DESENVOLVIMENTO DE MÓDULOS PARA PLANEAMENTO E CONTROLO DE EXECUÇÃO DE MISSÕES DE VEÍCULOS AÉREOS NÃO TRIPULADOS

Filipe Manuel Costa Ferreira

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Sistemas Autónomos
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto

2012

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Filipe Manuel Costa Ferreira, Nº 1040138, 1040138@isep.ipp.pt

Orientação científica: Jorge Estrela da Silva, jes@isep.ipp.pt

Co-Orientação científica: João Borges de Sousa, jtasso@fe.up.pt

Mestrado em Engenharia Electrotécnica e de Computadores

Área de Especialização de Sistemas Autónomos

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

31 de Outubro de 2012

“If you don't stand for something, you will fall for anything.”

Malcolm X

Agradecimentos

Quero agradecer à minha família e amigos, o apoio demonstrado ao longo deste percurso académico, especialmente durante o desenvolvimento desta tese.

Um grandioso obrigado à minha esposa Ana Teresa, que me deu sempre muita força e coragem nos momentos mais difíceis, e que sempre compreendeu a minha ausência ao longo destes meses.

Agradeço também a todos os colegas de trabalho do LSTS, especialmente ao Ricardo Bencatel, Sérgio Ferreira, Joel Cardoso, Joel Gomes, João Fortuna, José Pinto, Ricardo Martins e Paulo Dias por toda a ajuda disponibilizada ao longo desta tese, e pelo espírito de camaradagem existente.

Pretendo também agradecer a todos os meus colegas do LSA, especialmente ao Hugo Queirós, Maria Costa, Pedro Gonçalves, João Teixeira, Nuno Ribeiro, ao Paulo e ao Vando, por todos os momentos de trabalho e diversão que tivemos ao longo destes dois anos de mestrado, bem como pela companhia nas diretas e nos fins-de-semana passados no LSA a realizar trabalhos.

Não posso deixar de agradecer ao meu orientador, Prof. Jorge Estrela, por todo o apoio e disponibilidade demonstrados no desenvolvimento desta tese, assim como na definição dos requisitos que levaram à sua realização. Quero também agradecer ao meu co-orientador, Proj. João Sousa pela sua disponibilidade e por me ter permitido desenvolver a tese no meu local de trabalho, pondo à minha disposição todos os meios necessários para a realização da mesma.

Resumo

Com a evolução da tecnologia, os UAVs (*unmanned aerial vehicles*) são cada vez mais utilizados, não só em missões de risco para o ser Humano, mas também noutro tipo de missões, como é o caso de missões de inspeção, vigilância, busca e salvamento. Isto deve-se ao baixo custo das plataformas assim como à sua enorme fiabilidade e facilidade de operação.

Esta dissertação surge da necessidade de aumentar a autonomia dos UAVs do projeto PITVANT (Projeto de Investigação e Tecnologia em Veículos Aéreos Não Tripulados), projeto de investigação colaborativa entre a AFA (Academia da Força Aérea) e a FEUP (Faculdade de Engenharia da Universidade do Porto), relativamente ao planeamento de trajetórias entre dois pontos no espaço, evitando os obstáculos que intersetem o caminho.

Para executar o planeamento da trajetória mais curta entre dois pontos, foi implementado o algoritmo de pesquisa A*, por ser um algoritmo de pesquisa de soluções ótimas. A área de pesquisa é decomposta em células regulares e o centro das células são os nós de pesquisa do A*. O tamanho de cada célula é dependente da dinâmica de cada aeronave.

Para que as aeronaves não colidam com os obstáculos, foi desenvolvido um método numérico baseado em relações trigonométricas para criar uma margem de segurança em torno de cada obstáculo. Estas margens de segurança são configuráveis, sendo o seu valor por defeito igual ao raio mínimo de curvatura da aeronave à velocidade de cruzeiro.

De forma a avaliar a sua escalabilidade, o algoritmo foi avaliado com diferentes números de obstáculos. As métricas utilizadas para avaliação do algoritmo foram o tempo de computação do mesmo e o comprimento do trajeto obtido. Foi ainda comparado o desempenho do algoritmo desenvolvido com um algoritmo já implementado, do tipo *fast marching*.

Palavras-Chave

UAV, PITVANT, Planeamento, Trajetórias, A*

Abstract

With the evolution of technology, UAVs (unmanned aerial vehicles) are being used, not only on missions with risk to humans, but also on other types of missions, such as, inspection, surveillance, search and rescue. This is due to the low cost of the platforms, and their great ease of operation and reliability.

This project arises from the need to increase the autonomy of UAVs inside PITVANT project, regarding the planning of trajectories between two points in space, avoiding obstacles that intersect the way.

To plan the shortest path between two points, the A * search algorithm was implemented, because it gives an optimal path. The search area was broken down into regular cells and their centers are the nodes of the A * search algorithm. The size of each cell is dependent on the dynamics of each aircraft.

To avoiding aircraft from touching the obstacles, it was developed a numerical method based on trigonometric relationships, to create a safety margin around each obstacle. These safety margins are configurable, by default, these margins have the value of the minimum radius of curvature of the aircraft at cruising speed.

In order to evaluate its scalability, the algorithm was evaluated with different numbers of obstacles. The metric used to evaluate the algorithm were the computation time and the length of the path obtained.

The performance of the algorithm developed was also compared with an algorithm from fast marching family.

Keywords

UAV, PITVANT, Planning, Paths, A *

Résumé

Avec l'évolution de la technologie, des drones sont de plus en plus utilisés non seulement dans le risque pour la santé humaine des missions, mais aussi d'autres types de missions, telles que les missions, l'inspection, la surveillance, la recherche et de sauvetage . Cela est dû au faible coût de la plate-forme ainsi que leur fiabilité et leur grande facilité d'opération.

Ce document provient de la nécessité d'accroître l'autonomie des drones PITVANT projet (Projeto de Investigação e Tecnologia em Veículos Aéreos Não Tripulados), créée par l'AFA (Academia da Força Aérea) en partenariat avec le FEUP (Faculdade de Engenharia da Universidade do Porto) relative à la planification des chemins entre deux points dans l'espace, en évitant les obstacles qui croisent le chemin.

Pour exécuter la planification de le plus court chemin entre deux points, nous avons mis l'algorithme de recherche A *, comme un algorithme de recherche de solutions optimales. Le domaine de recherche est décomposé en cellules régulières et le centre des cellules sont les noeuds de la recherche A *. La taille de chaque cellule est dépendante de la dynamique de chaque aéronef.

Pour que les avions ne entrent pas encolisent pas en collision avec les obstacles, nous avons développé une méthode numérique basée sur les relations trigonométriques pour créer une marge de sécurité autour de chaque obstacle. Ces marges de sécurité sont paramétrables, et sa valeur est, par défaut, égale au rayon de courbure minimal de l'avion à la vitesse de croisière.

Afin d'évaluer son évolutivité, l'algorithme a été évalué avec des nombres différents d'obstacles. Les paramètres utilisés pour l'évaluation de l'algorithme ont été le temps de calcul et de la même longueur de la trajectoire obtenue. La performance de l'algorithme obtenu a été également comparée avec un algorithme dejà mis en oeuvre, comme le *fast marching*.

Mots-clés

UAV, PITVANT, planification, Trajectoires, A *

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

AFA - Academia da Força Aérea

AFCS - *Automatic Flight Control System*

ASV - *Autonomous Surface Vehicle*

AUV - *Autonomous Underwater Vehicle*

CCT - *Cloud Cap Technology*

COTS - *Commercial off-the-shelf*

CPU - *Central Processing Unit*

DRIL - Deteção, Reconhecimento, Identificação, Localização

DUNE - *Dune Uniform Navigational Environment*

EUA - Estados Unidos da América

FAP - Força Aérea Portuguesa

FEUP - Faculdade de Engenharia da Universidade do Porto

GCS - *Ground Control Station*

GPS - *Global Positioning System*

HRT - *Human Robot Team*

IMC - *Inter-Module Communication Protocol*

IMU - *Inertial Measurement Unit*

LSTS - Laboratório de Sistemas e Tecnologia Subaquática

PCC - *Piccolo Command Center*

PITVANT - Projeto de Investigação e Tecnologia em Veículos Aéreos Não Tripulados

PT - Planeamento de Trajetórias

RDT - *Rapidly Exploring Dense Tree*

RRT - *Rapidly Exploring Random Tree*

SPP - *Shortest Path Planner*

UAV - *Unmanned Aerial Vehicle*

UGV - *Unmanned Ground Vehicle*

USB - *Universal Serial Bus*

1. INTRODUÇÃO

Em Dezembro de 1903 *Orville Wright* fez o primeiro voo tripulado, controlado e com sustentação na História da aviação. Quinze anos mais tarde *Charles Kettering* desenvolveu o primeiro UAV (*Unmanned Aerial Vehicle*), que era um míssil de cruzeiro controlado por giroscópio, possuindo asas e motor. Este UAV foi denominado de *Kettering Bug* [1]. Na Figura 1 podem ser visualizadas imagens do primeiro voo de *Orville Wright*, assim como o *Kettering Bug.*

Na guerra do Vietname, os EUA (Estados Unidos da América) começaram a utilizar UAVs mais frequentemente para missões de reconhecimento e vigilância, no entanto os UAVs utilizados e a sua tecnologia era bastante rudimentar comparada com os padrões atuais.

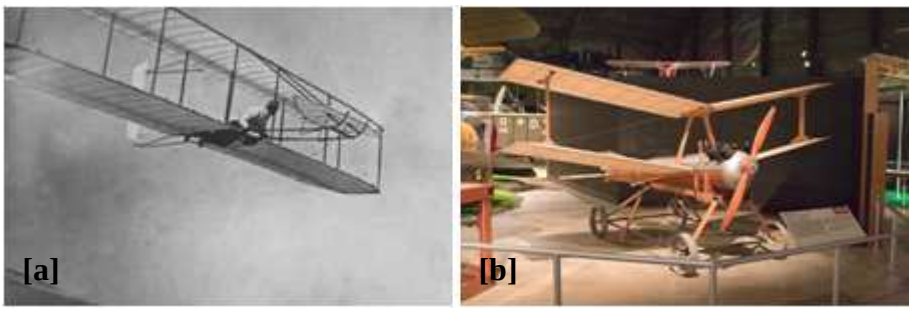

Figura 1 [a] Primeiro voo tripulado com sucesso, [b] Kettering Bug

Só em 1982 os UAVs da Força Aérea Israelita possuindo, tecnologia bem mais evoluída, conseguiram mostrar ao mundo a vantagem de serem utilizados UAVs em cenários de guerra, ao destruírem as defesas aéreas Sírias no Vale de Bekaa durante a operação "Paz para a Galileia" [17].

A utilização com êxito dos UAVs nessa operação fez com que muitos países, particularmente os EUA, reconhecessem a importância do uso de UAVs em operações militares, e foram consequentemente criadas políticas para o financiamento de programas de desenvolvimento desta tecnologia [17].

No seguimento dos programas criados pelos EUA para o desenvolvimento dos UAVs, foram desenvolvidas várias plataformas com diversas tecnologias para missões distintas, que foram posteriormente utilizadas com sucesso na operação "Tempestade no Deserto" entre 1991 e 1993, e também nas operações de segurança nos conflitos da Bósnia e do Kosovo [17].

Após o 11 de Setembro de 2001, a investigação e desenvolvimento na área dos UAVs e das suas tecnologias sofreu uma evolução esmagadora, com mais de vinte tipos diferentes de UAVs desenvolvidos, que vão desde UAVs com menos de 1kg de peso à descolagem e de algumas centenas de euros, até UAVs com mais de 10000kg de peso à descolagem que ultrapassam a dezena de milhão de euros, pertencentes às forças da coligação. Estes UAVs serviram as operações militares "*Enduring Freedom*" e "*Iraqi Freedom*" [17].

Apesar do enorme desenvolvimento dos UAVs nas últimas décadas, também foram sendo desenvolvidos outros veículos com tecnologias diferentes para ambientes distintos, entre os quais estão os AUVs (*Autonomous Underwater Vehicles*), os ASVs (*Autonomous Surface Vehicles*) e os UGVs (*Unmanned Ground Vehicles*). Com o desenvolvimento de todas estas tecnologias distintas, foi sendo necessário desenvolver conceitos de HRTs (*human robot teams*) [13] para aplicação em exercícios considerados de risco para o ser Humano. Estas HRTs podem ser homogéneas, onde os robots são do mesmo tipo, e.g. veículos aéreos, ou terrestres, ou aquáticos; e podem também ser heterogéneas, onde existem veículos de tipos diferentes, e.g. veículos aéreos e aquáticos.

Existem dois modos de operar os veículos em HRTs. No primeiro modo, o operador controla o robot à distância, usando uma comunicação *wired*, ou *wireless*. No segundo

modo, que é abordado nesta dissertação, o veículo é autónomo, i.e., o operador necessita apenas de planear, comandar e monitorizar a execução do trajeto definido.

Actualmente os UAVs são usados em inúmeras aplicações, quer de ordem civil ou militar, como é o caso da vigilância aérea e recolha de informações em ambientes hostis e de difícil acesso [2], evitando desta forma o uso de aeronaves tripuladas com custos mais dispendiosos, e por conseguinte eliminando o risco de prejuízos para a saúde da tripulação em caso de acidente, são também utilizados em cenários DRIL (Deteção, Reconhecimento, Identificação e Localização) de ameaças no solo [3], em vigilância de grande escala [4], deteção e combate a incêndios [5], mapeamento de território [6] [7], patrulha da costa e de fronteiras [8], monitorização de tráfego [9] [10], oleodutos e redes de alta tensão [11], busca e salvamento [12], monitorização ambiental, seguimento de estruturas ambientais, entre outras aplicações.

Com o desenvolvimento desta dissertação, pretende-se aliviar a carga de trabalho do piloto, retirando a tarefa de PT (Planeamento de Trajetórias) do *loop* de operações, passando a ser executada de forma automática. Para isso, será necessário desenvolver módulos de software para planeamento e execução de trajectórias para UAVs, em ambientes com obstáculos. No decorrer do trabalho serão abordados vários métodos existentes de planeamento de trajetórias, tanto em espaço discreto como contínuo.

1.1. ENQUADRAMENTO

Esta dissertação foi realizada na FEUP, mais propriamente no LSTS, e surgiu da necessidade existente de aumentar a autonomia das aeronaves do projeto PITVANT (Projeto de Investigação e Tecnologia em Veículos Aéreos Não Tripulados), em termos de planeamento e controlo de trajetórias.

O projeto PITVANT proposto pela AFA em parceria com a FEUP foi aprovado por Sua Excelência o Ministro da Defesa Nacional e tem a duração de sete anos, com término em Dezembro de 2015. Este projeto enquadra-se no domínio natural das atividades da Força Aérea, o domínio aeronáutico no sentido mais amplo do termo. É um projeto de grande dimensão, tendo como objetivo geral desenvolver competências em diversas tecnologias, doutrinas, formação e treino, inerentes à nova valência do poder aéreo do século XXI [17].

Os módulos (planeamento e execução de trajetórias) a serem desenvolvidos no decorrer desta dissertação serão implementados no software Neptus [14], e DUNE (Dune Uniform Navigational Environment) desenvolvidos no LSTS (Laboratório de Sistemas e Tecnologia Subaquática) na FEUP, sob a supervisão dos elementos responsáveis por ambos os sistemas.

De modo a ser perceptível a dimensão do projeto é feita uma breve apresentação do mesmo.

1.1.1. PROJETO PITVANT

O projeto PITVANT tem como missão construir e desenvolver UAVs, munindo desta forma a Força Aérea Portuguesa (FAP) com diversas capacidades operacionais para aplicações militares.

Os principais objetivos do projeto PITVANT são:

- Desenvolver tecnologias em diversas áreas para integração em UAVs, das quais se destacam:
    - Projeto, construção e teste de plataformas de pequena e média dimensão para serem utilizados em diversos tipos de missões, quer de ordem militar ou civil.
    - Controlo cooperativo de vários veículos com iniciativa mista
    - Interoperabilidade de sistemas
    - Sistemas de visão avançados
    - Fusão de dados
    - Sistemas de navegação
- Desenvolver novos conceitos de operação de UAVs, de pequena e média dimensão, a serem utilizados em missões militares.
- Testar a utilização dos sistemas e tecnologias desenvolvidas num largo espectro de missões, tanto de ordem militar como civil, das quais se destacam:

- Missões SAR (*Search and Rescue*)
- Missões de combate executadas por equipas cooperativas de UAVs, algumas delas com iniciativa mista.

- Formar pessoal com capacidade para definição de requisitos, operação e manutenção de UAVs

Na Figura 2 estão apresentados alguns dos UAVs desenvolvidos no projeto PITVANT, que fazem parte da frota existente no mesmo.

1.2. MOTIVAÇÃO E OBJECTIVOS

Neste momento, para ser possível operar um UAV em segurança, são necessários imensos meios humanos e logísticos. É de notar que em condições normais são necessários vários elementos da equipa de operação para executar todas as tarefas inerentes à execução de um voo autónomo [27].

Esta dissertação surge da necessidade existente de diminuir a carga de trabalho do piloto, dotando os UAVs do projeto PITVANT com capacidades de planeamento e execução de trajectórias, aumentando a autonomia destes em termos de capacidades de decisão e controlo, dotando-os também de ferramentas imprescindíveis para a sua deslocação em segurança, tanto em ambientes estáticos, como dinâmicos, reduzindo simultaneamente a carga de trabalho do piloto, limitando-o à tarefa de monitorização do estado do UAV no que diz respeito a planeamento e execução de trajetórias.

Figura 2 Apresentação das plataformas Antex X02 e Antex X03

Os principais objetivos da realização deste trabalho são:

- Desenvolvimento de módulos de software que permitam planear o trajeto para a deslocação de um UAV desde um ponto A até um ponto B em segurança, evitando obstáculos que intersetem o trajeto.
- Os módulos de software devem ser integrados nos sistemas desenvolvidos pelo LSTS, para poderem ser testados e validados nos veículos existentes, necessitam ainda de obedecer às regras de programação existentes.
- Diminuir a carga de trabalho do piloto referente ao planeamento de trajetórias.

1.3. ESTRUTURA DA DISSERTAÇÃO

Para além deste capítulo esta dissertação possui ainda mais seis capítulos.

- No capítulo 2 é feito um levantamento bibliográfico, onde são apresentados os métodos existentes para construção de trajetos, tanto em espaço discreto como em espaço contínuo. São também apresentados vários métodos de pesquisa do percurso mais curto entre dois pontos no espaço. São ainda feitas referências a trabalhos que implementam estes métodos, e também a trabalhos que realizam alterações a estes métodos para satisfazerem as suas necessidades.

- No capítulo 3 é apresentado o contexto onde esta tese se insere, aqui são apresentados todos os sistemas considerados relevantes para a execução deste trabalho, tanto em termos de *hardware* como de *software*. É apresentado o projeto PITVANT, as plataformas existentes no mesmo, os pilotos automáticos utilizados nas plataformas, os sistemas computacionais, e a *toolchain* de software existente no LSTS.

- No capítulo 4 é feita a análise da solução a implementar, é definido o problema que se pretende resolver, é apresentado o processo de engenharia de sistemas a utilizar durante as várias etapas do projeto, são definidos os requisitos para a resolução do problema com base numa *user story* bastante abrangente, é ainda apresentada a descrição da solução identificada, assim como as métricas para avaliação do algoritmo implementado.

- No capítulo 5 é apresentada a implementação do sistema, é descrito o desenvolvimento do *plugin* criado no Neptus, assim como todas as fases e técnicas utilizadas no desenvolvimento do algoritmo em MATLAB.

- No capítulo 6 são apresentados os resultados dos testes ao algoritmo, são apresentados testes em simulação em cenários com diferentes números de obstáculos, para diferentes plataformas. São também apresentados resultados de simulações que comparam o algoritmo desenvolvido, com um algoritmo do tipo *fast marching.* Neste capítulo é também apresentado o resultado da simulação do algoritmo desenvolvido com um AUV. Para finalizar são apresentados os resultados dos testes no terreno.

- No capítulo 7 são descritos todos os passos realizados para cumprir os objetivos propostos, e é verificado o cumprimento desses mesmos objetivos. São também apresentados vários desenvolvimentos necessários para trabalho futuro.

2. PLANEAMENTO DE TRAJETÓRIAS

Como foi referido no capítulo 1, um dos principais objetivos deste trabalho é reduzir a carga de trabalho do piloto em relação ao PT, e consequentemente aumentar a autonomia dos UAVs em relação à mesma matéria.

O termo trajetória consiste no seguimento de um caminho parametrizado em função do tempo. No contexto do documento, o termo trajetória refere-se ao seguimento de um caminho mantendo a velocidade num determinado valor constante.

O PT é um dos sistemas existentes em veículos autónomos. É o sistema responsável por gerar o trajeto para o movimento do veículo de uma determinada posição até outra posição no espaço, evitando obstáculos previamente conhecidos e também detetados no terreno.

Ao longo dos tempos foram sendo desenvolvidos inúmeros algoritmos e técnicas de PT para veículos terrestres em espaços fechados, assim como para sistemas de manipuladores.

Nas últimas décadas, com o enorme desenvolvimento dos UAVs, estes algoritmos foram sendo adaptados para ambientes exteriores e desconhecidos. Para se escolher o algoritmo ou técnica de PT mais adequada a cada situação é necessário ter em conta dois fatores:

primeiro, se o espaço para planeamento da trajetória é conhecido, desconhecido, ou parcialmente conhecido; segundo, que tipo de missão se irá realizar, e.g. busca e salvamento, vigilância, deteção e seguimento de alvos, monitorização ambiental, entre outras.

O problema típico da deslocação de um robot de um ponto A até um ponto B assenta em dois momentos distintos:

- No primeiro momento é utilizado um algoritmo para geração de caminhos que unam esses dois pontos, e.g. o método *road map*[29], o método de decomposição em células[33], o *potential field* [34], métodos probabilísticos [33], entre outros. Estes algoritmos assentam na definição específica do meio ambiente, que é definido por um mapa que contém obstáculos conhecidos e desconhecidos. A geração de caminhos possíveis entre os dois pontos pode ser obtida através da discretização do mapa em pequenas áreas ou células, ou então convertendo o mapa num campo contínuo. Consequentemente o planeamento de trajetórias pode ser classificado como discreto ou contínuo. Neste capítulo serão abordados alguns dos métodos existentes para geração de trajetos entre dois pontos. O método *road map* e de decomposição em células transforma o meio ambiente num mapa discreto, enquanto o método *potential field* transforma o mapa numa função contínua [29].

- No segundo momento é implementado um algoritmo de procura do caminho óptimo (depende das métricas de desempenho pretendidas) entre os dois pontos no mapa, e.g., l*evel set* e *fast marching methods* [57], A*, D*.

É comum alguns algoritmos de PT desenvolverem trajetos poligonais, sem terem em conta as restrições cinemáticas e a dinâmica do veículo, sendo *a posteriori* necessário aplicar alguns métodos numéricos ao trajeto planeado [30],[31], [32], para o suavizar e para que o UAV o consiga seguir. A Figura 3 apresenta as várias fases existentes na maioria dos planeadores de trajetórias.

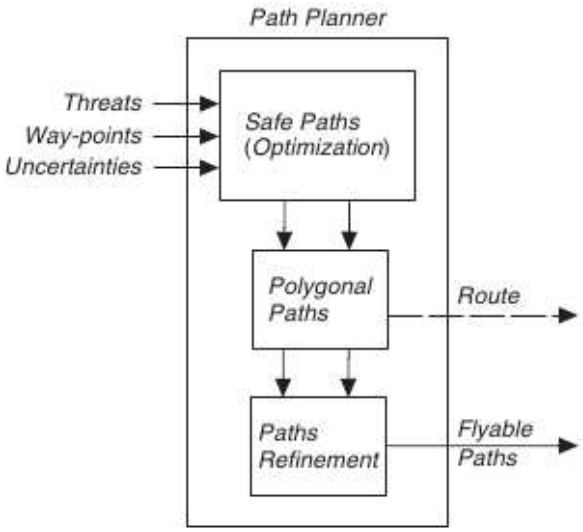

Figura 3 Estrutura típica dos planeadores de trajetórias [29]

2.1. ROAD MAP

O método *road map* (ver Figura 4) é uma rede de duas dimensões composta por linhas retas que ligam o ponto inicial (Ps - Figura 4) e final (Pf - Figura 4), sem intersetarem os obstáculos. Este algoritmo determina um conjunto de pontos que ao serem conectados por linhas retas produzem um conjunto de polígonos que envolvem cada obstáculo. Um algoritmo como o A* pode ser de seguida usado para gerar o trajeto mais curto entre os dois pontos. De seguida são apresentados dois métodos baseados em *road map*, estes métodos são os *visibility graphs*, e os diagramas de *Voronoi* [29].

2.1.1. VISIBILITY GRAPHS

Os *visibility graphs* (ver Figura 5) são uma rede de grafos que formam caminhos entre os vértices dos polígonos, foram um dos primeiros métodos de planeamento de trajetórias utilizado. No grafo **G = (V,E)**, os vértices **V** são os vértices dos obstáculos e **E** são as linhas que ligam cada vértice aos vértices visíveis. Este método só pode ser aplicado em ambientes com obstáculos poligonais [29]. O custo computacional deste método cresce à medida que o número de obstáculos vai aumentando, em [38] apenas os obstáculos que estão no trajeto entre o ponto inicial e final são considerados, tornando desta forma o algoritmo bastante mais rápido e capaz de correr a bordo do veículo para evitar obstáculos desconhecidos em tempo real.

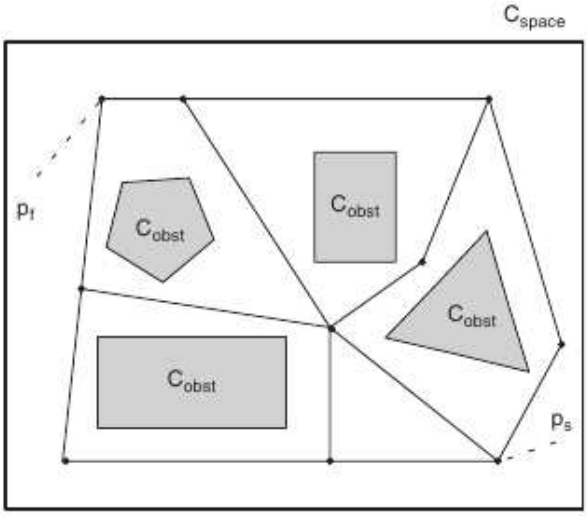

Figura 4 Método *Road Map* [29]

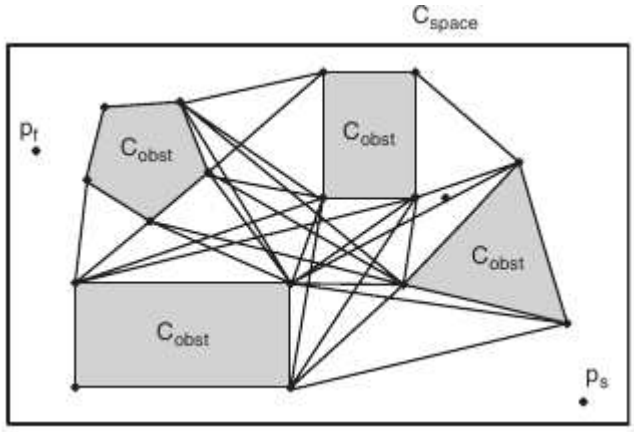

Figura 5 *Visibility Graph* [29]

2.1.2. VORONOI DIAGRAMS

Os diagramas de *Voronoi* são uma rede de grafos que formam polígonos em torno dos obstáculos. Para serem obtidos os polígonos dos diagramas de *Voronoi*, é necessário construir um conjunto de linhas que liguem os centros dos obstáculos, e posteriormente desenhar um conjunto de linhas perpendiculares às primeiras. As linhas perpendiculares são de seguida ajustadas para respeitar o número mínimo de vértices [29]. Em [35] são utilizados diagramas de *Voronoi* para criar trajetos seguros entre um conjunto de radares inimigos. Em [36] são utilizados diagramas de *Voronoi* para recuperar as comunicação com o veículo, separando a potência do sinal das comunicações por zonas. Em [37] são apresentados os diagramas *Zermelo-Voronoi,* que podem ser interpretados como diagramas de *Voronoi* dinâmicos. Na Figura 6 está apresentado um diagrama de *Voronoi*.

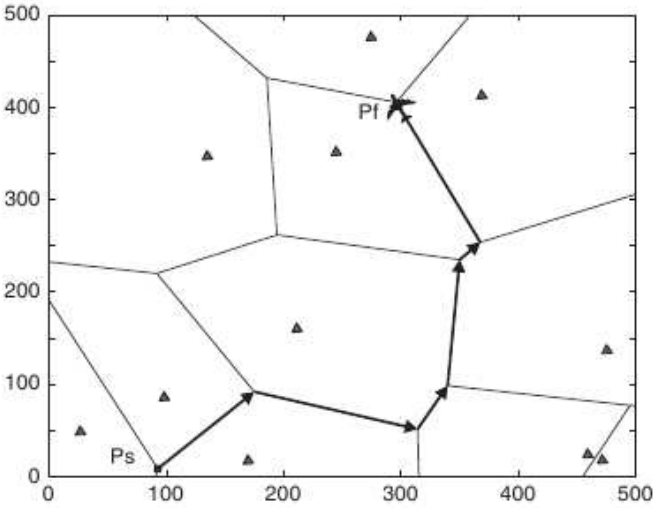

Figura 6 Diagrama *Voronoi* [29]

2.2. RAPIDLY EXPLORING DENSE TREES

Este método de PT baseia-se numa abordagem de amostragem e procura incrementais, que resulta num bom desempenho do algoritmo. A principal ideia deste método é incrementalmente construir uma árvore de procura, que vai aumentando a resolução. No limite a árvore cobre todo o espaço. Uma sequência densa de amostras é usada como guia na construção incremental da árvore. Se essa sequência for aleatória, a árvore é denominada RRT (*Rapidly Exploring Random Tree*).

Independentemente da sequência densa de amostras ser aleatória ou determinística, estas árvores de pesquisa pertencem às RDT (*Rapidly Exploring Dense Trees*).

Na Figura 7 é possível verificar que a RRT nas primeiras iterações alcança rapidamente zonas não exploradas, no entanto a RRT é densa no limite, com probabilidade um, o que significa que chega arbitrariamente a qualquer ponto do espaço [33].

Em [39] é apresentado o método *Transition-based* RRT que junta a força de exploração do algoritmo RRT que faz crescer árvores rapidamente em regiões do espaço não exploradas, com a eficiência dos métodos de otimização estocásticos, que usam testes de transição para aceitar ou rejeitar novos potenciais estados.

Figura 7 *Rapidly Exploring Random Tree*

2.3. POTENTIAL FIELD

A abordagem do *potential field*, apresentada por *Khatib* em 1985 [34], associa um campo potencial artificial ao espaço de estados do sistema. O ponto de destino é envolvido por uma campo potencial atrativo, enquanto o ponto inicial é envolvido por um campo potencial repulsivo. A ideia principal deste método prende-se com a movimentação do robot através do campo potencial, que será atraído pelo ponto de destino, enquanto vai sendo repelido pelos obstáculos [29], (ver Figura 8). Ao contrário do método *road map*, este método funciona em espaço contínuo, sendo a trajetória resultante traduzida por uma função contínua. Em [40] é utilizado o método *potential field* (ver Figura 9) em conjunto com o método de pesquisa A* para desenvolver um algoritmo de PT que permita mover um UAV autonomamente.

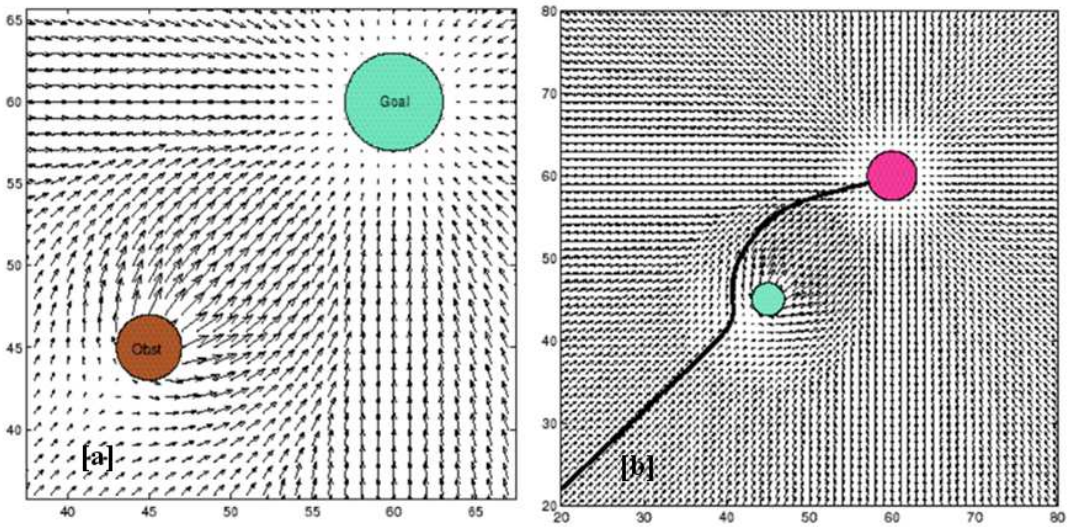

Figura 8 a) Vetor soma dos dois campos potenciais, b) Trajetória resultante do veículo [41]

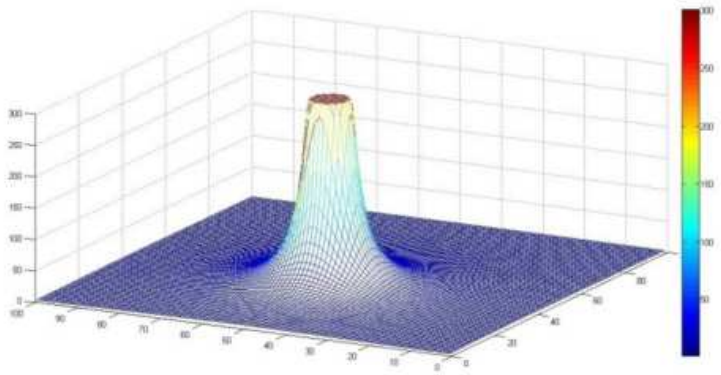

Figura 9 Modulação de um obstáculo com o método *potential field* [40]

2.4. DECOMPOSIÇÃO EM CÉLULAS

No método da decomposição em células, o espaço é dividido em células que não se sobrepõem. Podem existir células livres ou células ocupadas por obstáculos, o veículo só se pode movimentar entre cada célula adjacente, e que não contenha obstáculos. Existem inúmeras formas de dividir uma área que contém obstáculos em células, e.g. células exatas, retangulares, verticais, regulares, *quadtrees*.

2.4.1. CÉLULAS EXATAS

A decomposição em células exatas é um método que une os vértices dos obstáculos e os vértices do espaço, de forma a obter polígonos (ver Figura 10). Em [43] é usado o método de decomposição exata em células para planear o varrimento de uma determinada área por UAVs. Na Figura 11 estão apresentados dois métodos distintos de PT, com base em decomposição de células exatas.

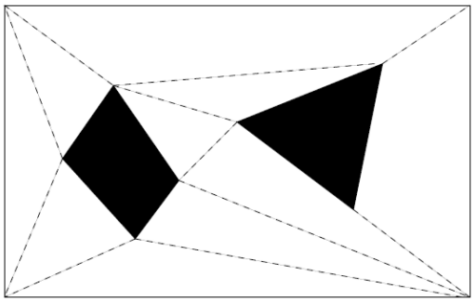

Figura 10 Decomposição em células exatas [42]

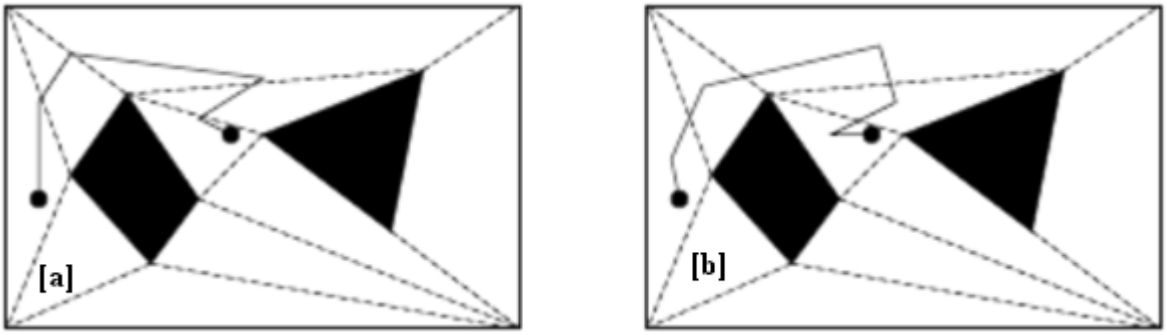

Figura 11 a) Planeamento da trajetória usando o centro das linhas das células, b) Planeamento da trajetória usando o centroide das células [42]

2.4.2. CÉLULAS RETANGULARES

A decomposição em células retangulares baseia-se em traçar linhas horizontais e verticais tangentes aos vértices mais extremos dos obstáculos. Este método não é muito utilizado devido à perda de resolução associada. Na Figura 12 está apresentado este método.

2.4.3. CÉLULAS VERTICAIS

No método da decomposição em células verticais são traçadas linhas verticais nos vértices dos obstáculos, assim como nos vértices dos limites do espaço, cujos ângulos internos sejam superiores a 90º. Na Figura 13 é possível identificar um espaço com obstáculos decomposto em células verticais, assim como os trajetos possíveis, obtidos através da união dos centroides de cada célula e os centros das linhas das mesmas.

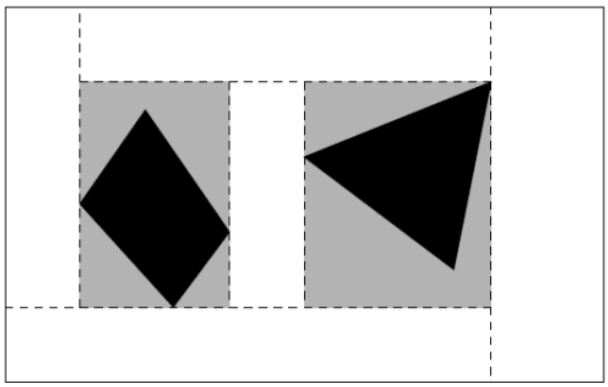

Figura 12 Decomposição em células retangulares [42]

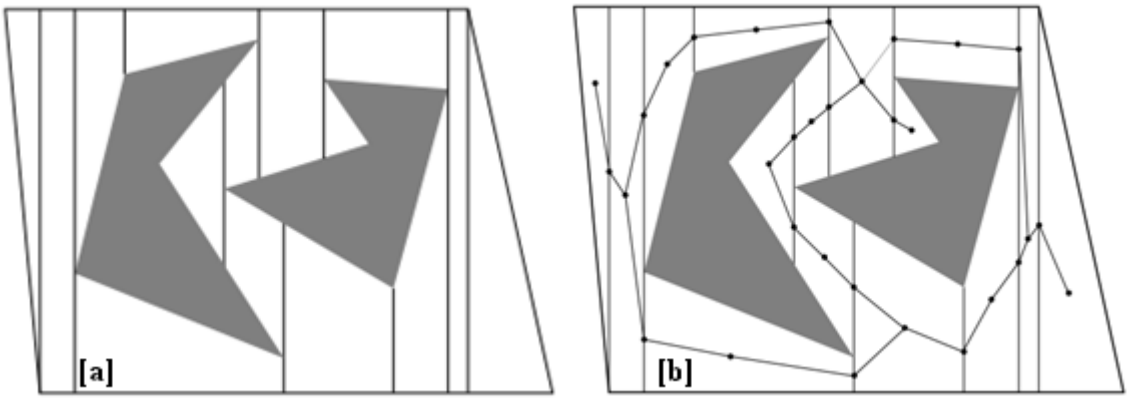

Figura 13 a) Decomposição em células verticais, b) *Roadmap* da decomposição em células verticais [33].

2.4.4. CÉLULAS REGULARES

Neste método o espaço é dividido em células com dimensão idêntica. As células ocupadas pelos obstáculos são consideradas células com um custo infinito, logo o veículo só pode deslocar-se nas células livres. O custo computacional aumenta à medida que diminui o tamanho de cada célula (ver Figura 14).

2.4.5. QUADTREES

As quadtrees são uma evolução às células regulares, no sentido de que todas as células não necessitam de ter a mesma dimensão, as células mais afastadas dos obstáculos podem ter dimensões maiores, e à medida que se chega perto dos obstáculos, as células vão-se dividindo em células cada vez mais pequenas, ver Figura 15. As *quadtrees* possuem um custo computacional de acesso às células e uma complexidade de implementação superiores ao das células regulares. No entanto, permitem uma maior resolução junto dos obstáculos mantendo a resolução desejada para o restante espaço computacional. Em [44] é apresentado um novo método de decomposição *quadtree*, designado *framed quadtree* (ver Figura 15). Este método diferencia-se das *quadtrees* comuns, por criar um conjunto de células pequenas nas bordas das restantes células. Este método tem um custo computacional superior ao método da decomposição em *quadtrees*, mas possui a vantagem de diminuir o comprimento do trajeto pretendido. Neste método, o veículo não necessita de deslocar-se até ao centro das células, para mover-se para as células vizinhas, o que em muitos casos é bastante benéfico para a eficiência do algoritmo de PT.

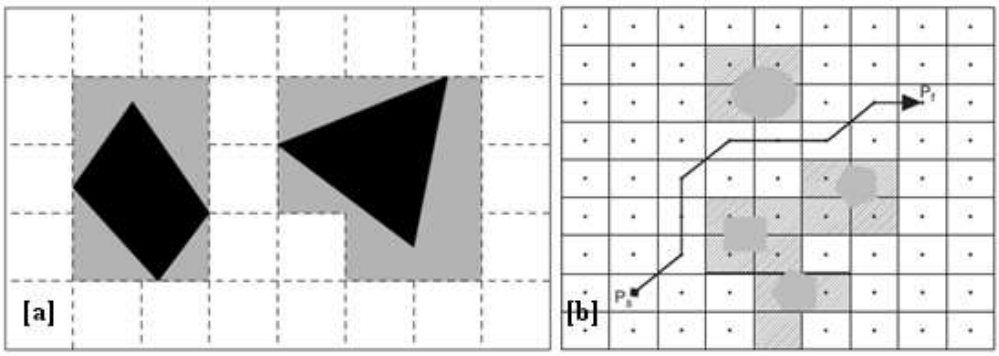

Figura 14 a) Decomposição em células regulares [42], b) Trajetória obtida em espaço decomposto em células regulares [29].

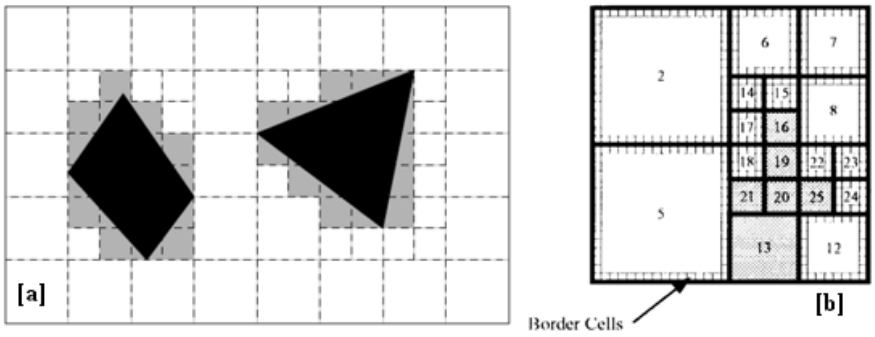

Figura 15 a) Decomposição *Quadtree* [29], b) *Framed Quadtree* [44]

2.5. CONTROLO ÓPTIMO

Existem diversas técnicas para planeamento de trajetórias baseadas em controlo ótimo. Entre eles encontram-se os *level set* e *fast marching methods*, que são duas técnicas desenvolvidas para seguimento da evolução de interfaces. Estas técnicas interligam a evolução duma interface (e.g., uma onda sísmica) com as equações que descrevem a dinâmica do sistema [45].

Na Figura 16 está apresentado um exemplo de um problema que foi resolvido recorrendo ao método *fast marching*. Supondo que uma pessoa está num parque de estacionamento no ponto A, e que o seu veículo está no ponto B, que os blocos pretos representam outros carros estacionadas, que a cor azul significa que o solo contém gelo, e que a cor castanha significa que o solo está seco (a). Para resolver este problema foi aplicada a técnica da propagação da onda a partir do ponto A, pode-se constatar que a onda move-se com maior velocidade no solo seco (b). A equação que descreve o tempo de chegada da frente de onda, que depende da velocidade e da direção é a equação Eikonal. O método de *fast*

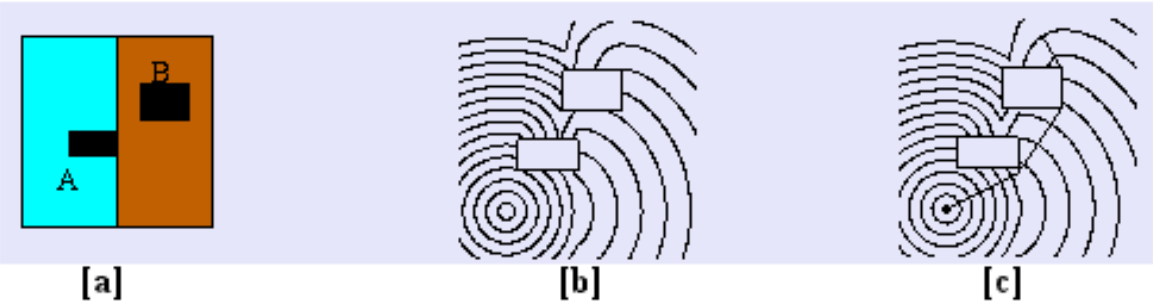

Figura 16 a) Cenário do problema, b) Propagação da onda a partir do ponto A, c) Trajetória obtida

marching é usado para resolver esta equação. Assim que a frente de onda colide com o ponto B, é possível descobrir o caminho mais curto entre os dois pontos; para isso é necessário construir um trajeto desde o ponto B até ao ponto A que seja sempre perpendicular às frentes de onda, como está apresentado em (c) [45].

2.5.1. FAST MARCHING

Este método é utilizado em problemas onde a função velocidade nunca muda de sinal, fazendo com que a frente de onda propague-se apenas numa direção, ou para a frente, ou para trás. Em [46] é utilizado o método *fast marching* para resolver um problema de computação de um diagrama de *Voronoi* definido através de um novo conceito designado por distância *boat*-sa*il*.

2.5.2. LEVEL SET METHODS

Este método por sua vez é utilizado em problemas onde a função velocidade pode ser positiva em determinadas posições, e negativa em outras posições, fazendo com a frente de onda propague-se para a frente em determinadas posições, e para trás noutras. Este método é bastante mais lento que o *fast marching.*

2.6. ESTRATÉGIAS DE PESQUISA SEM INFORMAÇÃO

Existem diversos métodos de pesquisa sem informação, e.g., Dijkstra, *breadth-first, uniform-cost, depth-first, depth-limited, Iterative Deepening Depth-first, bidirectional,* entre outros. Este termo significa que estas estratégias não possuem informação adicional sobre os estados, para além da disponibilizada na definição do problema. Estas estratégias apenas conseguem gerar sucessores e identificar se um nó é o de destino ou não. Todos estes métodos de pesquisa são diferenciados pela ordem com que expandem os nós [49].

2.6.1. DIJKSTRA

O algoritmo de pesquisa Dijkstra foi pela primeira vez apresentado por Edsger Dijkstra em 1959 [47]. É um método bastante conhecido para a determinação do caminho mais curto [58]. A procura é feita num grafo onde cada nó representa uma localização possível, e cada aresta representa um possível caminho entre os nós aos seus extremos, sendo associado um custo de deslocação não negativo a cada caminho.

Na Figura 17 está apresentada uma rede de grafos onde foi utilizado o método Dijkstra para encontrar o caminho mais curto entre o ponto O e o ponto T.

O algoritmo de pesquisa Dijkstra possui diversas etapas até conseguir encontrar o caminho mais curto entre dois pontos, nomeadamente:

1. Atribuir custos às linhas que unem os nós (estes custos podem traduzir-se na distância existente entre os nós), o nó inicial ou atual deve possuir um custo igual a zero e todos os outros devem possuir custos infinitos.
2. Colocar o nó inicial numa lista fechada, enquanto os restantes são colocados numa lista aberta.
3. Para o nó atual é necessário determinar o custo de deslocação desde este nó até todos os vizinhos que ainda se encontram na lista aberta, por exemplo, se o nó A for o nó atual com um custo atribuído de 2 e a linha que o liga ao nó B possui um custo de 2, então o custo da deslocação do nó inicial até B (através do nó A) é 4. Se este custo for inferior ao custo previamente obtido para o nó B, então o novo custo é atribuído ao nó B (este caso sucedeu no exemplo da Figura 17, inicialmente o custo da deslocação até B era 5 e passou para 4).

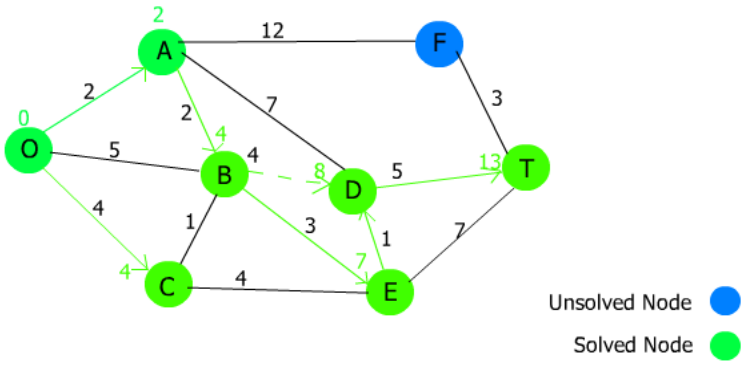

Figura 17 Rede de grafos [48]

4. Quando todos os nós vizinhos do nó atual foram avaliados, este sai da lista aberta e passa para a fechada, não será avaliado novamente, e fica com o menor custo identificado.
5. Se o nó de destino for atribuído à lista fechada então o caminho mais curto foi encontrado e o algoritmo termina.
6. O novo nó atual passa a ser o nó com menor custo presente na lista aberta e o algoritmo continua a partir do ponto 3

2.7. ESTRATÉGIAS DE PESQUISA COM INFORMAÇÃO (HEURÍSTICA)

Este termo significa que as estratégias de pesquisa possuem conhecimentos específicos do problema para além dos conhecidos transmitidos na definição do mesmo, e deste modo possuem soluções mais eficientes do que as estratégias de pesquisa sem informação prévia.

2.7.1. GREEDY BEST-FIRST

O método de pesquisa *greedy best-first* tenta sempre expandir o nó que está mais próximo do nó de destino. Este método utiliza apenas uma heurística para determinar o custo da deslocação de um determinado nó até ao nó de destino, como está apresentado na equação 1 (f(*n*) = função custo; h(*n*) = heurística utilizada). Neste caso a heurística utilizada é a distância euclidiana entre o nó atual e o de destino [49].

$$f(n) = h(n) \tag{1}$$

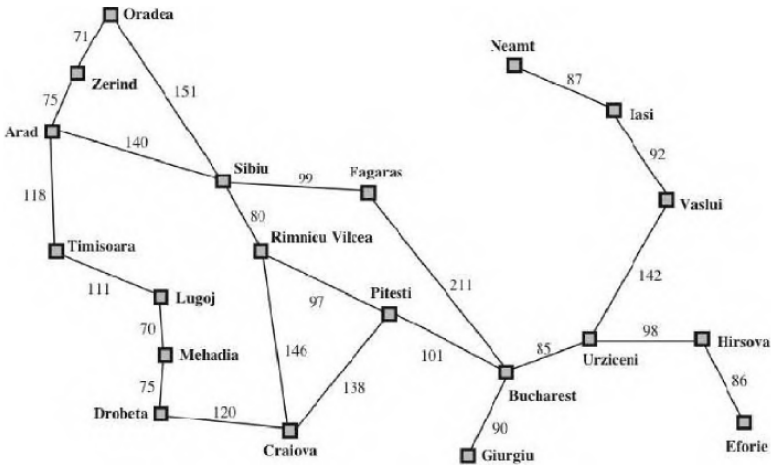

Figura 18 Mapa de estradas de uma parte da Romenia [49]

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	**Urziceni**	80
Iasi	226	Vaslui	199
Lngoj	244	Zerind	374

Figura 19 Distâncias euclidianas entre as cidades da Figura 18 e Bucharest [49]

Se for utilizado o método *greedy best-first* para determinar o trajeto entre a cidade de Arad e Bucharest, este funcionará da seguinte forma:

1. O primeiro nó a ser expandido será Sibiu, pois a heurística é menor que a heurística das cidades de Zerind e Tamiosoara.
2. O próximo nó a ser expandido será Fagaras pois possui a menor heurística, como Fagaras está conectado a Bucharest, não é necessário expandir mais nenhum nó.

É de notar que este tipo de procura não garante a otimalidade: o trajeto obtido através de Sibiu e Fagaras é 32 quilómetros mais longo do que Sibiu, Rimnicu Vilcea e Pitesti. Isto mostra a razão deste método ser ganancioso (*greedy*): em cada passo ele tenta chegar o mais próximo possível do nó de destino [49].

2.7.2. A *

O método de pesquisa A*, apresentado por *Hart, Nilsson* e *Raphael* em 1968 [50], é dos métodos mais utilizados para resolver problemas de PT. É uma extensão ao método Dijkstra, mas como usa heurísticas consegue alcançar melhores desempenhos. Este método avalia o nó que deve ser expandido, combinando o custo de chegar desde o nó inicial até ao nó n (**g(*n*)**), e o custo da deslocação desse nó até ao nó de destino (**h(*n*)**).

$$f(n) = g(n) + h(n) \quad (2)$$

Como **g(*n*)** é o custo da deslocação desde o nó inicial até ao nó **n**, e **h(*n*)** é o custo estimado desde o nó **n** até ao nó de destino, é possível afirmar que **f(*n*)** é o custo estimado da melhor solução para a deslocação até ao nó de destino, passando pelo nó **n**.

O A* é um método de pesquisa ótimo, se duas condições forem satisfeitas:

1. **Admissibilidade -** Isto significa que **h(*n*)** tem que ser uma heurística admissível. Uma heurística admissível é aquela que nunca estima um valor superior ao valor real. Se a heurística utilizada for a distância euclidiana, esta condição está satisfeita porque a menor distância entre dois pontos é uma linha reta.

2. **Consistência –** A consistência é uma condição necessária para que o método A* seja ótimo, mas apenas se este método for utilizado em pesquisas de grafos. A heurística **h(*n*)** é consistente se para todos os nós **n** e os seus sucessores **n'** gerados por uma ação **a,** o custo estimado da deslocação desde o nó **n** até ao nó final não for superior ao custo da deslocação desde o nó **n** até ao nó **n'** mais o custo estimado da deslocação desde o nó **n'** até ao nó de destino, como pode ser consultado na equação 3.

$$h(n) \leq c(n,a,n') + h(n') \quad (3)$$

Em [51] é utilizado o método A* para PT de UAVs. Em [40] é utilizado o método A* modificado no PT para UAVs, usando campos potenciais. Em [32] e [52] são utilizados métodos A* modificados para planear a trajetória de UAVs.

2.7.3. D*

O método de pesquisa D* apresentado por *Stentz* em [53] possui este nome por se assemelhar ao método A*, exceto no sentido de que os custos podem variar ao longo do tempo, tornando este método dinâmico. O método D* também garante que as trajetórias geradas são ótimas, ao contrário de outros métodos de PT dinâmicos.

3. PLATAFORMA DE OPERAÇÃO

3.1. PLATAFORMAS

Ao longo dos vários anos do projeto foram sendo projetadas, construídas e testadas diversas plataformas que são classificadas por grupos, de acordo com a Tabela 1, disponibilizada pelo departamento de defesa dos EUA [18].

3.1.1. ASA VOADORA

A Asa Voadora é um UAV relativamente pequeno, com pouca necessidade de manutenção; é uma plataforma que necessita ser lançada à mão ou por catapulta, e que permite testar diversos sensores, incluindo diversos tipos de câmaras. Na Figura 20 é possível visualizar o aspeto de uma Asa Voadora e na Tabela 2 é possível consultar as suas características operacionais.

Tabela 1 Nomenclatura e classificação dos vários sistemas

DoD Unmanned Aircraft Systems (As of 1 JULY 2011)

General Groupings	Depiction	Name	(Vehicles/GCS)	Capability/Mission	Command Level
Group 5 • > 1320 lbs • > FL180		•USAF/USN RQ-4A **Global Hawk/BAMS-D** Block 10 •USAF RQ-4B **Global Hawk** Block 20/30 •USAF RQ-4B **Global Hawk** Block 40	•9/3 •20/6 •5/2	•ISR/MDA (USN) •ISR •ISR/BMC	•JFACC/AOC-Theater •JFACC/AOC-Theater •JFACC/AOC-Theater
		•USAF MQ-9 **Reaper**	•73/85* *MQ-1/MQ-9 same GCS	•ISR/RSTA/EW/ STRIKE/FP	•JFACC/AOC- Support Corps, Div, Brig, SOF
Group 4 • > 1320 lbs • < FL180		•USAF MQ-1B **Predator**	•165/85*	•ISR/RSTA/STRIKE/FP	•JFACC/AOC-Support Corps, Div, Brig
		•USA MQ-1 **Warrior**/MQ-1C **Gray Eagle**	•31/11	•(MQ-1C Only-C3/LG)	•NA
		•USN **UCAS**- CVN Demo	•2/0	•Demonstration Only	•NA
		•USN MQ-8B **Fire Scout** VTUAV	•14/8	•ISR/RSTA/ASW/ ASUW/MIW/OMCM/ EOD/FP	•Fleet/Ship
		•SOCOM/DARPA/USA/USMC A160T **Hummingbird**	•8/3	•Demonstration Only	•NA
Group 3 • < 1320 lbs • < FL180 • < 250 knots		•USA MQ-5 **Hunter**	•45/21	•ISR/RSTA/BDA	•Corps, Div, Brig
		•USA/USMC/SOCOM RQ-7 **Shadow**	•368/265	•ISR/RSTA/BDA	•Brigade Combat Team
		•USN/USMC **STUAS**	•0/0	•Demonstration	•Small Unit
Group 2 • 21-55 lbs • < 3500 AGL • < 250 knots		•USN/SOCOM/USMC RQ-21A **ScanEagle**	•122/13	•ISR/RSTA/FORCE PROT	•Small Unit/Ship
Group 1 • 0-20 lbs • < 1200 AGL • < 100 knots		•USA / USN / USMC / SOCOM RQ-11 **Raven**	•5628/3752	•ISR/RSTA	•Small Unit
		•USMC/ SOCOM **Wasp**	•540/270	•ISR/RSTA	•Small Unit
		•SOCOM SUAS AECV **Puma**	•372/124	•ISR/RSTA	•Small Unit
		•USA **gMAV** / USN **T-Hawk**	•270/135	•ISR/RSTA/EOD	•Small Unit

Figura 20 UAV Asa-Voadora

Tabela 2 Características Operacionais da Asa-Voadora

Peso máximo à descolagem	3,5 kg
Envergadura	2 m
Velocidade máxima	90 km/h
Carga útil máxima	1 kg
Autonomia máxima	1h00
Altitude máxima	500 m
Motor eléctrico	
Lançado à mão ou por catapulta, com voo e aterragem autónomos	

3.1.2. ANTEX-X02 (ALFA)

A família de plataformas Antex-X02, ou alfas como são informalmente designados, são as plataformas com mais horas de voo acumuladas no projeto. São plataformas projetadas e construídas na AFA. Como são plataformas relativamente pequenas, o seu manuseio é bastante facilitado e em pouco tempo é possível ter esta plataforma pronta a voar. Na Figura 21 está apresentado o alfa-05 e na Tabela 3 podem ser consultadas as suas características operacionais.

3.1.3. ANTEX-X03

As plataformas Antex-X03 foram também projetadas e construídas na AFA. São as plataformas com maior dimensão no projeto PITVANT e consequentemente são as que possuem maior capacidade de carga, permitindo albergar componentes com dimensões superiores, assim como testar sensores que não são possíveis testar nas outras plataformas. Na Figura 22 está representado um antex-X03 do projeto e na Tabela 4 as suas características operacionais.

3.1.4. EXTENDED

A plataforma Extended é a mais recente plataforma projetada e construída pela AFA. Esta plataforma veio colmatar algumas desvantagens existentes nas plataformas anteriores nomeadamente, o Antex-X02 é um UAV que apesar de ser bastante completo, possui pouco espaço interno para payload. O Antex-X03, apesar de possuir imenso espaço para payload, é extremamente grande e requer bastante tempo para todos os procedimentos logísticos e de preparação para voo, tornando morosos simples testes de sensores. Na Figura 23 está apresentado o Extended-00 e na Tabela 5 as suas características operacionais.

3.1.5. PILATUS

O Pilatus é uma plataforma COTS (Commercial off-the-shelf), bastante mais económica que as plataformas desenvolvidas pela AFA, que serve sobretudo para testar e validar sensores e algoritmos, que só depois serão integrados nas plataformas desenvolvidas no projeto. Na Figura 24 está ilustrado um Pilatus, e na Tabela 6 as suas Características Operacionais.

3.1.6. CULARIS

O Cularis é também uma plataforma COTS. Ainda mais económica que o pilatus, de dimensão bastante mais reduzida, de transporte bastante fácil e de montagem extremamente rápida. É uma plataforma que apesar de não possuir muito espaço interno para *payload*, é muito útil para testar algoritmos, nomeadamente o algoritmo de voo em formação, iniciativa mista, planeamento de trajetória, entre outros. Na Figura 25 está apresentado um Cularis, e na Tabela 7 as suas Características Operacionais.

Figura 21 UAV ANTEX-X02

Tabela 3 Características Operacionais do Antex-X02

Peso máximo à descolagem	10 kg
Envergadura	2,4 m
Velocidade máxima	150 km/h
Carga útil máxima	4 kg
Autonomia máxima	5h00
Altitude máxima	2000 m
Motor a combustão ou eléctrico	
Descolagem, Voo e aterragem Autónomos	

Figura 22 UAV ANTEX-X03

Tabela 4 Características Operacionais do Antex-X03

Peso máximo à descolagem	150 kg
Envergadura	7 m
Velocidade máxima	130 km/h
Carga útil máxima	30 kg
Autonomia máxima	15h00
Altitude máxima	4500 m
Motor a combustão	
Descolagem, Voo e Aterragem Autónomos	

Tabela 5 Características Operacionais do Extended

Figura 23 Extend

Peso máximo à descolagem	20 kg
Envergadura	3,5 m
Velocidade máxima	120 km/h
Carga útil máxima	10 kg
Autonomia máxima	3h00
Altitude máxima	3000 m
Motor a combustão ou eléctrico	
Descolagem, Voo e Aterragem Autónomos	

Tabela 6Características Operacionais do Pilatus

Figura 24 UAV Pilatus

Peso máximo à descolagem	14 kg
Envergadura	2,82 m
Velocidade máxima	90 km/h
Carga útil máxima	5 kg
Autonomia máxima	1h20
Altitude máxima	2000 m
Motor a combustão	
Descolagem, Voo e aterragem Autónomos	

Tabela 7 Características Operacionais do Cularis

Figura 25 UAV Cularis

Peso máximo à descolagem	2.1 kg
Envergadura	2,64 m
Velocidade máxima	90 km/h
Carga útil máxima	0.5 kg
Autonomia máxima	0h40
Altitude máxima	2000 m
Motor elétrico	
Lançado à mão ou por catapulta, Voo e Aterragem Autónomos	

3.2. PILOTO AUTOMÁTICO

O piloto automático ou auto-piloto é um dispositivo electrónico utilizado para controlar veículos espaciais, aéreos, aquáticos, terrestres, e até mísseis, sem intervenção Humana constante. O piloto automático é muitas vezes diretamente associado aos veículos aéreos, se bem que nos aviões o piloto automático é habitualmente designado por AFCS (*Automatic Flight Control System*). [21]

O princípio de funcionamento do controlo de um voo autónomo pode ser consultado na Figura 26. O piloto automático recebe informações sobre a posição, a atitude, velocidade e altitude do avião através de diversos sensores, nomeadamente, acelerómetros, giroscópios, tubos de *pitot*, GPS, entre outros. De acordo com os valores obtidos, o piloto automático calcula as referências para a velocidade e posição pretendidas, que são traduzidas em comandos para os servos das respetivas superfícies de controlo (ver Figura 27), e volta novamente a receber informações sobre o estado do avião através dos diversos sensores, e o ciclo repete-se pois este controlador é de malha fechada.

No PITVANT e no LSTS são utilizados dois tipos distintos de pilotos automáticos, o Piccolo e o Ardupilot Mega2.

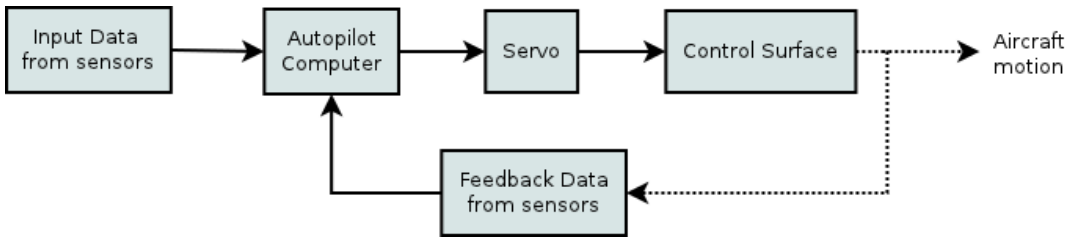

Figura 26 Controlador em malha fechada de um voo autónomo [21]

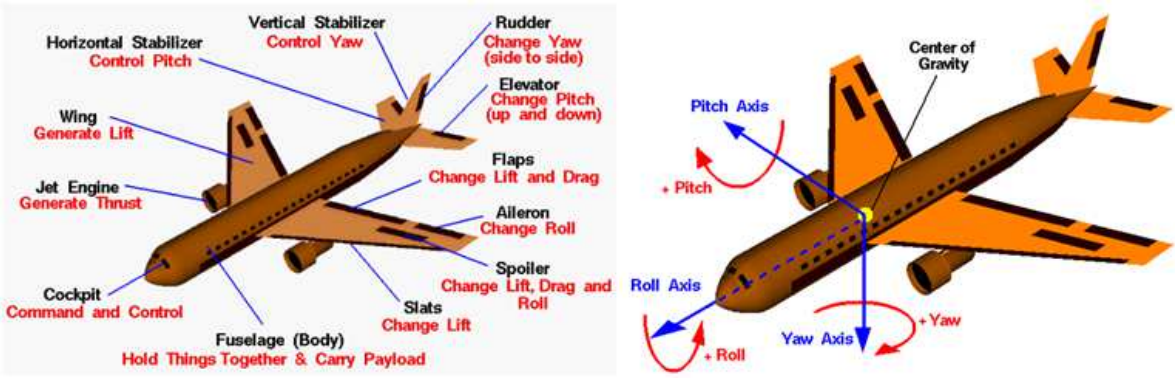

Figura 27 Superfícies de controlo e respetivos ângulos de Euler [19]

3.2.1. PICCOLO

O Piccolo é um piloto automático comercial, desenvolvido pela CCT (*Cloud Cap Technology)*, que é uma subsidiária integral da *Goodrich Corporation*. O Piccolo é um piloto automático bastante completo, uma vez que reúne, numa pequena caixa, o núcleo do piloto automático, sensores de voo e de navegação, comunicação *wireless* e interfaces para *payload* [20]. Existem versões diferentes do Piccolo, como pode ser visualizado na Figura 28. O Piccolo II custa aproximadamente o dobro do Piccolo SL, possuindo características que o Piccolo SL não possui. No momento de adquirir um destes pilotos automáticos, é necessário avaliar as necessidades da plataforma onde será integrado e analisar qual destes responde melhor a essas mesmas necessidades. O Piccolo comunica com a GCS (*Ground Control Station*) através de um *link* de rádio, e esta é ligada por porta série a um computador onde está instalado o software PCC (*Piccolo Command Center)* desenvolvido pela CCT. Este é o software de comando e monitorização da aeronave. É de Salientar que o software de operação desenvolvido no LSTS permite neste momento substituir o PCC, pois foi sendo desenvolvido no sentido de integrar o Piccolo na sua arquitetura como pode ser visto mais à frente neste trabalho. O PITVANT e o LSTS possuem vários piccolos II e SL integrados nas suas aeronaves devido à sua robustez e fiabilidade.

Figura 28 Piccolo II e Piccolo SL [20]

3.2.2. ARDUPILOT MEGA 2

O *ardupilot mega2* é um piloto automático comercial de baixo custo produzido pela *DIY Drones* (ver Figura 29) É uma versão do *arduino* exclusiva para controlar aeronaves. Utiliza um software de código aberto, designado *arduplane*, para controlar aeronaves de asa fixa, que permite o seguimento de *waypoints* 3D, bem como o envio de comandos em tempo real para a aeronave [22]. O *ardupilot mega2* comunica através de um *link* de rádio com um recetor ligado por USB (*Universal Serial Bus*) ao computador. O *ardupilot mega2* pode ser operado através de duas aplicações, o *Ardupilot* M*ission Planner e o QGround Control*. À semelhança do Piccolo, o *ardupilot mega2* também pode ser operado através do Neptus. O LSTS possui vários *ardupilot mega2* nas suas plataformas pois, como possuem código aberto, são excelentes cobaias para integração de algoritmos desenvolvidos no LSTS para resolver necessidades existentes no mesmo.

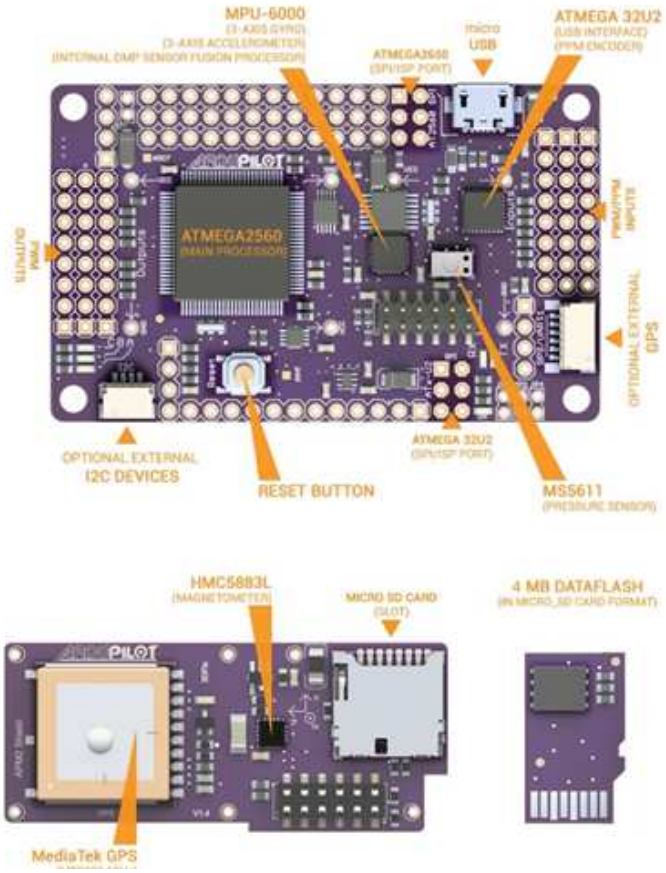

Figura 29 Ardupilot Mega 2

3.2.3. SISTEMA COMPUTACIONAL

Hoje em dia, a operação autónoma de aeronaves é mais facilmente integrada recorrendo a um ou vários sistemas computacionais, onde são agregados os dados dos sensores e executados os diversos algoritmos do piloto automático. Em [23] utilizam sistemas computacionais tipo PC-104 a bordo das suas aeronaves, desempenhando funções de: primeiro, controlo de voo e algoritmos de localização; segundo, processamento de dados dos sensores, terceiro, fusão de informação descentralizada, quarto, missões dinâmicas e PT.

No PITVANT e no LSTS são integrados nas plataformas um dos dois sistemas computacionais existentes, a escolha desta integração depende da necessidade computacional que cada aeronave possua, os sistemas computacionais são o PC104 e a IGEP.

Durante a realização deste trabalho, estes sistemas computacionais serão utilizados para ser possível realizar testes em campo, pois é neles que está instalado o DUNE que corre nos veículos, e que é fundamental para interação com o piloto automático.

PC-104

O PC-104 é um sistema computacional embebido controlado pelo PC-104 *Consortium* que obedece a dimensões, localização dos furos de assemblagem, tipo de fornecimento de energia, entre outros [24]. O PC-104 utilizado no LSTS possui um processador Geode, com arquitetura X86. Por ter dimensões pré estabelecidas entre outros fatores, o PC-104 possui inúmeros fabricantes a desenvolver e construir não só o sistema computacional, mas também diversos módulos, nomeadamente, expansão USB/Firewire, expansão *frame grabber*, entre outros, que se vão encaixando para servir as necessidades de uma determinada aplicação. Este sistema computacional é bastante utilizado em sistemas de controlo industrial e em veículos. Nos pilatus, o software DUNE (apresentado mais à frente neste trabalho) está instalado num PC-104 (ver Figura 30).

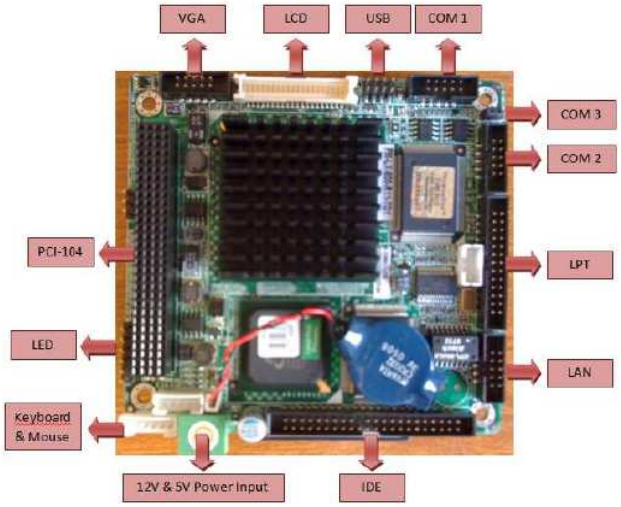

Figura 30 Sistema Computacional PC-104

IGEP

A IGEP é um sistema computacional de processamento industrial, consegue ter desempenhos equivalentes a muitos computadores portáteis, sem ocupar o mesmo espaço e sem produzir ruído [25]. É um sistema computacional bastante importante para ter a bordo de uma aeronave autónoma. Nos cularis o software DUNE (apresentado mais à frente neste trabalho) está instalado numa IGEP (ver Figura 31). A IGEP utilizada no LSTS possui o processador OMAP3530 da *Texas Instruments*, e a sua arquitetura é ARM.

3.3. SOFTWARE DESENVOLVIDO NO LSTS

Em [15] e [16] é possível observar os desenvolvimentos de *software* efetuados e os previstos entre a FEUP e a AFA.

3.3.1. NEPTUS

O Neptus, desenvolvido no LSTS, é uma *Framework* distribuída de comando, controlo e monitorização, para operações com veículos, sistemas e operadores humanos numa rede. Forma uma infra-estrutura de software para a interface com os operadores no desenvolvimento de veículos autónomos. Foi inicialmente desenvolvido para AUVs, mas neste momento já integra os UAVs na sua arquitetura. O ambiente gráfico para interação com o utilizador é designado por consola, na consola existe o mapa da área de missão, existem informações relativas à atitude, posição e velocidade do veículo, assim como

Figura 31 Sistema Computacional IGEP

diversos *icons* que correspondem aos diversos *plugins* existentes. O operador pode escolher qual a consola mais adequada à missão/veículo e quais os *plugins* de interesse para a mesma.

O Neptus suporta neste momento diversos tipos de veículos em simultâneo, nomeadamente, veículos aéreos, sub-marinos e de superfície, permitindo deste modo a existência de missões conjuntas [28] (ver Figura 32).

O objectivo do Neptus é permitir aos operadores desenhar planos de missão, iniciá-los quando desejado, e supervisioná-los. O operador Neptus pode também ser envolvido no planeamento de missões através de manobras que envolvam operações remotas por parte dos operadores.

O Neptus suporta todas as fases de vida de uma missão, nomeadamente, representação do mundo, planeamento, simulação, execução e posterior análise dos dados [14]. Na Figura 33 está ilustrado o aspeto da consola UAV *Basic* existente no Neptus.

O Neptus foi desenvolvido em Java, e pode ser instalado em várias plataformas, nomeadamente Linux, Windows, Android.

3.3.2. DUNE

Como já foi referido anteriormente, o DUNE foi desenvolvido no LSTS e é o *software* que corre a bordo dos veículos. É responsável não só por todas as interações com sensores, *payload*, e atuadores, mas também para comunicação, navegação, controlo, manobras, execução de planos e supervisão do veículo [28].

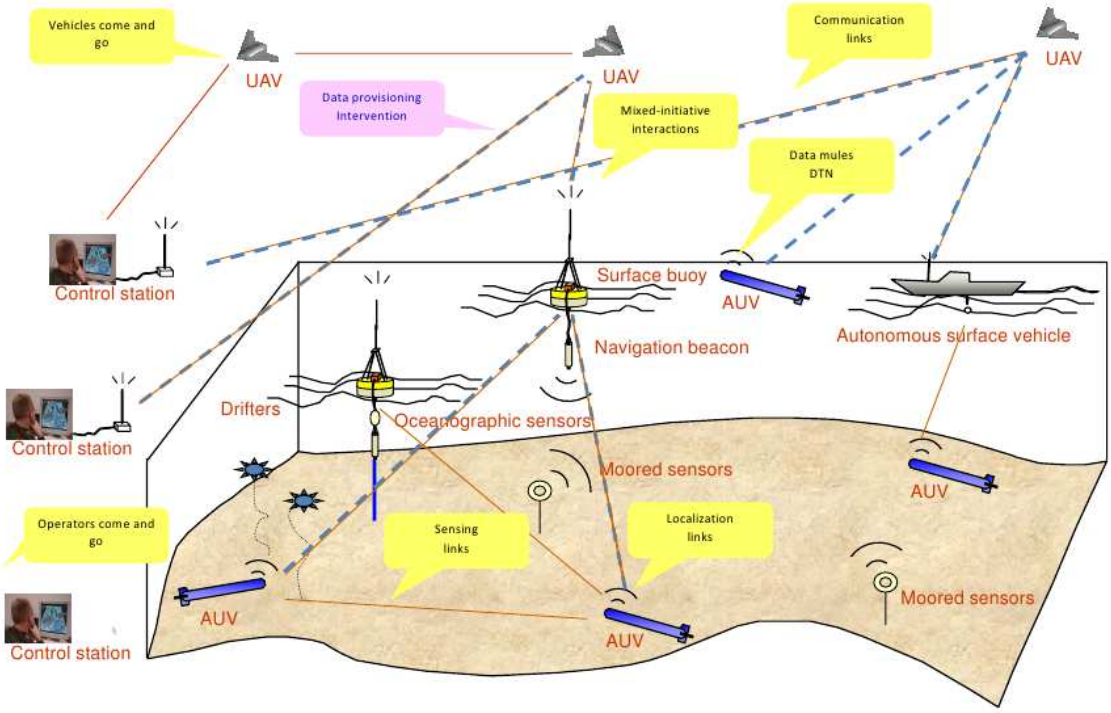

Figura 32 Conceito de rede de veículos presente no LSTS

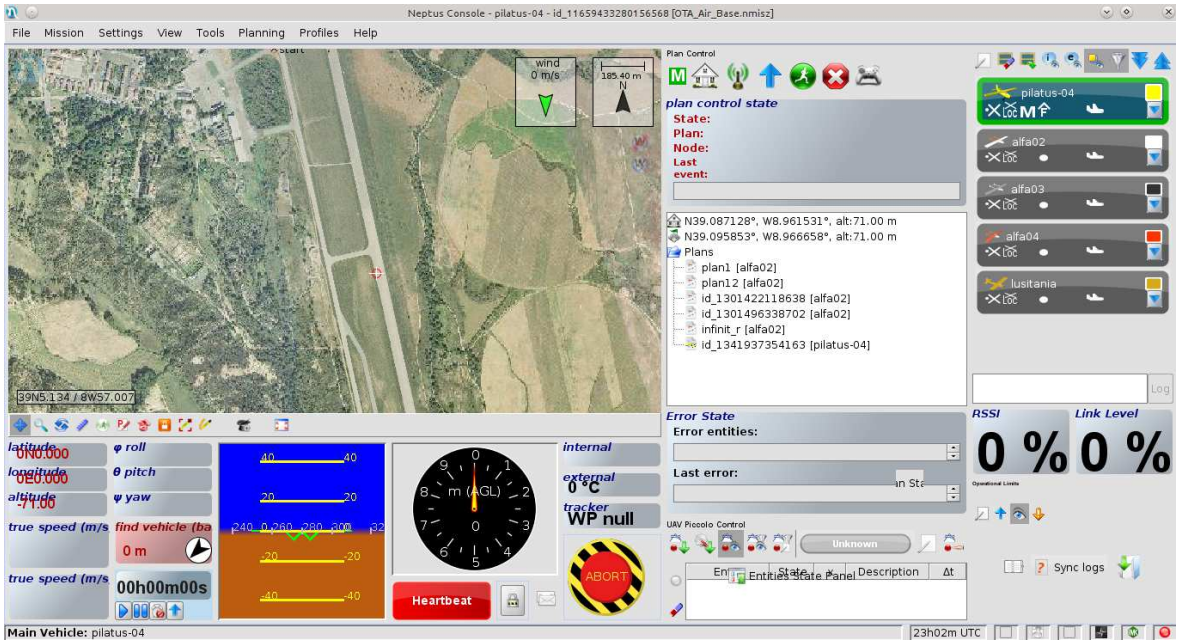

Figura 33 Consola de operação UAV Basic

No núcleo do DUNE existe uma camada abstrata, permitindo desta forma a sua independência da arquitetura do CPU (*Central Processing Unit)* assim como do sistema operativo. O Dune funciona como um mecanismo de transporte de mensagens, onde tarefas

independentes correm em *threads* de execução separadas. Todas as tarefas estão ligadas a um barramento, onde podem publicar ou subscrever mensagens (ver Figura 34) [28].

O DUNE comunica com o Neptus através do protocolo de mensagens IMC (apresentado mais à frente), comunica com o Piccolo através de um protocolo que traduz IMC para o protocolo proprietário da CCT e com o Ardupilot Mega2 através de um protocolo que traduz IMC para zigbee.

Inicialmente o DUNE estava apenas direcionado para AUVs, mas atualmente existem imensas tarefas DUNE desenvolvidas para UAVs, o que torna possível o controlo dos mesmos através do NEPTUS. Existem também muitas tarefas que tanto funcionam com AUVs como com UAVs, mediante configurações distintas.

3.3.3. IMC

O IMC (*Inter-Module Communication Protocol)* é um protocolo orientado a mensagens (ver Figura 36), desenvolvido e implementado no LSTS, para comunicação entre veículos heterogéneos, sensores, operadores e todos os módulos de software presentes no LSTS. O protocolo IMC compreende diferentes grupos de mensagens lógicas para veículos em rede e operações com sensores. Define uma infra-estrutura que é modular e fornece camadas diferentes para o controlo e aquisição de dados dos sensores. A Figura 35 mostra a sequência de mensagens IMC existentes no AUV *Seascout Light.*

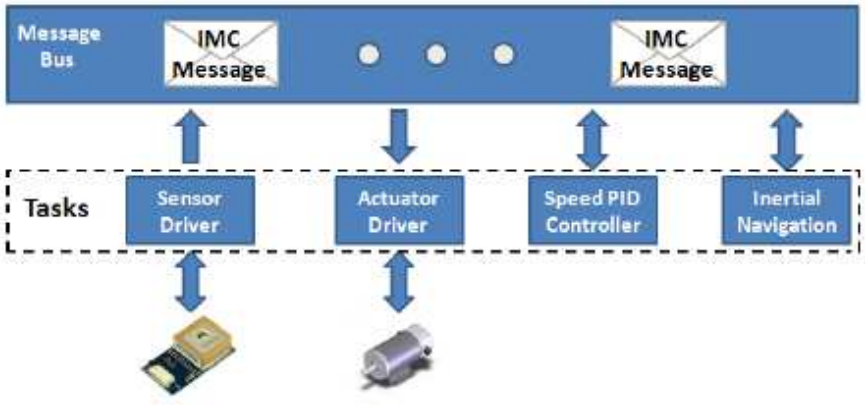

Figura 34 Conceito de transporte de mensagens por trás da implementação de tarefas DUNE

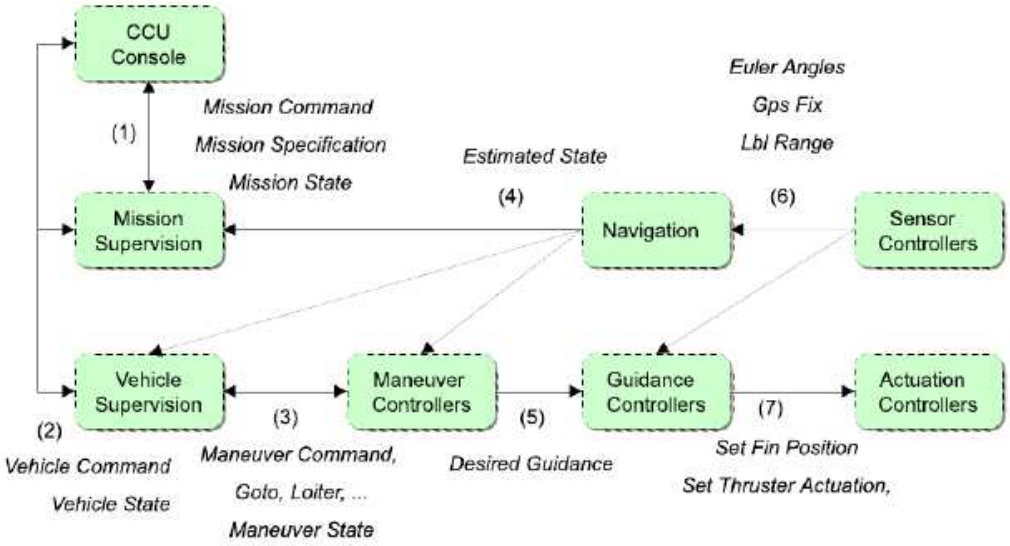

Figura 35 Fluxo de mensagens IMC do *Seascout light*

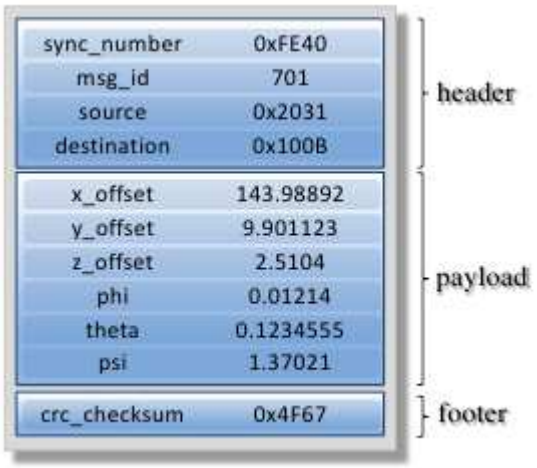

Figura 36 Exemplo da estrutura de uma mensagem IMC [28]

4. ABORDAGEM

4.1. DEFINIÇÃO DO PROBLEMA

Com o crescimento do projeto PITVANT, torna-se cada vez mais importante aumentar a autonomia de bordo dos UAVs. No Projeto ainda não existem desenvolvimentos em termos de PT evitando obstáculos. Este tipo de desenvolvimentos são essenciais em missões que possuam espaço aéreo interdito, deteção de incêndios, missões em ambientes montanhosos (ver Figura 37), entre outras, e são importantes não só para o aumento da autonomia de bordo da plataforma, mas também para a diminuição da carga de trabalho do piloto. O planeamento pode ser executado *offline*, ou *online*. No primeiro caso o planeamento pode ser executado na estação de terra e enviado posteriormente para o veículo. Neste modo, o trajeto é gerado no MATLAB e são enviados os *waypoints* do mesmo para o Neptus, sendo este responsável pelo envio dos *waypoints* para o controlador de seguimento de *waypoints* do piloto automático (ver Figura 38). No segundo caso, o PT é executado no próprio veículo, que utiliza sensores para identificação de obstáculos e em tempo real calcula a sua trajetória. Neste modo, o DUNE calcula a trajetória com base na informação disponibilizada pelos sensores de obstáculos e envia os *waypoints* do trajeto para o piloto automático (ver Figura 39).

Como ainda estão a ser desenvolvidos os algoritmos de processamento de imagem e de deteção de características nas imagens, o algoritmo de planeamento de trajetórias evitando obstáculos será executado *offline*.

O problema que se pretende resolver é a falta de sistemas para PT evitando obstáculos no contexto do projeto PITVANT. Para solucionar este problema pretende-se desenvolver algoritmos acerca da temática e integrá-los nas ferramentas de software existentes no LSTS apresentadas no capítulo três.

4.2. PROCESSO DE ENGENHARIA DE SISTEMAS

Durante o desenvolvimento deste trabalho será utilizado o método de engenharia de sistemas presente na Figura 40, recorrendo à norma 1220 do IEEE [54].

Inicialmente são introduzidas as entradas no processo, ou seja, qual o problema que se pretende resolver, são depois analisados os requisitos e identificados os requisitos necessários para solucionar o problema; de seguida é feita uma análise funcional do sistema e são validadas as funções necessárias e as relações entre elas; por fim é desenvolvida a solução e validada em função dos requisitos definidos.

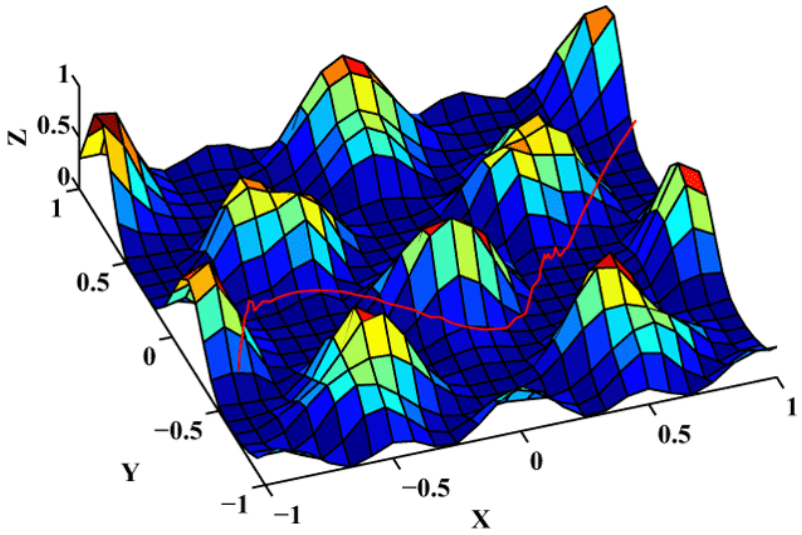

Figura 37 Exemplo do planeamento de uma trajetória em ambiente montanhoso [55]

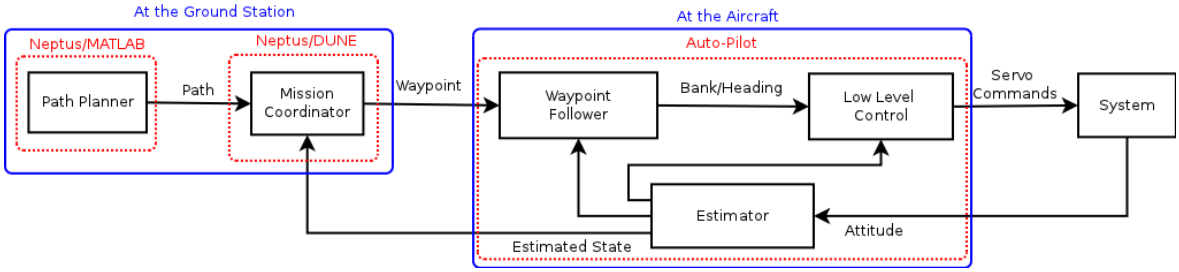

Figura 38 Arquitetura do planeamento de trajetórias *offline*

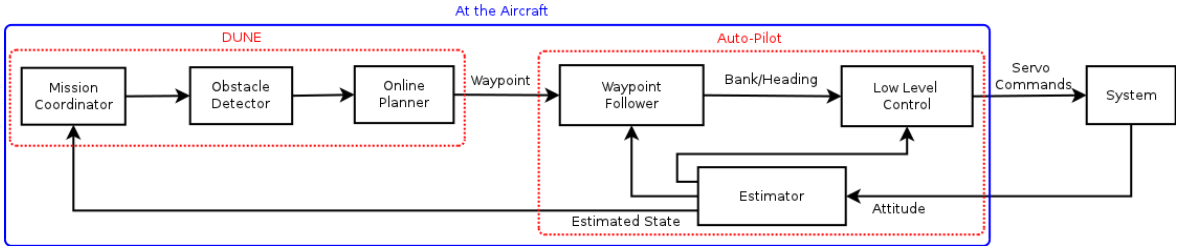

Figura 39 Arquitetura do planeamento de trajetórias *online*

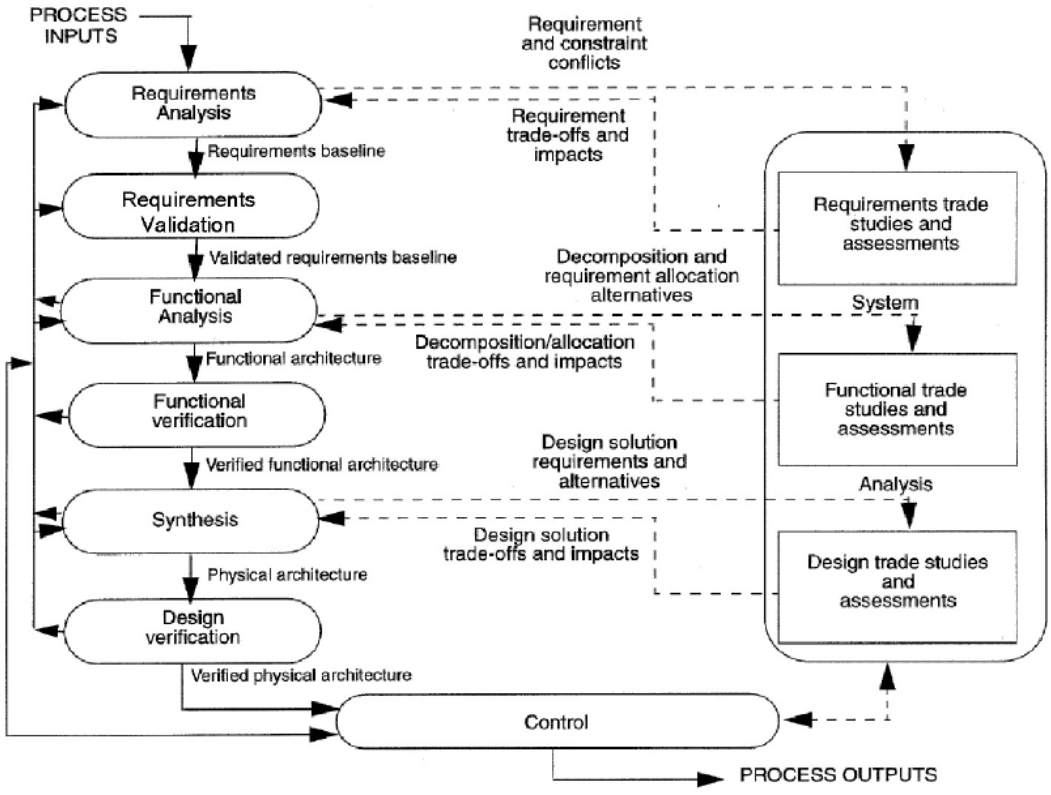

Figura 40 Processo de Engenharia de Sistemas

4.3. REQUISITOS

Como foi apresentado na secção 4.2, a identificação dos requisitos são a fase posterior às entradas no processo de engenharia de sistemas. Muitas vezes os requisitos servem de etapas para a fase de implementação.

4.3.1. USER STORY

Imaginemos um cenário de operações onde é pretendido voar um UAV desde o interior norte até ao interior sul de Portugal para deteção de incêndios, ou para transporte de mercadorias, ou para vigilância das linhas de alta tensão, entre outras aplicações, e que durante o trajeto existem montanhas, zonas interditas a voos, turbinas eólicas, entre outros obstáculos.

Se avaliarmos os requisitos necessários, em termos computacionais, para cumprir esta missão, podemos identificar os requisitos necessários à resolução do problema inicial. É de notar que os requisitos identificados para esta missão são praticamente idênticos a qualquer outra missão que necessite de PT evitando obstáculos.

Os requisitos identificados para o cenário em causa são:

- O piloto tem de ter a possibilidade de inserir os waypoints de partida e de destino, os limites espaciais e os obstáculos existentes durante o trajeto (planeamento offline).
- O algoritmo deve poder suportar obstáculos convexos e não convexos
- O algoritmo deve garantir uma margem de segurança configurável, em torno de todos os obstáculos como pode ser vista na Figura 41.
- Como o planeamento é *offline*, o algoritmo deve ser inserido na arquitetura do Neptus

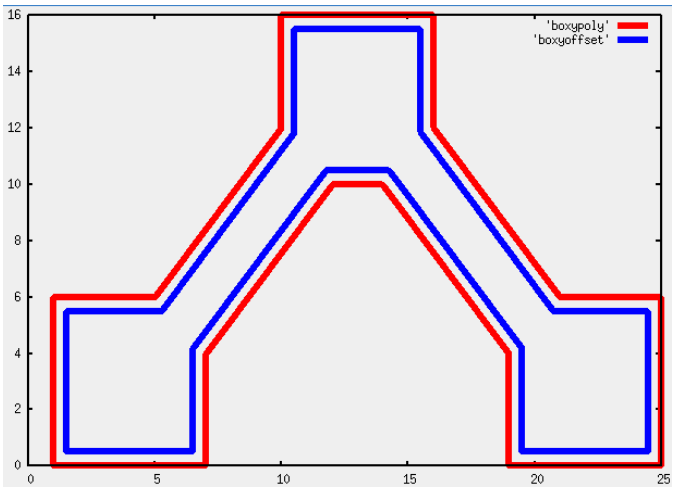

Figura 41 Exemplo do offset de segurança construído em torno dos polígonos

- O algoritmo deve ser capaz de produzir um trajeto que não se sobreponha a nenhum obstáculo.
- Caso não exista trajeto possível entre dois pontos, deve ser apresentada uma mensagem de erro.
- A saída do algoritmo deve ser um plano de voo constituído por uma sequência de *waypoints.*

4.4. DESCRIÇÃO DA SOLUÇÃO

Para o desenvolvimento do algoritmo de PT será utilizado o software MATLAB, por ser uma boa ferramenta de "prototipagem rápida".

Neptus

Como foi referido no capítulo três, o Neptus é um software para comando, controlo e monitorização de missões usando veículos autónomos. Suporta todas as fases de vida de uma missão, nomeadamente, a representação do mundo, planeamento, simulação, execução e posterior análise dos dados. Na Figura 42 está apresentada a arquitetura simplificada do Neptus. É de notar que o código fonte está separado de todos os *plugins* existentes.

Para o desenvolvimento do algoritmo de PT, será criado um novo *plugin* designado SPP (*ShortestPathPlanner)*, que será responsável pela criação de um menu, permitindo ao

piloto a introdução dos *waypoints* definidos nos requisitos. As coordenadas de todos esses *waypoints* são inseridas num ficheiro. O MATLAB recebe esse ficheiro e calcula a trajetória. As coordenadas dos *waypoints* identificados para a trajetória são também inseridas num ficheiro, ao qual o Neptus tem acesso. Depois de ler o ficheiro, o Neptus desenha no mapa da consola os *waypoints* do trajeto. Se o piloto concordar com o trajeto calculado, pode enviar o trajeto para o DUNE através de IMC, o DUNE por sua vez envia os waypoints, ou referências de *bank/heading* para o piloto automático (ver Figura 43).

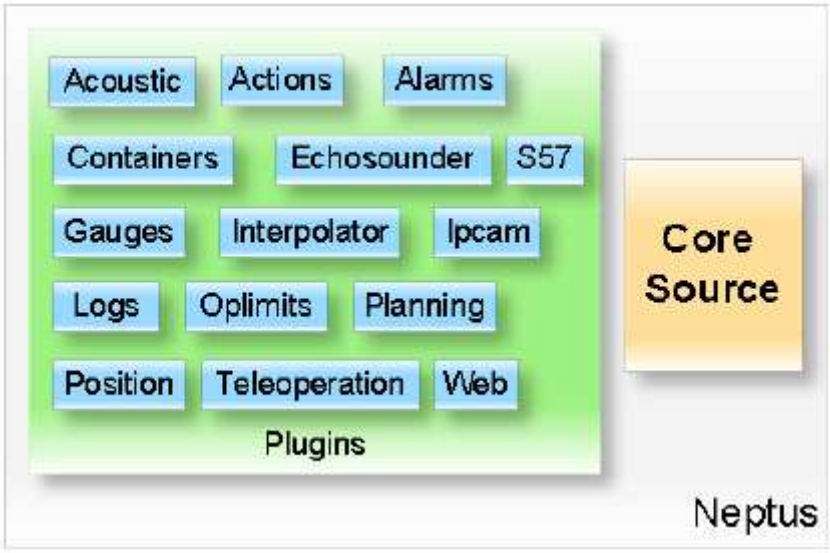

Figura 42 Arquitetura simplificada do Neptus [13]

MATLAB

Quando a função de MATLAB desenvolvida recebe o ficheiro com as coordenadas de interesse para o cálculo da trajetória necessita de criar uma margem de segurança em torno de cada obstáculo, de criar uma grelha de células regulares com dimensão associada à aeronave utilizada, e de iniciar o algoritmo de procura do trajeto mais curto A*.

CÉLULAS REGULARES

As células regulares serão o método utilizado para a aplicação do método de procura A*, pois possuem maior resolução que as células exatas, verticais ou retangulares.

A*

O método de PT que será utilizado é o A*, apresentado no capítulo dois. Como os obstáculos serão conhecidos à partida, não será implementado um algoritmo de pesquisa dinâmico, tipo D*. O *greedy best-first* apesar de não garantir que o trajeto seja o mais curto, pode ser uma opção a ter em conta devido à diminuição do tempo computacional associado.

4.5. MÉTRICAS

As métricas são indicadores que permitem avaliar o desempenho dos algoritmos. Para a avaliação dos algoritmos desenvolvidos neste trabalho serão utilizadas como métricas de avaliação, o tempo de cálculo das trajetórias, assim como a distância obtida para cada trajeto.

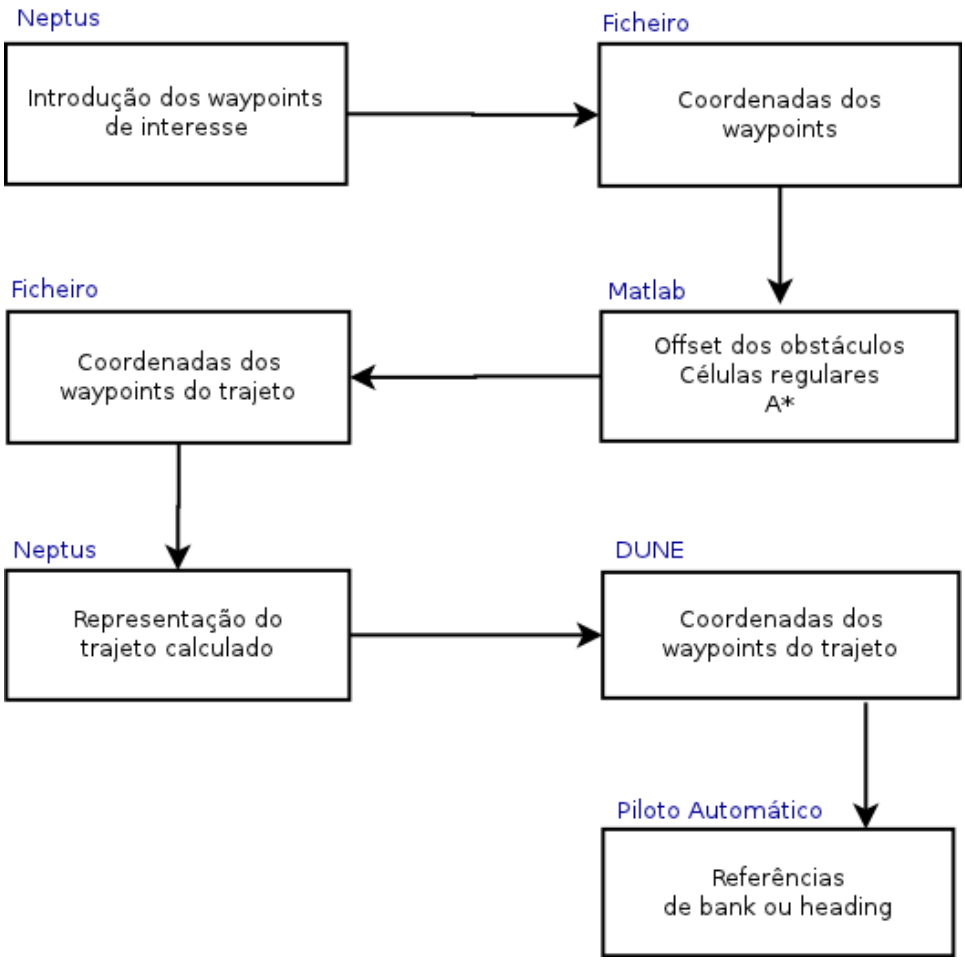

Figura 43 Decomposição funcional do sistema

5. IMPLEMENTAÇÃO

Após identificação dos requisitos fundamentais para a resolução do problema proposto, e das funcionalidades necessárias para que os requisitos sejam cumpridos, inicia-se a fase de implementação do sistema.

Os principais desenvolvimentos foram realizados no Neptus e no MATLAB, pelo que serão os sistemas abordados neste capítulo.

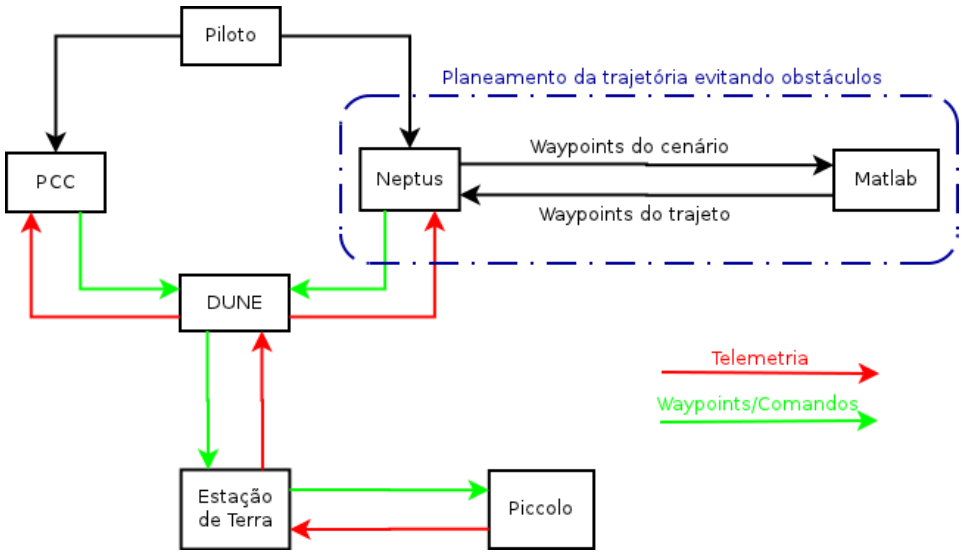

Figura 44 Arquitetura do sistema com planeamento *offline*

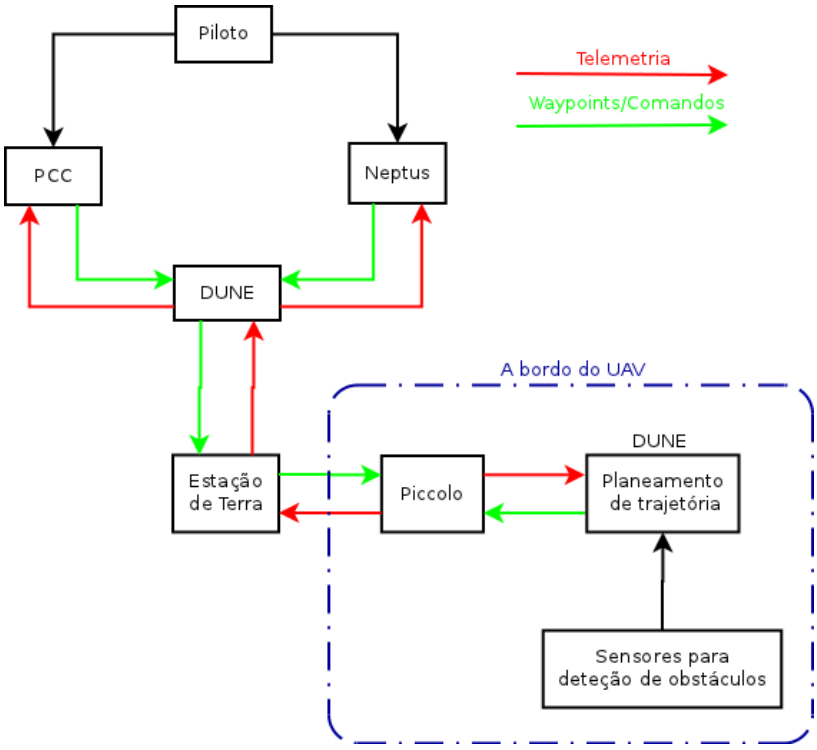

Figura 45 Arquitetura do sistema com planeamento *online*

Na Figura 44 está apresentada a arquitetura do sistema com o PT a ser executado *offline*. Quando o Neptus recebe do MATLAB, o plano de voo constituído por *waypoints*, envia esse plano por IMC para o DUNE, sendo este responsável por enviar para o piccolo os *waypoints*, ou referências de bank/heading.

Na Figura 45 está apresentada a arquitetura do sistema com o PT a ser executado *online*. Para que este planeamento possa ser executado, é necessário que o DUNE corra a bordo da aeronave, instalado num PC, e que a aeronave possua sensores para deteção de obstáculos e.g. câmaras, assim como algoritmos de processamento de imagem inseridos no DUNE. Neste cenário o Neptus e o DUNE em terra deixam de ser essenciais à operação, sendo apenas necessário o PCC para monitorização do estado da plataforma.

5.1. NEPTUS

O Neptus está estruturado de forma modular (ver Figura 42), de forma a facilitar a sua manutenção e expansibilidade, tentando manter a maior independência possível entre os diversos módulos.

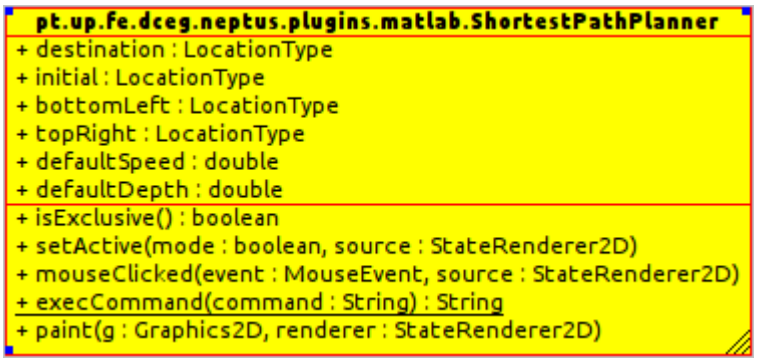

Figura 46 Membros e métodos implementados no plugin *shortestPathPlanner*

Respeitando esta filosofia foi desenvolvido o novo *plugin* SPP na estrutura do Neptus. Este plugin cria um menu na consola do Neptus, permitindo ao piloto identificar no mapa da consola os *waypoints* de interesse para a missão, nomeadamente o ponto inicial, final, fronteiras e obstáculos. Este plugin é responsável por iniciar o MATLAB, executar o algoritmo de PT desenvolvido, pela receção dos *waypoints* obtidos e a sua apresentação no mapa da consola do Neptus.

Como o Neptus é desenvolvido em java, este plugin também foi criado nesta linguagem. Na Figura 46 estão apresentados os métodos e membros implementados.

Na Figura 47 está representada a arquitetura funcional do *plugin.* Na Figura 48 está apresentada a consola UAV *Basic* do Neptus, com a apresentação do menu do *plugin* desenvolvido.

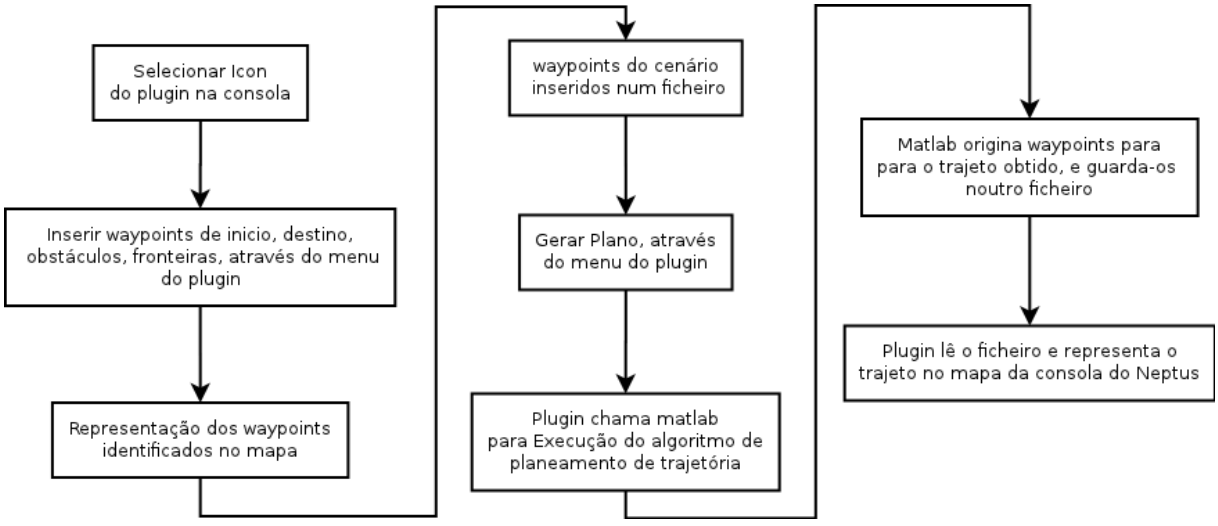

Figura 47Arquitetura funcional do *plugin* desenvolvido no Neptus

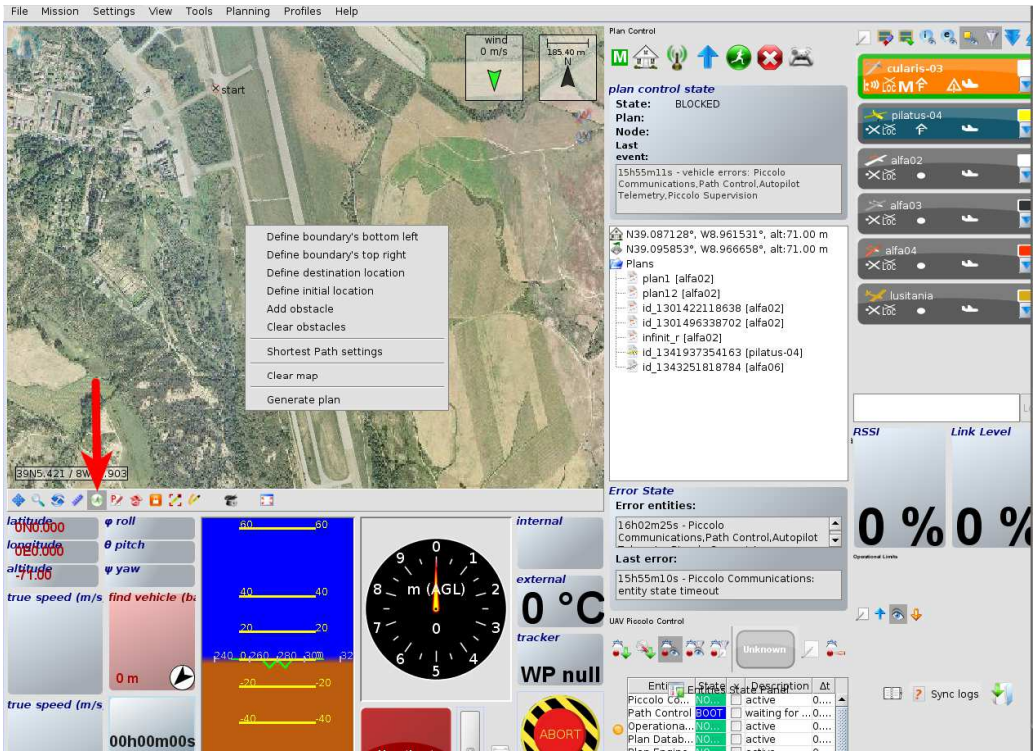

Figura 48 Menu do *plugin* desenvolvido no Neptus

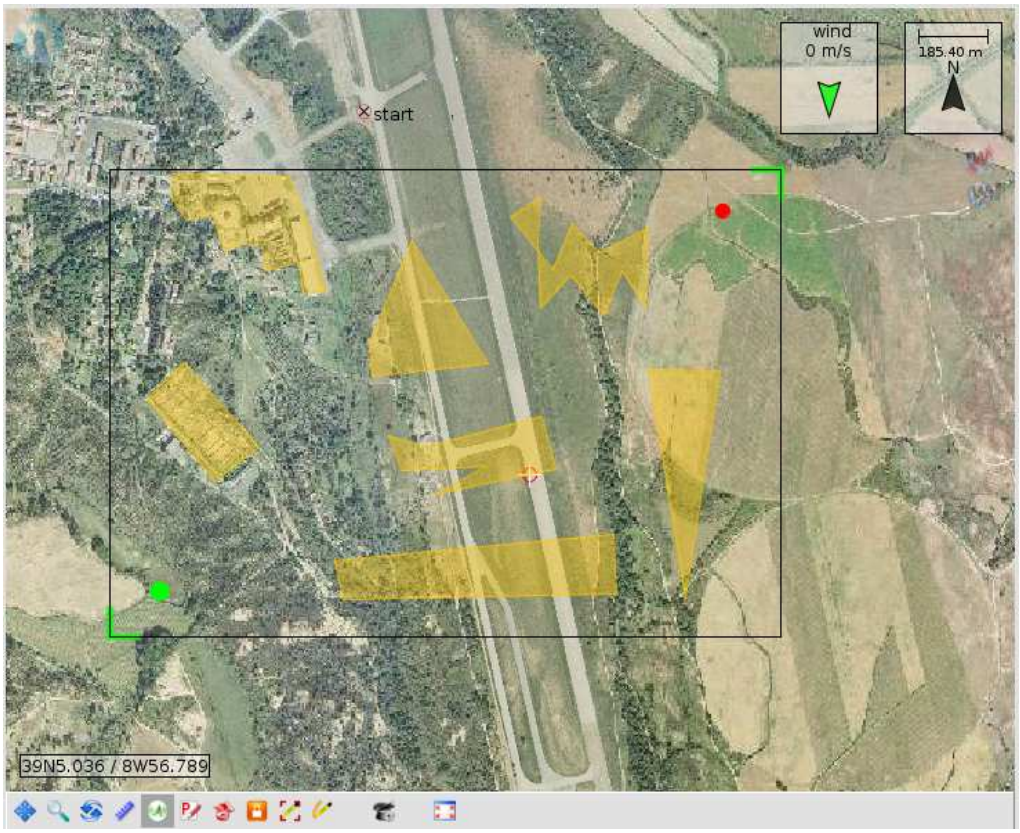

Figura 49 Definição dos obstáculos, limites espaciais, ponto inicial e final, através do plugin do Neptus

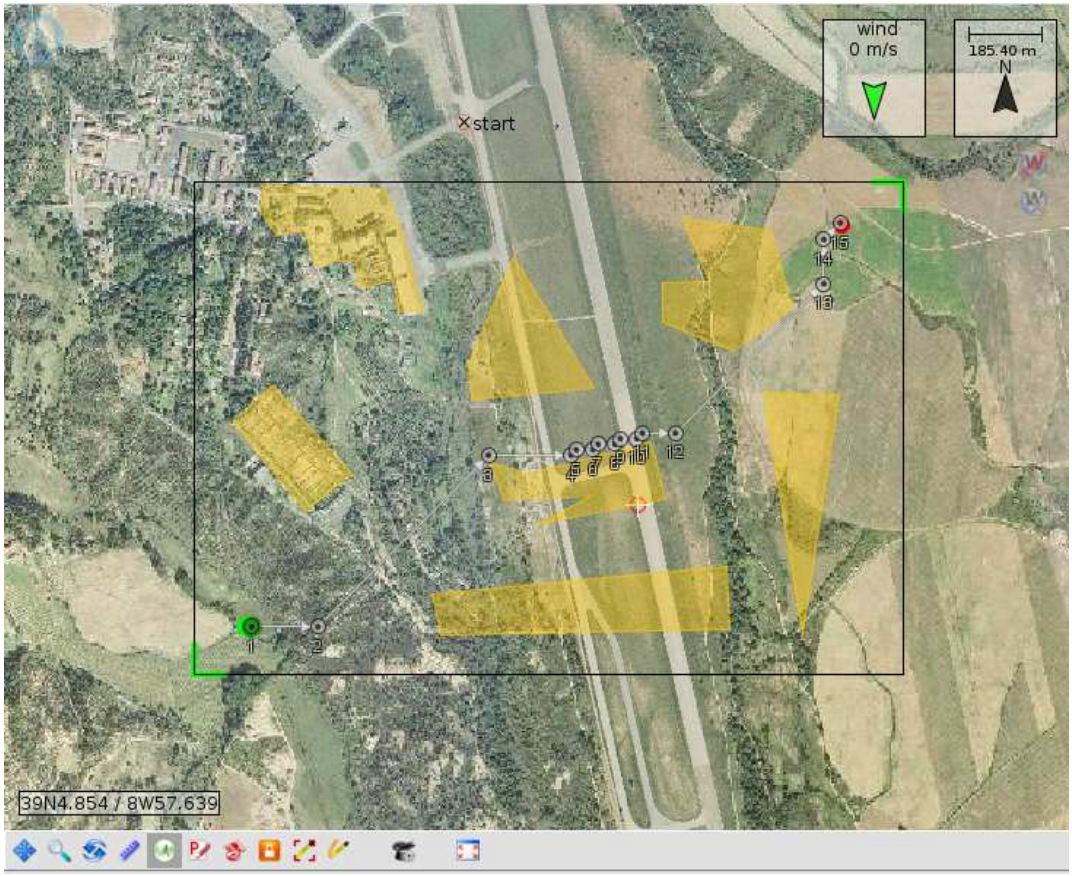

Figura 50 Trajeto obtido através do algoritmo de planeamento de trajetória evitando obstáculos, desenvolvido em MATLAB

A seta vermelha presente na Figura 48 representa o icon do *plugin* criado no Neptus, Depois de selecionar o *icon*, o menu presente na mesma figura é apresentado. Na Figura 49 estão representados a amarelo os obstáculos simulados, os limites espaciais, e o ponto inicial e de destino. No Anexo A está apresentado um tutorial de utilização do *plugin* SPP.

Depois de ser selecionada a opção *generate plan* presente no menu do *plugin* SPP, o algoritmo desenvolvido em MATLAB é executado com as entradas presentes no ficheiro e, depois de calculado o trajeto, os waypoints obtidos são inseridos noutro ficheiro que o *plugin* SPP interpreta para representar o trajeto planeado no mapa da consola do Neptus, como pode ser constatado na Figura 50.

5.2. MATLAB

Quando o plugin SPP do Neptus chama o MATLAB, ele está a chamar a função principal do algoritmo de PT, que se designa por *shortestpath.* Esta função possui dois argumentos: o primeiro argumento é o ficheiro criado pelo *plugin* SPP, que contém o nome da aeronave, as coordenadas dos obstáculos, dos limites espaciais, do ponto inicial e de destino; já o segundo argumento é um ficheiro criado por esta própria função, onde estão descritas as coordenadas dos waypoints identificados para o trajeto.

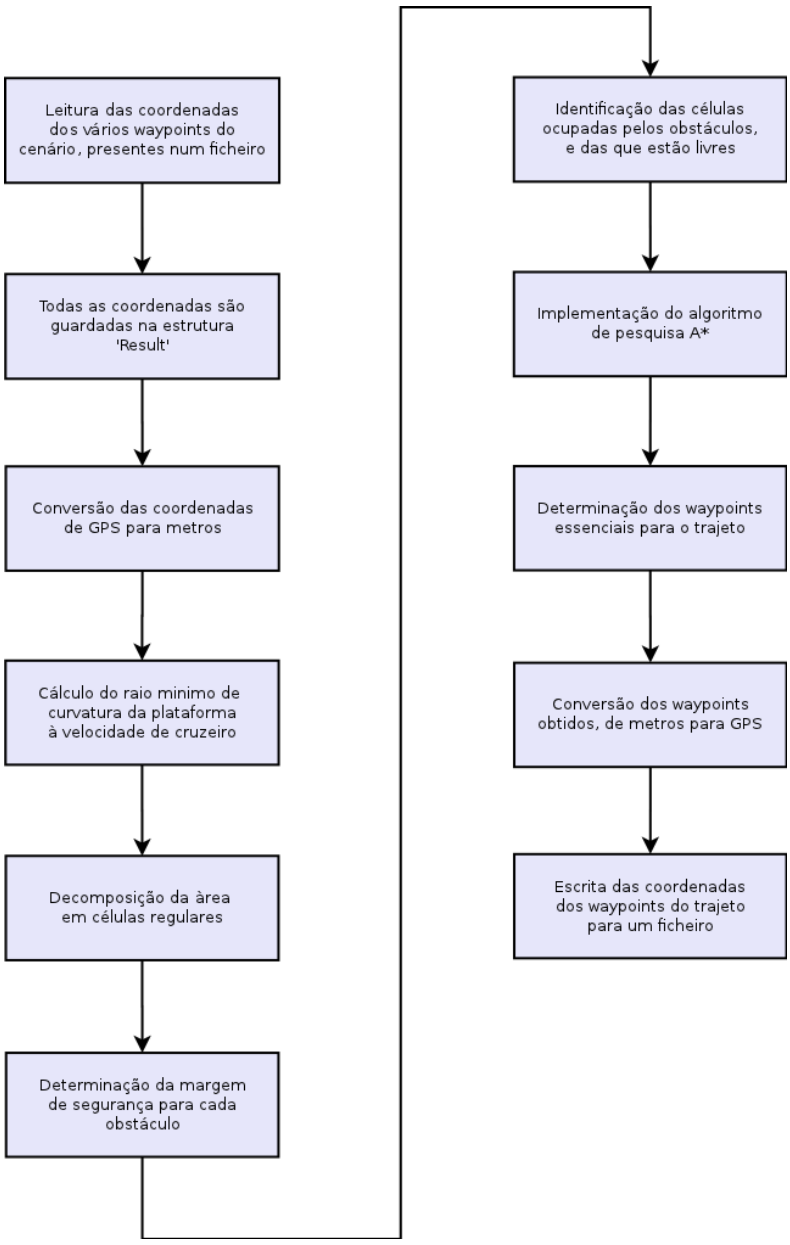

Figura 51 Arquitetura do algoritmo de planeamento de trajetória implementado em MATLAB

Na Figura 51 está representada a arquitetura do algoritmo de PT implementado no MATLAB. No inicio, todas as coordenadas dos obstáculos, das posições inicial e final, e dos limites espaciais do cenário são lidas de um ficheiro com recurso à função *fetchIniData* e são guardadas numa estrutura. São posteriormente convertidas de GPS para metros, para facilitar todas as operações seguintes necessárias.

5.2.1. RAIO MÍNIMO DE CURVATURA DA AERONAVE

No desenvolvimento deste algoritmo, o espaço definido pelos limites do cenário de operação, foi decomposto em células regulares, e a aeronave desloca-se entre o centro de cada célula e o centro das células adjacentes. Suponhamos que a plataforma está a deslocar-se no sentido Sul-Norte, como está representado na Figura 52, e pretende virar 90 graus para Este. Para que a plataforma consiga deslocar-se para o centro da célula pretendida, o tamanho de cada célula precisa ser maior ou igual ao raio mínimo de curvatura da plataforma à velocidade de cruzeiro, como pode ser consultado através da seta castanha da Figura 52, caso contrário poderá não conseguir cumprir esse objetivo.

A solução apresentada não garante que o UAV não colida com os obstáculos caso exista vento. Para que o efeito do vento seja tido em conta, é necessário corrigir o cálculo do raio mínimo de curvatura da plataforma à velocidade de cruzeiro, multiplicando o tempo de curvatura pela velocidade máxima do vento.

Como o PT é feito offline, é necessário garantir que o trajeto planeado só é executado quando não existirem obstáculos entre a posição do UAV e a posição inicial da trajetória calculada, e que ao redor do UAV existe uma área suficientemente grande sem obstáculos para que este adquira a direção correta.

O raio mínimo de curvatura de uma aeronave à velocidade de cruzeiro é dado pela equação 4, onde:

- $\mathbf{V_{cruise}}$ – Velocidade cruzeiro da aeronave
- **g** – Aceleração da gravidade
- $\mathbf{bank_{max}}$ – Ângulo máximo segundo o eixo do X

$$raio_{min} = \frac{V_{cruise}^2}{g * \tan(bank_{max})} \quad (4)$$

A velocidade de cruzeiro pode ser calculada através da equação 5, onde:

- **m** – Massa da aeronave
- **CL_{cruise}** – Coeficiente de sustentação da aeronave
- **WA** – Área de cada asa
- **ρ** – Densidade do ar

$$V_{cruise}^2 = \frac{m * g}{CL_{cruise} * WA * \rho} * 2 \quad (5)$$

Após substituição da equação 5 na equação 4, o raio mínimo de curvatura à velocidade atual, pode ser obtido através da equação 6.

$$raio_{min} = \frac{2 * m}{CL_{cruise} * WA * \rho * \tan(bank_{max})} \quad (6)$$

Quando a função *shortestpath* lê o ficheiro criado pelo *plugin* SPP no Neptus, obtém o nome da plataforma que está selecionada para voo no Neptus. No mesmo diretório da função *shortestpath* existe um ficheiro XML para cada plataforma, onde estão acessíveis todos os valores necessários para o cálculo do raio mínimo de curvatura à velocidade de cruzeiro de cada plataforma.

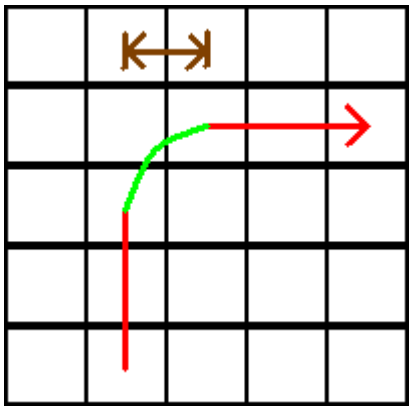

Figura 52 Exemplo elucidativo do raio mínimo de curvatura da plataforma à velocidade de cruzeiro

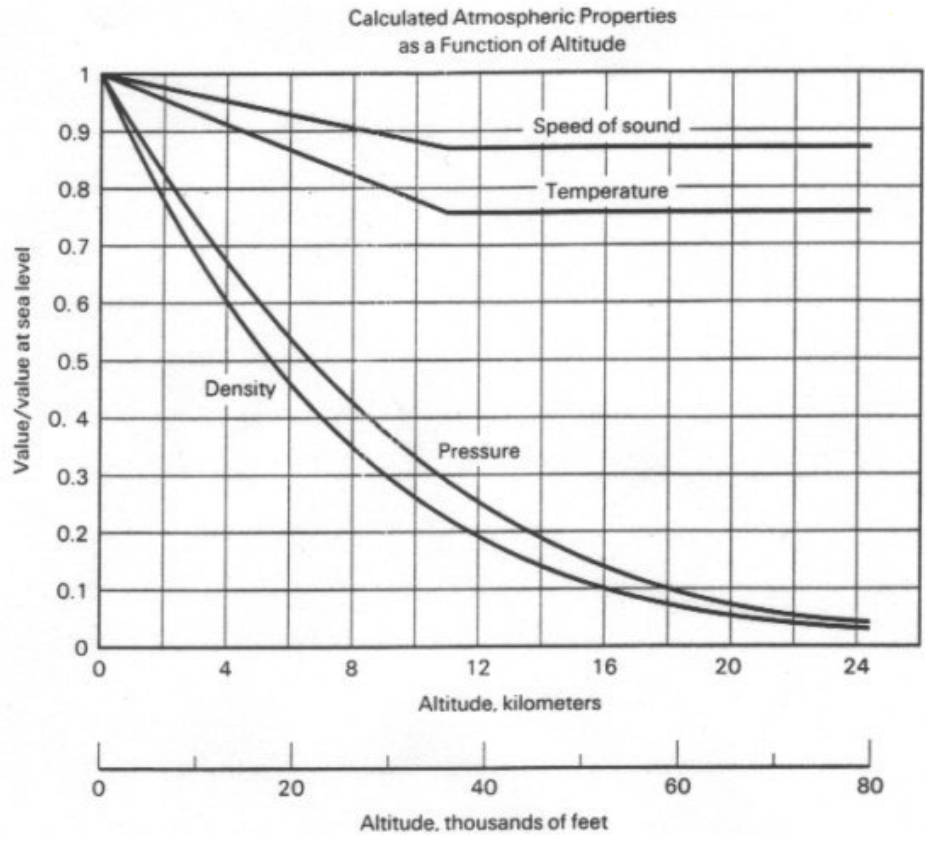

Figura 53 Densidade do ar em função da altitude [56]

A densidade do ar é o único parâmetro que varia durante o voo, pois depende da altitude a que a aeronave se encontra. Na Figura 53 é possível visualizar a variação da densidade do ar com a altitude. No caso deste trabalho, como as altitudes de voo das aeronaves do projeto são relativamente baixas, foi definido o valor fixo de 1kg/m^3 para a densidade do ar.

5.2.2. CÉLULAS REGULARES

A decomposição do espaço em células regulares é realizada após o cálculo do raio mínimo de curvatura à velocidade de cruzeiro para a plataforma em questão. O número de células obtidas para o eixo do X e para o eixo do Y é calculado dividindo a distância em X e em Y do espaço de operações pelo resultado da equação 6 (raio mínimo de curvatura). De seguida é criada uma matriz com o número de células calculadas. Todas as células são inicializadas a zero.

5.2.3. DISTÂNCIA DE SEGURANÇA EM TORNO DOS OBSTÁCULOS

Se não existir uma zona de segurança em torno dos obstáculos, onde não é permitido ao modelo simplificado da aeronave entrar, os algoritmos de PT podem produzir trajetos que coloquem em risco tanto os obstáculos como a aeronave.

No desenvolvimento do algoritmo *shortestpath* foi criado um método numérico, baseado em relações trigonométricas, para ser gerada a margem de segurança (*offset*) em torno dos obstáculos. Este método foi implementado em cada vértice de cada obstáculo. Na Figura 54 está representado este método aplicado ao vértice número 2 do obstáculo.

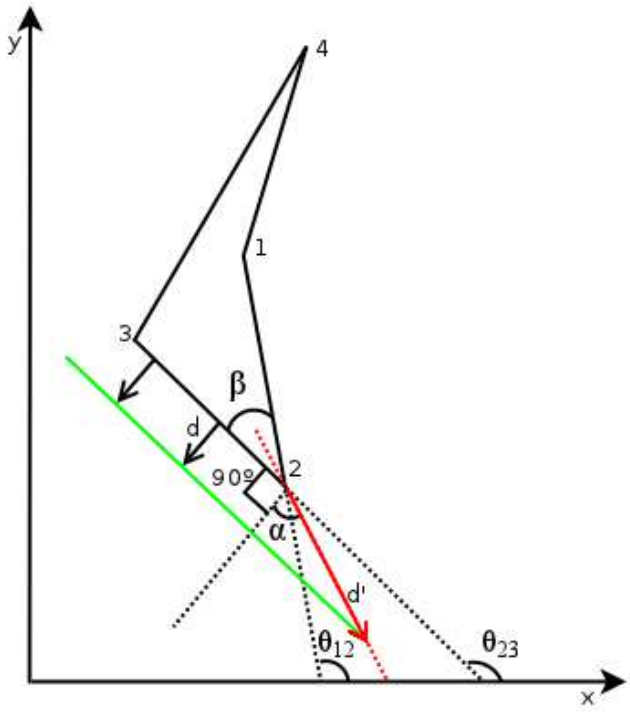

Figura 54 Representação da metodologia utilizada para calcular a distância de segurança em torno dos obstáculos

A primeira etapa deste método é calcular o ângulo θ (ângulo entre cada face do obstáculo e o eixo do X) para cada segmento de reta do obstáculo. A equação 7 apresenta o cálculo do ângulo θ para o segmento de reta 1-2.

$$\theta_{12} = \text{atan2}\,(Y_2 - Y_1\,,\, X_2 - X_1) \tag{7}$$

A segunda etapa deste método prende-se com o cálculo do ângulo β (ângulo existente entre dois segmentos de reta contíguos). O cálculo do ângulo β para o exemplo da Figura 54 está apresentado na equação 8.

$$\beta = 180 - (\theta_{23} - \theta_{12}) \tag{8}$$

De seguida é necessário calcular o ângulo α, recorrendo a equação 9.

$$\alpha = 90 - \beta / 2 \tag{9}$$

A distância de segurança existente em torno de cada obstáculo pode ser configurável. Durante o desenvolvimento do algoritmo foi considerada, por defeito, igual ao raio mínimo de curvatura da aeronave à velocidade de cruzeiro, este é o valor mínimo que garante que a aeronave não colide com os obstáculos (sem efeitos de vento). Esta distância está representada na Figura 54 com a letra **d**. Para ser possível determinar essa distância nos vértices que unem dois segmentos de reta (d'), é necessário utilizar a equação 10.

$$d' = d / \cos(\alpha) \tag{10}$$

Na Figura 55 está apresentado a azul um obstáculo, a verde a margem de segurança com valor igual ao raio mínimo de curvatura da aeronave à velocidade de cruzeiro e os pontos vermelhos representam os centros de cada célula (pontos entre os quais a aeronave pode-se deslocar). As setas vermelhas representam a trajetória da aeronave em duas situações distintas: primeiro, em cima a margem de segurança não chega ao centro das células, segundo, em baixo a margem de segurança chega ao centro das células. É de notar que se a aeronave possuir qualquer uma das direções apresentadas, irá entrar na margem de segurança, mas como esta tem o valor do raio mínimo de curvatura, a aeronave não irá colidir com o obstáculo.

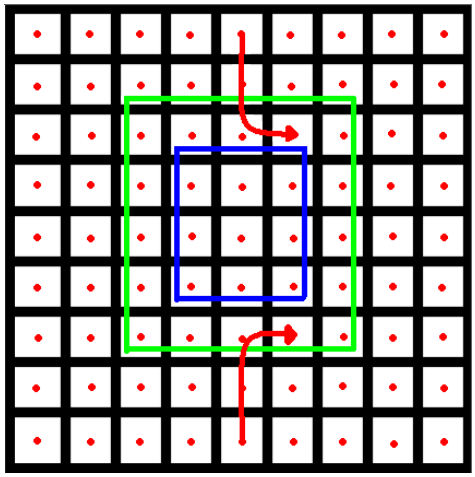

Figura 55 Margem de segurança criada em torno dos obstáculos

Finalmente, as coordenadas dos vértices do novo polígono (polígono original + área de segurança) são obtidas através das equações 11 e 12.

$$X_{2\text{-security}} = X_{2\text{-obs}} - d' * \cos(\theta + 90 + \alpha) \quad (11)$$

$$Y_{2\text{-security}} = Y_{2\text{-obs}} - d' * \text{sen}(\theta + 90 + \alpha) \quad (12)$$

5.2.4. IDENTIFICAÇÃO DE CÉLULAS LIVRES E OCUPADAS

Para determinar quais as células que estão ocupadas pelos obstáculos (obstáculo + margem de segurança) foi utilizada a função *inpolygon* que recebe como parâmetros os centros de todas as células e as coordenadas dos vértices dos obstáculos, e se os obstáculos passarem pelo centro das células, então as células são consideradas ocupadas, e é atribuído o valor 1 a essas células, as células que não possuem obstáculos a passar no seu centro são consideradas livres, e é atribuído o valor 0 a essas células.

5.2.5. ALGORITMO DE PESQUISA A*

A Figura 58 apresenta um fluxograma do processo de pesquisa da trajetória desenvolvido. O algoritmo implementado foi o A*, pronuncia-se *A star*. Como foi referido no capítulo anterior, este algoritmo foi implementado numa rede de células regulares. Para ser mais percetivel o conceito do algoritmo, este está sumariamente apresentado a seguir.

- São criadas duas matrizes, designadas por **lista aberta** e **lista fechada**. Os campos contidos em cada elemento das listas estão apresentados na Figura 56 na Figura 57. Na lista fechada são guardados os índices das células que possuem obstáculos, bem como os índices das células já avaliadas e com avaliação dos seus sucessores. Já na lista aberta são guardados os índices das células que vão sendo expandidas, como também os índices das células que as originaram. São também guardados os valores de h(n) (heurística – distância euclidiana entre a célula atual e a célula de destino), g(n) (custo da deslocação desde a célula inicial até a célula atual) e f(n) (custo de chegar desde a célula atual até à célula de destino). O campo ***Flag*** está a zero quando essa célula já foi avaliada e considerada para o trajeto.

- Inserir os índices das células ocupadas na lista fechada. Atribuir a posição inicial do veículo à célula atual. Inserir a célula atual na lista aberta, colocar a *flag* a zero e inserir os índices da célula na lista fechada também.

- Enquanto a célula atual for diferente da célula de destino e ainda existirem células para avaliação fazer:

    - Expandir todas as células contíguas à célula atual. As que não existem na lista aberta são inseridas, as que já existem na lista aberta são avaliadas e é verificado se o novo custo f(n) é inferior ao anteriormente calculado. Se assim for, é atualizado o novo custo f(n), assim como os índices da célula mãe e o custo g(n).

    - Depois de avaliadas todas as células contíguas sucessoras, é identificada a célula presente na lista aberta com o valor de f(n) mais baixo, o campo *flag* desta célula é colocado a zero e os índices da célula são inseridos na lista fechada

- Quando a célula atual é igual à célula de destino, os seus índices são inseridos na lista fechada. De seguida são guardados numa matriz os índices x e y da célula mãe que originou a célula atual, e assim sucessivamente até os índices da célula mãe serem idênticos à célula inicial, quando assim for, o trajeto está definido nesta nova matriz.

Flag	Índice X da célula atual	Índice Y da célula atual	Índice X da célula mãe	Índice Y da célula mãe	h(n)	g(n)	f(n)

Figura 56 Campos contidos em cada elemento da lista aberta

Índice X da célula atual	Índice Y da célula atual

Figura 57 Campos contidos em cada elemento da lista fechada

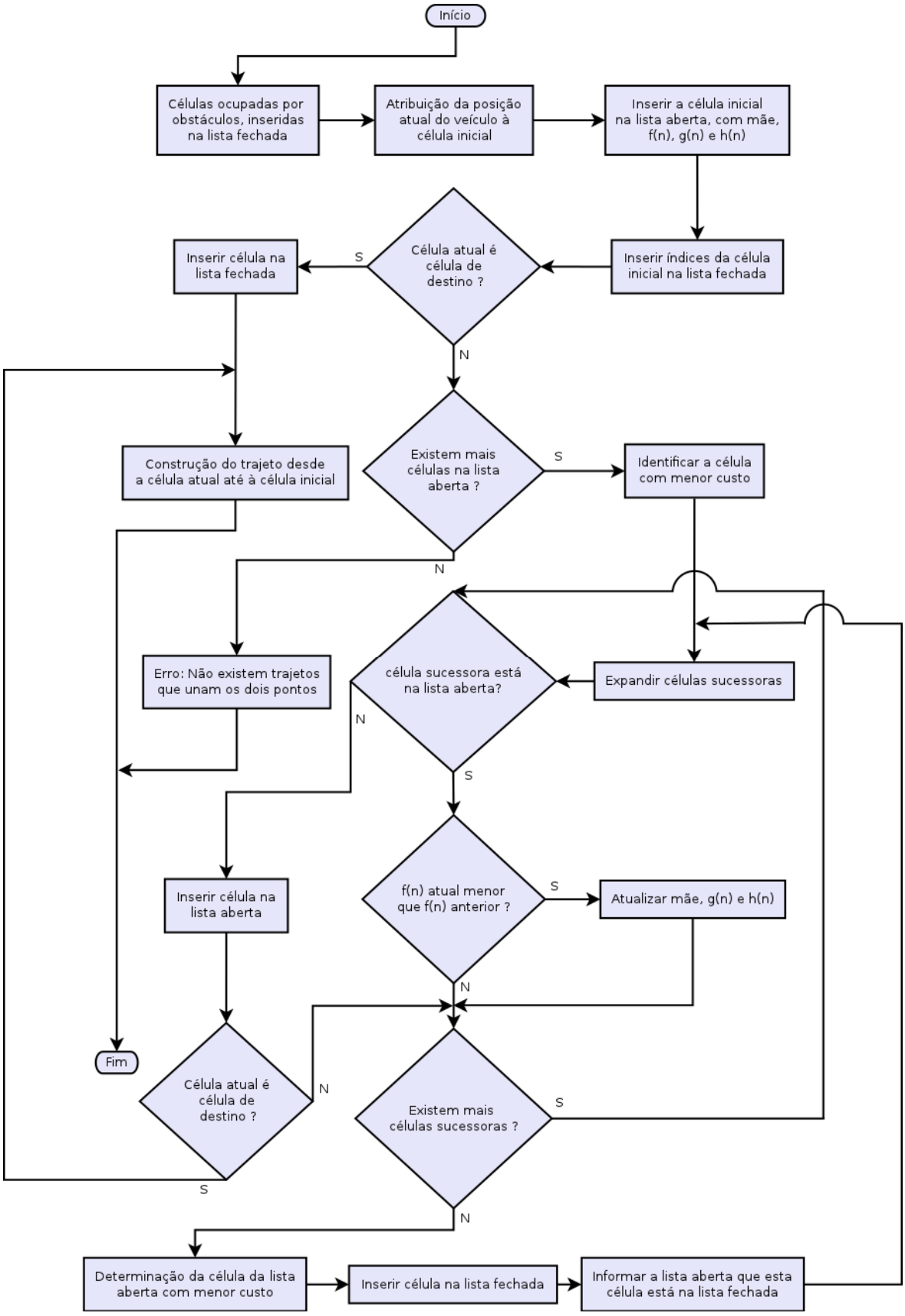

Figura 58 Fluxo de tarefas existentes no algoritmo de pesquisa A*

5.2.6. DETERMINAÇÃO DOS WAYPOINTS NECESSÁRIOS AO TRAJETO

O algoritmo de pesquisa A* guarda numa matriz as posições de todas as células identificadas para o trajeto pretendido. É de notar que a maioria delas estão na mesma linha, quer na vertical, horizontal ou diagonal. Como não é praticável enviar um *waypoint* para a aeronave por cada célula identificada, foi utilizada a função *collinear* para eliminar todos os *waypoints* que se encontrem entre os dois *waypoints* extremos de cada linha. Como mostra a Figura 59, só os *waypoints* identificados pelos pontos verdes é que são escritos no ficheiro e lidos pelo Neptus.

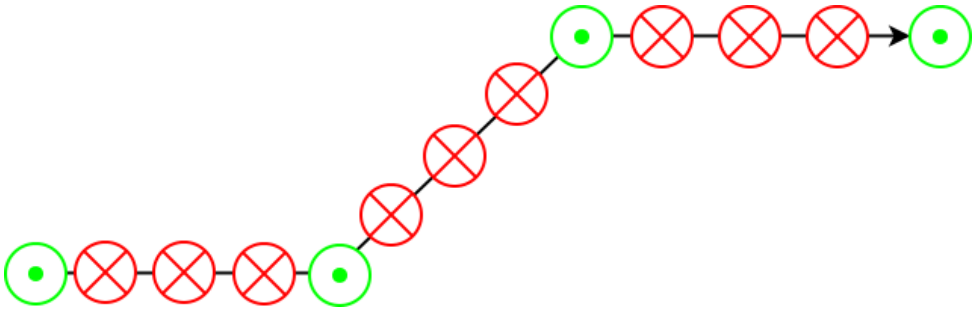

Figura 59 *Waypoints* descartados e *waypoints* enviados para a plataforma

6. RESULTADOS

Depois de descritos os métodos utilizados na implementação, é possível apresentar alguns resultados e condições de teste. Os testes no terreno foram realizados no Aeroclube da Costa Verde em Espinho e na base da FAP, na Ota.

6.1. TESTES EM SIMULAÇÃO

Os testes em simulação realizados pretendem avaliar o algoritmo de PT em parâmetros distintos, nomeadamente nos tempos de computação do algoritmo e nas distâncias dos trajetos obtidos, para aeronaves distintas. Para os testes em simulação é necessário um computador com o Neptus e o MATLAB instalado. O fluxo de informação trocada entre os dois sistemas está apresentado na Figura 60.

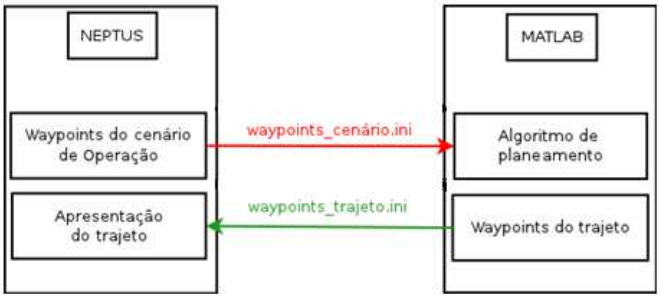

Figura 60 Fluxo de informação trocada entre o Neptus e o MATLAB

6.1.1. TESTES EFETUADOS

Foram realizadas diversas simulações para avaliação do algoritmo de PT desenvolvido, com plataformas e cenários distintos. Foram utilizadas as plataformas alfa, pilatus e cularis, e foram simulados cenários de operação com diferentes números de obstáculos. Na Tabela 8 estão apresentados os valores referentes ao raio mínimo de curvatura à velocidade de cruzeiro, tempo de computação do algoritmo, número de células da grelha de pesquisa e a distância do trajeto calculado, para as diversas plataformas e para o cenário de operação apresentado na Figura 61.

Tabela 8 Valores dos parâmetros avaliados no algoritmo, para os cenários da Figura 61

Plataforma	Raio mínimo De curvatura (m)	Nº de células	Tempo de computação (s)	Distância do Trajeto (m)
Pilatus 03	125	494	0,1860	4048
Pilatus 04	88	1036	0,2599	3793
Alfa 02	84	1131	0,2818	3923
Alfa 03	74	1485	0,3388	3861
Alfa 05	82	1200	0,3377	3849
Alfa 06	68	1764	0,4213	3876
Cularis 05	14	41888	50,6186	3818

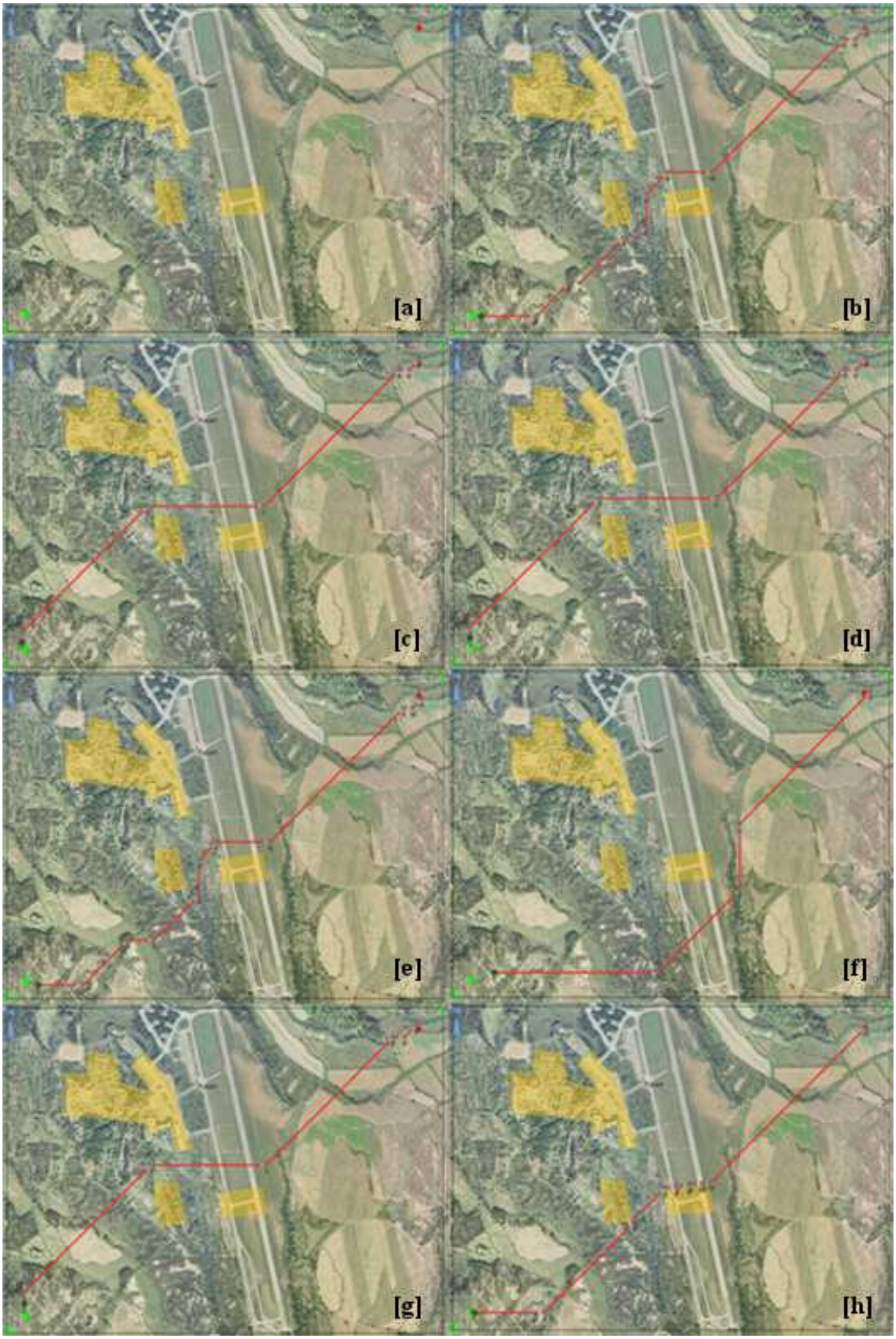

Figura 61 Cenário de operação com poucos obstáculos (a) e trajetos obtidos após a execução do algoritmo de planeamento de trajetórias desenvolvido, para as plataformas: (b) – Alfa02, (c) – Alfa03, (d) Alfa05, (e) – Alfa06, (f) – Pilatus03, (g) – Pilatus04, (h) – Cularis05

É de notar que os trajetos são distintos devido à diferente decomposição do espaço de operação em células. A dimensão das células é diferente mediante a plataforma que está a ser considerada. A linha vermelha indica o trajeto planeado, os polígonos a amarelo estão a simular os obstáculos e o ponto verde e vermelho são o local de partida e de destino respetivamente.

Através da análise dos dados da Tabela 8 é possível verificar que o tempo de computação do algoritmo é menor quanto maior for a dimensão das células, mas também é de salientar que neste caso a resolução da grelha de pesquisa é mais baixa. Para aumentar essa resolução é necessário diminuir a dimensão das células, o que obriga a um aumento do tempo de computação do algoritmo, o que em alguns casos é bastante significativo, como é o caso do cularis 05. A distância do trajeto obtido é menor quanto maior for a resolução da grelha de pesquisa.

Na Tabela 9 estão apresentados os valores referentes ao raio mínimo de curvatura à velocidade de cruzeiro, tempo de computação do algoritmo, número de células da grelha de pesquisa e a distância do trajeto calculado, para as diversas plataformas e para o cenário de operação apresentado na Figura 63. Nesta simulação, existem mais obstáculos que na simulação anterior. Contudo, apesar do maior número de obstáculos e das distâncias dos trajetos obtidos serem superiores, observa-se que os tempos de computação do algoritmo são ligeiramente inferiores aos da simulação anterior.

Isto deve-se principalmente ao facto de existirem mais obstáculos. Se analisarmos a Figura 62 reparamos que após a criação da margem de segurança em torno dos obstáculos, passam a existir poucas células livres entre os obstáculos. Então o algoritmo não irá possuir muitas células para expandir, o que se traduz numa diminuição do custo computacional e consequentemente numa diminuição do tempo de computação.

São apresentados na Figura 62, Figura 64 e Figura 65 os *plots* executados pelo MATLAB durante a execução do algoritmo de planeamento para o alfa02, alfa06 e cularis05, onde a linha vermelha representa os limites dos obstáculos existentes, a linha azul representa a margem de segurança em torno dos obstáculos (está definida para possuir o valor do raio mínimo de curvatura à velocidade de cruzeiro), as células amarelas representam células ocupadas pelos obstáculos mais a margem de segurança (são inseridas na lista fechada no inicio do algoritmo), as células verdes são células que estão livres, representando posições possíveis para o trajeto, as células preta e branca representam a posição inicial e de destino

respetivamente. Na Figura 65 é possível perceber o aumento da resolução da grelha à medida que se reduz a dimensão das células.

Tabela 9 Valores dos parâmetros avaliados no algoritmo, para os cenários da Figura 63

Plataforma	Raio mínimo De curvatura (m)	Nº de células	Tempo de computação (s)	Distância do Trajeto (m)
Pilatus 03	125	494	0,1757	4853
Pilatus 04	88	1036	0,2346	4794
Alfa 02	84	1131	0,2512	4533
Alfa 03	74	1485	0,3138	4529
Alfa 05	82	1200	0,2566	4725
Alfa 06	68	1764	0,3158	4024
Cularis 05	14	41888	37,1119	3921

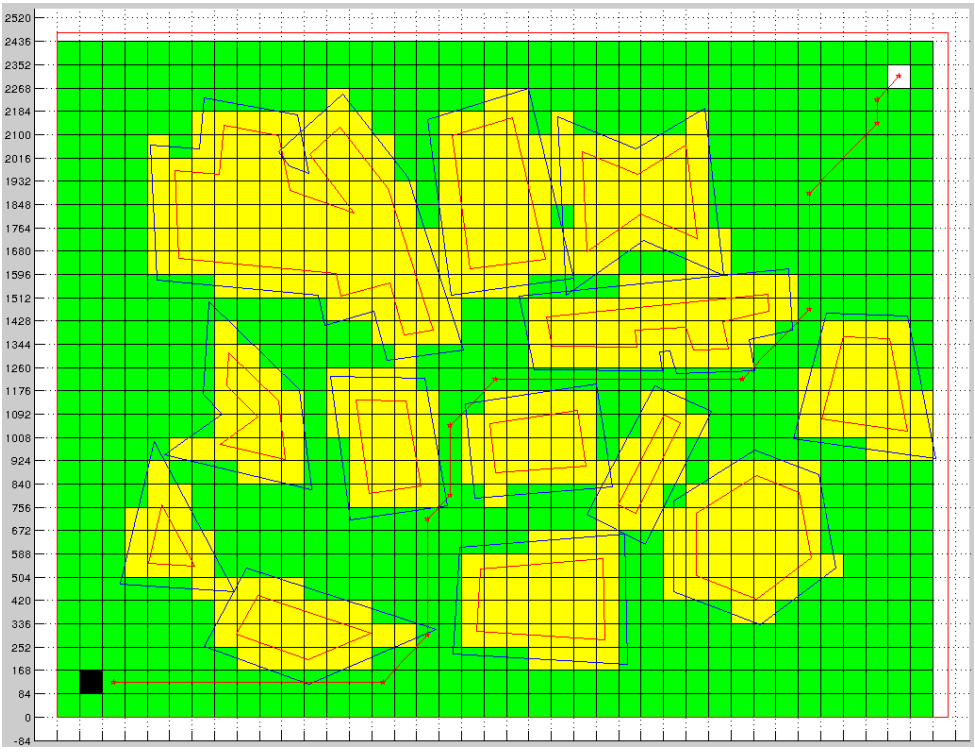

Figura 62 Apresentação dos obstáculos, margens de segurança e trajeto obtido para o Alfa02

Figura 63 Cenário de operação com muitos obstáculos (a) e trajetos obtidos após a execução do algoritmo de planeamento de trajetórias desenvolvido, para as plataformas: (b) – Alfa02, (c) – Alfa03, (d) Alfa05, (e) – Alfa06, (f) – Pilatus03, (g) – Pilatus04, (h) – Cularis05

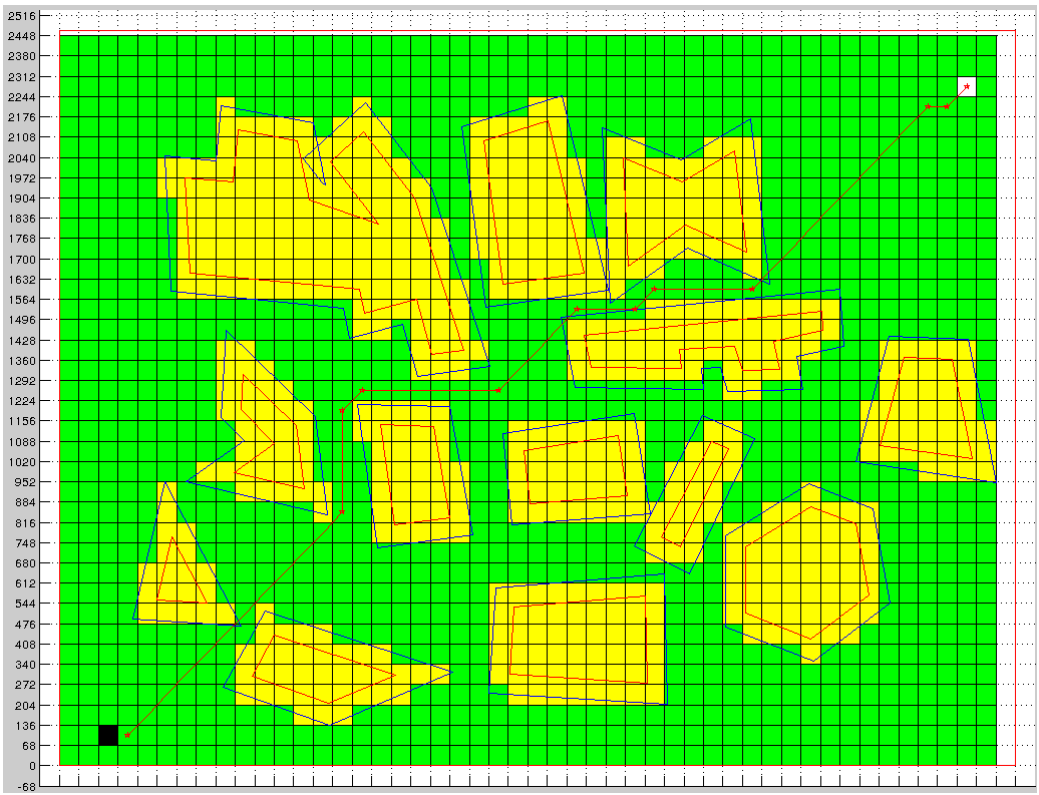

Figura 64 Apresentação dos obstáculos, margens de segurança e trajeto obtido para o Alfa06

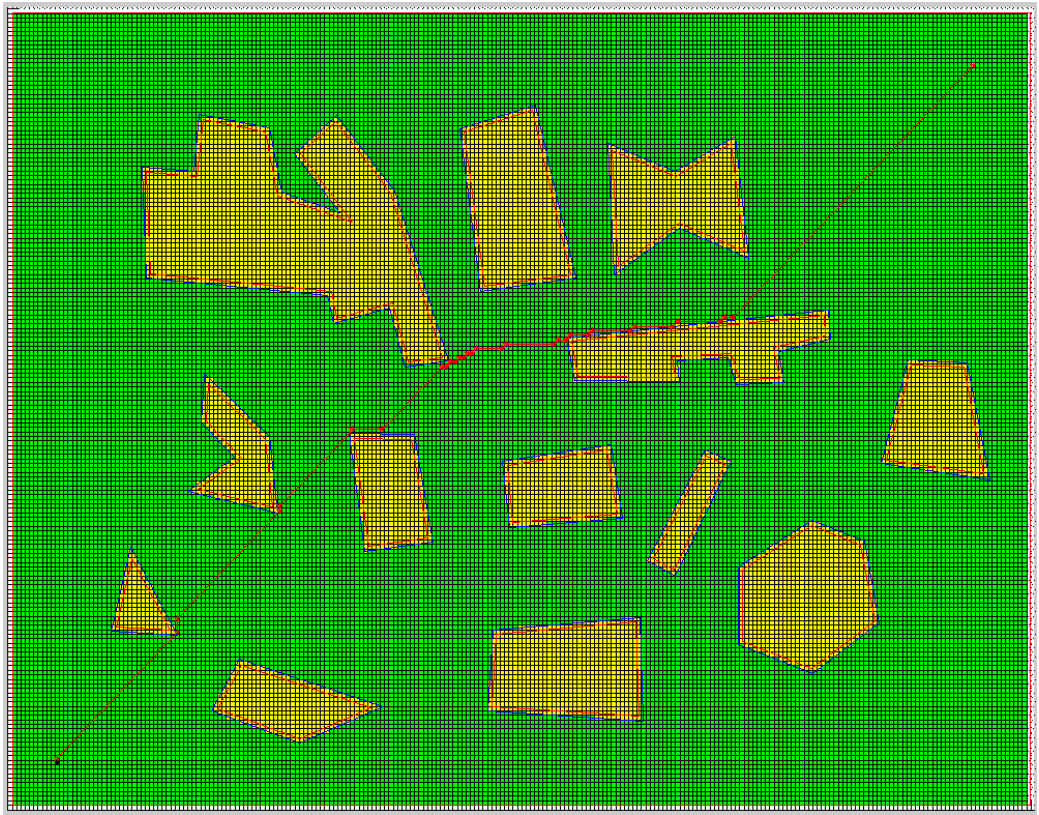

Figura 65 Apresentação dos obstáculos, margens de segurança e trajeto obtido para o Cularis05

O algoritmo desenvolvido foi também testado com AUVs. Na Figura 66 e Figura 67 estão apresentados o cenário de operação e o trajeto calculado. É importante salientar que neste momento o algoritmo não toma em consideração as correntes marítimas no cálculo do trajeto.

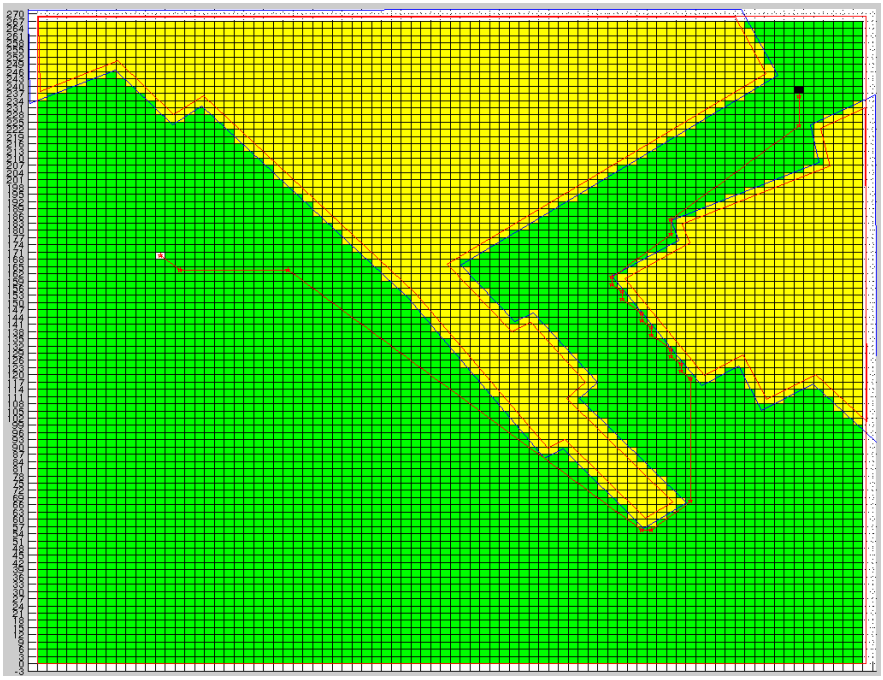

Figura 66 Apresentação dos obstáculos, margens de segurança e trajeto obtido para o AUV seacon2

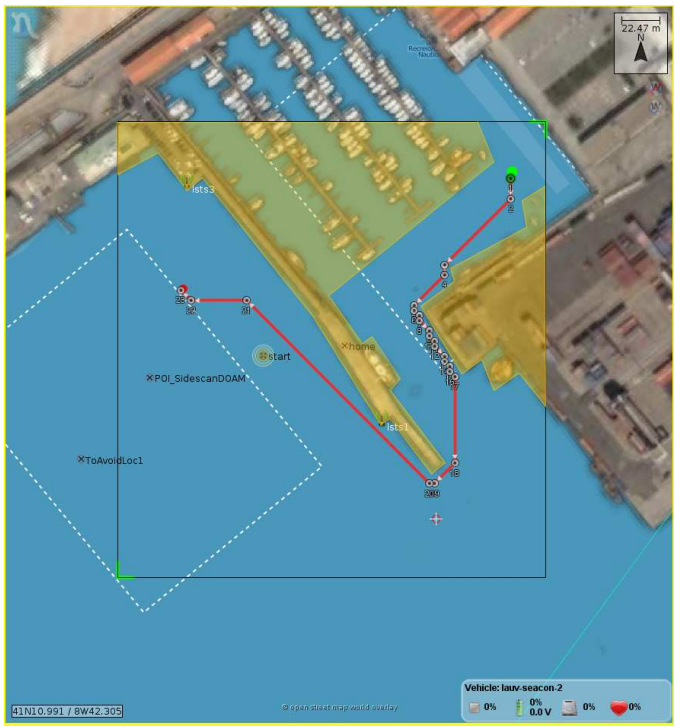

Figura 67 Cenário de operação e trajeto obtido para o Seacon 2

6.1.2. COMPARAÇÃO COM ALGORITMO *FAST MARCHING*

O algoritmo desenvolvido foi comparado com um algoritmo existente do tipo *fast marching*. Para que os dados obtidos através dos testes tenham relevância, é necessário que os cenários de operação e os procedimentos de teste sejam o mais parecidos possível.

Para efeitos de comparação, a margem de segurança em torno dos obstáculos é irrelevante. Por uma questão de simplicidade e para que as simulações sejam executadas nas mesmas condições, foi usada uma margem de segurança nula para ambos os algoritmos.

Como o algoritmo de PT desenvolvido utiliza oito direções de pesquisa, o algoritmo *fast marching* foi simulado também com oito direções de pesquisa.

Ambos os algoritmos foram simulados para o mesmo cenário de operação, que foi a pista de aviação da base aérea da FAP na Ota. Foram simulados com o mesmo número de obstáculos, com os mesmos limites espaciais e com as mesmas posições de partida e destino. A área de operação possui 3405 metros de comprimento por 2543 metros de largura.

As simulações foram executadas para diferentes números de células existentes na grelha de pesquisa (ver Tabela 10). O número de células está diretamente relacionado com a dimensão de cada célula (i.e., o comprimento da sua aresta), logo, quanto maior for a dimensão de cada célula menor será o número de células existentes na grelha de pesquisa.

Cada valor para o tempo de computação apresentado na Tabela 10 e na Tabela 11 foi obtido calculando uma média de dez medições para compensar os efeitos da troca de contexto do processador (*polling*).

Os algoritmos foram simulados em ambientes com diferentes números de obstáculos. Na Figura 68 está apresentado o cenário de operação com um obstáculo, que foi usado para os testes apresentados na Tabela 10. Estão também apresentados os trajetos obtidos utilizando os dois algoritmos de PT, para uma grelha de pesquisa de 38363 células.

Na Tabela 10 são apresentados os tempos de computação e os comprimentos dos trajetos calculados utilizando os algoritmos de PT A* e *fast marching*, em função do número de células existentes na grelha de pesquisa.

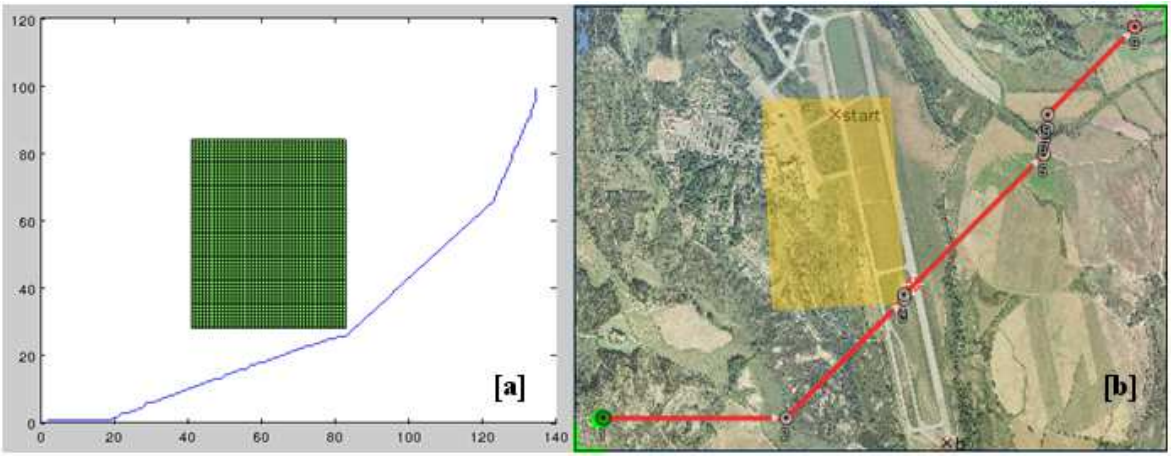

Figura 68 (a) Trajeto planeado com algoritmo *fast marching*, com poucos obstáculos, (b) Trajeto planeado com algoritmo A*, com poucos obstáculos

Tabela 10 Resultados das simulações dos algoritmos *fast marching* e A* com poucos obstáculos, para diferentes números de células

Dimensão de cada célula (m)	Nº de células	Tempo de computação A* para 8 direções de pesquisa (s)	Tempo de computação *Fast Marching* para 8 direções de pesquisa (s)	Comprimento do Trajeto A* para 8 direções de pesquisa (m)	Comprimento do Trajeto *Fast Marching* para 8 direções de pesquisa(m)	Comprimento do Trajeto *Fast Marching* para 360 direções de pesquisa(m)
125	540	0,2035	0,2188	3953	4250	4000
115	638	0,2084	0,2128	3915	4255	3910
105	768	0,2147	0,2208	3976	4200	3885
95	910	0,2291	0,2176	4056	4085	3895
85	1160	0,2430	0,2200	4039	4165	3910
75	1485	0,3032	0,2240	4107	4275	4050
65	2028	0,3832	0,2392	4084	4420	4160
55	2806	0,4500	0,2484	4065	4565	4290

45	4200	0,8077	0,2724	4078	4455	4185
35	6984	1,5374	0,3344	4066	4445	4200
25	13736	4,0636	0,4344	4103	4650	4400
15	38363	24,23	0,7757	4120	4740	4455
5	345948	1489	4,9676	4129	4740	4445

Na Figura 69 está apresentado um gráfico de linhas, onde se consegue interpretar melhor alguns resultados presentes na Tabela 10.

As últimas três linhas da Tabela 10 não foram inseridas no gráfico da Figura 69 porque como são valores bastante superiores aos restantes, tornam as escalas demasiadamente grandes para os restantes valores, dificultando a sua análise.

Através da análise do gráfico da Figura 69, observa-se o seguinte: primeiro, o algoritmo fast marching possui tempos de computação quase sempre inferiores aos do A*; segundo, à medida que o número de células vai aumentando, o custo computacional do algoritmo A* cresce a uma taxa superior à do fast marching.

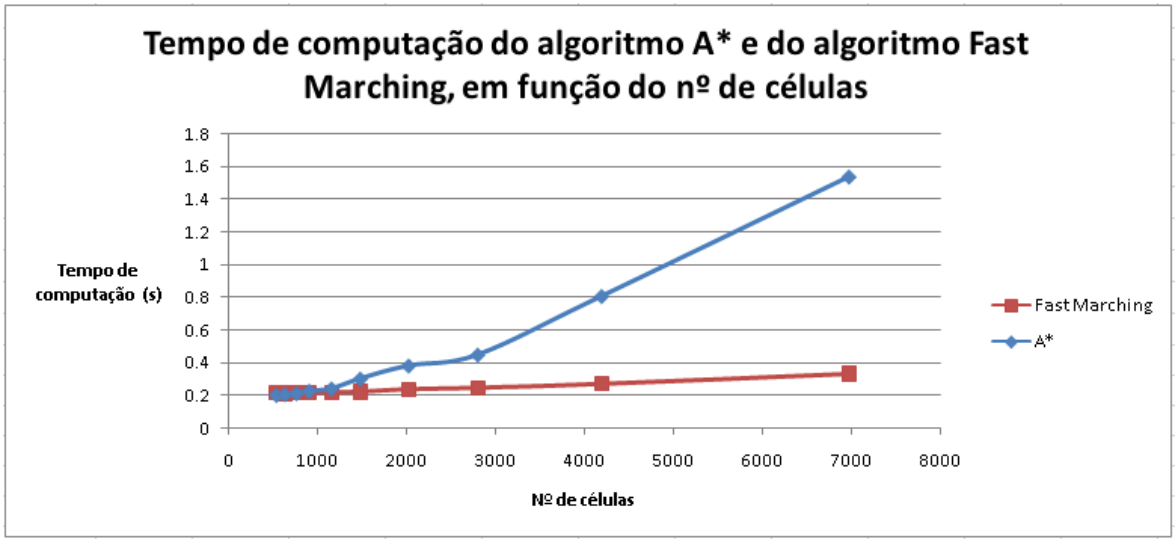

Figura 69 Tempo de computação do algoritmo A* e do algoritmo *fast marching* em função do número de células, com poucos obstáculos

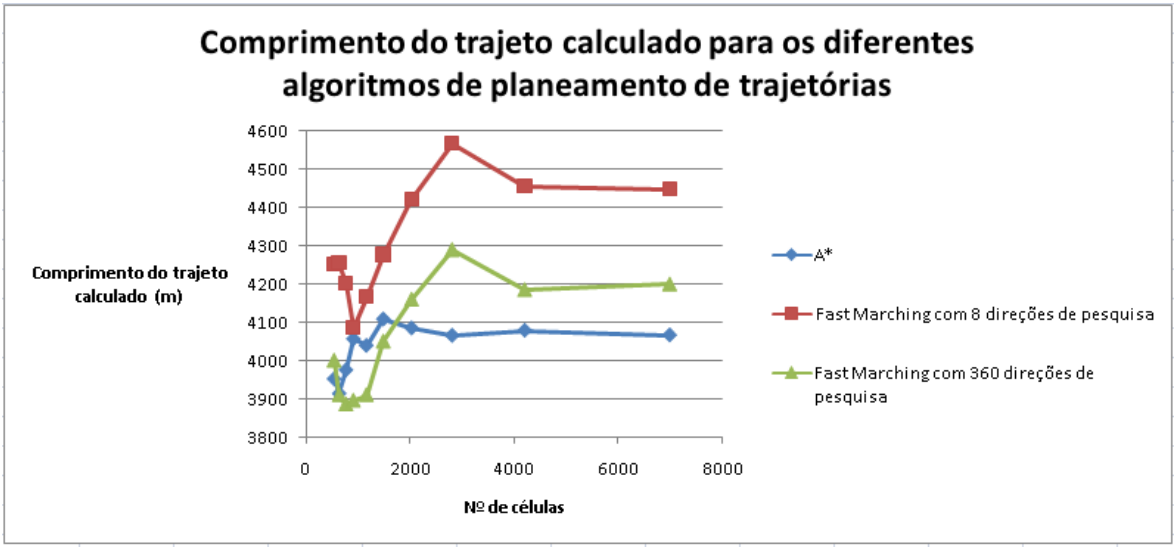

Figura 70 Comprimento do trajeto calculado para os diferentes algoritmos em função do número de células, com poucos obstáculos

Na Figura 70 está representado um gráfico que apresenta o comprimento do trajeto calculado em função do número de células, para o algoritmo A*, e para o algoritmo *fast marching* com 8 e 360 direções de pesquisa.

Através da análise deste gráfico conclui-se que o algoritmo A* possui o comprimento do trajeto sempre inferior ao do *fast marching* com 8 direções. E que o algoritmo *fast marching* com 360 direções consegue diminuir o comprimento do trajeto na ordem dos 250 metros, em relação ao mesmo algoritmo com 8 direções de pesquisa.

O crescimento do custo computacional do algoritmo A* a uma taxa superior que o do algoritmo *fast marching* não é algo que possa ser considerado normal. Este crescimento deve-se aos métodos de implementação utilizados, ao facto da lista aberta não permanecer ordenada e ao facto de não controlarmos a forma como o MATLAB gere dinamicamente os vetores/matrizes. Contudo na maior parte dos casos, os tempos obtidos são aceitáveis.

Na Figura 71 está apresentado o cenário de operação com vários obstáculos, que foi usado para os testes apresentados na Tabela 11. Estão também apresentados os trajetos obtidos utilizando os dois algoritmos de PT, para uma grelha de pesquisa de 38363 células.

Na Tabela 11 são apresentados os tempos de computação e os comprimentos dos trajetos calculados utilizando os algoritmos de PT A* e *fast marching*, em função do número de células existentes na grelha de pesquisa.

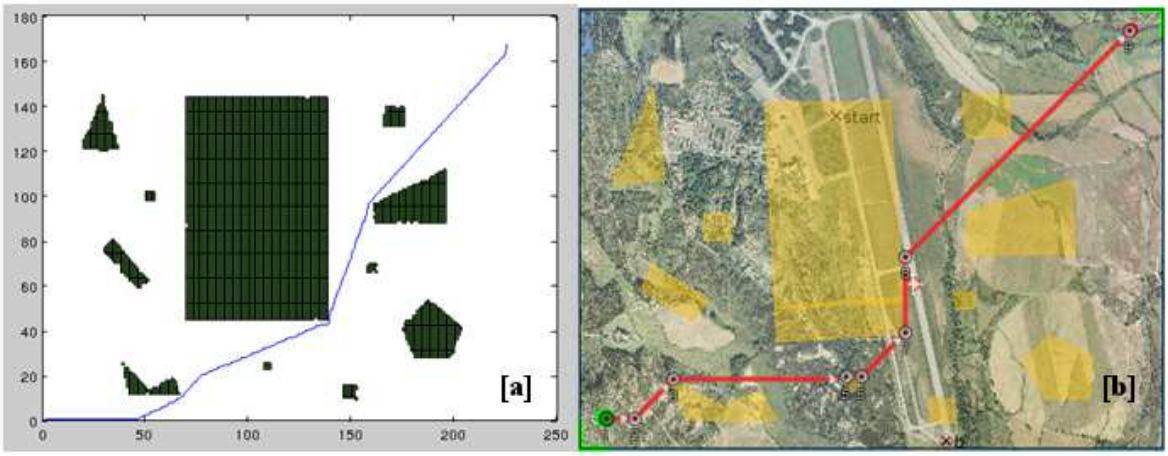

Figura 71 (a) Trajeto planeado com algoritmo *fast marching*, com muitos obstáculos, (b) Trajeto planeado com algoritmo A*, com muitos obstáculos

Tabela 11 Resultados das simulações dos algoritmos *fast marching* e A* com muitos obstáculos, para diferentes números de células

Dimensão de cada célula (m)	Nº de células	Tempo de computação A* para 8 direções de pesquisa (s)	Tempo de computação *Fast Marching* para 8 direções de pesquisa (s)	Comprimento do Trajeto A* para 8 direções de pesquisa (m)	Comprimento do Trajeto *Fast Marching* para 8 direções de pesquisa(m)	Comprimento do Trajeto *Fast Marching* para 360 direções de pesquisa(m)
125	540	0,2166	0,2168	4099	4250	4125
115	638	0,2208	0,2152	3982	4140	3910
105	768	0,2286	0,2168	4099	4410	4200
95	910	0,2351	0,2205	4167	4275	3990
85	1160	0,2489	0,2176	4139	4505	4080
75	1485	0,2789	0,2184	4195	4500	4200
65	2028	0,3572	0,2212	4198	4295	4020

55	2806	0,3903	0,2532	4161	4565	4290
45	4200	0,5937	0,2612	4184	4590	4365
35	6984	1,2251	0,3460	4169	4620	4410
25	13736	3,2935	0,4336	4235	4675	4450
15	38363	20,19	0,8105	4225	4725	4485
5	345948	1436	5,1572	4240	4740	4520

Na Figura 72 está apresentado um gráfico de linhas, onde se consegue interpretar melhor alguns resultados presentes na Tabela 11.

Através da análise deste gráfico tiram-se conclusões semelhantes às que foram tiradas na análise do gráfico da Figura 69. Apesar do aumento do número de obstáculos no cenário de operação, os tempos de computação mantiveram-se equivalentes.

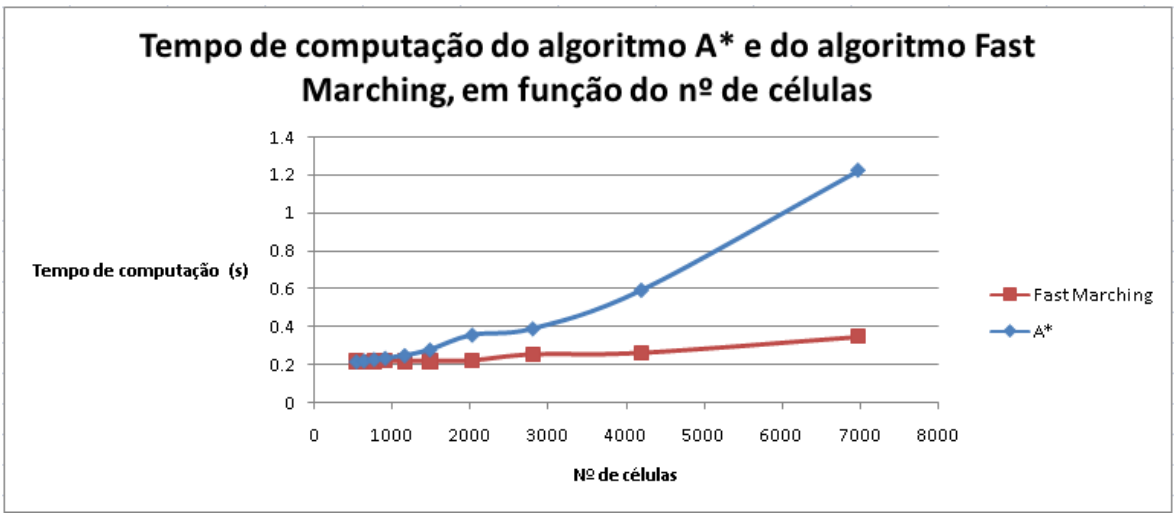

Figura 72 Tempo de computação do algoritmo A* e do algoritmo fast marching em função do número de células, com vários obstáculos

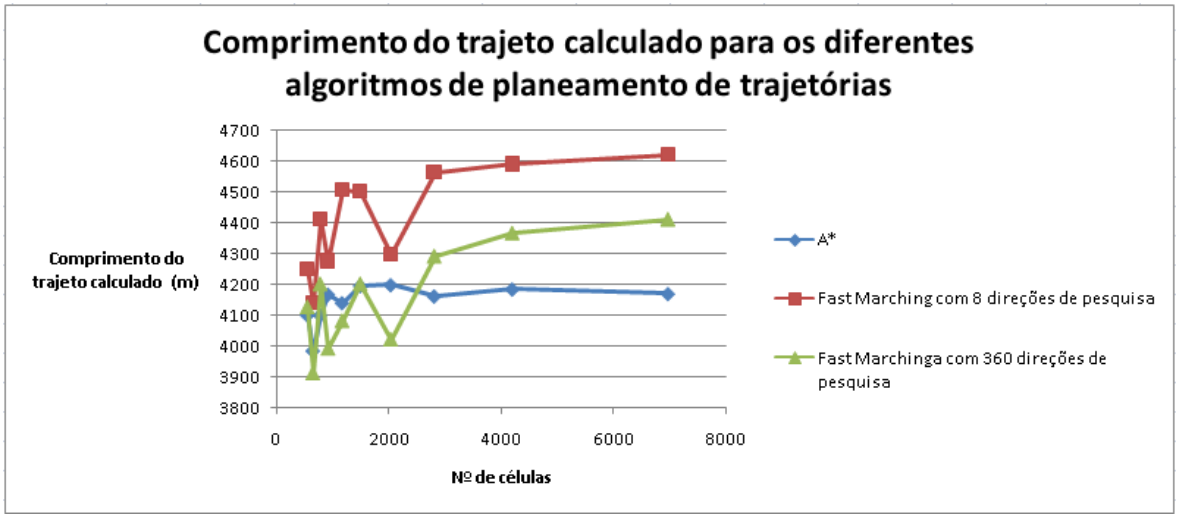

Figura 73 Comprimento do trajeto calculado para os diferentes algoritmos em função do número de células, com vários obstáculos

Na Figura 73 está representado um gráfico que apresenta o comprimento do trajeto calculado em função do número de células, para o algoritmo A*, e para o algoritmo *fast marching* com 8 e 360 direções de pesquisa.

Através da análise deste gráfico retiram-se conclusões semelhantes às da Figura 70. O algoritmo A* possui o comprimento do trajeto sempre inferior ao do *fast marching* com 8 direções. O algoritmo *fast marching* com 360 direções consegue diminuir o comprimento do trajeto na ordem dos 200 metros, em relação ao mesmo algoritmo com 8 direções de pesquisa.

Na Tabela 10 e na Tabela 11 existe uma coluna dedicada ao comprimento do trajeto calculado com o algoritmo *fast marching*, com 360 direções de pesquisa. Esta coluna mostra que o comprimento do trajeto diminui à medida que se aumentam o número de direções de pesquisa, que no limite traduz-se no caminho ótimo.

Na Figura 74 estão apresentadas duas imagens do mesmo cenário. Em **(a)** o trajeto é obtido com 8 direções de pesquisa, enquanto em **(b)** o trajeto é obtido com 360 direções de pesquisa. É de notar a diferença existente dentro dos círculos vermelhos das duas imagens. Com 360 direções de pesquisa o trajeto é criado logo na direção do vértice do obstáculo (b), o que não acontece com 8 direções de pesquisa (a).

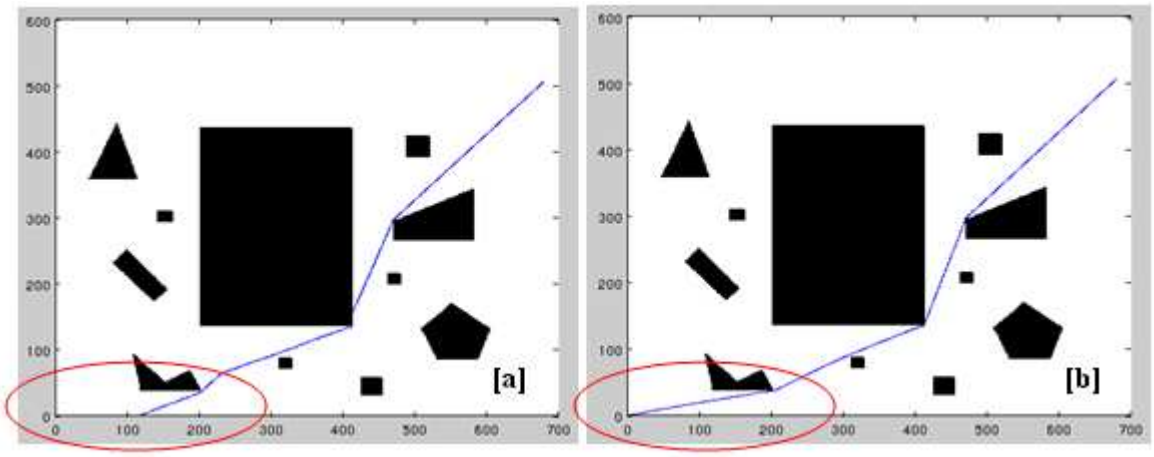

Figura 74 (a) Trajeto obtido com o algoritmo *fast marching* com 8 direções de pesquisa, (b) Trajeto obtido com o mesmo algoritmo, com 360 direções de pesquisa

6.2. TESTES NO TERRENO

Os testes no terreno pretendem avaliar se as aeronaves conseguem executar o trajeto obtido através do algoritmo de planeamento desenvolvido. Na Figura 75 está apresentada a arquitetura do sistema de testes no terreno.

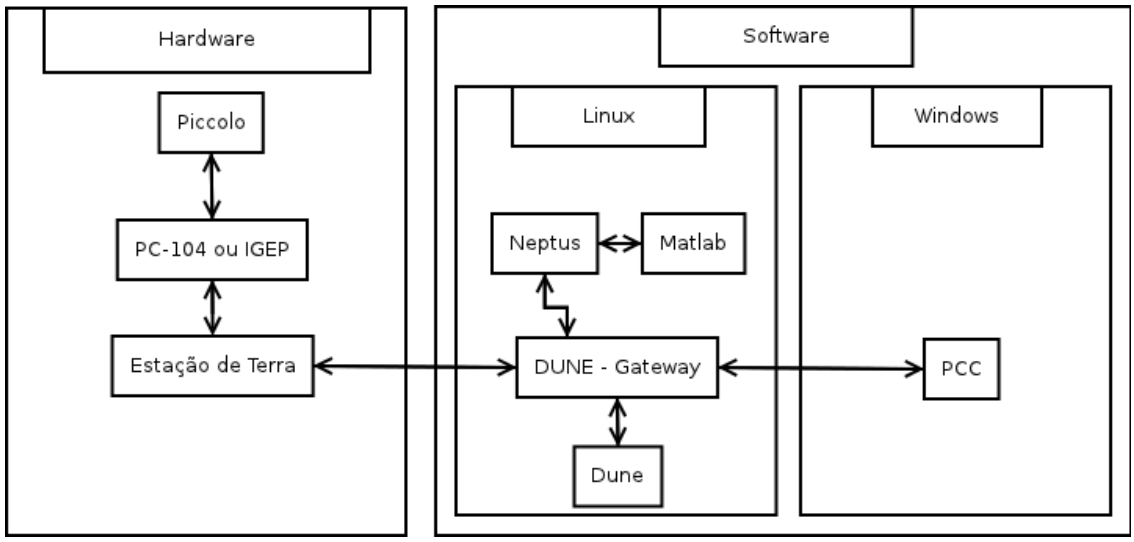

Figura 75 Arquitetura do sistema de testes no terreno

6.2.1. TESTES EFETUADOS

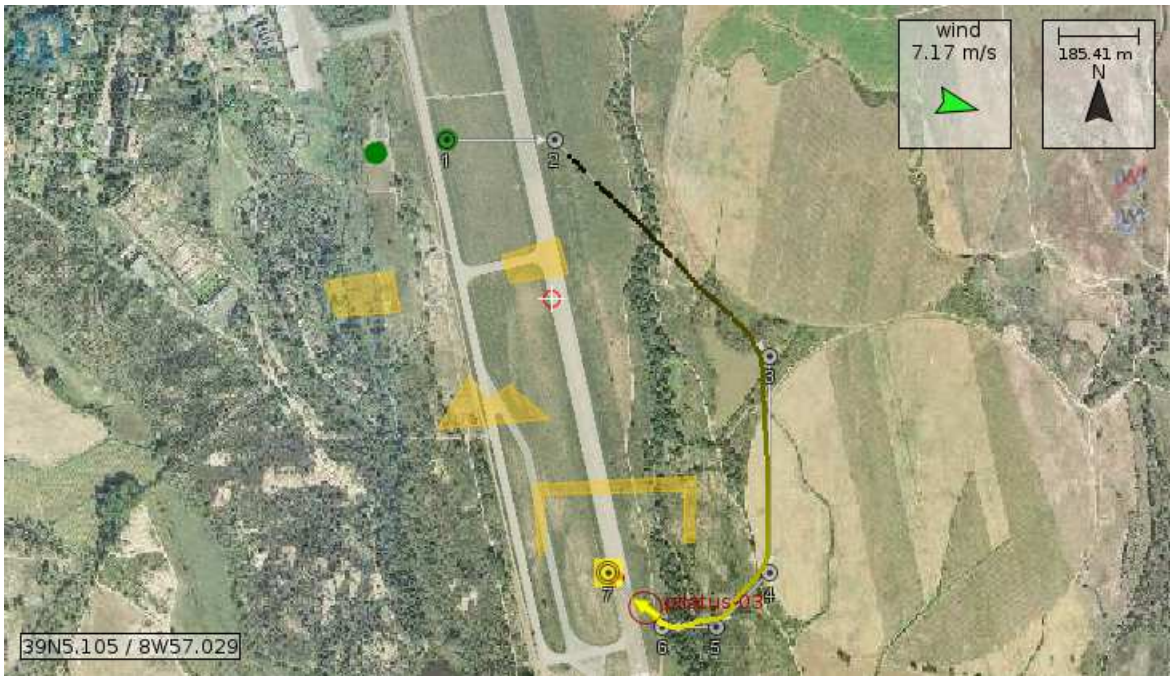

Figura 76 Execução do trajeto planeado

Foram executados diversos testes com diferentes plataformas, e em todos os testes as plataformas conseguiram seguir o trajeto planeado, como é possível observar na Figura 76. As aeronaves conseguem executar os trajetos planeados, sem colidirem com os obstáculos. Relembra-se que o algoritmo não prevê velocidades do vento nos seus cálculos. Estes resultados indicam que a estratégia de colocar a margem de segurança em torno dos obstáculos, do tamanho do raio mínimo de curvatura de cada aeronave é adequada pois os trajetos obtidos são trajetos possíveis de serem seguidos.

7. CONCLUSÕES E TRABALHO FUTURO

Neste capítulo final serão apresentados alguns pontos fundamentais do desenvolvimento deste projeto, assim como alguns tópicos para desenvolvimento futuro.

7.1. RESUMO DO TRABALHO DESENVOLVIDO

O objetivo fundamental da realização deste projeto é o desenvolvimento de módulos de software que permitam planear o trajeto para a deslocação de um UAV desde um ponto A até um ponto B em segurança, evitando obstáculos que intersetem o trajeto. Para concretizar este objetivo foram realizados os seguintes passos:

- Foram estudadas as tecnologias existentes nos UAVs, para perceber as capacidades e limitações destes.
- Como um dos objetivos é a integração da solução encontrada nos sistemas desenvolvidos no LSTS, foi necessário estudar a *toolchain* de *software* existente.
- Foi feita a revisão do estado da arte, para permitir identificar as soluções existentes nesta área, e identificar um ponto de partida para o desenvolvimento da solução.

- Foram estudados algoritmos para definição de caminhos em espaço discreto e em espaço contínuo.

- Foram também estudados algoritmos de pesquisa de trajetórias entre dois pontos no espaço.

- Foi implementado um *plugin* no Neptus para permitir definir o cenário de operações

- Foi implementado em MATLAB o algoritmo A* para pesquisar o trajeto mais curto entre dois pontos numa grelha de células regulares.

- Foram executadas diversas simulações, e posteriores testes no terreno para avaliação do algoritmo desenvolvido, e para recolha de dados essenciais a futuros desenvolvimentos.

7.2. SATISFAÇÃO DOS OBJETIVOS

O principal objetivo proposto foi atingido com sucesso. Foi desenvolvido um sistema que permite calcular *offline* o trajeto mais curto entre dois pontos em espaço discreto, onde a dimensão de cada célula é configurável. Com a introdução do algoritmo desenvolvido no decorrer das missões de UAVs, é aliviada a carga de trabalho do piloto.

Em relação aos algoritmos de pesquisa conclui-se que o método A* torna-se mais lento à medida que o número de células aumenta, mas que por vezes o tempo de computação é inferior quando existem mais obstáculos devido ao número reduzido de células livres. Já o algoritmo do tipo *fast marching* possui um tempo de computação bastante baixo, para o número de células que avalia, no entanto este algoritmo não assegura que o UAV consiga voar o trajeto encontrado.

Apesar de não ser um objetivo inicial, a preocupação de que o algoritmo também funcionasse com os AUVs foi tida em conta e, tal como discutido no capítulo anterior, este objetivo também foi cumprido.

7.3. TRABALHO FUTURO

Apesar de os objetivos inicialmente propostos terem sido alcançados, foram sendo detetadas, durante a implementação, melhorias a adicionar aos desenvolvimentos, que podem ser atribuídas a futuros desenvolvimentos, entre elas estão:

- Migrar todo o código de MATLAB para o Neptus (java) para um planeamento offline, e também para o Dune (C++) para um planeamento online, tornando desta forma o algoritmo bastante mais rápido, sem ser necessário recorrer ao MATLAB.
- Ao utilizar o Neptus, deverá ser possível guardar os obstáculos.
- Quando o piloto acaba de desenhar cada obstáculo no neptus, o neptus deve criar e apresentar a margem de segurança, para que o operador a possa identificar.
- Definir a margem de segurança em torno dos obstáculos tendo em conta a informação do vento.
- Implementar uma função que determine a densidade do ar em função da altitude e da temperatura.
- Implementar o algoritmo de pesquisa D* no Dune para que, com a ajuda de sensores que detetem obstáculos, seja possível evitar os mesmos, ainda que estes estejam em movimento.
- Implementar o método das *quadtrees* e das *frammed quadtrees* na decomposição da área de pesquisa em células. Esta implementação servirá para verificar se o trajeto obtido é mais curto, e se for, se esse comprimento justifica o aumento do custo computacional.
- Implementar o método potential field e RRT para poder analisar as diferenças existentes em termos de tempo e custo computacional.
- Alterar o algoritmo A* desenvolvido para suportar planeamento de trajetórias em três dimensões.

Referências Documentais

[1] J. Sullivan, “Revolution or Evolution? The Rise of the UAVs”, Sibley School of Mechanical and Aerospace Engineering Cornell University Ithaca, NY 14853

[2] R. Beard, T. Mclain, D. Nelson, D. Kingston, D. Johanson, “Decentralized Cooperative Aerial Surveillance Using Fixed-Wing Miniature UAVs”, Vol. 94, No. 7, July 2006, Proceedings of the IEEE

[3] S. Lentilhac, “UAV Flight Plan Optimized for Sensor Requirements” Thales Airborn System, Pessas, France, Based on a presentation at Radar 2009 Bordeaux, 2010 IEEE

[4] D. Kingston, R. Beard, R. Holt, “Decentralized Perimeter Surveillance Using a Team of UAVs”, IEEE Transactions on Robotics, Vol. 24, NO. 6, December 2008

[5] D. Casbeer, S. Li, R. Beard, R. Mehra, T. Mclain, “Forest fire monitoring using multiple small UAVs”, 2005 American Control Conference, June 8-10, 2005. Portland, OR, USA

[6] K. Tahar, A. Ahmad, W. Akib, “UAV-Based Stereo Vision for Photogrammetric Survey in Aerial Terrain Mapping” 2011 International Conference on Computer Applications and Industrial Electronics (ICCAIE 2011)

[7] M. Bryson, S. Sukkarieh, “Active Airborne Localization and Exploration in Unknown Environments using Inertial SLAM”, 2006 IEEE AC paper #1079, Version 3, Updated Oct, 27 2005

[8] A. Girard, A. Howell, J. Hedrick, “Border Patrol and Surveillance Missions using Multiple Unmanned Air Vehicles”, Decision and Control, CDC. 43rd IEEE Conference on 17 Dec. 2004

[9] F. Heintz, P. Rudol, P. Doherty, “From Images to Traffic Behavior – A UAV Tracking and Monitoring Application”, Information Fusion, 2007 10th International Conference on 9-12 July 2007

[10] A. Puri, K. Valavanis, M.Kontitsis, “Statistical Profile Generation for Traffic Monitoring Using Real-time UAV based Video Data”, Mediterranean Conference on Control and Automation, 27-29 July 2007, Athens – Greece

[11] B. Wang, X. Chen, Q. Wang, L. Liu, H. Zhang, B. Li, “Power Line Inspection with A Flying Robot”, 2010 1st International Conference on Applied Robotics for the Power Industry Delta Centre-Ville, Montréal, Canada, October 5-7, 2010

[12] Y. Naidoo, R. Stopforth, G. Bright, “Development of an UAV for Search & Rescue Applications”, IEEE African 2011- The Falls Resort and Conference Center, Livingstone, Zambia, 13-15 September 2011

[13] António Sérgio Ferreira, “Interface de Operação para Veículos não Tripulados”, Tese de Mestrado, Faculdade de Engenharia da Universidade do Porto, 2011

[14] P. Dias, R. Gomes, J. Pinto, G. Gonçalves, J. Sousa, F. Pereira, “Mission Planning and Specification in the Neptus Framework”, Proceedings of the 2006 IEEE International Conference on Robotics and Automation, Orlando, Florida – May 2006

[15] R. Bencatel, J. Correia, J. Sousa, G. Gonçalves, E. Pereira, “Video tracking control algorithms for unmanned air vehicles”, ASME Dynamic Systems and Control Conference, Michigan, USA, October 2008

[16] E. Pereira, R. Bencatel, J. Correia, L. Félix, G. Gonçalves, J. Morgado, J. Sousa, “Unmanned air vehicles for coastal and environmental research”, 10th International Coastal Symposium , Lisbon, Portugal, April 2009

[17] “O Programa de Investigação e Tecnologia em Veículos Aéreos Autónomos Não-Tripulados Da Academia Da Força Aérea”, Cadernos do IDN nº4, 3ªB revisão, 29 de Julho 2009

[18] “Unmanned Systems Integrated Roadmap FY2011-2036”, Department of defense of the United States of America, 11-S-3613

[19] NASA, http://www.grc.nasa.gov/WWW/k-12/airplane/rotations.html, acedido em Julho de 2012

[20] Cloud Cap Technology, http://www.cloudcaptech.com/, acedido em Julho de 2012

[21] How Stuff Works, http://science.howstuffworks.com/transport/flight, acedido em Julho de 2012

[22] Diy Drones, http://code.google.com/p/ardupilot-mega/wiki/APM2board, acedido em Julho de 2012

[23] D. Cole, A. Goktogan, P. Thompson e S. Sukkarieh, “Mapping and Tracking”, IEEE Robotics & Automation Magazine, June 2009

[24] PC-104, http://www.pc104.org/index.php, acedido em Julho de 2012

[25] IGEP, http://igep.es/products/processor-boards/igepv2-board, acedido em Julho de 2012

[26] R. Martins, P. Dias, E. Marques, J. Sousa, J. Pinto, F. Pereira, "IMC: A Communication Protocol for Networked Vehicles and Sensors" LSTS - Underwater Systems and Technology Laboratory, Faculdade de Engenharia da Universidade do Porto

[27] J. Sousa, G. Gonçalves, A. Costa, J. Morgado, “*Mixed Initiative control of unmanned air vehicles systems: the* PITVANT R&D UAV *Program”*

[28] J. Pinto, P. Calado, J. Braga, P. Dias R. Martins, E. Marques, J. Sousa, “Implementation of a Control Architecture for Networked Vehicle Systems”

[29] A. Tsourdos, B. White, M. Shanmugavel, “Cooperative Path Planning of Unmanned Aerial Vehicles”, Aerospace Series, Wiley, 2011

[30] Y. Qu, Q. Pan, J. Yan, “Flight Path Planning of UAV Based on Heuristically Search and Genetic Algorithms”, Northwestern Polytechnical University

[31] S. Mittal, K. Deb, “Three-Dimensional Offline Path Planning for UAVs Using Multiobjective Evolutionary Algorithms”, IEEE, 2007

[32] Lixia, Xiejun, C. Manyi, X.ieming, W. Zhike, “Path Planning for UAV Based on Improved Heuristic A* Algorithm”, IEEE, 2009

[33] S. Lavalle, “ Planning Algorithms”, University of Illinois, 2006

[34] O. Khatib, “The Potential Field Approach And Operational Space Formulation In Robot Control”, Artificial Intelligence Laboratory Stanford University, 1985

[35] S. Bortoff, “Path Planning for UAVs”, Proceedings of the American Control Conference, Chicago, Illinois, June 2000

[36] O. Hammouri, M. Matalgah, “Voronoi Path Planning Technique for Recovering Communication in UAVs, IEEE, 2008

[37] E. Bakolas, P. Tsiotras, “The Zermelo-Voronoi Diagram: a Dynamic Partition Problem”

[38] R. Omar, D. Gu, “Visibility Line Based Methods for UAV Path Planning”, ICROS-SICE International Joint Conference 2009, August 18-21, 2009, Fukuoka International Congress Center, Japan

[39] L. Jaillet, Juan Cortés, T. Siméon, “Transition-based RRT for Path Planning in Continuous Cost Spaces”, International Conference on Intelligent Robots and Systems, IEEE, 2008

[40] T. Khuswendi, H. Hindersah, W. Adiprawita, “UAV Path Planning Using Potential Field and Modified Receding Horizon A* 3D Algorithm”, International Conference on Electrical Engineering and Informatics, 17-19 July 2011, Bandung, Indonesia

[41] B. Kuipers, “Potential Fields and Model Predictive Control”, Lecture 7, CS 344R, Robotics

[42] L. Reis, N. Lau, “Intelligent Robotics – Mapping and Navigation”

[43] Y. Li, H. Chen, M. Er, X. Wang, “Coverage path planning for UAVs based on enhanced exact cellular decomposition method”, Elsevier, 2010

[44] D. Chen, R. Szczerba, J. Uhran, “A Framed-Quadtree Approach for Determining Euclidean Shortest Paths in a 2-D Environment”, IEEE Transactions on Robotics And Automation, Vol. 13, No. 5, October 1997

[45] Berkeley, http://math.berkeley.edu/~sethian/2006/, acedido em Julho de 2012

[46] T. Nishida, K. Sugihara, “FEM-like Fast Marching Method for the Computation of the Boat-Sail Distance and the Associated Voronoi Diagram”, University of Tokyo, 2003

[47] E. Dijkstra, “A note on Two Problems in Connexion with Graphs”, Numerische Mathematik 1, 269-271, 1959

[48] Dijkstra, http://optlab-server.sce.carleton.ca/POAnimations2007/DijkstrasAlgo.html, acedido em Julho de 2012

[49] S. Russell, P. Norvig, “Artificial Intelligence A Modern Approach”, Pearson, third edition, 2010

[50] P. Hart, N. Nilsson, B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”, IEEE Transactions of Systems Science and Cybernetics, vol. ssc-4, no.2, July, 1968

[51] Y. Qu, Q. Pan, J. Yan, “Flight Path Planning of UAV Based on Heuristically Search and Genetic Algorithms”, IEEE, 2005

[52] X. Yang, M. Ding, C. Zhou, “Fast Marine Route Planning for UAV Using Improved Sparse A* Algorithm”, 2010 Fourth International Conference on Genetic and Evolutionary Computing

[53] A. Stentz, “Optimal and Efficient Path Planning for Partially-Known Environments”, IEEE, 1994

[54] IEEE standard for application and management of the system engineering process. IEEE Std 1220 -2005 (Revision of IEEE Std 1220-1998), páginas 1-87, 2005

[55] C. Cheng, K. Fallahi, H. Leung, “Cooperative Path Planner for UAVs Using ACO Algorithm With Gaussian Distribution Functions”, International Symposium on Circuits and Systems, 2009

[56] Densidade do ar, http://www.aerospaceweb.org/question/atmosphere/q0046b.shtml, acedido em Julho de 2012

[57] J. Sethian, “Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision and Materials Science”, Cambridge University Press, 1999

[58] T. Cormen, C. Leiserson, R. Rivest, C. Stein, “Introduction to Algorithms”, Second Edition. MIT Press and McGraw-Hill, 2001.

Anexo A. Tutorial de utilização do plugin SPP

1 – Executar o Neptus

2 – Abrir uma consola, e selecionar o veículo

3 – Selecionar o icon presente na Figura 77, indicado com uma seta amarela, será apresentado o menu da mesma figura.

4 – Selecionar no menu, as opções pretendidas, começando por definir o canto inferior esquerdo e superior direito do cenário de operação, o ponto inicial e final do trajeto, e adicionar os obstáculos pretendidos.

5 – Selecionar a opção para gerar o plano de voo

6 – Se o piloto concordar com o trajeto calculado que apareceu no mapa da consola do Neptus, pode executar esse trajeto e monitorizá-lo.

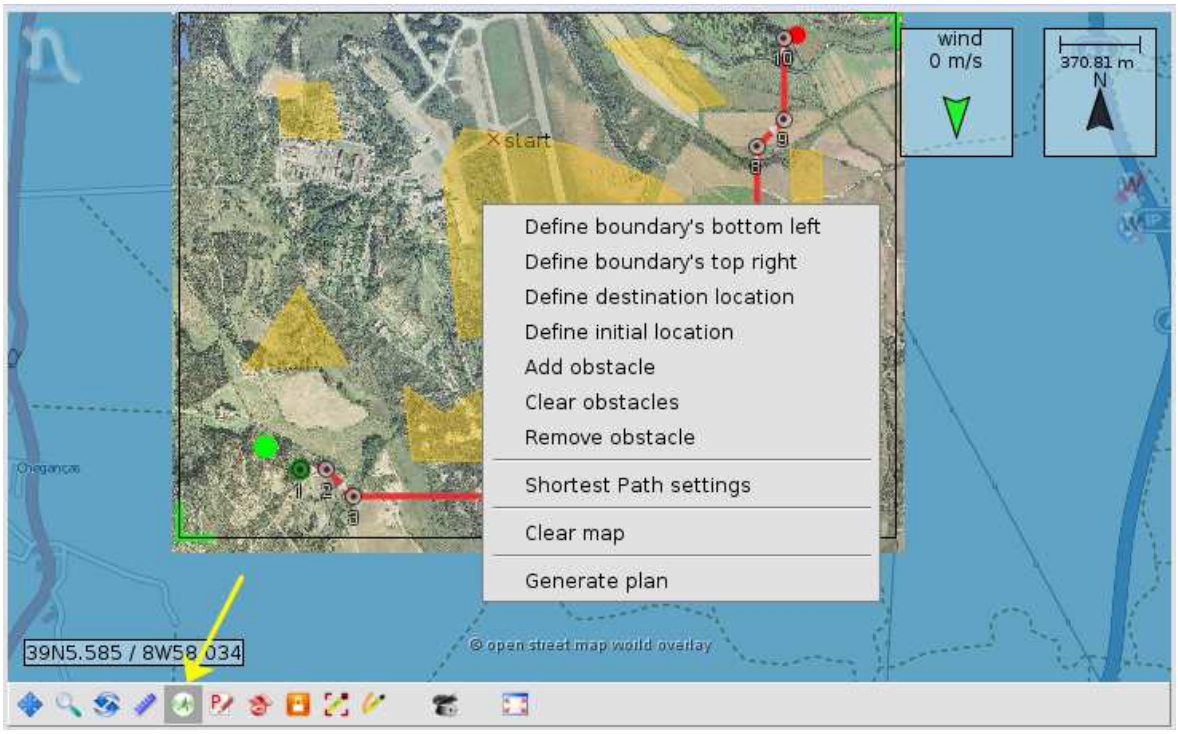

Figura 77 Menu do plugin SPP