotic motion planning: Potential functions.

[38] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, pages 269 – 271, 1959.

[39] Ludia E. Kavraki, Mihail N. Kolountzakis, and Jean-Claude Latombe. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, 1998 IEEE Transactions*, 1998.

[40] Václav Hlaváč. Motion planning methods. pages 22–24.

[41] Philip. An algorithm for finding a path for a point robot.

[42] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 1991.

[43] Karim Shaban and Felix Duvallet. Motion planning (wavefront algorithm). 2006.

[44] Howie Choset. Robotic motion planning: Potential functions. pages Robotics Institute 16–735.

[45] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. *ICRA Workshop on Open Source Software*, 2009.

[46] William Curran, Thomas Thornton, Benjamin Arvey, and William D. Smart. Evaluating impact in the ros ecosystem. *International Conference on Robotics and Automation (ICRA)*, 2015.

[47] Nate Koenig and Andrew Howard. Gazebo. 2014. Software available at `http://gazebosim.org/`.

[48] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. *International Conference on Intelligent Robots and Systems*, 2004.

[49] Daniel Cook, Andrew Vardy, and Ron Lewis. A survey of auv and robot simulators for multi-vehicle operations. 2014.

[50] Wei Qian, Zeyang Xia, Jing Xiong, Yangzhou Gan, Yangchao Guo, Shaokui Weng, Hao Deng, Ying Hu, and Jianwei Zhang. Manipulation task simulation using ros and gazebo. *International Conference on Robotics and Biomimetics*, 2014.

[51] Mengmi Zhang, Hailong Qin, Menglu Lan, Jiaxin Lin, Shuai Wang, Kaijun Liu, Feng Lin, and Ben M. Chen. A high fidelity simulator for a quadrotor uav using ros and gazebo. *IECON2015*, 2015.

[52] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar von Stryk. Comprehensive simulation of quadrotor uavs using ros and gazebo. *SIMPAR*, 2012.

[53] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. Usarsim: a robot simulator for research and education. *International Conference on Robotics and Automation*, 2007.

[54] Saeid Mokaram, Khairulmizam Samsudin, Abdul Rahman Ramli, and Hamideh Kerdegari. Mobile robots communication and control framework for usarsim. *International Conference on Intelligent and Advanced Systems*, 2012.

[55] Alima AlDahak AlShamsi, Mohamed AIMarzouqi, and Lakmal Seneviratne. Evaluating usarsim for use in fire search and rescue. *International Conference on Internet Technology and Secured Transactions*, 2011.

[56] Shogo Okamoto, Kensuke Kurose, Satoshi Saga, Kazunori Olino, and Satoshi Tadokoro. Validation of simulated robots with realistically modeled dimensions and mass in usarsim. *International Workshop on Safety, Security and Rescue Robotics*, 2008.

[57] Jijun Wang. Usarsim a game-based simulation of the nist reference arenas. Software available at `http://sourceforge.net/projects/usarsim/`.

[58] Olivier Michel. Cyberbotics ltd. webotstm: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 2004.

[59] Olivier Michel, Valérie Michel, Fabien Roher, Stefania Pedrazzi, and David Mansolino. Webots. 1998. Software available at `https://www.cyberbotics.com/`.

[60] Zhang Xing-long, Weng Xu-dong, and Yang Zhe. Mobile robot simulation based on webots. *International Conference on Natural Computation*, 2011.

[61] Songmin Jia, Liwen Gao, Jinhui Fan, Jun Yan, Xiuzhi Li, and Jinbo Sheng. Intelligent tennis wheelchair control method based on webots platform. *International Conference on Mechatronics and Automation*, 2011.

[62] J.M. Ibarra Zannatha, L.E. Figueroa Medina, R. Cisneros Limón, and P. Mejía Alvarez. Behavior control for a humanoid soccer player using webots. *International Conference on Electronics, Communications and Computers*, 2011.

[63] Coppelia Robotics. v-rep. Software available at `http://www.coppeliarobotics.com/`.

[64] LAAS-CNRS. Morse, the modular openrobots simulation engine. 2013. Software available at `https://www.openrobots.org/wiki/morse/`.

[65] A. Dias, J. Almeida, N. Dias, P. Lima, and E. Silva. Simulation environment for multi-robot cooperative 3d target perception. *International Conference, SIMPAR 2014*, 2014.

[66] Gilberto Echeverria, Séverin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich. Simulating complex robotic scenarios with morse. *International Conference, SIMPAR 2012*, 2012.

[67] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: Morse. *International Conference on Robotics and Automation*, 2011.

[68] Zhang Tianwu Zhao Chunhui, Wang Rongzhi and Pan Quan. Visual odometry and scene matching integrated navigation system in uav. *17th International Conference on Information Fusion (FUSION)*, 2014.

[69] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 2011.