ACADEMIA MILITAR
MILITARY ACADEMY
TÉCNICO LISBOA

Enxame de Drones: Otimização de Áreas de Pesquisa

João Pedro Figueiredo Gomes de Oliveira

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica Militar

Orientadores: Prof. Doutor José Silvestre Serra da Silva
Prof. Doutor Alexandre José Malheiro Bernardino

Outubro 2024

Declaração

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Agradecimentos

Em primeiro lugar, gostaria de agradecer aos meus orientadores, Professor Doutor José Silvestre Serra da Silva e Professor Doutor Alexandre José Malheiro Bernardino, pela disponibilidade, conselhos e paciência que tiveram comigo ao longo da realização da dissertação.

Agradeço a todos os instrutores e docentes que privaram comigo na Academia Militar, que contribuíram para a minha formação enquanto militar e aluno e que tiveram impacto na pessoa que sou hoje.

Agradeço aos camaradas da turma Foxtrot 2018/2019. A vossa companhia e amizade foi determinante para ultrapassar diversas dificuldades.

Agradeço aos meus amigos Tomás, Rui, Pedro, Gonçalo e Vasco. Os vossos conselhos e companhia ao longo destes largos anos foram determinantes para chegar ao dia de hoje! Espero que se mantenham as experiências por muitos mais anos.

Um agradecimento especial aos meus pais por tudo o que me proporcionaram para poder chegar a este patamar e aos meus irmãos Ricardo e Daniela por estarem presentes em todos os momentos da minha vida e terem sido os meus guias nesta jornada. Agradeço também aos meus avós, Conceição e Fernando, por ensinarem a ver sempre o copo meio cheio.

Por último, um agradecimento à minha namorada que me acompanha desde o primeiro dia da minha jornada nesta instituição, tanto nos bons como, especialmente, nos maus momentos.

Abstract

The use of Unmanned Aerial Vehicles (UAVs) in military and police contexts has been drastically increasing in recent years, both individually and in swarms. One of the tasks they can perform is area coverage, which can later be directed towards a Search and Rescue (SAR) operation or forest and urban surveillance.

This dissertation aims to find the optimized solution with minimum time for a Vehicle Routing Problem (VRP). To achieve this, an optimization problem is formulated using a Mixed-Integer Linear Programming (MILP) approach to find the minimum time value. The solution is tested for convex areas separated by sweep segments parallel to each other, thus generating a back and forth sweep pattern. It is also tested for non-convex areas that are decomposed through Delaunay triangulation.

The methods were validated in Python, and it was found that the MILP formulation is essential to achieve the minimum time. Area coverage for non-convex areas, in most cases, obtained lower time values, which is important to get the mission completed faster, when compared to convex polygon method. However, some weaknesses were observed regarding the division of flight time for each UAV.

Keywords

UAV, Area Coverage, MILP, Minimum Time, Delaunay Triangulation.

Resumo

A utilização de Veículos Aéreos Não Tripulados (VANTs) por meios militares e policiais tem vindo a aumentar drasticamente nos últimos anos, tanto individualmente como em enxame. Uma das tarefas que podem executar é a cobertura de uma área que, posteriormente, pode ser direcionada para uma Operação de Busca e Salvamento (OBS) ou para uma vigilância florestal ou urbana.

A presente dissertação tem como objetivo encontrar a solução otimizada com o tempo mínimo para um Problema de Rotas Veículares (PRVs). Para tal, é formulado um problema de otimização com recurso a um problema de Programação Linear Inteira Mista (MILP) para encontrar o valor mínimo. A solução é testada para áreas convexas separadas por segmentos de varrimento paralelos entre si, gerando assim um padrão de varrimento de trás para a frente. É testada também para áreas não convexas que são decompostas através da triangulação de Delaunay.

Os métodos foram implementados em Python e foi assimilado que a formulação do MILP é essencial para atingir o tempo mínimo. A cobertura de área para polígonos não convexos obteve, em maior parte dos casos, valores temporais inferiores, que são importantes para completar a missão mais rápido, quando comparado com o método para polígonos convexos. No entanto, mostrou algumas fragilidades quanto à divisão do tempo de voo para cada VANT.

Palavras Chave

VANT, Cobertura de Área, MILP, Tempo mínimo, Triangulação de Delaunay.

Conteúdo

1 Introdução 1

1.1 Definição do Problema 4

1.2 Objetivo 5

1.3 Estrutura do documento 5

2 Conceitos Teóricos 7

2.1 Arquiteturas de Enxame 9

2.1.1 Arquitetura de Infraestrutura 9

2.1.2 Arquitetura Rede Flying Ad-Hoc 10

2.2 Área de Varrimento 10

2.2.1 Polígonos e Formas 11

2.2.2 Padrões de Pesquisa 12

2.3 Algoritmos Inspirados na Biologia 12

2.4 Problema de Rotas de Veículares 14

3 Trabalhos Relacionados 15

4 Metodologia 23

4.1 Decomposição de Área Convexa 25

4.2 Problema de Otimização 28

4.3 Triangulação de Delaunay 30

5 Resultados 33

5.1 Impacto das Restrições 35

5.2 Área Convexa 37

5.3 Área Não Convexa 40

5.3.1 Área Triangular 40

5.3.2 Área Quadrangular 41

5.3.3 Área em L 42

5.3.4 Estrela de 4 pontas 43

5.3.5 Forma em C 44

5.3.6 Pontos inseridos manualmente 44

6 Conclusão **47**

6.1 Conclusões Finais 49

6.2 Trabalho Futuro e Limitações 50

Bibliografia **51**

A Conceitos teóricos complementares **55**

A.1 MAVLink 55

B Código do Projeto **57**

Lista de Figuras

2.1 Arquitetura baseada numa Estação de Controlo Terrestre (ECT). 10
2.2 Arquitetura Flying Ad-Hoc Network (FANET) para um enxame de Veículos Aéreos Não Tripulados (VANTs) 11
2.3 Vários tipos de formas 11
2.4 Polígono convexo gerado a partir de uma forma côncava. 12
2.5 Padrão Back and Forth através de varrimento paralelo. 12
2.6 Padrão em espiral a iniciar por fora e a terminar no centro. 13
2.7 Representação gráfica de um Problema de Rota Veícular (PRV) demonstrando os nós a visitar. 14

4.1 Direção de varrimento para o menor número de curvas. 25
4.2 Direção de varrimento para o maior número de curvas. 25
4.3 Rotação aplicada ao polígono para obter h_{min}. 26
4.4 Área de captação do sensor do VANT. 26
4.5 Polígono dividido por segmentos de varrimento paralelos. 27
4.6 Grafo composto pelos vértices dos segmentos de varrimento. 27
4.7 Triangulação de Delaunay para um conjunto de pontos V 31
4.8 Triangulação de Delaunay aplicada aos pontos inseridos. 31
4.9 Triangulação de Delaunay apenas considerada dentro da zona de interesse. 32
4.10 Divisão da área em zonas de pesquisa consoante o número de VANTs. 32

5.1 Impacto da utilização de restrições de minimização temporal na solução do problema. . . 36
5.2 Influência da aplicação da restrição para evitar diagonais. 36
5.3 Influência de implementar a restrição que obriga que cada nó seja visitado por apenas um VANT. 37
5.4 Influência do número de operadores para a utilização de VANTs no problema. 38
5.5 Diminuição da autonomia dos VANTs leva à implementação de mais um VANTs. 38

5.6 Impacto do número de operadores na trajetória dos VANTs. 39
5.7 Tempo de processamento do otimizador para o diferente número de segmentos de varrimento. 40
5.8 Resultado para área triangular. 41
5.9 Resultado para área quadrangular com ponto inicial em (0,0). 42
5.10 Resultado para área quadrangular com ponto inicial coicidente com o centro equidistante para os 2 VANTs que vão realizar a pesquisa. 42
5.11 Comparação de resultados para área em ”L”. 43
5.12 Comparação de resultados para ambos os métodos quando se utiliza uma área com a forma de uma estrela de 4 pontas. 43
5.13 Comparação de resultados para ambos os métodos para uma área com forma de ”C”. . . 44
5.14 Comparação de resultados para uma área inicial côncava com mais variações. 45

A.1 Estrutura do protocolo Micro Air Vehicle Link (MAVLink). 56

Lista de Tabelas

4.1 Explicação resumida das variáveis e constantes utilizadas no problema de otimização. . . 28

5.1 Valores das constantes utilizadas para o problema. 35
5.2 Parâmetros iniciais definidos para cada cenário. 35
5.3 Resultados temporais obtidos conforme a implementação das restrições. 37
5.4 Parâmetros iniciais para teste em área convexa. 37
5.5 Valores temporais máximos e número de VANTs utilizados para cada cenário. 39
5.6 Parâmetros iniciais utilizados para comparar ambos os métodos. 40
5.7 Resultados comparativos entre o tempo de cada VANT para os dois métodos. 45
5.8 Resultados temporais, em segundos, obtidos comparando os dois métodos para os vários cenários. 45

Listagens

B.1 Script principal tese.py 57

B.2 Script para calcular a divisão de área por linhas paralelas Strips.py 60

B.3 Script Waypoints.py para estabelecer os waypoints dos VANTs. 63

B.4 Scrit Time.py que calcula o tempo que cada VANT demora a passar pelos waypoints todos. 63

B.5 Script Plot_Path.py para imprimir graficamente o trajeto dos VANTs. 63

B.6 Script cost.py que implementa o otimizador Gurobi para realizar a otimização. 65

B.7 Script AreaDecomp.py para realizar o algoritmo da triangulação de Delaunay. 67

B.8 Script NonConv.py é o script principal para o método de trajetória com a triangulação de Delaunay. 69

Acrónimos

ACO Ant Colony Optimization

ECT Estação de Controlo Terrestre

FANET Flying Ad-Hoc Network

FOV Campo de Visão

IA Inteligência Artificial

MAVLink Micro Air Vehicle Link

MILP Programação Linear Inteira Mista

OBS Operação de Busca e Salvamento

PRC Planeamento de Rota de Cobertura

PRV Problema de Rota Veícular

PSO Particle Swarm Optimization

SAR Search and Rescue

TSP Travelling Salesman Problem

UAV Unmanned Aerial Vehicle

VANT Veículo Aéreo Não Tripulado

VRP Vehicle Routing Problem

1

Introdução

Conteúdo

1.1 Definição do Problema 4
1.2 Objetivo 5
1.3 Estrutura do documento 5

Um Veículo Aéreo Não Tripulado (VANT) é uma aeronave genérica concebida para funcionar sem piloto humano a bordo [1]. A capacidade dos VANTs de transportar cargas, de pesquisar áreas e outras utilizações aéreas, sem um piloto humano a bordo, é uma proposta atractiva. Com uma tripulação a bordo, existe o risco de ferimentos ou morte se ocorrer um erro crítico durante o voo. Com um sistema de aeronaves não tripuladas, estas preocupações são atenuadas [2]. O sistema de um VANT é composto principalmente pela sua estrutura física, equipamento de telemetria, telecontrolo e comunicação, cargas úteis de missão, equipamento de processamento de informação e vários equipamentos de apoio. Atualmente, a nomenclatura VANT pode ser referida por muitas terminologias diferentes, tais como Drone, Veículo Remotamente Pilotado, Aeronave Não Tripulada, entre outras [3].

A expressão enxame de VANTs denota um grupo de VANTs que realiza uma missão onde todas as tarefas estão interligadas. Cada VANT recebe uma tarefa a ser executada. Alguns VANTs podem ser usados para reconhecer ou identificar objetos próximos usando camaras RGB ou camaras térmicas. Outros podem usar radares de abertura sintética para reconhecer objetos a uma distância de vários quilómetros. Consoante a arquitetura de comunicação implementada, todos os VANTs podem comunicar entre si, e apenas um dos VANT é responsável por comunicar com a Estação de Controlo Terrestre (ECT), ou então, cada VANT comunica diretamente com a ECT. Quando apenas um VANT comunica com a ECT ele é utilizado como elemento de transmissão de mensagens de controlo da ECT para qualquer um dos VANTs [4]. A utilização de enxames de VANTs para fazer levantamento de dados em diversas áreas de estudo em simultâneo com a finalidade de realizar tarefas de vigilância e Operações de Busca e Salvamento (OBSs), oferece um leque de opções ao utilizador para poder realizar com sucesso e aumentar a eficiência da missão que quer cumprir, que apenas com o emprego de recursos humanos não seria tão eficaz. O desenvolvimento destas tecnologias aliada a outras formas de tratamento de informação, pode fazer toda a diferença para salvar vidas humanas ou evitar tragédias que podem ser antecipadas com a vigilância e preparação necessárias. Quando o enxame de VANTs realiza missões reais, ele voa em formação sempre que necessário. No processo de aproximação ao alvo, ele não só precisa de evitar obstáculos em tempo real, mas também precisa de evitar a colisão da trajetória entre VANTs no enxame. Como resultado, o enxame de VANTs requer uma grande quantidade de recursos de comunicação e computação para garantir a interação das informações de comunicação em tempo real e o cálculo das informações obtidas [5]. Durante calamidades naturais, como inundações, incêndios, terramotos e tempestades, há dificuldade de acesso aos locais e demora na realização de OBSs [6, 7]. Os VANTs utilizados em OBS podem acelerar estas operações, tornando-as mais rápidas. Os VANTs podem ser montados com vários sensores, como camaras RGB e camaras de visão noturna, úteis para fazer estimativas do desastre, encontrar e localizar sobreviventes de inundações. Além disso, capturam e enviam imagens aéreas em tempo real para a estação terrestre para melhor visualização do cenário. Alguns VANTs são projetados para transportar alguns

quilos de itens essenciais necessários em situações de emergência. Com o uso de enxames de VANTs, as OBS são aceleradas [8].

Uma OBS, que também pode ser definida como um problema de Planeamento de Rota de Cobertura (PRC), requer a navegação numa grande área num período de tempo reduzido para resgatar sobreviventes [9]. Num PRC, a área alvo é normalmente dividida em regiões não intersectadas, denominadas células, utilizando uma técnica de decomposição. A dimensão e a resolução das células podem variar consoante o tipo de decomposição e deve ser aplicada uma estratégia específica para garantir a cobertura completa. No caso das células maiores, são necessários vários movimentos para cobrir completamente uma unidade, enquanto que nas células mais pequenas é suficiente um único movimento. Estas células têm tipicamente a mesma dimensão de um robô (cobertura terrestre) ou são proporcionais ao alcance do sensor (cobertura aérea), representando apenas um ponto na trajetória projectada [10].

1.1 Definição do Problema

Um enxame de VANTs, através de sensores como as câmaras que cada drone possui, está habilitado a realizar missões com a finalidade de efetuar a pesquisa de uma área num determinado ambiente em que uma missão militar/policial está a ser executada com recurso a um enxame de VANTs. O objetivo dessa missão pode ser realizar uma pesquisa numa determinada área, quer seja para reconhecimento, uma OBS ou uma vigilância a uma zona florestal, rural ou urbana. Assim, é necessário antecipar a melhor distribuição de tarefas dentro do enxame, o melhor, mais rápido ou mais seguro caminho para completar a tarefa e a forma mais eficiente e económica de utilizar os recursos de uma forma realista, considerando os custos de empenhamento e economia de forças. Na formulação do problema é assumido que não existe nenhuma informação prévia sobre a área de pesquisa, ou seja, a incerteza é igual em toda a área.

Numa vigilância a uma zona florestal, que é uma zona extensa, o ideal é cobrir a maior área possível e no menor tempo possível para ser conseguirmos obter dados mais atualizados e em maior quantidade. No caso de uma OBS, em que a zona a pesquisar é mais específica, pode ser necessário decompor a área de diferentes formas e distribuir pelos VANTs do enxame para facilitar a obtenção do objetivo da missão.

1.2 Objetivo

O objetivo deste trabalho inclui o desenvolvimento de um método para definir uma trajetória óptima para um enxame de VANTs quadricópteros, evitando obstáculos, utilizando os sensores e as comunicações dos VANTs com a sua base de operações [8]. Utilizar uma metodologia de decomposição de área para tornar a atribuição de zonas de varrimento mais eficiente para um enxame de VANTs e distribuir tarefas a cada membro do enxame de forma mais eficaz. A metodologia será validada usando scripts em Python e o otimizador Gurobi [11] para testar os algoritmos propostos e analisar os comportamentos das metodologias para diferentes cenários.

1.3 Estrutura do documento

A presente dissertação está dividida em seis capítulos. O capítulo 1 faz uma breve introdução ao trabalho, descreve o problema e a motivação para a realização do trabalho, bem como os objectivos a atingir e a estrutura do relatório apresentado. O capítulo 2 detalha conceitos relevantes para o trabalho, nomeadamente noções básicas de uma arquitetura de enxame, protocolos de comunicação, padrões de pesquisa e formas de decomposição de área. No capítulo 3 é apresentada uma revisão da literatura sobre artigos relacionados com aplicações de enxames de drones e métodos utilizados para realizar cobertura de área. No capítulo 4 é apresentada a metodologia para obter a solução do problema, primeiro para uma área convexa e depois para uma área não convexa. O capítulo 5 apresenta os resultados para ambos os tipos de área e mostra também o impacto que a definição do problema de otimização tem na solução final do problema. Finalmente, o capítulo 6 refere as conclusões obtidas sobre o trabalho realizado e propostas futuras a melhorar.

2

Conceitos Teóricos

Conteúdo

2.1 Arquiteturas de Enxame 9
2.2 Área de Varrimento 10
2.3 Algoritmos Inspirados na Biologia 12
2.4 Problema de Rotas de Veículares 14

Neste capítulo irão ser abordados conceitos teóricos sobre a implementação de um enxame de VANTs, como pode ser classificada uma área de pesquisa e quais os padrões que um VANT habitualmente utiliza para realizar a missão.

2.1 Arquiteturas de Enxame

Tradicionalmente, um VANT é controlado por uma ECT, que na maioria dos casos, é representada por um computador com um software de controlo em terra. As informações são enviadas através de um transmissor-recetor que envia e recebe dados dos VANTs ligados. Os dados de telemetria incluem tradicionalmente informações de GPS, velocidade no solo e outros parâmetros recolhidos pelos sensores da carga útil. Os enxames actuais utilizam um de dois tipos gerais de arquitetura de comunicação: podem utilizar uma arquitetura de enxame baseada em infraestruturas e uma arquitetura Flying Ad-Hoc Network (FANET) [12].

2.1.1 Arquitetura de Infraestrutura

Uma ECT é uma arquitetura baseada em infraestrutura que recebe informações de telemetria dos elementos do enxame e envia comandos de volta a cada um deles individualmente. Uma operação de voo pode ser pré-programada a bordo de cada VANT e a ECT é simplesmente utilizada para observar os sistemas. Noutros casos, a ECT envia comandos para o controlador de voo em cada VANT, comunicando em tempo real. Este tipo de enxames não são totalmente autónomos, uma vez que continuam a necessitar de um controlo central para os direcionar para uma tarefa atribuída [12, 13].

Uma ECT é responsável pela coordenação de cada VANT numa arquitetura de enxame baseada em infraestruturas, o que provoca uma falta de redundância do sistema. Em caso de ataque, a dependência do enxame em relação à estação de controlo pode pôr em causa a operação que está a ser realizada. Para que esta arquitetura funcione, é necessário que os VANTs estejam dentro do alcance de propagação da ECT. Quando se utilizam frequências de rádio não licenciadas, é mais fácil que as interferências sejam um obstáculo de comunicação.

Uma desvantagem de um enxame baseado em infraestruturas é a fraca capacidade de carga útil de pequenos VANTs, que limita a execução de canais de comunicação fiáveis. Além disso, apenas a ECT é capaz de coordenar a tomada de decisões de todos os VANTs. A figura 2.1 representa uma arquitetura de enxame baseada na infraestrutura [12].

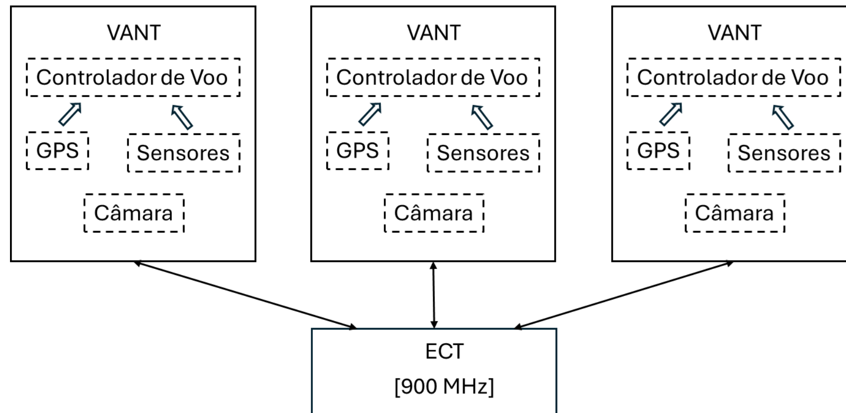


Figura 2.1: Arquitetura baseada numa ECT, adaptado de [12].

2.1.2 Arquitetura Rede Flying Ad-Hoc

Uma FANET é um tipo de arquitetura em que pelo menos um membro do enxame está ligado a uma estação terrestre ou satélite, os restantes VANTs do enxame comunicam entre si sem necessidade de um ponto de acesso. Tal como num sistema baseado em infraestrutura, os VANTs realizam as suas missões sem ajuda humana, como um piloto automático.

Uma rede ad-hoc não necessita de routers ou pontos de acesso porque é uma rede sem fios que não necessita de infra-estruturas físicas para estabelecer uma rede. Para criar uma rede funcional, são utilizados algoritmos de routeamento dinâmicos para atribuir e reatribuir cada nó da rede.

Uma grande vantagem da utilização de uma FANET é a exploração da redundância, uma vez que todo o enxame não está dependente de uma estação de controlo para efetuar as operações a realizar. Esta utilização da comunicação direta entre os elementos do enxame ajuda a melhorar a tomada de decisões do grupo. Por outro lado, um aspeto negativo são as limitações de tamanho, peso e potência, pois para que uma FANET seja estabelecido de forma eficiente, cada VANT precisa de ter hardware que lhe permita estabelecer canais de comunicação. As limitações de potência afectam a distância a que os VANTs podem comunicar [12–14].

2.2 Área de Varrimento

O objetivo que queremos alcançar é referente a uma cobertura de área. A área de varrimento definida vai ser alvo dessa cobertura, que irá originar os trajetos de modo a que o varrimento seja de forma eficiente [15]. A cobertura da área envolve duas operações: a decomposição da área em células e o planeamento de trajetos no interior das mesmas [16].

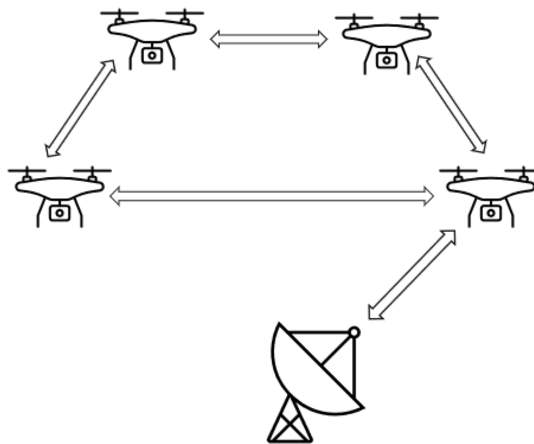


Figura 2.2: Arquitetura FANET para um enxame de VANTs, fonte própria.

2.2.1 Polígonos e Formas

A classificação da forma que a área de pesquisa possui é um fator importante a ter em conta e que vai influenciar o processo de definir as rotas dentro das mesmas. As formas podem ser classificadas de três maneiras, como mostra a figura 2.3.

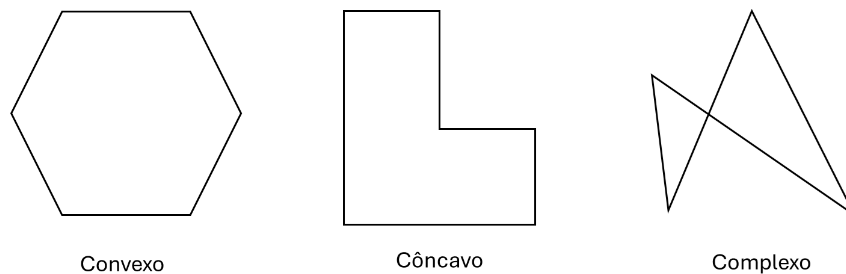


Figura 2.3: Vários tipos de formas, adaptado de [17].

A forma convexa é a mais simples, não possui nenhum ângulo interno superior a 180° e não se interseta a ela própria. Uma forma côncava possui um ângulo interno superior a 180° e também não se interseta a ela própria. Uma forma complexa interseta-se a ela própria.

Um método simples de gerar uma área convexa é através da envolvente convexa. Este método retira vértices interiores e conecta todos os vértices que possuam segmentos externos, como se pode ver na figura 2.4 a tracejado.

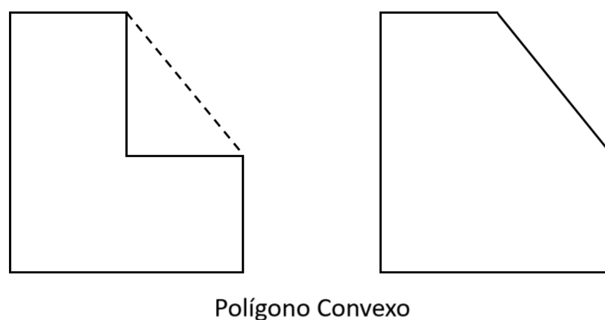


Figura 2.4: Polígono convexo gerado a partir de uma forma côncava, adaptado de [17].

2.2.2 Padrões de Pesquisa

Os padrões de pesquisa a utilizar para analisar uma determinada área podem afetar componentes determinantes como o tempo que demoram a completar a missão, pois cada padrão vai ter uma resposta diferente consoante o formato da área em questão [10].

Um dos padrões utilizados na literatura é o padrão para a frente e para trás, em inglês usualmente conhecido como *Back and Forth*, que consiste em executar varrimentos lineares paralelos entre si. A distância entre cada um destes segmentos vai depender do sensor do VANT a ser utilizado. Existe uma certa sobreposição entre as imagens, como se pode ver na figura 2.5, que vai influenciar a distância entre segmentos.

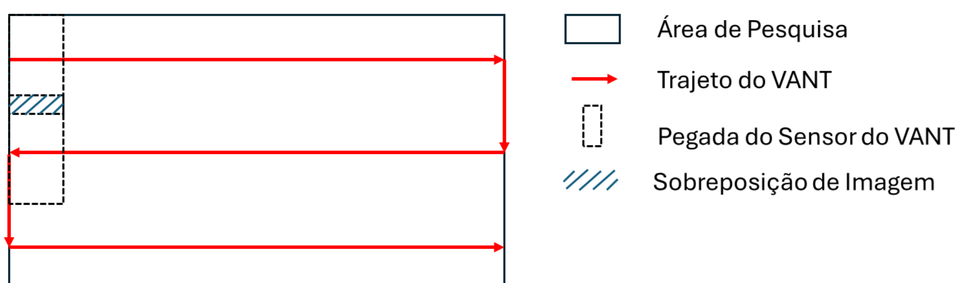


Figura 2.5: Padrão Back and Forth através de varrimento paralelo, fonte própria.

O padrão em espiral baseia-se em começar o varrimento ao longo do perímetro exterior da área de pesquisa e, a partir daí, começar o trajeto para o interior num padrão em espiral, como se pode ver pela figura 2.6.

2.3 Algoritmos Inspirados na Biologia

Os algoritmos explicados a seguir são utilizados para reproduzir o comportamento de grupo que os animais adoptam na natureza. Estes comportamentos ajudam o grupo/enxame a trabalhar de forma

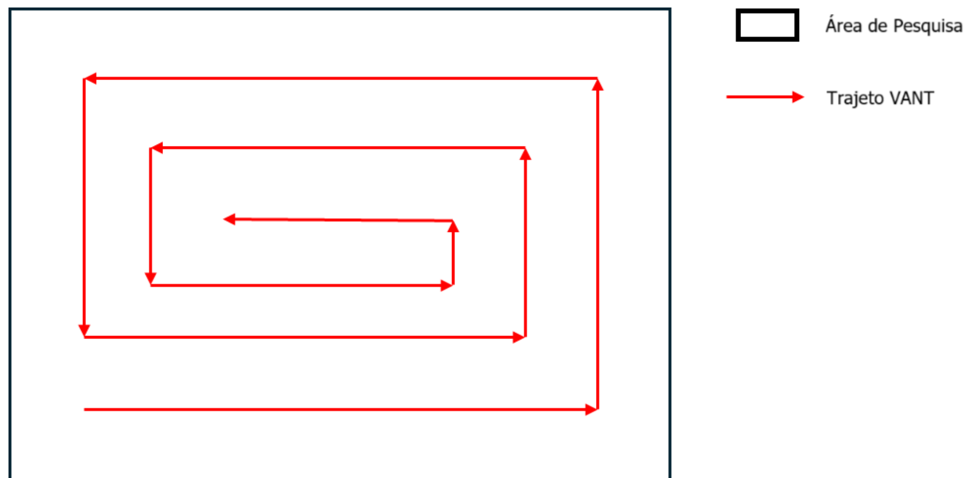


Figura 2.6: Padrão em espiral a iniciar por fora e a terminar no centro, fonte própria.

mais eficiente, quer seja para distribuir melhor e mais eficientemente as tarefas ou para o enxame descobrir o melhor caminho para um determinado objetivo. O algoritmo Particle Swarm Optimization (PSO) simula o movimento de um enxame de partículas num espaço de pesquisa multidimensional, progredindo em direção a uma solução ótima, em que a posição de cada partícula representa uma solução candidata e é iniciada aleatoriamente. Em cada passo do processo iterativo, a velocidade de cada partícula é actualizada individualmente com base na velocidade anterior da partícula, na melhor posição alguma vez ocupada pela partícula e na melhor posição alguma vez ocupada por qualquer partícula do enxame [18].

O algoritmo Ant Colony Optimization (ACO) é inspirado na forma como as formigas encontram o caminho mais curto para os nutrientes no mundo real. As formigas biológicas depositam uma hormona chamada feromona que atrai outras formigas e destaca o caminho para uma fonte de alimento. À medida que mais formigas percorrem esse caminho, mais feromona é depositada, aumentando a probabilidade de outras formigas seguirem esse caminho. Para evitar erros ao longo do tempo devido a variações no caminho, as feromonas evaporam-se. Este algoritmo de otimização é normalmente utilizado para resolver o problema Travelling Salesman Problem (TSP) [19] em que o objetivo deste problema é encontrar o caminho hamiltoniano com menor custo num dado grafo totalmente ligado. Um caminho hamiltoniano é um tipo específico de caminho que visita cada vértice exatamente uma vez num grafo [20].

A probabilidade de uma colónia de formigas se deslocar para um nó que ainda não foi visitado é calculada e influenciada pela quantidade de feromona na aresta (caminho) para esse nó. Quando a matriz de nós não visitados estiver vazia, a colónia terá o ciclo hamiltoniano completo. Um ciclo hamiltoniano é um caso especial de um caminho hamiltoniano em que o último vértice do caminho está ligado ao primeiro vértice, formando um ciclo fechado. Após cada aresta percorrida, os níveis de

feromona serão actualizados, encorajando as formigas subsequentes a escolher outras arestas para criar soluções diferentes. Este algoritmo demonstrou ser uma abordagem muito eficaz para problemas de *routing* em redes de telecomunicações, tais como o custo de utilização de ligações que variam ao longo do tempo [21].

2.4 Problema de Rotas de Veículares

Um Problema de Rota Veícular (PRV) é o nome genérico atribuído a um problema que envolve a visita, com uma viatura, a clientes. O objetivo é encontrar o valor mínimo de distância a percorrer (ou tempo ou custo). Este tipo de problema surge muito frequentemente em situações práticas que não estão diretamente relacionadas com a entrega de mercadorias. Por exemplo, a recolha de correio das caixas de correio, o transporte de crianças por autocarros escolares, as visitas domiciliárias de um médico, as rondas de inspeção de manutenção preventiva, etc., são todos PRVs nos quais a operação de "entrega" pode ser uma recolha, recolha e/ou entrega, ou nenhuma delas; e nos quais os "bens" e "veículos" podem assumir uma variedade de formas, algumas das quais podem nem sequer ser de natureza física [22].

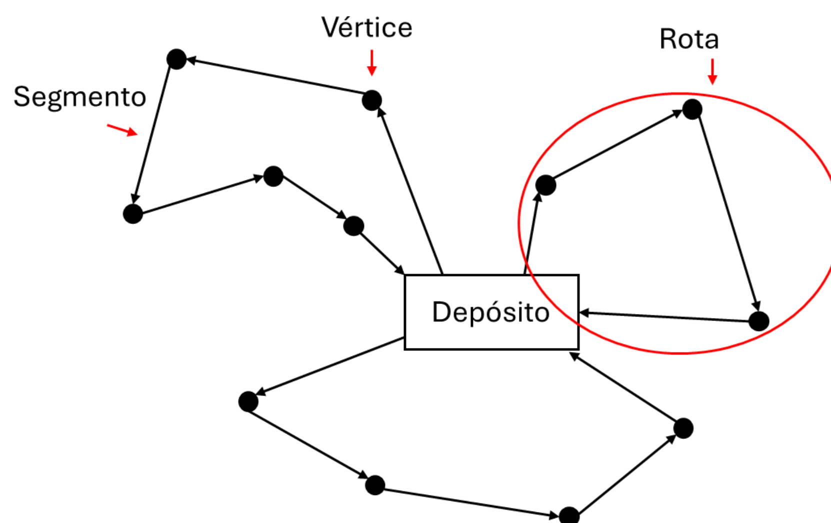


Figura 2.7: Representação gráfica de um PRV demonstrando os nós a visitar.

Pode-se observar pela figura 2.7 como funciona a lógica de um PRV. A viatura sai do depósito e tem que passar por todos os vértices pelo menos uma vez. O problema gerado é composto por um grafo $G = (V, E)$ composto pelos vértices, V, e pelas arestas ,E. Cada aresta representa o caminho entre os vértices i e j e possui o respetivo custo C_{ij} [23]. Assim, estão traçados os aspetos iniciais para mais à frente se iniciar o processo de otimização da trajetória.

3

Trabalhos Relacionados

Para otimizar o padrão de um enxame de VANTs na monitorização de incêndios florestais, Saffre et al. [24] utilizam VANTs equipados com sensores infravermelhos para detetar a assinatura de calor de um incêndio distante. A estrutura que foi desenvolvida considera o VANT em um de três estados: procura, seguimento ou evasão. Os VANTs transitam entre estados com o uso de certos estímulos que iniciam uma mudança de comportamento com a intenção de cercar o fogo para saber onde estão os limites das chamas. O ambiente de simulação foi num terreno plano com dois cenários: simétrico (sem vento) e tendencioso (com vento); a velocidade do VANT foi configurada para 10 m/s. A influência do tamanho do enxame pareceu não ser linear: no cenário simétrico, enquanto menos de 10% dos ensaios resultaram num cerco bem sucedido por um enxame composto por apenas 4 unidades, este valor salta para mais de 90% quando se adiciona um único VANT à frota. Quando o cenário tendencioso foi testado com 5 VANTs, o enxame ficou aquém de atingir 50% de taxa de sucesso, enquanto com 6 VANTs atinge 75%. À medida que a área ardida aumenta, o número de drones necessários para atingir o objetivo aumenta linearmente.

Arranz et al. [25] propõem um sistema para efetuar a vigilância de uma determinada área e para procurar e seguir alvos terrestres, para aplicações de segurança e das forças de segurança. Para tal, propõem um sistema híbrido de Inteligência Artificial (IA), integrando a *deep reinforcement learning* numa arquitetura centralizada de enxame de VANTs. Um controlador central do enxame é responsável pela distribuição de diferentes tarefas de pesquisa e seguimento entre cada VANT do enxame. O cenário de simulação tem como objetivo a busca e seguimento de alvos numa área de operação quadrada com 6 km. Foram realizadas várias análises com alterações ao cenário: uma análise estatística com 10 simulações para avaliar o comportamento dos VANTs com o modelo obtido, exploração da capacidade de adaptação do enxame a um número diferente de alvos e como o enxame se comporta com uma variação do número de VANTs. No primeiro caso, o sistema foi capaz de detetar 95% dos alvos. A média do tempo de aquisição do alvo foi de 2,8 s para o primeiro alvo e 71,9 s para o quinto e último. No segundo caso, com 8 VANTs e alterando o número de alvos progressivamente (um, três, cinco e seis), o enxame gasta mais tempo a encontrar alvos à medida que estes aumentam. Para o último caso, em que há 5 alvos e o número de VANTs foi estabelecido em 8, 9 e 10, o tempo de aquisição de alvos diminui significativamente à medida que o número de VANTs aumenta.

Fan et al. [26] propõem um método de planeamento da trajetória de pesquisa de um enxame de VANTs baseado na probabilidade de contenção. Para melhorar a eficiência da pesquisa do enxame em áreas de desastre, é introduzido o gráfico de probabilidade de distribuição do alvo na informação prévia. Com base no conceito de probabilidade de contenção, é construído um método de divisão de áreas de tarefa para áreas poligonais e circulares, e é construída a correspondente trajetória de pesquisa. Em seguida, a influência dos factores, incluindo a probabilidade de contenção, a probabilidade de deteção e a probabilidade de sucesso na eficiência da pesquisa, é classificada e a função objetivo da otimização

da trajetória de pesquisa é construída. Para verificar a eficiência do método utilizado foram realizadas simulações. Uma área de $240km \times 360km$ está dividida por uma grelha de 40×60 com um gradiente de cores que representa a probabilidade de existir um alvo em cada célula. São utilizados 6 VANTs com dois pontos iniciais distintos. São geradas então 6 áreas poligonais para ser realizada a pesquisa sendo que nas zonas em que existe maior probabilidade, as áreas são mais pequenas para permitir que mais VANTs as analisem. O método desenvolvido, nas zonas de menor probabilidade aumenta o espaçamento entre varrimentos para diminuir o tempo de busca, enquanto que nas zonas de maior probabilidade a distância entre segmentos é diminuída para aumentar a eficiência. Com este método foi atingida uma taxa de sucesso de 94%.

Kim et al. [27] propõem um modelo de envolvimento e um algoritmo de atribuição óptima de tarefas para operações de enxames VANTs em ambientes marítimos hostis. Os autores estabeleceram os seguintes pressupostos: um navio pode atacar o VANT dentro de um determinado alcance com uma taxa de probabilidade de acerto constante; e a capacidade de um navio para atacar VANTs é reduzida se vários VANTs estiverem envolvidos. Inicialmente, é formulada uma função objetivo para a atribuição óptima de tarefas de acordo com a interação entre VANTs e navios. Depois, é utilizada uma modificação do algoritmo PSO normal para otimizar a missão de atribuição de tarefas. O Social-Learning PSO [28] foi o algoritmo utilizado para resolver a tarefa de atribuição de tarefas. Esta modificação imita o comportamento social dos animais em que cada partícula aprende com ela própria e não com os dados históricos de um indivíduo e do enxame, como no PSO convencional. Foram efectuadas simulações numéricas para testar a solução proposta. Foram efectuadas simulações de Monte Carlo com 30 execuções independentes para simular a incerteza do ambiente marítimo, utilizando o MATLAB. No cenário de simulação, foram utilizados 10 navios e o número de VANTs varia entre 5 e 160 e utilizam dois tipos de formações: retangular e circular. Em todos os casos, o algoritmo proposto obtém sempre melhores resultados em comparação com o PSO genérico. No caso circular, com o algoritmo modificado, a formação circular atinge uma taxa de destruição de 100% com 90 VANTs, enquanto que o PSO original apenas atinge o total com 120 VANTs. Para a formação retangular com um ângulo de direção de 180º, o número de VANTs é 90 para o modificado em comparação com 120 para o original. Com um ângulo de direção de 0º, o algoritmo modificado atinge uma taxa de 100% para uma frota de VANTs de 160, enquanto que o algoritmo original atinge uma taxa de destruição máxima de 85% para os mesmos 160.

O estudo de Cho et al. [9] propõe um método para gerar um trajeto que cobre todos os nós estabelecidos, no mais curto espaço de tempo, com múltiplos VANTs heterogéneos. O modelo proposto, que é um modelo de Programação Linear Inteira Mista (MILP) baseado num método de decomposição que utiliza uma grelha hexagonal, foi verificado através de uma análise de simulação que avalia o desempenho de um VANT real. A área de pesquisa neste estudo está dividida em células de grelha

hexagonal, assumindo que cada UAV se move através do centro de cada hexágono. A área que um VANT consegue analisar é referente às características da câmara e os autores consideram como sendo uma área circular que engloba cada célula. A rota é gerada através do problema de programação linear em que cada célula representa um nó e através da aplicação de restrições o caminho vai ser traçado consoante a estratégia definida, que neste caso é minimizar o tempo. As simulações foram realizadas em Python com o auxílio da biblioteca Gurobi para resolver o problema linear. Os resultados foram testados para uma área, de formato semelhante, com 21, 31 e 45 nós. Num ensaio a área estava decomposta através de células hexagonais e noutro ensaio com células quadrangulares. Quando foi utilizada a decomposição quadrangular para a mesma área, o número de nós aumentou para 28, 43 e 56. Em todos os testes, quando foi utilizada a divisão em hexágonos, o tempo foi em média 10,49% mais baixo do que nas células quadrangulares.

No estudo realizado por Skorobogatov et al. [29], os autores apresentam uma solução para dividir qualquer área complexa em várias partes, de forma a que consigam solucionar o problema de planear o melhor trajeto para cada VANT de um enxame. Para dividir a área, os autores utilizaram o algoritmo de triangulação de Delaunay para fazer a divisão de uma área não convexa em várias partes. Cada triângulo vai representar um nó de um grafo direto que vai ser utilizado para escolher como se vão juntar os subpolígonos que foram gerados. A trajetória dos VANTs é atribuída segundo um padrão *back and forth*. O estudo de caso foi realizado através de uma simulação em Python em que os autores utilizaram uma área não convexa irregular com uma forma semelhante a um ”L” e utilizaram a técnica de decomposição de área para dividir consoante o número de drones a serem empenhados. Foi possível observar que o padrão de pesquisa *back and forth* foi otimizado porque os VANTs utilizaram direções de varrimento diferentes para cada subpolígono.

Para lidar com o problema de exercer uma cobertura de área rápida para vários VANTs, Luna et al. [30] propoem uma abordagem que gera um caminho com o padrão *back and forth* e os autores acrescentam um algoritmo *bin packing trajectory planner* (BINPAT) e *Powell optimized*-BINPAT. Ambos os métodos utilizam algoritmos heurísticos, atribuição de somas lineares e técnicas de minimização para otimizar a tarefa de planeamento de rota. As simulações foram realizadas em Gazebo com o auxílio do software PX4. Os algoritmos foram testado para dois cenários, um com os segmentos de varrimento mais longos (máximo de 320m) e outro para os segmentos mais curtos (máximo de 150m). Em ambos os cenários, o coeficiente de custo de variação foi sempre menor para o POWELL-BINPAT. Nos testes realizados no terreno foi utilizado o cenário do segmento mais curto. Os drones utilizados possuíam uma estrutura *hexacopter S550* um *Pixhawk autopilot* e outros componentes como hélices, motor, sensor GPS e câmara, antena wireless Dual Band, recetor e um *Onboard PC*. Os resultados obtidos foram semelhantes aos da simulação em Gazebo em que o algoritmo POWELL-BINPAT obteve sempre melhores resultados. Para este cenário, em que foram utilizados 3 VANTs para analizar a área,

o POWELL-BINPAT demorou 285 segundos a realizar a tarefa enquanto que o BINPAT demorou 300 segundos.

No trabalho realizado por Albani et al. [31] é repensada a abordagem de captar informações de alta qualidade sobre uma determinada área de cultivo com a utilização de VANTs. Os autores propoem um sistema multi-drone descentralizado para um problema de cobertura de campo e mapeamento de ervas daninhas, que é eficiente, intrinsecamente robusto e escalável para diferentes tamanhos de grupos. Para resolver este problema é utilizada uma estratégia de implementação descentralizada baseada no comportamento coletivo de um enxame de abelhas [32]. Esta estratégia designa dinamicamente os VANTs para analisar áreas diferentes e reatribiu outras zonas quando necessário. As simulações foram realizadas utilizando o ambiente de simulação MASON. Para testar a mobilização dinâmica de VANTs do enxame foi atribuído um cenário em que os VANTs têm que se mover para diferentes áreas e detetar ervas daninhas. Foram consideradas 9 áreas numa grelha 3×3 e, por sua vez, cada área é composta por 2500 células numa grelha 50×50. Algumas células são geradas aleatóriamente como células de ervas daninhas. Foram testados três cenários para um número de VANTs $N \in \{18, 27, 36\}$. Para o cenário com 18 VANTs a área foi toda analisada e a função de erro atingiu o menor valor depois de 3000 segundos, para 27 VANTs foi mínima para 2500 segundos e por último, com 36 agentes foi atingido o erro mínimo em 2000 segundos.

Para cobrir uma área eficazmente utilizando vários VANTs Araújo et al. [33] implementaram uma cobertura de área dividida em duas etapas: decomposição da área em células e planeamento da rota dentro das células. A área é decomposta utilizando uma técnica de varrimento por linhas e a rota é traçada através de um método que determina o número mínimo de voltas que o VANT tem que realizar para minimizar o tempo da missão. Para permitir ao operador interação e prioridade na cobertura de área é aplicado um planeador *lawnmower/Zamboni*. As simulações foram realizadas em MATLAB com um cenário que era composto por uma área de 500000 m^2, com 2 VANTs com uma velocidade de 18 m/s e a largura da imagem que o VANT é capaz de analisar de 140 m, é esta largura que vai determinar a distância entre as filas do trajeto. Com os parâmetros que foram utilizados, a incerteza máxima do problema estabilizou aos 180 segundos. Os testes práticos foram realizados com um planador Cularis para uma área de 125000 m^2 com uma velocidade de movimento de 11 m/s e uma largura de imagem de 72 m. Durante a realização do teste existiu um vento entre 10 a 12 m/s que influenciou o resultado final em que a máxima incerteza estabilizou ao fim de 800 segundos, cerca de 13 minutos.

Os autores Pellegrino et al. [34] abordam o problema de cobrir uma determinada área, com um enxame de drones, dividindo a área em blocos. A área é dividida consoante o número de VANTs disponíveis e de seguida a forma utilizada para fazer a partição da área é através de diagramas de *Voronoi* em que cada célula é atribuida a um VANT ativo. As simulações foram realizadas com o ROS e o ambiente de simulação criado em Gazebo. Além dos ROS/Gazebo implementaram também

o `drone_middleware framework` para facilitar o controlo de múltiplos VANTs. O cenário de simulação é composto por uma área de 400 m^2 dividida em células $1m \times 1m$, com 8 VANTs disponíveis para serem utilizados e os VANTs que são utilizados na tarefa são distribuidos equitativamente pela área toda antes de começarem a missão. Entre a utilização de 1 a 4 VANTs para analisar a área, o tempo de duração diminui bastante. Para 1 VANT demorou 35 minutos, com 2 demorou 18 minutos e com 3 e 4 demorou 12 minutos e 9 minutos, respetivamente. Entre 5 e 8 VANTs a diminuição de tempo teve um comportamento linear entre 8 e 5 minutos. Esta diferença na variação pode-se explicar devido às regiões de *Voronoi* se tornarem mais irregulares à medida que o número de VANTs aumenta.

O trabalho de Balampanis et al. [35] aborda os problemas da decomposição de área em células e da divisão de uma região costeira para uma equipa de VANTs heterogéneos, com uma abordagem que tem em conta o campo de visão dos sensores a bordo. O processo de divisão em células exatas é realizado em duas etapas. Na primeira etapa, um algoritmo de regiões crescentes efectua uma divisão isotrópica da área com base nas localização inicial dos VANTs e das suas capacidades relativas. De seguida, são aplicados dois novos algoritmos para executar o processo de divisão, que visam resolver situações de impasse que geram regiões e sub-áreas não atribuídas acima ou abaixo das capacidades relativas dos VANTs. Os algoritmos propostos são implementados em C++ e para a estrutura do problema é utilizado o ROS. Foram escolhidos três cenários ao longo da costa da Grécia com vários arquipélagos para implementar a metodologia. Para testar a divisão de área foram testadas duas variantes do algoritmo AWP para 3 VANTs e para diferentes parâmetros de campo de visão e alcance a divisão foi bem feita, os piores casos foi para quando os pontos iniciais dos VANTs eram muito próximos e podia causar sobreposição das células. O trajeto é escolhido através de uma espiral da fronteira para o centro. Para testar o trajeto foi selecionado um dos cenários específicos com 3 VANTs e a área foi toda coberta com sucesso.

Os autores Szklany et al. [36], para resolver um problema de cobertura de área, apresentam um algoritmo intitulado "Tsunami" que auxilia na recolha de dados de enxames de UAV. O Tsunami divide dinamicamente os ambientes, evita colisões aéreas e responde a alterações na dimensão do enxame no decorrer de missões longas devido a descargas de bateria e falhas. O Tsunami tem duas fases: a fase offline e a fase de execução. A fase offline pega num ambiente de entrada fornecido pelo utilizador, discretiza-o numa série de pontos de passagem GPS e calcula a travessia nos pontos de passagem resultante. Usando essa travessia, a fase de execução envia drones, a partir de um ponto inicial, para visitar os pontos de interesse. Para cada drone, é calculada uma trajetória que evita as trajectórias de voo de outros drones activos. À medida que os drones esgotam as suas baterias, regressam ao ponto inicial até estarem prontos para serem novamente despachados. Este processo continua até que todo o ambiente seja explorado. O algoritmo foi validade em Python e é utilizado um estudo de caso sobre uma plantação de soja para analisar e recolher dados da mesma. O algoritmo foi testado em

comparação com o algoritmo de decomposição celular SCoPP, que utiliza diagramas de Voronoy para realizar a decomposição de área. Os algoritmos foram testados para um enxame de 5 a 20 VANTs, onde o Tsunami obteve sempre valores temporais inferiores ao SCoPP. Para 5 VANTs o Tsunami demorou 4 horas, contra quase 6 horas do SCoPP e para 20 VANTs o Tsunami demorou 1 hora e 40 minutos contra 2 horas do SCoPP.

4

Metodologia

Conteúdo

4.1 Decomposição de Área Convexa 25
4.2 Problema de Otimização 28
4.3 Triangulação de Delaunay 30

A metodologia para resolver os problemas de decomposição de área e rotas veículares pode ser dividida em duas partes. Na primeira parte, a área a ser analisada é decomposta em linhas paralelas que o VANT utilizará para fazer o varrimento. Os vértices destas linhas vão dar origem aos pontos que vão ser utilizados para resolver o PRV. As próximas secções detalham cada método.

A implementação, para uma área convexa, foi baseada no trabalho desenvolvido em [37] e adaptada para uma validação em Python e testada para cenários próprios.

4.1 Decomposição de Área Convexa

Uma abordagem para resolver o problema de cobertura de área é decompô-la em zonas de varrimento e determinando uma sequência de varrimento que vai gerar o caminho para realizar a cobertura dessa área.

O varrimento de linhas planares fundamentado em [38], divide a região de cobertura em regiões monótonas, ou seja, é dividida em subregiões que respeitam uma certa repetição (monotonicidade) referente a um eixo ou direção. Estas subregiões podem ser varridas através de movimentos para trás e para a frente ao longo de segmentos paralelos entre si.

Para diminuir o tempo que leva um VANT a cobrir a área designada, é necessário diminuir o número de mudanças de direção que ele irá fazer. Cada vez que há uma curva o VANT tem que diminuir a velocidade e mudar de direção e depois voltar a aumentar a velocidade.

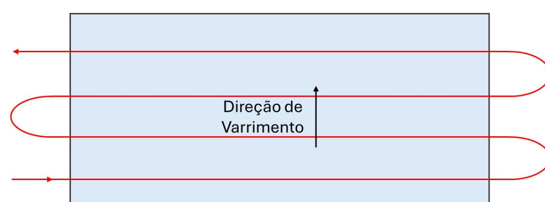


Figura 4.1: Direção de varrimento para o menor número de curvas.

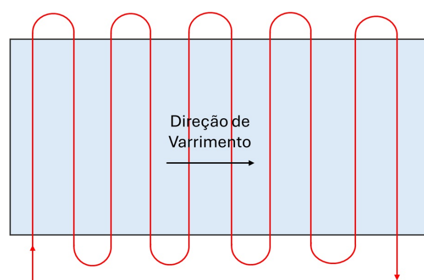


Figura 4.2: Direção de varrimento para o maior número de curvas.

Para encontrar a melhor direção de varrimento de um dado polígono, são testadas várias rotações entre 0º e 360º, iteradas de grau a grau, e selecionada qual a rotação que apresenta a menor altura

h_{min}, que representa o ponto com o maior valor de coordenada y em relação ao eixo x. Ao determinar h_{min} é possível saber qual é a direção que vai possuir o maior segmento de varrimento existente para o polígono analisado. Depois de encontrada a altura mínima sabemos qual é a direção de varrimento ótima, que vai ser perpendicular à altura (ver figura 4.3).

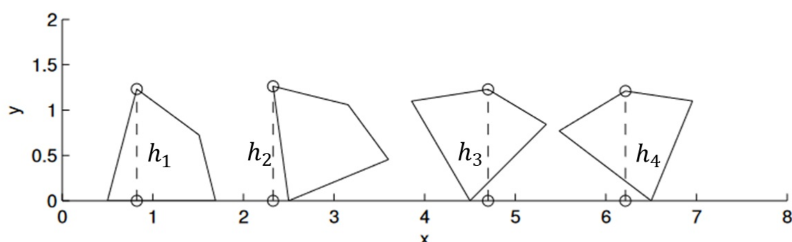


Figura 4.3: Rotação aplicada ao polígono para obter h_{min}.

Com o cálculo da altura mínima e, por sua vez, da direção de varrimento ótima é possível distribuir os segmentos a serem percorridos pela área de cobertura. A distância entre dois segmentos é obtida através das características técnicas da câmara a ser utilizada por determinado VANT. Como pode ser visto na figura 4.4, assumindo que a câmara está sempre apontada para baixo e perpendicular ao solo, sabendo a largura do sensor, l, da distância focal da lente da câmara, f, e a distância entre a câmara e o chão, H (altura de voo), é possível calcular a área que a câmara analisa.

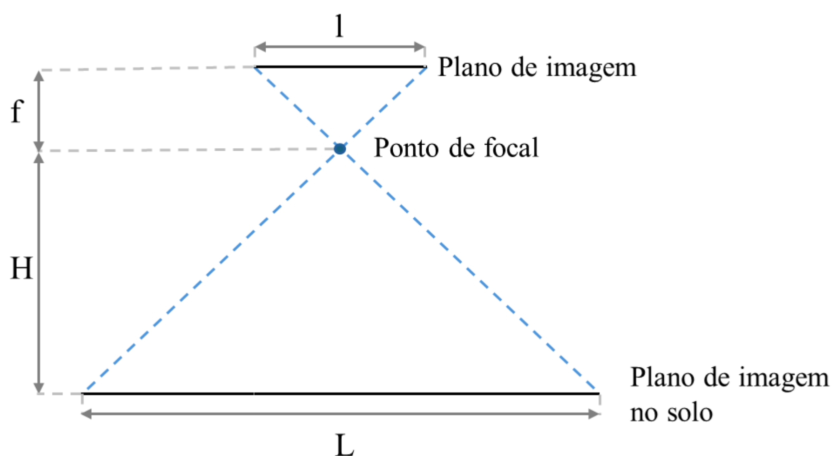


Figura 4.4: Área de captação do sensor do VANT.

O número de segmentos é calculado por:

$$N_l = \left\lceil \frac{h_{\min}}{L(1-s)} \right\rceil \tag{4.1}$$

e a distância entre dois segmentos é:

$$d_l = \frac{h_{\min}}{N_l} \tag{4.2}$$

onde $s \in (0,1)$ representa o rácio de sobreposição entre duas imagens. Esta sobreposição é utilizada para juntar as imagens que compõem um mapa aéreo.

Considere-se um sistema de coordenadas em que a direção ótima seja paralela ao eixo x. Os segmentos de varrimento podem ser definidos por pontos no plano (x, y) com y idêntico dado por:

$$y_i = i \times d_l - \frac{d_l}{2} \quad i = 1, ..., N_l \tag{4.3}$$

as coordenadas x, dos vértices de cada segmento de varrimento, são definidas pelos pontos em que a reta horizontal de coordenadas y_i intercepta os limites da zona a cobrir. A equação (4.3) vai gerar os segmentos obervados na figura 4.5.

Os pontos que vão constituir os nós do grafo $G = (V, E)$ são as extremidades de cada segmento de varrimento, juntamente com as coordenadas do ponto inicial. Cada nó é numerado de forma a que o ponto inicial recebe o número 1 e os vértices do primeiro segmento de varrimento recebem os números 2 e 3, o segundo segmente recebe os números 4 e 5. No final, cada segmento de cobertura está associado a nós pares e ímpares subsequentes. O conjunto de arestas, E, é composto por todas as linhas que ligam os N nós do grafo, formando assim um grafo completo, como se pode ver na figura 4.6.

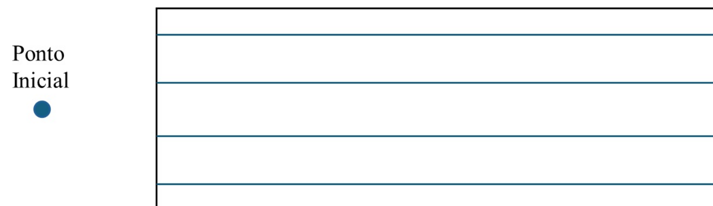


Figura 4.5: Polígono dividido por segmentos de varrimento paralelos.

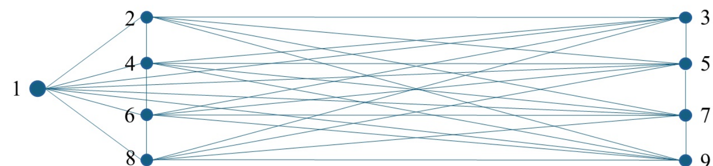


Figura 4.6: Grafo composto pelos vértices dos segmentos de varrimento.

Matematicamente, o grafo G pode ser representado por uma matriz de custo C, de tamanho $N \times N$, cujos elementos C_{ij} são dados pela distância euclidiana entre as coordenadas espaciais dos nós i e j.

Tabela 4.1: Explicação resumida das variáveis e constantes utilizadas no problema de otimização.

Símbolo	Explicação	Tipo
C_{ij}	Matriz de Custo	Variável
$X_{ij}^k \in 0,1$	Variável binária que define se o VANT faz a rota entre i e j	Variável
$V_{ij}^k \in \mathbb{R}$	Velocidade do VANT k entre i e j	Constante
$t_s \in \mathbb{R}$	Tempo que demora a colocar um VANT pronto para a missão	Constante
L_k	Autonomia do VANT k	Constante
$m \in \mathbb{N}$	Nrº de VANTs a utilizar na missão	Variável
$M \in \mathbb{N}$	Nrº de VANTs disponíveis	Constante
$O \in \mathbb{N}$	Nrº de operadores para colocar os VANTs prontos	Constante
$N \in \mathbb{N}$	Nrº de vértices do grafo	Variável
d_k	Tempo que demora a colocar o VANT k pronto para a missão	Variável

4.2 Problema de Otimização

O problema de otimização a resolver utiliza a seguinte notação. A constante C_{ij} representa o custo de travessia entre os vértices i e j. A variável binária $X_{ij}^k \in \{0,1\}$ é utilizada para definir se o VANT de índice k se vai mover (1) ou não (0) do vértice i para j. A velocidade com que o VANT k se move de i para j é definida por $V_{ij}^k \in \mathbb{R}$. A constante $t_s \in \mathbb{R}$ é o tempo individual que o VANT demora a ser configurado e L_k representa a autonomia do VANT k. A variável $m \in \mathbb{N}$ representa o número de VANTs que estão designados para a tarefa. $M \in \mathbb{N}$ é o número total de VANTs disponíveis, $O \in \mathbb{N}$ é a constante do número de operadores de VANTs e $N \in \mathbb{N}$ é o número de vertices do grafo. Por fim, a variável d_k representa o tempo necessário desde que o VANT k é designado até que inicia a missão. É de notar que a constante $t_s \in \mathbb{R}$ e a variável d_k não são iguais, como é explicado a seguir. A Tabela 4.1 apresenta uma explicação detalhada de todas as variáveis e constantes que são utilizadas no problema de otimização.

A fórmula matemática que representa o tempo gasto pelo VANT k a completar a sua rota é dada por (4.4):

$$T_k = \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{C_{ij}}{V_{ij}^k} X_{ij}^k + d_k \tag{4.4}$$

Apesar do objetivo ser diminuir o tempo de missão para o enxame na sua globalidade, este problema é equivalente a minimizar o tempo de missão do VANT que tiver a maior duração. Como queremos minimizar o T_k máximo, o problema é na verdade um problema min-max. Para transformar num problema linear é introduzida uma variável extra ν, que representa a rota de maior duração de todos os VANTs.

O problema de otimização que queremos resolver é o seguinte:

$$min(\nu) \tag{4.5}$$

sujeito a:

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{C_{ij}}{V_{ij}^{k}}X_{ij}^{k}+d_k\leq\nu,\quad k=1,...,M \tag{4.6}$$

$$t_s\left\lceil\frac{k}{O}\right\rceil\sum_{j=1}^{N}X_{ij}^{k}=d_k,\quad k=1,...,M \tag{4.7}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{C_{ij}}{V_{ij}^{k}}X_{ij}^{k}\leq L_k,\quad k=1,...,M \tag{4.8}$$

$$\sum_{k=1}^{M}\sum_{i=1}^{N}X_{ij}^{k}=1,\quad j=2,3,4,...,N \tag{4.9}$$

$$\sum_{i=1}^{N}X_{ip}^{k}-\sum_{j=1}^{N}X_{pj}^{k}=0,\quad p=1,2,3,...,N,\quad k=1,2,3,...,M \tag{4.10}$$

$$u_i-u_j+N\sum_{k=1}^{M}X_{ij}^{k}\leq N-1,\quad i,j=2,3,4,...,N \tag{4.11}$$

$$\sum_{k=1}^{M}X_{i,i+1}^{k}+\sum_{k=1}^{M}X_{i+1,i}^{k}=1,\quad i=2,4,6,...,N \tag{4.12}$$

$$\sum_{k=1}^{M}X_{i,i+1}^{k}=\sum_{k=1}^{M}\sum_{j=\{1,3,...\}\setminus\{i+1\}}X_{i+1,j}^{k},\quad i=2,4,6,...,N \tag{4.13}$$

$$\sum_{k=1}^{M}X_{i,i-1}^{k}=\sum_{k=1}^{M}\sum_{j=\{1\}\cup\{2,4,...\}\setminus\{i-1\}}X_{i-1,j}^{k},\quad i=3,5,7,...,N \tag{4.14}$$

$$\sum_{k=1}^{M}\sum_{j=1}^{N}X_{1j}^{k}=m \tag{4.15}$$

$$m\leq M \tag{4.16}$$

O termo d_k explicado em (4.7) e utilizado na equação (4.6) é referente ao tempo de configuração do VANT, ou seja, o tempo que o operador demora a colocar o VANT pronto para iniciar a missão. No caso de apenas existir um operador, cada VANT vai ter um tempo d_k acumulativo, por exemplo, se existiram dois VANTs, enquanto o VANT 1 está a ser preparado, o VANT 2 está à espera. A variável d_k e a constante t_s apenas vão ter o mesmo valor quando existir um operador para cada VANT.

A primeira condição para garantir a solução do problema de otimização é mostrada na equação (4.8) que garante que o tempo máximo de voo do VANT k é igual ou inferior à duração da bateria L_k. Esta restrição pode tornar o problema inviável no caso de a autonomia do VANT não ser suficiente para cobrir toda a área. Nesse caso, teriam que ser incrementado o número do VANTs disponíveis. No

caso de a aresta entre dois vértices ser maior do que a distância que a autonomia do VANT permite realizar, o problema fica sem solução. Nesse caso, voltamos ao primeiro passo de calcular quais são os segmentos de varrimento mas noutra direção.

Para garantir que os nós do grafo são visitados apenas uma vez por um único UAV são adicionadas as condições das equações (4.9) e (4.10). A equação (4.9) obriga que cada nó seja visitado por apenas um VANT (exceto o ponto de partida). A equação (4.10) garante que o VANT que chega a um nó é o mesmo que sai desse nó. Ambas servem essencialmente para evitar colisões entre VANTs.

A equação (4.11) garante que o trajeto de cada UAV começa e termina no ponto inicial e que o trajeto não tem ciclos internos, onde $u_i \in \mathbb{Z}, i = 2, 3, 4, ..., N$.

Para garantir que a solução do problema vai obrigar os UAVs a cobrir a área modelada pelo grafo G, a condição da equação (4.12) é adicionada. Esta equação garante que cada UAV que visita o nó inicial de um segmento é obrigado a visitar o outro nó do mesmo segmento. Esta condição é essencial para garantir que a solução é de facto uma solução de cobertura de área.

As duas condições das equações (4.13) e (4.14) evitam que os VANTs atravessem a área seguindo um segmento que não é paralelo à direção de varrimento. Na prática, evita que os UAVs façam diagonais, apesar de não comprometer a execução da tarefa.

Por último, para garantir que o número de UAVs utilizados, m, é inferior ou igual ao número máximo de UAVs disponíveis, M, são adicionadas as equações (4.15) e (4.16).

4.3 Triangulação de Delaunay

Nesta secção é apresentada uma abordagem para o caso em que a área de pesquisa que queremos processar não é um polígono convexo. Esta análise para uma área não convexa é necessária para minimizar a perda de tempo sofrida pelos VANTs em analisar espaços que fazem parte de um polígono convexo mas não são do nosso interesse. Para fazer a decomposição de área é aplicada a triangulação de Delaunay [39].

A triangulação de Delaunay é uma triangulação tal que nenhum ponto, no conjunto de pontos V, está dentro do círculo circunscrito de qualquer triângulo, conforme se pode ver pela figura 4.7.

Neste trabalho, a triangulação de Delaunay é aplicada através de bibliotecas Python [40] já predefinidas. Tendo em conta que temos um número M de VANTs disponíveis, e depois da triangulação ser implementada, temos que concatenar os triângulos de forma a criar áreas de pesquisa para que possam ser atribuídas a cada VANT.

Para testar a implementação da biblioteca Python, foram utilizados os dois casos que se podem ver nas figuras 4.8(a), que representa uma forma mais simples em ”L”, e na figura 4.8(b), que é um polígono mais complexo inserido manualmente com a biblioteca ginput do Python [41] que permite ao utilizador

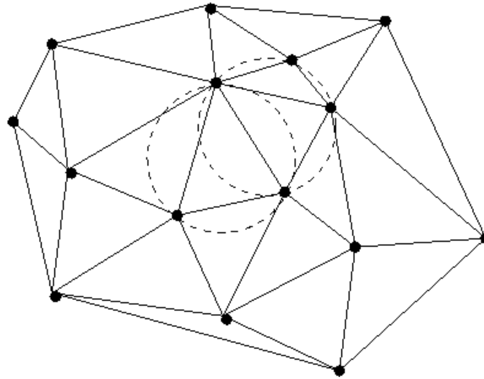
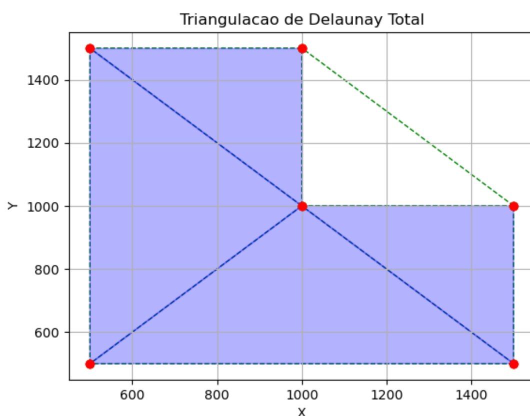
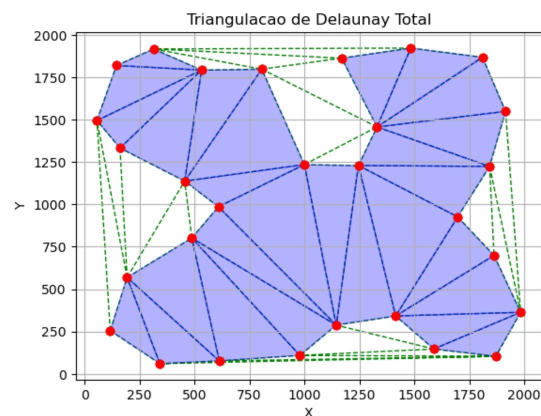


Figura 4.7: Triangulação de Delaunay para um conjunto de pontos V

clicar manualmente os pontos. É possível observar que a triangulação é bem feita e representa o primeiro passo da decomposição de área.



(a) Pontos definidos com o formato L.

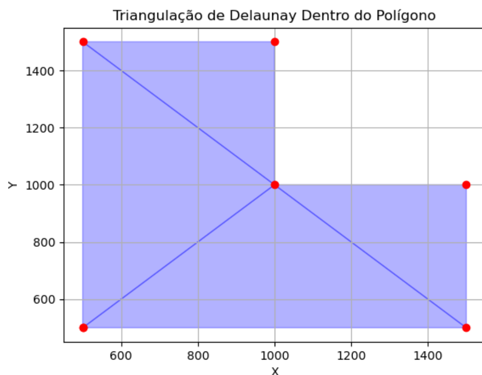


(b) Pontos inseridos manualmente.

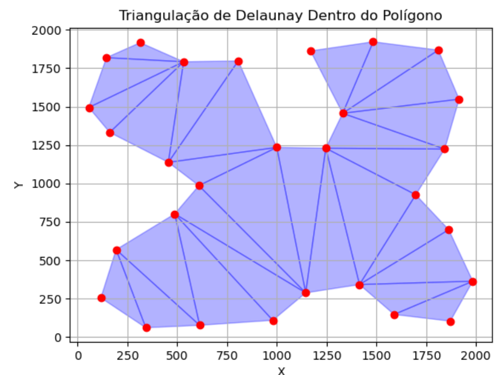
Figura 4.8: Triangulação de Delaunay aplicada aos pontos inseridos.

Porém, o produto final da triangulação não é completamente o desejado porque, como é realizada entre todos os pontos do polígono, existem zonas que não pertencem à área de interesse mas que também são consideradas. É necessário eliminar as zonas exteriores à área de interesse, como mostra a figura 4.9.

O último passo deste processo é juntar a triangulação de forma a criar zonas de pesquisa. O método estabelecido para realizar essa tarefa consiste em dividir a área no mesmo número de VANTs disponíveis, por exemplo, se $M = 2$ o polígono é dividido em 2 e se $M = 4$ o polígono divide-se em 4. A forma de juntar a triangulação corresponde a um conjunto de processos ordenados da seguinte forma: em primeiro lugar os triângulos são ordenados consoante a sua área em ordem decrescente e cada um dos que tiver maior área é alocado para um VANT, os triângulos ficam marcados como



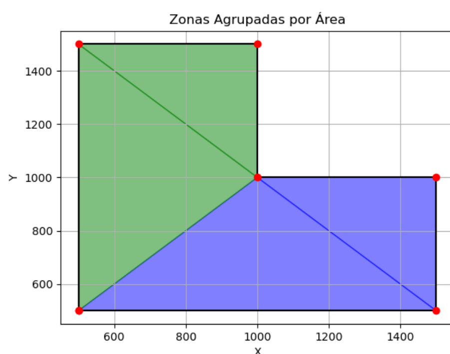
(a) Triangulação mais simples dentro da zona de em L.



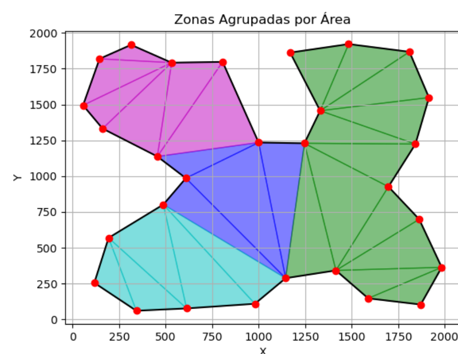
(b) Triangulação dentro da zona mais complexa.

Figura 4.9: Triangulação de Delaunay apenas considerada dentro da zona de interesse.

usados à medida que são alocados para os VANTs. Em seguida, enquanto a fila não estiver vazia, a função retira um triângulo da fila e tenta encontrar triângulos adjacentes a ele que ainda não foram alocados. Se um triângulo adjacente for encontrado, ele é adicionado à mesma zona, a sua área é contabilizada, e ele é marcado como usado. Esse novo triângulo adjacente também é adicionado à fila para expandir a zona em torno dele. Para cada triângulo que ainda não foi usado, a função identifica a zona com a menor área acumulada e aloca o triângulo a essa zona, equilibrando as áreas entre as zonas. Ao produto final do algoritmo, que vai resultar em várias zonas consoante a divisão feita, vai ser aplicado o método anterior de divisão em linhas paralelas, resultando então num polígono não convexo com zonas com direções de varrimento diferentes. Depois de aplicar o algoritmo aos polígonos não convexos mostrados nas figuras anteriores ficamos com os resultados das figuras 4.10(a) e 4.10(b).



(a) Divisão para $M = 2$.



(b) Divisão para $M = 4$.

Figura 4.10: Divisão da área em zonas de pesquisa consoante o número de VANTs.

5

Resultados

Conteúdo

5.1 Impacto das Restrições 35
5.2 Área Convexa 37
5.3 Área Não Convexa 40

Neste capítulo irão ser apresentados os resultados do problema de otimização e o comportamento da rota dos VANTs para uma área convexa e para uma área decomposta com triangulação de Delaunay.

Os testes foram realizados utilizando a versão Python 3.9 com um processador Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz e 16 GB RAM no sistema operativo Windows 11. O solucionador utilizado para o MILP foi o Gurobi Optimizer versão 11.0.2.

Tabela 5.1: Valores das constantes utilizadas para o problema.

t_s (min)	L_k (min)	V_{ij} [m/s]	res_h (pixel)	res_v (pixel)	hFOV (graus)	vFOV (graus)
2	30	45	8064	6048	65	49

A tabela 5.1 apresenta os valores base que foram utilzadas ao longo dos diferentes testes. As resolução horizontal e vertical, bem como o Campo de Visão (FOV) horizontal e vertical são utilizadas, em função da altura de voo, para calcular a largura e comprimento da imagem que o drone é capaz de captar.

5.1 Impacto das Restrições

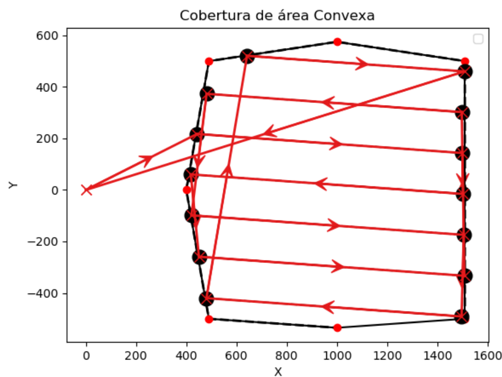
A tabela 5.2 mostra os parâmetros iniciais utilizados para cada um dos cenários da presente secção.

Tabela 5.2: Parâmetros iniciais definidos para cada cenário.

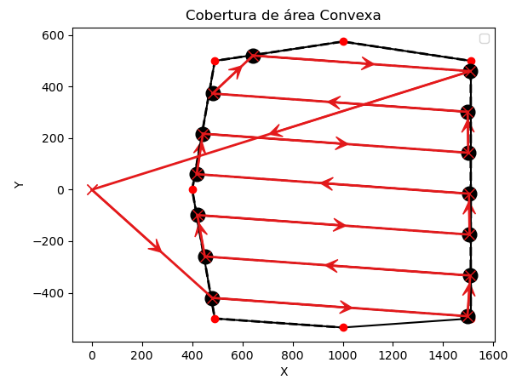
	M	O	Altitude de voo	Sobreposição
Restrição Tempo mínimo	1	1	150	0.1
Restrição Diagonais	1	1	200	0.1
Restrição Nó	3	3	250	0.1

No primeiro exemplo, queremos verificar a importância de utilizar a restrição da equação (4.6). Vamos verificar qual é o comportamento do VANT quando não se tem em conta o custo total da operação. É possível observar na imagem 5.1(a) que o VANT percorre caminhos desnecessários para a minimização do tempo de operação, com uma duração de 18 minutos e 10 segundos. Quando se tem em conta a relação entre o tempo da operação e o custo total, como mostra a equação (4.6), o trajeto do VANT já é otimizado e observa-se uma diminuição do tempo de operação para 15 minutos e 54 segundos.

No segundo caso, para uma área pentagonal, queremos demonstrar como as restrições das equações (4.13) e (4.14), referentes a evitar diagonais na rota, impactam o trajeto definido pelo VANT. Podemos observar pela figura 5.2(a) que o VANT executa um trajeto na diagonal com um tempo total de 11 minutos e 36 segundos, que apesar de não comprometer o sucesso da missão não é o ideal por motivos de captação de imagem. Ao adicionarmos as restrições, o trajeto passa a não executar nenhum movimento entre pontos que não sejam de vértices seguidos com uma duração de 11 minutos e 39



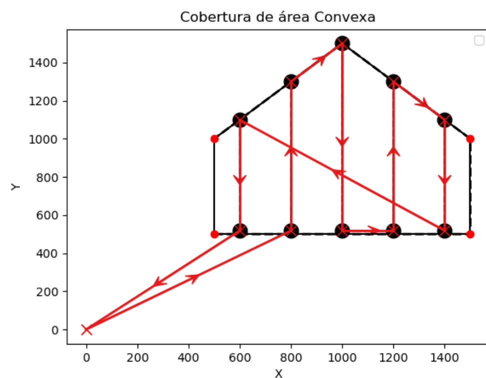
(a) Sem restrição do tempo mínimo da operação.



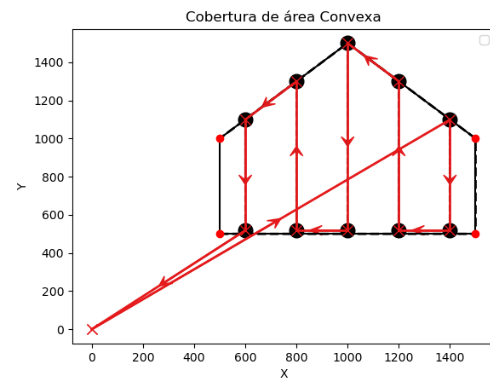
(b) Com restrição do tempo mínimo da operação.

Figura 5.1: Impacto da utilização de restrições de minimização temporal na solução do problema.

segundos, como se vê na figura 5.6(b). Apesar de neste caso o tempo aumentar 3 segundos depois de ser implementada a restrição, continua a ser ideal utilizá-la para manter todas as imagens com a mesma orientação.



(a) Sem restrições de evitar diagonais

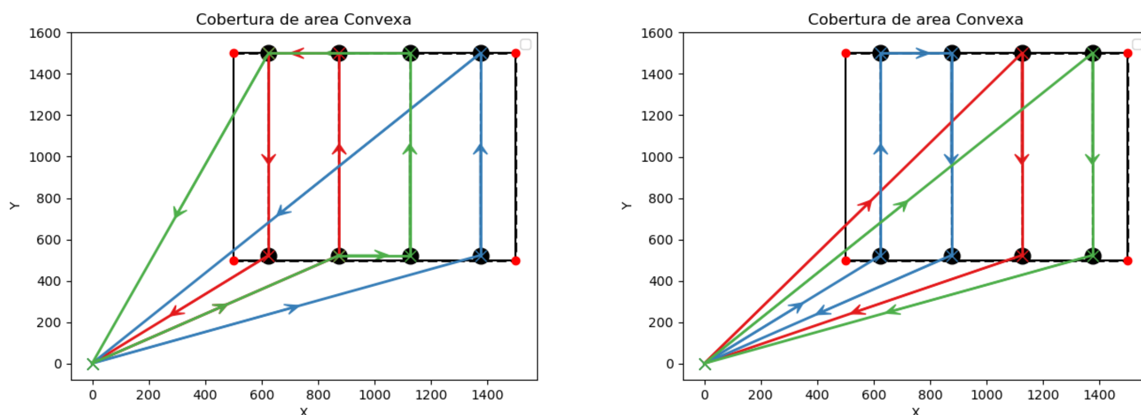


(b) Com restrição de evitar diagonais das equações (4.13) e (4.14).

Figura 5.2: Influência da aplicação da restrição para evitar diagonais.

Para verificar o efeito da equação (4.9), que obrigada cada nó a ser visitado por apenas 1 VANT, foi testado um cenário em que se utilizam quatro segmento de varrimento. Quando não se utiliza a restrição, o trajeto realizado pelo primeiro e terceiro VANTs parte do ponto inicial para o mesmo vértice, como se pode ver pela figura 5.3(a). Quando se implementa a restrição, já não se verificam vértices a serem visitados por mais do que um VANT e o trajeto fica completo com um tempo de 7 minutos e 58 segundos para ambos os casos. Apesar de o resultado temporal ser igual em ambos os casos, é

essencial aplicar a restrição para evitar colisões durante a missão.



(a) Sem restrição de um nó ser visitado por apenas um VANT.

(b) Com restrição de um nó ser visitado por apenas um único VANT.

Figura 5.3: Influência de implementar a restrição que obriga que cada nó seja visitado por apenas um VANT.

A tabela 5.3 apresenta os valores temporais obtidos para cada cenário testado nesta secção.

Tabela 5.3: Resultados temporais obtidos conforme a implementação das restrições.

	Restrição Tempo mínimo	Restrição Diagonais	Restrição Nó
Tempo sem (s)	1090	696	478
Tempo com (s)	954	699	478

5.2 Área Convexa

Os parâmetros utilizados nos próximos cenários são os que estão referênciados na tabela 5.4. Os que não são referidos não sofrem alterações e são os que estão referidos no início deste capítulo.

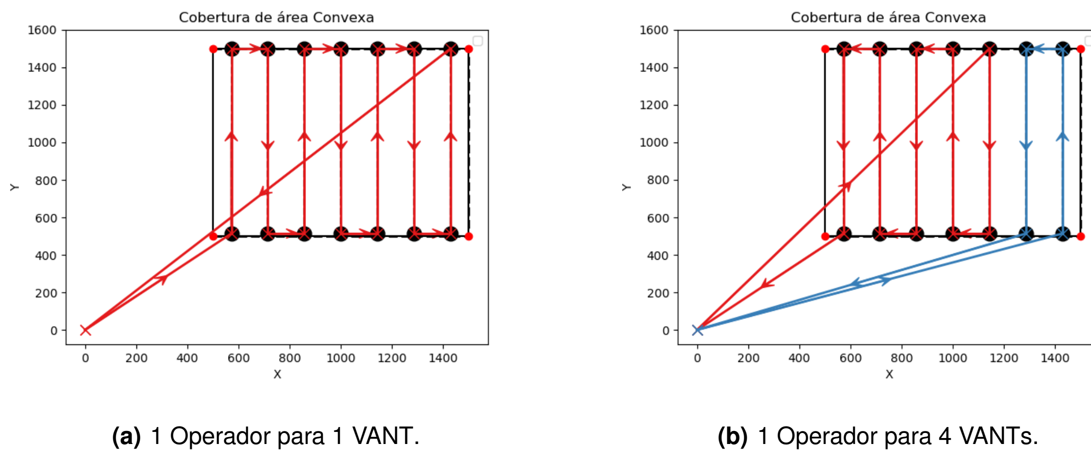
Tabela 5.4: Parâmetros iniciais para teste em área convexa.

		M	O	t_s	L_k	Altitude	Sobreposição
Cenário 1	Teste 1	1	1	4	30	150	0.2
	Teste 2	4					
Cenário 2		4	1	4	12	150	0.2
Cenário 3	Teste 1	4	2	4	30	150	0.2
	Teste 2	4	3				

Para verificar a aproximação deste método a um caso real são testados diferentes cenários como os que se vão ver a seguir.

No primeiro cenário vamos utilizar sempre a mesma zona quadrangular com uma área de 10^6 m^2 para analisar os diferentes resultados deste método. O tempo de setup de cada VANT foi estabelecido

como 4 minutos. Na figura 5.4(a) foi utilizado 1 VANT ($M = 1$) para realizar a pesquisa e teve a duração total de 18 minutos e 7 segundos. No teste da figura 5.4(b) o número de VANTs passou para 4 ($M = 4$) mas apenas havia 1 operador ($O = 1$). Como neste cenário o tempo de setup dos VANTs restantes, além do primeiro, é acumulativo, a solução obtida pelo otimizador foi de apenas utilizar 2 VANTs para resolver o problema. O tempo máximo foi de 14 minutos e 52 segundos.



(a) 1 Operador para 1 VANT.

(b) 1 Operador para 4 VANTs.

Figura 5.4: Influência do número de operadores para a utilização de VANTs no problema.

No segundo cenário, a única alteração efetuada nos parâmetros iniciais em relação ao cenário da figura 5.4(b) é que a autonomia dos VANTs foi diminuida para 12 minutos. É possível observar que o otimizador adicionou mais 1 VANT para solucionar o problema. Este VANT é adicionado porque a solução tem que continuar a obedecer à restrição da equação (4.8). Neste caso, o tempo máximo foi de 17 minutos e 59 segundos. Este aumento de tempo verifica-se porque como só há 1 operador, o terceiro VANT só iniciou a missão 12 minutos depois do início da mesma.

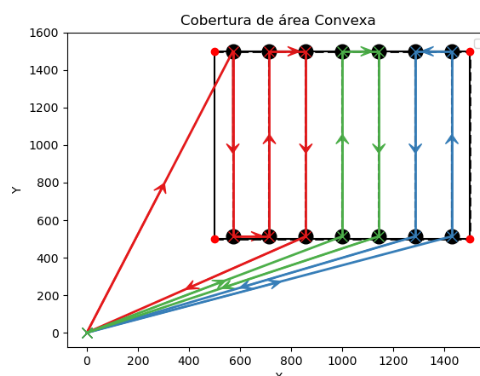
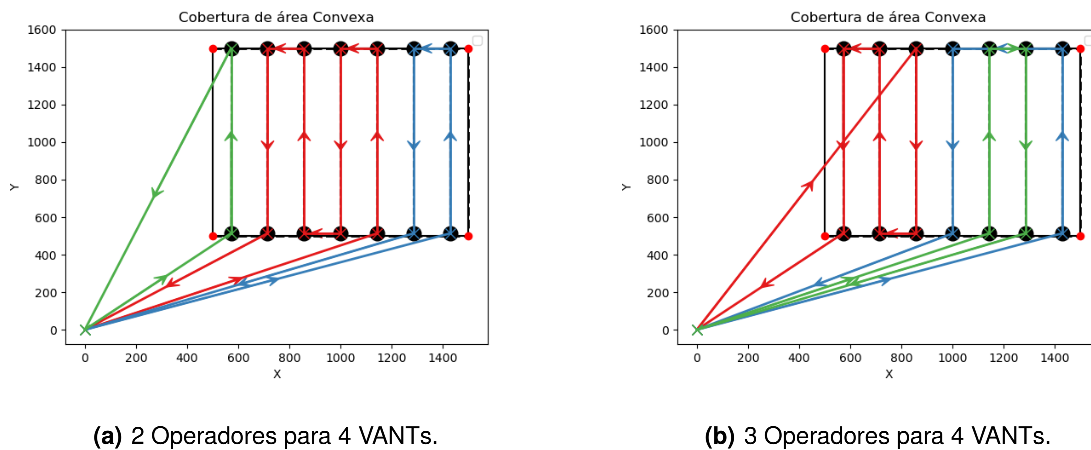


Figura 5.5: Diminuição da autonomia dos VANTs leva à implementação de mais um VANTs.

No terceiro cenário, visualizado na figura 5.6(a), é utilizada uma equipa de 4 VANTs ($M = 4$) que

são operados por 2 pessoas ($O = 2$). O tempo máximo de pesquisa obtido foi de 12 minutos e 40 segundos e dos 4 VANTs disponíveis apenas foram utilizados 3. Em seguida, ao adicionar mais um operador ($O = 3$), a melhor solução encontrada pelo otimizador continua a ser de utilizar 3 VANTs porém, é visivel que o 3º VANT faz o varrimento de 2 segmentos em vez de 1 como no caso anterior. O tempo máximo de pesquisa foi de 11 minutos e 39 segundos, referente ao cenário da figura 5.6.



(a) 2 Operadores para 4 VANTs.

(b) 3 Operadores para 4 VANTs.

Figura 5.6: Impacto do número de operadores na trajetória dos VANTs.

A tabela 5.5 mostra os valores, em segundos, obtidos para cada experiência e o número de VANTs, m, empenhado em cada missão.

Tabela 5.5: Valores temporais máximos e número de VANTs utilizados para cada cenário.

	Cenário 1		Cenário 2	Cenário 3	
	Teste 1	Teste 2		Teste 1	Teste 2
Tempo [s]	1087	892	1079	760	699
m	1	2	3	3	3

O custo, em tempo de processamento, que o otimizador Gurobi demora a efetuar todas as iterações até encontrar a melhor solução do problema, foi um dos fatores a ter em conta quando se selecionam alguns do parâmetros para o voo dos VANTs. O número de segmentos de varrimento tem impacto no tamanho do grafo G que é utilizado pelo otimizador. É possível observar pela figura 5.7 que até 6 segmentos de varrimento o otimizador demora menos do que 1 segundo a encontrar solução para o problema. Quando o problema passa para 7 segmentos há um aumento considerável para quase 9 segundos e para 8 segmentos aumenta ainda mais para 28 segundos. O maior aumento observa-se para 9 segmentos quando passa para 362 segundos (6 minutos e 2 segundos) de processamento até encontrar a solução. Para 10 segmentos não foi possível obter uma solução em tempo útil.

Figura 5.7: Tempo de processamento do otimizador para o diferente número de segmentos de varrimento.

5.3 Área Não Convexa

Para verificar se o método proposto produz os resultados desejados, é aplicada a mesma simulação Python que foi utilizada anteriormente. Vão ser apresentados os resultados do método com a triangulação de Delaunay para várias formas diferentes em comparação com o método para polígonos convexos. Apesar de uma forma triangular e quadrangular serem formas convexas, são apresentadas nesta secção para comparar o comportamento em ambos os métodos.

A tabela 5.6 mostra os parâmetros iniciais utilizados para cada uma das seguintes subsecções.

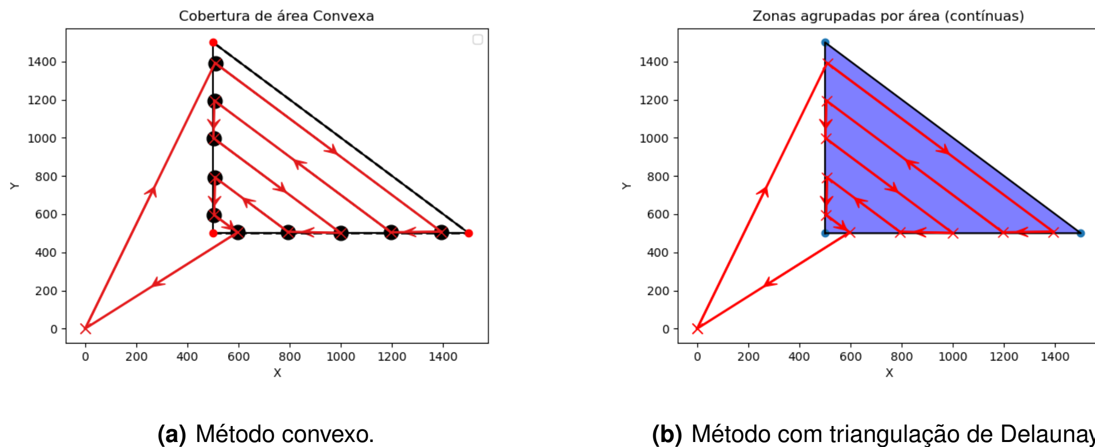
Tabela 5.6: Parâmetros iniciais utilizados para comparar ambos os métodos.

Forma Área	M	O	Altitude de Voo (m)	Sobreposição	t_s	L_k
Triangular	1	1	150	0.1	4	30
Quadrangular	2	2	150			
L	2	2	150			
Estrela 4 pontas	4	4	250			
C	3	3	200			
Pontos Manuais	4	4	250			

5.3.1 Área Triangular

No primeiro cenário vamos testar o comportamento de ambos os métodos para uma área triangular referente a um triângulo retângulo com $0,5 \times 10^6$ m^2. Neste caso, para a forma como o método com a triangulação de Delaunay está desenvolvido só é possível utilizar 1 VANT $(M = 1)$ e o respetivo operador $(O = 1)$. Para este caso, como seria de esperar por observação da figura 5.8, o tempo da missão é igual nos dois métodos porque depois de se aplicar a triangulação, a área total para analisar

é a mesma para ambos os casos e as variáveis utilizadas no processo de otimização são as mesmas. Ambos os métodos realizaram a missão em 12 minutos e 41 segundos.



(a) Método convexo.

(b) Método com triangulação de Delaunay.

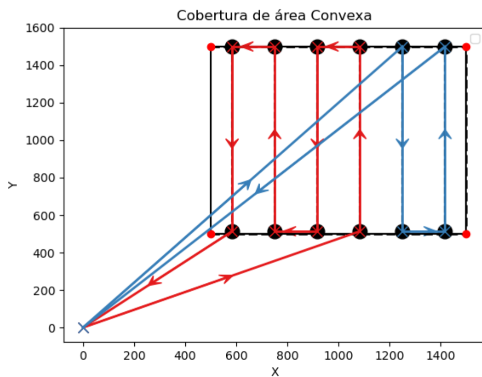
Figura 5.8: Resultado para área triangular.

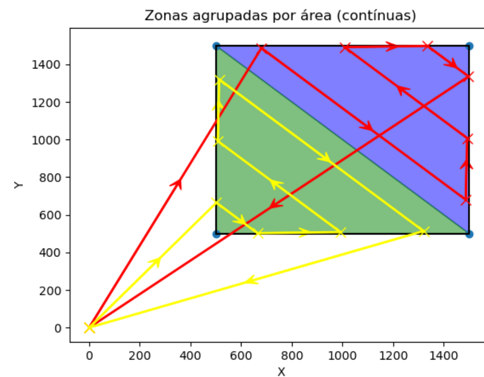
5.3.2 Área Quadrangular

Testar o método desenvolvido com a triangulação de Delaunay para uma área quadrangular é uma boa forma de perceber as principais diferenças para o método convexo.

O primeiro caso referente a uma área quadrangular de 10^6 m^2 com a utilização de 2 VANTs ($M = 2$) e 2 operadores ($O = 2$) em que o ponto de partida dos VANTs está nas coordenadas [0,0]. Para o método convexo o tempo máximo foi de 12 minutos e 33 segundos e para a triangulação de Delaunay foi de 10 minutos e 36 segundos. Como se pode observar nos trajetos da figura 5.9, o facto de o ponto de partida estar descentralizado teve influência no tempo do segundo VANT da figura 5.9(a) porque tem que percorrer uma distância maior.

Para tentar evitar diferenças nas condições de teste, foram realizados novos testes em que o ponto inicial dos VANTs coincide com a fronteira da área e com o centro dos segmentos de varrimento, ou seja, para o método convexo será em [1000, 500] e para o método com a triangulação de Delaunay será em [500, 1500]. Para este caso, o tempo para o convexo foi de 9 minutos e 54 segundos e para a triangulação de Delaunay foi de 9 minutos e 12 segundos. Ainda assim foi possível observar uma redução do tempo. Como se pode ver pela figura 5.10 o trajeto até à área de pesquisa já não teve influência no tempo de voo.

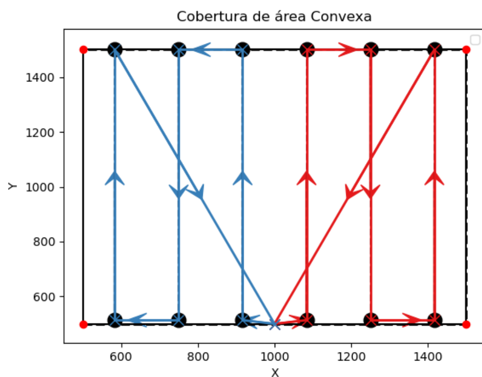


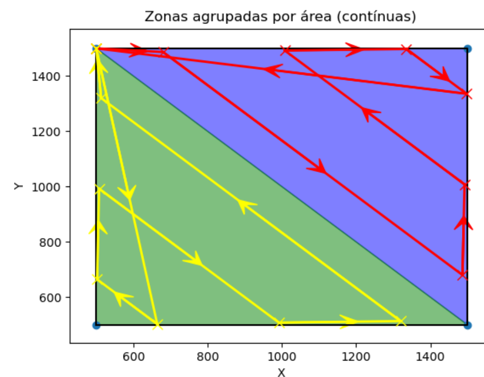


(a) $M = 2$ para método simples com ponto inicial em (0,0).

(b) $M = 2$ para triangulação de Delaunay com ponto inicial em (0,0).

Figura 5.9: Resultado para área quadrangular com ponto inicial em (0,0).





(a) Método simples para $M = 2$ com ponto inicial em (1000,500).

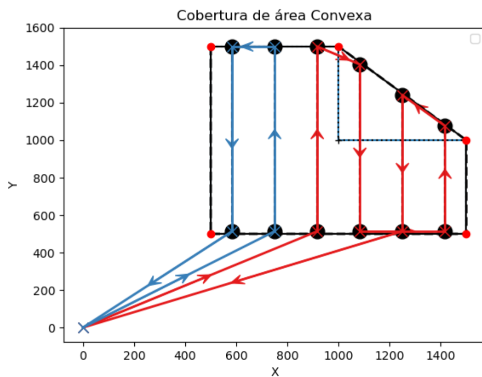
(b) Método com triangulação de Delaunay para $M = 2$ com ponto inicial em (500,1500).

Figura 5.10: Resultado para área quadrangular com ponto inicial coicidente com o centro equidistante para os 2 VANTs que vão realizar a pesquisa.

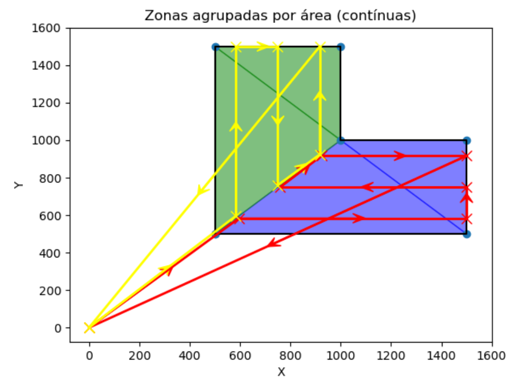
5.3.3 Área em L

Neste cenário, é comparado este novo método com o método convexo para uma área em L.

Quando se aplica a triangulação de Delaunay para realizar a decomposição de área, o tempo total de pesquisa diminui. A desvantagem de não realisar a decomposição de área é que há zonas que não são do nosso interesse mas que vão ser analisadas na mesma, neste caso houve 14.29% da área, fora da zona de interesse, que foi analisada. O tempo mínimo foi obtido quando se aplica triangulação e teve uma duração de 10 minutos e 37 segundos contra 11 minutos e 45 segundos.



(a) Solução para método simples com área convexa.

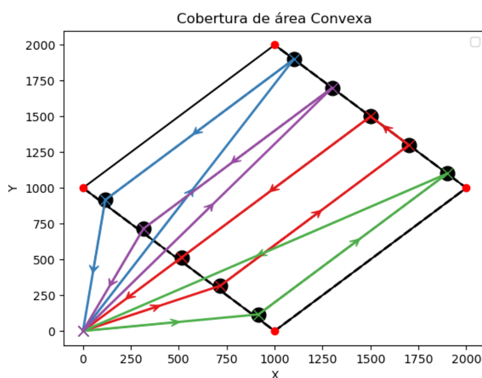


(b) Decomposição de área com triangulação.

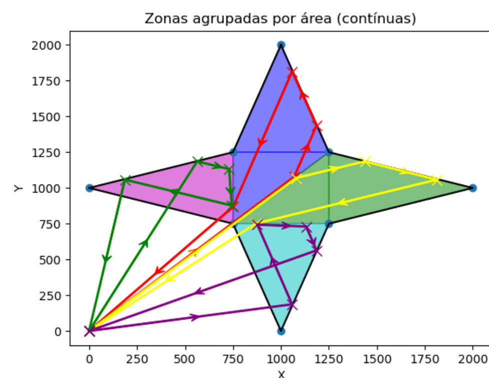
Figura 5.11: Comparação de resultados para área em ”L”.

5.3.4 Estrela de 4 pontas

Para realizar a cobertura de área para uma forma específica como uma estrela de 4 pontas, vamos utilizar uma frota de 4 VANTs ($M = 4$), e 4 operadores ($O = 4$). A altura de voo foi estabelecida em 250 metros. Para o método anterior com o polígono convexo, que neste caso considera um losango. O tempo máximo foi de 10 minutos e 6 segundos. Para a decomposição de área com a triangulação de Delaunay, o tempo máximo foi de 9 minutos e 54 segundos.



(a) Solução para área convexa de uma estrela de 4 pontas.

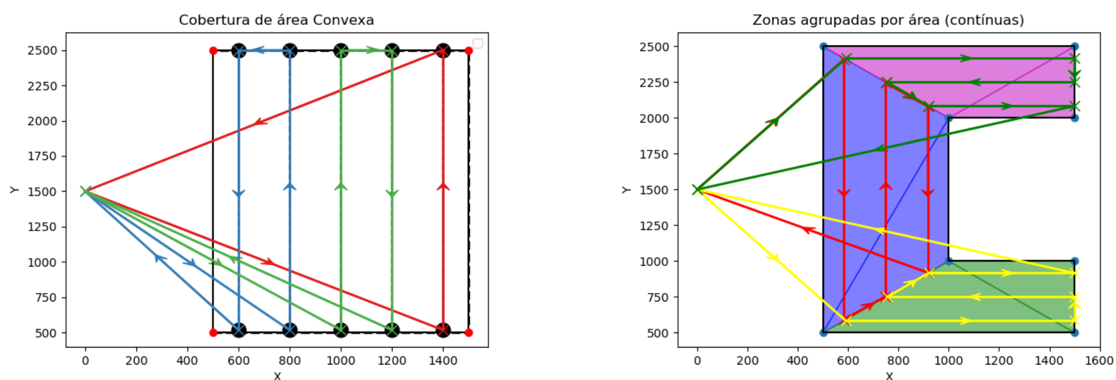


(b) Decomposição de área com triangulação para uma estrela de 4 pontas.

Figura 5.12: Comparação de resultados para ambos os métodos quando se utiliza uma área com a forma de uma estrela de 4 pontas.

5.3.5 Forma em C

Neste cenário, a forma côncava apresentada é semelhante a um ”C”. Neste caso, de forma a analisar apenas a área de interesse, é vantajoso realizar a decomposição de área na tentativa de obter melhores resultados. Foi utilizada uma frota com 3 VANTs ($M = 3$) e 3 operadores ($O = 3$) e o ponto inicial, em ambos os casos, foi estabelecido em (0, 1500). Para o primeiro método com a área convexa, o tempo máximo foi de 13 minutos e 29 segundos e para a decomposição de área com triangulação o tempo máximo foi de 13 minutos e 27 segundos. O trajeto escolhido pelos VANTs foi o demonstrado nas figuras 5.13(a) e 5.13(b).



(a) Solução para a área convexa com $M = 3$.

(b) Decomposição de área com triangulação para $M = 3$.

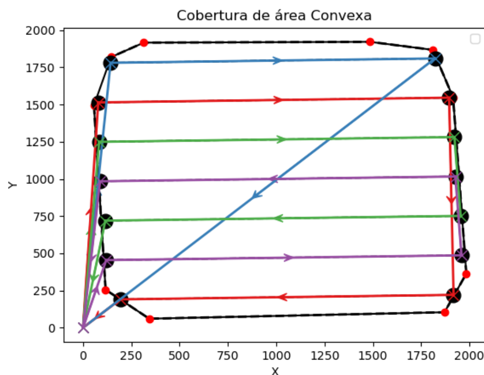
Figura 5.13: Comparação de resultados para ambos os métodos para uma área com forma de ”C”.

5.3.6 Pontos inseridos manualmente

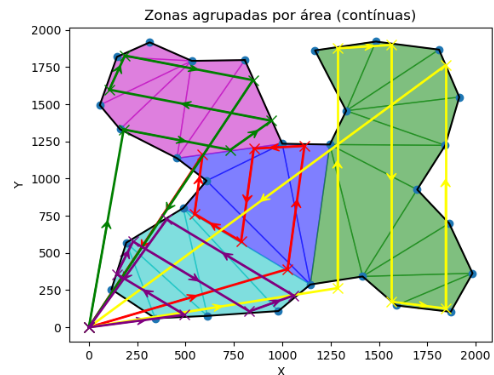
No próximo caso, a área a analisar é a que foi exemplificada na secção anterior na figura 4.10. Estando a área definida vamos voltar a analisar os resultados para os dois métodos utilizados.

Neste cenário, em ambos os casos, foram utilizados 4 VANTs ($M = 4$) e cada um com o seu operador ($O = 4$). Os pontos foram inseridos manualmente com o uso da função ginput [41]. Ao utilizar o método simples para uma área convexa, como se pode ver na figura 5.14(a), o tempo máximo foi de 12 minutos e 52 segundos. Quando se aplica a triangulação de Delaunay para decompor a área, observou-se um incremento do tempo para 16 minutos e 30 segundos.

A análise deste resultado leva-nos a identificar um problema de juntar a área para cada VANT ao utilizar o método da triangulação de Delaunay. Como se pode ver na tabela 5.7, que mostra os tempos de cada VANT, a discrepância entre o tempo que cada um demora a analisar a área é grande quando se utiliza a triangulação de Delaunay, enquanto que para a divisão mais simples o tempo é



(a) Solução para o algoritmo do capitulo 3.



(b) Decomposição de área com triangulação.

Figura 5.14: Comparação de resultados para uma área inicial côncava com mais variações.

muito semelhante. Isto deve-se ao facto de para a área convexa o otimizador recebe o input que tem 4 VANTs e minimiza o tempo. Para a triangulação de Delaunay, o otimizador recebe 1 VANT para cada área, ou seja, cada área é independente e não está interligada. Uma possível solução seria discretizar cada segmento de varrimento e interligar o trajeto dos VANTs.

Tabela 5.7: Resultados comparativos entre o tempo de cada VANT para os dois métodos.

	Convexo	Delaunay
VANT 1	772	626
VANT 2	722	990
VANT 3	735	708
VANT 4	693	548

A tabela 5.8 apresenta os resultados temporais obtidos para todos os testes da última secção. É possível observar que, em maior parte dos casos, a cobertura de área realizada com triangulação de Delaunay obtém valores temporais inferiores do que a metodologia para uma área convexa.

Tabela 5.8: Resultados temporais, em segundos, obtidos comparando os dois métodos para os vários cenários.

Forma Área		Convexo	Delaunay
Triangular		761	761
Quadrangular	Teste 1	753	636
	Teste 2	594	552
L		705	637
Estrela 4 pontas		606	594
C		809	807
Pontos Manuais		772	990

6

Conclusão

Conteúdo

6.1 Conclusões Finais 49

6.2 Trabalho Futuro e Limitações 50

Esta dissertação tem como objetivo aplicar um método otimizado para realizar a cobertura de uma área de pesquisa com o auxílio de um enxame de VANTs. A cobertura de área é uma tarefa essencial para o comando e controlo e dotar os comandantes de informações os auxiliam no processo de tomada de decisão.

Este capítulo sintetiza as conclusões obtidas ao longo da realização da dissertação e os objetivos que foram alcançados. Apresentam-se também perspetivas de trabalho futuro.

6.1 Conclusões Finais

Nas últimas decadas, a utilização de VANTs tem vindo a ganhar preponderância, uma vez que a utilização dos mesmos permite realizar tarefas, que se fossem efetuadas por humanos, iriam necessitar de muitos mais recursos, despender de mais tempo e permitem alcançar locais de acesso reduzido. A combinação de vários VANTs na realização das missões aumenta a eficiência e diminui o tempo para concluir a tarefa.

Uma decomposição de área eficiente pode potênciar a missão dos VANTs e pode também garantir que o varrimento é efetuado na totalidade e com a menor redundância possível para poupar tempo. Desta forma, a decomposição de área para polígonos convexos foi feita através de decomposição linear paralela, que divide o polígono em segmentos de varrimento paralelos entre si, enquanto que para polígonos não convexos, a área foi decomposta através da triangulação de Delaunay e posteriormente, a cada célula nova gerada pela triangulação, foi aplicado o mesmo método de decomposição linear paralela.

Para uma área convexa foi analisado como os parâmetros iniciais do cenário influenciam o resultado final, como o número de VANTs disponíveis e o número de operadores. O número de operadores tem influência no número de VANTs que vão ser acionados, em alguns casos é melhor aplicar menos VANTs do que empenhar todos. Por exemplo numa OBS num terreno remoto, em que é necessário deslocar os VANTs até lá, se só existir um operador, a solução final pode ser otimizada para dois VANTs em vez de três.

Para uma área não convexa foi aplicada a triangulação de Delaunay para realizar a decomposição de área. Este método foi desenvolvido porque em alguns casos, com a área convexa, são analisadas zonas fora da área de interesse. Um comportamento observado, usando este método em comparação aos resultados para um polígono convexo, foi que os VANTs podem efetuar rotas com direções diferentes entre células criadas pela triangulação, enquanto que para uma área convexa só existe uma direção de varrimento.

A realização da presente dissertação permitiu a aquisição e consolidação de conhecimentos relativamente à formulação de problemas de otimização e a métodos para decomposição e classificação de

uma área ou polígono. Houve também uma aquisição de conhecimentos relativamente à programação na linguagem Python e software de simulação, que apesar de não terem sido apresentados no trabalho foram testados durante a participação no exercício ARTEX, em Santa Margarida, para tentar incorporar na dissertação, como o QGroundControl e o MissionPlanner.

6.2 Trabalho Futuro e Limitações

Foi verificado durante a análise de resultados, que para a maior parte dos polígonos, o método aplicado para uma área não convexa, através da triangulação de Delaunay, obteve melhores resultados. No entanto, foi identificada uma fragilidade para áreas mais complexas porque ao ser feita a divisão da área, cada novo polígono fica atribuido a um só VANT fazendo assim com que haja uma discrepância nos tempos de voo de cada VANT se existirem percursos muito diferentes. Uma sugestão para mitigar este problema seria discretizar os segmentos de varrimento e aplicar o problema de otimização a esses pontos, podendo atribuir o mesmo segmento a VANT diferentes, por exemplo. Outra recomendação seria, quando se aplica a triangulação de Delaunay, acrescentar pontos para aumentar a triangulação, de forma a que cada triângulo corresponda à imagem captada pela câmara do VANT. Em seguida pode-se aplicar o problema de otimização para ser definida a rota do VANT e tentar outros padrões de pesquisa.

Bibliografia

[1] F. Remondino, L. Barazzetti, F. Nex, M. Scaioni, and D. Sarazzi, "UAV photogrammetry for mapping and 3D modeling: Current status and future perspectives," in *Proceedings of the International Conference on Unmanned Aerial Vehicle in Geomatics (UAV-g): 14-16 September 2011, Zurich, Switzerland*. International Society for Photogrammetry and Remote Sensing (ISPRS), 2011, pp. 25–31.

[2] M. Campion, P. Ranganathan, and S. Faruque, "A review and future directions of UAV swarm communication architectures," in *2018 IEEE international conference on electro/information technology (EIT)*. IEEE, 2018, pp. 0903–0908.

[3] B. Fan, Y. Li, R. Zhang, and Q. Fu, "Review on the technological development and application of UAV systems," *Chinese Journal of Electronics*, vol. 29, no. 2, pp. 199–207, 2020.

[4] B. Siemiatkowska and W. Stecz, "A framework for planning and execution of drone swarm missions in a hostile environment," *Sensors*, vol. 21, no. 12, p. 4150, 2021.

[5] G. Wang, W. Yao, X. Zhang, and Z. Li, "A mean-field game control for large-scale swarm formation flight in dense environments," *Sensors*, vol. 22, no. 14, p. 5437, 2022.

[6] A. Al-Naji, A. G. Perera, S. L. Mohammed, and J. Chahl, "Life signs detector using a drone in disaster zones," *Remote Sensing*, vol. 11, no. 20, p. 2441, 2019.

[7] L. M. Gladence, V. M. Anu, A. Anderson, I. Stanley, S. Revathy *et al.*, "Swarm intelligence in disaster recovery," in *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 2021, pp. 1–8.

[8] V. Subbarayalu and M. A. Vensuslaus, "An intrusion detection system for drone swarming utilizing timed probabilistic automata," *Drones*, vol. 7, no. 4, p. 248, 2023.

[9] S.-W. Cho, J.-H. Park, H.-J. Park, and S. Kim, "Multi-UAV coverage path planning based on hexagonal grid decomposition in maritime search and rescue," *Mathematics*, vol. 10, no. 1, p. 83, 2021.

[10] T. M. Cabreira, L. B. Brisolara, and F. J. Paulo R, “Survey on coverage path planning with unmanned aerial vehicles,” *Drones*, vol. 3, no. 1, p. 4, 2019.

[11] Gurobi Optimization, LLC. (2024) Python API Overview. Accessed: 2024-5-13. [Online]. Available: https://www.gurobi.com/documentation/11.0/refman/py_python_api_overview.html#sec:Python

[12] M. Campion, P. Ranganathan, and S. Faruque, “UAV Swarm Communication and Control Architectures: a Review,” *Journal of Unmanned Vehicle Systems*, vol. 7, no. 2, pp. 93–106, 2018.

[13] I. Bekmezci, O. K. Sahingoz, and Ş. Temel, “Flying Ad-Hoc Networks (FANETs): A Survey,” *Ad Hoc Networks*, vol. 11, no. 3, pp. 1254–1270, 2013.

[14] O. K. Sahingoz, “Networking Models in Flying Ad-Hoc Networks (FANETs): Concepts and Challenges,” *Journal of Intelligent & Robotic Systems*, vol. 74, pp. 513–527, 2014.

[15] J. F. Araujo, P. Sujit, and J. B. Sousa, “Multiple UAV area decomposition and coverage,” in *2013 IEEE symposium on computational intelligence for security and defense applications (CISDA)*. IEEE, 2013, pp. 30–37.

[16] H. Choset, “Coverage for robotics–a survey of recent results,” *Annals of mathematics and artificial intelligence*, vol. 31, pp. 113–126, 2001.

[17] G. Öst, “Search path generation with UAV applications using approximate convex decomposition,” Master’s thesis, Universidade de Linköping, Suécia, 2012.

[18] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95-international conference on neural networks*, vol. 4, 1995, pp. 1942–1948.

[19] M. Dorigo and L. M. Gambardella, “Ant colonies for the travelling salesman problem,” *biosystems*, vol. 43, no. 2, pp. 73–81, 1997.

[20] M. Brand, M. Masuda, N. Wehner, and X.-H. Yu, “Ant colony optimization algorithm for robot path planning,” in *2010 international conference on computer design and applications*, vol. 3. IEEE, 2010, pp. V3–436.

[21] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[22] N. Christofides, “The vehicle routing problem,” *Revue française d’automatique, informatique, recherche opérationnelle. Recherche opérationnelle*, vol. 10, no. V1, pp. 55–70, 1976.

[23] J. R. Montoya-Torres, J. L. Franco, S. N. Isaza, H. F. Jiménez, and N. Herazo-Padilla, “A literature review on the vehicle routing problem with multiple depots,” *Computers & Industrial Engineering*, vol. 79, pp. 115–129, 2015.

[24] F. Saffre, H. Hildmann, H. Karvonen, and T. Lind, "Monitoring and cordoning wildfires with an autonomous swarm of unmanned aerial vehicles," *Drones*, vol. 6, no. 10, p. 301, 2022.

[25] R. Arranz, D. Carramiñana, G. d. Miguel, J. A. Besada, and A. M. Bernardos, "Application of Deep Reinforcement Learning to UAV Swarming for Ground Surveillance," *Sensors*, vol. 23, no. 21, 2023.

[26] X. Fan, H. Li, Y. Chen, and D. Dong, "UAV Swarm Search Path Planning Method Based on Probability of Containment," *Drones*, vol. 8, no. 4, p. 132, 2024.

[27] J. Kim, H. Oh, B. Yu, and S. Kim, "Optimal Task Assignment for UAV Swarm Operations in Hostile Environments," *International Journal of Aeronautical and Space Sciences*, vol. 22, pp. 456–467, 2021.

[28] R. Cheng and Y. Jin, "A social learning particle swarm optimization algorithm for scalable optimization," *Information Sciences*, vol. 291, pp. 43–60, 2015.

[29] G. Skorobogatov, C. Barrado, E. Salamí, and E. Pastor, "Flight planning in multi-unmanned aerial vehicle systems: Nonconvex polygon area decomposition and trajectory assignment," *International Journal of Advanced Robotic Systems*, vol. 18, no. 1, p. 1729881421989551, 2021.

[30] M. A. Luna, M. S. Isaac, A. R. Ragab, P. Campoy, P. Flores Peña, and M. Molina, "Fast multi-UAV path planning for optimal area coverage in aerial sensing applications," *Sensors*, vol. 22, no. 6, p. 2297, 2022.

[31] D. Albani, D. Nardi, and V. Trianni, "Field coverage and weed mapping by UAV swarms," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 4319–4325.

[32] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied mathematics and computation*, vol. 214, no. 1, pp. 108–132, 2009.

[33] J. Araújo, P. Sujit, and J. Sousa, "Multiple UAV area decomposition and coverage," in *2013 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2013, pp. 30–37.

[34] G. Pellegrino, G. Mota, F. Assis, S. Gorender, and A. Sá, "Simple area coverage by a dynamic set of unmanned aerial vehicles," in *2020 X Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, 2020, pp. 1–8.

[35] F. Balampanis, I. Maza, and A. Ollero, "Coastal areas division and coverage with multiple UAVs for remote sensing," *Sensors*, vol. 17, no. 4, p. 808, 2017.

[36] M. Szklany, A. Cohen, and J. Boubin, “Tsunami: Scalable, Fault Tolerant Coverage Path Planning for UAV Swarms,” in *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2024, pp. 711–717.

[37] G. S. Avellar, G. A. Pereira, L. C. Pimenta, and P. Iscold, “Multi-UAV routing for area coverage and remote sensing with minimum time,” *Sensors*, vol. 15, no. 11, pp. 27 783–27 803, 2015.

[38] W. H. Huang, “Optimal line-sweep-based decompositions for coverage algorithms,” in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, vol. 1. IEEE, 2001, pp. 27–32.

[39] O. R. Musin, “Properties of the delaunay triangulation,” in *Proceedings of the thirteenth annual symposium on Computational geometry*, 1997, pp. 424–426.

[40] SciPy Developers. (2024) scipy.spatial.delaunay. Accessed: 2024-04-28. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Delaunay.html

[41] Matplotlib Developers. (2024) matplotlib.pyplot.ginput. Accessed: 2024-04-08. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.ginput.html

[42] S. Veena, S. Vaitheeswaran, and H. Lokesha, “Towards the development of secure mavs,” in *ICRAMAV-2014 (3rd International Conference)*, 2014.

[43] N. Butcher, A. Stewart, and S. Biaz, “Securing the mavlink communication protocol for unmanned aircraft systems,” *Appalachian State University, Auburn University, USA*, 2013.

[44] S. Atoev, K.-R. Kwon, S.-H. Lee, and K.-S. Moon, “Data analysis of the MAVLink communication protocol,” in *2017 International Conference on Information Science and Communications Technologies (ICISCT)*. IEEE, 2017, pp. 1–3.

Conceitos teóricos complementares

A.1 MAVLink

O protocolo Micro Air Vehicle Link (MAVLink) permite que exista a comunicação entre entidades por um canal sem fios. Quando é implementado em VANTs, utiliza comunicação bidirecional entre o VANT e a ECT. O VANT é responsável por enviar dados telemétricos e informações enquanto que a ECT controla e envia comandos para o VANT [42]. As mensagens MAVLink transportam os comandos transmitidos pela ECT para o VANT e fornecem feedback (por exemplo, telemetria e estado do sistema) do VANT para a ECT, permitindo ao piloto manter o controlo da aeronave. Estas mensagens são enviadas como pacotes de dados entre a estação terrestre e o piloto automático para pilotar corretamente o VANT [43]. Uma mensagem MAVLink é enviada por via eletrónica através do canal de comunicação, seguida de uma *checksum* para correção de erros. Se a *checksum* não corresponder, significa que a mensagem está corrompida e será rejeitada. O MAVLink utiliza um sinal de início de pacote (STX) para sincronizar o início de uma mensagem codificada. Quando o sinal de início do pacote é recebido, o comprimento do pacote (n) é lido e, após n bytes, a soma de controlo (CKA e CKB) é verificada. Se a *checksum* corresponder, o pacote descodificado é processado, é transmitida uma mensagem ACK e aguarda-se o

próximo sinal de início. Bytes de mensagem alterados ou perdidos resultarão numa falha na *checksum*, fazendo com que o pacote seja descartado, e o dispositivo recetor retoma a escuta do próximo pacote de sinal de início. O MAVLink utiliza um número de sequência (SEQ) para cada pacote como um mecanismo de segurança para monitorizar a deteção de perda de pacotes. Se a taxa de perda de pacotes parecer significativa, o piloto comandará o VANT para voltar ao ponto inicial ou, pelo menos, reduzir o alcance operacional [44].

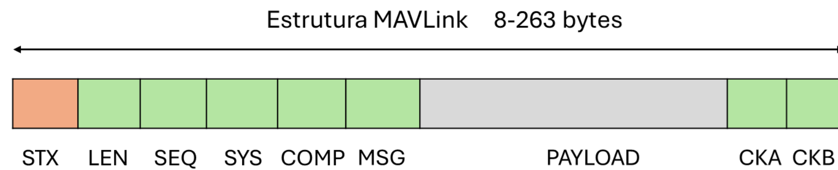


Figura A.1: Estrutura do protocolo MAVLink.

B

Código do Projeto

Neste anexo B está esplanado o código que foi utilizado para a validação dos algoritmos.

Listagem B.1: Script principal tese.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull
from Strips import findStrips, plotStrips
from Waypoints import findWaypoints
from cost import lip3
from Time import time
from Plot_Path import plot_uav_path


if __name__ == "__main__":

    # Configuracoes do UAV
    uavNumber = 3
    uavSetupTime = 4
    uavFlightTime = 30
    uavSpeed = 45 #(m/s)
    flightAltitude = 200
    O = 4
    h = 1

```

```
22      horizontalResolution = 8064
23      verticalResolution = 6048
24      hfieldOfView = 65
25      vfieldOfView = 49
26
27      sidelap = 0.1
28
29      uav = np.ones((uavNumber, 1))
30
31      # Ajustar o valor se houver apenas um UAV
32      if uavNumber == 1:
33          uav = np.array([[1], [0]])
34
35      imageWidth = flightAltitude * 2 * np.tan(np.radians(hfieldOfView / 2))
36      imageLength = flightAltitude * 2 * np.tan(np.radians(vfieldOfView / 2))
37      print("imageWidth: ", imageWidth)
38      print("imageLength: ", imageLength)
39
40
41      fig, ax = plt.subplots()
42      plt.title('Cobertura de area Convexa')
43      plt.xlabel('X')
44      plt.ylabel('Y')
45
46
47      # Coordenadas definidas manualmente
48      # coordenadas_x = [500, 100, 100, 500, 300]
49      # coordenadas_y = [100, 100, 500, 500,800]
50
51      # coordenadas_x = [500, 500, 1500, 1500, 1000, 1300, 700, 800]
52      # coordenadas_y = [500, 1000, 500, 1000,1400, 400, 400, 1400]
53
54      # coordenadas_x = [1000, 750, 0, 750,1000, 1250,2000,1250]
55      # coordenadas_y = [2000, 1250, 1000,750, 0,750,1000,1250]
56
57      coordenadas_x = [500, 1500,1500,1000,1000,1500,1500,500]
58      coordenadas_y = [500, 500, 1000,1000,2000,2000,2500,2500]
59
60      # coordenadas_x = [500, 500, 1500]
61      # coordenadas_y = [500, 1500, 500]
62
63      # coordenadas_x = [490, 400, 490, 1000, 1510, 1400, 1510, 1000]
64      # coordenadas_y = [500, 0, -500, -535, -500, 0, 500, 575]
65
66      # coordenadas_x = [500, 500, 1500, 1500]
67      # coordenadas_y = [500, 1500, 500, 1500]
68
69      # coordenadas_x = [-250, 250, 400, 0,-400]
70      # coordenadas_y = [500, 500, 1100, 1500,1100]
71
72      # coordenadas_x = [500, 500, 1000, 1000, 1500, 1500]
73      # coordenadas_y = [500, 1500, 1500, 1000, 1000, 500]
74
75      # coordenadas_x = [50, 50, 150, 150, 100, 100]
76      # coordenadas_y = [-500, 500, -500, 500, 800, -800]
77
78      # coordenadas_x = [490, 1510, 1510, 490]
79      # coordenadas_y = [-1000, -1000, 1000, 1000]
80
```

```
81     # coordenadas_x = [806.4516129, 532.25806452, 314.51612903, 145.16129032,
        56.4516129, 161.29032258, 455.64516129, 608.87096774, 487.90322581,
       193.5483871, 116.93548387, 342.74193548, 612.90322581, 979.83870968,
       1145.16129032, 1415.32258065, 1588.70967742, 1870.96774194,
       1983.87096774, 1862.90322581, 1697.58064516, 1842.74193548,
       1915.32258065, 1810.48387097, 1483.87096774, 1169.35483871,
       1330.64516129, 1245.96774194, 1000.]
82     # coordenadas_y = [1797.61904762, 1792.20779221, 1916.66666667,
       1819.26406926, 1494.58874459, 1332.25108225, 1137.44588745,
       985.93073593, 801.94805195, 569.26406926, 255.41125541, 60.60606061,
       76.83982684, 109.30735931, 287.87878788, 341.99134199, 147.18614719,
       103.8961039, 363.63636364, 699.13419913, 926.40692641, 1224.02597403,
        1548.7012987, 1867.96536797, 1922.07792208, 1862.55411255,
       1456.70995671, 1229.43722944, 1234.84848485]
83
84     # coordenadas_x = [500, 2000, 2000, 500]
85     # coordenadas_y = [-500, -500, 500, 500]
86
87     # coordenadas_x = [490, 1510, 1310, 490,700,400]
88     # coordenadas_y = [-1000, -1000, 800, 1000,1200,600]
89
90     # coordenadas_x = [490, 1650, 1310, 490,700,400]
91     # coordenadas_y = [-1000, -1000, 800, 1000,1200,600]
92
93     # Converter as coordenadas em um array numpy
94     points = np.array(list(zip(coordenadas_x, coordenadas_y)))
95
96
97     # Exibir as coordenadas
98     print("Pontos definidos manualmente:", points)
99
100    # Desenhar o Convex Hull
101    hull = ConvexHull(points)
102    for simplex in hull.simplices:
103        ax.plot(points[simplex, 0], points[simplex, 1], 'k-')
104
105
106
107    start_point = [0, 1500]
108    lmin, lmax, V, laneDist = findStrips(coordenadas_x, coordenadas_y,
           sidelap, imageWidth, imageLength,start_point)
109    plotStrips(lmin, lmax)
110
111
112    X, v, m = lip3(V, uav, uavSpeed, uavSetupTime, uavFlightTime, h, 0)
113
114    # Verificando o valor de m
115    if m > 1 and h == 0:
116        while m > 1:
117            uavMaxTimeIndex = np.argmax(v)
118            uav[uavMaxTimeIndex] = 0
119            if isinstance(v, list):
120                for i in range(len(v)):
121                    if v[i] == 0:
122                        uav[i] = 0
123            else:
124                if v == 0:
125                    uav[:] = 0  # Define todos os valores de uav como 0
```

```
126             waypoints = findWaypoints(X[:, :, uavMaxTimeIndex], V,start_point
                )
127             print("m1: ", m)
128             visitedWaypoints, order = np.max(X[:, :, uavMaxTimeIndex], axis=0
                ), np.argmax(X[:, :, uavMaxTimeIndex], axis=0)
129             newV = []
130             j = 0
131             for i in range(len(V)):
132                 if i == 0:
133                     newV.append([0, 0])
134                     j += 1
135                 elif visitedWaypoints[i] == 0:
136                     newV.append(V[i])
137                     j += 1
138             print("V ANTES: ", V)
139             V = np.array(newV)
140             print("V DEPOIS: ", V)
141             X, v, m = lip3(V, uav, uavSpeed, uavSetupTime, uavFlightTime, h,
                0)
142             print("m2: ", m)
143 
144         uavMaxTimeIndex = np.argmax(v)
145         uav[uavMaxTimeIndex] = 0
146         if isinstance(v, list):
147             for i in range(len(v)):
148                 if v[i] == 0:
149                     uav[i] = 0
150         else:
151             if v == 0:
152                 uav[:] = 0  # Define todos os valores de uav como 0
153         waypoints.append(findWaypoints(X[:, :, uavMaxTimeIndex], V,
               start_point))
154     else:
155         waypoints = []
156         for i in range(int(m)):
157             waypoints.append(findWaypoints(X[:, :, i], V,start_point))
158 
159     # Calcular tempos
160     print("WAYPOINTS: ", waypoints)
161     t, t_voo = time(waypoints, uavSpeed, uavSetupTime, 0)
162     print("laneDist: ", laneDist)
163     print("Tempos totais:", t)
164     print("Tempos de voo:", t_voo)
165 
166     # Plotar caminho dos UAVs
167     plot_uav_path(waypoints)
168 
169     plt.legend()
170     plt.show()
```

Listagem B.2: Script para calcular a divisão de área por linhas paralelas Strips.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.spatial import ConvexHull
4 from matplotlib.path import Path
5 
6 def inpolygon(x, y, xv, yv):
7     path = Path(np.column_stack((xv, yv)))
```

```
8       return path.contains_points(np.column_stack((x, y)))
9
10
11  def findStrips(x, y, sidelap, imageWidth, imageLength,start_point):
12      x = np.array(x)
13      y = np.array(y)
14      # Calcula o casco convexo e a regiao a ser pesquisada
15      points = np.vstack((x, y)).T
16      hull = ConvexHull(points)
17      k = hull.vertices
18      print('Convex hull indices:', k)
19
20      # Plot a regiao e o casco convexo
21      #plt.plot(np.append(x, x[0]), np.append(y, y[0]), 'k+:')
22      plt.plot(points[hull.vertices, 0], points[hull.vertices, 1], 'k--', lw=2)
23      plt.plot(points[hull.vertices, 0], points[hull.vertices, 1], 'ro')
24      plt.show()
25
26      # Apenas os pontos do casco convexo interessam
27      x = x[k]
28      y = y[k]
29      print('Convex hull vertices:')
30      print('x:', x)
31      print('y:', y)
32
33      # Rodar a regiao de interesse para encontrar o angulo de
34      # varrimento que utiliza o menor numero de linhas
35      areaWidth = np.max(x) - np.min(x)
36      thetamin = 0
37      print('Largura de area inicial: ', areaWidth)
38
39
40      for i in range(360):
41          theta = i * 2 * np.pi / 360
42          R = np.array([[np.cos(theta), -np.sin(theta)],
43                        [np.sin(theta), np.cos(theta)]])
44          aux = R @ np.vstack((x, y))
45          if (np.max(aux[0, :]) - np.min(aux[0, :])) < areaWidth:
46              areaWidth = np.max(aux[0, :]) - np.min(aux[0, :])
47              thetamin = theta
48
49      print('Angulo de rotacao otimo: ',thetamin)
50
51      # Rodar a regiao para o angulo escolhido para facilitar os calculos
            subsequentes
52      R = np.array([[np.cos(thetamin), -np.sin(thetamin)],
53                    [np.sin(thetamin), np.cos(thetamin)]])
54      aux = R @ np.vstack((x, y))
55      x = aux[0, :]
56      y = aux[1, :]
57
58      print('Rotacao x:', x)
59      print('Rotacao y:', y)
60
61      areaWidth = np.max(x) - np.min(x)
62      areaLength = np.max(y) - np.min(y)
63      print('Rotacao da largura de area: ',areaWidth)
64      print('Rotacao de comprimento de area: ',areaLength)
65
```

```
66      numberOfLanes = int(np.ceil(areaWidth / (imageWidth * (1 - sidelap))))
67      laneDist = areaWidth / numberOfLanes
68      print('Numero de segmentos:', numberOfLanes)
69      print('Distancia entre segmentos: ',laneDist)
70
71      lanemin = []
72      lanemax = []
73
74      for i in range(numberOfLanes):
75          xi = np.min(x) + laneDist * i + laneDist / 2
76          print('Segmento ', i + 1,'centro da coordenada x: ', xi)
77
78
79          delta = imageLength / 10
80          k = 0
81          miny = np.min(y)
82          while k * delta <= areaLength:
83              if inpolygon([xi], [miny + k * delta], x, y):
84                  miny = miny + k * delta
85                  break
86              k += 1
87          print('Segmento ', i+1,' min y: ',miny)
88
89          k = 0
90          maxy = np.max(y)
91          while k * delta <= areaLength:
92              if inpolygon([xi], [maxy - k * delta], x, y):
93                  maxy = maxy - k * delta
94                  break
95              k += 1
96          print('Segmento ', i+1,' max y: ',maxy)
97
98          lanemin.append([xi, miny])
99          lanemax.append([xi, maxy])
100
101     lanemin = np.array(lanemin)
102     lanemax = np.array(lanemax)
103
104     lmin = (R.T @ lanemin.T).T
105     lmax = (R.T @ lanemax.T).T
106
107     print('Lmin:', lmin)
108     print('Lmax:', lmax)
109
110     # Construcao dos Vertices
111     V = np.zeros((numberOfLanes * 2 + 1, 2))
112     V[0] = start_point
113     for i in range(1, numberOfLanes * 2 + 1):
114         if i % 2 == 0:
115             V[i] = lmin[i // 2 - 1]
116         else:
117             V[i] = lmax[(i - 1) // 2]
118
119     print('Vertices V:', V)
120     plt.show()
121
122     return lmin, lmax, V, laneDist
123
124 def plotStrips(lmin, lmax):
```

```
125     plt.plot(lmin[:, 0], lmin[:, 1], 'ko', lmax[:, 0], lmax[:, 1], 'ko',
            linewidth=3, markersize=12)
126     for i in range(len(lmin)):
127         plt.plot([lmin[i, 0], lmax[i, 0]], [lmin[i, 1], lmax[i, 1]], 'k--')
```

Listagem B.3: Script Waypoints.py para estabelecer os waypoints dos VANTs.

```
1 import numpy as np
2
3
4 def findWaypoints(X, V,start_point):
5     visitedWaypoints, order = np.max(X, axis=0), np.argmax(X, axis=0)
6     waypoints = []
7     j = 0
8     i = 0
9     while i < np.sum(visitedWaypoints):
10        coord = [V[order[j], 0], V[order[j], 1]]
11        assert isinstance(coord, list), "Erro: coordenadas nao sao uma lista.
              "
12        waypoints.append(coord)
13        i += 1
14        j = order[j]
15    waypoints.insert(0, start_point)  # Adiciona o ponto inicial
16
17    return waypoints
```

Listagem B.4: Scrit Time.py que calcula o tempo que cada VANT demora a passar pelos waypoints todos.

```
1 import numpy as np
2
3 def time(waypoints, uavSpeed, uavSetupTime, O):
4
5     t = []
6     t_voo = []
7
8     for k in range(len(waypoints)):
9         sum1 = 0
10        for i in range(1, len(waypoints[k])):
11            sum1 += np.linalg.norm(np.array(waypoints[k][i]) - np.array(
                  waypoints[k][i-1])) / (uavSpeed * 1000 / 60)
12        t.append(sum1 + np.ceil((k + 1) / O) * uavSetupTime)
13        t_voo.append(sum1)
14
15    return t, t_voo
```

Listagem B.5: Script Plot_Path.py para imprimir graficamente o trajeto dos VANTs.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import matplotlib.cm as cm
4
5 def plot_uav_path(waypoints):
6     if isinstance(waypoints, int):
7         print("waypoints e um inteiro! Verifique sua definicao.")
8         return
9
10    cmap = cm.get_cmap('Set1')
11    clr = cmap.colors
12
```

```
13      for k in range(len(waypoints)):
14          color = clr[k % len(clr)]
15          if isinstance(waypoints[k], int):
16              print(f"waypoints[{k}] e um inteiro! Verifique sua definicao.")
17              return
18
19          for i in range(len(waypoints[k])):
20              if isinstance(waypoints[k][i], int):
21                  print(f"waypoints[{k}][{i}] e um inteiro! Verifique sua
                        definicao.")
22                  return
23
24              plt.plot(waypoints[k][i][0], waypoints[k][i][1], 'x', color=color
                    ,
25                       linewidth=2, markersize=8, markerfacecolor=color)
26
27              if i > 0:
28                  # Desenhar linha entre pontos consecutivos
29                  plt.plot([waypoints[k][i-1][0], waypoints[k][i][0]],
30                           [waypoints[k][i-1][1], waypoints[k][i][1]],
31                           color=color, linewidth=2)
32
33                  # Calcular as coordenadas do meio da linha
34                  mid_x = (waypoints[k][i-1][0] + waypoints[k][i][0]) / 2
35                  mid_y = (waypoints[k][i-1][1] + waypoints[k][i][1]) / 2
36
37                  # Calcular o deslocamento da seta
38                  dx = mid_x - waypoints[k][i-1][0]
39                  dy = mid_y - waypoints[k][i-1][1]
40
41                  # Desenhar seta com base no inicio da linha e ponta no meio
42                  plt.arrow(waypoints[k][i-1][0], waypoints[k][i-1][1], dx, dy,
43                            head_width=50, head_length=50, fc=color, ec=color,
44                            shape='full', overhang=0.5, width=3)
45
46
47  # Passo 1: Desenhar a triangulacao de Delaunay total
48  def plotar_triangulacao_total(pontos, triangulacao):
49      plt.figure()
50      plt.triplot(pontos[:, 0], pontos[:, 1], triangulacao.simplices, 'go--',
            linewidth=1)
51      plt.plot(pontos[:, 0], pontos[:, 1], 'ro')
52      plt.title("Triangulacao de Delaunay Total")
53      plt.xlabel("X")
54      plt.ylabel("Y")
55      plt.grid(True)
56      plt.show()
57
58  # Passo 2: Desenhar a triangulacao de Delaunay dentro do poligono
59  def plotar_triangulos_dentro_poligono(pontos, triangulos):
60      plt.figure()
61      for tri_pts, triangulo in triangulos:
62          x, y = triangulo.exterior.xy
63          plt.fill(x, y, color='blue', alpha=0.3)
64      plt.plot(pontos[:, 0], pontos[:, 1], 'ro')
65      plt.title("Triangulacao de Delaunay Dentro do Poligono")
66      plt.xlabel("X")
67      plt.ylabel("Y")
68      plt.grid(True)
```

```
69      plt.show()
70
71  # Passo 3: Desenhar as zonas agrupadas por area
72  def plotar_zonas_agrupadas(pontos, poligono, zonas):
73      plt.figure()
74      cores = ['b', 'g', 'm', 'c', 'y']  # Cores para diferentes zonas
75      for i, zona in enumerate(zonas):
76          for triangulo in zona:
77              x, y = triangulo.exterior.xy
78              plt.fill(x, y, color=cores[i % len(cores)], alpha=0.5)
79      plt.plot(*poligono.exterior.xy, 'k')
80      plt.plot(pontos[:, 0], pontos[:, 1], 'ro')
81      plt.title("Zonas Agrupadas por Area")
82      plt.xlabel("X")
83      plt.ylabel("Y")
84      plt.grid(True)
85      plt.show()
```

Listagem B.6: Script cost.py que implementa o otimizador Gurobi para realizar a otimização.

```
1  from gurobipy import Model, GRB, quicksum
2  import numpy as np
3
4  def lip3(V, uav, uavSpeed, uavSetupTime, uavFlightTime, h, O):
5      if len(V) < 2:
6          raise ValueError("O numero de vertices deve ser pelo menos 2.")
7      if len(uav) < 1:
8          raise ValueError("O numero de UAVs deve ser pelo menos 1.")
9      if O == 0:
10         raise ValueError("O valor de O nao pode ser zero.")
11
12     numberOfVertices = len(V)
13     uavNumber = len(uav)
14
15     # Construcao da matriz de custos
16     C = np.zeros((numberOfVertices, numberOfVertices))
17     for i in range(numberOfVertices):
18         for j in range(numberOfVertices):
19             if i != j:
20                 C[i, j] = np.linalg.norm(V[i] - V[j]) / (uavSpeed * 1000 / 60
                   )
21
22     # Definicao do problema
23     model = Model("UAV_VRP")
24
25     # Variaveis de decisao
26     v = model.addVar(name="v", vtype=GRB.CONTINUOUS, lb=0)
27     s = model.addVar(name="s", vtype=GRB.CONTINUOUS, lb=0)
28     X = model.addVars(numberOfVertices, numberOfVertices, uavNumber, vtype=
          GRB.BINARY, name="X")
29     u = model.addVars(numberOfVertices, vtype=GRB.INTEGER, lb=0, name="u")
30     m = model.addVar(name="m", vtype=GRB.INTEGER, lb=0)
31
32     # Restricoes
33     constraints = []
34
35     # Restricao #1 - Custo total de operacao
36     for k in range(uavNumber):
37         sum1 = quicksum(C[i][j] * X[i,j,k] for i in range(numberOfVertices)
```

```
             for j in range(numberOfVertices) if i != j)
38          model.addConstr(v >= sum1 + quicksum(X[0,j,k] for j in range(
                numberOfVertices)) * np.ceil((k+1)/O) * uavSetupTime)
39          model.addConstr(sum1 <= uavFlightTime)
40
41      # Restricao 2 - Cada vertice deve ser visitado uma vez e por um VANT
42      for j in range(1, numberOfVertices):
43          model.addConstr(1 == quicksum(X[i,j,k] for i in range(
                numberOfVertices) for k in range(uavNumber)))
44
45      # Restricao 3 - Ao visitar um vertice, o VANT deve sair daquele vertice
46      for k in range(uavNumber):
47          for p in range(numberOfVertices):
48              model.addConstr(quicksum(X[i,p,k] for i in range(numberOfVertices
                    )) - quicksum(X[p,j,k] for j in range(numberOfVertices)) == 0
                    )
49
50      # Restricao 4 - Numero de VANTs utilizado
51      model.addConstr(quicksum(X[0,j,k] for j in range(numberOfVertices) for k
            in range(uavNumber)) == m)
52      model.addConstr(m <= sum(uav))
53
54      # Restricao 5 - Ciclo
55      for i in range(1, numberOfVertices):
56          for j in range(1, numberOfVertices):
57              if i != j:
58                  model.addConstr(u[i] - u[j] + numberOfVertices * quicksum(X[i
                        ,j,k] for k in range(uavNumber)) <= numberOfVertices - 1)
59
60      # Restricao 6 - Cada linha deve ser percorrida por apenas um VANT e em
            apenas um sentido.
61      for i in range(1, numberOfVertices, 2):
62          if i + 1 < numberOfVertices:
63              model.addConstr(1 == quicksum(X[i,i+1,k] for k in range(uavNumber
                    )) + quicksum(X[i+1,i,k] for k in range(uavNumber)))
64
65     # Restricao 7 - Evita diagonais
66      for i in range(1, numberOfVertices, 2):
67          if i + 1 < numberOfVertices:
68              model.addConstr(quicksum(X[i,i+1,k] for k in range(uavNumber)) ==
69                              quicksum(X[i,j,k] for k in range(uavNumber) for j
                                    in range(2, numberOfVertices, 2)))
70
71              model.addConstr(quicksum(X[i+1,i,k] for k in range(uavNumber)) ==
72                              quicksum(X[i+1,j,k] for k in range(uavNumber) for
                                    j in range(1, numberOfVertices, 2)))
73
74      # Funcao objetivo
75      if h == 1:
76          model.setObjective(0.999 * v + 0.001 * v, GRB.MINIMIZE)
77      else:
78          model.setObjective(v, GRB.MINIMIZE)
79
80
81      # Resolver o problema
82      model.optimize()
83
84      # Extrair os valores das variaveis de decisao
85      X_val = np.zeros((numberOfVertices, numberOfVertices, uavNumber))
```

```
86      for i in range(numberOfVertices):
87          for j in range(numberOfVertices):
88              for k in range(uavNumber):
89                  X_val[i, j, k] = X[i, j, k].x
90
91      v_val = v.x
92      m_val = m.x
93
94      return X_val, v_val, m_val
```

Listagem B.7: Script AreaDecomp.py para realizar o algoritmo da triangulação de Delaunay.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from shapely.geometry import Polygon
4  from collections import deque
5
6  # Funcao para capturar pontos usando ginput
7  def capturar_pontos():
8      # plt.figure()
9      # plt.title("Clique para definir os pontos (feche a janela quando
           terminar)")
10     # plt.xlabel("X")
11     # plt.ylabel("Y")
12     # plt.xlim(0, 2000)
13     # plt.ylim(0, 2000)
14     # pontos = plt.ginput(n=-1, timeout=0)
15     # plt.close()
16
17     # pontos = np.array(pontos)
18     # coordenadas_x = [806.4516129, 532.25806452, 314.51612903, 145.16129032,
            56.4516129, 161.29032258, 455.64516129, 608.87096774, 487.90322581,
           193.5483871, 116.93548387, 342.74193548, 612.90322581, 979.83870968,
           1145.16129032, 1415.32258065, 1588.70967742, 1870.96774194,
           1983.87096774, 1862.90322581, 1697.58064516, 1842.74193548,
           1915.32258065, 1810.48387097, 1483.87096774, 1169.35483871,
           1330.64516129, 1245.96774194, 1000.]
19     # coordenadas_y = [1797.61904762, 1792.20779221, 1916.66666667,
           1819.26406926, 1494.58874459, 1332.25108225, 1137.44588745,
           985.93073593, 801.94805195, 569.26406926, 255.41125541, 60.60606061,
           76.83982684, 109.30735931, 287.87878788, 341.99134199, 147.18614719,
           103.8961039, 363.63636364, 699.13419913, 926.40692641, 1224.02597403,
            1548.7012987, 1867.96536797, 1922.07792208, 1862.55411255,
           1456.70995671, 1229.43722944, 1234.84848485]
20
21     # coordenadas_x = [-250, 250, 400, 0,-400]
22     # coordenadas_y = [500, 500, 1100, 1500,1100]
23
24     # coordenadas_x = [500, 500, 1500, 1500]
25     # coordenadas_y = [500, 1500, 1500, 500]
26
27     # coordenadas_x = [500, 500, 1500]
28     # coordenadas_y = [500, 1500, 500]
29
30     # coordenadas_x = [1000, 0,1000,2000]
31     # coordenadas_y = [2000, 1000, 0,1000]
32
33     coordenadas_x = [500, 1500,1500,1000,1000,1500,1500,500]
34     coordenadas_y = [500, 500, 1000,1000,2000,2000,2500,2500]
```

```
35
36
37     pontos = np.array(list(zip(coordenadas_x, coordenadas_y)))
38     return pontos
39
40 # Funcao para calcular a area de um triangulo
41 def area_triangulo(triangulo):
42     return triangulo.area
43
44 # Funcao para obter os triangulos dentro do poligono
45 def triangulos_dentro_poligono(pontos, triangulacao, poligono):
46     triangulos = []
47     for simplex in triangulacao.simplices:
48         tri_pts = pontos[simplex]
49         triangulo = Polygon(tri_pts)
50         # Adiciona o triangulo se ele estiver completamente dentro do
              poligono
51         if poligono.contains(triangulo):
52             triangulos.append((tri_pts, triangulo))
53     return triangulos
54
55 # Funcao para verificar se dois triangulos sao adjacentes (compartilham uma
       aresta)
56 def sao_adjacentes(triangulo1, triangulo2):
57     # Verifica se dois triangulos compartilham uma aresta
58     coords1 = list(triangulo1.exterior.coords)[:-1]
59     coords2 = list(triangulo2.exterior.coords)[:-1]
60     shared_vertices = 0
61     for p1 in coords1:
62         if p1 in coords2:
63             shared_vertices += 1
64     return shared_vertices == 2
65
66 # Funcao para agrupar triangulos de forma contigua por area
67 def agrupar_por_area_contiguo(triangulos, numberUAV):
68     # Ordena os triangulos por area de forma decrescente
69     triangulos.sort(key=lambda x: x[1].area, reverse=True)
70
71     # Iniciar cada zona com o maior triangulo restante
72     zonas = [[] for _ in range(numberUAV)]
73     zonas_areas = [0] * numberUAV
74     usados = [False] * len(triangulos)
75     fila = deque()
76
77     # Adicionando o maior triangulo de cada zona
78     for i in range(numberUAV):
79         if i >= len(triangulos): break
80         zonas[i].append(triangulos[i][1])
81         zonas_areas[i] += triangulos[i][1].area
82         usados[i] = True
83         fila.append((i, triangulos[i][1]))  # (zona_id, triangulo)
84
85     while fila:
86         zona_id, triangulo_atual = fila.popleft()
87         for j in range(len(triangulos)):
88             if not usados[j] and sao_adjacentes(triangulo_atual, triangulos[j
                  ][1]):
89                 zonas[zona_id].append(triangulos[j][1])
90                 zonas_areas[zona_id] += triangulos[j][1].area
```

```
91                  usados[j] = True
92                  fila.append((zona_id, triangulos[j][1]))
93
94      # Caso ainda hajam triangulos nao alocados, atribui-los as zonas mais
        proximas
95      for i in range(len(triangulos)):
96          if not usados[i]:
97              menor_zona = np.argmin(zonas_areas)
98              zonas[menor_zona].append(triangulos[i][1])
99              zonas_areas[menor_zona] += triangulos[i][1].area
100
101     return zonas
102
103 # Funcao para plotar as zonas agrupadas e imprimir suas areas
104 def plotar_zonas(pontos, poligono, zonas):
105     plt.figure()
106     plt.plot(pontos[:, 0], pontos[:, 1], 'o')
107     plt.plot(*poligono.exterior.xy, 'k')
108     cores = ['b', 'g', 'm', 'c', 'y']  # Cores para diferentes zonas
109     total_area = 0
110     for i, zona in enumerate(zonas):
111         zona_area = sum(triangulo.area for triangulo in zona)
112         total_area += zona_area
113         print('Area da Zona: ',zona_area)
114         for triangulo in zona:
115             x, y = triangulo.exterior.xy
116             plt.fill(x, y, color=cores[i % len(cores)], alpha=0.5)
117     print('Area Total: ',total_area)
118     plt.title("Zonas agrupadas por area (continuas)")
119     plt.xlabel("X")
120     plt.ylabel("Y")
121     plt.show()
```

Listagem B.8: Script NonConv.py é o script principal para o método de trajetória com a triangulação de Delaunay.

```
1  from Strips import findStrips
2  from Waypoints import findWaypoints
3  from cost import lip3
4  from Time import time
5  import numpy as np
6  import matplotlib.pyplot as plt
7  from scipy.spatial import Delaunay
8  from shapely.geometry import Polygon
9  from AreaDecomp import capturar_pontos, triangulos_dentro_poligono,
       agrupar_por_area_contiguo, plotar_zonas
10 from Plot_Path import plotar_triangulacao_total,
       plotar_triangulos_dentro_poligono, plotar_zonas_agrupadas
11
12
13
14 def plot_strips_separado(lmin, lmax):
15     plt.plot(lmin[:, 0], lmin[:, 1], 'ko', lmax[:, 0], lmax[:, 1], 'ko',
           linewidth=3, markersize=12)
16     for i in range(len(lmin)):
17         plt.plot([lmin[i, 0], lmax[i, 0]], [lmin[i, 1], lmax[i, 1]], 'k--')
18
19 def plot_uav_path_with_areas(waypoints, zonas, colors, uav_idx):
20     for i, zona in enumerate(zonas):
21         for triangulo in zona:
```

```
22              x, y = triangulo.exterior.xy
23              plt.fill(x, y, color=colors[i % len(colors)], alpha=0.5)
24
25      for k, uav_waypoints in enumerate(waypoints):
26          for i in range(len(uav_waypoints)):
27              plt.plot(uav_waypoints[i][0], uav_waypoints[i][1], 'x',
28                       color=colors[uav_idx], linewidth=2, markersize=8)
29              plt.text(uav_waypoints[i][0], uav_waypoints[i][1], f'{uav_idx +
                    1}-{i + 1}', fontsize=12, color='black')
30              if i > 0:
31                  plt.plot([uav_waypoints[i-1][0], uav_waypoints[i][0]],
32                           [uav_waypoints[i-1][1], uav_waypoints[i][1]],
33                           color=colors[uav_idx], linewidth=2)
34
35  # Funcao para plotar os caminhos dos UAVs sem as zonas
36  def plot_uav_paths_only(waypoints, colors, uav_idx):
37      for k, uav_waypoints in enumerate(waypoints):
38          for i in range(len(uav_waypoints)):
39              plt.plot(uav_waypoints[i][0], uav_waypoints[i][1], 'x',
40                       color=colors[uav_idx], linewidth=2, markersize=8)
41              if i > 0:
42                  # Desenhar linha entre pontos consecutivos
43                  plt.plot([waypoints[k][i-1][0], waypoints[k][i][0]],
44                           [waypoints[k][i-1][1], waypoints[k][i][1]],
45                           color=colors[uav_idx], linewidth=2)
46
47                  # Calcular as coordenadas do meio da linha
48                  mid_x = (waypoints[k][i-1][0] + waypoints[k][i][0]) / 2
49                  mid_y = (waypoints[k][i-1][1] + waypoints[k][i][1]) / 2
50
51                  # Calcular o deslocamento da seta
52                  dx = mid_x - waypoints[k][i-1][0]
53                  dy = mid_y - waypoints[k][i-1][1]
54
55                  # Desenhar seta com base no inicio da linha e ponta no meio
56                  plt.arrow(waypoints[k][i-1][0], waypoints[k][i-1][1], dx, dy,
57                            head_width=50, head_length=50, fc=colors[uav_idx],
                                ec=colors[uav_idx],
58                            shape='full', overhang=0.5, width=3)
59
60
61
62
63  def process_zone(zone_vertices, uav_settings, uav_id):
64      sidelap = uav_settings['sidelap']
65      imageWidth = uav_settings['imageWidth']
66      imageLength = uav_settings['imageLength']
67      start_point = uav_settings['start_point']
68
69      # Calcular strips para a zona
70      lmin, lmax, V, laneDist = findStrips(zone_vertices[:, 0], zone_vertices
            [:, 1], sidelap, imageWidth, imageLength, start_point)
71
72      # Otimizar rota para o UAV especifico nesta zona
73      uav_current = uav_settings['uav'][uav_id]
74      X, v, m = lip3(V, [uav_current], uav_settings['uavSpeed'], uav_settings['
            uavSetupTime'],
75                     uav_settings['uavFlightTime'], uav_settings['h'],
                           uav_settings['O'])
```

```
    # Gerar waypoints para a zona especifica
    waypoints = []
    if m > 0:  # Garante que haja ao menos um caminho encontrado
        waypoints.append(findWaypoints(X[:, :, 0], V, start_point))

    # Calcular tempos
    t, t_voo = time(waypoints, uav_settings['uavSpeed'], uav_settings['
        uavSetupTime'], uav_settings['O'])

    return {
        "waypoints": waypoints,
        "v": v,
        "m": m,
        "t": t,
        "t_voo": t_voo,
        "laneDist": laneDist
    }

# Configuracoes gerais e do UAV
uav_settings = {
    'uavNumber': 3,
    'uavSetupTime': 4,
    'uavFlightTime': 500,
    'uavSpeed': 45,
    'flightAltitude': 200,
    'O': 1,
    'h': 1,
    'sidelap': 0.1,
    'start_point': [0, 1500]
}

# Definir largura e comprimento da imagem com base na altitude de voo
uav_settings['imageWidth'] = uav_settings['flightAltitude'] * 2 * np.tan(np.
    radians(65 / 2))
uav_settings['imageLength'] = uav_settings['flightAltitude'] * 2 * np.tan(np.
    radians(49 / 2))
uav_settings['uav'] = np.ones((uav_settings['uavNumber'], 1))

# Captura de pontos para definir a area
pontos = capturar_pontos()
poligono = Polygon(pontos)
triangulacao = Delaunay(pontos)

triangulos = triangulos_dentro_poligono(pontos, triangulacao, poligono)

zonas = agrupar_por_area_contiguo(triangulos, uav_settings['uavNumber'])

# Cores para os UAVs
uav_colors = ['red', 'yellow', 'green', 'purple','white']

# Processar cada zona e associar a um UAV
resultados = []
for idx, zona in enumerate(zonas):
    zona_vertices = np.array([list(triangulo.exterior.coords)[:-1] for
        triangulo in zona]).reshape(-1, 2)

    # Processa cada zona e envia os segmentos correspondentes para a funcao
        de custo
```

```
130     resultado = process_zone(zona_vertices, uav_settings, idx)
131     resultados.append(resultado)
132
133 # Exibir resultados e desenhar caminhos dos UAVs
134 plt.figure()
135 plotar_zonas(pontos, poligono, zonas)  # Desenha as areas coloridas
136 for idx, resultado in enumerate(resultados):
137     plot_uav_paths_only(resultado['waypoints'], uav_colors, idx)
138 plt.show()
139
140
141 #Funcoes para gerar os plots
142 plotar_triangulacao_total(pontos, triangulacao)
143 plotar_triangulos_dentro_poligono(pontos, triangulos)
144 plotar_zonas_agrupadas(pontos, poligono, zonas)
145
146 for idx, resultado in enumerate(resultados):
147     print('WAYPOINTS para Zona', idx + 1, ':', resultado['waypoints'])
148     print('Tempos totais para Zona ',idx + 1,':', resultado['t'])
149     print('Tempos de voo para Zona ',idx + 1,':', resultado['t_voo'])
```